Доступ к данным при помощи ADO.NET

Лекция 10

Общие характеристики

- Установить соединение с БД
- Создать и заполнить данными объект DataSet
- Вернуть изменения обратно в БД

- SQL
- OLE DB

Пространства имен

System.Data	Главное пространство. Определен тип DataSet
System.Data. Common	Типы для пройвайдеров
System.Data. OleDb	Типы для SQL-запросов к OLE DB-совместимым БД
System.Data. SqlClient	Типы для взаимодействия с MS SQL Server
System.Data. SqlTypes	«Родные» типы данных MS SQL Server

Типы пространства имен System.Data

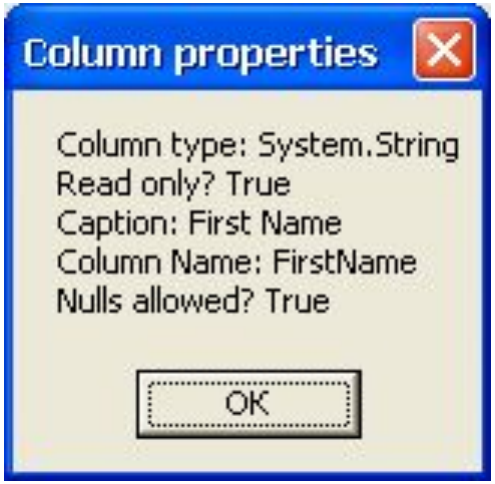
DataColumn DataColumnCollection	Один столбец в объекте DataTable Все столбцы в DataTabel
Constraint ConstraintCollection	Ограничения, накладываемые на DataColumn Все ограничения
DataRow DataRowCollection	Строка в DataTable Все строки в DataTable
DataRowView DataView	Настроенное представление строки Представление всего объекта DataTable
DataSet	Основной объект, создаваемый на клиентском компьютере для взаимодействия с БД
DataRelation DataRelationCollection	Отношение между таблицами Набор всех отношений
DataTable DataTableCollection	Таблица Набор всех таблиц

Свойства класса DataColumn

AllowDbNull	Может ли столбец содержать пустые значения
AutoIncrement AutoIncrementSeed AutoIncrementStep	Настройка автоматического приращения значений в столбце. По умолчанию отключено.
Caption	Заголовок столбца для отображения в приложении
ColumnMapping	Способ представления столбца в формате XML
ColumnName	Имя столбца в коллекции Columns (по ум. Column1, …)
DataType	Тип данных в столбце
DefaultValue	Значение данных в столбце по умолчанию
Expression	Выражение для фильтрации новых строк
Ordinal	Устанавливает порядковый номер столбца в Columns
ReadOnly	Определяет столбец только для чтения
Table	Возвращает DataTable, которому принадлежит DataColumn
Unique	Позволяет разрешить/запретить повторяющиеся значения

Создание объекта DataColumn

```
protected void btnColumn_Click (object sender, System.EventArgs e)
{
    // Создаем столбец FirstName
    DataColumn colFName = new DataColumn();

    // Настраиваем его параметры
    colFName.DataType = Type.GetType("System.String");
    colFName.ReadOnly = true;
    colFName.Caption = "First Name";
    colFName.ColumnName = "FirstName";

    // А теперь извлекаем информацию о них
    string temp = "Column type: " + colFName.DataType + "\n" + "Read only? " +
      colFName.ReadOnly + "\n" + "Caption: " +
      colFName.Caption + "\n" + "Column Name: " +
      colFName.ColumnName + "\n" + "Nulls allowed?" +    colFName.AllowDBNull;

      MessageBox.Show(temp, "Column Properties");
}
```




```
DataColumn colFName =
new DataColumn("FirstName“,Type.GetType("System.String");
```

Добавляем объект DataColumn в DataTable

```
// Создаем столбец myColumn
DataColumn myColumn = new DataColumn();


// Создаем таблицу myTable
    DataTable myTable = new DataTable("MyTable");

// Свойство Columns возвращает объект типа
// DataColumnCollection. Для добавления столбца
// в таблицу используется метод Add();
    myTable.Columns.Add(myColumn);
```

Делаем столбец первичным ключом таблицы

```
// Столбец EmpID будет первичным
DataColumn colEmpID = new
    DataColumn("EmpID",Type.GetType(“System.Int32"));
colEmpID.Caption = “Employee ID”;
colEmpID.AllowDBNull=false;
colEmpID.Unique=true;

// Еще надо воспользоваться свойством
//  DataTable.PrimaryKey
```

Настройка автоматического увеличения значений

```
// Создаем столбец данных
DataColumn myColumn = new DataColumn();
myColumn.ColumnName = "Foo";
myColumn.DataType = System.Type.GetType("System.Int32");

// Настраиваем автоматическое увеличение значений
myColumn.AutoIncrement = true;
myColumn.AutoIncrementSeed = 500;
myColumn.AutoIncrementStep = 12;

// Добавляем этот столбец в таблицу
DataTable myTable = new DataTable("MyTable");
myTable.Columns.Add(myColumn);

// Добавляем 20 строк
DataRow r;
for(int i = 0; i < 20; i++) { r = myTable.NewRow();
                    myTable.Rows.Add(r);   }

// А теперь выводим значения для каждой строки
string temp = "";
DataRowCollection rows = myTable.Rows;
for(int i = 0; i < myTable.Rows.Count; i++)   {   DataRow currRow = rows[i];
                                  temp += currRow["Foo"] + " ";  }

MessageBox.Show(temp, "These values brought ala auto-increment");
```

Тип DataRow

- Структура таблицы определяется как коллекция объектов DataColumn
- DataColumnCollection хранит ее в объекте DataTable
- Коллекция объектов DataRow – данные, хранящиеся в таблице
- Объект DataRow не нужно создавать напрямую

Тип DataRow

```
// Создаем объект таблицы
DataTable myTable = new DataTable("Employees");

// … здесь формируем структуру таблицы (столбцы)

// Создаем строку
DataRow row = empTable.NewRow();
Row[“EmpID”]=102;
Row[“FirstName”]=“Joe”;
Row[“LastName”]=“Blow”;

// Добавляем ее во внутреннюю коллекцию строк
empTable.Rows.Add(row)
```

Члены класса DataRow

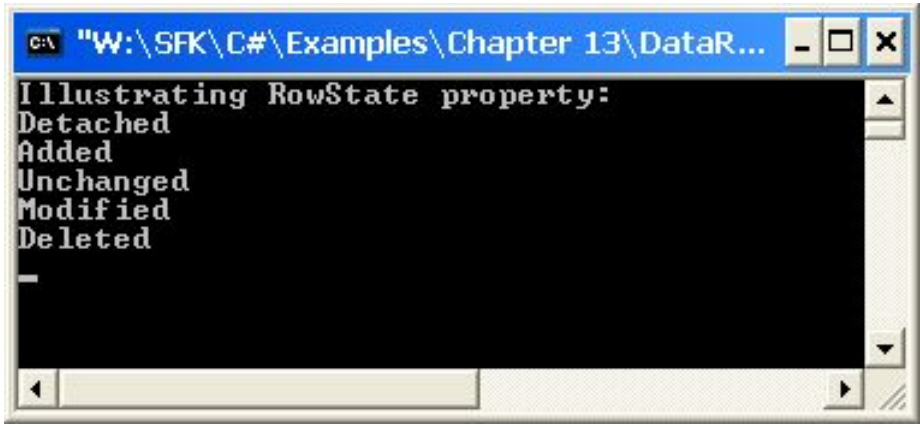
AcceptChanges() RejectChanges()	Для записи в строку (или отказа) всех изменений с момента последнего вызова AccepChanges()
BeginEdit() EndEdit() CancelEdit()	Начать, завершить, прекратить операции редактирования для объекта DataRow
Delete	Помечает строку для удаления при вызове AcceptChanges()
HasErrors GetColumnsInErrors() GetColumnError() ClearErrors() RowError	Возвращает логическое значение, определяющее наличие ошибки в значениях для данной строки. Дополнительные правила уточнят информацию об ошибке.
IsNull()	Содержит ли строка в указанном поле пустое значение
ItemArray	Получить/установить значения всех полей строки
RowState	Текущее состояние объекта DataRow
Table	Указатель на таблицу, содержащую текущий DataRow

