**Лекция 2. Чтение данных из базы данных**

1. Выполнение команд.Объект Command
2. Объект DataReader
3. Вызов хранимых процедур
4. Выполнение команд SQL
5. **Выполнение команд. Объект Command**

Каждый поставщик данных .NET Framework имеет собственный объект Command, наследуемый от [DbCommand](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.common.dbcommand). Поставщик данных .NET Framework для SQL Server - объект [SqlCommand](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.sqlclient.sqlcommand), поставщик данных .NET Framework для OLE DB содержит объект [OleDbCommand](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.oledb.oledbcommand), поставщик данных .NET Framework для ODBC - объект [OdbcCommand](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.odbc.odbccommand), а поставщик данных .NET Framework для Oracle - объект [OracleCommand](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.oracleclient.oraclecommand). Каждый из этих объектов предоставляет методы выполнения команд с учетом типа команды и требуемого возвращаемого значения, как описано в следующей таблице.

| **Метод** | **Возвращаемое значение** |
| --- | --- |
| ExecuteReader | Выполняет запрос select и возвращает объект DataReader, который является оболочкой однонаправленного курсора, доступного только для чтения |
| ExecuteScalar | Выполняет запрос select и возвращает значение первого поля первой строки из набора строк, сгенерированного командой. Этот метод обычно применяется при выполнении агрегатной команды select, использующей функции вроде count() или sum() для вычисления единственного значения. |
| ExecuteNonQuery | Выполняет команды, отличные от select, такие как SQL-операторы вставки, удаления или обновления записей. Возвращаемое значение указывает количество строк, обработанных командой. Метод ExecuteNonQuery() также можно использовать для выполнения команд определения данных, которые создают, изменяют и уничтожают объекты базы данных (наподобие таблиц, индексов, ограничений и т.п.) |
| ExecuteXMLReader | Возвращает значение типа [XmlReader](https://docs.microsoft.com/ru-ru/dotnet/api/system.xml.xmlreader). Этот метод предусмотрен только для объекта SqlCommand. |

Объект Command имеет свойство CommandType, которые определяют тип используемой команды. Возможные значения этого свойства приведены в таблице.

|  |  |
| --- | --- |
| Значение CommandType | Описание |
| CommandType.Text | Команда будет выполнять прямой оператор SQL. Оператор SQL указывается в свойстве CommandText. |
| CommandType.StoredProcedure | Команда будет выполнять хранимую процедуру в источнике данных. Свойство CommandText представляет имя хранимой процедуры. |
| CommandType.TableDirect | Команда будет опрашивать все записи таблицы. CommandText — имя таблицы, из которой команда извлечет все записи. (Эта опция предназначена только для обратной совместимости с некоторыми драйверами OLE DB. Она не поддерживается поставщиком данных SQL Server и не работает так хорошо, как тщательно направленный запрос.) |

Пример использования команды

**Добавление объекта**

Ниже приведен пример использования команды для добавления одного объекта в таблицу Persons базы данных myshop, которая ранее была создана:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Data.SqlClient;

namespace MyShop

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

string connectionString = "Data Source=MyPC\\SQLEXPRESS;Initial Catalog=Shop;Integrated Security = True";

SqlConnection Con = new SqlConnection(connectionString);

Con.Open();

string sqlExpression = "INSERT INTO Employees (LName,FName,Birthdate) VALUES('Мороз', 'Василий','1956-12-30')";

SqlCommand command = new SqlCommand();

command.CommandType = CommandType.Text;

command.CommandText = sqlExpression;

command.Connection = Con;

int number = command.ExecuteNonQuery();

MessageBox.Show("Добавлено объектов: " + number);

Con.Close();

}

}

}

Для упрощения кода подключение к базе данных и выполнение команд включено в функцию Form1\_Load().

Для вставки объекта используется sql-выражение INSERT, которое имеет следующий синтаксис:

INSERT INTO имя\_таблицы (столбец1, столбец2, столбецN ) VALUES (значение1, значение2, значениеN)

В данном случае в базе данных имеется таблица Employees, в которой есть три столбца - Id int (счетчик, первичный ключ), LName и FName – varchar(20) и BirthDate - date. Поэтому в программе оператор INSERT определяет все столбцы, кроме Id.

