**Лабораторная работа № 3. Работа с объектом DataSet.**

**Цель работы:**

– получить практические навыки создания и использования таблиц в объекте DataSet.

**Теоретические сведения**

1. **Объект DataAdapter**

В отношении работы с объектом DataSet технология ADO.NET предлагает совершенно другую философию. При подключении к базе данных, прочитанная из нее информация записывается в DataSet. При изменении информации в DataSet, соответствующая информация в таблице базы данных не изменяется. При необходимости можно повторно подключиться к исходному источнику данных и записать в него все изменения, внесенные в DataSet, за одну операцию.

Чтобы извлечь записи из базы и использовать их для наполнения таблицы в DataSet, нужно использовать другой объект ADO.NET — **DataAdapter**. Объект DataAdapter является специфичным для поставщика, поэтому для каждого поставщика предусмотрены разные классы DataAdapter (SqlDataAdapter, OracleDataAdapter и т.д.).

DataAdapter служит мостом между **одним** DataTable в DataSet и источником данных. Он включает все доступные команды для выполнения запросов и обновления источника данных. Чтобы позволить DataAdapter редактировать, удалять и добавлять строки, потребуется указать объекты Command для свойств UpdateCommand, DeleteCommand и InsertCommand объекта DataAdapter. Чтобы применить DataAdapter для наполнения DataSet, необходимо установить SelectCommand.

Класс DataAdapter предоставляет три ключевых метода, которые перечислены в таблице ниже:

|  |  |
| --- | --- |
| **Методы DataAdapter** | |
| **Метод** | **Описание** |
| *Fill()* | Добавляет DataTable к DataSet за счет выполнения запроса в SelectCommand. Если запрос возвращает множественные результирующие наборы, этот метод добавит множество объектов DataTable за раз. Этот метод можно также использовать для добавления данных к существующему объекту DataTable. |
| *FillSchema()* | Добавляет DataTable к DataSet за счет выполнения запроса в SelectCommand и извлечения только информации о схеме. Этот метод не добавляет никаких данных к DataTable. Вместо этого он просто предварительно конфигурирует DataTable с помощью детальной информации об именах столбцов, типах данных, первичных ключах и ограничениях уникальности. |
| *Update()* | Проверяет все изменения отдельного объекта DataTable и применяет пакет этих изменений к источнику данных за счет выполнения соответствующих операций InsertComrnand, UpdateCommand и DeleteCommand |

Для получения данных через объект SqlDataAdapter необходимо организовать подключение к БД и выполнить команду SELECT. Есть несколько способов создания SqlDataAdapter:

1. Использовать конструктор без параметров, а команду SELECT и подключение установить позже

***SqlDataAdapter adapter = new SqlDataAdapter();***

1. Передать в конструктор объект SqlCommand

***SqlDataAdapter adapter = new SqlDataAdapter(command);***

1. Можно в конструкторе установить sql-выражение SELECT и объект SqlConnection

***SqlDataAdapter adapter = new SqlDataAdapter(sql, connection);***

1. В конструкторе установить sql-выражение SELECT и строку подключения

***SqlDataAdapter adapter = new SqlDataAdapter(sql, connectionString);***

Для работы с DataSet обычно используются элементы управления, которые могут заполняться из внешнего источника данных, например, DataGridView в Windows Forms.

Создание проекта с использованием DataAdapter.

1. Создать проект Windows Form.
2. На форму в проекте добавить элемент DataGridView.
3. В код формы поместить следующий текст:

***using System;***

***using System.Collections.Generic;***

***using System.ComponentModel;***

***using System.Data;***

***using System.Drawing;***

***using System.Linq;***

***using System.Text;***

***using System.Threading.Tasks;***

***using System.Windows.Forms;***

***using System.Data.SqlClient;***

***namespace shop***

***{***

***public class Form1 : Form***

***{***

***public Form1()***

***{***

***InitializeComponent();***

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

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

***string sqlStr = "SELECT \* FROM Employees";***

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

***{***

***// Создание объекта DataAdapter***

***SqlDataAdapter adapter = new SqlDataAdapter(sqlStr, con);***

***// Создание объекта Dataset***

***DataSet ds = new DataSet();***

***// Заполняем Dataset***

***adapter.Fill(ds);***

***// Отображаем данные***

***dataGridView1.DataSource = ds.Tables[0];***

***}***

***}***

***}***

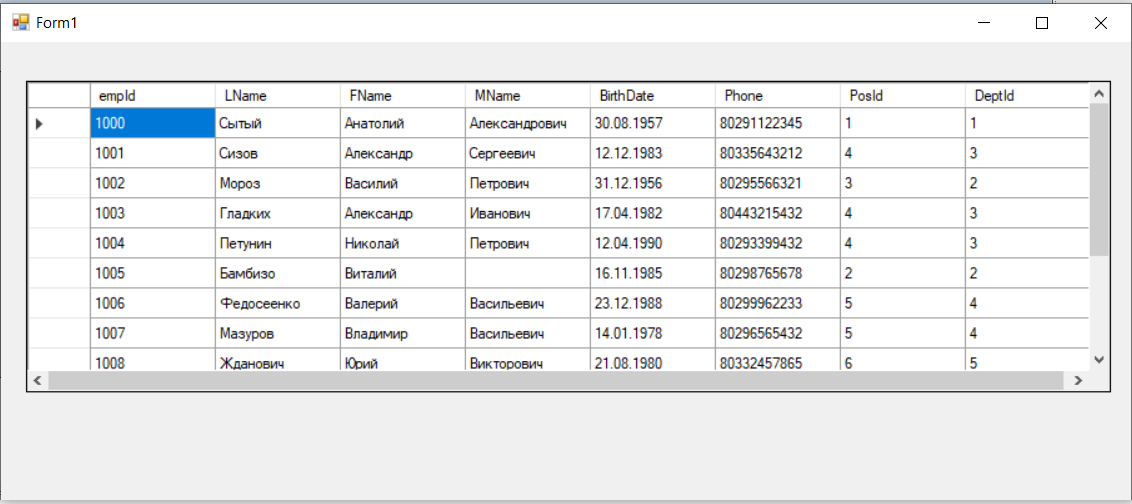
***}***

В элемент DataGridView загружаются все данные из таблицы persons базы данных myshop. Для загрузки данных создается объект SqlDataAdapter, который принимает объект подключения и sql-выражение SELECT. Затем создается объект DataSet и с помощью метода adapter.Fill() в него загружаются данные. Дальше происходит установка источника данных для DataGridView:

***dataGridView1.DataSource = ds.Tables[0];***

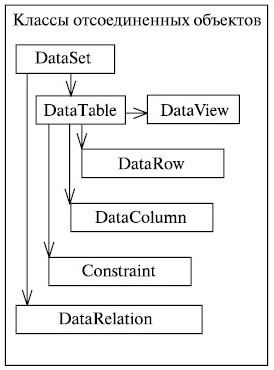
В качестве источника устанавливается таблица в DataSet с индексом, равным 0 (***ds.Tables[0)***. Каждая таблица представляет объект DataTable, и в DataSet может быть определено несколько таких таблиц. Но в данном случае в DataSet есть только одна таблица, которую можно получить из коллекции Tables по индексу. Открытие подключения командой con.Open() не требуется. DataAdapter сам открывает подключение и после загрузки инофрмации закрывает его.

Результат выполнения кода показан ниже.



1. **Объект DataSet**

Объект DataSet предназначен для хранения данных, считанных с помощью DataAdapter или созданных пользователем. Фактически DataSet –это представление данных в памяти, представляющее собой единообразную реляционную модель независимо от источника данных. Он может использоваться с несколькими различными источниками данных, включая XML-данные. DataSet представляет собой полный набор данных, включая связанные таблицы, ограничения и связи между таблицами. DataSet содержит в себе два важных объекта: коллекцию из нуля или более таблиц (доступных через свойство Tables) и коллекцию из нуля или более отношений, которые можно применять для связывания таблиц между собой (представленных свойством Relationships).