Перечисление DataRowState

Deleted	Строка была изменена при помощи метода DataRow.Delete()
Detached	Строка создана, но не является частью DataRowCollection
Modified	Изменена, но AcceptChanges() еще не вызывался
New	Строка добавлена, но AcceptChanges() еще не вызывался
Unchanged	Строка не изменена с момента последнего вызова AcceptChanges()

Иллюстрация состояния DataRow

```
DataTable myTable = new DataTable("Employees"); // Создаем DataTable из одного столбца
DataColumn colID = new DataColumn("EmpID", Type.GetType("System.Int32"));
myTable.Columns.Add(colID);

DataRow myRow; // Начинаем работу с DataRow

myRow = myTable.NewRow(); // Создаем объект DataRow (пока в состоянии Detached)
Console.WriteLine(myRow.RowState.ToString());

myTable.Rows.Add(myRow);   // Теперь добавлем его в таблицу
Console.WriteLine(myRow.RowState.ToString());

myTable.AcceptChanges();      // Записываем изменения
Console.WriteLine(myRow.RowState.ToString());

myRow["EmpId"] = 100; // Изменяем строку
Console.WriteLine(myRow.RowState.ToString());

myRow.Delete(); // А теперь ее удаляем
Console.WriteLine(myRow.RowState.ToString());
myRow.AcceptChanges();
```

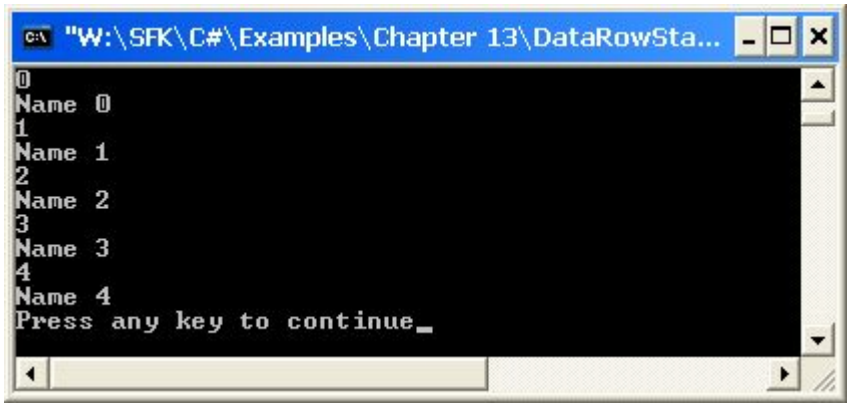


Свойство ItemArray

- ItemArray – свойство класса DataRow
- Позволяет получить полный «снимок» строки в виде массива объектов типа System.Object
- При помощи этого свойства можно вставить новую строку в таблицу, не указывая явно значения для каждого столбца

Свойство ItemArray

```
// Объявляем массив
object [] myVals = new object[2];
DataRow dr;

// Создаем новые строки и добавляем их в DataRowCollection
for (int i = 0; i < 5; i++)
{
        myVals[0] = i;
        myVals[1] = "Name " + i;
        dr = myTable.NewRow();
        dr.ItemArray = myVals;
        myTable.Rows.Add(dr);
}
// А теперь выводим каждое из значений
foreach(DataRow r in myTable.Rows)
{
        foreach(DataColumn c in myTable.Columns)
                Console.WriteLine(r[c]);
}
```



Свойства класса DataTable

CaseSensitive	Будет ли при сравнении символьных данных учитываться регистр символов
ChildRelations	Коллекция подчиненных отношений
Columns	Набор столбцов для таблицы
Constraints	Коллекция ограничений
DataSet	Ссылка на источник данных (если есть)
DafaultView	Настроенное представление для таблицы
MinimumCapacity	Исходное число строк (по умолчанию – 25)
ParentRelations	Коллекция родительских отношение
PrimaryKey	Массив столбцов, являющиеся первичными ключами
Rows	Набор строк таблицы
TableName	Получить/установить имя таблицы

Создание объекта DataTable

```
// Создаем объект DataTable
DataTable inventoryTable = new DataTable(“Invetory”);

// Объявляем переменную DataColumn
DataColumn myDataColumn;

// Создаем столбец CarID
myDataColumn = new DataColumn();
myDataColumn.DataType = Type.GetType("System.Int32");
myDataColumn.ColumnName = "CarID";
myDataColumn.ReadOnly = true;
myDataColumn.AllowDBNull = false;
myDataColumn.Unique = true;

// Настраиваем CarID как счетчик и добавляем его в таблицу
myDataColumn.AutoIncrement = true;
myDataColumn.AutoIncrementSeed = 1000;
myDataColumn.AutoIncrementStep = 10;
inventoryTable.Columns.Add(myDataColumn);
```

Создание объекта DataTable

```
// Создаем столбец Make и добавляем его в таблицу
myDataColumn = new DataColumn();
myDataColumn.DataType = Type.GetType("System.String");
myDataColumn.ColumnName = "Make";
inventoryTable.Columns.Add(myDataColumn);

// Создаем столбец Color и добавляем его в таблицу
myDataColumn = new DataColumn();
myDataColumn.DataType = Type.GetType("System.String");
myDataColumn.ColumnName = "Color";
inventoryTable.Columns.Add(myDataColumn);

// Создаем и добавляем последний столбец — PetName
myDataColumn = new DataColumn();
myDataColumn.DataType = Type.GetType("System.String");
myDataColumn.ColumnName = "PetName";
myDataColumn.AllowDbNull = true;
inventoryTable.Columns.Add(myDataColumn);
```

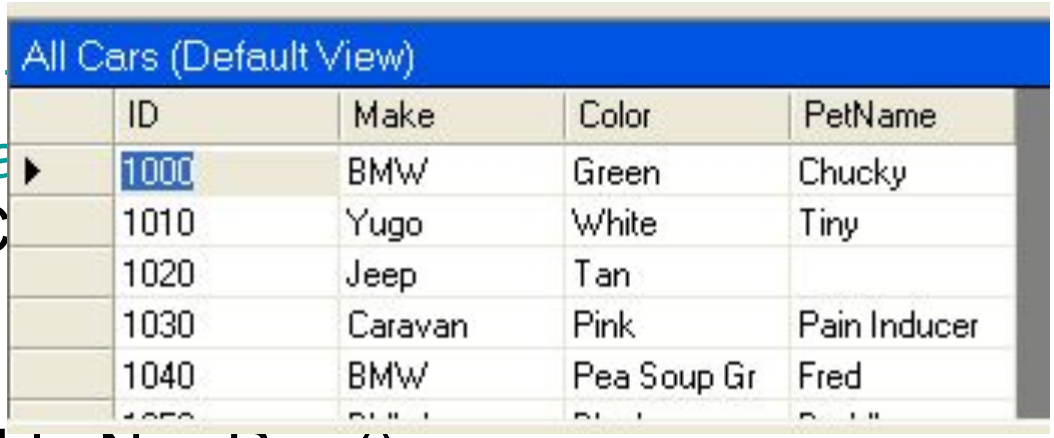
Определяем первичный ключ

```
// В качестве первичного ключа используем столбец CarID
// Первичный ключ может быть композитным, поэтому
// используется массив объектов DataColumn
DataColumn [ ] PK = new DataColumn[1];
PK[0] = inventoryTable.Columns[“CarID”];
inventoryTable.PrimaryKey = PK;
```

