**ADO.NET –** часть платформы .NET Framework, представляющая различные службы для доступа к реляционным данным.

Пользовательские приложения, могут использовать ADO.NET для соединения, обработки и обновления данных в различных источниках.

Архитектура ADO.NET ET

Двумя основными компонентами ADO.NET является **поставщик данных** .NET Framework и **автономная модель** хранения данных.

**Поставщик данных** .NET Framework используется для соединения с базой данных, выполнения команд и получение результатов выполнения команд.

**Автономная часть** архитектуры, представленная в виде класса DataSet, является расположенным в оперативной памяти кэшем данных, для хранения результатов, получаемых от поставщика данных.

Разделение архитектуры на две независимые части позволяет использовать технологию ADO.NET для построения многоуровневых приложений, а так же для создания приложений, использующих различные источники данных.

* **Data Source** – указывает имя экземпляра SQL Server, к которому нужно подключиться
* •**Initial Catalog** – параметр, указывающий на имя базы данных на сервере, к которой нужно подключиться
* •**Integrated Security** – позволяет использовать для подключения к серверу данные учетной записи Windows или имя входа SQL Server.
* •**User Id** – позволяет указать имя входа SQL Server для подключения к серверу
* **Password** – пароль имени входа SQL Server

**Пул соединений** снижает количество открытий новых соединений. **Пул** управляет соединениями с помощью поддержания набора активных соединений для каждой конфигурации данного соединения. Каждый раз, когда пользователь вызывает метод **Open** в соединении, организатор пулов ищет в пуле доступное соединение. Если соединение пула доступно, вместо открытия нового соединения он возвращает его участнику

string conStr = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB; Integrated Security=True"; //строка подключения

// string conStr = @"Data Source=(local)\SQLEXPRESS;Initial Catalog=ShopDB; Integrated Security=True";

// string conStr = @"Data Source=localhost\SQLEXPRESS;Initial Catalog=ShopDB; Integrated Security=True";

// Следую инструкциям строки подключения следует найти на локальном компьютере экземпляр SQL Server с именем SQLEXPRESS,

// поискать каталог ShopDB и попытаться получить доступ к источнику данных через доверительное подключение,

// используя для этого вашу учетную запись Microsoft Windows

SqlConnection connection = new SqlConnection(conStr);

try

{

connection.Open(); // открытие физического подключения к источнику данных

Console.WriteLine(connection.State);

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

connection.Close(); //закрытие физического соединения с источником данных

Console.WriteLine(connection.State);

}

\*\*\*

// Для демонстрации этой программы нужно создать базу данных EvelDB, а так же зарегистрировать пользователя SQLServer

// Пошаговая инструкция по созданию базы данных и пользователя находится в файле "CreatingEvelDB.xps".

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

{

string conStr = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;" + // построение строки подключения

"User ID=" + userNameTextBox.Text + ";" +

"Password=" + passwordTextBox.Text + ";";

using (SqlConnection connection = new SqlConnection(conStr))

{

try

{

connection.Open();

MessageBox.Show("Connection opened to " + connection.Database); // вывод на экран информации о подключении к базе данных

}

catch (Exception exception)

{

MessageBox.Show(exception.Message);

}

}

\*\*\*

ConnectionStringBuilder

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

{

SqlConnectionStringBuilder connectionStringBuilder = new SqlConnectionStringBuilder(); // создание конструктора строк подключения

#if SetProperty

connectionStringBuilder.DataSource = @".\SQLEXPRESS"; // используйте конструктор строк подключения для

connectionStringBuilder.InitialCatalog = "ShopDB"; // предотвращения изменения пользователем структуры строки подключения

connectionStringBuilder.UserID = userNameTextBox.Text;

connectionStringBuilder.Password = passwordTextBox.Text;

#else

connectionStringBuilder["Data Source"] = @".\SQLEXPRESS"; // используйте конструктор строк подключения для

connectionStringBuilder["Initial Catalog"] = "ShopDB"; // предотвращения изменения пользователем структуры строки подключения

connectionStringBuilder["User ID"] = userNameTextBox.Text;

connectionStringBuilder["Password"] = passwordTextBox.Text;

#endif

using (SqlConnection connection = new SqlConnection(connectionStringBuilder.ConnectionString))

{

try

{

connection.Open();

MessageBox.Show("Connection opened to " + connection.Database);

}

catch (Exception exception)

{

MessageBox.Show(exception.Message);

}

}

}

\*\*\*

Pool

string conStr = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB; Integrated Security=true; Pooling = true"; // включение или отключение пула для этого подключения

DateTime start = DateTime.Now;

for (int i = 0; i < 1000; i++)

{

SqlConnection connection = new SqlConnection(conStr);

connection.Open(); // при включенном пуле физическое соединение не создается, а берется из пула соединений

connection.Close(); // при включенном пуле физическое соединение не разрывается, а помещается в пул

}

TimeSpan stop = DateTime.Now - start;

Console.WriteLine(stop.TotalSeconds);

\*\*\*

App config

конфигурационного файла

var setting = new ConnectionStringSettings

{

Name = "MyConnectionString1", //имя строки подключения в конфигурационном файле

ConnectionString = @"Data Source=.\SQLEXPRESS;

Initial Catalog=ShopDB;

Integrated Security=True;"

};

Configuration config; // Объект Config представляет конфигурационный файл

config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None); // Объект ConfigurationManager предоставляет доступ к файлам конфигурации

config.ConnectionStrings.ConnectionStrings.Add(setting);

config.Save();

Console.WriteLine("Строка подключения записана в конфигурационный файл.");

// Получение строки подключения.

Console.WriteLine(ConfigurationManager.ConnectionStrings["MyConnectionString1"].ConnectionString);

}

\*\*\*

Шифрование строки

Configuration config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);

config.ConnectionStrings.ConnectionStrings.Add(new ConnectionStringSettings("COnnectionStr1", "SomeConnectionString"));

config.Save();

ConnectionStringsSection section = config.GetSection("connectionStrings") as ConnectionStringsSection;

if (section.SectionInformation.IsProtected)

{

// Расшифровать секцию

section.SectionInformation.UnprotectSection();

}

else

{

// Зашифровать секцию.

section.SectionInformation.ProtectSection(

"DataProtectionConfigurationProvider");

}

// Сохранить файл конфигурации.

config.Save();

// Проверка шифрования

Console.WriteLine("Protected={0}", section.SectionInformation.IsProtected);

Console.WriteLine(ConfigurationManager.ConnectionStrings["COnnectionStr1"].ConnectionString);

}

\*\*\*

Ole Db

using System.Data.OleDb;

class Program

{

static void Main(string[] args)

{

ConnectToAccessDB();

ConnectToExcelDB();

}

// Подключение к Microsoft Access

private static void ConnectToAccessDB()

{

var conn = new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0; Data Source= D:\ADO.NET\DATA\Access.mdb");

try

{

conn.Open();

Console.WriteLine("Connection to .mdb(AccessDB) file opened successfully");

}

catch (Exception e)

{

Console.WriteLine("Error: " + e.Message);

}

finally

{

conn.Close();

Console.WriteLine("Connection closed");

}

}

// Подключение Microsoft Excel

private static void ConnectToExcelDB()

{

var conn = new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0; Data Source=D:\ADO.NET\DATA\Excel.xls; Extended Properties=""Excel 8.0""");

try

{

conn.Open();

Console.WriteLine("Connection to .xls(ExcelDB) file opened successfully");

}

catch (Exception e)

{

Console.WriteLine("Error: " + e.Message);

}

finally

{

conn.Close();

Console.WriteLine("Connection closed");

}

}

\*\*\*

Команды

**SqlCommand –** объект подключаемой части технологии ADO.NET, позволяющий выполнять инструкции T-SQL над источником данных.

После создания объекта

**SqlCommand**

можно выполнять

T

-

SQL

инструкции

над

источником данных, для чего объект

**SqlCommand**

имеет ряд методов,

позволяющих выполнять разные типы инструкций.

Методы объекта

**SqlCommand**

для выполнения команд:

* **1)ExecuteNonQuery()** – этот метод предназначен для выполнения команд, не возвращающих значения. Такие команды могут выполнять следующие инструкции T-SQL: INSERT, DELETE, ALTER, DROP, CREATE.
* **2)ExecuteScalar() –** этот метод предназначен для выполнения команд, возвращающих скалярные значения.
* **3)EcecuteReader() –** этот метод предназначен для выполнения команд, возвращающих данные в табличном представлении(SELECT).
* Объект **DataReader** представляет собой аналог пожарного курсора только для чтения, позволяющий просматривать табличные данные, которые возвращает команда.
* Получить объект **DataReader** можно с помощью метода **ExecuteReader()** объекта **Comman**

\*\*\*

string conStr = @"Data Source=.\SQLEXPRESS; Initial Catalog=ShopDB; Integrated Security=True;"; // создание строки подключения

SqlConnection connection = new SqlConnection(conStr);

//\*\*\*\*\*\*\*\*\*\*Первый способ\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SqlCommand cmd = new SqlCommand();

cmd.Connection = connection;

cmd.CommandText = "Some T-SQL Command";

//\*\*\*\*\*\*\*\*\*\*Второй способ\*\*\*\*\*\*\*\*\*\*\*\*\*\*

cmd = connection.CreateCommand();

cmd.CommandText = "Some T-SQL Command";

//\*\*\*\*\*\*\*\*\*\*Третий способ\*\*\*\*\*\*\*\*\*\*\*\*\*\*

cmd = new SqlCommand("Some T-SQL Command", connection);

\*\*\*

Execute

static void Main(string[] args)

{

string conStr = @"Data Source=.\SQLEXPRESS; Initial Catalog=ShopDB; Integrated Security=True;"; // создание строки подключения

SqlConnection connection = new SqlConnection(conStr);

connection.Open(); // открытие подключения

SqlCommand cmd = new SqlCommand("SELECT Phone FROM Customers WHERE CustomerNo = 1", connection); // создание команды, возвращающей скалярное значение

string phoneNumber = (string)cmd.ExecuteScalar(); // выполнение команды

Console.WriteLine(phoneNumber);

}

\*\*\*

ExecuteNonQuery

SqlCommand insertCommand = connection.CreateCommand(); // создание команды на вставку данных

insertCommand.CommandText = "INSERT Customers VALUES ('Alex', 'Petrov', 'Petrovich', 'Заворотная 7', NULL, 'Kiyv', '(063)8569584', '2010-01-01')";

int rowAffected = insertCommand.ExecuteNonQuery(); // выполнение команды на вставку

Console.WriteLine("INSERT command rows affected: "+rowAffected);

// DELETE command

SqlCommand deleteCommand = connection.CreateCommand(); // создание команды на удаление данных

deleteCommand.CommandText = "DELETE Customers WHERE Phone = '(063)8569584'";

rowAffected = deleteCommand.ExecuteNonQuery(); // выполнение команды на удаление

Console.WriteLine("DELETE command rows affected: " + rowAffected);

connection.Close();

\*\*\*

SqlDataReader

SqlCommand cmd = new SqlCommand("SELECT \* FROM Customers", connection); // построение команды, возвращающей данные в табличном представлениии

SqlDataReader reader = cmd.ExecuteReader();

// с помощью объекта SqldataReder можно просматривать резельтаты запроса строка за строкой

// метод Read() возвращает значение true или false в зависимости от того, достигнут ли конец пакета строк, пришедших от сервера

// так же метод Read при каждом его вызове перемещается к следующей строке пакета, пришедшего от сервера

while (reader.Read())

{

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

{

Console.WriteLine(reader.GetName(i) + ": " + reader[i]);

}

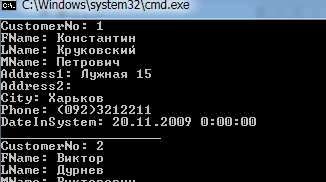
Console.WriteLine(new string('\_', 20));

}

reader.Close();

connection.Close();

}



\*\*\*

GetFieldValue

SqlCommand cmd = new SqlCommand("SELECT \* FROM Customers", connection); // построение команды, возвращающей данные в табличном представлениии

connection.Open();

SqlDataReader reader = cmd.ExecuteReader(); // метод ExecuteReader возвращает

// с помощью объекта SqldataReder можно просматривать резельтаты запроса строка за строкой

// метод Read() возвращает значение true или false в зависимости от того, имеется ли следующая строка, которую можно посмотреть

// так же метод Read при каждом его вызове перемещается к следующей строке набора строк, пришедших тот сервера

while (reader.Read())

