# Работа с данными в соединенной среде.

Как только вы подключились к БД SQL Server, объект SqlConnection может служить отправной точкой для многих операций, в том числе для создания команд, начала транзакций, получения логической структуры данных.

## Создание и выполнение объектов Command

Объекты Command содержат необходимую информацию для выполнения запросов SQL, хранимых процедур, функций и т. д. в базе, возвращения данных в приложение и проведения таких операций с каталогами" базы, как создание, изменение и удале­ние объектов. Другими словами, с помощью объекта Command можно выполнять любые доступные SQL-запросы.

ПРИМЕЧАНИЕ

Объекты *Command,* подобно объектам *Connection,* существуют для всех провай­деров данных .NET Framework. При работе с провайдером данных .NET Framework для SQL Server мы будем применять объект *SqlCommand,* а если вы используете провайдер данных .NET Framework для OLE DB — объект *OleDbCommand.*

Основными свойствами объекта *Command* являются *CommandText, CommandType* и *Connection.* Задайте свойству *CommandType* значение, представляющее собой коман­ду, которую необходимо выполнить в источнике данных. Например, чтобы выполнить стандартный SQL-запрос, задайте свойству *CommandType* значение *Text {SqlCom­mand.CommandType -ComandType.Text),* а затем присвойте свойству *CommandText* строку с запросом SQL, который вы хотите выполнить. Для выполнения хранимой процедуры задайте свойству *CommandType* значение *StoredProcedure* (*SqlCommand.CommandType* - *CommandType.StoredProcedure),* после чего задайте имя хранимой процедуры свойству *CommandText.*. В табл. 2-1 представлены основные объекты *Command* провайдера данных .NET Framework, доступные в ADO.NET, и источники данных, к которым объекты*Command* должны получать доступ.

Табл. 2-1. Объекты *Command*

|  |  |
| --- | --- |
| Имя | Целевой источник данных |
| *SqlCommand OleDb Command* | Базы данных SQL Server 2000 и SQL Server 2005  Источники данных OLE DB (например, базы данных Access через Jet 4.0) |
| *OdbcCommand* | Такие источники данных открытого интерфейса взаимодействия с базами данных (ODBC), как имя источника данных (DSN), указанное в диалоговом окне ODBC Data Source Administrator |
| *OracleCommand* | Базы данных Oracle 7.3,8i или 9i |

В табл. 2-2 представлены распространенные свойства объектов *Command* провай­дера данных .NET Framework.

**Табл. 2-2.** Распространенные свойства объекта *Command*

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| *CommandText* | Задает любое доступное выражение SQL или имя любой доступной хранимой процедуры. Значение *CommandType* определяет способ выполнения (см. ниже) |
| *CommandTimeout* | Время, указанное в секундах, до завершения попытки выполнить команду |
| *CommandType* | Как правило, задается значение *Text*(выполняет *CommandText* как выражение SQL) или *StoredProcedure*(выполняет хранимую процедуру, заданную в свойстве *CommandText)* |
| *Connection* | Задает объект *Connection,*который эта команда должна использовать |
| *Parameters* | Коллекция параметров команд. При выполнении параметризованных запросов или хранимых процедур в эту коллекцию необходимо добавить объекты *Parameter* |
| *Transaction* | *SqlTransaction,*в рамках которой выполняется *SqlCommand* |

**Табл. 2-3.** Распространенные методы объекта *Command*

|  |  |
| --- | --- |
| **Название** | **Описание** |
| *Cancel* | Пытается отменить выполнение команды |
| *ExecuteNon Query* | Выполняет выражения SQL или хранимые процедуры, которые не возвращают данные |
| *ExecuteReader* | Выполняет команды, возвращающие данные в виде таблицы (или строк) |
| *ExecuteScalar* | Выполняет выражения SQL или хранимые процедуры, которые возвращают одно значение. Если вызвать *ExecuteScalar с*выра- |
| *ExecuteXML Reader* | жением, возвращающим строки данных, запрос будет выполнен, но возвратит только первый столбец первой строки. Остальные столбцы или строки будут проигнорированы  Возвращает данные в формате XML. Возвращает объект *System. Xml Xml Reader* |

**Табл. 2-4.** Распространенные события объекта *Command*

|  |  |
| --- | --- |
| Имя | Описание |
| *Disposed* | Происходит, когда команда удаляется |
| *StatementCompleted* | Происходит только по завершении выполнения выражения SQL  (только *SqlCommand)* |

Класс *SqlCommand* используется для выполне­ния запросов к вашей БД. Чтобы выполнить запрос, свойству *Connection* объекта *SqlCommand* следует задать объект *Connection.* Объект *SqlConnection* предоставляет метод *CreateCommand,* позволяющий упростить данный процесс. Этот метод воз­вращает новый объект *SqlCommand,* уже инициализированный для использования вашего объекта *SqlConnection.*

Одно из преимуществ метода *CreateCommand* состоит в том, что он доступен во всех классах *Connection* в .NET Framework. Таким образом, этот подход дает возмож­ность создать объект *Command* для данного объекта *Connection* в обобщенной форме.

Объект *SqlConnection* также позволяет начинать транзакции. Метод *BeginTransaction* для объекта *SqlConnection* возвращает соединению новый открытый объект

**Создание и настройка объектов *Command***

Создайте объекты *Command,* создав экземпляр нужного объекта *Command* и установив свойства *CommandType*, *CommandText* и допустимый объект *Connection.*

***Создание объекта Command, выполняющего запрос SQL***

Чтобы выполнить команды, которые выполняют SQL-запросы к БД, ус­тановите:

*CommandType -* значение *Text,*

*CommandText —*строку SQL-запроса.

