Операции с данными. Практический пример

Последнее обновление: 31.10.2015

Создадим полноценное приложение, которое будет выполнять все эти операции. Итак, создадим новый проект по типу Windows Forms. Новое приложение будет работать с базой данных футболистов. В качестве подхода взаимодействия с БД выберем Code First.

Вначале добавим в проект новый класс, который описывает модель футболистов:

```
class Player
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Position { get; set; }
    public int Age { get; set; }
}
```

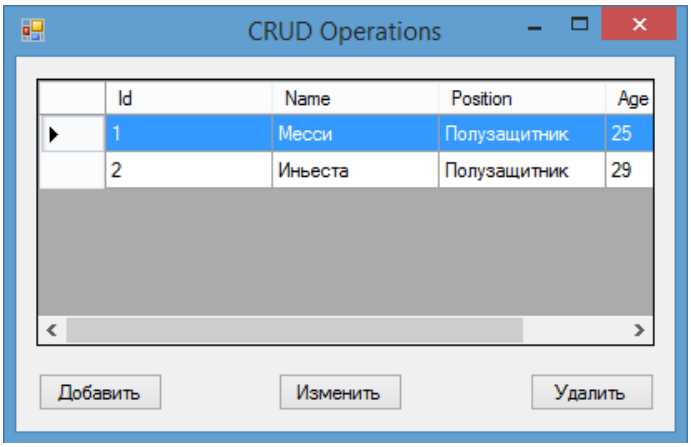
Тут всего четыре свойства: id, имя, позиция на поле и возраст. Также добавим в проект через NuGet пакет Entity Framework и новый класс контекста данных:

```
using System.Data.Entity;

class SoccerContext : DbContext
{
    public SoccerContext()
        :base("DefaultConnection")
    { }

    public DbSet<Player> Players { get; set; }
}
```

В файл конфигурации *app.config* после секции `configSections` добавим узел **connectionStrings**, в котором определим строку подключения DefaultConnection:

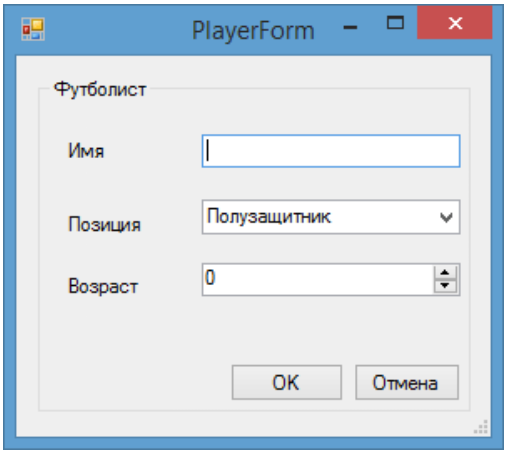
```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <connectionStrings>
   <add name="DefaultConnection" connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=Players;Integrated
Security=True"
       providerName="System.Data.SqlClient"/>
 </connectionStrings>
  <!--остальное содержимое-->
</configuration>
```

Теперь визуальная часть. По умолчанию в проекте уже есть форма Form1. Добавим на нее элемент DataGridView, который будет отображать все данные из БД, а также три кнопки на каждое действие - добавление, редактирование, удаление, чтобы в итоге форма выглядела так:



У элемента DataGridView установим в окне свойств для свойства **AllowUserToAddRows** значение `False`, а для свойства **SelectionMode** значение `FullRowSelect`, чтобы можно было выделять всю строку.

Это основная форма, но добавление и редактирование объектов у нас будет происходить на вспомогательной форме. Итак, добавим в проект новую форму, которую назовем *PlayerForm*. Она будет иметь следующий вид:



Здесь у нас текстовое поле для ввода имени, далее выпадающий список ComboBox, в который мы через свойство Items добавляем четыре позиции. И последнее поле - NumericUpDown для ввода чисел для указания возраста. У всех этих трех полей установим свойство **Modifiers** равным **Protected Internal**, чтобы эти поля были доступны из главной формы.

Также есть две кнопки. Для кнопки "OK" в окне свойств для свойства **DialogResult** укажем значение OK, а для кнопки "Отмена" для того же свойства установим значение Cancel.