Метод ExecuteNonOuery() возвращает число вставленных строк (число добавленных в таблицу объектов), которое выводится на экран с помощью оператора MessageBox.Show.

1. **Объект DataReader**

Класс DataReader позволяет читать данные, возвращенные командой SELECT, по одной строке за раз, в однонаправленном, доступном только для чтения потоке. Иногда это называют курсором. DataReader представляет наиболее быстрый способ доступа к данным. Ниже перечислены основные методы DataReader:

|  |  |
| --- | --- |
| ***Методы класса DataReader*** | |
| **Метод** | **Описание** |
| *Read()* | Перемещает курсор строки на следующую строку в потоке. Этот метод также должен быть вызван перед чтением первой строки данных (Когда DataReader создается впервые, курсор строки помещается в позицию непосредственно перед первой строкой.) Метод Read() возвращает true, если существует следующая строка для чтения, или false, если прочитана последняя строка в наборе |
| *GetValue()* | Возвращает значение, сохраненное в поле с указанным именем столбца или индексом, внутри текущей выбранной строки. Тип возвращенного значения — ближайший тип .NET, наиболее соответствующий встроенному значению, хранимому в источнике данных. |
| *GetValues()* | Сохраняет значения текущей строки в массиве. Количество сохраняемых полей зависит от размеров массива, переданного этому методу. С помощью свойства DataReader.FieldCount можно определить действительное количество полей в строке и воспользоваться этой информацией для создания массива нужного размера, если нужно сохранить в нем все поля |
| *GetInt32(), GetChar(), GetDateTime()*, Get...() | Эти методы возвращают значение поля с указанным индексом в текущей строке, причем тип данных указывается в имени метода. Эти методы не поддерживают типы, допускающие NULL-значения. Если поле может содержать NULL, это придется проверить перед вызовом одного из методов. Чтобы проверить на NULL-значение нужно сравнить непреобразованное значение (которое можно извлечь по позиции методом GetValue() или по имени с помощью индексатора DataReader) с константой DBNull.Value |
| *NextResult()* | Если команда, которая сгенерировала DataReader, возвратила более одного набора строк, этот метод перемещает указатель на следующий набор и устанавливает его непосредственно перед первой строкой |
| *Close()* | Закрывает модуль чтения. Если исходная команда запустила хранимую процедуру, возвратившую выходное значение, это значение может быть прочитано из соответствующего параметра после закрытия модуля чтения |

**Метод ExecuteReader() и DataReader**

Для иллюстрации работы DataReader необходимо создать проект WindowsForms. При этом будет создана форма с именем Form1. На этой форме нужно разместить элемент управления ListBox с именем listBox1, кнопку (Button) с именем btRead и надписью на кнопке “Читать”, кнопку (Button) с именем btСlean и надписью на кнопке “Очистить”.

В примере создается простая команда запроса, которая должна при нажатии кнопки «Читать» вывести все фамилии, имена и отчества всех сотрудников из таблицы Persons базы данных myshop в элемент ListBox1. При нажатии кнопки «Очистить» элемент ListBox1 должен очищаться. Команда создается при загрузке страницы. Соединение открывается, и команда выполняется методом ExecuteReader(), который возвращает SqlDataReader. Получив DataReader, можно организовать цикл для прохождения по его записям, вызывая метод Read() в теле цикла. Этот метод перемещает курсор строки на следующую запись (при первом вызове — на первую строку). В следующем примере цикл продолжается до тех пор, пока Read() не вернет false, после чего он завершается. В примере приведены только тексты обработчиков событий нажатия кнопок btRead и btClean.

private void btRead\_Click(object sender, EventArgs e)

{

// Создание строки подключения

string connectionString = "Data Source=MYPC\\SQLEXPRESS;Initial Catalog=Shop;Integrated Security=True";

SqlConnection con = new SqlConnection(connectionString);

// Создание команды

SqlCommand command = new SqlCommand();

command.Connection = con;

command.CommandType = CommandType.Text;

command.CommandText = "SELECT \* FROM Employees";

//или так:

//SqlCommand command=new SqlCommand("SELECT \* FROM Employees", con);

string result = "";

con.Open();

SqlDataReader reader = command.ExecuteReader();

// Проход в цикле по записям

while (reader.Read())

{

result = reader.GetValue(1) + " " + reader.GetValue(2);