Следующий код показывает, как создать экземпляр объекта *Command,* кото­рый выполняет запрос SELECT, возвращающий всех клиентов образца базы данных Northwind:

SqlCommand CustomersCommand = new SqlCommand();

CustomersCommand.Connection = NorthwindConnection; CustomersCommand.CommandType = CommandType.Text;

CustomersCommand.CommandText = "SELECT CustomerlD,CompanyName FROM

Customers";

***Создание объекта Command, выполняющего хранимую процедуру***

Для выполнения команд, которые запускают существующие хранимые процедуры в базе данных, задайте свойству *CommandType* значение *StoredProcedure,* а свойству *CommandText* — имя хранимой процедуры, которую необходимо выполнить.

Следующий код показывает, как создать экземпляр объекта *Command,* который выполняет хранимую процедуру, названную TenMostExpensiveProducts, в образце базы данных Northwind:

SqlCommand TopTenCommand = new SqlCommand();

TopTenCommand.Connection = NorthwindConnection;

TopTenCommand.CommandType = CommandType.StoredProcedure; TopTenCommand.CommandText = "TenMostExpensiveProducts"

**Получение данных из БД с помощью объекта Command**

Объект *Command* имеет следующие методы для выполнения запросов и хранимых процедур:

|  |  |
| --- | --- |
| *int ExecuteNonQuery();* | Запрос или хранимая процедура не возвращают никаких данных, например, запрос типа INSERT, DELETE, UPDATE, CREATE TABLE и др. Неявно возвращает число обработанных записей в БД |
| *(datatype) ExecuteScalar();* | Запрос или хранимая процедура возвращают единственное значение. Результаты вызова *ExecuteScalar необходимо привести* к ожидаемому типу данных (*datatype*) |
| *DataReader ExecuteReader();* | Запрос или хранимая процедура возвращают набор строк в объекте типа DataReader |

**Проверка схемы набора результатов**

Во время написания кода вы можете и не знать схемы набора результатов, например, если создаете специальное средство запроса, в котором пользователь задает запрос, или если вы используете запросы типа *SELECT* \* *FROM,* а администратор БД добавил в запрашиваемую вами таблицу дополнительные столбцы.

Если вы по каким бы то ни было причинам заранее не знаете схемы набора ре­зультатов, для ее определения можно воспользоваться разными методами объекта *SqlDataReader.*

|  |  |
| --- | --- |
| Метод *SqlDataReader* | Описание |
| *FieldCount* | число возвращаемых запросом полей (определить число возвращенных строк – невозможно) |
| *GetName* | название поля |
| *GetFieldType* | тип данных |
| *GetOrdinal* | определение порядкового номера поля по имени |
| *GetShemaTable* | возвращает информацию схемы в виде таблицы (рис. 2-1) |

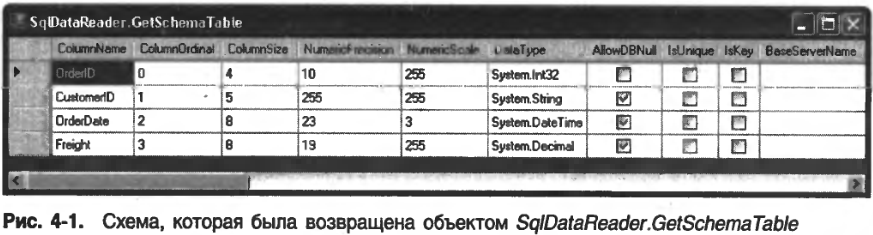


Рис.2-1. Схема, которая была возвращена методом SqlDataReader.GetShemaTable

//пример получения параметров *FieldCount, GetName, GetFieldType, GetDataTypeName схемы данных*

string strConn, strSQL;

strConn = @"Data Source=.\SQLExpress;"Initial Catalog=Northwind; "+

" Trusted\_Connection=Yes";

strSQL = "SELECT OrderID, CustomerID, OrderDate, Freight FROM Orders";

SqlConnection cn = new SqlConnection(strConn);

cn.0pen();

SqlCommand cmd = new SqlCommand(strSQL, cn);

// значение CommandBehavior.SchemaOnly позволяет читать только схему данных

SqlDataReader rdr = cmd.ExecuteReader(CommandBehavior.SchemaOnly);

for (int intField = 0; intField < rdr.FieldCount; intField++)

{

Console.WriteLine("Field #{0}", intField);

Console.WriteLine(" Name; {0}', rdr.GetName(intField));

Console.WriteLine(" .NET DataType: {0}",rdr.GetFieldType(intField).Name);

Console.WriteLine(" Database DataType: {0}",rdr.GetDataTypeName(intField));

Console.WriteLine();

}

rdr.Close();

SqlCommand ExecuteXMLCommand = new SqlCommandO; ExecuteXM [.Command.Connection = NorthwindConnection; ExecuteXMLCommand.CommandType = CommandType.Text;

// Добавление директивы For XML Auto для получения данных в формате XML ExecuteXMLCommand.CommandText = "SELECT CustomerlD FROM Customers ” +

" For XML Auto";

ExecuteXMLCommand.Connection.Open();

System.Xml.XmlReader reader = ExecuteXMLCommand.ExecuteXmlReader();

// Добавьте здесь код для итерации по XMLReader

reader. Close();

ExecuteXMLCommand.Connection.Close();

**Выполнение нескольких SQL-запросов с помощью *DataReader***

Объекты *Command* и *DataReader* можно также использовать для получения резуль­татов нескольких SQL-запросов. Для этого задайте свойству *CommandText* объекта *Command* несколько SQL-запросов, разделенных точкой с запятой (;). После вызова метода *Execute Reader* объект *DataReader* будет хранить число наборов результатов, для каждого выполненного SQL-запроса. Для получения доступа к данным, возвра­щенным дополнительными запросами, используйте метод *NextResult объекта DataReader.*

К примеру, следующий код создает *SqlCommand* и задает ему два отдельных SQL-запроса, которые получают данные из разных таблиц. Чтобы получить доступ к дополнительным данным, проверьте значение метода *DataReader.NextResult().* Если он возвратит *True* — значит, в считывателе есть другой набор результатов; если же *False* — в считывателе больше нет набора результатов.