Заполняем данными и отображаем

```
// Пусть уже есть массив типа ArrayList объектов Car
// Последовательно создаем строки и заполняем их.
    foreach(Car c in arTheCars)
    {
     DataRow newRow;
     newRow = inventoryTable.NewRow();
     newRow["Make"] = c.make;
     newRow["Color"] = c.color;
     newRow["PetName"] = c.petName;
     inventoryTable.Rows.Add(newRow);
    }

// Пусть есть объект DataGrid.
// Связываем таблицу с этим объектом
    CarDataGrid.DataSource = inventoryTable;
```

Можно и так




```
// Пусть уже есть массив
// Последовательно созда
    foreach(Car c in arTheC
    {
     DataRow newRow;
     newRow = inventoryTable.NewRow();
     newRow[1] = c.make;
     newRow[2] = c.color;
     newRow[3] = c.petName;
     inventoryTable.Rows.Add(newRow);
    }

// Пусть есть объект DataGrid.
// Связываем таблицу с этим объектом
    CarDataGrid.DataSource = inventoryTable;
```

Удаление строк из таблицы

```
protected void btnRemoveRow_Click (object sender, System.EventArgs e)
{
    try
    {inventoryTable.Rows[(int.Parse(txtRemove.Text))].Delete();
     inventoryTable.AcceptChanges();
    }
    catch(Exception ex)
    {MessageBox.Show(ex.Message);
    }
}
```

Пока AcceptChanges() не вызван

Можно даже восстановить строк

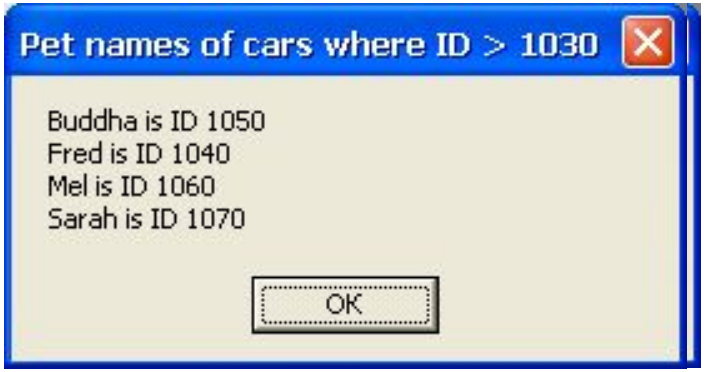
inventoryTable.RejectChanges();

Применение фильтров и порядка сортировки

```
// Создаем фильтр для отбора
string filterStr = "Make='" + txtMake.Text + "'";
string strMake = null;

// Находим все строки, соответсвующие нашему условию
DataRow[] makes = inventoryTable.Select(filterStr);

//  Выводим информацию о найденных строках
if(makes.Length == 0)
       { MessageBox.Show("Sorry, no cars...", "Selection erro
        }
else  {   for (int i = 0; i < makes.Length; i++)
     {     DataRow temp = makes[i];
           strMake += temp["PetName"].ToString() + "\n";
     }
     MessageBox.Show(strMake, txtMake.Text + " type(s):");
    }
```

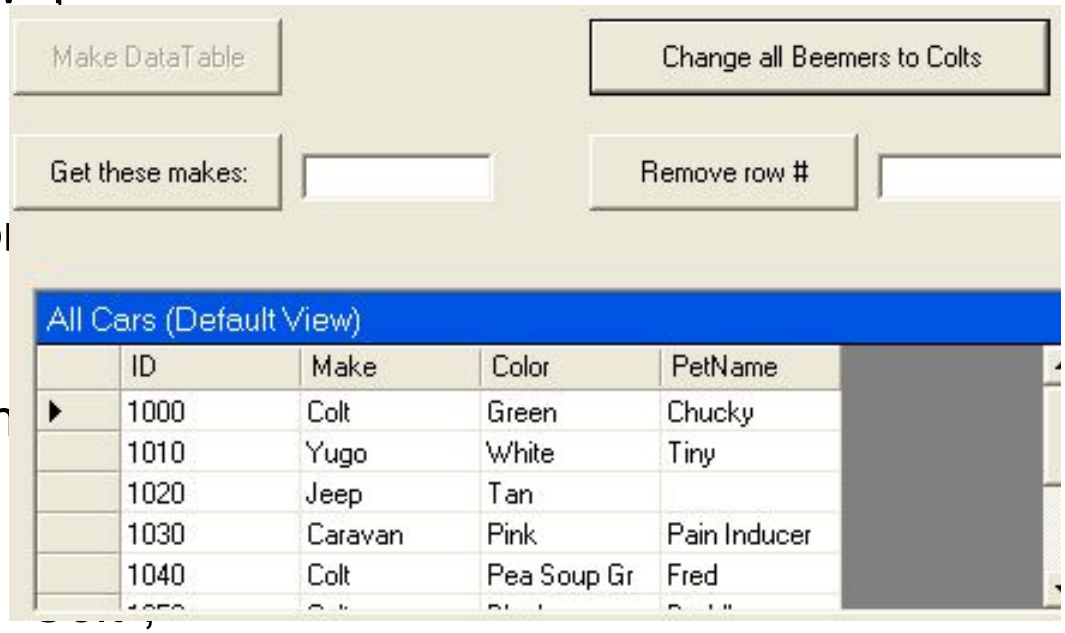
Применение фильтров и порядка сортировки

```
DataRow[] properIDs;
string newFilterStr = "ID > '1030'";
properIDs = inventoryTable.Select(newFilterStr,”PetName”);
string strIDs = null;

for(int i = 0; i < properIDs.Length; i++)
{
    DataRow temp = properIDs[i];
    strIDs += temp["PetName"].ToString(
       + " is ID " + temp["ID"] + "\n";
}

MessageBox.Show(strIDs, "Pet names of cars where ID > 1030");
```



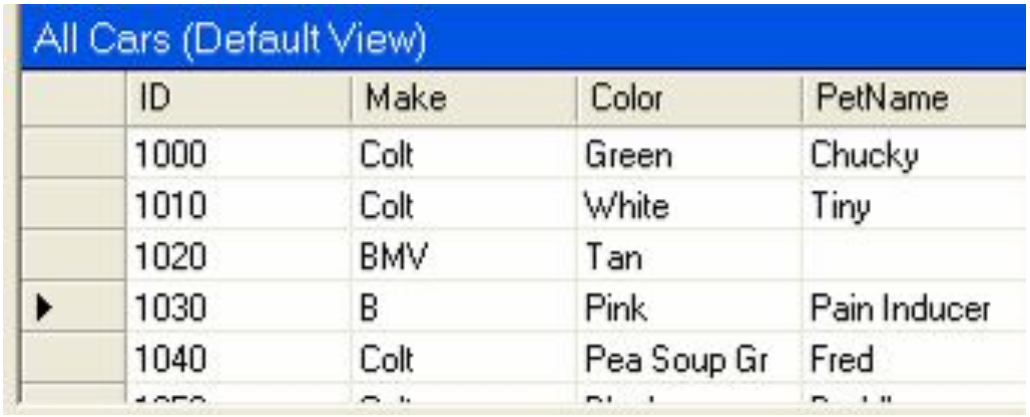
Внесение изменений в строки

```
protected void btnChange_Click (object sender, System.EventArgs e)
{
    // Создаем фильтр для отбора строк
    string filterStr = "Make='BMW'";
    string strMake = null;

    // Получаем строки
    DataRow[] makes =   inventor

    //  Изменяем их
    for(int i = 0; i < makes.Length
    {
      DataRow temp = makes[i];
      strMake += temp["Make"] =
      makes[i] = temp;
    }
}
```



