Автономный уровень

Автономный уровень ADO.NET позволяет смоделировать в памяти данные из базы данных, внутри вызывающего уровня, за счет применения многочисленных членов из пространства имен System.Data, таких как DataSet, DataTable, DataRow, DataColumn, DataView и DataRelation.

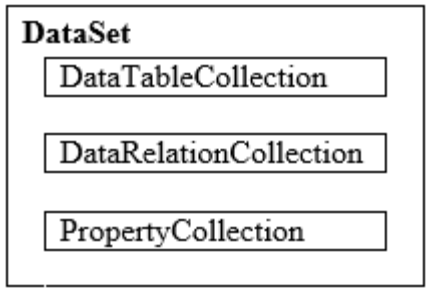
Возникает иллюзия, что вызывающий уровень постоянно подключен к внешнему источнику данных, хотя на самом деле все операции выполняются с локальной копией реляционных данных

Автономные типы можно использовать без подключения к базе данных.

Объекты ***адаптеров данных*** выполняют связующую роль между клиентским уровнем и реляционной базой данных. С их помощью можно получить объекты DataSet, поработать с их содержимым и отправить измененные строки обратно для дальнейшей обработки.

Роль объектов DataSet

Объект DataSet является представлением реляционных данных, находящимся в памяти.



- DataTableCollection содержит отдельные объекты DataTable.
- DataRelationCollection содержит ограничения и связи между таблицами.
- ExtendedProperties позволяет связать с DataSet любую дополнительную информацию в виде пар имя/значение.

Свойства класса DataSet

Свойство	Назначение
CaseSensitive	Указывает, чувствительны ли к регистру букв сравнения строк в объектах DataTable. По умолчанию равно false (сравнения строк выполняются без учета регистра букв)
DataSetName	Задает понятное имя для данного DataSet. Обычно это значение передается через параметр конструктора
EnforceConstraints	Задает или получает значение, определяющее, применяются ли правила ограничений при выполнении любых обновлений (по умолчанию равно true)
HasErrors	Получает значение, определяющее, имеются ли ошибки в любой строке любого из объектов DataTable данного DataSet
RemotingFormat	Позволяет определить, как DataSet должен сериализовать свое содержимое (в виде двоичного файла или, по умолчанию, XML)

Роль объектов DataSet

```
static void Main(string[] args)
{
    // Создание объекта DataSet и добавление нескольких свойств.
    DataSet carsInventoryDS = new DataSet("Car Inventory");
    carsInventoryDS.ExtendedProperties["TimeStamp"] = DateTime.Now;
    carsInventoryDS.ExtendedProperties["DataSetID"] = Guid.NewGuid();
    carsInventoryDS.ExtendedProperties["Company"] = "Супер-гипер-магазин Mikko";
    Console.ReadLine();
}
```

Роль объектов DataSet

Основные методы класса DataSet

Метод	Назначение
AcceptChanges()	Отправляет все изменения, выполненные в данном DataSet после его загрузки или последнего вызова AcceptChanges()
Clear()	Полностью очищает DataSet, удаляя все строки в каждом DataTable
Clone()	Клонирует структуру DataSet, в том числе и всех DataTable, а также все отношения и ограничения
Copy()	Копирует структуру и данные текущего DataSet
GetChanges()	Возвращает копию DataSet, содержащую все изменения, которые были выполнены в данном DataSet после его загрузки или последнего вызова AcceptChanges(). У этого метода есть перегруженные варианты, которые позволяют получить только новые строки, только измененные строки или только удаленные строки
HasChanges()	Выдает, содержит ли DataSet изменения, т.е. новые, удаленные или измененные строки
Merge()	Объединяет данный DataSet с указанным DataSet
ReadXml()	Позволяет определить структуру объекта DataSet и заполнить его данными на основе XML-схемы и данных из потока
RejectChanges()	Отменяет все изменения, которые были выполнены в данном DataSet после его загрузки или последнего вызова AcceptChanges()
WriteXml()	Позволяет записать содержимое DataSet в поток

Работа с объектами DataColumn

Тип ***DataColumn*** представляет один столбец в объекте DataTable.

- Любому столбцу в таблице базы данных можно назначить набор ограничений в виде первичного ключа, значения по умолчанию, разрешения только на чтение информации и т.д.
- Каждый столбец таблицы должен относиться к одному из разрешенных в СУБД типов данных.

Свойство	Назначение
AllowDBNull	Указывает, может ли данный столбец содержать пустые значения. По умолчанию содержит значение true
AutoIncrement **AutoIncrementSeed** **AutoIncrementStep**	Применяются для настройки поведения автоинкремента для данного столбца. Это может оказаться удобным, если нужно обеспечить уникальность значений в этом DataColumn (например, если он содержит первичные ключи). По умолчанию DataColumn не поддерживает автоинкрементное поведение
Caption	Задает или получает заголовок, который должен отображаться для данного столбца. Это позволяет определить более наглядные варианты для имен столбцов в базе данных
ColumnMapping	Определяет представление DataColumn при сохранении DataSet в виде XML-документа с помощью метода DataSet.WriteXml(). Можно указать, что столбец данных должен быть записан как XML-элемент, XML-атрибут, простое текстовое содержимое, либо его следует полностью проигнорировать

Работа с объектами DataColumn

Свойство	Назначение
ColumnName	Задает или получает имя столбца из коллекции Columns (т.е его внутреннее представление в DataTable). Если не занести значение в ColumnName явно, то по умолчанию там находится слово "Column" с числовыми суффиксами по формуле n+1 (т.е. Column1, Column2, Column3 и т.д.)
DataType	Определяет тип данных (логический, строковый, с плавающей точкой и т.д.), хранящихся в данном столбце
Def aultValue	Задает или получает значение по умолчанию, заносимое в данный столбец при вставке новых строк
Expression	Задает или получает выражение для фильтрации строк, вычисления значения столбца или создания агрегированного столбца
Ordinal	Задает или получает числовое положение столбца в коллекции Columns, содержащейся в DataTable
Readonly	Определяет, предназначен ли данный столбец только для чтения после добавления строки в таблицу. По умолчанию равно false
Table	Получает объект DataTable, содержащий данный DataColumn
Unique	Задает или получает значение, указывающее, должны ли быть уникальными значения во всех строках данного столбца, или допустимы совпадения. При присвоении столбцу ограничения первичного ключа свойство Unique должно содержать значение true

Работа с объектами DataColumn

Предположим, нужно смоделировать столбцы таблицы Inventory. Поскольку столбец CarID должен быть первичным ключом таблицы, его необходимо создать только для чтения, содержащим уникальные значения и не допускающим пустые значения.

```
static void FillDataSet(DataSet ds)
{
    // Создание столбцов данных, соответствующих "реальным"
    // столбцам таблицы Inventory из базы данных AutoLot.
    DataColumn carIDColumn = new DataColumn("CarID", typeof(int));
    carIDColumn.Caption = "Car ID";
    carIDColumn.ReadOnly = true;
    carIDColumn.AllowDBNull = false;
    carIDColumn.Unique = true;
    DataColumn carMakeColumn = new DataColumn("Make", typeof(string));
    DataColumn carColorColumn = new DataColumn("Color", typeof(string));
    DataColumn carPetNameColumn = new DataColumn("PetName", typeof(string));
    carPetNameColumn.Caption = "Друж.Имя";
    // Добавление объектов DataColumn в DataTable.
    DataTable inventoryTable = new DataTable("Inventory");
    inventoryTable.Columns.AddRange(new DataColumn[] { carIDColumn, carMakeColumn, carColorColumn, carPetNameColumn });
}
```

Одним из аспектов DataColumn является возможность автоинкремента: если в какую-либо таблицу добавляется новая строка, то значение автоинкрементного поля устанавливается автоматически, на основании предыдущего значения и шага автоинкремента.

```
static void FillDataSet(DataSet ds)
{
    DataColumn carIDColumn = new DataColumn("CarID", typeof(int));
    carIDColumn.ReadOnly = true;
    carIDColumn.Caption = "Car ID";
    carIDColumn.AllowDBNull = false;
    carIDColumn.Unique = true;
    carIDColumn.AutoIncrement = true;
    carIDColumn.AutoIncrementSeed = 0;
    carIDColumn.AutoIncrementStep = 1;
}
```

Работа с объектами DataRow

Коллекция объектов DataColumn представляет схему объекта DataTable, а коллекция объектов DataRow представляет конкретные данные в таблице.