Из соображений производительности обычно DataSet используется для работы с небольшим подмножеством от общего объема информации источника данных. К тому же, таблицы в DataSet не обязаны отображаться непосредственно на таблицы источника данных. Одна таблица может содержать результаты запроса по одной таблице или результаты запроса, который объединяет данные из нескольких таблиц.

Каждая запись коллекции DataSet.Tables — это DataTable. Объект DataTable содержит собственные коллекции — коллекцию Columns объектов DataColumn (описывающие имя и тип данных каждого поля) и коллекцию Rows объектов DataRow(содержащих действительные данные каждой записи).

Каждая запись в DataTable представлена объектом DataRow. Каждый объект DataRow представляет одиночную запись в таблице, которая была извлечена из источника данных. DataRow является контейнером для действительных значений полей. Получить доступ к ним можно по имени поля, например myRow["FieldName"]. DataSet является автономным объектом не связанным с источником объектом. Все изменения проводятся локально в объекте DataSet, расположенном в памяти. DataSet никогда не полагается на какое-либо соединение с источником данных.

**Объект DataTableCollection**

Объект **DataSet**  содержит коллекцию из нуля или более таблиц, представленных объектами **DataTable.** DataTableCollection содержит все объекты **DataTable** в DataSet.

Объект **DataTable** определен в пространстве имен System.Data и представляет одну таблицу данных. Он содержит коллекцию столбцов, представленных объектом DataColumnCollection, и ограничений, представленных объектом ConstraintCollection, которые вместе определяют схему таблицы. Объект **DataTable** также содержит коллекцию строк, представленных DataRowCollection, которая содержит данные в таблице. Вместе со своим текущим состоянием объект DataRow сохраняет обе свои версии (текущую и исходную), чтобы определить изменения значений, хранимых в строке.

**Объект DataView**

DataView позволяет создавать различные представления данных, которые хранятся в DataTable. Эта возможность часто используется в приложениях связывания данных. С помощью объекта DataView можно представлять данные в таблице с различными порядками сортировки, а также фильтровать данные по состоянию строки или на основе критерия фильтра.

**Объект DataRelationCollection**

**DataSet** содержит связи в своем объекте DataRelationCollection. Связь, представленная объектом DataRelation, связывает строки в одном объекте **DataTable** со строками в другом объекте **DataTable**. Связь аналогична пути соединения, который может существовать между столбцами первичного и внешнего ключей реляционной базы данных. Объект **DataRelation** определяет совпадающие столбцы в двух таблицах DataSet.

Связи позволяют переходить от одной таблицы к другой в DataSet. Важными элементами

**DataRelation** являются имя связи, имена связанных таблиц, а также связанные столбцы в каждой таблице. Связи могут быть построены с более чем одним столбцом для каждой таблицы путем указания массива объектов DataColumn в качестве ключевых столбцов. При добавлении связи в DataRelationCollection можно дополнительно добавить **UniqueConstraint** и **ForeignKeyConstraint**  чтобы применить ограничения целостности при внесении изменений в связанные значения столбцов.

**3. Заполнение объекта DataSet из базы данных**

Обычно объект DataSet заполняется данными из базы данных. Для иллюстрации рассмотрим проект WindowsForm с именем MyShop3, содержащий форму и размещенные на ней следующие элементы управления

1. Таблица (dataGridView) с именем dataGridView1.
2. Кнопку (Button) с именем btRead и надписью «Читать».
3. Кнопку (Button) с именем btClear и надписью «Очистить».

Код формы может иметь следующий вид

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Data.SqlClient;

namespace MyShop3

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

}

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

{

// Организация подключения

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

SqlConnection con = new SqlConnection(connectionString);

string command = "SELECT Lname, FName, MName, Birthdate FROM Employees";

//Создание объекта DataAdapter

SqlDataAdapter adapter = new SqlDataAdapter(command, con);

// Создание и заполнение DataSet

DataSet dsEmployees = new DataSet();

adapter.Fill(dsEmployees, "Employees");

// Вывод из DataSet строк таблицы Employees в элемент dataGridView

dataGridView1.DataSource = dsEmployees.Tables[0];

}

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

{

dataGridView1.DataSource = null;

}

}

}

Сначала в коде создается соединение и определяется текст запроса SQL. На следующем шаге создается новый экземпляр класса SqlDataAdapter. После этого создается новый пустой объект DataSet и с помощью метода DataAdapter.Fill() выполняется запрос с помещением результата в новый объект DataTable с именем Persons этого DataSet.