{

Console.WriteLine(reader.GetFieldValue<int>(0)); // вывод на экран ID клиента испльзуя метод GetFieldValue

Console.WriteLine( // вывод на экран ФИО клиента испльзуя метод GetString

reader.GetString(2)+" "+

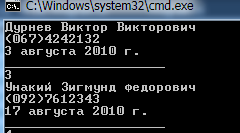
reader.GetString(1)+" "+

reader.GetString(3)

);

Console.WriteLine(reader.GetFieldValue<string>(7)); // вывод на экран номера телефона клиента

Console.WriteLine("{0:D}",reader.GetDateTime(8)); // вывод на экран поля DataInSystem клиента



\*\*\*

Индексаторы

SqlCommand cmd = new SqlCommand("SELECT \* FROM Customers", connection); // построение команды, возвращающей данные в табличном представлениии

// метод ExecuteReader возвращает новый объект SqlDataReader

using ( SqlDataReader reader = cmd.ExecuteReader())

{

// с помощью объекта SqldataReder можно просматривать резельтаты запроса строка за строкой

// метод Read() возвращает значение true или false в зависимости от того, имеется ли следующая строка, которую можно посмотреть

// так же метод Read при каждом его вызове перемещается к следующей строке набора строк, пришедших тот сервера

while (reader.Read())

{

Console.WriteLine(reader[0]); // вывод на экран ID клиента испльзуя перегрузку оператора с целочисленным индексом

Console.WriteLine( // вывод на экран ФИО клиента испльзуя перегрузку оператора со строковым индексом

reader["LName"] + " " +

reader["Fname"] + " " +

reader["MName"]

);

Console.WriteLine(reader[7]); // вывод на экран номера телефона клиента

Console.WriteLine("{0:D}", reader[8]); // вывод на экран поля DataInSystem клиента

//Console.WriteLine(reader.GetFieldValue<DateTime>(8));

Console.WriteLine(new string('\_', 20));

}

} // при выходе из блока using redader будет закрываться автоматически

connection.Close();

}

\*\*\*

Async

async private void getdataAsyncButtom\_Click(object sender, EventArgs e)

{

using (SqlConnection connection = new SqlConnection(conStr))

{

await connection.OpenAsync(); // асинхронное открытие соединения

SqlCommand cmd = new SqlCommand("WAITFOR DELAY '00:00:10'", connection); // асинхронное выполнение команды

await cmd.ExecuteNonQueryAsync();

MessageBox.Show("Command executed async");

}

}

\*\*\*

NextResult

SqlCommand cmd = new SqlCommand("SELECT \* FROM Customers WHERE CustomerNo = 1; SELECT \* FROM Employees WHERE EmployeeID = 1;", connection); // создание пакета запросов

SqlDataReader reader = cmd.ExecuteReader();

Console.WriteLine("press any key to see data from Customers");

Console.ReadKey();

WriteReaderData(reader); // вывод на экран данных

Console.WriteLine("press any key to see data from Employees");

Console.ReadKey();

reader.NextResult(); // переход к следующему запросу

WriteReaderData(reader); // вывод данных на экран

public static void WriteReaderData(DbDataReader reader)

{

while (reader.Read()) // вывод данных возвращаемых вторым запросом

{

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

Console.WriteLine(reader.GetName(i)+": "+reader[i]);

Console.WriteLine(new string('\_', 20));

}

}

\*\*\*

Проверка на Null

while (reader.Read())

{

Console.WriteLine("Name: "+reader.GetString(2) + " " + reader.GetString(1) + " " + reader.GetString(3));

//if (reader[5] == DBNull.Value) //ошибка времени выполнения.

if (reader.IsDBNull(5)) // метод IsDbNull позволяет проверить наличие данных в указанном поле источника данных

Console.WriteLine("Address Line 2: " + "No Data");

else

Console.WriteLine("Address Line 2: " + reader[5]);

Console.WriteLine();

}

\*\*\*

Транзакции и уровни изолирования

SqlCommand cmd = new SqlCommand("UPDATE Customers SET Phone = 'TEST' WHERE CustomerNo = 1", connection);

//cmd = new SqlCommand("UPDATE Customers SET Phone = '(052)1245789' WHERE CustomerNo = 1", connection);

try

{

connection.Open();

cmd.Transaction = connection.BeginTransaction();

cmd.ExecuteNonQuery();

throw new Exception();

cmd.Transaction.Commit();

Console.WriteLine("Transaction commited");

}

catch (Exception)

{

cmd.Transaction.Rollback();

Console.WriteLine("Transaction rollback");

}

}

\*\*\*

Ждать окончания роботы первой транзакции

string conStr = @" Initial Catalog=ShopDB; Data Source=.\SQLEXPRESS; Integrated Security=True;"; // создание строки подключения

SqlConnection connection = new SqlConnection(conStr);

SqlCommand cmd = new SqlCommand("SELECT LName, FName, Phone FROM Customers", connection);

Console.WriteLine("Step 2. Press any key to read Customers");

Console.ReadKey();

connection.Open();

cmd.Transaction = connection.BeginTransaction(IsolationLevel.ReadCommitted);

SqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

Console.WriteLine("{0} {1}: {2}", reader[0], reader[1], reader[2]);

}

connection.Close();

}

\*\*\*

Параметризированный запрос

Console.WriteLine("Введите ID клиента");

var customerNo = Console.ReadLine();

// не используйте конкатенацию строк для запросов во избежание изменения структуры запроса пользователем

string commandStr = string.Format("SELECT \* FROM Customers WHERE CustomerNo = {0};", customerNo); // для создания параматризированного запроса используется метод string.Format

using (SqlConnection connection = new SqlConnection(conStr)) // создание подключения

{

connection.Open();

SqlCommand cmd = new SqlCommand(commandStr, connection);

using (SqlDataReader reader = cmd.ExecuteReader()) // выполнение запроса и чтение результатов

{

while (reader.Read())

{

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

Console.WriteLine("{0}: {1}", reader.GetName(i), reader[i]);

Console.WriteLine(new string('-', 20));

}

}

\*\*\*

SqlParametr

var conStr = @"Data Source=.\SQLEXPRESS; Initial Catalog=ShopDB; Integrated Security=True;"; // создание строки подключения

var commandStr = "SELECT \* FROM Customers WHERE CustomerNo = @CustomerNo;"; // строка с запросом

Console.WriteLine("Enter customer ID");

var customerNo = Console.ReadLine(); // получение ID клиента от пользователя

SqlConnection connection = new SqlConnection(conStr); // создание подключения

SqlCommand cmd = new SqlCommand(commandStr, connection); // создание команды

cmd.Parameters.AddWithValue("CustomerNo", customerNo); // добавление параметра в коллекцию параметров команды

connection.Open();

\*\*\*

Parametr

string conStr = @"Data Source=.\SQLEXPRESS; Initial Catalog=ShopDB; Integrated Security=True;"; // создание строки подключения

SqlConnection connection = new SqlConnection(conStr);

SqlCommand cmd = new SqlCommand("SET @Parameter = 2;", connection);

SqlParameter parameter = cmd.Parameters.Add(new SqlParameter("Parameter", System.Data.SqlDbType.Int));

parameter.Direction = System.Data.ParameterDirection.Output; // указание направления параметра

connection.Open();

cmd.ExecuteNonQuery();

Console.WriteLine("Parameter value: " + parameter.Value); // вывод на экран значения параметра после выполнения запроса

\*\*\*

Хранимые процедури

string conStr = @"Data Source=.\SQLEXPRESS; Initial Catalog=ShopDB; Integrated Security=True;"; // создание строки подключения

SqlConnection connection = new SqlConnection(conStr);

SqlCommand cmd = new SqlCommand("EXECUTE selectEmp", connection); // создание команды, выполняющей хранимую процедуру selectEmp

connection.Open();

SqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

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

Console.WriteLine("{0}: {1}", reader.GetName(i), reader[i]);

Console.WriteLine();

}

connection.Close();

\*\*\*

Хранимая процедура с параметром

// код хранимой процедуры selectEmp: CREATE proc dbo.proc\_p1 @EmployeeID nvarchar(50)

// AS

// SELECT \* from dbo.Employees

// WHERE EmployeeID = @EmployeeID

string conStr = @"Data Source=.\SQLEXPRESS; Initial Catalog=ShopDB; Integrated Security=True;"; // создание строки подключения

SqlConnection connection = new SqlConnection(conStr);

Console.WriteLine("Enter employeeID");

int employeeID = int.Parse(Console.ReadLine()); // получение данных от пользователя

SqlCommand cmd = new SqlCommand("proc\_p1", connection) { CommandType = System.Data.CommandType.StoredProcedure }; // создание команды, вызывающей хранимую процедуру

cmd.Parameters.AddWithValue("@EmployeeID", employeeID); // добавление одного параметра

connection.Open();

SqlDataReader reader = cmd.ExecuteReader(); // выполнение команды

while (reader.Read())

{

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

Console.WriteLine("{0}: {1}", reader.GetName(i), reader[i]);

Console.WriteLine();

}

\*\*\*

Return

// код хранимой процедуры CREATE PROCEDURE ProcedureReturnValue

// AS

// BEGIN

// return 1;

// END

string conStr = @"Data Source=.\SQLEXPRESS; Initial Catalog=ShopDB; Integrated Security=True;"; // создание строки подключения

SqlConnection connection = new SqlConnection(conStr);

SqlCommand cmd = new SqlCommand("ProcedureReturnValue", connection) { CommandType = System.Data.CommandType.StoredProcedure };

SqlParameter parameter = cmd.Parameters.Add(new SqlParameter());

parameter.Direction = System.Data.ParameterDirection.ReturnValue; // после выполнения комманды parameter будет содержать возвращаемое значение хранимой процедуры

connection.Open();

cmd.ExecuteNonQuery();

Console.WriteLine(parameter.Value);

\*\*\*

Для объекта **DataTable** можно задать следующие ограничения:

* **1)UniqueConstraint** - предоставляет ограничение на набор столбцов, в которых все значения должны быть уникальными. Следует пользоваться этим ограничением в том случае, когда необходимо гарантировать уникальность комбинаций значений различных полей таблицы
* **2)PrimaryKey** – особый вид ограничения на уникальность. Первичный ключ таблицы может быть только один
* **3)ForeignKeyConstraint** – ограничение, гарантирующее что нельзя создать строку в дочерней таблице, которая ссылается на несуществующую строку родительской таблицы.

\*\*\*

Автономная часть

DataTable table = new DataTable("MyFirstTable");

DataColumn firstColumn = new DataColumn("First Column", typeof(int));

DataColumn secondColumn = new DataColumn("Second column", typeof(string));

DataColumnCollection columnCollection = table.Columns;

columnCollection.AddRange(new DataColumn[]{firstColumn, secondColumn});

foreach (DataColumn column in table.Columns)

Console.WriteLine("{0}: {1};", column.ColumnName, column.DataType);

\*\*\*

New row

DataTable table = new DataTable();

table.Columns.Add(new DataColumn("Column1", typeof(int)));

table.Columns.Add(new DataColumn("Column2"));

DataRow newRow = table.NewRow();

newRow["Column1"] = 1; // индексатор объекта DataRow в качастве строкового индекса принимает имя поля в строке к которому нужно обратиться

//newRow[0] = 1;

newRow["COlumn2"] = "One";

//newRow[1] = "One"; // индексатор объекта DataRow в качастве целочисленного индекса принимает индекс поля в строке к которому нужно обратиться

Console.WriteLine("table.Rows.Count: " + table.Rows.Count); // выведется 0

table.Rows.Add(newRow); // строка становится строкой таблицы при добавлении её в коллекцию Rows таблицы

Console.WriteLine("table.Rows.Count: " + table.Rows.Count); // выведется 1

Console.WriteLine();

foreach (DataRow row in table.Rows)

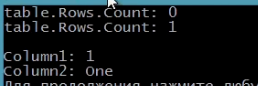
{

foreach (DataColumn column in table.Columns)

Console.WriteLine("{0}: {1}", column.ColumnName, row[column]);

}

}



\*\*\*

Создание таблиц по схеме

// This method creates new DataTable with schema same to SqlDataReader

private static DataTable CreateSchemaFromReader(SqlDataReader reader, string tableName)

{

DataTable table = new DataTable(tableName);

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

table.Columns.Add(new DataColumn(reader.GetName(i), reader.GetFieldType(i)));

return table;

}

// This method write data to DataTable whith same schema as DataReader

private static void WriteDataFromReader(DataTable table, SqlDataReader reader)

{

while (reader.Read())

{

DataRow row = table.NewRow();