ExecuteSqlCommand.CommandText = "SELECT CustomerlD, CompanyName FROM " & "Customers; SELECT ProductName, UnitsInStock FROM Products";

SqlDataReader reader = ExecuteSqlCommand.ExecuteReader();

bool MoreResults = false;

do

{

while (reader.Read())

for (int i = 0; i < reader.FieldCount; i++)

{

results.Append(reader[i].ToString() + "\t");

results.Append(Environment.NewLine);

}

MoreResults = reader. NextResult();

} while (MoreResults);

## Выполнение выражений SQL и вызов хранимых процедур

1. Создайте приложение Windows и назовите его *ExecutingCommands.*
2. Добавьте в Forml TextBox и установите следующие свойства:
3. *Name* - ResultsTextBox
4. *Multiline* - True
5. *ScrollBars* - Both
6. Добавьте кнопку под TextBox и установите следующие свойства: a *Name* - ExecuteSqlButton

*Text -* Execute SQL

1. Добавьте вторую кнопку и установите следующие свойства:
2. *Name -* ExecuteSprocButton
3. *Text* - Execute Sproc
4. Добавьте третью кнопку и установите следующие свойства:
5. *Name* - CreateTableButton
6. *Text* - Create Table

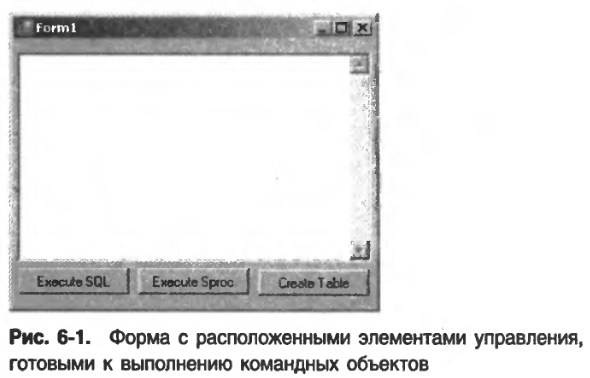


Рис. 2-2. Форма с расположенными элементами управления, готовыми к выполнению командных объектов

1. Теперь напишите код для выполнения команды, которая выполняет простй SQL-запрос SELECT и заполняет текстовое поле списком значений *CustomerlD* и *CompanyName* из таблицы Customers базы дан­ных Northwind. Вспомните, что при запуске выражений SQL необходимо вызвать метод *ExecuteReader* объекта *Command.* Вызов метода *Execute Reader* возвращает объект *DataReader* для доступа к данным, которые были получены с помощью выражения SQL. Объект *DataReader* представляет собой поток строк только для чтения, которые должны быть последовательно прочитаны во время передвижения с помощью курсора, чтение возможно только вперед. Потом нужно закрыть и удалить DataReader, вызвав метод DataReader.Close().

Дважды щелкните кнопку Execute SQL, чтобы создать обработчик события и открыть код.

Добавьте следующий код в класс *Form1,* затем добавьте ссылку на пространство имен *SqlClient* и создайте подключение с областью види­мости текущей формы к базе данных Northwind.

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using System.Data.SqlClient;

namespace ExecutingCommands

{

public partial class Forml : Form

{

private SqlConnection NorthwindConnection = new SqlConnection ("Data

Source=.\\sqlexpress;Initial Catalog=Northwind;

Integrated Security=True");

}

}

1. Добавьте следующий код в обработчик события *ExecuteSqlButton\_Click:*

// Создание StringBuilder для хранения результатов запроса

System.Text.StringBuilder results = new System.Text.StringBuilder();

// Создание экземпляра объекта Command

SqlCommand ExecuteSqlCommand = new SqlCommandO;

// Присвоение команды подключения к базе данных Northwind

ExecuteSqlCommand.Connection = NorthwindConnection;

// Выполнение инструкции SQL с помощью CommandType = Text

ExecuteSqlCommand.CommandType = CommandType.Text;

// Присвоение CommandText инструкции SQL, которую необходимо выполнить

ExecuteSqlCommand.CommandText =

"SELECT CustomerlD, CompanyName FROM Customers";

// Перед выполнением команды необходимо открыть подключение

ExecuteSqlCommand.Connection.Open();

// Присвоение результатов инструкции SQL объекту DataReader

SqlDataReader reader = ExecuteSqlCommand.ExecuteReader();

while (reader.Read())

{

for (int i = 0; i < reader.FieldCount; i++)

results.Append(reader[i].ToString() + "\t");

results.Append(Environment.NewLine);

}

// Закрытие объекта DataReader и подключения reader. Close();

reader. Close();

ExecuteSqlCommand.Connection.Close();

ResultsTextBox.Text = results.ToString();

1. Запустите приложение и щелкните кнопку Execute SQL (рис. 2-3).