Никакого кода данная форма не будет содержать. Теперь перейдем к основной форме Form1, которая и будет содержать всю логику. Весь ее код:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Windows.Forms;

namespace CRUD
{
    public partial class Form1 : Form
    {
        SoccerContext db;
        public Form1()
        {
            InitializeComponent();
```

```
            db = new SoccerContext();
            db.Players.Load();

            dataGridView1.DataSource = db.Players.Local.ToBindingList();
        }
        // добавление
        private void button1_Click(object sender, EventArgs e)
        {
            PlayerForm plForm = new PlayerForm();
            DialogResult result = plForm.ShowDialog(this);

            if (result == DialogResult.Cancel)
                return;

            Player player = new Player();
            player.Age = (int)plForm.numericUpDown1.Value;
            player.Name = plForm.textBox1.Text;
            player.Position = plForm.comboBox1.SelectedItem.ToString();

            db.Players.Add(player);
            db.SaveChanges();

            MessageBox.Show("Новый объект добавлен");
        }
        // редактирование
        private void button2_Click(object sender, EventArgs e)
        {
            if(dataGridView1.SelectedRows.Count>0)
            {
                int index = dataGridView1.SelectedRows[0].Index;
                int id=0;
                bool converted = Int32.TryParse(dataGridView1[0, index].Value.ToString(),out id);
                if(converted==false)
                    return;

                Player player = db.Players.Find(id);

                PlayerForm plForm = new PlayerForm();

                plForm.numericUpDown1.Value = player.Age;
                plForm.comboBox1.SelectedItem = player.Position;
                plForm.textBox1.Text = player.Name;

                DialogResult result = plForm.ShowDialog(this);

                if (result == DialogResult.Cancel)
                    return;

                player.Age = (int)plForm.numericUpDown1.Value;
                player.Name = plForm.textBox1.Text;
                player.Position = plForm.comboBox1.SelectedItem.ToString();

                db.SaveChanges();
                                dataGridView1.Refresh(); // обновляем грид
                MessageBox.Show("Объект обновлен");

            }
        }
        // удаление
        private void button3_Click(object sender, EventArgs e)
        {
            if (dataGridView1.SelectedRows.Count > 0)
```

```
            {
                int index = dataGridView1.SelectedRows[0].Index;
                int id = 0;
                bool converted = Int32.TryParse(dataGridView1[0, index].Value.ToString(), out id);
                if (converted == false)
                    return;

                Player player = db.Players.Find(id);
                db.Players.Remove(player);
                db.SaveChanges();

                MessageBox.Show("Объект удален");
            }
        }
    }
}
```

Чтобы получить данные из бд, используется выражение db.Players. Однако нам надо кроме того выполнить привязку к элементу DataGridView и динамически отображать все изменения в случае добавления, редактирования или удаления. Поэтому вначале используется метод `db.Players.Load()`, который загружает данные в объект DbContext, а затем выполняется привязка (`dataGridView1.DataSource = db.Players.Local.ToBindingList()`)

Добавление

При добавлении объекта использует вторая форма:

```
Player player = new Player();
player.Age = (int)plForm.numericUpDown1.Value;
player.Name = plForm.textBox1.Text;
player.Position = plForm.comboBox1.SelectedItem.ToString();

db.Players.Add(player);
db.SaveChanges();
```

Для добавления объекта используется метод **Add**, определенный у класса DbSet. В этот метод передается новый объект, свойства которого формируются из полей второй формы. Метод Add устанавливает значение Added в качестве состояния нового объекта. Поэтому метод db.SaveChanges() сгенерирует выражение INSERT для вставки модели в таблицу.

Редактирование

Редактирование имеет похожую логику. Только вначале мы передаем значения свойств объекта во вторую форму, а после получаем с нее же измененные значения для свойств объекта.

В данном случае контекст данных автоматически отслеживает, что объект был изменен, и при вызове метода `db.SaveChanges()` будет сформировано SQL-выражение UPDATE для данного объекта, которое обновит объект в базе данных.

Удаление

С удалением проще всего: получаем по id нужный объект в бд и передаем его в метод `db.Players.Remove(player)`. Данный метод установит статус объекта в `Deleted`, благодаря чему Entity Framework при выполнении метода db.SaveChanges() сгенерирует SQL-выражение DELETE.

Назад **Содержание** **Вперед**