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

row[i] = reader[i];

table.Rows.Add(row);

}

}

static void Main(string[] args)

{

string conStr = @"Data Source=.\SQLEXPRESS; Initial Catalog=ShopDB; Integrated Security=True;"; // создание строки подключения

SqlConnection connection = new SqlConnection(conStr);

connection.Open();

SqlCommand cmd = new SqlCommand("SELECT \* FROM Customers", connection);

SqlDataReader reader = cmd.ExecuteReader();

DataTable table = CreateSchemaFromReader(reader, "Customers"); // создание новой таблицы на основе схемы, предоставляемой DataReader

foreach (DataColumn column in table.Columns)

Console.WriteLine("{0}: {1}", column.ColumnName, column.DataType);

WriteDataFromReader(table, reader); // запись данных в таблицу с помощью DataReader

Console.WriteLine();

foreach (DataRow row in table.Rows)

{

foreach (DataColumn column in table.Columns)

Console.WriteLine("{0}: {1}", column.ColumnName, row[column]);

Console.WriteLine();

}

reader.Close();

connection.Close();

}

\*\*\*

GetSchemaTable

static void Main(string[] args)

{

string conStr = @"Data Source=.\SQLEXPRESS; Initial Catalog=ShopDB; Integrated Security=True;"; // создание строки подключения

SqlConnection connection = new SqlConnection(conStr);

connection.Open();

SqlCommand cmd = new SqlCommand("SELECT \* FROM Customers", connection);

SqlDataReader reader = cmd.ExecuteReader();

DataTable schemaTable = reader.GetSchemaTable(); //получение информации о схеме таблицы Customers

foreach (DataRow row in schemaTable.Rows) // вывод на экран информации, предоставляемой методом GetSchemaTable

{

foreach (DataColumn column in schemaTable.Columns)

Console.WriteLine("{0}: {1}", column.ColumnName, row[column]);

Console.WriteLine();

}

DataTable customers = new DataTable("Customers");

foreach (DataRow row in schemaTable.Rows)

{

var dataColumnToInsert = new DataColumn((string)row["ColumnName"], (Type)row["DataType"]);

customers.Columns.Add(dataColumnToInsert); // добавление столбцов в таблицу customers

}

Console.WriteLine(new string('-', 20));

foreach (DataColumn customersColumn in customers.Columns)

Console.WriteLine("{0}: {1}", customersColumn.ColumnName, customersColumn.DataType); // вывод имен и типов данных столбцов таблицы Customers

reader.Close();

connection.Close();

}

\*\*\*

**UniqueConstraint** - предоставляет ограничение на набор столбцов, в которых все значения должны быть уникальными. Следует пользоваться этим ограничением в том случае, когда необходимо гарантировать уникальность комбинаций значений различных полей таблицы

**PrimaryKey** – особый вид ограничения на уникальность. Первичный ключ таблицы может быть только один

**ForeignKeyConstraint** – ограничение, гарантирующее что нельзя создать строку в дочерней таблице, которая ссылается на несуществующую строку родительской таблицы.

ReadOnly

DataTable table = new DataTable();

DataColumn column = table.Columns.Add("ReadonlyColumn", typeof(string));

column.ReadOnly = true; // столбец таблицы c именем ReadonlyColumn доступен только для чтения

DataRow newRow = table.NewRow();

newRow[0] = "ReadonlyValue";

table.Rows.Add(newRow);

Console.WriteLine(table.Rows[0][0]);

table.Rows[0][0] = "NewValue"; // ОШИБКА времени выполнения

}

\*\*\*

AllowDBNull

DataTable table = new DataTable();

DataColumn column = table.Columns.Add("AllowDBNullColumn", typeof(int));

column.AllowDBNull = false;

DataRow newRow = table.NewRow();

newRow[0] = DBNull.Value;

table.Rows.Add(newRow); //ошибка времени выполнения

Console.WriteLine(table.Rows[0][0]);

\*\*\*

MaxLength

DataTable table = new DataTable();

DataColumn column = table.Columns.Add("MaxLengthConstraintColumn", typeof(string));

column.MaxLength = 5;

DataRow newRow = table.NewRow();

newRow[0] = "Some value";

table.Rows.Add(newRow); // ошибка времени выполнения

Console.WriteLine(table.Rows[0][0]);

\*\*\*

Unique

DataColumn column = table.Columns.Add("UniqueColumn", typeof(string));

column.Unique = true;

DataRow newRow = table.NewRow();

newRow[0] = "NonUniqueValue";

table.Rows.Add(newRow);

newRow = table.NewRow();

newRow[0] = "NonUniqueValue";

table.Rows.Add(newRow); // ошибка времени выполнения при нарушении ограничения Unique

Console.WriteLine(table.Rows[0][0]);

Console.WriteLine(table.Rows[1][0]);

PrimaryKey Constraint

DataTable table = new DataTable();

DataColumn column1 = table.Columns.Add("Column1", typeof(string));

DataColumn column2 = table.Columns.Add("Column2", typeof(string));

table.Constraints.Add(new UniqueConstraint(column1, false));

Console.WriteLine("is unique: " + table.Columns[0].Unique);

Console.WriteLine("Primary key columns count: "+ table.PrimaryKey.Length);

if (table.PrimaryKey.Length != 0)

Console.WriteLine("Primary key column name: " + table.PrimaryKey[0].ColumnName);

else

Console.WriteLine("Primary key column name: No data");

DataSet Foreign Key

DataSet – виртуальная бд

DataSet ds = new DataSet(); // создание DataSet

DataTable parentTable = new DataTable(); // родительская таблица

DataTable childTable = new DataTable(); // дочерняя таблица

DataColumn childColumn = childTable.Columns.Add("ChildColumn", typeof(int));

DataColumn parentColumn = parentTable.Columns.Add("ParentColumn", typeof(int));

// ограничение ForeignKeyConstraint будет работать если родительская и дочерняя таблица находятся в одном объекте DataSet

ds.Tables.AddRange(new DataTable[] { childTable, parentTable });

parentTable.Constraints.Add(new UniqueConstraint(parentColumn));

childTable.Constraints.Add(new ForeignKeyConstraint(parentColumn, childColumn));

DataRow parentRow = parentTable.NewRow();

parentRow[0] = 1;

parentTable.Rows.Add(parentRow);

DataRow childRow = childTable.NewRow();

childRow[0] = 1;

// после создания ограничения ForeignKeyConstraint нельзя добавлять в дочернюю таблицу строку, ссылающиеся на несуществующие строки из родительской таблицы

childRow[0] = 0;

childTable.Rows.Add(childRow);

Table Extension

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

{

string connectionString = @"Data Source=.\SQLEXPRESS; Initial Catalog=ShopDB; Integrated Security=True;"; // создание строки подключения

DataSet ds = new DataSet();

DataTable customers = new DataTable("Customers");

DataTable orders = new DataTable("Orders");

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

SqlCommand customersCmd = new SqlCommand("SELECT CustomerNo, LName, FName, Address1, Phone FROM Customers", connection);

SqlCommand ordersCmd = new SqlCommand("SELECT OrderID, CustomerNo, OrderDate FROM Orders", connection);

SqlDataReader ordersReader = ordersCmd.ExecuteReader(); // получение DataReader для таблицы OrderDetails

// метод LoadWithSchema позволяет на основе объекта DataReader создать объект DataTable

//с ограничениями для столбцов как в базе данных и заполнить эту таблицу данными

orders.LoadWithSchema(ordersReader);

ordersReader.Close();

SqlDataReader customersReader = customersCmd.ExecuteReader();

customers.LoadWithSchema(customersReader);

customersReader.Close();

}

// объект DataReader не имеет информации об ограничениях объектов DataTable, таких как

// UniqueConstraint, ForeignKeyConstraint и PrimaryKey, поэтому прийдется их создать вручную

customers.PrimaryKey = new DataColumn[] { customers.Columns[0] };

ds.Tables.AddRange(new DataTable[] { customers, orders });

// создание ограничения ForeignKeyConstraint для таблицы OrderDetails

var FK\_CustomersOrders = new ForeignKeyConstraint(customers.Columns["CustomerNo"], orders.Columns["CustomerNo"]);

orders.Constraints.Add(FK\_CustomersOrders);

parentGridView.DataSource = customers; // связывание элемента управления parentGridView с таблицей Products

childDataGridView.DataSource = orders; // Связывание элемента управления childDataGridView c таблицей OrderDetails

}

}

public static void LoadWithSchema(this DataTable table, SqlDataReader reader)

{

table.CreateSchemaFromReader(reader);

table.Load(reader); // Метод Load используется для загрузки в таблицу DataTable строк из источника данных.

}

// этот метод создает ограничения на столбцы таблице на основе полученного объекта DataReader

private static void CreateSchemaFromReader(this DataTable table, SqlDataReader reader)

{

DataTable schemaTable = reader.GetSchemaTable(); // Метод Возвращает таблицу описывающую метаданные столбца объекта SqlDataReader.

foreach (DataRow schemaRow in schemaTable.Rows)

{

DataColumn column = new DataColumn((string)schemaRow["ColumnName"]); // создание столбца с именем столбца в источнике данных

column.AllowDBNull = (bool)schemaRow["AllowDbNull"]; // получение значения свойства AllowDBNull

column.DataType = (Type)schemaRow["DataType"]; // получение значения свойства DataType

column.Unique = (bool)schemaRow["IsUnique"]; // получение значения свойства Unique

column.ReadOnly = (bool)schemaRow["IsReadOnly"]; // получение значения свойства Readonly

column.AutoIncrement = (bool)schemaRow["IsIdentity"]; // получение значения свойства AutoIncrement

if (column.DataType == typeof(string)) // если поле типа string

column.MaxLength = (int)schemaRow["ColumnSize"]; // получить значение свойства MaxLength

if (column.AutoIncrement == true) // Если поле с автоинкрементом

{ column.AutoIncrementStep = -1; column.AutoIncrementSeed = 0; } // задать свойства AutoIncrementStep и AutoIncrementSeed

table.Columns.Add(column); // добавить созданный столбец в коллекцию Columns таблицы

}

}

Элементы перечисления

**DataRowState**

* **1)Detached** - Строка была создана, но не является частью какой-либо **DataRowCollection**. Объект **DataRow** имеет это состояние сразу после своего создания и перед добавлением в коллекцию, а также если он был удален из коллекции.
* **2)Unchanged** - Строка не была изменена с момента последнего вызова **AcceptChanges**.
* **3)Added** - Строка была добавлена в коллекцию **DataRowCollection** и метод **AcceptChanges** не был вызван.
* **4)Deleted** - Строка была удалена с помощью метода **Delete** объекта **DataRow**.
* **5)Modified** - Строка была изменена и объект **AcceptChanges** не был вызван.