В коде не открывается соединение за счет вызова con.Open(). Вместо этого DataAdapter открывает и закрывает связанное с ним соединение автоматически, когда вызывается метод Fill(). В качестве альтернативы соединение можно также открывать и закрывать «вручную». Если соединение открыто на момент вызова Fill(), то DataAdapter использует это соединение и не станет закрывать его автоматически. Такой подход удобен, если нужно быстро выполнить множество последовательных операций с источником данных и избежать дополнительных накладных расходов, связанных с повторным открытием и закрытием соединения.

Последний шаг — отображение содержимого DataSet. Проще всего всего это можно сделать с помощью объекта dataGridView, указав таблицу Persons в качестве источника данных этого объекта.

Форма c результатом чтения данных из базы данных показана на рис.3.1

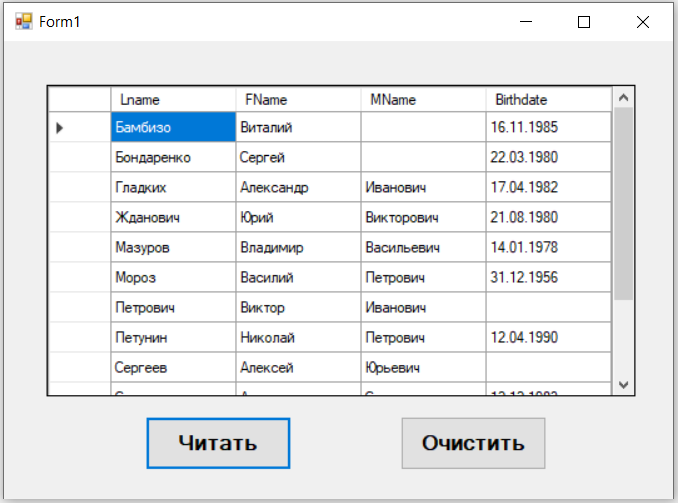


Рис.3.1. Форма с результатами чтения данных.

#### 

1. **Работа с объектом DataSet**

# 4.1 Создание набора данных

Экземпляр объекта [DataSet](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.dataset) создается путем вызова конструктора [DataSet](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.dataset), в котором можно задать имя объекта. Если имя для объекта [DataSet](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.dataset) не задано, то ему присваивается имя «NewDataSet».

Следующий пример кода демонстрирует создание объекта [DataSet](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.dataset).

***DataSet dsShop = new DataSet("Shop");***

## 

**4.2 Добавление новой таблицы данных в DataSet**

Объект **DataTable** представляет одну таблицу реляционных данных в памяти, которая может быть заполнена из источника данных, например с помощью **DataAdapter**. Доступ к коллекции таблиц в **наборе данных** осуществляется через свойство **Tables** объекта **DataSet** .

После создания объекта [DataTable](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datatable) в объекте [DataSet](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.dataset) можно выполнять такие же действия, как и при использовании таблицы в базе данных. Можно добавлять, просматривать, изменять и удалять данные в таблице. Можно отслеживать ошибки и события; кроме того, можно запрашивать находящиеся в таблице данные.

Схема или структура таблицы представляется столбцами и ограничениями. Столбцы таблицы могут сопоставляться со столбцами источника данных, содержать вычисляемые значения, автоматически увеличивать значения или содержать значения первичного ключа.

Помимо схемы, в **DataTable** также должны содержаться строки для хранения и упорядочения данных. Класс [DataRow](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datarow) представляет фактические данные, содержащиеся в таблице. Для получения, вычисления и обработки данных в таблице используется объект **DataRow** и его свойства и методы. При доступе и изменении данных в строке объект **DataRow** сохраняет как текущее, так и исходное состояние.

С помощью одного или нескольких связанных столбцов между таблицами можно создавать связи типа «родители-потомки». Связь между объектами **DataTable** создается с помощью [DataRelation](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datarelation). Затем объекты **DataRelation** можно использовать для возврата связанных дочерних или родительских строк определенной строки.