Элемент управления DataGrid

- DataRow.BeginEdit();
- DataRow.EndEdit();
- DataRow.CancelEdit();



Тип DataView

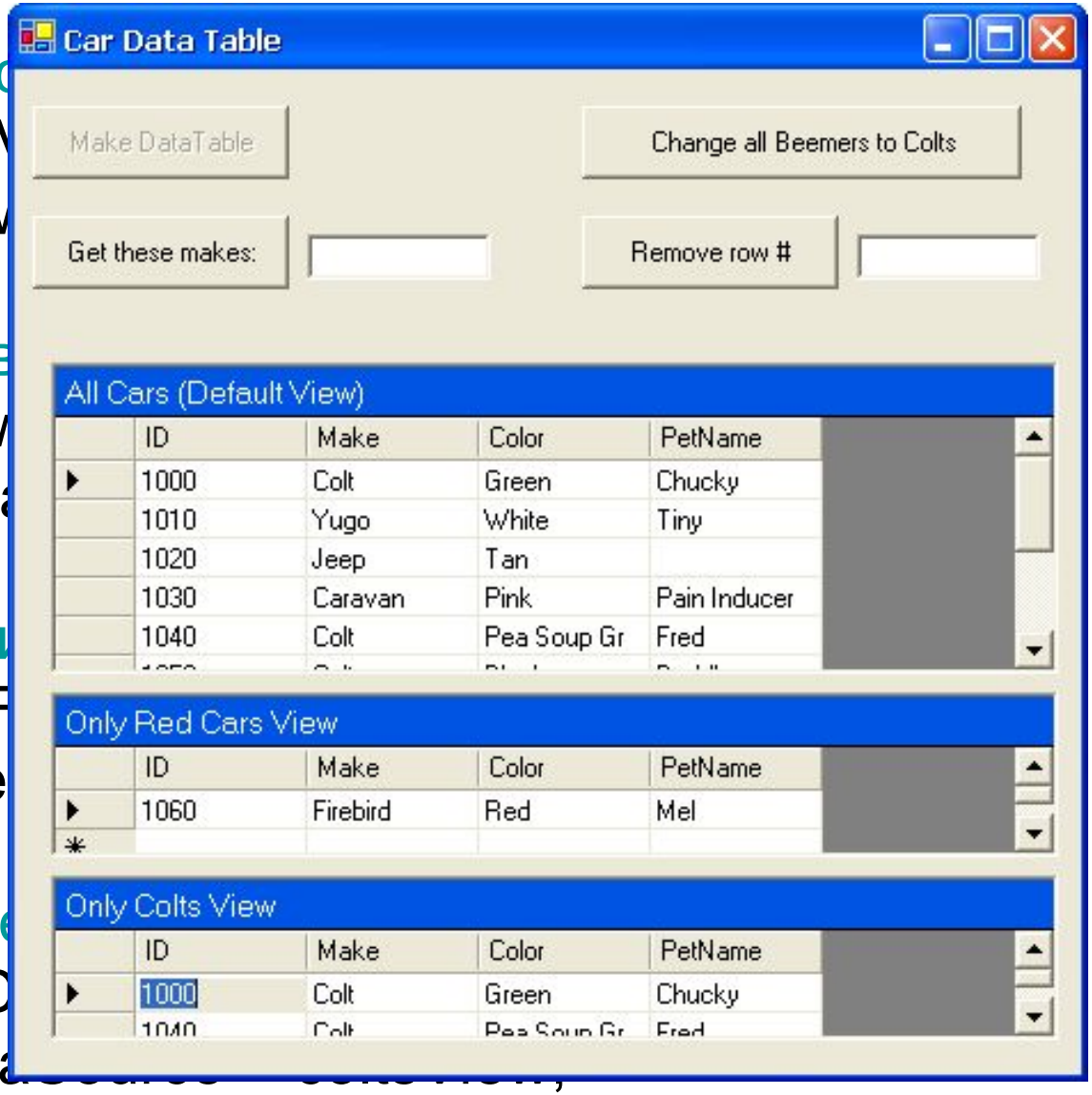
- Специальным образом настроенное отображение данных из таблицы
- Для одной и той же таблицы можно создать неограниченное число представлений

Члены класса DataView

AddNew()	Добавляет новую строку через DataView
AllowDelete AllowEdit AllowNew	Позволяют настроить возможности проведения через DataView соответствующих операций
Delete()	Удаление строки по порядковому номеру
RowFilter	Получить/установить выражение для фильтрации строк
Sort	Настройка сортировки строк в DataView
Table	Получить/указать базовую таблицу для DataTable

Тип DataView

```
// Переменные класс
    DataView redCarsV
    DataView coltsView
. . . . . .
// Определяем базов
    redCarsView = new
    coltsView = new Da

// Конфигурируем фи
    redCarsView.RowF
    coltsView.RowFilte

// Привязываем к эле
    RedCarViewGrid.D
    ColtsViewGrid.Data
```



Класс DataSet

- Создаваемый в ОП набор таблиц, связанных между собой отношениями и снабженными средствами проверки целостности данных
- Свойство Tables – набор таблиц (DataTableCollection)
- Свойство Relations – отношения между таблицами (DataRelationCollection)

Свойства DataSet

CaseSensitive	Учитывать ли регистр букв в строковых операциях
DataSetName	Установить/получить имя данного объекта (обычно в конструкторе)
DefaultViewManager	Представление по умолчанию для отображения данных
EnforceConstraints	Отключить/включить проверку соответствия ограничениям
HasErrors	Проверка наличия ошибок в DataSet
Relations	Коллекция отношений между таблицами
Tables	Коллекция таблиц