\*\*\*

Row State Data Row

DataTable table = new DataTable();

table.Columns.Add(new DataColumn("Column1"));

DataRow row = table.NewRow();

row[0] = "SomeValue";

Console.WriteLine(row.RowState); // Detached

table.Rows.Add(row);

Console.WriteLine(row.RowState); // Added

table.AcceptChanges();

//table.RejectChanges();

Console.WriteLine(row.RowState); // Unchenged

\*\*\*

Row State

// Значение Unchanged свойства RowState строки говорит о том, что строка не была изменена с момента последнего вызова AcceptChanges.

namespace RowState

{

class Program

{

private static void LoadData(string commandString, string connectionString, DataTable table)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

SqlCommand cmd = new SqlCommand(commandString, connection);

SqlDataReader reader = cmd.ExecuteReader();

table.Load(reader); // метод Load при добавлении строк устанавливает им состояние Unchanged

reader.Close();

}

}

private static void SimpleLoadData(string commandString, string connectionString, DataTable table)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

SqlCommand cmd = new SqlCommand(commandString, connection);

SqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

var newRow = table.NewRow();

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

newRow[i] = reader[i];

table.Rows.Add(newRow);

}

reader.Close();

}

}

\*\*\*

Row State Deleted

DataTable table = new DataTable();

table.Columns.Add(new DataColumn("Column1"));

DataRow row = table.NewRow();

Console.WriteLine("Table rows count: " + table.Rows.Count);

Console.WriteLine("RowState: " + row.RowState);

Console.WriteLine();

row[0] = "SomeValue";

table.Rows.Add(row);

table.AcceptChanges();

Console.WriteLine("Table rows count: " + table.Rows.Count);

Console.WriteLine("RowState: " + row.RowState);//Unchanged

Console.WriteLine();

table.Rows[0].Delete(); // строка подготовлена к удалению. На самом деле она еще находится в коллекции строк объекта DataTable

Console.WriteLine("Table rows count: " + table.Rows.Count);

Console.WriteLine("RowState: " + row.RowState);

Console.WriteLine();

table.AcceptChanges(); // При вызове метода AcceptChanges строки подготовленные к удалению удаляются из коллекции строк таблицы

//table.RejectChanges(); // Метод RejectChanges может анулировать все изменения, произведенные со строками относительно последнего вызова метода AcceptChanges

Console.WriteLine("Table rows count: " + table.Rows.Count);

Console.WriteLine("RowState: " + row.RowState);

\*\*\*

Modified

DataTable table = new DataTable();

table.Columns.Add("Column1");

table.LoadDataRow(new[] { "Value" }, true);

table.Rows[0][0] = "NewValue";

Console.WriteLine("Before accept changes: "+table.Rows[0].RowState);

table.AcceptChanges();

Console.WriteLine("After accept changes: "+table.Rows[0].RowState);

\*\*\*

SetAdded

DataTable table = new DataTable();

table.Columns.Add("Column1");

table.LoadDataRow(new[] { "Value" }, true);

Console.WriteLine("Before call SetAdded() - "+table.Rows[0].RowState);

table.Rows[0].SetAdded();

Console.WriteLine("After call SetAdded() - " + table.Rows[0].RowState);

Begin Edit

DataTable table = new DataTable();

table.Columns.Add("Column1");

table.LoadDataRow(new[] { "Value" }, true);

Console.WriteLine("Initial value: "+table.Rows[0][0]);

table.Rows[0].BeginEdit();

table.Rows[0][0] = "NewValue";

Console.WriteLine("Purposed value: " + table.Rows[0][0]+ "\n");

Console.WriteLine("Accept changes? Yes/No");

var choice = Console.ReadLine();

switch (choice)

{

case "Yes":

table.Rows[0].EndEdit();

break;

case "No":

table.Rows[0].CancelEdit();

break;

default:

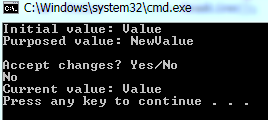
table.Rows[0].EndEdit();

break;

}

Console.WriteLine("Current value: " + table.Rows[0][0]);

}



\*\*\*

DataRowVersion

DataTable table = new DataTable();

table.Columns.Add("Column1");

table.LoadDataRow(new object[] { "one" }, true);

table.LoadDataRow(new object[] { "two" }, true);

table.LoadDataRow(new object[] { "three" }, true);

for (int i = 0; i < table.Rows.Count; i++)

table.Rows[i][0] = "ChangedValue";

foreach (DataRow row in table.Rows)

{

Console.WriteLine("Column1 current value: " + row[0, DataRowVersion.Current]); // будут выведены текущие значения строк

Console.WriteLine("Column1 original value: " + row[0, DataRowVersion.Original]); // будут выведены оригинальные значения строк

Console.WriteLine("RowState: " + row.RowState);

Console.WriteLine();

}

}

\*\*\*

DataAdapter

Удобное заполнение автономной модели

string connectionStr = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

SqlConnection connection = new SqlConnection(connectionStr);

SqlCommand cmd = new SqlCommand("SELECT \* FROM Customers", connection);

DataTable customers = new DataTable("Customers");

SqlDataAdapter adapter = new SqlDataAdapter(cmd); // одна из перегрузок конструктора DataAdapter принимает объект Command

//SqlDataAdapter adapter = new SqlDataAdapter("SELECT \* FROM Customers", connection);

adapter.Fill(customers); // метод Fill объекта DataAdapter позволяет заполнить таблицу данными

foreach (DataRow row in customers.Rows)

{

foreach (DataColumn column in customers.Columns)

Console.WriteLine("{0}: {1}", column.ColumnName, row[column]);

Console.WriteLine();

}

\*\*\*

DataAdapter Fill Grid

DataSet ds = new DataSet();

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Customers; SELECT \* FROM Employees";

SqlDataAdapter adapter = new SqlDataAdapter(commandString, connectionString);

adapter.Fill(ds); // заполнение DataSet данными с помощью DataAdapter

// Объект DataAdapter по умолчанию не создает ограничения для столбцов таблиц

label1.Text = ds.Tables[0].TableName; // какое имя у первой таблицы в объекте DataSet?

label2.Text = ds.Tables[1].TableName; // какое имя у второй таблице в объекте DataSet?

dataGridView1.DataSource = ds.Tables[0]; // Связывание первой таблицы DataSet с элементом управления dataGridView1

dataGridView2.DataSource = ds.Tables[1]; // Связывание второй таблицы DataSet с элементом управления dataGridView2

\*\*\*

Table Mapping Имя таблиц Grid

DataSet ds = new DataSet();

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Customers; SELECT \* FROM Employees";

SqlDataAdapter adapter = new SqlDataAdapter(commandString, connectionString);

adapter.TableMappings.Add("Table", "Customers"); // объект TableMapping стоит использовать для задания своих имен для таблиц, генерируемых адаптером данных

adapter.TableMappings.Add("Table1", "Employees"); //

adapter.Fill(ds);

label1.Text = ds.Tables[0].TableName; // какое имя у первой таблицы в объекте DataSet?

label2.Text = ds.Tables[1].TableName; // какое имя у второй таблице в объекте DataSet?

dataGridView1.DataSource = ds.Tables[0];

dataGridView2.DataSource = ds.Tables[1];

Collunm Mapping Имя столбцов

DataSet ds = new DataSet();

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Customers; SELECT \* FROM Employees";

SqlDataAdapter adapter = new SqlDataAdapter(commandString, connectionString);

adapter.MissingMappingAction = MissingMappingAction.Passthrough;

DataTableMapping customersMapping = adapter.TableMappings.Add("Table", "Customers");

var cusomersColumnMappings = new DataColumnMapping[]

{

new DataColumnMapping("CustomerNo", "ID"), // переименовать CustomerNo в ID

new DataColumnMapping("FName", "Имя"), // переименовать FName в Имя

new DataColumnMapping("LName","Фамилия"), // переименовать LName в Фамимлия

new DataColumnMapping("MName","Отчество"), // ....

new DataColumnMapping("Address1","Адрес1"),

new DataColumnMapping("Address2","Адрес2"),

new DataColumnMapping("City","Город"),

new DataColumnMapping("Phone","Номер телефона"),

new DataColumnMapping("DateInSystem","Дата регистрации")

};

customersMapping.ColumnMappings.AddRange(cusomersColumnMappings);

DataTableMapping employeesMapping = adapter.TableMappings.Add("Table1", "Employees");

var employeesColumnMappings = new DataColumnMapping[]

{

new DataColumnMapping("EmployeeID", "ID"),

new DataColumnMapping("FName", "Имя"),

new DataColumnMapping("LName","Фамилия"),

new DataColumnMapping("MName","Отчество"),

new DataColumnMapping("Salary","Ставка"),

new DataColumnMapping("PriorSalary","Премия"),

new DataColumnMapping("HireDate","Дата приема"),

new DataColumnMapping("TerminationDate","Дата увольнения"),

new DataColumnMapping("ManagerEmpID","ID менеджера")

};

employeesMapping.ColumnMappings.AddRange(employeesColumnMappings);

adapter.Fill(ds);

label1.Text = ds.Tables[0].TableName; // какое имя у первой таблицы в объекте DataSet?

label2.Text = ds.Tables[1].TableName; // какое имя у второй таблице в объекте DataSet?

dataGridView1.DataSource = ds.Tables[0];

dataGridView2.DataSource = ds.Tables[1];

\*\*\*

MissingSchemaAction Ограничение

DataSet ds = new DataSet();

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString =

"SELECT CUstomerNo, LName, FName, MName, Address1, Phone FROM Customers;"+

"SELECT OrderID, CustomerNo, OrderDate FROM Orders";

SqlDataAdapter adapter = new SqlDataAdapter(commandString, connectionString);

// используя свойство MissingSchemaAction можно заставить адаптер данных создавать ограничения для таблиц

// ограничения ForeignKeyConstraint с помощью адаптера данных создать нельзя

adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;

adapter.TableMappings.Add("Table", "Customers");

adapter.TableMappings.Add("Table1", "Orders");

adapter.Fill(ds);

// Ограничения уровня объекта DataTable прийдется создавать вручную

var FK\_CustomersOrders = new ForeignKeyConstraint("FK\_CustomersOrders",

ds.Tables[0].Columns["CustomerNo"],

ds.Tables[1].Columns["CustomerNo"]);

ds.Tables["Orders"].Constraints.Add(FK\_CustomersOrders);

label1.Text = ds.Tables[0].TableName; // какое имя у первой таблицы в объекте DataSet?

label2.Text = ds.Tables[1].TableName; // какое имя у второй таблице в объекте DataSet?

dataGridView1.DataSource = ds.Tables[0];

dataGridView2.DataSource = ds.Tables[1];

}

\*\*\*

Fill перегрузка PartOfRow

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Customers";

DataTable table = new DataTable();

SqlDataAdapter adapter = new SqlDataAdapter(commandString, connectionString);

int step = 2;

for (int i = 0; adapter.Fill( i, step, table) > 0; i += step)

{

Console.WriteLine(table.Rows.Count);

foreach (DataRow row in table.Rows)

{

foreach (DataColumn col in table.Columns)

Console.WriteLine("{0} {1}", col.ColumnName, row[col]);

Console.WriteLine();

}

Console.ReadKey();

Console.Clear();

}

\*\*\*

Пейджинг

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Customers";

SqlDataAdapter adapter;

int i = 0, step = 2;

public Form1()

{

InitializeComponent();

}

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

{

adapter = new SqlDataAdapter(commandString, connectionString);

DataTable table = new DataTable();

adapter.Fill(0, step, table);

dataGridView1.DataSource = table;

}

private void button1\_Click(object sender, EventArgs e)

{

this.NextPage(step);

}

private void button2\_Click(object sender, EventArgs e)

{

this.NextPage(-step);

}

private void NextPage(int step)

{

DataTable table = new DataTable();

if (step > 0)

adapter.Fill(i += step, step, table);

else

adapter.Fill(i += step, -step, table);

dataGridView1.DataSource = table;

}

}

**DataRelation** показывает отношение дочерний-родительский между двумя объектами **DataTable**, которые находятся в объекте **DataSet**.

Для создания отношения нужно указать имя отношения, ссылаемый столбец родительской таблицы и ссылающийся столбец дочерней таблицы

После создания связи между таблицами можно просматривать связанные данные, для чего у объекта **DataRow** есть ряд методов.

**GetChildRows** – этот метод позволяет получить массив **DataRow[]** дочерних записей относительно текущей строки.

**GetParentRow** – этот метод предназначен для получения единственной родительской строки относительно дочерней. Используется для таблиц со связью один ко многим.

**GetParentRows** – этот метод предназначен для получения массива **DataRow[]** родительских строк относительно текущей строки. Используется для таблиц со связью многие ко многим.