Член	Назначение
HasErrors **GetColumnsInError()** **GetColumnError()** **ClearErrors()** **RowError**	Свойство HasErrors возвращает логическое значение, означающее наличие ошибок. Если они есть, то метод GetColumnsInError() позволяет получить ошибочные столбцы, а метод GetColumnError() – получить описание ошибки. Метод ClearErrors() удаляет из строки всю информацию об ошибках. Свойство RowError позволяет создать текстовое описание ошибки для данной строки
ItemArray	Свойство, задающее или получающее все значения столбцов строки в виде массива объектов
RowState	Свойство, позволяющее зафиксировать текущее состояние объекта DataRow в содержащем его DataTable с помощью значений перечисления RowState (новый, измененный, не измененный или удаленный)
Table	Свойство, позволяющее получить ссылку на объект DataTable, содержащий данный объект DataRow
AcceptChanges() **RejectChanges()**	Методы для фиксации или отмены всех изменений, выполненных в данной строке с момента последнего вызова AcceptChanges()
BeginEdit() **EndEdit()** **CancelEdit()**	Методы, начинающие, заканчивающие или отменяющие операцию редактирования для объекта DataRow
Delete()	Метод, помечающий данную строку для удаления при вызове метода AcceptChanges()
IsNull()	Метод, получающий значение, которое указывает, содержит ли заданный столбец пустое значение

Работа с объектами DataRow

Невозможно напрямую создать экземпляр типа DataRow, т.к. у него нет общедоступного конструктора

```
// Ошибка! Нет общедоступного конструктора!
DataRow r = new DataRow();
```

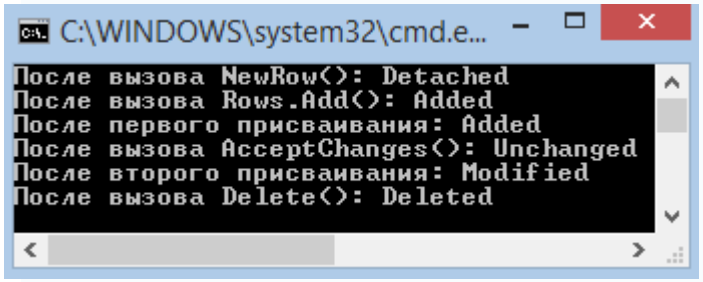
Новый объект DataRow можно получить из конкретного DataTable с помощью метода DataTable.NewRow(), который позволяет получить очередное место в таблице.
После этого можно заполнить каждый столбец с помощью индексатора типа. При этом можно указать либо строковое имя, присвоенное объекту DataColumn, либо номер его позиции (начиная с нуля)

```
static void FillDataSet(DataSet ds)
{
    Создание столбцов данных, соответствующих "реальным" столбцам таблицы Inventory из базы данных AutoLot.
    // Добавление объектов DataColumn в DataTable.
    DataTable inventoryTable = new DataTable("Inventory");
    inventoryTable.Columns.AddRange(new DataColumn[] { carIDColumn, carMakeColumn, carColorColumn, carPetNameColumn });
    // Добавление нескольких строк в таблицу Inventory.
    DataRow carRow = inventoryTable.NewRow();
    carRow["Make"] = "BMW";
    carRow["Color"] = "Black";
    carRow["PetName"] = "Hamlet";
    inventoryTable.Rows.Add(carRow);
    carRow = inventoryTable.NewRow();
    // Столбец 0 содержит автоинкрементное
    // поле, поэтому начинаем с первого.
    carRow[1] = "Saab";
    carRow[2] = "Red";
    carRow[3] = "Sea Breeze";
    inventoryTable.Rows.Add(carRow);
}
```

Свойство **RowState** применяется для программной идентификации множества всех строк таблицы, которые изменили свое первоначальное значение, были вставлены и т.п.

Значение	Назначение
Added	Строка была добавлена в DataRowCollection, а AcceptChanges() еще не был вызван
Deleted	Строка была помечена для удаления с помощью метода Delete() класса DataRow, а AcceptChanges() еще не был вызван
Detached	Строка была создана, но не включена ни в какой DataRowCollection. Объект DataRow находится в этом состоянии после его создания, но до занесения в какую-либо коллекцию, либо после исключения из коллекции
Modified	Строка была изменена, а AcceptChanges() еще не был вызван
Unchanged	Строка не была изменена после последнего вызова AcceptChanges()

```
private static void ManipulateDataRowState()
{
    // Создание DataTable.
    DataTable temp = new DataTable("Temp");
    temp.Columns.Add(new DataColumn("TempColumn", typeof(int)));
    // RowState = Detached (т.е. еще не принадлежит никакому DataTable).
    DataRow row = temp.NewRow();
    Console.WriteLine("После вызова NewRow(): {0}", row.RowState);
    // RowState = Added.
    temp.Rows.Add(row);
    Console.WriteLine("После вызова Rows.Add(): {0}", row.RowState);
    // RowState = Added.
    row["TempColumn"] = 10;
    Console.WriteLine("После первого присваивания: {0}", row.RowState);
    // RowState = Unchanged.
    temp.AcceptChanges();
    Console.WriteLine("После вызова AcceptChanges(): {0}", row.RowState);
    // RowState = Modified.
    row["TempColumn"] = 11;
    Console.WriteLine("После второго присваивания: {0}", row.RowState);
    // RowState = Deleted.
    temp.Rows[0].Delete();
    Console.WriteLine("После вызова Delete(): {0}", row.RowState);
}
```



Работа с объектами DataRow

С помощью свойства **DataRowVersion** объект DataRow отслеживает три возможные версии содержащихся в нем данных.

Значение	Назначение
Current	Представляет текущее значение строки, даже после выполнения изменений
Default	Стандартный вариант DataRowState. Если значение DataRowState равно Added, Modified или Deleted, то стандартной версией является Current. Для значения DataRowState, равного Detached, стандартной версией является Proposed
Original	Представляет значение, первоначально вставленное в DataRow, или значение при последнем вызове AcceptChanges()
Proposed	Значение строки, редактируемой в настоящий момент с помощью вызова BeginEdit()

Схема влияния методов объекта DataRow на значение свойства DataRowVersion произвольной строки

- При изменении значения строки поле вызова метода DataRow.BeginEdit() становятся доступны значения Current и Proposed.
- При вызове метода DataRow.CancelEdit() значение Proposed удаляется.
- После вызова DataRow.EndEdit() значение Proposed меняется на Current.
- После вызова метода DataRow.AcceptChanges() значение Original становится равным значению Current. То же самое происходит и при вызове DataTable.AcceptChanges().
- После вызова DataRow.RejectChanges() значение Proposed отбрасывается, и версия становится равной Current.