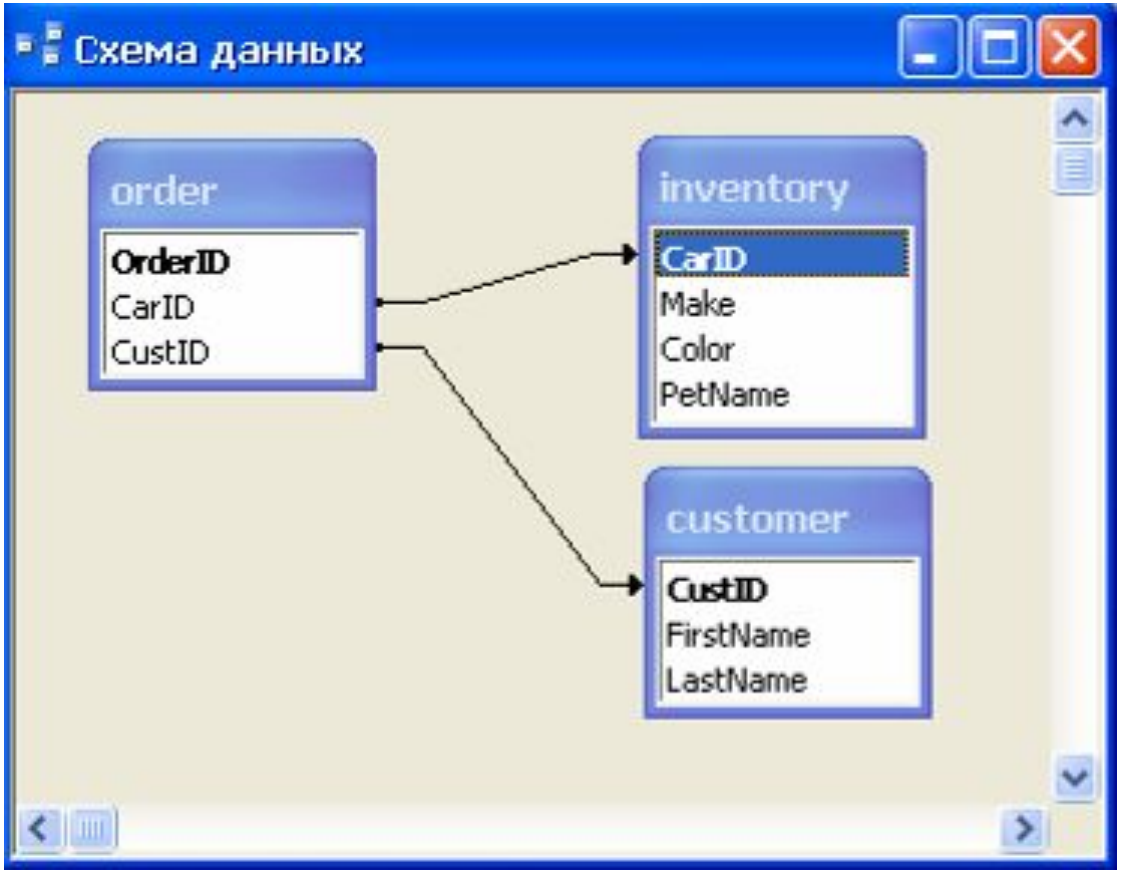
Методы DataSet

AcceptChanges()	Сохранить все изменения, произведенные с момента последнего вызова данного метода
Clear()	Полная очистка DataSet – удаляются все строки из всех таблиц
Clone()	Клонирует структуру DataSet (структуру таблиц и отношений)
Copy()	Копирует DataSet (вместе с данными)
GetChanges()	Возвращает копию DataSet, которая содержит все изменения с момента последнего вызова AcceptChanges
GetChildRelations()	Коллекция подчиненных отношений

Методы DataSet

GetParentRelations()	Коллекция родительских отношений
HasChanges()	Получение информации об изменениях
Merge()	Слияние различных объектов DataSet
ReadXML() ReadXMLSchema()	Считывание данных в формате XML в DataSet из потока
RejectChanges()	Отменить все изменения, внесенные в DataSet с момента создания или вызова AcceptChanges
WriteXML() WriteXMLSchema()	Запись данных в формате XML в DataSet в поток

Создание объекта DataSet



Создание объекта DataSet

```
// Inventory DataTable.
private DataTable inventoryTable = new DataTable("Inventory");

// Customers DataTable.
private DataTable customersTable =
                        new DataTable("Customers");

// Orders DataTable.
private DataTable ordersTable = new DataTable("Orders");

// Our DataSet
private DataSet carsDataSet = new DataSet("CarDataSet");
```

Создадим два простых класса

```
public class Car
{
    // Переменные мы объявляем как public исключительно для упрощения доступа  к ним
    public string petName, make, color;
    public Car (string petName, string make, string Color)
    {
      this.petName = petName;
      this.color = color;
      this.make = make;
    }
}

public class Customer
{
    public Customer (string fName, string lName, int currentOrder)
    {
      this.firstName = fName;
      this.lastName = lName;
      this.currCarOrder = currentOrder;
    }
    public string firstName, lastName;
    public int currCarOrder;
}
```

Списки автомобилей и заказчиков

```
private ArrayList arTheCars, arTheCustomers;

public mainForm()
{
    // Заполняем массив автомобилей соответствующими объектами
    arTheCars = new ArrayList();
    arTheCars.Add(new Car("Chucky", "BMW", "Green"));
    ...
    // Производим ту же операцию с массивом заказчиков
    arTheCustomers = new ArrayList();
    arTheCustomers.Add(new Customer("Dave", "Brenner", 1020));
    ...
    // Создаем объекты DataTable (самым обычным способом, рассмотренным в предыдущих  разделах)
    MakeInventoryTable();
    MakeCustomerTable();
    MakeOrderTable();

    // Создаем объект DataRelation (об этом чуть позже)
    BuildTableRelationship();

    // Привязываем две таблицы к элементам управления DataGrid (первый параметр —
    // DataSet, второй — таблица в DataSet)
    CarDataGrid.SetDataBinding(carsDataSet, "Inventory");
    CustomerDataGrid.SetDataBinding(carsDataSet, "Customers");
}
```

Добавляем таблицу в DataSet

```
private void MakeOrderTable()
{
...
// Добавляем таблицу в DataSet
carsDataSet.Tables.Add(customerstable);

// Создаем столбцы OrderID, CustID и CarID и добавляем их в таблицу
      ...
// Назначаем столбец CarID первичным ключом
      ...
// Добавляем несколько заказов
for(int i = 0; i < arTheCustomers.Count; i++)
    {
      DataRow newRow;
      newRow = ordersTable.NewRow();
      Customer c = (Customer)arTheCustomers[i];
      newRow["CustID"] = i;
      newRow["CarID"] = c.currCarOrder;
      carsDataSet.Tables["Orders"].Rows.Add(newRow);
    }
}
```

```
carsDataSet.Tables.Add(ordersTable);
carsDataSet.Tables.Add(inventoryTable);
```

Моделируем отношения между таблицами

```
private void BuildTableRelationShip()
{
        // Создаем объект DataRelation
        DataRelation dr = new DataRelation("CustomerOrder",
        // Родительская таблица
        carsDataSet.Tables["Customers"].Columns["CustID"],
        // Подчиненная таблица
        carsDataSet.Tables["Orders"].Columns["CustID"];

        // Добавляем объект DataRelation в DataSet
        carsDataSet.Relations.Add(dr);

        // Создаем еще один объект DataRelation
        dr = new DataRelation("InventoryOrder",
        // Родительская таблица
        carsDataSet.Tables["Inventory"].Columns["CarID"],
        // Подчиненная таблица
        carsDataSet.Tables["Orders"].Columns["CarID"];

        // Добавляем и этот объект DataRelation в DataSet
        carsDataSet.Relations.Add(dr);
}
```

Свойства типа DataRelation

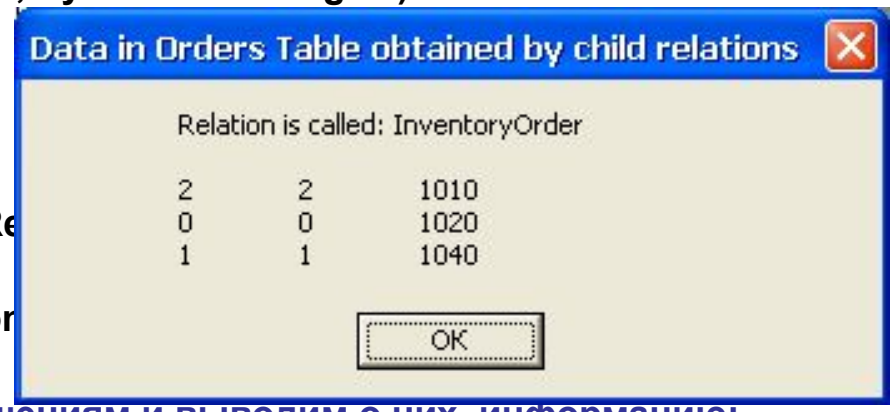
ChildColumns ChildConstraint ChildTable	Получение информации о подчиненной таблице, участвующей в отношении, а также ссылку на эту таблицу
DataSet	Ссылка на объект DataSet, к которому принадлежит данное соотношение
ParentColumns ParentConstraint ParentTable	Получение информации о родительской таблице, участвующей в отношении, а также ссылку на эту таблицу
RelationName	Получить/задать имя для данного отношения