\*\*\*

DataRelation

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Products; SELECt \* FROM OrderDetails;";

DataSet shopDB = new DataSet("ShopDB");

SqlDataAdapter adapter = new SqlDataAdapter(commandString, connectionString);

adapter.Fill(shopDB); // заполнение DataSet

DataTable products = shopDB.Tables[0]; // получение ссылки на таблицу Products

DataTable orderDetails = shopDB.Tables[1]; // получение ссылки на таблицу OrderDetails

// создание отношения между таблицами Products и OrderDetails

var ProductsOrderDetailsRel = new DataRelation("Products\_OrderDetails", // имя отношения

products.Columns["ProdID"], // поле родительской таблицы

orderDetails.Columns["ProdID"], // поле дочерней таблицы

true); // создавать/не создавать ограничения

// после созания ограничения его нужно добавить в коллекцию Relations объекта DataSet, в которой содержаться таблицы

// без этого шага отношение не будет работать

shopDB.Relations.Add(ProductsOrderDetailsRel);

Console.WriteLine("Products primary key columns number: " + products.PrimaryKey.Length); // объект DataRelation не добавляет ограничение первичного ключа

Console.WriteLine("ProdID column Unique= " + products.Columns["ProdID"].Unique); //столбцу родительской таблицы добавлено ограничение на уникальность

Console.WriteLine("ProdID column AllowDBNull= " + products.Columns["ProdID"].AllowDBNull); //столбцу родительской таблицы добавлено ограничение на уникальность

var orderDetailsConstraint = orderDetails.Constraints[0] as ForeignKeyConstraint;

Console.WriteLine("OrderDetails foreign key constraint name: " + orderDetailsConstraint.ConstraintName); // добавлено ограничение ForeignKeyConstraint

}

\*\*\*

ChildRow

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Customers; SELECT \* FROM Orders;";

DataSet shopDB = new DataSet("ShopDB");

SqlDataAdapter adapter = new SqlDataAdapter(commandString, connectionString);

adapter.Fill(shopDB);

DataTable customers = shopDB.Tables[0];

DataTable orders = shopDB.Tables[1];

// создание связи таблицы Customers с таблицей Orders

shopDB.Relations.Add("Customers\_Orders", customers.Columns["CustomerNo"], orders.Columns["CustomerNo"]);

foreach (DataRow customerRow in customers.Rows)

{

// метод GetChaildRows получает дочерние строки в виде массива DataRow[]

DataRow[] chilRows = customerRow.GetChildRows("Customers\_Orders");

if (chilRows.Length != 0) // если существуют дочерние записи

{

Console.WriteLine("{0} {1} {2}", customerRow[2], customerRow[1], customerRow[3]);

foreach (DataRow ordersRow in chilRows)

Console.WriteLine("\tOrderId: {0}, OrderDate: {1};", ordersRow["OrderID"], ordersRow["OrderDate"]);

Console.WriteLine();

}

}

}

\*\*\*

ParentRow

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Customers; SELECT \* FROM Orders;";

DataSet shopDB = new DataSet("ShopDB");

SqlDataAdapter adapter = new SqlDataAdapter(commandString, connectionString);

adapter.Fill(shopDB);

DataTable customers = shopDB.Tables[0];

DataTable orders = shopDB.Tables[1];

shopDB.Relations.Add("Customers\_Orders", customers.Columns["CustomerNo"], orders.Columns["CustomerNo"]);

foreach (DataRow ordersRow in orders.Rows)

{

var customerRow = ordersRow.GetParentRow("Customers\_Orders"); // метод GetParrentRow возвращает одну строку

Console.WriteLine("OrderId: "+ordersRow["OrderID"]+"\n"+

"OrderDate: "+ ordersRow["OrderDate"]+"\n"+

"CustomerName: " + customerRow[2] +" "+ customerRow[1] +" "+ customerRow[3]);

Console.WriteLine();

}

\*\*\*

ToItSelf

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

DataSet shopDB = new DataSet("ShopDB");

SqlDataAdapter adapter = new SqlDataAdapter("SELECT \* FROM Employees", connectionString);

adapter.TableMappings.Add("Table", "Employees");

adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey; // копирование схемы и создание первичных ключей на таблицах

adapter.Fill(shopDB);

var employees = shopDB.Tables[0];

shopDB.Relations.Add("Employee\_Employee", employees.Columns["EmployeeID"], employees.Columns["ManagerEmpID"], false); // создание связи сам к себе

foreach (DataRow employee in employees.Rows)

{

DataRow manager = employee.GetParentRow("Employee\_Employee"); // получение непосредственного начальника

if (manager != null)

{

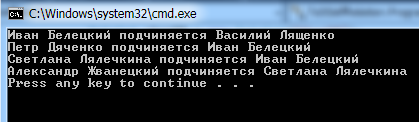
string empName = employee[1] + " " + employee[2];

string managerName = manager[1] + " " + manager[2];

Console.WriteLine(empName + " подчиняется " + managerName);

}

}



\*\*\*

Список подчинения

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

DataSet shopDB = new DataSet("ShopDB");

SqlDataAdapter adapter = new SqlDataAdapter("SELECT \* FROM Employees", connectionString);

adapter.TableMappings.Add("Table", "Employees");

adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;

adapter.Fill(shopDB);

var employees = shopDB.Tables[0];

// создание связи сам к себе

var employeeToHimSelf = shopDB.Relations.Add("Employee\_Employee", employees.Columns["EmployeeID"], employees.Columns["ManagerEmpID"]);

foreach (DataRow employee in employees.Rows)

{

ShowSubordinates(employee, employeeToHimSelf, ""); // вызов рекурсивного метода, выводящего план подчинения

Console.WriteLine();

}

}

private static void ShowSubordinates(DataRow employee, DataRelation relation, string indent)

{

Console.WriteLine(indent + employee[1] + " " + employee[2]);

indent += " "; // отступ

foreach (DataRow subordinate in employee.GetChildRows(relation))

ShowSubordinates(subordinate, relation, indent); // рекурсивный вызов метода для близжайшего подчиненного

}

}

\*\*\*

WriteXml Сериализация

static void Main(string[] args)

{

DataSet shopDB = CreateShopDBDataSet(); // создание базы данных ShopDB

shopDB.WriteXmlSchema(@”D:\ADO.NET\DATA\ShopDbSchema.xml”); // запсиь схемы ShopDB в XML файл

shopDB.WriteXml(@»D:\ADO.NET\DATA\ShopDBData.xml»); // запись данных ShopDB в XML файл

}

private static DataSet CreateShopDBDataSet()

{

string connectionString = @”Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True”;

string commandString = “SELECT \* FROM Customers;” +

“SELECT \* FROM Orders;” +

“SELECT \* FROM OrderDetails;” +

“SELECT \* FROM Products”;

DataSet shopDB = new DataSet(“ShopDB”);

// создание адаптера данных для базы данных ShopDB

SqlDataAdapter adapter = new SqlDataAdapter(commandString, connectionString);

adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;

// Маппинг таблиц для ShopDB

adapter.TableMappings.Add(“Table”, “Customers”);

adapter.TableMappings.Add(“Table1”, “Orders”);

adapter.TableMappings.Add(“Table2”, “OrderDetails”);

adapter.TableMappings.Add(“Table3”, “Products”);

// получение аднных с помощью адаптера данных

adapter.Fill(shopDB);

// получение ссылок на таблицы

var customers = shopDB.Tables[“Customers”];

var orders = shopDB.Tables[“Orders”];

var orderDetails = shopDB.Tables[“OrderDetails”];

var products = shopDB.Tables[“Products”];

// создание связей для таблиц

shopDB.Relations.Add(“Customers\_Orders”, customers.Columns[“CustomerNo”], orders.Columns[“CustomerNo”]);

shopDB.Relations.Add(“Orders\_OrderDetails”, orders.Columns[“OrderID”], orderDetails.Columns[“OrderID”]);

shopDB.Relations.Add(“Products\_OrderDetails”, products.Columns[“ProdID”], orderDetails.Columns[“ProdID”]);

return shopDB;

}

\*\*\*

ReadXml Десериализация

static void Main(string[] args)

{

DataSet shopDB = new DataSet();

shopDB.ReadXmlSchema(@"D:\ADO.NET\DATA\ShopDbSchema.xml"); // чтение схемы базы данных ShopDB

shopDB.ReadXml(@"D:\ADO.NET\DATA\ShopDBData.xml"); // чтение данных базы данных ShopDB

// получение ссылок на таблицы

var customers = shopDB.Tables["Customers"];

var orders = shopDB.Tables["Orders"];

var orderDetails = shopDB.Tables["OrderDetails"];

var products = shopDB.Tables["Products"];

ShowCustomersInfo(customers);

}

// создание метода, выводящего данные о продажах

private static void ShowCustomersInfo(DataTable customers)

{

foreach (DataRow customer in customers.Rows)

{

if (customer.GetChildRows("Customers\_Orders").Length != 0)

{

Console.WriteLine("{0} {1} {2}", customer[2], customer[1], customer[3]);

Console.WriteLine();

foreach (DataRow order in customer.GetChildRows("Customers\_Orders"))

{

Console.WriteLine("\t OrderID:{0}, {1:D}", order["OrderID"], order["OrderDate"]);

foreach (DataRow orderDetail in order.GetChildRows("orders\_OrderDetails"))

{

DataRow product = orderDetail.GetParentRow("Products\_OrderDetails");

Console.WriteLine("\t\t LineItem:{0} - {2}, {1:C}",

orderDetail["LineItem"],

orderDetail["TotalPrice"],

product["Description"].ToString().Trim());

}

Console.WriteLine();

}

Console.WriteLine(new string('-', 20));

}

\*\*\*

Many to Many

static void Main(string[] args)

{

DataSet shopDB = new DataSet();

shopDB.ReadXmlSchema(@"D:\ADO.NET\DATA\ShopDbSchema.xml"); // чтение схемы базы данных ShopDB

shopDB.ReadXml(@"D:\ADO.NET\DATA\ShopDBData.xml"); // чтение данных базы данных ShopDB

// получение ссылок на таблицы

var customers = shopDB.Tables["Customers"];

var orders = shopDB.Tables["Orders"];

var orderDetails = shopDB.Tables["OrderDetails"];

var products = shopDB.Tables["Products"];

ShowCustomersInfo(customers);

}

// создание метода, выводящего данные о продажах

private static void ShowCustomersInfo(DataTable customers)

{

foreach (DataRow customer in customers.Rows)

{

if (customer.GetChildRows("Customers\_Orders").Length != 0)

{

Console.WriteLine("{0} {1} {2}", customer[2], customer[1], customer[3]);

Console.WriteLine();

foreach (DataRow order in customer.GetChildRows("Customers\_Orders"))

{

Console.WriteLine("\t OrderID:{0}, {1:D}", order["OrderID"], order["OrderDate"]);

foreach (DataRow orderDetail in order.GetChildRows("orders\_OrderDetails"))

{

DataRow product = orderDetail.GetParentRow("Products\_OrderDetails");

Console.WriteLine("\t\t LineItem:{0} - {2}, {1:C}",

orderDetail["LineItem"],

orderDetail["TotalPrice"],

product["Description"].ToString().Trim());

}

Console.WriteLine();

}

Console.WriteLine(new string('-', 20));

}

\*\*\*

Relation Expression

static void Main(string[] args)

{

DataSet shopDB = new DataSet();

shopDB.ReadXmlSchema(@"D:\ADO.NET\DATA\ShopDbSchema.xml");

shopDB.ReadXml(@"D:\ADO.NET\DATA\ShopDBData.xml");

shopDB.AcceptChanges();

var orders = shopDB.Tables["Orders"];

var orderDetails = shopDB.Tables["OrderDetails"];

orders.Columns.Add("TotalSold", typeof(double), "SUM(Child(Orders\_OrderDetails).TotalPrice)"); // добавление расчитываемого столбца, используемого связь

foreach (DataRow order in orders.Rows)

{

Console.WriteLine("OrderId:{0},\nOrderDate:{1},\nTotalSold:{2}", order["OrderID"], order["OrderDate"], order["TotalSold"]);

Console.WriteLine();

}

}

\*\*\*

Rule

Перечисление **Rule** определяет действие, которое должно выполнено для обеспечения ограничения **ForeignKeyConstraint**

Элементы перечисления **Rule**

* **1)None –** при удалении родительской строки никаких действий для дочерних строк не применять.
* **2)Cascade –** при удалении родительской строки удалять все связанные дочерние строки.
* **3)SetNull –** при удалении родительской строки связанным дочерним строкам присвоить значение **DBNull.Value**
* **4)SetDefault –** при удалении родительской строки связанным дочерним строкам присвоить значение по умолчанию

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

{

DataSet shopDB = new DataSet();

shopDB.ReadXmlSchema(@"D:\ADO.NET\DATA\ShopDbSchema.xml");

shopDB.ReadXml(@"D:\ADO.NET\DATA\ShopDBData.xml");

shopDB.AcceptChanges();

DataTable customers = shopDB.Tables["Customers"];

DataTable orders = shopDB.Tables["Orders"];

DataTable orderDetails = shopDB.Tables["OrderDetails"];

var FK\_Customers\_Orders = orders.Constraints["Customers\_Orders"] as ForeignKeyConstraint;

// при удалении/изменении строки из таблицы Customers будут удаляться/изменяться все связанные строки из таблицы Orders

FK\_Customers\_Orders.DeleteRule = Rule.Cascade;

FK\_Customers\_Orders.UpdateRule = Rule.Cascade;

var FK\_Orders\_OrderDetails = orderDetails.Constraints["Orders\_OrderDetails"] as ForeignKeyConstraint;

FK\_Orders\_OrderDetails.DeleteRule = Rule.Cascade; //при удалении строки из таблицы Customers будет ошибка

label1.Text = customers.TableName;

dataGridView1.DataSource = customers; // связывание таблицы customers c элементом управления dataGridView1

label2.Text = orders.TableName;

dataGridView2.DataSource = orders; // связывание таблицы orders c элементом управления dataGridView2

label3.Text = orderDetails.TableName;

dataGridView3.DataSource = orderDetails;

}

\*\*\*

DataRow Version

DataSet shopDB = new DataSet();

shopDB.ReadXmlSchema(@"D:\ADO.NET\DATA\ShopDbSchema.xml");

shopDB.ReadXml(@"D:\ADO.NET\DATA\ShopDBData.xml");

shopDB.AcceptChanges();

DataTable orders = shopDB.Tables["Orders"];

DataTable customers = shopDB.Tables["Customers"];

// ForeignKeyConstraint FK\_Customers\_Orders = orders.Constraints["Customers\_Orders"] as ForeignKeyConstraint;

// FK\_Customers\_Orders.DeleteRule = Rule.SetNull;

customers.Rows[0].Delete();

var ordersChildRows = customers.Rows[0].GetChildRows("Customers\_Orders", DataRowVersion.Original);

// ordersChildRows = customers.Rows[0].GetChildRows("Customers\_Orders");

foreach (var ordersRow in ordersChildRows)

Console.WriteLine("OrderID: {0}, orderDate: {1:D}",ordersRow[0, DataRowVersion.Original], ordersRow[2, DataRowVersion.Original]);

Класс **DataView** представляет настраиваемое и допускающее привязку данных представление объекта **DataTable,** для сортировки, фильтрации, поиска, изменения и навигации.