//вставка очередной фамилии в позицию 0

listBox1.Items.Insert(0, result);

}

// Закрыттие DataReader

reader.Close();

}

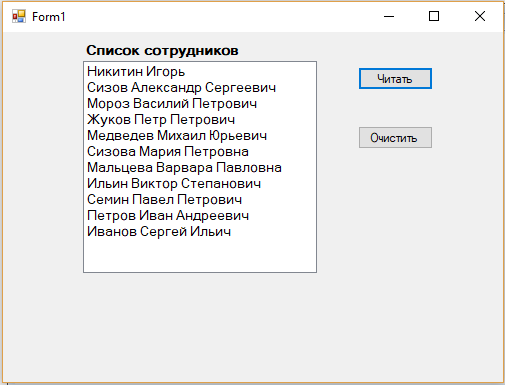
private void btClean\_Click(object sender, EventArgs e)

{

listBox1.Items.Clear();

}

При запуске этой страницы будет получен следующий результат:



Данные, возвращаемые DataReader, всегда доступны только для чтения.

**Метод ExecuteScalar()**

Метод ExecuteScalar() возвращает значение сохраненной в первом поле первой строки результирующего набора, сгенерированного запросом SELECT команды. Этот метод обычно применяется для выполнения запросов, возвращающих единственное поле, возможно, вычисленное агрегатной функцией SQL вроде COUNT() или SUM().

Следующая процедура демонстрирует, как можно с таким подходом получить (и отобразить на странице) количество записей таблицы Employees. В форму Form1 необходимо добавить кнопку элемент (Button) с именем btSum и надписью «Всего» и элемент Label с именем label2. При нажатии кнопки «Всего» в элемент label2 будет помещен текст с информацией о количестве сотрудников.

***private void btSum\_Click(object sender, EventArgs e)***

***{***

***string connectionString = "Data Source=MYPC\\SQLEXPRESS ;Initial Catalog=myshop;Integrated Security=True";***

***SqlConnection con = new SqlConnection(connectionString);***

***// Создать команду***

***SqlCommand command = new SqlCommand("SELECT COUNT (\*) FROM Persons", con);***

***int i = 0;***

***con.Open();***

***// Получить значение COUNT***

***i = (int)command.ExecuteScalar();***

***label4.Text = "Число сотрудников: " + i;***

***con.Close();***

***}***

Код достаточно прост, но стоит отметить, что необходимо привести возвращаемое значение к правильному типу, поскольку ExecuteScalar() возвращает объект.

**3. Вызов хранимых процедур**

Хранимая процедура представляет собой пакет из одного или более операторов SQL, сохраненный в базе данных.

Хранимые процедуры подобны функциям в том, что они являются хорошо инкапсулированными блоками логики, которые могут принимать данные (через входные параметры) и возвращать данные (через результирующие наборы и выходные параметры).

Рассмотрим пример вызова процедуры, текст которой приведен ниже

***CREATE PROCEDURE InsertPerson***

***-- Add the parameters for the stored procedure here***

***@LName varchar(20),***

***@FName varchar(20),***

***@MName varchar(20),***

***@EmpID int OUTPUT***

***AS***

***BEGIN***

***-- SET NOCOUNT ON added to prevent extra result sets from***

***SET NOCOUNT ON;***

***-- Insert statements for procedure here***

***INSERT INTO dbo.Persons***

***(LName, FName, Mname)***

***VALUES (@LName, @FName, @MName );***

***SET @EmpID = @@IDENTITY***

***END***

Хранимая процедура принимает три параметра — фамилию, имя и отчество. Возвращает она идентификатор вновь созданной записи через выходной параметр EmpID, который извлекается после оператора INSERT с помощью функции @@IDENTITY. Без использования хранимой процедуры было бы довольно неудобно получить автоматически сгенерированное значение идентификатора только что вставленной новой записи.