Рис. 2-3. Форма, отображающая данные после выполнения команды SQL

Напишите код для выполнения команды, кото­рая запускает хранимую процедуру. Обратите внимание, что при запуске хранимых процедур, возвращающих строки, все равно необходимо вызывать метод *Execute Reader* объекта *Command.*

1. Создайте в БД хранимую процедуру Sproc, которая возвращает те же данные, что и запрос "SELECT CustomerlD, CompanyName FROM Customers"
2. Дважды щелкните кнопку Execute Sproc, чтобы создать обработчик события *Click* и открыть код.

Для этого обработчика события вы можете скопировать и повторно использовать большую часть кода из предыдущего примера. Как правило, в эффективном произ­водственном приложении вы бы переделали похожий код в отдельный метод, но в нашем случае просто для ясности скопируйте код, который необходимо повторно использовать, в дополнительные обработчики события *Click* кнопки.

Основное различие между вызовом команды SQL и вызовом хранимой процедуры состоит в том, что в последнем случае вы задаете свойству *CommandType* значение *StoredProcedure,* а свойству *CommandText —* имя хранимой процедуры. После этого вы вызываете метод *ExecuteReader* точно так же, как при выполнении выражения SQL, и обращаетесь к помощи считывателя таким же образом, как это делали ранее.

1. Добавьте следующий код в обработчик события *ExecuteSprocButtonjClick:*

System.Text.StringBuilder results = new System.Text.StringBuilder(); SqlCommand ExecuteSprocCommand = new SqlCommand(); ExecuteSprocCommand.Connection = NorthwindConnection; ExecuteSprocCommand.CommandType = CommandType.StoredProcedure; ExecuteSprocCommand.CommandText = "SProc"; ExecuteSprocCommand.Connection.Open();

SqlDataReader reader = ExecuteSprocCommand.ExecuteReader();

while (reader.Read())

{

for (int i = 0; i<reader.FieldCount;i++)

results.Append(reader[i].ToString() + "\t");

results.Append(Envi ronment.NewLine);

}

reader.Close();

ExecuteSprocCommand.Connection.Close();

ResultsTextBox.Text = results.ToString();

Напишите код для выполнения команды, которая создает новую таблицу в базе данных. Для выполнения операций с ка­талогом нужно применить метод *ExecuteNonQuery* объекта *Command.* Для этого метода нет необходимости использовать *Data Reader,* поскольку команда не воз­вратит никаких данных. После создания таблицы просто проверьте базу данных в Server Explorer и убедитесь в том, что команда была успешно выполнена.

1. Дважды щелкните кнопку Create Table, чтобы создать обработчик события *Click* и открыть код.
2. Добавьте в обработчик события *CreateTableButtonjClick* следующий код:

SqlCommand CreateTableCommand = new SqlCommand();

CreateTableCommand.Connection = NorthwindConnection;

CreateTableCommand.CommandType = CommandType.Text;

CreateTableCommand.CommandText =

"CREATE TABLE Salespersons ([SalesPersonID] [int] IDENTITY(1, 1) " +

"NOT NULL, [FirstName] [nvarchar](50) NULL, [LastName] " +

" [nvarchar](50) NULL)";

CreateTableCommand.Connection.Open();

CreateTableCommand.ExecuteNonQuery();

CreateTableCommand.Connection.Close();

1. Запустите приложение и щелкните кнопку Create Table. Перейдите в Server Explorer и обновите узел *Tables* базы данных Northwind. Убедитесь в существовании новой таблицы Salespersons.

**Работа с параметрами в командах SQL**

Что такое параметр и зачем его использовать

Параметр можно описать, как тип переменной, используемый для передачи и получе­ния значений между вашим приложением и базой данных. Так же, как и переменная в приложении, параметры создаются для содержания определенных типов данных, ко­торые задаются при помощи типов, указанных в перечислении *System.Data.SqlDbType.* Перечисление *SqlDbType* содержит список типов, доступных в SQL Server, в отличие от переменных приложения, которым обычно присваивается один из базовых типов данных .NET Framework.

Когда вы хотите быстро изменить критерий запроса, передайте значения пара­метров выражениям SQL (и хранимым процедурам). Как правило, параметры ис­пользуются в выражении WHERE SQL- запроса. Кроме того, параметры позволяют контролировать, как ввод пользователя заносится в запрос, и фактически исключают потенциальные атаки SQL- вставками.

ПРИМЕЧАНИЕ

SQL Server использует знак @ в качестве префикса для обозначения поименованных параметров, так что синтаксис типа @City в выражении SQL представляет собой параметр. Другие базы данных (например, источники данных Microsoft Office Access и OLE DB) не используют поименованные параметры. Вместо этого они представ­ляют параметры со знаком вопроса (?). При работе с этими типами источников данных для отслеживания того, какие значения используются для каждого параметра применяется порядок параметров.

К примеру, с помощью параметра можно передать значение для столбца City и быстро изменить результаты своего запроса, изменив значение параметра. Другими словами, установка различных значений параметру и выполнение запроса опреде­ляют набор результатов, который возвратит запрос. Это демонстрируют следующие запросы SQL:

-- Инструкция SQL с именованным параметром SELECT CustomerlD, CompanyName, City FROM Customers WHERE City = @City

— Инструкция SQL с неименованным параметром SELECT CustomerlD, CompanyName, City FROM Customers WHERE City = ?