Основная функция **DataView –** обеспечение привязки данных форм Windows и веб-форм.

Кроме того объект **DataView** можно настроить для предоставления подмножество данных из объекта **DataTable**, что позволяет использовать несколько элементов управления, отображающих данные из одного объекта **DataTable.**

\*\*\*

Find

DataSet ds = new DataSet();

string connecionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Customers";

SqlDataAdapter adapter = new SqlDataAdapter(commandString, connecionString);

adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;

adapter.Fill(ds);

DataTable customers = ds.Tables[0];

DataRow customersRow = customers.Rows.Find(1);

foreach (DataColumn customersColumn in customers.Columns)

Console.WriteLine(customersColumn.ColumnName+" "+customersRow[customersColumn]);

\*\*\*

Find композитный ключ

DataSet ds = new DataSet();

string connecionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM OrderDetails";

SqlDataAdapter adapter = new SqlDataAdapter(commandString, connecionString);

adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;

adapter.Fill(ds);

DataTable orderDetails = ds.Tables[0];

DataRow customersRow = orderDetails.Rows.Find(new object[] { 1, 2 });

foreach (DataColumn customersColumn in orderDetails.Columns)

Console.WriteLine(customersColumn.ColumnName + " " + customersRow[customersColumn]);

\*\*\*

Select

DataSet ds = new DataSet();

string connecionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Customers";

SqlDataAdapter adapter = new SqlDataAdapter(commandString, connecionString);

adapter.Fill(ds);

DataTable customers = ds.Tables[0];

DataRow[] customersRows = customers.Select("Address1 = 'Лужная 15'");

DataRow[] customersRows = customers.Select("Address1 LIKE 'Лужная%'");

DataRow[] customersRows = customers.Select("City = 'Харьков' OR City = 'Чернигов'");

DataRow[] customersRows = customers.Select("City = 'Киев'", "DateInSystem desc");

foreach (var customersRow in customersRows)

{

foreach (DataColumn customersColumn in customers.Columns)

Console.WriteLine(customersColumn.ColumnName + " " + customersRow[customersColumn]);

Console.WriteLine();

}

\*\*\*

DataView

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Customers";

DataTable customers = new DataTable();

customers.TableName = "Customers";

SqlDataAdapter adapter = new SqlDataAdapter(commandString, connectionString);

adapter.Fill(customers);

DataView customersView = new DataView(); // используя коструктор по умолчанию

customersView.Table = customers;

// DataView customersView = new DataView(customers); // используя конструктор с одним параметром

foreach (DataRowView viewRow in customersView)

{

Console.WriteLine("{0} {1} {2}",viewRow["LName"], viewRow["FName"], viewRow["MName"]);

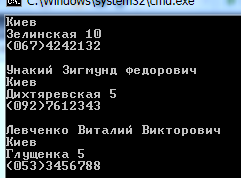
Console.WriteLine(viewRow["City"]);

Console.WriteLine(viewRow["Address1"]);

Console.WriteLine(viewRow["Phone"]);

Console.WriteLine();

}



DataView с сортировкой и фильтром

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Customers";

DataTable customers = new DataTable();

customers.TableName = "Customers";

SqlDataAdapter adapter = new SqlDataAdapter(commandString, connectionString);

adapter.Fill(customers);

// создание DataView c фильтром и сортировкой

DataView customersView = new DataView(customers, "CustomerNo > 2", "LName, FName", DataViewRowState.Unchanged);

foreach (DataRowView viewRow in customersView)

{

Console.WriteLine("CustomerNo: "+viewRow["CustomerNo"]);

Console.WriteLine("{0} {1} {2}", viewRow["LName"], viewRow["FName"], viewRow["MName"]);

Console.WriteLine(viewRow["City"]);

Console.WriteLine(viewRow["Address1"]);

Console.WriteLine(viewRow["Phone"]);

Console.WriteLine();

}

\*\*\*

RowState в ComboBox

private void button1\_Click(object sender, EventArgs e)

{

view.RowFilter = textBox1.Text;

view.Sort = textBox2.Text;

view.RowStateFilter = (DataViewRowState)Enum.Parse(typeof(DataViewRowState), comboBox1.Text, true);

}

\*\*\*

FindRows

public partial class Form1 : Form

{

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Customers";

DataTable customers = new DataTable();

SqlDataAdapter adapter;

DataView customersView;

public Form1()

{

InitializeComponent();

adapter = new SqlDataAdapter(commandString, connectionString);

adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;

adapter.Fill(customers);

customersView = new DataView(customers, "", "City", DataViewRowState.CurrentRows);

dataGridView1.DataSource = customers;

}

private void button1\_Click(object sender, EventArgs e)

{

DataRowView[] findingRows = customersView.FindRows(textBox1.Text);

dataGridView2.DataSource = findingRows.ToList();

}

}

Linq to DataSet

DataSet ShopDB = new DataSet();

ShopDB.ReadXmlSchema(@"D:\ADO.NET\DATA\ShopDbSchema.xml");

ShopDB.ReadXml(@"D:\ADO.NET\DATA\ShopDbData.xml");

DataTable customers = ShopDB.Tables["Customers"];

var query = from customer in customers.AsEnumerable() // по умолчанию объекты DataTable не реализуют интерфейс IEnumerable

select new { FName = customer["FName"], LName = customer["LName"] };

foreach (var customerInfo in query)

{

Console.WriteLine(customerInfo.LName +" "+ customerInfo.FName);

}

}

AsDataView

DataSet ShopDB = new DataSet();

ShopDB.ReadXmlSchema(@"D:\ADO.NET\DATA\ShopDbSchema.xml");

ShopDB.ReadXml(@"D:\ADO.NET\DATA\ShopDbData.xml");

var customers = ShopDB.Tables["Customers"];

//var orders = ShopDB.Tables["Orders"];

//var orderDetails = ShopDB.Tables["OrderDetails"];

//var products = ShopDB.Tables["Products"];

InitializeComponent();

var query = from cust in customers.AsEnumerable()

where cust.Field<int>("CustomerNo") == 1 || cust.Field<int>("CustomerNo") == 2

orderby cust["CustomerNo"] descending

select cust;

dataGridView2.DataSource = query.AsDataView();

dataGridView1.DataSource = customers;

}

\*\*\*

Join

DataSet ShopDB = new DataSet();

ShopDB.ReadXmlSchema(@"D:\ADO.NET\DATA\ShopDbSchema.xml");

ShopDB.ReadXml(@"D:\ADO.NET\DATA\ShopDbData.xml");

var customers = ShopDB.Tables["Customers"];

var orders = ShopDB.Tables["Orders"];

var inner = from customer in customers.AsEnumerable() // InnerJoin

join order in orders.AsEnumerable()

on customer["CustomerNo"] equals order["CustomerNo"]

select new { Fname = customer["FName"], LName = customer["LName"], OrderDate = order["OrderDate"] };

var outer = from customer in customers.AsEnumerable() // OuterJoin

join order in orders.AsEnumerable()

on customer["CustomerNo"] equals order["CustomerNo"] into joinGroup

select new { Fname = customer["FName"], LName = customer["LName"], OrderDate = joinGroup };

foreach (var item in inner)

{

Console.WriteLine("{2:D} {1} {0}", item.LName, item.Fname, item.OrderDate);

}

Console.WriteLine(new string('-', 20));

foreach (var item in outer)

{

Console.WriteLine("{1} {0}", item.LName, item.Fname);

foreach (var item2 in item.OrderDate)

{

Console.WriteLine("\t" + item2["OrderDate"]);

}

Console.WriteLine();

}

\*\*\*

Join

DataSet ShopDB = new DataSet();

ShopDB.ReadXmlSchema(@"D:\ADO.NET\DATA\ShopDbSchema.xml");

ShopDB.ReadXml(@"D:\ADO.NET\DATA\ShopDbData.xml");

var customers = ShopDB.Tables["Customers"];

var orders = ShopDB.Tables["Orders"];

var orderDetails = ShopDB.Tables["OrderDetails"];

var products = ShopDB.Tables["Products"];

var query = from customer in customers.AsEnumerable()

from order in orders.AsEnumerable()

from orderDetail in orderDetails.AsEnumerable()

from product in products.AsEnumerable()

where (int)customer["CustomerNo"] == (int)order["CustomerNo"] &&

(int)order["OrderId"] == (int)orderDetail["OrderId"] &&

(int)orderDetail["ProdID"] == (int)product["ProdID"]

select new

{

Customer = customer["LName"].ToString() + " " + customer["FName"].ToString(),

OrderDate = order["OrderDate"],

LineItem = orderDetail["LineItem"],

TotalPrice = orderDetail["TotalPrice"],

Description = product["Description"].ToString().Trim()

};

dataGridView1.DataSource = query.ToList();

\*\*\*

Объединение 4 таблиц

DataSet ShopDB = new DataSet();

ShopDB.ReadXmlSchema(@"D:\ADO.NET\DATA\ShopDbSchema.xml");

ShopDB.ReadXml(@"D:\ADO.NET\DATA\ShopDbData.xml");

var customers = ShopDB.Tables["Customers"];

var orders = ShopDB.Tables["Orders"];

var orderDetails = ShopDB.Tables["OrderDetails"];

var products = ShopDB.Tables["Products"];

var query = from customer in customers.AsEnumerable()

join order in orders.AsEnumerable()

on customer["CustomerNo"] equals order["CustomerNo"] into ordersGroup

from order in ordersGroup.DefaultIfEmpty() // DefaultIfEmpty вернет null, если не существует связанных данных

join orderDetail in orderDetails.AsEnumerable()

on (order == null) ? null : order["OrderId"] equals orderDetail["OrderId"] into orderdetailsGroup

select

new

{

FName = customer["FName"],

Sum = orderdetailsGroup.Sum(a => Convert.ToDecimal(a["totalPrice"]))

};

dataGridView1.DataSource = query.ToList();

\*\*\*

Типизированный объект **DataSet** представляет собой класс, производный от класса **DataSet**. Он наследует все методы, события и свойства класса**DataSet**. Кроме того, типизированный **DataSet** предоставляет строго типизированные методы, события и свойства.

Это означает, что к таблицам и столбцам можно обращаться путем указания имен вместо использования методов на основе коллекций. Помимо повышения понятности кода, типизированный объект **DataSet** позволяет использовать редактор кода Visual Studio .NET для автоматического завершения вводимых строк.

Как создать типизированный DataSet

* 1.Откройте Visual Studio и создайте новый проект.
* 2.В Solution Explorer выберите DataSet из контекстного меню. Назовите его ИмяБазыДанных.xsd.
* 3.Откройте Server Explorer Toolbox, и перейдите к БД ИмяБазыДанных SQL Server 2012.
* 4.Перетащите таблицу ИмяТаблици в окно DataSet Designer.
* 5.Для каждой добавленной таблицы Visual Studio создаст строго типизированную DataTable (с именем, основанным на имени исходной таблицы) и TableAdapter. В DataTable колонки уже определены.
* 6.TableAdapter – это объект, который мы будем использовать при заполнении таблицы. У нас есть используемый по умолчанию метод Fill(), который находит каждую строку из данной таблицы.