Работа с объектами DataTable

Основные члены типа DataTable

Член	Назначение
CaseSensitive	Указывает, чувствительны ли к регистру символов строковые сравнения в таблице. По умолчанию равно false
ChildRelations	Возвращает коллекцию дочерних отношений для данного DataTable (если они есть)
Constraints	Получает коллекцию ограничений, поддерживаемых данной таблицей
Copy()	Метод, копирующий схему и дату DataTable в новый экземпляр
DataSet	Получает DataSet, содержащий данную таблицу (если он есть)
DefaultView	Получает специализированное представление таблицы, которое может содержать отфильтрованное представление или позицию курсора
ParentRelations	Получает коллекцию родительских отношений для данного DataTable
PrimaryKey	Получает или задает массив столбцов, которые выступают в качестве первичных ключей для таблицы данных
RemotingFormat	Позволяет определить формат сериализации объектом DataSet его содержимого (двоичный или XML) для уровня .NET Remoting
TableName	Получает или задает имя таблицы. Значение этого свойства можно также задать через параметр конструктора

```
static void FillDataSet(DataSet ds)
{
    Создание столбцов данных, соответствующих "реальным" столбцам таблицы Inventory из базы данных AutoLot.
    // Добавление объектов DataColumn в DataTable.
    DataTable inventoryTable = new DataTable("Inventory");
    inventoryTable.Columns.AddRange(new DataColumn[] { carIDColumn, carMakeColumn, carColorColumn, carPetNameColumn });
    // Указание первичного ключа для данной таблицы.
    inventoryTable.PrimaryKey = new DataColumn[] { inventoryTable.Columns[0] };
    // После чего добавление таблицы в DataSet.
    ds.Tables.Add(inventoryTable);
    Добавление нескольких строк в таблицу Inventory.
}
```

```
static void FillDataSet(DataSet ds)
{
    #region Создание столбцов данных, соответствующих "реальным" столбцам таблицы Inventory из базы данных AutoLot.
    DataColumn carIDColumn = new DataColumn("CarID", typeof(int));
    carIDColumn.Caption = "Car ID";
    carIDColumn.ReadOnly = true;
    carIDColumn.AllowDBNull = false;
    carIDColumn.Unique = true;
    carIDColumn.AutoIncrement = true;
    carIDColumn.AutoIncrementSeed = 0;
    carIDColumn.AutoIncrementStep = 1;
    DataColumn carMakeColumn = new DataColumn("Make", typeof(string));
    DataColumn carColorColumn = new DataColumn("Color", typeof(string));
    DataColumn carPetNameColumn = new DataColumn("PetName", typeof(string));
    carPetNameColumn.Caption = "Друж.Имя";
    #endregion
    // Добавление объектов DataColumn в DataTable.
    DataTable inventoryTable = new DataTable("Inventory");
    inventoryTable.Columns.AddRange(new DataColumn[] { carIDColumn, carMakeColumn, carColorColumn, carPetNameColumn });
    // Указание первичного ключа для данной таблицы.
    inventoryTable.PrimaryKey = new DataColumn[] { inventoryTable.Columns[0] };
    // После чего добавление таблицы в DataSet.
    ds.Tables.Add(inventoryTable);
    #region Добавление нескольких строк в таблицу Inventory.
    DataRow carRow = inventoryTable.NewRow();
    carRow["Make"] = "BMW";
    carRow["Color"] = "Black";
    carRow["PetName"] = "Hamlet";
    inventoryTable.Rows.Add(carRow);
    carRow = inventoryTable.NewRow();
    // Столбец 0 содержит автоинкрементное
    // поле, поэтому начинаем с первого.
    carRow[1] = "Saab";
    carRow[2] = "Red";
    carRow[3] = "Sea Breeze";
    inventoryTable.Rows.Add(carRow);
    #endregion
}
```

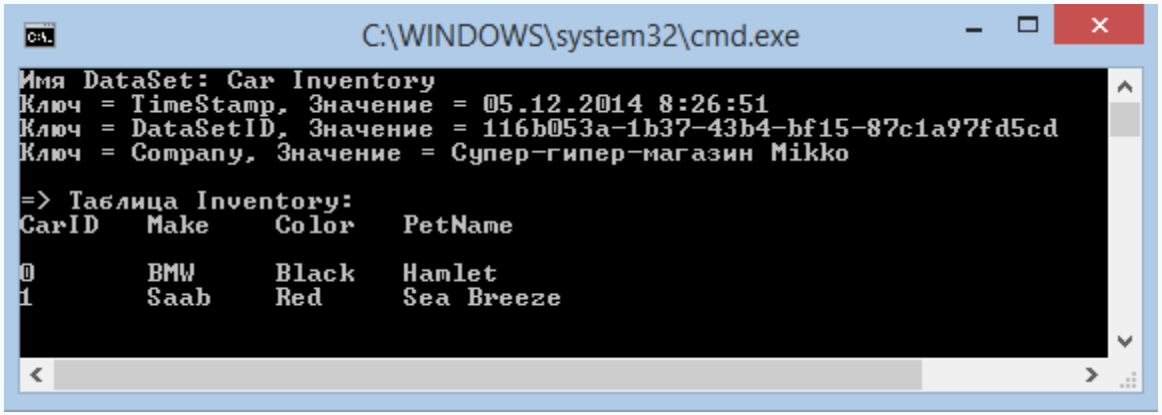
Работа с объектами DataTable

```
static void PrintDataSet(DataSet ds)
{
    // Вывод имени и расширенных свойств.
    Console.WriteLine("Имя DataSet: {0}", ds.DataSetName);
    foreach (System.Collections.DictionaryEntry de in ds.ExtendedProperties)
    {
        Console.WriteLine("Ключ = {0}, Значение = {1}", de.Key, de.Value);
    }
    Console.WriteLine();
    // Вывод каждой таблицы.
    foreach (DataTable dt in ds.Tables)
    {
        Console.WriteLine("=> Таблица {0}:", dt.TableName);
        // Вывод имен столбцов.
        for (int curCol = 0; curCol < dt.Columns.Count; curCol++)
        {
            Console.Write(dt.Columns[curCol].ColumnName + "\t");
        }
        Console.WriteLine("\n");
        // Вывод содержимого DataTable.
        for (int curRow = 0; curRow < dt.Rows.Count; curRow++)
        {
            for (int curCol = 0; curCol < dt.Columns.Count; curCol++)
            {
                Console.Write(dt.Rows[curRow][curCol].ToString() + "\t");
            }
            Console.WriteLine();
        }
    }
}
```

Метод PrintDataSet() перебирает все метаданные DataSet (используя коллекцию ExtendedProperties) и все DataTable в этом DataSet, выводя имена столбцов и значения строк с помощью индексаторов

Работа с объектами DataTable

```
static void Main(string[] args)
{
    // Создание объекта DataSet и добавление нескольких свойств.
    DataSet carsInventoryDS = new DataSet("Car Inventory");
    carsInventoryDS.ExtendedProperties["TimeStamp"] = DateTime.Now;
    carsInventoryDS.ExtendedProperties["DataSetID"] = Guid.NewGuid();
    carsInventoryDS.ExtendedProperties["Company"] = "Супер-гипер-магазин Mikko";
    FillDataSet(carsInventoryDS);
    PrintDataSet(carsInventoryDS);
    Console.ReadLine();
}
```



Обработка данных из DataTable с помощью объектов DataTableReader

Тип **DataTableReader** работает точно так же, как и объект чтения данных конкретного поставщика данных.

Объекты DataTable поддерживают метод **CreateDataReader()**, позволяющий получать данные из DataTable с помощью схемы навигации, похожей на тип чтения данных.

```
static void PrintTable(DataTable dt)
{
    // Создание объекта DataTableReader.
    DataTableReader dtReader = dt.CreateDataReader();
    // DataTableReader работает так же, как и DataReader.
    while (dtReader.Read())
    {
        for (int i = 0; i < dtReader.FieldCount; i++)
        {
            Console.Write("{0}\t", dtReader.GetValue(i).ToString().Trim());
        }
        Console.WriteLine();
    }
    dtReader.Close();
}
```

Объект чтения читает данные находящейся таблицы DataTable, а не из реальной базы данных, поэтому здесь подключение к базе данных не требуется.

Тип DataTableReader очень удобен, если нужно быстро загрузить данными объект DataTable, не путаясь во внутренних коллекциях строк и столбцов.

```
// Вывод содержимого DataTable.
for (int curRow = 0; curRow < dt.Rows.Count; curRow++)
{
    for (int curCol = 0; curCol < dt.Columns.Count; curCol++)
    {
        Console.Write(dt.Rows[curRow][curCol].ToString() + "\t");
    }
    Console.WriteLine();
}
```

Сериализация объектов DataTable и DataSet в формате XML

Методы **WriteXml()** и **ReadXml()** позволяют сохранить и получить содержимое объекта DataSet (или DataTable) в локальном файле в виде XML-документа.