Для вызова хранимой процедуры можно создать команду SqlCommand, которая послужит оболочкой для вызова хранимой процедуры. Эта команда принимает те же три параметра на входе и использует @@IDENTITY для получения и возврата идентификатора новой записи. Так же понадобится добавить параметры хранимой процедуры в коллекцию Command.Parameters. При этом необходимо указать точный тип данных и длину параметра, чтобы они соответствовали деталям в базе данных. Для вызова процедуры в форму можно добавить кнопку с именем btAdd и надписью «Добавить».

private void btAdd\_Click(object sender, EventArgs e)

{

string connectionString = "Data Source=MYPC\\SQLEXPRESS ;Initial Catalog=shop;Integrated Security=True";

SqlConnection con = new SqlConnection(connectionString);

// Создать команду для вызова хранимой процедуры InsPerson

SqlCommand cmd = new SqlCommand("InsPerson", con);

cmd.CommandType = CommandType.StoredProcedure;

// Указать параметры

cmd.Parameters.AddWithValue("@LName", txtLName.Text);

cmd.Parameters.AddWithValue("@FName", txtFName.Text);

cmd.Parameters.AddWithValue("@MName", txtMName.Text);

// Последний параметр является выходным (output)

cmd.Parameters.Add(new SqlParameter("@EmpID", SqlDbType.Int, 4));

cmd.Parameters["@EmpID"].Direction = ParameterDirection.Output;

con.Open();

try

{

cmd.ExecuteNonQuery();

// Получить вновь сгенерированный идентификатор

int empID = (int)cmd.Parameters["@EmpID"].Value;

label7.Text = "Новому сотруднику присвоен ID: " + empID.ToString();

}

finally

{

con.Close();

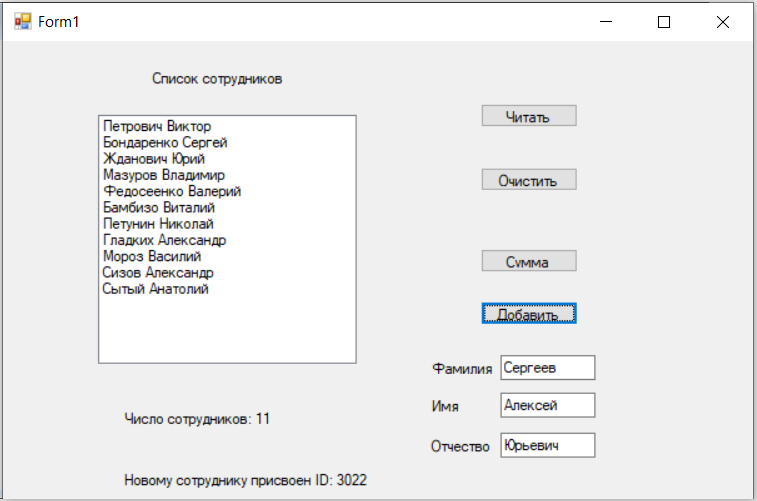
}

}

Последний параметр является выходным, что позволяет хранимой процедуре вернуть информацию в код. Хотя этот объект Parameter создается аналогично, необходимо указать его как выходной параметр, установив значение его свойства Direction в Output.

После выполнения кола в таблице Employees появится новая запись.

Внешний вид формы



## Выполнение команд SQL

### Добавление записей

Выполним команду по добавлению одной записи в таблицу Persons базы данных myshop при нажатии кнопки «Добавить запись» с именем btAddRec. Данные для добавляемой записи вводятся пользователем в текстовые поля txtLName, txtFName, txtMName.

Обработчик события «нажатие кнопки btAddRec» может иметь следующий вид:

***private void btAddRec\_Click(object sender, EventArgs e)***

***{***

***// Создание строки подключения***

***string connectionString = "Data Source=MyPC\\SQLEXPRESS;Initial Catalog=myshop;***

***Integrated Security=True";***

***//Создание подключения***

***SqlConnection con = new SqlConnection(connectionString);***

***con.Open();***

***//Создание команды***

***string sqlExpression = "INSERT INTO Persons (LName, FName, " +***

***"MName) VALUES ('" + txtLName.Text + "', '" + txtFName.Text + "', '"***

***+ txtMName.Text +"' )";***

***SqlCommand com = new SqlCommand(sqlExpression, con);***

***int number = com.ExecuteNonQuery();***

***label6.Text = "Добавлено объектов:" + number;***

***}***

В данном случае известно, что в базе данных есть таблица employees, в которой есть столбцы - ***empLastName, empFirstName, empPatronymic*** .