\*\*\*

Строго типизированный DataSet

Добавить елемент DataSet

ShopDB shopDB = new ShopDB(); // создание строготипизированного DataSet

DataRow customersRow = shopDB.Customers.NewRow(); // Создание новой строки. Тип DataRow

// заполнение полей строки данными

customersRow[0] = 1;

customersRow[1] = "Алексей";

customersRow[2] = "Петров";

customersRow[3] = "Николаевич";

customersRow[4] = "Лужная 7";

customersRow[5] = DBNull.Value;

customersRow[6] = "Киев";

customersRow[7] = "(096)4578596";

customersRow[8] = "2009/09/18";

// добавление строки в коллекцию строк таблицы

shopDB.Customers.Rows.Add(customersRow);

// вывод данных на экран

foreach (DataRow row in shopDB.Customers.Rows)

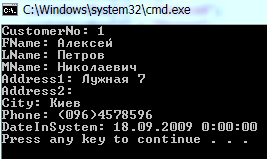
{

foreach (DataColumn column in shopDB.Customers.Columns)

Console.WriteLine("{0}: {1}",column.ColumnName,row[column]);

}

}



\*\*\*

ShopDB shopDB = new ShopDB();

// создание строки для таблицы Customers

ShopDB.CustomersRow customersRow = shopDB.Customers.NewCustomersRow();

// Для заполнения полей строки используются строготипизированные свойства

customersRow.FName = "Алексей";

customersRow.LName = "Петров";

customersRow.MName = "Николаевич";

customersRow.Address1 = "Лужная 7";

customersRow.Address2 = null;

customersRow.City = "Киев";

customersRow.Phone = "(096)4578596";

customersRow.DateInSystem = new DateTime(2009, 09, 18);

// добавление созданной строки в коллекцию строк таблицы

shopDB.Customers.Rows.Add(customersRow);

// еще один способ добавления строки в соллекцию строк таблицы

shopDB.Customers.AddCustomersRow("Николай", "Александров", "Анатольевич", "Московская 15", null, "Чернигов", "(063)0215478", new DateTime(2008, 05,15));

foreach (DataRow row in shopDB.Customers.Rows)

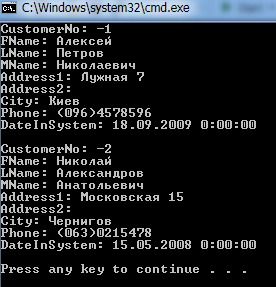
{

foreach (DataColumn column in shopDB.Customers.Columns)

Console.WriteLine("{0}: {1}", column.ColumnName, row[column]);

Console.WriteLine();

}



\*\*\*

Find

ShopDB shopDB = new ShopDB();

//ShopDB.CustomersRow customersRow = shopDB.Customers.NewCustomersRow();

shopDB.Customers.AddCustomersRow("Николай", "Александров", "Анатольевич", "Московская 15", null, "Чернигов", "(063)0215478", new DateTime(2008, 05, 15));

shopDB.Customers.AddCustomersRow("Константин", "Нимазов", "Алексеевич", "Октябрьская 5", null, "Киев", "(093)5487478", new DateTime(2009, 06, 20));

shopDB.Customers.AddCustomersRow("Андрей", "Макин", "Абрамович", "Сеновальная 7", null, "Витебск", "(098)7123478", new DateTime(2006, 12, 1));

// Использовани строго типизированного метода Find для нахождения строки по первичному ключу

var selectedCustomer = shopDB.Customers.FindByCustomerNo(-3);

selectedCustomer.BeginEdit();

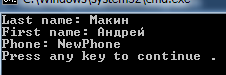
selectedCustomer.Phone = "NewPhone";

selectedCustomer.EndEdit();

Console.WriteLine("Last name: " + selectedCustomer.LName);

Console.WriteLine("First name: " + selectedCustomer.FName);

Console.WriteLine("Phone: " + selectedCustomer.Phone);



\*\*\*

Робота с null

ShopDB shopDB = new ShopDB();

ShopDB.CustomersRow customersRow = shopDB.Customers.NewCustomersRow();

shopDB.Customers.AddCustomersRow("Николай", "Александров", "Анатольевич", "Московская 15", null, "Чернигов", "(063)0215478", new DateTime(2008, 05, 15));

shopDB.Customers.AddCustomersRow("Константин", "Нимазов", "Алексеевич", "Октябрьская 5", null, "Киев", "(093)5487478", new DateTime(2009, 06, 20));

shopDB.Customers.AddCustomersRow("Андрей", "Макин", "Абрамович", "Сеновальная 7", null, "Витебск", "(098)7123478", new DateTime(2006, 12, 1));

var selectedCustomer = shopDB.Customers.FindByCustomerNo(-3);

selectedCustomer.BeginEdit();

selectedCustomer.SetMNameNull(); // присвоение полю строки значния DbNull.Value

selectedCustomer.EndEdit();

Console.WriteLine("Last name: " + selectedCustomer.LName);

Console.WriteLine("First name: " + selectedCustomer.FName);

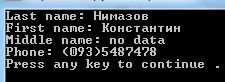
if (selectedCustomer.IsMNameNull()) // проверка на DbNull.Value

Console.WriteLine("Middle name: no data");

else

Console.WriteLine("Middle name: "+ selectedCustomer.MName);

Console.WriteLine("Phone: " + selectedCustomer.Phone);



\*\*\*

Merge

DataSet simpleShopDb = new DataSet();

simpleShopDb.ReadXmlSchema(@"D:\ADO.NET\DATA\ShopDbSchema.xml");

simpleShopDb.ReadXml(@"D:\ADO.NET\DATA\ShopDBData.xml");

ShopDB strongedShopDB = new ShopDB();

//Слияние обычно применяется для клиентских приложений с целью включения последних изменений из источника данных в существующий класс DataTable. Это позволяет клиентскому приложению иметь класс DataTable, обновленный последними данными из источника.

strongedShopDB.Customers.Merge(simpleShopDb.Tables["Customers"]);

foreach (DataRow row in strongedShopDB.Customers.Rows)

{

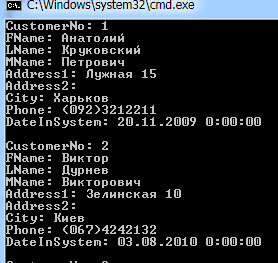
foreach (DataColumn column in strongedShopDB.Customers.Columns)

Console.WriteLine("{0}: {1}", column.ColumnName, row[column]);

Console.WriteLine();

}

}



\*\*\*

DataSet simpleShopDb = new DataSet();

simpleShopDb.ReadXmlSchema(@"D:\ADO.NET\DATA\ShopDbSchema.xml");

simpleShopDb.ReadXml(@"D:\ADO.NET\DATA\ShopDBData.xml");

ShopDB strongedShopDB = new ShopDB();

strongedShopDB.Customers.Merge(simpleShopDb.Tables["Customers"]);

strongedShopDB.Orders.Merge(simpleShopDb.Tables["Orders"]);

var selectedCustomer = strongedShopDB.Customers.FindByCustomerNo(1);

Console.WriteLine("{0} {1} {2}", selectedCustomer.LName, selectedCustomer.FName, selectedCustomer.MName+"\n");

foreach (var ordersRow in selectedCustomer.GetOrdersRows())

{

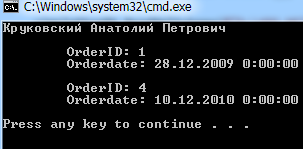
Console.WriteLine("\tOrderID: "+ordersRow.OrderID);

Console.WriteLine("\tOrderdate: "+ordersRow.OrderDate);

Console.WriteLine();

}

}



\*\*\*

ShopDB shopDB = new ShopDB();

new CustomersTableAdapter().

Fill(shopDB.Customers);

new OrdersTableAdapter().

Fill(shopDB.Orders);

new OrderDetailsTableAdapter().

Fill(shopDB.OrderDetails);

new ProductsTableAdapter().

Fill(shopDB.Products);

foreach (var customer in shopDB.Customers)

{

var orders = customer.GetOrdersRows();

if (orders.Length > 0)

{

Console.WriteLine("{0} {1} {2}", customer.LName, customer.FName, customer.MName);

Console.WriteLine();

foreach (var order in orders)

{

Console.WriteLine("\tOrderId:{0}, {1}", order.OrderID, order.OrderDate);

foreach (var orderDetail in order.GetOrderDetailsRows())

{

var product = orderDetail.ProductsRow;

Console.WriteLine("\t\tLineItem:{0} - {1}, {2:C}",

orderDetail.LineItem, product.Description.Trim(), orderDetail.TotalPrice);

}

Console.WriteLine();

}

}

}

\*\*\*

GetChanges

DataSet simpleShopDb = new DataSet();

simpleShopDb.ReadXmlSchema(@"D:\ADO.NET\DATA\ShopDbSchema.xml");

simpleShopDb.ReadXml(@"D:\ADO.NET\DATA\ShopDBData.xml");

ShopDB strongedShopDB = new ShopDB();

strongedShopDB.Customers.Merge(simpleShopDb.Tables["Customers"]);

strongedShopDB.AcceptChanges();

var selectedCustomer = strongedShopDB.Customers.FindByCustomerNo(1);

selectedCustomer.Phone = "New Value";

// Метод GetChanes() получает копию класса DataSet, содержащую все изменения, внесенные после его последней загрузки или после вызова метода AcceptChanges.

DataSet datasetWithChanges = strongedShopDB.GetChanges();

ShopDB inst = datasetWithChanges as ShopDB;

foreach (DataRow row in datasetWithChanges.Tables["Customers"].Rows)

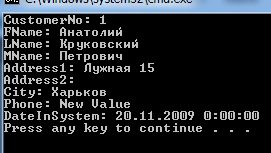
{

foreach (DataColumn column in datasetWithChanges.Tables["Customers"].Columns)

Console.WriteLine("{0}: {1}", column.ColumnName, row[column]);

}

}



\*\*\*

TableAdapter

ShopDB shopDB = new ShopDB();

// Для каждой таблицы типизированного DataSet создается адаптер, который позволяет заполнить эту таблицу данными

var customersTableAdapter = new ShopDBTableAdapters.CustomersTableAdapter();

// Заполнение таблиц данными производится с помощью вызова метода Fill

customersTableAdapter.Fill(shopDB.Customers);

foreach (DataRow customersRow in shopDB.Customers.Rows)

{

foreach (DataColumn customersColumn in shopDB.Customers.Columns)

Console.WriteLine("{0}: {1}", customersColumn.ColumnName, customersRow[customersColumn]);

Console.WriteLine();

}

\*\*\*

var customersTableAdapter = new ShopDBTableAdapters.CustomersTableAdapter();

ShopDB.CustomersDataTable customers = customersTableAdapter.GetData();

foreach (DataRow customersRow in customers.Rows)

{

foreach (DataColumn customersColumn in customers.Columns)

Console.WriteLine("{0}: {1}", customersColumn.ColumnName, customersRow[customersColumn]);

Console.WriteLine();

}

\*\*\*

Вставка в источник

var customers = new ShopDB.CustomersDataTable();

var customersTableAdapter = new ShopDBTableAdapters.CustomersTableAdapter { ClearBeforeFill = true };

customersTableAdapter.Insert("Test", "Test", "Test", "Test", "Test", "Test", "Test", new DateTime(9999 / 12 / 31));

customers = customersTableAdapter.GetData();

DataRow[] testRows = customers.Select("Phone = 'Test'");

foreach (var testRow in testRows)

{

foreach (DataColumn customersColumn in customers.Columns)

Console.WriteLine("{0}: {1}", customersColumn.ColumnName, testRow[customersColumn]);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Console.WriteLine(new string('-', 20));

Console.WriteLine("Total test row(s) deleted: " + DeleteTestRows());

customers = customersTableAdapter.GetData();

testRows = customers.Select("Phone = 'Test'");

foreach (var testRow in testRows)

{

foreach (DataColumn customersColumn in customers.Columns)

Console.WriteLine("{0}: {1}", customersColumn.ColumnName, testRow[customersColumn]);

}

}

private static int DeleteTestRows()

{

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "DELETE Customers WHERE Phone ='Test';";

SqlConnection connection = new SqlConnection(connectionString);

connection.Open();

SqlCommand deleCommand = new SqlCommand(commandString, connection);

int rowsAffected = deleCommand.ExecuteNonQuery();

connection.Close();

return rowsAffected;

}

}

\*\*\*

Ключевой функцией любого приложения базы данных является возможность обновления данных, хранимых в базе данных.

В ADO.NET обновление данных включает использование **DataAdapter** и **DataSet**, а также объектов **Command**. Обновление может включать использование транзакций.

С помощью метода **Update** объекта **DataAdapter** можно передавать отложенные изменения в источник данных.