Методы **WriteXmlSchema()** и **ReadXmlSchema()** сохраняют и загружают в файле *.xsd только схему

```
static void DataSetAsXml(DataSet carsInventoryDS)
{
    // Сохранение данного DataSet в виде XML.
    carsInventoryDS.WriteXml("carsDataSet.xml");
    carsInventoryDS.WriteXmlSchema("carsDataSet.xsd");
    // Очистка DataSet.
    carsInventoryDS.Clear();
    // Загрузка DataSet из XML-файла.
    carsInventoryDS.ReadXml("carsDataSet.xml");
}
```

Содержимое файла carsDataSet.xml

```
<?xml version="1.0" standalone="yes"?>
<Car_x0020_Inventory>
 <Inventory>
  <CarID>0</CarID>
  <Make>BMW</Make>
  <Color>Black</Color>
  <PetName>Hamlet</PetName>
 </Inventory>
 <Inventory>
  <CarID>1</CarID>
  <Make>Saab</Make>
  <Color>Red</Color>
  <PetName>Sea Breeze</PetName>
 </Inventory>
</Car_x0020_Inventory>
```

Сериализация объектов DataTable и DataSet в двоичном формате

Содержимое объекта DataSet (или DataTable) можно также сохранить в компактном двоичном формате.

Чтобы сохранить объект DataTable или DataSet в двоичном формате, надо занести в свойство RemotingFormat значение SerializationFormat.Binary и воспользоваться типом BinaryFormatter.

```
static void DataSetAsBinary(DataSet carsInventoryDS)
{
    // Установка признака двоичной сериализации.
    carsInventoryDS.RemotingFormat = SerializationFormat.Binary;
    // Сохранение данного DataSet в двоичном виде.
    FileStream fs = new FileStream("BinaryCars.bin", FileMode.Create);
    System.Runtime.Serialization.Formatters.Binary.BinaryFormatter bFormat =
        new System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();
    bFormat.Serialize(fs, carsInventoryDS);
    fs.Close();
    // Очистка DataSet.
    carsInventoryDS.Clear();
    // Загрузка DataSet из двоичного файла.
    fs = new FileStream("BinaryCars.bin", FileMode.Open);
    DataSet data = (DataSet)bFormat.Deserialize(fs);
}
```

BinaryCars.bin

```
000028c0  47 75 69 64 0B 00 00 00  02 5F 61 02 5F 62 02 5F  Guid....._a._b._
000028d0  63 02 5F 64 02 5F 65 02  5F 66 02 5F 67 02 5F 68  c._d._e._f._g._h
000028e0  02 5F 69 02 5F 6A 02 5F  6B 00 00 00 00 00 00 00  ._i._j._k.......
000028f0  00 00 00 00 08 07 07 02  02 02 02 02 02 02 02 FB  ................
00002900  A1 F5 BF 60 B6 81 4D B6  30 45 27 97 B0 73 E5 01  ...`..M.0E'..s..
00002910  21 00 00 00 09 00 00 00  09 2A 00 00 00 05 00 00  !........*......
00002920  00 05 00 00 00 0F 22 00  00 00 02 00 00 00 08 00  ......".........
00002930  00 00 00 01 00 00 00 11  23 00 00 00 02 00 00 00  ........#.......
00002940  06 2B 00 00 00 03 42 4D  57 06 2C 00 00 00 04 53  .+....BMW.,....S
00002950  61 61 62 11 24 00 00 00  02 00 00 00 06 2D 00 00  aab.$........-..
00002960  00 05 42 6C 61 63 6B 06  2E 00 00 00 03 52 65 64  ..Black......Red
00002970  11 25 00 00 00 02 00 00  00 06 2F 00 00 00 06 48  .%......../....H
00002980  61 6D 6C 65 74 06 30 00  00 00 0A 53 65 61 20 42  amlet.0....Sea B
00002990  72 65 65 7A 65 01 26 00  00 00 0C 00 00 00 09 31  reeze.&........1
000029a0  00 00 00 02 00 00 00 02  00 00 00 01 27 00 00 00  ............'...
000029b0  0C 00 00 00 09 32 00 00  00 02 00 00 00 02 00 00  .....2..........
000029c0  00 01 28 00 00 00 0C 00  00 00 09 33 00 00 00 02  ..(........3....
```

Привязка объектов DataTable к графическим интерфейсам Windows Forms

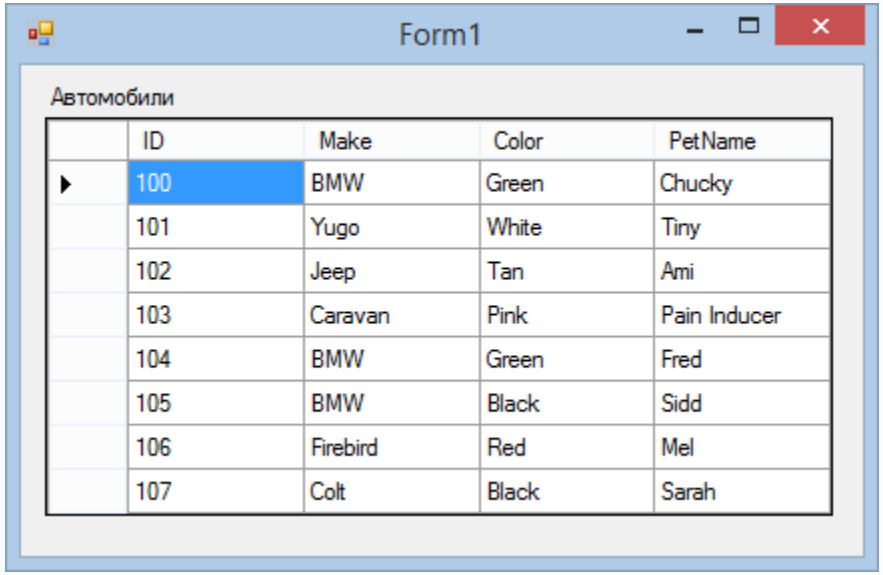
```
public partial class MainForm : Form
{
    public class Car
    {
        public int ID { get; set; }
        public string PetName { get; set; }
        public string Make { get; set; }
        public string Color { get; set; }
    }

    // Коллекия объектов Car.
    List<Car> listCars = null;
    // Информация об автомобилях на складе.
    DataTable inventoryTable = new DataTable();

    public MainForm()
    {
        InitializeComponent();
        // Добавление в список нескольких автомобилей.
        listCars = new List<Car>()
        {
            new Car {ID = 100, PetName = "Chucky", Make = "BMW", Color = "Green" },
            new Car {ID = 101, PetName = "Tiny", Make = "Yugo", Color = "White" },
            new Car {ID = 102, PetName = "Ami", Make = "Jeep", Color = "Tan" },
            new Car {ID = 103, PetName = "Pain Inducer", Make = "Caravan", Color = "Pink" },
            new Car {ID = 104, PetName = "Fred", Make = "BMW", Color = "Green" },
            new Car {ID = 105, PetName = "Sidd", Make = "BMW", Color = "Black" },
            new Car {ID = 106, PetName = "Mel", Make = "Firebird", Color = "Red" },
            new Car {ID = 107, PetName = "Sarah", Make = "Colt", Color = "Black" },
        };
        CreateDataTable();
    }

    private void CreateDataTable()...

}
```

В примере создан объект DataTable, содержащий несколько DataColumn, которые представляют различные столбцы и строки данных. При этом строки заполняются с помощью переменной-члена List<T>.