Объект **DataTable** можно создать с помощью соответствующего конструктора **DataTable**. Его можно добавить в **набор данных** с помощью метода **Add**, чтобы добавить его в коллекцию **Tables** объекта **DataTable**.

Объекты **DataTable** также можно создавать в **наборе данных** с помощью методов **Fill** или **FillSchema** объекта **DataAdapter.** При первом создании таблицы DataTable не имеет схемы (то есть структуры). Чтобы определить схему таблицы, необходимо создать и добавить [DataColumn](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumn) объекты в коллекцию **Columns** таблицы. Кроме того, можно определить первичный ключевой столбец для таблицы, а также создать и добавить объекты **ограничений** в коллекцию **ограничений** таблицы. Определив схему для **DataTable**, можно добавить строки данных в таблицу, добавив объекты **DataRow** в коллекцию **Rows** таблицы.

При создании DataTable значение свойства [TableName](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datatable.tablename) указывать не обязательно; свойство можно указать в другое время, либо оставить его пустым. Однако при добавлении таблицы без значения **TableName** в DataSetэтой таблице будет присвоено имя по умолчанию TableN, где N – текущий номер таблицы, начинающийся с 0.

В следующем примере создается экземпляр объекта **DataTable** и ему присваивается имя Positions.

***DataTable posTable = new DataTable("Positions");***

В следующем примере создается экземпляр **DataTable** путем его добавления в коллекцию **Tables** **набора данных**.

***DataSet ds = new DataSet();***

***DataTable posTable = ds.Tables.Add("Positions");***

# Определение схемы DataTable

Схема, или структура, таблицы представляется столбцами и ограничениями. Схема [DataTable](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datatable) определяется с использованием объектов [DataColumn](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumn), а также объектов [ForeignKeyConstraint](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.foreignkeyconstraint) и [UniqueConstraint](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.uniqueconstraint). Столбцы таблицы могут сопоставляться со столбцами источника данных, содержать вычисляемые значения, автоматически увеличивать значения или содержать значения первичного ключа.

В ссылках по имени на столбцы, связи и ограничения таблицы учитывается регистр. Поэтому в таблице могут существовать столбцы, связи и ограничения, имеющие одинаковое имя, но отличающиеся регистром. Правило учета регистра не применяется, если существует только один столбец, одна связь или ограничение с конкретным именем. Это означает, что если в таблице нет другого объекта столбца, связи или ограничения, имя которого совпадает с именем этого конкретного объекта столбца, связи или ограничения, то на объект можно ссылаться по имени, используя любой регистр, и исключение в этом случае не возникает.

# Добавление столбцов в DataTable

Объект [DataTable](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datatable) содержит коллекцию объектов [DataColumn](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumn), на которые ссылается свойство **Columns** таблицы. Эта коллекция столбцов наряду с ограничениями определяет схему, или структуру, таблицы.

Объекты **DataColumn** создаются в таблице с помощью конструктора **DataColumn** или путем вызова метода **Add** свойства **Columns** таблицы, то есть [DataColumnCollection](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumncollection). Метод **Add** принимает необязательные аргументы **ColumnName**, **DataType** и **Expression** и создает новый **столбец** данных в качестве члена коллекции. Он также принимает существующий объект **DataColumn** и добавляет его в коллекцию и возвращает ссылку на добавленный **DataColumn-столбец** по запросу. В следующем примере в **таблицу DataTable**добавляются четыре столбца.

***DataTable empTable = new DataTable("Employees");***

***DataColumn idEmpl = empTable.Columns.Add("EmployeeID", typeof(Int32));***

***idCol.AllowDBNull =* false*;***

***idCol.Unique =* true*;***

***empTable.Columns.Add("LName", typeof(String));***

***empTable.Columns.Add("FName", typeof(String));***

***empTable.Columns.Add("MName", typeof(Double));***

Обратите внимание, что в этом примере свойству для столбца idPers присвоено значение, которое не разрешает значения **DBNull** и ограничивает значения уникальными. Однако если определить столбец idPers в качестве первичного ключевого столбца таблицы, то свойству **AllowDbNull** автоматически будет присвоено значение **false** , а свойству **UNIQUE** будет автоматически присвоено значение **true**.