Переход между таблицами, участвующими в отношении

```
string strInfo = "";
DataRow drCust = null;
DataRow[ ] drsOrder = null;

int theCust = int.Parse(this.txtCustID.Text); // Получаем CustID из

// Теперь, основываясь на этом CustID, получаем всю строку и
drCust = carsDataSet.Tables["Customers"].Rows[theCust];
strInfo += "Cust #" + drCust["CustID"].ToString() + "\n";

// Теперь производим переход от таблицы Customers в таблиц
drsOrder = drCust.GetChildRows(carsDataSet.Relations["Custom

// Получаем имя заказчика
foreach(DataRow r in drsOrder) strInfo += "Order Number: " + r["OrderID"] + "\n";

// Теперь переходим от таблицы Orders к таблице Inventory
DataRow[ ] drsInv =     drsOrder[0].GetParentRows(carsDataSet.Relations["InventoryOrder"]);

// Получаем информацию об автомобилях
foreach(DataRow r in drsInv)
{ strInfo += "Make: " + r["Make"] + "\n";
  strInfo += "Color: " + r["Color"] + "\n";
  strInfo += "PetName: " + r["PetName"] + "\n";
}

MessageBox.Show(strInfo, "Info based on cust ID");
```

Получаем информацию о таблицах, подчиненных по отношению к таблице

```
protected void btnGetChildRels_Click (object sender, System.EventArgs e)
{
    DataRelationCollection relCol;
    DataRow[ ] arrRows;
    string info = "";
    relCol = carsDataSet.Tables["inventory"].ChildRe

    info += "\tRelation is called: " + relCol[0].Relation

    // При помощи цикла проходим по всем отношениям и выводим о них  информацию:
    foreach(DataRelation dr in relCol)
    {
      foreach(DataRow r in inventoryTable.Rows)
      {
            arrRows = r.GetChildRows(dr);

            // Выводим значения каждого столбца в строке
            for (int i = 0; i < arrRows.Length; i++)
            {
                foreach(DataColumn dc in arrRows[i].Table.Columns)
                        info += "\t" + arrRows[i][dc];
                info += "\n";
            }
            MessageBox.Show(info, "Data in Orders Table obtained by child relations");
      }
}
```



Запись объектов DataSet в формате XML

```
protected void btnToXML_Click (object sender, System.EventArgs e)
    {
      carsDataSet.WriteXml("cars.xml");
      MessageBox.Show("Wrote CarDataSet to XML file in app directory");
      btnReadXML.Enabled = true;
    }
```

```
<?xml version="1.0" standalone="yes" ?>
- <CarDataSet>
  - <Inventory>
      <CarID>1000</CarID>
      <Make>BMW</Make>
      <Color>Green</Color>
      <PetName>Chucky</PetName>
    </Inventory>
  - <Inventory>
      <CarID>1010</CarID>
      <Make>Yugo</Make>
      <Color>White</Color>
      <PetName>Tiny</PetName>
    </Inventory>
  - <Inventory>
      <CarID>1020</CarID>
      <Make>Jeep</Make>
      <Color>Tan</Color>
      <PetName />
    </Inventory>
  - <Inventory>
```

Чтение объектов DataSet в формате XML

```
protected void btnReadXML_Click (object sender, System.EventArgs e)
{
    // Очищаем и удаляем имеющийся объект DataSet
    carsDataSet.Clear();
    carsDataSet.Dispose();
    MessageBox.Show("Just cleared data set...");
    carsDataSet = new DataSet("CarDataSet");

    carsDataSet.ReadXml("Cars.xml");

    MessageBox.Show("Reconstructed data set from XML file...");
    btnReadXML.Enabled = false;

    // Настраиваем привязки к элементам управления DataGrid
    CarDataGrid.SetDataBinding(carsDataSet, "Inventory");
    CustomerDataGrid.SetDataBinding(carsDataSet, "Customers");
}
```

```
using System.Xml;
```

Управляемые провайдеры ADO.NET

- Шлюз к хранилищу данных
  - OLE DB
    - SQL Server
    - MS Access
    - Oracle
  - SQL (MS SQL Server)

Пространство имен System.Data.OleDb

OleDbCommand	Запрос SQL к источнику данных
OleDbConnection	Открытое соединение с БД
OleDbDataAdapter	Соединение и набор команд для заполнения DataSet
OleDbDataReader	Считывание потока данных
OleDbErrorCollection OleDbError OleDbException	Ошибки и предупреждения. Исключение.
OleDbParameterCollection OleDbParameter	Параметры для хранимых процедур

Установление соединения с OleDbConnection

```
OleDbConnection cn = new OleDbConnection();

// MS Access
 cn.ConnectionString =
    "Provider=Microsoft.JET.OLEDB.4.0;" +
          @"data source = D:\Access DB\cars.mdb";

// SQL Server
cn.ConnectionString = "Provider=SQLOLEDB.1;" +
              "Integrated Security=SSPI;" +
              "Persist Security Info=False;" +
              "Initial Catalog=Cars;" +
              "Data Source=(local);";
```

Члены класса OleDbConnection

BeginTransaction() CommitTransaction() RollbackTransaction()	Используются для того, чтобы программным образом начать транзакцию ее или отменить
Close()	Закрывает соединение с источником данных
ConnectionString	Позволяет настроить строку подключения при установлении соединения
ConnectionTimeOut	Время тайм-аута при установке соединения
Database	Получить/установить название текущей БД
DataSource	Получить/установить имя источника данных
Open()	Открывает соединение с БД
Provider	Получить/установить имя провайдера
State	Информация о текущем состоянии соединения

Объектно-ориентированное построение команды SQL
Класс OleDbCommand

Cancel()	Прекращает выполнение команды
CommandText	Текст вопроса на языке SQL
CommandTimeOut	Тайм-аут выполнения команды (30 сек)
CommandType	Определить, как именно будет интерпретирован текст запроса, заданный через свойство
Connection	Ссылка на объект OleDbConnection
ExecuteReader()	Возвращает объект OleDbDataReader
Parameters	Коллекция параметров OleDbParameterCollection
Prepare()	Готовит команду к выполнению (компиляция на источнике данных)

Пример построения SQL-запроса

```
// Первый вариант
string strSQL = "SELECT Make FROM Inventory WHERE Color='Red'";
OleDbCommand myCommand = new OleDbCommand(strSQL, cn);
```

```
// Второй вариант
string strSQL = "SELECT Make FROM Inventory WHERE Color='Red'";
OleDbCommand myCommand = new OleDbCommand();
myCommand.Connection = cn;
myCommand.CommandText = strSQL;
```

Обращение к хранимой процедуре

```
// Открываем соединение с источником данных
OleDbConnection cn = new OleDbConnection();
cn.ConnectionString =
      "Provider=SQLOLEDB.1;" +
      "Integrated security=SSPI;" +
      "Persist Security Info=False;" +
      "Initial Catalog=Cars;" +
      "Data Source=BIGMANU;";
cn.Open();

// А теперь создаем и настраиваем объект OleDbCommand для
//  хранимой процедуры:
OleDbCommand myCommand = new OleDbCommand("GetPetName", cn);
myCommand.CommandType = CommandType.StoredProcedure;
```