Здесь метод ExecuteNonOuery() возвращает число затронутых строк (в данном случае добавленных в таблицу записей). Хотя возвращать результат метода необязательно, но данный результат может использоваться в качестве проверки, что операция добавление, прошла успешно.

### Обновление записей

### Ниже приведен обработчик события «нажатие кнопки Обновить» (btUpdate), который обновляет записи в таблице Persons, заменяя пустые даты рождения датой 01.01.2000 всем сотрудникам.

***private void btUpdate\_Click(object sender, EventArgs e)***

***{***

***//Создание подключения***

***// Создание строки подключения***

***string connectionString = "Data Source=MyPC\\SQLEXPRESS;Initial Catalog=myshop;Integrated Security=True";***

***SqlConnection con = new SqlConnection(connectionString);***

***con.Open();***

***//Создание команды***

***string sqlExpression = "UPDATE Persons SET BirthDate=’2000-01-01’ WHERE BirthDate Is Null";***

***SqlCommand com = new SqlCommand(sqlExpression, con);***

***int number = com.ExecuteNonQuery();***

***label6.Text = "Изменено записей:" + number;***

***}***

### Удаление записей

### Ниже приведен обработчик события «нажатие кнопки Удалить» (btDelete), который удаляет записи из таблицы employees, в которых указана дата рождения сотрудника меньше, чем 2000-01- 01

***private void btDelete\_Click(object sender, EventArgs e)***

***{***

***//Создание подключения***

***// Создание строки подключения***

***string connectionString = "Data Source=MyPC\\SQLEXPRESS;Initial Catalog=myshop;Integrated Security=True";***

***SqlConnection con = new SqlConnection(connectionString);***

***con.Open();***

***//Создание команды***

***string sqlExpression = "DELETE FROM Persons WHERE BirthDate<'2000-01-01'";***

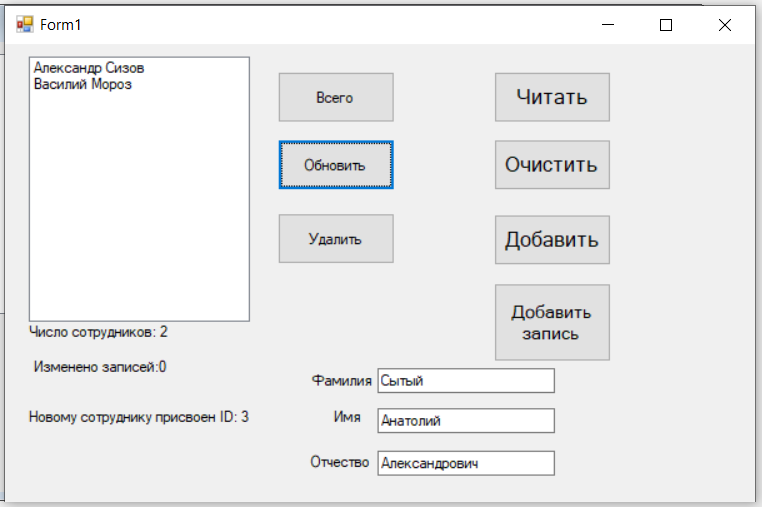
***SqlCommand com = new SqlCommand(sqlExpression, con);***

***int number = com.ExecuteNonQuery();***

***label6.Text = "Удалено записей:" + number;***

***}***

Форма со всеми элементами управления имеет вид



Полный код приложения имеет вид:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Data.SqlClient;

namespace MyShop

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

string connectionString = "Data Source=MyPC\\SQLEXPRESS;Initial Catalog=myshop;Integrated Security = True";

SqlConnection Con = new SqlConnection(connectionString);

Con.Open();

string sqlExpression = "SELECT COUNT(\*) FROM Persons";

SqlCommand command = new SqlCommand();

command.CommandType = CommandType.Text;

command.CommandText = sqlExpression;

command.Connection = Con;

string number = command.ExecuteScalar().ToString();

MessageBox.Show("Всего записей: " + number);

Con.Close();

}

private void btRead\_Click(object sender, EventArgs e)

{

// Создание строки подключения

string connectionString = "Data Source=MYPC\\SQLEXPRESS;Initial Catalog=myshop;Integrated Security=True";

SqlConnection con = new SqlConnection(connectionString);