Если имя столбца не указано, то столбцу присваивается имя по умолчанию ColumnN, где N – номер столбца в коллекции.

# Создание вычисляемых столбцов.

Можно определить выражение для столбца, позволяя тем самым хранить в нем значение, вычисляемое из значений других столбцов той же строки или из значений столбцов нескольких строк таблицы. Для определения вычисляемого выражения используется свойство [Expression](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumn.expression) целевого столбца, свойство [ColumnName](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumn.columnname) - для ссылки на другие столбцы в выражении. Тип данных [DataType](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumn.datatype) столбца выражения должен соответствовать типу данных значения, возвращаемого выражением.

Можно задать свойство **Expression** для существующего объекта **DataColumn** или включить свойство в качестве третьего аргумента, [DataColumn](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumn) передаваемого конструктору, как показано в следующем примере.

***workTable.Columns.Add("Total", typeof(Double));***

***workTable.Columns.Add("SalesTax", typeof(Double), "Total \* 0.086");***

Выражение может содержать ссылки на другие столбцы выражений, но циклическая ссылка, при которой два выражения ссылаются друг на друга, вызовет исключение.

# [Создание столбцов AutoIncrement](https://docs.microsoft.com/ru-ru/dotnet/framework/data/adonet/dataset-datatable-dataview/creating-autoincrement-columns)

Чтобы обеспечить наличие в столбце уникальных значений, можно указать, что значения в столбце должны увеличиваться автоматически при добавлении новых строк к таблице. Чтобы создать автоматическое приращение [DataColumn](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumn), нужно присвоить свойству [AutoIncrement](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumn.autoincrement) столбца значение **true**. Значения такого столбца начинаются со значения, определенного в свойстве [AutoIncrementSeed](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumn.autoincrementseed), и при добавлении каждой строки значение столбца **AutoIncrement** увеличивается на значение, определенное в свойстве [AutoIncrementStep](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumn.autoincrementstep) столбца.

В следующем примере показано, как создать столбец, значения в котором начинаются с 200 и увеличиваются с шагом 3.

***DataColumn idEmp = empTable.Columns.Add( "EmployeeID", typeof(Int32));***

***idEmp.AutoIncrement =* true*;***

***idEmp.AutoIncrementSeed = 200;***

***idEmp.AutoIncrementStep = 3;***

## 

# Определение первичных ключей

База данных обычно содержит столбец или группу столбцов, уникально определяющих каждую строку в таблице. Такие столбцы или группы столбцов называются первичными ключами.

При определении [DataColumn](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumn) одного объекта в качестве [PrimaryKey](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datatable.primarykey) для [DataTable](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datatable) свойству [AllowDBNull](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumn.allowdbnull) столбца автоматически присваивается **значение false** , а свойству [Unique](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumn.unique) — **значение true**.

Свойство PrimaryKey принимает в качестве значения массив из одного или нескольких объектов **DataColumn** , как показано в следующих примерах. В первом примере в качестве первичного ключа определяется один столбец.

***empTable.PrimaryKey = new DataColumn[] {empTable.Columns["EmployeeID"]};***

Следующий пример определяет два столбца в качестве первичного ключа.

***personTable.PrimaryKey = new DataColumn[] {personTable.Columns["LName"],***

***personTable.Columns["FName"]};***

# 

# [Ограничения DataTable](https://docs.microsoft.com/ru-ru/dotnet/framework/data/adonet/dataset-datatable-dataview/datatable-constraints)

Ограничения позволяют принудительно поддерживать целостность данных [DataTable](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datatable). Ограничение представляет собой автоматическое правило, применяемое к столбцу или связанным столбцам и определяющее порядок действий при каком-либо изменении содержимого строки. Ограничения применяются, если свойство System.Data.DataSet.EnforceConstraints объекта [DataSet](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.dataset) имеет **значение true**. В ADO.NET имеется два типа ограничений: [ForeignKeyConstraint](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.foreignkeyconstraint) и [UniqueConstraint](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.uniqueconstraint). По умолчанию оба ограничения создаются автоматически при создании связи между двумя или более таблицами путем добавления [DataRelation](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datarelation) в DataSet.