Передача параметров хранимым процедурам

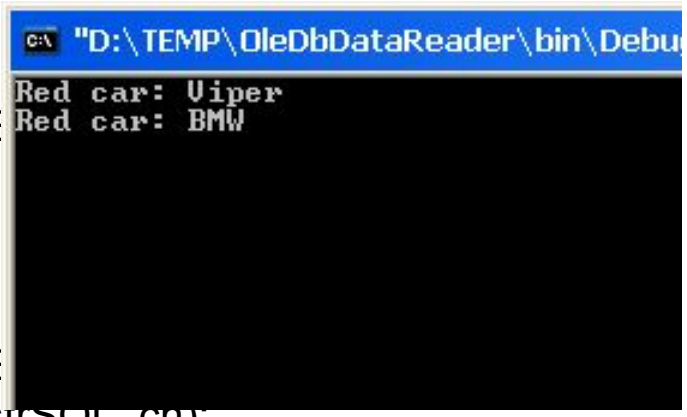
```
// Создаем объект для наших параметров
OleDbparameter theParam = new OleDbParameter();

// Параметр для передачи хранимой процедуре
theParam.ParameterName = "@carID";
theParam.DbType = OleDbType,Integer;
theParam.Direction = ParameterDirecton.Input;
theParam.Value = 1; // CarID = 1
myCommand.Parameters.Add(theParam);

// Параметр для возврата значений из хранимой процедуры
theParam = new OleDbParameter();
theParam.ParameterName = "@PetName";
theParam.DbType = OleDbType.Char;
theParam.Size = 20;
theParam.Direction = ParameterDirection.Output;
myCommand.Parameters.Add(theParam);
```

Вызов хранимой процедуры

```
// Создаем объект для наших параметров
myCommand.ExecuteNonQuery();

// Выводим результат
Console.WriteLine
    (“Car ID: “+myCommand.Parameters[“@CarID”].Value);
Console.WriteLine
    (“PetName: “+myCommand.Parameters[“@petName”].Value);
```

Работа с OleDbDataReader

```
// Первый шаг: устанавливаем соединение
OleDbConnection cn = new OleDbConnection();
cn.ConnectionString = @"Provider=Microsoft.JET.OLEDB
      @"data source = D:\Access DB\cars.mdb";
cn.Open();

// Второй шаг: создаем команду SQL
string strSQL = "SELECT Make FROM Inventory WHERE
OleDbCommand myCommand = new OleDbCommand(strSQL, cn);

// Третий шаг: получаем объект OleDbDataReader при помощи метода
ExecuteReader()
OleDbDataReader myDataReader;
myDataReader = myCommand.ExecuteReader();

// Четвертый шаг: проходим циклом по всем возращаемым данным
while (myDataReader.Read())
{     Console.WriteLine("Red car: " + myDataReader["Make"].ToString());
}

myDataReader.Close();
cn.Close();
```



Тип OleDbDataAdapter

- Извлечь информацию из БД и заполнить ею объект DataTable в DataSet при помощи метода OleDbDataAdapter.Fill

- public int Fill(DataSet DS, string tableName);

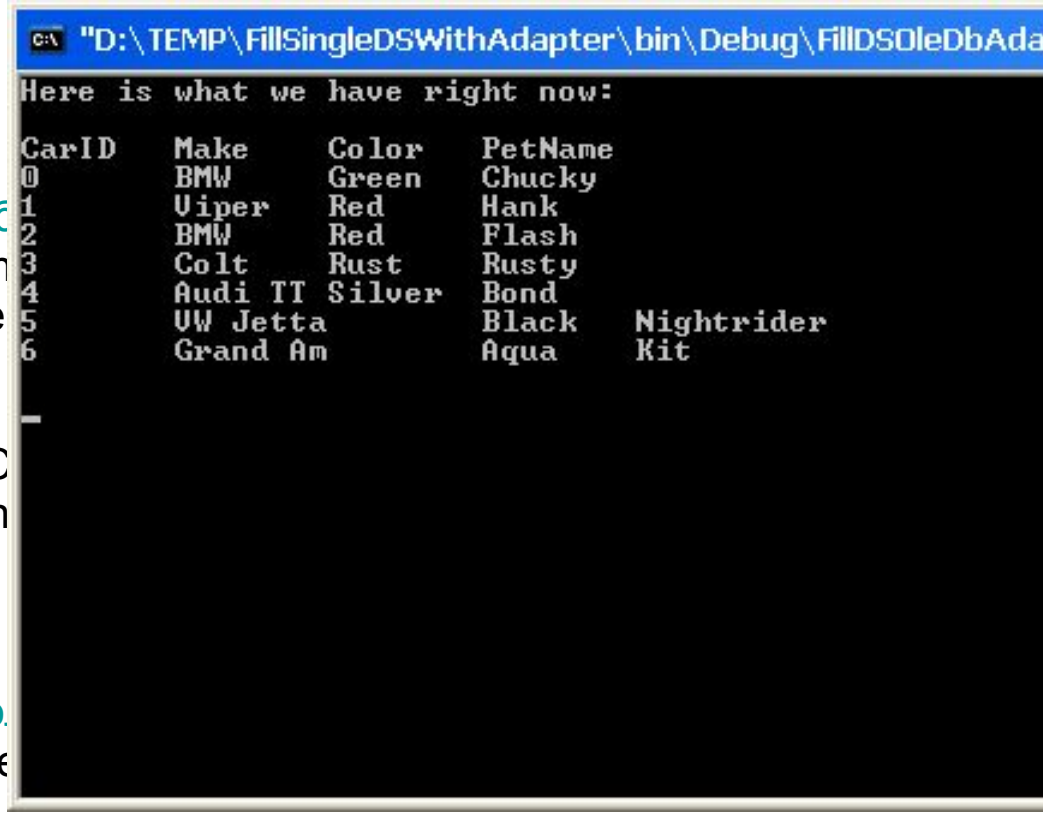
- public int Fill(DataSet DS, string tableName, int startRecord, int maxRecord);

Заполнение данными объекта DataSet при помощи OleDbDataReader

```
// Шаг 1: открываем соединение с базой данных Cars
OleDbConnection cn = new OleDbConnection();
cn.ConnectionString = "Provider=Microsoft.JET.OLEDB.4.0; data source =Cars.mdb";
cn.Open();

// Шаг 2: Создаем OleDbDataAdapter при помощи команды SELECT
string sqlSELECT = "SELECT * FROM Inventory";
OleDbDataAdapter dAdapt = new OleDbDataAdapter(sqlSELECT, cn);

// Шаг 3: Создаем и заполняем объект DataSet, а потом закрываем соединение
DataSet myDS = new DataSet("CarsDataSet");
try         {
                dAdapt.Fill(myDS, "Inventory");
            }
catch(Exception ex) {  Console.WriteLine(ex.Message);
            }
finally             {  cn.Close();
            }

// Вспомогательная функция для вывода содержимого таблицы
PrintTable(myDS);
```

Примерный вид функции PrintTable

```
public static void PrintTable(DataSet ds)
{
    // Получаем таблицу Inventory из об
    ConsoleWriteLine("Here is what we h
    DataTable invTable = ds.Tables["Inve

    // Выводим имена столбцов
    for(int curCol = 0; curCol < Invtable.C
    { Console.WriteLine(invTable.Column
    }
    Console.WriteLine();

    // Выводим значения из каждого по
    for(int curRow = 0; curRow < invTable
    {
        for(int curCol = 0; curCol < Invtable.Columns.Count; curCol++)
      Console.Write(InvTable.Rows[curRow][curCol].ToString().Trim() +"\t");
        Console.WriteLine()
    }
}
```