Привязка объектов DataTable к графическим интерфейсам Windows Forms

```
private void CreateDataTable()
{
    // Создание схемы таблицы.
    DataColumn carIDColumn = new DataColumn("ID", typeof(int));
    DataColumn carMakeColumn = new DataColumn("Make", typeof(string));
    DataColumn carColorColumn = new DataColumn("Color", typeof(string));
    DataColumn carPetNameColumn = new DataColumn("PetName", typeof(string));
    carPetNameColumn.Caption = "Pet Name";
    inventoryTable.Columns.AddRange(new DataColumn[] { carIDColumn, carMakeColumn, carColorColumn, carPetNameColumn });
    // Последовательное создание строк из элементов списка.
    foreach (Car c in listCars)
    {
        DataRow newRow = inventoryTable.NewRow();
        newRow["ID"] = c.ID;
        newRow["Make"] = c.Make;
        newRow["Color"] = c.Color;
        newRow["PetName"] = c.PetName;
        inventoryTable.Rows.Add(newRow);
    }
    // Привязка DataTable к carInventoryGridView.
    carInventoryGridView.DataSource = inventoryTable;
}
```

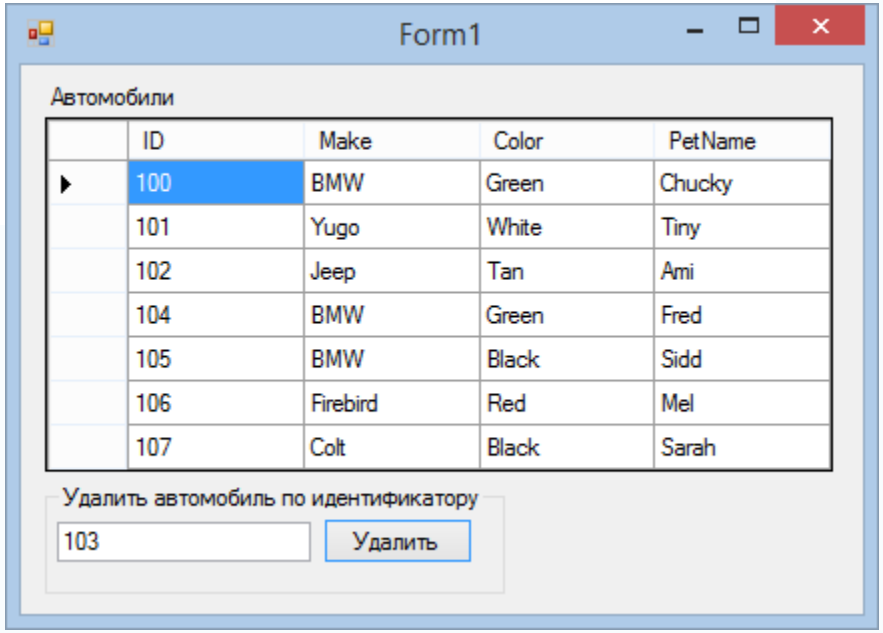
Таблица inventoryTable присваивается свойству DataSource объекта DataGridView.
Данное свойство – единственное, что нужно для привязки DataTable к объекту DataGridView.



	ID	Make	Color	PetName
▶	100	BMW	Green	Chucky
	101	Yugo	White	Tiny
	102	Jeep	Tan	Ami
	103	Caravan	Pink	Pain Inducer
	104	BMW	Green	Fred
	105	BMW	Black	Sidd
	106	Firebird	Red	Mel
	107	Colt	Black	Sarah

Удаление строк из DataTable

```
// Удаление данной строки из DataRowCollection.
private void btnRemoveRow_Click(object sender, EventArgs e)
{
    try
    {
        // Поиск строки, которую нужно удалить.
        DataRow[] rowToDelete = inventoryTable.Select(string.Format("ID={0}", int.Parse(txtCarToRemove.Text)));
        // Удаление.
        rowToDelete[0].Delete();
        inventoryTable.AcceptChanges();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```



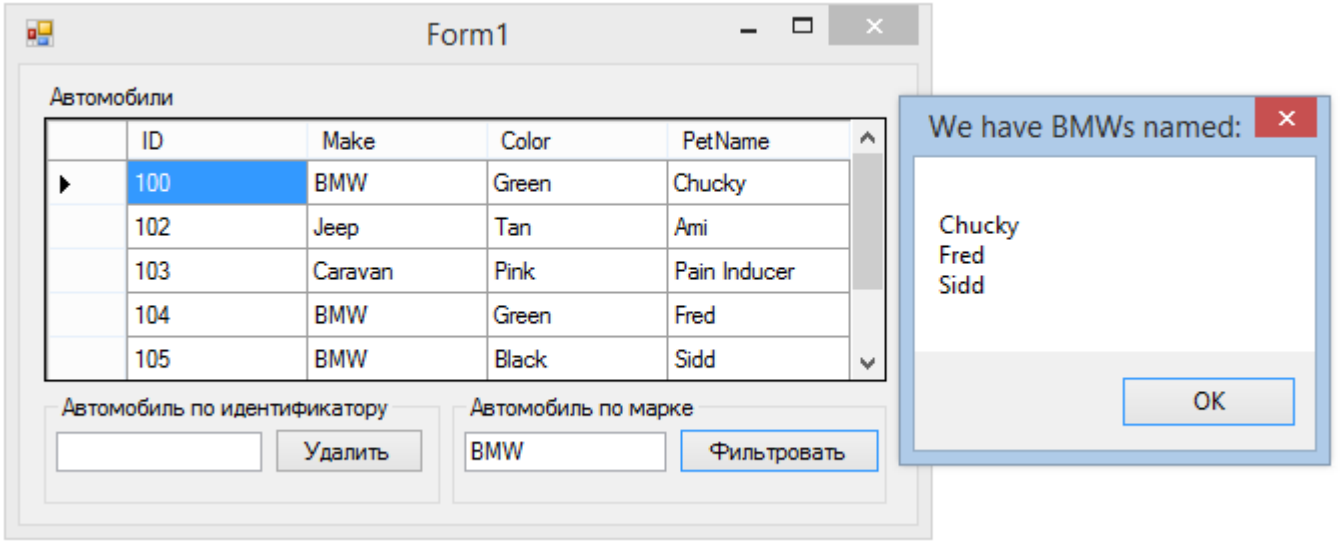
Метод удаляет указанную пользователем строку (по идентификатору автомобиля) из находящегося в памяти объекта DataTable.

Метод Select() из класса DataTable позволяет указать критерий поиска, который похож на обычный синтаксис SQL и возвращает массив объектов, удовлетворяющих критерию поиска.

Выборка строк с помощью фильтра

```
private void btnDisplayMakes_Click(object sender, EventArgs e)
{
    // Создание фильтра на основании введенных пользователем данных.
    string filterStr = string.Format("Make= '{0}'", txtMakeToView.Text);
    // Поиск всех строк, удовлетворяющих фильтру.
    DataRow[] makes = inventoryTable.Select(filterStr);
    if (makes.Length == 0)
        MessageBox.Show("Sorry, no cars...", "Selection error!"); // ничего не найдено
    else
    {
        string strMake = null;
        for (int i = 0; i < makes.Length; i++)
        {
            // Получение значения PetName из текущей строки.
            strMake += makes[i]["PetName"] + "\n";
        }
        // Вывод названий всех найденных автомобилей указанной марки.
        MessageBox.Show(strMake, string.Format("We have {0}s named:", txtMakeToView.Text));
    }
}
```

Метод Select() класса DataTable позволяет также сделать выборку подмножества записей для их отображения

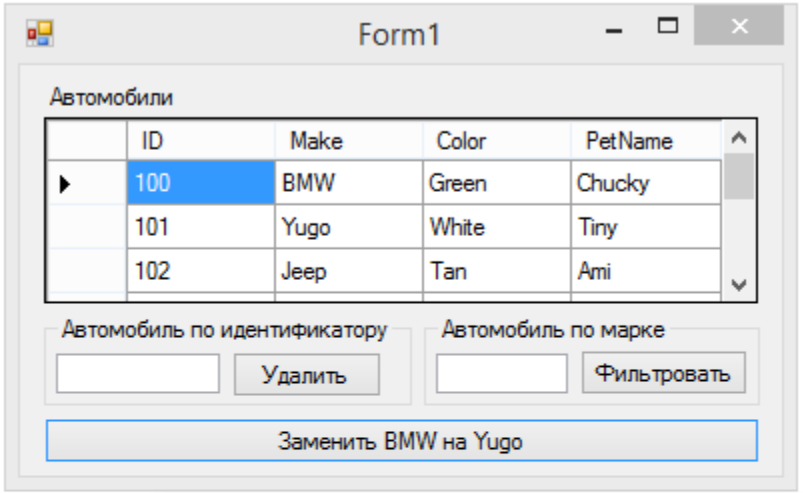


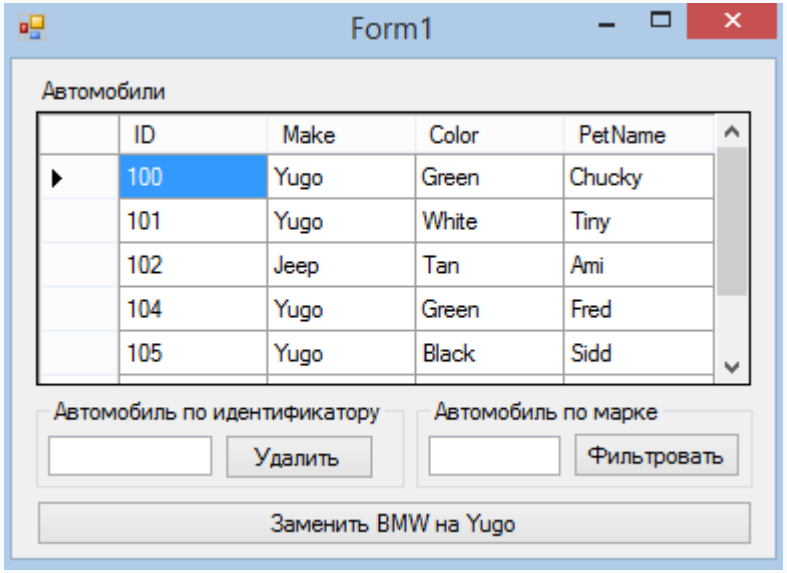