## Ограничение ForeignKeyConstraint

**ForeignKeyConstraint** применяет правила, связанные с распространением обновлений и удалений в связанных таблицах. Например, если значение в строке одной таблицы обновлено или удалено и это же значение также используется в одной или нескольких связанных таблицах, **ForeignKeyConstraint** определяет, что происходит в связанных таблицах.

Свойства [DeleteRule](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.foreignkeyconstraint.deleterule) и [UpdateRule](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.foreignkeyconstraint.updaterule) объекта **ForeignKeyConstraint** определяют действие, которое должно быть предпринято, когда пользователь пытается удалить или обновить строку в связанной таблице. В следующей таблице описаны различные параметры, доступные для свойств **DeleteRule** и **UpdateRule** объекта **ForeignKeyConstraint**.

| **Установка правил** | **Описание** |
| --- | --- |
| **Cascade** | Удалить или обновить связанные строки. |
| **SetNull** | Присвоить столбцам в связанных строках значение **Null**. |
| **SetDefault** | Присвоить столбцам в связанных строках значение по умолчанию. |
| **None** | Не выполнять никаких действий в связанных строках. Это значение по умолчанию. |

Например, **если свойство объекта ForeignKeyConstraint** имеет значение **None**, то родительская строка не может быть удалена, если она содержит дочерние строки.

Примеры. Создание таблицы продаж (Sales), в которой в каждой продаже записывается идентификатор сотрудника, произвевшего продажу, то есть в таблицу продаж нужно вставить столбец EmpID, являющийся внешним ключом, связанным с первичным ключом EmployeeID таблицы Employees

***ForeignKeyConstraint EmpSaleFK = new ("EmpSaleFK",***

***dsShop.Tables["Employees"].Columns["EmployeeID"],***

***dsShop.Tables["Sales"].Columns["EmpID"]);***

***EmpSaleFK.DeleteRule = Rule.None;***

***dsShop.Tables["Sales"].Constraints.Add(EmpSaleFK);***

## Ограничение UniqueConstraint

Ограничение **UniqueConstraint**, которое может быть назначен одному столбцу или массиву столбцов в **DataTable**, гарантирует, что все данные в указанном столбце или столбцах уникальны для каждой строки. Можно создать ограничение уникальности для столбца или массива столбцов с помощью конструктора **UniqueConstraint**. Можно также создать ограничение уникальности для столбца, задав свойству **UNIQUE** столбца **значение true**. При определении столбца или группы столбцов в качестве первичного ключа таблицы автоматически создается ограничение уникальности для заданного столбца или группы столбцов. В следующем примере создается **UniqueConstraint** для двух столбцов **таблицы товаров (Goods). В таблице Goods не должно быть двух и более товаров с одинаковым названием и одинаковой ценой одновременно.**

***DataTable goodsTable = dsShop.Tables["Goods"];***

***UniqueConstraint goodUnique = new UniqueConstraint(new DataColumn[]***

***{goodsTable.Columns["good"], goodsTable.Columns["price"]};***

***dsShop.Tables["Goods"].Constraints.Add(goodUnique);***

## Пример

Следующий пример показывает, как создать объект [DataSet](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.dataset), добавить новый объект [DataTable](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datatable) к [DataSet](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.dataset), а затем добавить к таблице пять объектов [DataColumn](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumn). В конце код задает один столбец в качестве столбца первичного ключа.