Перегрузки метода

**DataAdapter.Update**

**:**

* **1)DataAdapter.Update(DataSet)** – обновляет источник на основе данных, предоставленных объектом **DataSet**.
* **2)DataAdapter.Update(DataTable)** – обновляет источник на основе данных, предоставленных объектом **DataTable**.
* **3)DataAdapter.Update(DataRow[])** – обновляет источник на основе данных, предоставленных массивом объектов **DataRow.**
* Обновление выполняется построчно. Для каждой вставленной, измененной и удаленной строки метод **Update** определяет тип изменений, выполненных в ней (Insert, Update или Delete).В зависимости от типа изменений шаблон команд **INSERT**, **UPDATE,** или **DELETE** выполняется для распространения измененной строки в источник данных.
* Проще говоря, когда приложение вызывает метод Update, объект **DataAdapter** проверяет свойство **RowState** и выполняет необходимые операторы **INSERT**, **UPDATE** или **DELETE** для передачи отложенных изменений в источник данных

\*\*\*

static class CustomersUpdates

{

private static void InsertCustomers(DataRow customersRow)

{

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "INSERT Customers " +

"VALUES (@FName, @LName, @MName, @Address1, @Address2, @City, @Phone, @DateInSystem) SELECT CustomerNo FROM Customers WHERE CustomerNo = @@IDENTITY";

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

SqlCommand cmd = new SqlCommand(commandString, connection);

cmd.Parameters.AddWithValue("FName", customersRow[1]);

cmd.Parameters.AddWithValue("LName", customersRow[2]);

cmd.Parameters.AddWithValue("MName", customersRow[3]);

cmd.Parameters.AddWithValue("Address1", customersRow[4]);

cmd.Parameters.AddWithValue("Address2", customersRow[5]);

cmd.Parameters.AddWithValue("City", customersRow[6]);

cmd.Parameters.AddWithValue("Phone", customersRow[7]);

cmd.Parameters.AddWithValue("DateInSystem", customersRow[8]);

try

{

customersRow.Table.Columns[0].ReadOnly = false;

customersRow.SetField<int>(0, (int)cmd.ExecuteScalar());

}

catch (Exception)

{

throw;

}

finally

{

customersRow.Table.Columns[0].ReadOnly = true;

}

customersRow.AcceptChanges();

}

}

public static void InsertCustomers(DataTable customers)

{

foreach (DataRow row in customers.Rows)

{

if (row.RowState == DataRowState.Added)

InsertCustomers(row);

}

}

}

public partial class AddCustomerDialog : Form

{

DataTable table;

public AddCustomerDialog(DataTable table)

{

this.table = table;

InitializeComponent();

}

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

{

//MessageBox.Show(table.Columns["LName"].AllowDBNull.ToString());

this.CenterToScreen();

}

private void button1\_Click(object sender, EventArgs e)

{

DataRow newRow = table.NewRow();

newRow["FName"] = textBox1.Text.Count() == 0 ? null : textBox1.Text;

newRow["LName"] = textBox2.Text.Count() == 0 ? null : textBox2.Text;

newRow["Address1"] = textBox3.Text.Count() == 0 ? null : textBox3.Text;

newRow["City"] = textBox4.Text.Count() == 0 ? null : textBox4.Text;

newRow["Phone"] = textBox5.Text.Count() == 0 ? null : textBox5.Text;

try

{

table.Rows.Add(newRow);

DialogResult = System.Windows.Forms.DialogResult.OK;

CustomersUpdates.InsertCustomers(table);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

finally

{

this.Close();

}

}

private void button2\_Click(object sender, EventArgs e)

{

DialogResult = System.Windows.Forms.DialogResult.Cancel;

this.Close();

}

}

public partial class Form1 : Form

{

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Customers";

DataTable customers = new DataTable("Customers");

SqlDataAdapter adapter;

public Form1()

{

InitializeComponent();

this.CenterToScreen();

}

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

{

adapter = new SqlDataAdapter(commandString, connectionString);

adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;

adapter.FillSchema(customers, SchemaType.Mapped);

customers.Columns["CustomerNo"].AutoIncrementSeed = -1;

customers.Columns["CustomerNo"].AutoIncrementStep = -1;

adapter.Fill(customers);

dataGridView1.DataSource = customers;

}

private void button1\_Click(object sender, EventArgs e)

{

System.Windows.Forms.DialogResult result = new AddCustomerDialog(customers).ShowDialog();

}

private void button2\_Click(object sender, EventArgs e)

{

}

\*\*\*

DeleteRows

static class CustomersUpdates

{

private static void DeleteCustomers(DataRow customersRow)

{

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

StringBuilder builder = new StringBuilder();

string commandString = "DELETE Customers " +

"WHERE CustomerNo = @CustomerNo";

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

SqlCommand cmd = new SqlCommand(commandString, connection);

cmd.Parameters.AddWithValue("CustomerNo", customersRow[0, DataRowVersion.Original]);

cmd.ExecuteNonQuery();

}

}

public static void DeleteCustomers(DataTable customers)

{

foreach (DataRow row in customers.Rows)

{

if (row.RowState == DataRowState.Deleted)

DeleteCustomers(row);

}

customers.AcceptChanges();

}

}

public partial class Form1 : Form

{

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Customers";

DataTable customers = new DataTable("Customers");

public Form1()

{

InitializeComponent();

}

private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

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

{

SqlDataAdapter adapter = new SqlDataAdapter(commandString, connectionString);

adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;

adapter.Fill(customers);

dataGridView1.ReadOnly = true;

dataGridView1.DataSource = customers;

}

private void button1\_Click(object sender, EventArgs e)

{

DialogResult res = MessageBox.Show("Вы уверены что хотите удалить этого клиента?", "DeleteDialog", MessageBoxButtons.OKCancel);

if (res == System.Windows.Forms.DialogResult.OK)

{

(dataGridView1.CurrentRow.DataBoundItem as DataRowView).Row.Delete();

CustomersUpdates.DeleteCustomers(customers);

}

}

\*\*\*

AdapterApdate

public partial class MainForm : Form

{

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Customers";

DataTable customers = new DataTable("Customers");

SqlDataAdapter adapter;

public MainForm()

{

InitializeComponent();

this.CenterToScreen();

}

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

{

adapter = new SqlDataAdapter(commandString, connectionString);

adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;

ConfigureCustomersAdapter(adapter);

adapter.FillSchema(customers, SchemaType.Mapped);

customers.Columns[0].AutoIncrementSeed = -1;

customers.Columns[0].AutoIncrementStep = -1;

adapter.Fill(customers);

dataGridView1.DataSource = customers;

adapter.RowUpdated += adapter\_RowUpdated;

}

void adapter\_RowUpdated(object sender, SqlRowUpdatedEventArgs e)

{

if (e.StatementType == StatementType.Insert)

{

var insertedRow = e.Row;

try

{

insertedRow.Table.Columns[0].ReadOnly = false;

insertedRow[0] = e.Command.Parameters["NewCustomerNo"].Value;

}

catch (Exception)

{

throw;

}

finally

{

insertedRow.Table.Columns[0].ReadOnly = true;

}

}

}

private static void ConfigureCustomersAdapter(SqlDataAdapter customersAdapter)

{

#region Configure UpdateCommand

string commandString = "UPDATE Customers " +

"SET FName = @FName," +

"LName = @LName," +

"MName= @Mname," +

"Address1 = @Address1," +

"Address2 = @Address2," +

"City = @City," +

"Phone = @Phone," +

"DateInSystem = @DateInSystem " +

"WHERE CustomerNo = @CustomerNo";

customersAdapter.UpdateCommand = new SqlCommand(commandString,

customersAdapter.SelectCommand.Connection);

var updateParameters = customersAdapter.UpdateCommand.Parameters;

updateParameters.Add("CustomerNo", SqlDbType.Int, 0, "CustomerNo");

updateParameters.Add("FName", SqlDbType.NVarChar, 20, "FName");

updateParameters.Add("LName", SqlDbType.NVarChar, 20, "Lname");

updateParameters.Add("MName", SqlDbType.NVarChar, 20, "MName");

updateParameters.Add("Address1", SqlDbType.NVarChar, 20, "Address1");

updateParameters.Add("Address2", SqlDbType.NVarChar, 20, "Address2");

updateParameters.Add("City", SqlDbType.NVarChar, 20, "City");

updateParameters.Add("Phone", SqlDbType.NVarChar, 20, "Phone");

updateParameters.Add("DateInSystem", SqlDbType.Date, 0, "DateInSystem");

#endregion

#region Configure DeleteCommand

customersAdapter.DeleteCommand = new SqlCommand("DELETE Customers WHERE CustomerNo = @CustomerNo",

customersAdapter.SelectCommand.Connection);

var deleteParameters = customersAdapter.DeleteCommand.Parameters;

deleteParameters.Add("@CustomerNo", SqlDbType.Int, 0, "CustomerNo");

#endregion

#region Configure InsertCommand

customersAdapter.InsertCommand =

new SqlCommand("INSERT Customers " +

"VALUES (@FName, @LName, @MName, @Address1, @Address2, @City, @Phone, @DateInSystem);"+

"DECLARE @NewCustomer int; "+

"SET @NewCustomerNo = @@IDENTITY;",

customersAdapter.SelectCommand.Connection);

var insertParameters = customersAdapter.InsertCommand.Parameters;

insertParameters.Add("FName", SqlDbType.NVarChar, 20, "FName");

insertParameters.Add("LName", SqlDbType.NVarChar, 20, "Lname");

insertParameters.Add("MName", SqlDbType.NVarChar, 20, "MName");

insertParameters.Add("Address1", SqlDbType.NVarChar, 20, "Address1");

insertParameters.Add("Address2", SqlDbType.NVarChar, 20, "Address2");

insertParameters.Add("City", SqlDbType.NVarChar, 20, "City");

insertParameters.Add("Phone", SqlDbType.NVarChar, 20, "Phone");

insertParameters.Add("DateInSystem", // имя параметра

SqlDbType.Date, // тип данных в источнике

0, // длина столбца

"DateInSystem"); // имя столбца в источнике данных

var outputParameter = insertParameters.Add("NewCustomerNo", SqlDbType.Int);

outputParameter.Direction = ParameterDirection.Output;

#endregion

}

private void button1\_Click(object sender, EventArgs e)

{

try

{

adapter.Update(customers);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

//customers.Clear();

//adapter.Fill(customers);

}

}

\*\*\*

CommandBuilderWork

static void Main(string[] args)

{

string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=ShopDB;Integrated Security=True";

string commandString = "SELECT \* FROM Customers";

DataTable customers = new DataTable("Customers");

SqlDataAdapter adapter = new SqlDataAdapter(commandString, connectionString);

SqlCommandBuilder commandBuilder = new SqlCommandBuilder(adapter);

adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;

adapter.Fill(customers);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Adding Data\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

customers.LoadDataRow(new object[] { -1, "TEST", "TEST", "TEST", "TEST", "TEST", "TEST", "TEST", "9999/12/31" }, false);

adapter.Update(customers);

customers.Clear();

adapter.Fill(customers);

Console.WriteLine("Customers after adding test rows");

foreach (DataRow row in customers.Rows)

{

foreach (DataColumn column in customers.Columns)

Console.WriteLine("{0}: {1}", column.ColumnName, row[column]);

Console.WriteLine();

}

Console.ReadKey();

Console.Clear();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Changing Data\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DataRow[] changedRows = customers.Select("Phone = 'TEST'");

foreach (var singleRow in changedRows)

{

singleRow[1] = "ChangedValue";

singleRow[2] = "ChangedValue";

singleRow[3] = "ChangedValue";

}

adapter.Update(customers);

customers.Clear();

adapter.Fill(customers);

Console.WriteLine("Customers after changing test rows");

foreach (DataRow row in customers.Rows)

{

foreach (DataColumn column in customers.Columns)

Console.WriteLine("{0}: {1}", column.ColumnName, row[column]);

Console.WriteLine();

}

Console.ReadKey();

Console.Clear();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Deleting Data\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DataRow[] rowsToDelete = customers.Select("Phone = 'TEST'");

foreach (var singleRow in rowsToDelete)

{

singleRow.Delete();

}

adapter.Update(customers);

customers.Clear();

adapter.Fill(customers);

Console.WriteLine("Customers after deleting test rows");

foreach (DataRow row in customers.Rows)

{

foreach (DataColumn column in customers.Columns)

Console.WriteLine("{0}: {1}", column.ColumnName, row[column]);

Console.WriteLine();

}

Console.ReadKey();

Console.Clear();

}