```
// Вывод результатов в порядке убывания.
makes = inventoryTable.Select(filterStr, "PetName DESC");
```

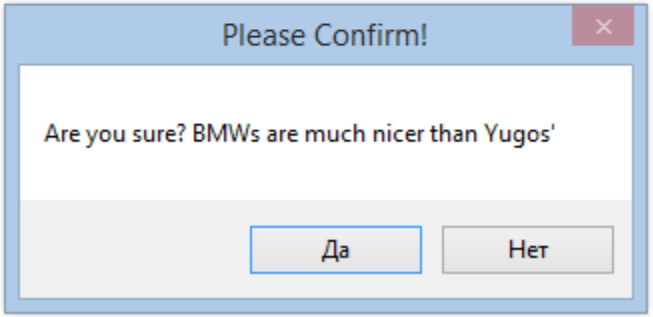
```
// Вывод всех автомобилей с ID, большим 5
makes = inventoryTable.Select("ID > 5", "PetName DESC");
```

Изменение строк в DataTable

```
// Поиск с помощью фильтра всех строк, которые нужно изменить.
private void btnChangeMakes_Click(object sender, EventArgs e)
{
    // Проверка выбора пользователя.
    if (DialogResult.Yes == MessageBox.Show("Are you sure? BMWs are much nicer than Yugos'",
            "Please Confirm!", MessageBoxButtons.YesNo))
    {
        // Создание фильтра.
        string filterStr = "Make='BMW'";
        string strMake = string.Empty;
        // Поиск всех строк, удовлетворяющих фильтру.
        DataRow[] makes = inventoryTable.Select(filterStr);
        // Замена всех BMW на Yugo.
        for (int i = 0; i < makes.Length; i++)
        {
            makes[i]["Make"] = "Yugo";
        }
    }
}
```







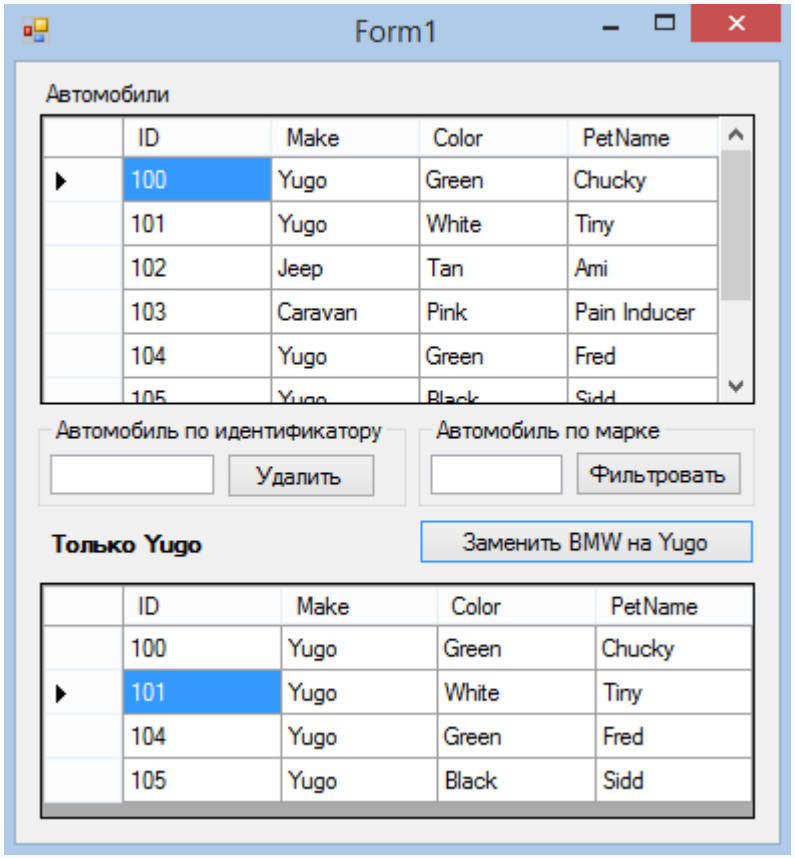
Работа с типом DataView

Объект представления – это альтернативное представление таблицы (или набора таблиц).

В ADO.NET тип DataView позволяет программным образом извлекать подмножество данных из DataTable в отдельный объект.

```
// Отображение содержимого DataTable.
DataView yugosOnlyView;

public MainForm()
{
    InitializeComponent();
    Добавление в список нескольких автомобилей.
    CreateDataTable();
    // Создание представления.
    CreateDataView();
}

private void CreateDataTable()...

// Удаление данной строки из DataRowCollection.
private void btnRemoveRow_Click(object sender, EventArgs e)...

private void btnDisplayMakes_Click(object sender, EventArgs e)...

// Поиск с помощью фильтра всех строк, которые нужно изменить.
private void btnChangeMakes_Click(object sender, EventArgs e)...

private void CreateDataView()
{
    // Указание таблицы для создания данного представления.
    yugosOnlyView = new DataView(inventoryTable);
    // Настройка представлений с помощью фильтра.
    yugosOnlyView.RowFilter = "Make = 'Yugo'";
    // Привязка к новой графической таблице.
    dataGridYugoView.DataSource = yugosOnlyView;
}
```

Серьезным преимуществом наличия нескольких представлений одной и той же таблицы является возможность привязать их к различным графическим элементам наподобие DataGridView.



Работа с адаптерами данных

Класс адаптеров данных применяется для заполнения наборов данных DataSet с помощью объектов DataTable; кроме того, они могут отправлять измененные DataTable назад в базу данных для обработки.

Адаптер данных определяет четыре свойства: SelectCommand, InsertCommand, UpdateCommand и DeleteCommand.

При создании объекта адаптера данных для конкретного поставщика данных (например, SqlDataAdapter) можно передать строку с текстом команды, используемом объектом команды SelectCommand.

После должной настройки каждого из четырех объектов команд можно вызвать метод Fill() и получить объект DataSet (DataTable).

При необходимости сохранить измененный объект DataSet (DataTable) в базе данных для обработки можно вызвать метод Update().

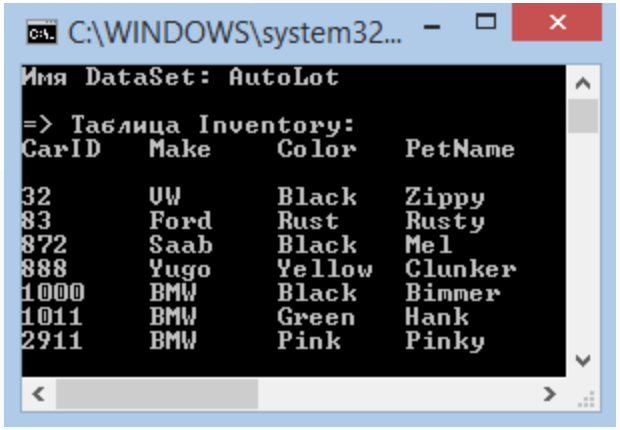
При работе с объектом адаптера данных не нужно открывать или закрывать подключение к базе данных. Все это делается автоматически.

Член	Назначение
Fill()	Выполняет команду SQL SELECT (указанную в свойстве SelectCommand) для запроса к базе данных и загрузки этих данных в объект DataTable
SelectCommand **InsertCommand** **UpdateCommand** **DeleteCommand**	Содержат SQL-команды, отправляемые в хранилище данных при вызовах методов Fill() и Update()
Update()	Выполняет команды SQL INSERT, UPDATE и DELETE (указанных свойствами InsertCommand, UpdateCommand и DeleteCommand) для сохранения в базе данных изменений, выполненных в DataTable

Работа с адаптерами данных

Для создания адаптера данных используется строковый литерал, который преобразуется в SQL-оператор Select. Из этого значения адаптер создает объект команды, который потом можно получить с помощью свойства SelectCommand.

