**Лабораторная работа №2. Чтение данных из базы данных.**

**Цель работы:**

– получить практические навыки создания и использования провайдера данных для чтения.

**Теоретические сведения**

## 

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. Если вы обратитесь к полю по индексу и нечаянно передадите неверный индекс, ссылающийся на несуществующее поле, то получите исключение IndexOutOfRangeException. Используя индексатор для DataReader, можно получить значение по имени поля |
| *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. Обратите внимание, что если попытаться присвоить возвращенное значение переменной неверного типа, возникнет исключение InvalidCastException. Кроме того, эти методы не поддерживают типов, допускающих NULL-значения. Если поле может содержать null, это придется проверить перед вызовом одного из методов. Чтобы проверить на null-значение, сравните 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 следующую строку 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**

В следующем примере создается простая команда запроса, которая должна вернуть все записи из таблицы Employees базы данных Northwind. 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:

***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 trainingDB***

***{***

***public partial class Form1 : Form***

***{***

***public Form1()***

***{***

***InitializeComponent();***

***}***

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

***{***

***// Создание строки подключения***

***string connectionString = "Data Source=MYPC;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);***

***list1.Items.Insert(0,result);***

***}***

***// Закрыттие DataReader***

***reader.Close();***

***}***

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

***{***

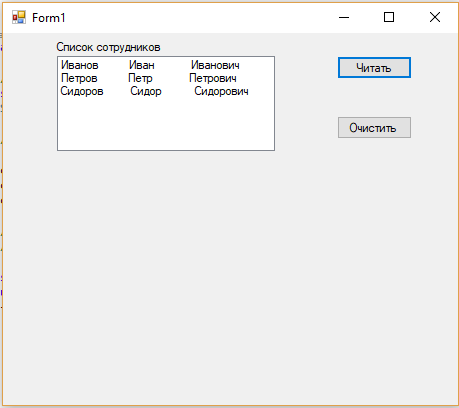
***list1.Items.Clear();***

***}***

***}***

***}***

При запуске этой страницы будет получен следующий результат:



DataReader построен как способ получить и изучить строки, возвращенные в ответ на запрос как можно быстрее. Это делает DataReader эффективным при возвращении больших объемов данных.

Данные, возвращаемые DataReader, всегда 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:

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

***{***

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

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

***// Создать команду***

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

***int i = 0;***

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

***// Получить значение COUNT***

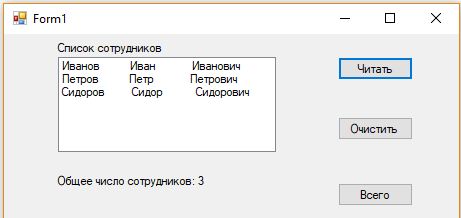
***i = (int)command.ExecuteScalar();***

***label2.Text = "Общее число сотрудников: " + i; // 9***

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

***}***

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



**Метод ExecuteNonQuery()**

Метод ExecuteNonQuery() выполняет команды, которые не возвращают результирующих наборов, такие как INSERT, DELETE или UPDATE. Метод ExecuteNonQuery() возвращает одну порцию информации — количество обработанных записей (или -1, если команда отлична от INSERT, DELETE или UPDATE). Применение этого метода класса Command аналогично предыдущим методам.

**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 InsEmployee***

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

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

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

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

***@EmployeeID int OUTPUT***

***AS***

***BEGIN***

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

***-- interfering with SELECT statements.***

***SET NOCOUNT ON;***

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

***INSERT INTO Employees***

***(FName, MName, LName, EmploymentDate)***

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

***SET @EmployeeID = @@IDENTITY***

***END***

Хранимая процедура принимает три параметра — фамилию, имя и отчество. Возвращает она идентификатор вновь созданной записи через выходной параметр EmployeeID, 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:

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

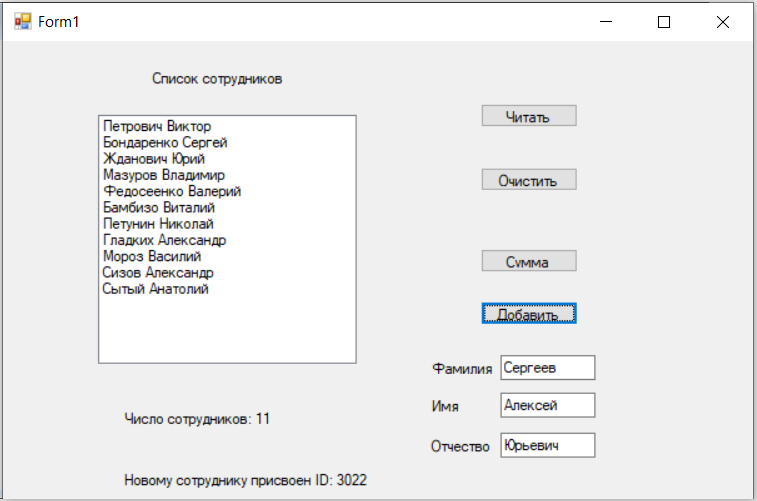
{

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.

**Задание на лабораторную работу**

Разработать провайдер данных для чтения и организовать чтение данных из базы данных в приложение.

1. Создать приложение (Windows Forms) для чтения данных из таблицы базы данных, содержащей сведения о людях.
2. Отобразить прочитанные данные в виде списка.
3. Организовать чтение данных, возвращаемых агрегатной функцией.
4. В базе данных создать хранимую процедуру для записи данных в таблицу и возврата идентификатора новой записи.
5. В приложении создать поля для ввода и отображения данных и кнопку для вызова хранимой процедуры записи данных в базу данных.

**Содержание отчета**

1. Тема и цель работы
2. Задание на лабораторную работу
3. Содержимое таблицы базы данных.
4. Вид формы для чтения и отображения данных.
5. Код программы формирования формы и чтения данных.
6. Код созданной хранимой процедуры.
7. Код вызова хранимой процедуры и передачи ей параметров.
8. Выводы

**Контрольные вопросы**

1. Какие объекты обеспечивают отсоединенный режим работы в ADO.NET?
2. Что нужно сделать для организации соединения с базой данных?
3. Какие основные свойства и методы содержит класс Command?
4. Какие действия выполняет метод Read() объекта DataAdapter&?
5. Какие методы объекта DataAdapter позволяют получить значения полей таблицы базы данных?
6. Для чего используется метод ExecuteScalar объекта DataAdapter?
7. Для чего используется метод ExecuteNonQuery объекта DataAdapter?
8. Как можно запустить на выполнение хранимую процедуру базы данных с помощью провайдера данных?