Автоматизация Code First и EF Power Tools

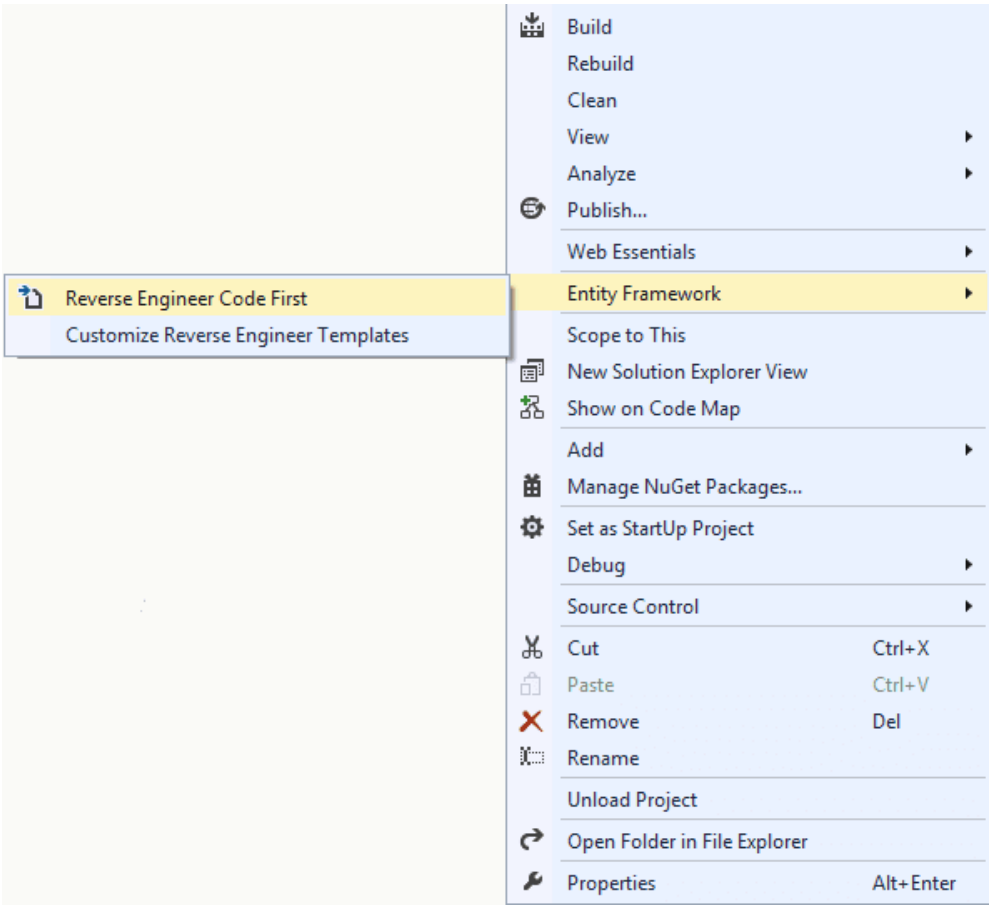
Последнее обновление: 31.10.2015

Кроме вышеописанной функциональности Microsoft предлагает нам полезный инструмент, также призванный автоматизировать данный процесс. Этот инструмент называется **EF Power Tools**.

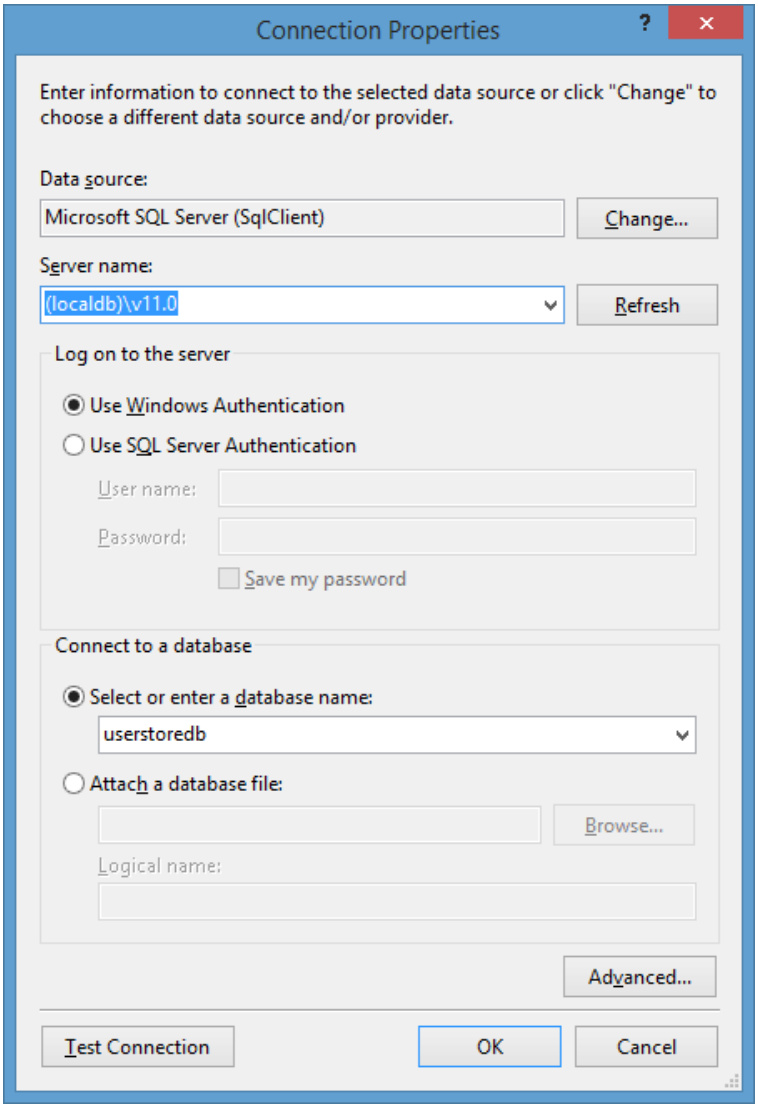
EF Power Tools представляет собой надстройку к Visual Studio. Эту надстройку, а также краткое описание можно найти на странице Entity Framework Power Tools . Для установки достаточно нажать на этой веб-странице на кнопку "Загрузка" и дальше следовать инструкциям.