// Создание команды

SqlCommand command = new SqlCommand();

command.Connection = con;

command.CommandType = CommandType.Text;

command.CommandText = "SELECT \* FROM Persons";

//или так:

//SqlCommand command=new SqlCommand("SELECT \* FROM Employees", con);

string result = "";

con.Open();

SqlDataReader reader = command.ExecuteReader();

// Проход в цикле по записям

while (reader.Read())

{

result = reader.GetValue(1) + " " + reader.GetValue(2);

listBox1.Items.Insert(0, result); //вставка очередной фамилии в позицию 0

}

// Закрыттие DataReader

reader.Close();

}

private void btClean\_Click\_1(object sender, EventArgs e)

{

listBox1.Items.Clear();

}

private void btAdd\_Click(object sender, EventArgs e)

{

string connectionString = "Data Source=MYPC\\SQLEXPRESS ;Initial Catalog=myshop;Integrated Security=True";

SqlConnection con = new SqlConnection(connectionString);

// Создать команду для вызова хранимой процедуры InsertPerson

SqlCommand cmd = new SqlCommand("InsertPerson", con);

cmd.CommandType = CommandType.StoredProcedure;

// Указать параметры

cmd.Parameters.AddWithValue("@LName", txtLName.Text);

cmd.Parameters.AddWithValue("@FName", txtFName.Text);

cmd.Parameters.AddWithValue("@MName", txtMName.Text);

// Последний параметр является выходным (output)

cmd.Parameters.Add(new SqlParameter("@EmpID", SqlDbType.Int, 4));

cmd.Parameters["@EmpID"].Direction = ParameterDirection.Output;

con.Open();

try

{

cmd.ExecuteNonQuery();

// Получить вновь сгенерированный идентификатор

int empID = (int)cmd.Parameters["@EmpID"].Value;

label7.Text = "Новому сотруднику присвоен ID: " + empID.ToString();

}

finally

{

con.Close();

}

}

private void btSum\_Click(object sender, EventArgs e)

{

string connectionString = "Data Source=MYPC\\SQLEXPRESS ;Initial Catalog=myshop;Integrated Security=True";

SqlConnection con = new SqlConnection(connectionString);

// Создать команду

SqlCommand command = new SqlCommand("SELECT COUNT (\*) FROM Persons", con);

int i = 0;

con.Open();

// Получить значение COUNT

i = (int)command.ExecuteScalar();

label4.Text = "Число сотрудников: " + i; // 9

con.Close();

}

private void AddRec\_Click(object sender, EventArgs e)

{

//Создание подключения

// Создание строки подключения

string connectionString = "Data Source=MYPC\\SQLEXPRESS ;Initial Catalog=myshop;Integrated Security=True";

SqlConnection con = new SqlConnection(connectionString);

con.Open();

//Создание команды

string sqlExpression = "INSERT INTO Persons (LName, FName, MName)" +

" VALUES('" + txtLName.Text + "', '" + txtFName.Text + "', '" + txtMName.Text + "' )";

SqlCommand com = new SqlCommand(sqlExpression, con);

int number = com.ExecuteNonQuery();

label6.Text = "Добавлено объектов:" + number;

}

private void btUpdate\_Click(object sender, EventArgs e)

{

//Создание подключения

// Создание строки подключения

string connectionString = "Data Source=MyPC\\SQLEXPRESS;Initial Catalog=myshop;Integrated Security=True";

SqlConnection con = new SqlConnection(connectionString);

con.Open();

//Создание команды

string sqlExpression = "UPDATE Persons1 SET BirthDate = '2000-01-01' WHERE BirthDate IS Null;";

SqlCommand com = new SqlCommand(sqlExpression, con);

int number = com.ExecuteNonQuery();

label6.Text = "Изменено записей:" + number;

}

private void btDel\_Click(object sender, EventArgs e)

{

//Создание подключения

// Создание строки подключения

string connectionString = "Data Source=MyPC\\SQLEXPRESS;Initial Catalog=myshop;Integrated Security=True";

SqlConnection con = new SqlConnection(connectionString);

con.Open();

//Создание команды

string sqlExpression = "DELETE FROM Persons WHERE BirthDate<'2000-01-01'";

SqlCommand com = new SqlCommand(sqlExpression, con);

int number = com.ExecuteNonQuery();

label6.Text = "Удалено записей:" + number;

}

}

}