```
static void Main(string[] args)
{
    // Жестко закодированная строка подключения.
    string cnStr = @"Integrated Security = SSPI;Initial Catalog=AutoLot;Data Source=(local)";
    // Создание объекта DataSet вызывающим процессом.
    DataSet ds = new DataSet("AutoLot");
    // Передача адаптеру текста команды Select и подключения.
    SqlDataAdapter dAdapt = new SqlDataAdapter("Select * From Inventory", cnStr);
    // Заполнение DataSet новой таблицей с именем Inventory.
    dAdapt.Fill(ds, "Inventory");
    // Вывод содержимого DataSet.
    PrintDataSet(ds);
    Console.ReadLine();
}
```

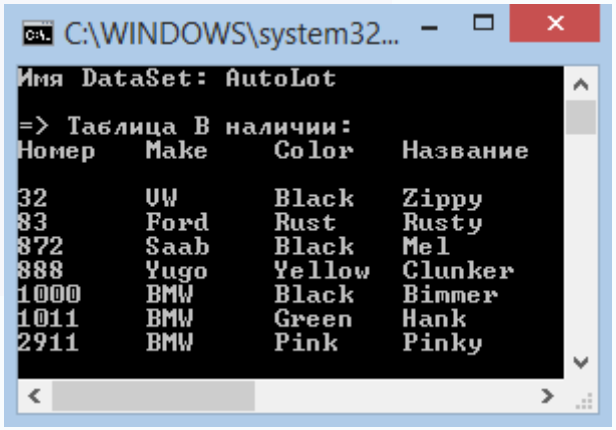


Экземпляр класса DataSet должен быть создан вызывающим процессом, и уже затем передан в метод Fill(), в который можно передать второй, не обязательный, аргумент – строку, с чьей помощью будет сформировано свойство TableName нового объекта DataTable

Замена имен из базы данных более понятными названиями

Объекты адаптеров данных содержат внутреннюю строго типизированную коллекцию DataTableMappingCollection объектов типа System.Data.Common.DataTableMapping.
Доступ к этой коллекции возможен через свойство TableMappings объекта адаптера данных, и используется для задания отображаемых имен при выводе содержимого объекта DataTable.

```
static void Main(string[] args)
{
    // Жестко закодированная строка подключения.
    string cnStr = @"Integrated Security = SSPI;Initial Catalog=AutoLot;Data Source=(local)";
    // Создание объекта DataSet вызывающим процессом.
    DataSet ds = new DataSet("AutoLot");
    // Передача адаптеру текста команды Select и подключения.
    SqlDataAdapter dAdapt = new SqlDataAdapter("Select * From Inventory", cnStr);
    // Соответствие имен столбцов базы данных и понятных названий.
    DataTableMapping custMap =
    dAdapt.TableMappings.Add("Inventory", "В наличии");
    custMap.ColumnMappings.Add("CarID", "Номер");
    custMap.ColumnMappings.Add("PetName", "Название");
    // Заполнение DataSet новой таблицей с именем Inventory.
    dAdapt.Fill(ds, "Inventory");
    // Вывод содержимого DataSet.
    PrintDataSet(ds);
    Console.ReadLine();
}
```



Проект AutoLotDAL

```
namespace AutoLotDisconnectedLayer
{
    public class InventoryDALDisLayer
    {
        // Значения полей.
        private string cnString = string.Empty;
        private SqlDataAdapter dAdapt = null;
        public InventoryDALDisLayer(string connectionString)
        {
            cnString = connectionString;
            // Настройка SqlDataAdapter.
            ConfigureAdapter(out dAdapt);
        }

        private void ConfigureAdapter(out SqlDataAdapter dAdapt)
        {
            // Создание адаптера и заполнение SelectCommand.
            dAdapt = new SqlDataAdapter("Select * From Inventory", cnString);
            // Динамическое получение остальных объектов команд
            // во время выполнения с помощью SqlCommandBuilder.
            SqlCommandBuilder builder = new SqlCommandBuilder(dAdapt);
        }
    }
}
```

При использовании адаптера данных для модификации таблиц в наборе данных DataSet вначале нужно указать в свойствах UpdateCommand, DeleteCommand и InsertCommand допустимые объекты команд.

Для упрощения создания объектов адаптеров данных в каждом поставщике данных ADO.NET, разработанном Microsoft, имеется тип построителя команд – SqlCommandBuilder. Он автоматически генерирует значения в свойствах InsertCommand, UpdateCommand и DeleteCommand объекта SqlDataAdapter на основе первоначального объекта SelectCommand.

Проект AutoLotDAL

Построитель команд SqlCommandBuilder может автоматически генерировать команды SQL для использования их адаптером данных, если выполнены все следующие условия:

- SQL-команда Select работает только с одной таблицей (т.е. без объединений):
- у этой единственной таблицы имеется первичный ключ;
- в таблице должен быть столбец (или столбцы), который представляет первичный ключ, включенный в SQL-оператор Select.

Метод GetAllInventory вызывает метод Fill() объекта SqlDataAdapter для получения DataTable, представляющего все записи из таблицы Inventory базы данных AutoLot

```
public DataTable GetAllInventory()
{
    DataTable inv = new DataTable("Inventory");
    dAdapt.Fill(inv);
    return inv;
}
```

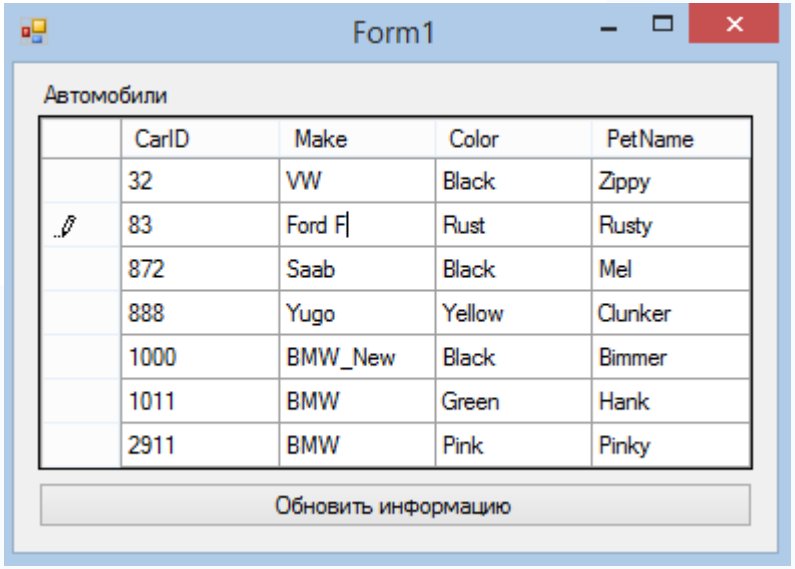
В методе UpdateInventory объект адаптера данных проверяет значение RowState у каждой строки входной таблицы. В зависимости от его значения (RowState.Added, RowState.Deleted или RowState.Modified) автоматически вызывается нужный объект команды.