В то же время есть некоторые ограничения: надстройка Entity Framework Power Tools работает только в полных версиях Visual Studio. В экспресс же версиях не работает.

Итак, если у вас полнофункциональная версия Visual Studio 2012 или 2013, то вы можете нажать в окне Solution Explorer (Обозреватель решений) на проект правой кнопкой мыши и увидеть в контекстном меню пункт **Entity Framework**:

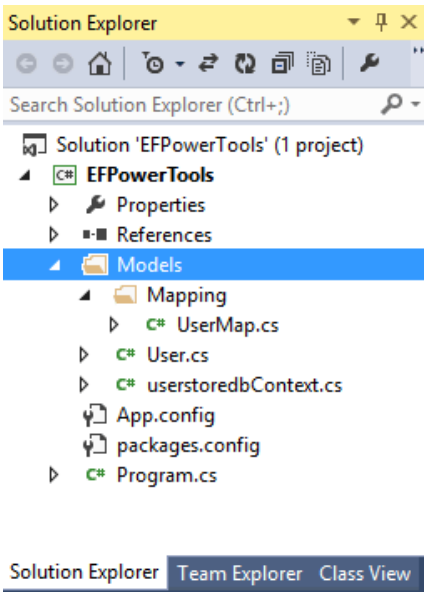


Выберем пункт **Entity Framework –> Reverse Engineer Code First**. Затем откроется окно настройки подключения, где нам надо задать сервер и базу данных, с которой нам надо взаимодействовать:



Я выбрал базу данных, которая была создана в прошлой теме.

Затем нажмем OK. После этого в проект будет добавлена папка Models, в которой будут находиться все созданные классы. По умолчанию для каждой таблицы создается свой класс, и также генерируется класс контекста данных:



Поскольку в моей базе данных была одна таблица Users, то автоматически был создан класс User, который отражает структуру таблицы:

```
using System;
using System.Collections.Generic;
```

```
namespace EFPowerTools.Models
{
    public partial class User
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
    }
}
```

Контекст данных userstoredbContext:

```
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using EFPowerTools.Models.Mapping;

namespace EFPowerTools.Models
{
    public partial class userstoredbContext : DbContext
    {
        static userstoredbContext()
        {
            Database.SetInitializer<userstoredbContext>(null);
        }

        public userstoredbContext()
            : base("Name=userstoredbContext")
        {
        }

        public DbSet<User> Users { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Configurations.Add(new UserMap());
        }
    }
}
```

Он имеет два конструктора. Статический конструктор призван выполнить начальную инициализацию данных. В данном случае он ничего не выполняет.

Стандартный конструктор обращается к конструктору базового класса (то есть класса DbContext) и передает ему название строки подключения (`base("Name=userstoredbContext")`).

Для взаимодействия с таблицей Users класс контекста имеет одноименное свойство `public DbSet<User> Users { get; set; }`

И в методе `OnModelCreating` выполняются действия при создании моделей. В данном случае с помощью класса `UserMap` настраивается конфигурация связей между классами и базой данных.

Этот класс UserMap выполняет сопоставление таблиц и их столбцов с классами и их свойствами:

```
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity.ModelConfiguration;

namespace EFPowerTools.Models.Mapping
{
    public class UserMap : EntityTypeConfiguration<User>
    {
        public UserMap()
        {
            // Primary Key
            this.HasKey(t => t.Id);
```

```
            // Properties
            this.Property(t => t.Name)
                .IsRequired()
                .HasMaxLength(50);

            // Table & Column Mappings
            this.ToTable("Users");
            this.Property(t => t.Id).HasColumnName("Id");
            this.Property(t => t.Name).HasColumnName("Name");
            this.Property(t => t.Age).HasColumnName("Age");
        }
    }
}
```

Остальная работа с базой данных будет происходить также, как и при стандартном подходе Code First. То есть, если нам надо добавить в таблицу новый объект User, то мы пишем:

```
User p = new User { Name = "Имя", Age = 30 };
userstoredbContext.Users.Add(p);
userstoredbContext.SaveChanges();
```

Конечно, в реальности базы данных, как правило, обладают более сложной структурой, и поэтому структура генерируемых классов также будет сложнее класса User. Но на данном примере уже понятно, что мы можем значительно автоматизировать часть работы по созданию классов моделей.

Назад **Содержание** **Вперед**