Пространство имен System.Data.SqlClient

SqlCommand	Запрос SQL к источнику данных
SqlConnection	Открытое соединение с БД
SqlDataAdapter	Соединение и набор команд для заполнения DataSet
SqlDataReader	Считывание потока данных
SqlErrorCollection SqlError SqlException	Ошибки и предупреждения. Исключение.
SqlParameterCollection SqlParameter	Параметры для хранимых процедур

Удаление записи при помощи SqlDataAdapter

```
// Шаг 1: Создаем соединение и объект SqlDataAdapter  (с командой SELECT)
SqlConnection cn = new SqlConnection
    ("server=(local);uid=sa;pwd=;database=Cars");

SqlDataAdapter dAdapt = new SqlDataAdapter("Select * from Inventory", cn);

// Шаг 2: Если в таблице уже присутствует запись с номером,
// совпадающим  с номером вставляемой нами записи, удаляем ее
cn.Open();
SqlCommand killCmd = new SqlCommand
        ("Delete from Inventory where CarID = '1111'", cn);
killCmd.ExecuteNonQuery();
cn.Close();
```

Подготовка представления команды INSERT

```
// Шаг 3: создаем команду INSERT
dAdapt.InsertCommand = new SqlCommand("INSERT INTO Inventory" +
"(CarID, Make, Color, PetName) VALUES" + "(@CarID, @Make, @Color, @PetName)", cn)";

// Шаг 4: Начинаем работу по созданию параметров для каждого столбца  в таблице Inventory
SqlParameter workParam = null;

// Параметр для столбца CarID
workParam = d.Adapt.InsertCommand.Parameters.Add(new SqlParameter("@CarID", SqlDbType.Int));
workParam.SourceColumn = "CarID";
workParam.SourceVersion = DataRowVersion.Current;

// Параметр для столбца Make
workParam = d.Adapt.InsertCommand.Parameters.Add(new SqlParameter("@Make", SqlDbType.VarChar));
workParam.SourceColumn = "Make";
workParam.SourceVersion = DataRowVersion.Current;

// Параметр для столбца Color
workParam = d.Adapt.InsertCommand.Parameters.Add(new SqlParameter("@Color", SqlDbType.VarChar));
workParam.SourceColumn = "Color";
workParam.SourceVersion = DataRowVersion.Current;

// Параметр для столбца PetName
workParam = d.Adapt.InsertCommand.Parameters.Add(new SqlParameter("@PetName",SqlDbType.VarChar));
workParam.SourceColumn = "PetName";
workParam.SourceVersion = DataRowVersion.Current;
```

Добавление новой записи

```
// Шаг 5: Создаем объект DataSet и заполняем его данными из базы данных / на источнике:
DataSet myDS = new DataSet();
dAdapt.Fill(myDS, "Inventory");
PrintTable(myDS);

// Шаг 6: добавляем новую запись в таблицу в DataSet
DataRow newRow = myDS.Tables["Inventory"].NewRow();
newRow["CarID"] = 1111;
newRow["Make"] = "SlugBug";
newRow["Color"] = "Pink";
newRow["PetName"] = "Cranky";
myDS.Tables["Inventory"].Rows.Add(newRow);

 // Шаг 7: Передаем изменения на источник данных и проверяем это
try {  dAdapt.Update(myDS, "Inventory");
       myDS.Dispose();
       myDS = new DataSet();
       dAdapt.Fill(myDS, "Inventory");
       PrintTable(myDS);
     }
catch(Exception e)   {  Console.Write(e.ToString());
              }
```

Изменение записей при помощи SqlDataAdapter

```
public static void Main()
{
    // Шаг 1: Создаем объект SqlDataAdapter и открываем соединение (аналогично
    // предыдущему примеру, поэтому эту часть кода опускаем)
      ...

    // Шаг 2: Создаем команду UPADTE
    dAdapt.UpdateCommand = new SqlCommand("UPDATE Inventory SET Make=@Make,
            Color=@Color, PetName=@PetName WHERE CarID = @CarID", cn);

    // Шаг 3: Создаем объекты параметров для каждого столбца в таблице Inventory.
    // Все так же, как и в предыдущем примере, только мы помещаем эти параметры
    // в коллекцию ParameterCollection для UpdateCommand. Например:
    SqlParameter workParam = null;
    workParam = dAdapt.UpdateCommand.Parameters.Add(new SqlParameter("@CarID",
                                            SqlDbType.Int));
    workParam.SourceColumn = "CarID";
    workParam.SourceVersion = DataRowVersion.Current;

    // Делаем то же самое для столбцов PetName, Make и Color
    ...
```

Изменение записей при помощи SqlDataAdapter

```
// Шаг 4: Заполняем данными объект DataSet
DataSet myDS = new DataSet();
dAdapt.Fill(myDS, "Inventory");
PrintTable(myDS);

// Шаг 5: Меняем значения столбцов во второй строке таблицы на 'FooFoo'
DataRow changeRow = myDS.Tables["Inventory"].Row[1];
changeRow["Make"] = "FooFoo";
changeRow["Color"] = "FooFoo";
changeRow["PetName"] = "FooFoo";

// Шаг 6: Сохраняем данные в базе данных и выводим значения на консоль
try    {      dAdapt.Update(myDS, "Inventory");
              myDS.Dispose();
              myDS = new DataSet();
              dAdapt.Fill(myDS, "Inventory");
              PrintTable(myDS);
       }
catch(Excception e) { Console.Write(e.ToString()); }
}
```

Автоматическое создание команд SQL

- Утомительно заполнение свойств
  - InsertCommand
  - UpdateCommand
  - DeleteCommand
- SqlCommandBuilder
  - Автоматически настраивает свойства по первоначальной команде SELECT
  - Должен быть определен первичный ключ

Автоматическое создание команд SQL

```
public class mainForm : System.Windows.Forms.Form
{
        private SqlConnection cn = new
                        SqlConnection("server=(local);uid=sa;pwd=;database=Cars");

        private SqlDataAdapter dAdapt;

        private SqlCommandbuilder invBuilder;
        private DataSet myDS = new DataSet();

        private System.Windows.Forms.DataGrid dataGrid1;
        private System.Windows.Forms.Button btnUpdateData;
        private System.ComponentMode1.Container components;

public mainForm();
{
        InitializeComponents();

        // Создаем исходную команду SELECT
        dAdapt = new SqlDataAdapter("SELECT * from Inventory", cn);

        // А сейчас команды INSERT, UPDATE и DELETE будут сгенерированы  автоматически
        invBuider = new SqlCommandBuilder(dAdapt);

        // Заполняем DataSet и привязываем к нему DataGrid
        dAdapt.Fill(myDS, "Inventory");
        dataGrid1.DataSource = myDS.Tables["Inventory"].DefaultView;
}
```

Код обновления БД

```
private void btnUpdateData_Click(object sender, System.EventArgs e)
{
  try
  {
        dataGrid1.Refresh();
        dAdapt.Update(myDS, "Inventory");
  }
  catch(Exception ex)
  {
            MessageBox.Show(ex.ToString());
  }
}
```

Заключение

- ADO.NET
  - System.Data
  - System.Data.SqlClient
  - System.Data.OleDb
- DataSet – представление в ОП любого количества таблиц и отношений между ними
- OleDbDataAdapter, SqlDataAdapter – связывают DataSet с истониками данных