```
public void UpdateInventory(DataTable modifiedTable)
{
    dAdapt.Update(modifiedTable);
}
```

Проект AutoLotDAL

```
InventoryDALDisLayer dal = null;
public MainForm()
{
    InitializeComponent();
    string cnStr = @"Data Source=(local);Initial Catalog=AutoLot;Integrated Security=True;Pooling=False";
    // Создание объекта доступа к данным.
    dal = new InventoryDALDisLayer(cnStr);
    // Заполнение графической таблицы!
    inventoryGrid.DataSource = dal.GetAllInventory();
}

private void btnUpdateInventory_Click(object sender, EventArgs e)
{
    // Получение измененных данных из графической таблицы.
    DataTable changedDT = (DataTable)inventoryGrid.DataSource;
    try
    {
        // Внесение изменений.
        dal.UpdateInventory(changedDT);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

После создания объекта InventoryDALDisLayer можно выполнить привязку DataTable, возвращенного вызовом GetAllInventory(), к объекту DataGridView.



Когда конечный пользователь щелкнет на кнопке «Обновить информацию», из графической таблицы выбирается модифицированный DataTable с помощью свойства DataSource и передается в метод UpdateInventory()

Объекты DataSet для нескольких таблиц и взаимосвязь данных

Объект DataSet может содержать несколько взаимосвязанных DataTable. В этом случае в коллекцию DataRelation данного DataSet можно вставить любое количество объектов DataRelation, которые описывают все взаимосвязи таблиц. Эти объекты позволяют клиентскому уровню выполнять навигацию между данными таблиц без обращения к сети.

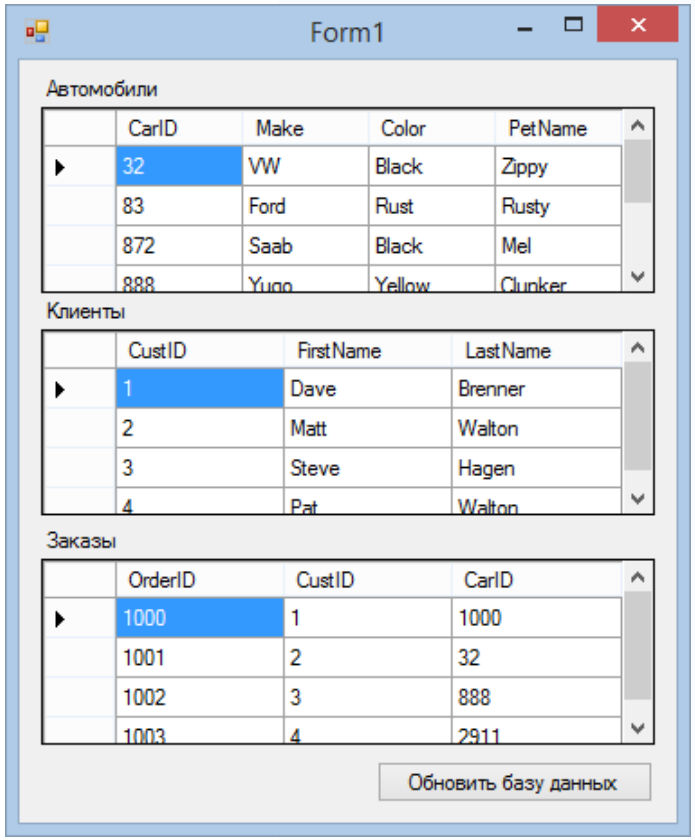
```
// Формирование DataSet.
private DataSet autoLotDS = new DataSet("AutoLot");
// Использование построителей команд для упрощения настройки адаптера данных.
private SqlCommandBuilder sqlCBInventory;
private SqlCommandBuilder sqlCBCustomers;
private SqlCommandBuilder sqlCBOrders;
// Адаптеры данных (для каждой таблицы).
private SqlDataAdapter invTableAdapter;
private SqlDataAdapter custTableAdapter;
private SqlDataAdapter ordersTableAdapter;
// Формирование строки подключения.
private string cnStr = string.Empty;
```

```
private void BuildTableRelationship()
{
    // Создание объекта отношения между данными CustomerOrder.
    DataRelation dr = new DataRelation("CustomerOrder",
                      autoLotDS.Tables["Customers"].Columns["CustID"],
                      autoLotDS.Tables["Orders"].Columns["CustID"]);
    autoLotDS.Relations.Add(dr);
    // Создание объекта отношения между данными InventoryOrder.
    dr = new DataRelation("InventoryOrder",
         autoLotDS.Tables["Inventory"].Columns["CarID"],
         autoLotDS.Tables["Orders"].Columns["CarID"]);
    autoLotDS.Relations.Add(dr);
}
```

Функция BuildTableRelationship() берет на себя рутинные действия по добавлению в объект autoLotDS двух объектов DataRelation.

Проект AutoLotDAL

```
public MainForm()
{
    InitializeComponent();
    // Получение строки подключения из файла *.config.
    cnStr = ConfigurationManager.ConnectionStrings["AutoLotSqlProvider"].ConnectionString;
    // Создание адаптеров.
    invTableAdapter = new SqlDataAdapter("Select * from Inventory", cnStr);
    custTableAdapter = new SqlDataAdapter("Select * from Customers", cnStr);
    ordersTableAdapter = new SqlDataAdapter("Select * from Orders", cnStr);
    // Генерация команд.
    sqlCBInventory = new SqlCommandBuilder(invTableAdapter);
    sqlCBOrders = new SqlCommandBuilder(ordersTableAdapter);
    sqlCBCustomers = new SqlCommandBuilder(custTableAdapter);
    // Добавление таблиц в DataSet.
    invTableAdapter.Fill(autoLotDS, "Inventory");
    custTableAdapter.Fill(autoLotDS, "Customers");
    ordersTableAdapter.Fill(autoLotDS, "Orders");
    // Создание отношений между таблицами.
    BuildTableRelationship();
    // Привязка к графическим элементам
    dataGridViewInventory.DataSource = autoLotDS.Tables["Inventory"];
    dataGridViewCustomers.DataSource = autoLotDS.Tables["Customers"];
    dataGridViewOrders.DataSource = autoLotDS.Tables["Orders"];
}

private void btnUpdateDatabase_Click(object sender, EventArgs e)
{
    try
    {
        invTableAdapter.Update(autoLotDS, "Inventory");
        custTableAdapter.Update(autoLotDS, "Customers");
        ordersTableAdapter.Update(autoLotDS, "Orders");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }

}
```



Переходы между взаимосвязанными таблицами

Объекты DataRelation позволяют программно переходить между взаимосвязанными таблицами.

```
private void btnGetOrderInfo_Click(object sender, EventArgs e)
{
    string strOrderInfo = string.Empty;
    DataRow[] drsCust = null;
    DataRow[] drsOrder = null;
    // Получить идентификатор клиента из текстового поля.
    int custID = int.Parse(this.txtCustID.Text);
    // На основе custID получить подходящую строку из таблицы Customers.
    drsCust = autoLotDS.Tables["Customers"].Select(string.Format("CustID= {0}", custID));
    strOrderInfo += string.Format("Customer {0}: {1} {2}\n",
                                  drsCust[0]["CustID"].ToString(),
                                  drsCust[0]["FirstName"].ToString(),
                                  drsCust[0]["LastName"].ToString());
    // Перейти из таблицы Customers в таблицу Orders.
    drsOrder = drsCust[0].GetChildRows(autoLotDS.Relations["CustomerOrder"]);
    // Проход в цикле по всем заказам этого клиента.
    foreach (DataRow order in drsOrder)
    {
        strOrderInfo += string.Format("\nOrder Number: {0}\n", order["OrderID"]);
        // Получить автомобиль, на который ссылается этот заказ.
        DataRow[] drsInv = order.GetParentRows(autoLotDS.Relations["InventoryOrder"]);
        // Получить информацию для (ОДНОГО) автомобиля из этого заказа.
        DataRow car = drsInv[0];
        strOrderInfo += string.Format("Make: {0}\n", car["Make"]); // Марка
        strOrderInfo += string.Format("Color: {0}\n", car["Color"]); // Цвет
        strOrderInfo += string.Format("Pet Name: {0}\n", car["PetName"]); // Дружественное имя
    }
    MessageBox.Show(strOrderInfo, "Order Details");
}
```

Результат работы проекта AutoLotDAL