**Типы параметров**

При выполнении объектов *Command* обычно используют параметры для передачи данных в базу. Такой тип параметра называют *Input.* Параметр для получения инфор­мации из базы данных относится к типу *Output.* Существует также третий тип пара метра — *Input/Output,* применяемый как для передачи, так и для получения данных при выполнении команды. Тип параметра указывается в свойстве *Direction* параметра и ему присваивается значение из перечисления *ParameterDirection.* Другими словами, при создании параметра его свойству *Direction* можно задать значение *Input, Output, InputOutput* или *ReturnValue.*

ПРИМЕЧАНИЕ

По умолчанию параметры являются параметрами *Input.*

**Создание параметров**

Параметры создаются путем создания экземпляра класса *Parameter* и установки его имени и типа данных, которые должны совпадать с именем параметра и типом данных, ожидаемых источником данных. Вы также можете установить свойство *ParameterDirection* параметра, чтобы выбрать тип создаваемого параметра.

Следующий код создает параметр *Input.*

SqlParameter TotalCostParameter = new SqlParameter(); TotalCostParameter.ParameterName = "@TotalCost";

TotalCostParameter.SqlDbType = SqlDbType.Money;

//Приведенный ниже код создает параметр *Output.*

SqlParameter TotalCostParameter = new SqlParameter("@TotalCost",

SqlDbType.Money);

TotalCostParameter.Direction = ParameterDirection.Output;

**Добавление параметров объектам Command**

У объектов *Command* есть свойство *Parameters,* представляющее собой коллекцию параметров для этой команды (например, свойство *SqlParameter.Parameters).* После создания параметра его необходимо добавить в коллекцию *Parameters* объекта *Com­mand,* который будет выполнять запрос SQL или хранимую процедуру, исполь­зующую параметр.

Следующий код демонстрирует, как параметр добавляется в объект *Command* (если предположить, что *GetCostCommand* уже существует):

GetCostCommand.Parameters.Add(TotalCostParameter);

## Создание и выполнение параметризованного SQL-запроса

Вы создадите в этом упражнении форму, выполняющую параметризованный запрос и дающую возможность пользователю вводить в текстовое поле значение, которое будет передаваться базе данных в качестве параметра запроса.

1. Создайте новое приложение Windows и назовите его *ParameterizedQueries.*
2. Добавьте в форму элемент управления *TextBox* и установите следующие свойс­тва:
3. *Name -* CityTextBox
4. *Text* - London
5. Добавьте *TextBox* и установите такие свойства:
6. *Name* - ResultsTextBox
7. *MuItiLine* - True
8. Добавьте кнопку и установите следующие свойства:
9. *Name* - ExecuteSqlButton
10. *Text* - Execute SQL

Теперь форма должна выглядеть, как на рис. 2-4.

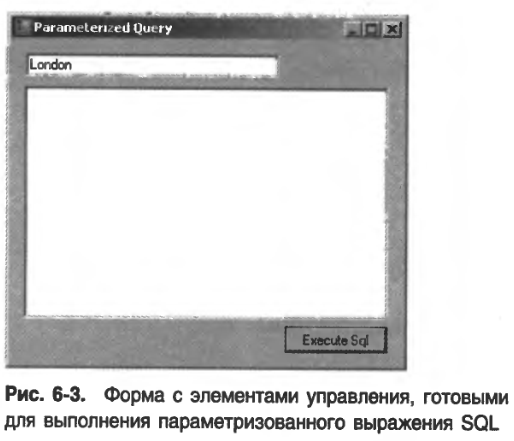


Рис. 2-4. Форма для выполнения параметризованного запроса

1. Дважды щелкните кнопку Execute SQL, чтобы создать обработчик события *Click* кнопки, и откройте код формы.
2. Добавьте ссылки на пространства имен *System.Data* и *System.Data.SqlClient.*
3. Добавьте код для создания подключения в форме.

На данном этапе код вашей формы должен выглядеть следующим образом (заме­ните *NorthwindConnection* правильной строкой подключения):

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using System.Data.SqlClient;

namespace ParameterizedQueries

{

public partial class Forml ; Form

{

private SqlConnection NorthwindConnection = new SqlConnection ("Data

Source=.\\sqlexpress;Initial Catalog= " +

"Northwind;Integrated Security=True");

public Form1()

{

InitializeComponent();

}

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

{

}

}

}

1. Добавьте представленный ниже код в метод *ExecuteSqlButton\_Click,* который создаст новый объект *Command* и задаст ему параметризованный запрос:

// Создание нового объекта Command

SqlCommand CustomersByCityCommand = new SqlCommand();

*//* Назначение свойств объекта Command

CustomersByCityCommand.Connection = NorthwindConnection; CustomersByCityCommand.CommandType = CommandType.Text; CustomersByCityCommand.CommandText =

"SELECT CustomerlD, CompanyName, City FROM Customers WHERE City = @City";

1. Чтобы создать параметр и задать его команде, добавьте после этого кода следующий (но в рамках обработчика события):

// Создание параметра @City

SqlParameter CityParameter = new SqlParameter();

// Назначение его имени и типа данных

CityParameter.РаrameterName = "@City";

CityParameter.SqlDbType = SqlDbType.NVarChar;

// Чтобы в столбце City базы данных могло содержаться значение null,

// мы задаем свойству IsNullable значение True

CityParameter.IsNullable = true;

// Добавление параметра к объекту Command

CustomersByCityCommand.Parameters.Add(CityParameter);

1. Теперь добавьте код, который задаст параметру значение, введенное в текстовое поле, коду задаст выполнение запроса и отобразит результаты в ResultsTextBox. Добавьте этот код под предыдущим, но продолжайте придерживаться обработчика события:

// Установка значений параметров текста в CityTextBox

CityParameter.Value = CityTextBox.Text;

// Создание StringBuilder для хранения результатов запроса System.Text.StringBuilder results =new System.Text.StringBuilder();

// Вы должны открыть подключение перед выполнением команды CustomersByCityCommand.Connection.Open();

// Добавление результатов инструкции SQL в DataReader

SqlDataReader reader = CustomersByCityCommand.ExecuteReaderO;

while (reader.Read())

{

for (int i=0; i< reader.FieldCount; i++)

results.Append(reader[i].ToString() + "\t");

results.Append(Environment.NewLine);

}

// Закрытие объекта DataReader и подключения

reader.Close();

CustomersByCityCommand.Connection.Close();

ResultsTextBox.Text = results.ToString();

1. Запустите приложение и щелкните кнопку Execute SQL. Приложение отобразит результаты запроса, как показано на рис2-5.

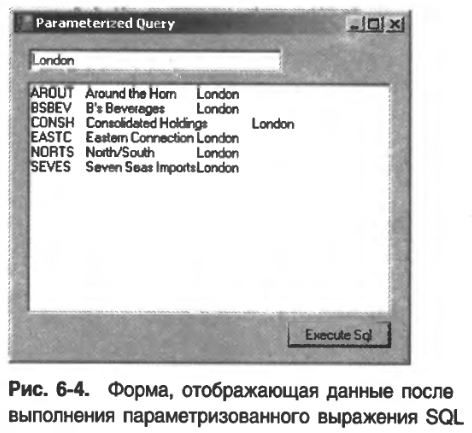


Рис. 2-5. Данные после выполнения запроса

1. Введите Madrid и повторно выполните запрос (щелкните кнопку Execute SQL).
2. Проверьте, чтобы результаты показывали клиентов только из того города, значение которого было передано параметру.

## Создание и выполнение параметризованной хранимой процедуры

1. Создайте новое приложение Windows и назовите его *ParameterizedStoredProcedure.*
2. Теперь добавьте в форму элемент управления *TextBox* и установите следующие свойства:
3. *Name* - CategoryNameTextBox
4. *Text* - Beverages
5. Добавьте второй *TextBox* и установите указанные ниже свойства:
6. *Name* - OrdYearTextBox
7. *Text -* 1997
8. Добавьте третий *TextBox* и установите такие свойства:
9. *Name -* ResultsTextBox
10. *MultiLine* - True
11. *ScrollBars* - Both
12. Добавьте кнопку и установите следующие свойства:
13. *Name* - ExecuteStoredProcedureButton
14. *Text* - Execute Stored Procedure

Теперь форма должна выглядеть, как на рис. 2-6.

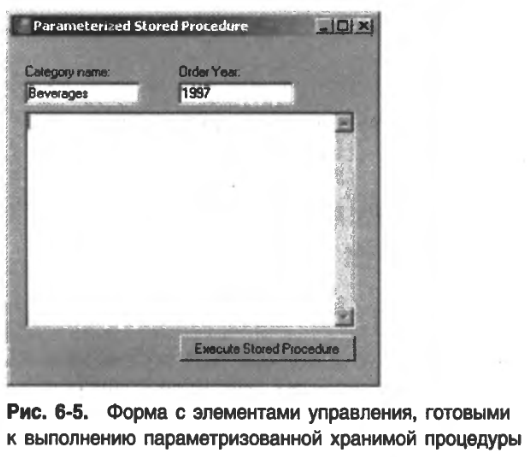


Рис. 2.6. Форма для выполнения параметризованной хранимой процедуры

1. Дважды щелкните кнопку Execute Stored Procedure, чтобы создать обработчик события *Click* кнопки, и откройте код формы.
2. Добавьте ссылки на пространства имен *System.Data* и *System.Data.SqlClient.*
3. Добавьте код для создания подключения в форме.

На данном этапе код вашей формы должен выглядеть следующим образом (замените *NorthwindConnection* правильной строкой подключения):

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using System.Data.SqlClient;

namespace ParameterizedStoredProcedureCS

{

public partial class Forml : Form

{

public Form1()

{

InitializeComponent();

}

private SqlConnection NorthwindConnection = new SqlConnection ("Data

Source=<ValidServerName>;Initial Catalog=" +

"Northwind;Integrated Security=True");

private void ExecuteStoredProcedureButton\_Click(object sender,

EventArgs e)

{

}

}

}

1. Добавьте представленный ниже код в метод *ExecuteStoredProcedureButton\_Click,* чтобы создать новый объект *Command* и задать ему хранимую процедуру Sales\_ByCategory:

// Создание нового объекта Command

SqlCommand SalesByCategoryCommand = new SqlCommandO;

// Назначение свойств объекта Command

SalesByCategoryCommand.Connection = NorthwindConnection; SalesByCategoryCommand.CommandType = CommandType.StoredProcedure; SalesByCategoryCommand.CommandText = "SalesByCategory";

1. Эта хранимая процедура принимает два параметра, поэтому, чтобы создать пара­метры и установить их команде, добавьте после предыдущего кода следующий:

// Создание параметра @CategoryName

SqlParameter CategoryNameParameter = new SqlParameter();

// Назначение его имени и типа данных

CategoryNameParameter.ParameterName = "@CategoryName"; CategoryNameParameter.SqlDbType = SqlDbType.NVarChar;

// Создание параметра @OrdYear

SqlParameter OrdYearParameter = new SqlParameter("@OrdYear",

SqlDbType.NVarChar);