***DataSet dsShop = new DataSet("Shop");***

***DataTable empTable = dsShop.Tables.Add("Employees");***

***DataColumn empID = empTable.Columns.Add("EmployeeID", typeof(Int32));***

***empID.AutoIncrement =* true*;***

***empID.AutoIncrementSeed = 200;***

***empID.AutoIncrementStep = 3;***

***empTable.Columns.Add("LName", typeof(string));***

***empTable.Columns.Add("FName", typeof(string));***

***empTable.Columns.Add("MName", typeof(string));***

***empTable.Columns.Add("BirthDate", typeof(DateTime));***

***empTable.Columns.Add("DeptID", typeof(Int32));***

***empTable.PrimaryKey = new DataColumn[] {empTable.Columns["EmployeeID"]};***

Аналогично можно добавить в DataSet таблицу “Depts” (отделы)

***DataTable deptsTable = dsShop.Tables.Add("Depts");***

***DataColumn deptID = deptsTable.Columns.Add("ID", typeof(Int32));***

***deptsTable.Columns.Add("Dept", typeof(string));***

***deptsTable.PrimaryKey = new DataColumn[] { deptID };***

**4.3 Добавление отношений DataRelation**

В наборе [DataSet](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.dataset) с несколькими объектами [DataTable](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datatable) можно использовать объекты [DataRelation](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datarelation) для связи таблиц друг с другом, для перехода по таблицам, а также для возвращения дочерних или родительских строк из связанных таблиц.

Аргументы, необходимые для создания **DataRelation** , — это имя создаваемого объекта **DataRelation**, а также массив из одной или нескольких [DataColumn](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datacolumn) ссылок на столбцы, которые служат в качестве родительских и дочерних столбцов связи (первичных и внешних ключей). После создания объекта **DataRelation** его можно использовать для перехода между таблицами и получения значений.

Добавление **DataRelation** в [DataSet](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.dataset) добавляет по умолчанию объект [UniqueConstraint](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.uniqueconstraint) в родительскую таблицу и [ForeignKeyConstraint](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.foreignkeyconstraint) в дочернюю таблицу.

В следующем примере кода создается **отношение DataRelation** с использованием двух [DataTable](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.datatable) объектов в [DataSet](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.dataset). Первым объектом DataTable является ранее созданная таблица Employees, содержащая первичный ключ EmployeeID, а вторым таблица Sales, содержащая, содержащая внешний ключ EmpID. В примере добавляется **отношение (связь) DataRelation** к коллекции отношений объекта [DataSet](https://docs.microsoft.com/ru-ru/dotnet/api/system.data.dataset). Первый аргумент в примере указывает имя создаваемого объекта **DataRelation**. Второй аргумент задает родительский **столбец** данных (первичный ключ), а третий аргумент задает дочерний **столбец** данных (внешний ключ).

dsShop.Relations.Add("EmpSalesRel", dsShop.Tables["Employees"].Columns["EmployeeID"],

dsShop.Tables["Sales"].Columns["EmpID"]);

**Задание на лабораторную работу**

1. Разработать форму (WindowsForms) для отображения одной из таблиц базы данных (из индивидуального задания).

2. Прочитать одну из таблиц базы данных в объект DataSet и отобразить данные таблицы в форме.

3. Создать вручную в DataSet таблицу , внести в нее программно 2-3 записи и отобразить таблицу в форме.

4. Создать в форме элементы для внесения данных в прочитанную из базы данных таблицу, находящуюся в DataSet.

5. Внести в прочитанную из базы данных таблицу, 2- 3 записи и сохранить изменения в базе данных.

6. Заново прочитать таблицу из базы данных и убедиться в том, что внесенные данные сохранились

**Содержание отчета**

1. Тема и цель работы
2. Задание на лабораторную работу
3. Содержимое таблицы базы данных.
4. Код чтение данных из базы данных в DataSet и отображения данных в форме.
5. Внешний вид формы.
6. Код создания и заполнения данными таблицы в DataSet.
7. Внешний вид формы с отображением данных созданной таблицы.
8. Код изменения данных в DataSet и сохранения таблицы в базе данных.
9. Внешний вид формы с отображением данных измененной таблицы.
10. Выводы

**Контрольные вопросы**

1. Для чего предназначен объект DataSet?
2. Какие объекты и методы используются для заполнения объекта DataSet?
3. Как можно создать таблицу в DataSet?
4. Как из объекта DataSet можно отобрать нужные строки из таблицы?
5. Для чего предназначен объект SqlCommandBuilder?
6. Какие объекты и методы используются для передачи данных из DataSet в базу данных?