// Добавление параметров к объекту Command

SalesByCategoryCommand.Parameters.Add(CategoryNameParameter);

SalesByCategoryCommand.Parameters.Add(OrdYearParameter);

1. Теперь добавьте код, который задаст параметрам значение, введенное в текстовое поле, коду задаст выполнение запроса и отобразит результаты в ResultsTextBox:

// Установка значений параметров текста в CategoryNameTextBox

// и OrdYearTextBox

CategoryNameParameter.Value = CategoryNameTextBox.Text;

OrdYearParameter.Value = OrdYearTextBox.Text;

// Создание StringBuilder для хранения результатов запроса System.Text.StringBuilder results = new System.Text.StringBuilder();

// Открытие подключения перед выполнением команды SalesByCategoryCommand.Connection.Open();

// Добавление результатов инструкции SQL в DataReader

SqlDataReader reader = SalesByCategoryCommand.ExecuteReader();

while (reader.Read())

{

for(int i = 0; i< reader.FieldCount; i++)

results.Append(reader[i].ToString() + "\t");

results.Append(Environment.NewLine);

}

// Закрытие DataReader и подключения

reader.Close();

SalesByCategoryCommand.Connection.Close();

ResultsTextBox.Text = results.ToString();

1. Запустите приложение и щелкните кнопку Execute Stored Procedure (рис. 2-7).

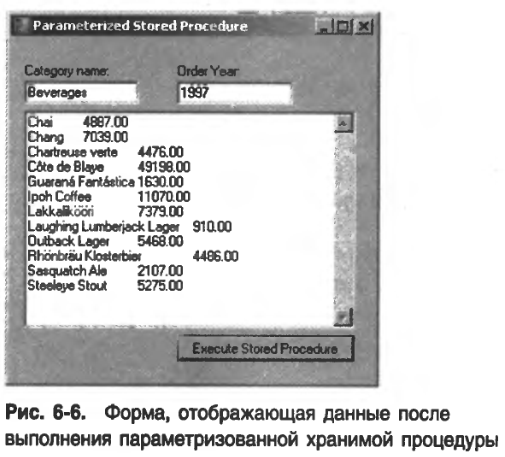


Рис. 2-7. Данные после выполнения параметризованной хранимой процедуры

1. Проверьте, отображают ли результаты список продукции указанной категории. Для этого введите имя другой категории (например, Condiments, Seafood или Produce) и выполните хранимую процедуру.

## Использование параметров *Input/Output* и *Output*

1. Создайте новое приложение Windows и назовите его InputOutputParameters.
2. Добавьте в форму элемент управления TextBox и установите следующие свойс­тва:
3. Name - OrderlDTextBox
4. Text - 10250
5. Добавьте второй TextBox и задайте его свойству Name значение FreightCostTextВох.
6. Добавьте кнопку и установите такие свойства:
7. Name - GetFreightCostButton
8. Text - Get Freight Cost

Под кнопкой добавьте второй набор элементов управления.

1. Добавьте элемент управления TextBox и установите указанные ниже свойства:
2. Name - CompanyNameTextBox
3. Text - Alfreds Futterkiste
4. Добавьте еще один TextBox и задайте его свойству Name значение ContactNameTextВох.
5. Добавьте кнопку и установите следующие свойства:
6. Name - GetContactNameButton
7. Text - Get Contact Name

Теперь форма должна выглядеть, как на рис. 2-8.

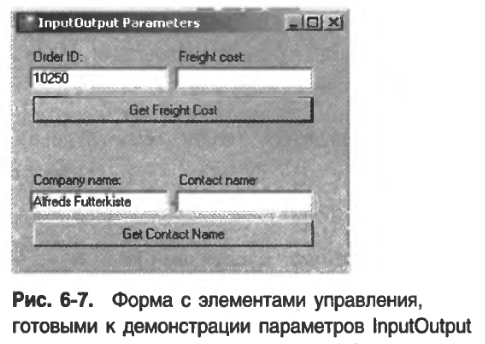


Рис. 2.8. Форма для демонстрации параметров InputOutput

1. Дважды щелкните кнопку Get Freight Cost, чтобы создать обработчик события.
2. Добавьте ссылки на пространства имен *System.Data* и *System.Data.SqlClient.*
3. Добавьте код для создания подключения в форме.

На данном этапе код вашей формы должен выглядеть следующим образом (заме­ните *NorthmndConnection* правильной строкой подключения):

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using System.Data.SqlClient;

namespace InputOutputParametersCS

{

public partial class Forml : Form

{

public Forml()

{

InitializeComponent();

}

private SqlConnection NorthwindConnection = new SqlConnection

("Data Source=<ValidServerName>;Initial Catalog=Northwind;" +

"Integrated Security=True");

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

{

}

}

}

1. Добавьте в обработчик события *GetFreigfrtCostButton* следующий код:

// Создание нового объекта Command

SqlCommand GetFreightCost = new SqlCommand();

// Установка его свойств

GetFreightCost.Connection = NorthwindConnection;

GetFreightCost.CommandType = CommandType.StoredProcedure;

GetFreightCost.CommandText = "GetFreightCost";

// Создание параметра Output, в котором будет содержаться стоимость груза SqlParameter FreightCostParameter = new SqlParameter(); FreightCostParameter.Direction = ParameterDirection.Output;

// Установка его имени и типа данных

FreightCostParameter.ParameterName = "@Freight";

FreightCostParameter.SqlDbType = SqlDbType.Money;

// Создание параметра OrderlD и установка его значения SqlParameter OrderIDParameter = new SqlParameter ("@0rderID", SqlDbType.Int);

OrderIDParameter.Value = OrderIDTextBox.Text;

*//* Добавление параметров к объекту Command

GetFreightCost.Parameters.Add(FreightCostParameter);

GetFreightCost.Parameters.Add(OrderIDParameter);

// Открытие подключения перед выполнением команды GetFreightCost.Connection.Open();

// Выполняем хранимую процедуру; так как мы используем параметры

// для доступа к данным, мы вызываем ExecuteNonQuery вместо ExecuteReader

GetFreightCost.ExecuteNonQuery();

Get FreightCost.Connection.Close();

FreightCostTextBox.Text = FreightCostParameter.Value.ToString();

1. Запустите приложение и щелкните кнопку Get Freight Cost.

Freight Cost TextBox отображает 65.83, стоимость перевозки для заказа номер 10250. Введите в OrderIdTextBox другие доступные номера OrderlD и запустите хранимую процедуру, чтобы проверить, содержат ли параметры вывода правильную стоимость перевозки для этих заказов.

Теперь, когда вы научились работать с параметрами вывода, возвращающими данные базы, реализуйте функциональность Get Contact Name, чтобы увидеть, как с помощью параметров *Input/Output* передавать данные в базу и получать их оттуда.

1. Дважды щелкните кнопку GetContactName для создания обработчика события
2. Добавьте в обработчик события GetContactName следующий код:

// Создание нового объекта Command

SqlCommand GetContactNameCommand = new SqlCommand();

// Установка его параметров

GetContactNameCommand.Connection = NorthwindConnection;

GetContactNameCommand.CommandType = CommandType.StoredProcedure;

GetContactNameCommand.CommandText = "GetContactName";

// Создание параметра InputOutput для отправки и получения данных

SqlParameter NameParameter = new SqlParameter();

NameParameter.Direction = ParameterDirection.InputOutput;

// Установка его имени, типа данных и значения

NameParameter.РаrameterName = "@Name";

NameParameter.SqlDbType = SqlDbType.NVarChar;

NameParameter.Value = CompanyNameTextBox.Text;

// Добавление параметров к объекту Command

GetContactNameCommand.Parameters.Add(NameParameter);

// Открытие подключения перед выполнением команды

GetContactNameCommand.Connection.Open();

// Выполнение хранимой процедуры

GetContactNameCommand.ExecuteNonQuery();

GetContactNameCommand.Connection.Close();

ContactNameTextBox.Text = NameParameter.Value.ToString();

1. Запустите приложение и щелкните кнопку Get Contact Name.

Contact name TextBox отображает запись с контактным именем для Alfreds Fut- terkiste, Maria Anders. Введите названия других доступных компаний из таблицы Customers — и Contact name TextBox отобразит их контакты.

**Контрольные вопросы**

1. Какие настройки *CommandType* необходимо использовать для выполнения выражения SQL, которое создает новую таблицу в базе данных?
2. Какой тип данных возвращается при вызове метода *ExecuteScalar* команды?
3. Какие три основных свойства необходимо задать объекту *Command* для вы­полнения?
4. Какие настройки свойства объекта *Command* нужны для выполнения хранимой

процедуры (выберите все, что применяется)?

1. *CommandType* = Text, *CommandText - имя\_хранимой\_процедуры.*
2. *CommandType* - Text, *CommandText* - *SQL-синтаксис\_для\_выполнения\_хранимой\_процедуры*.
3. *CommandType - StoredProcedure, CommandText - SQL-синтаксис для выполне- ния\_хранимой\_процедуры.*
4. *CommandType* - *StoredProcedure, CommandText* - *имя\_хранимой\_процедуры.*
5. Что необходимо сделать для доступа к полученным данным в виде таблицы после

начала выполнения команды, которая выполняется асинхронно (выберите все, что

применяется)?

1. Вызвать метод *EndExecuteNonQuery.*
2. Вызвать метод *EndExecuteReader.*
3. Подождать, пока не произойдет событие *StatementCompleted,* и воспользоваться *DataReader.*
4. Подождать, пока произойдет событие *StatementCompleted,* и вызвать метод *EndExecuteReader,* после чего воспользоваться *DataReader.*
5. Как с помощью одного *DataReader* можно выполнить несколько Select-запросов?
6. Вызвать метод *ExecuteReader* двух объектов *Command* и привязать результаты к одному и тому же экземпляру *DataReader.*
7. Дважды вызвать метод *ExecuteReader* одного объекта *Command.*
8. Задать свойству *Command.Command.Text* несколько выражений SQL, разделен­ных точкой с запятой.
9. Задать свойству *Command.Command.Type* несколько наборов результатов.
10. В каких случаях обычно используют параметр *Input* (выберите все, что применя­ется)?
11. Когда значение параметра создается на основе ввода пользователя.
12. При использовании параметра для передачи данных из приложения в базу.
13. Если команде задано выполнение выражения с оператором WHERE.
14. Когда значение параметра передано выражению Insert.
15. Каковы три основных типа параметров?
16. Input, Integer, String.
17. Integer, String, DateTime.
18. int, varchar, nvarchar.
19. Input, Output, InputOutput.
20. Как можно определить текущий тип данных *SqlParameter* (тип, ожидаемый SQL Server)?
21. Параметр представляет тип данных .NET Framework вашего приложения.
22. Команда ожидает тип столбца или данных в SQL Server.
23. Его представляет тип столбца в *DataTable.*
24. Это любой тип, указанный в перечислении *SqlDbDataType.*