Модель данных и хранилище

Исходный код проекта

Нам необходим способ работы с базой данных, созданной в предыдущей статье, и ее содержимым внутри создаваемого приложения ASP.NET Framework. Для этого мы планируем применять инфраструктуру **Entity Framework**, которая поддерживает объектно-реляционное отображение. Последние версии Entity Framework включают удобное средство под названием *"сначала код"* (code-first). Идея состоит в том, что мы можем определить классы в модели и затем сгенерировать на их основе базу данных.

Упомянутое средство великолепно подходит для проектов, разрабатываемых с нуля, но такие проекты немногочисленны и встречаются редко. Вместо этого мы собираемся продемонстрировать вариацию подхода "сначала код", при которой необходимые классы модели данных создаются в приложении и затем ассоциируются с существующей базой данных (в качестве которой выступает база данных, созданная в предыдущей статье).

Создание класса модели данных

Мы должны создать класс, который будет представлять строки в таблице базы данных GameStore. Каждая строка таблицы базы данных содержит описание игры в онлайновом магазине, поэтому мы создаем новый файл класса по имени Game.cs в папке Models проекта. Содержимое этого файла приведено в примере ниже:

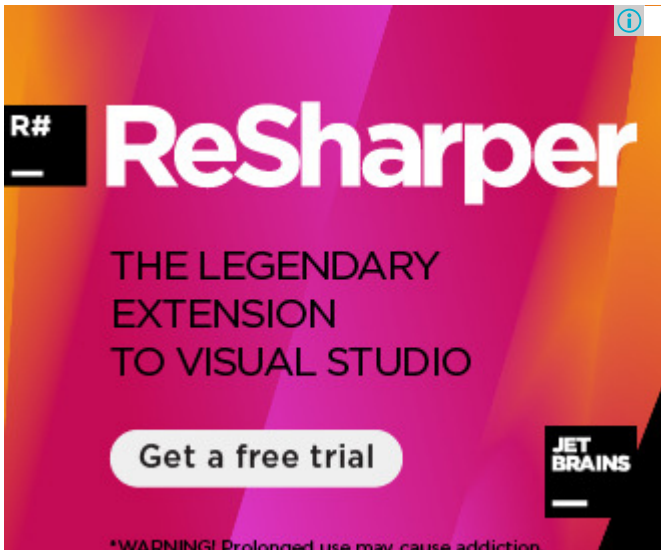
```
namespace GameStore.Models
{
    public class Game
    {
        public int GameId { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public string Category { get; set; }
        public decimal Price { get; set; }
    }
}
```

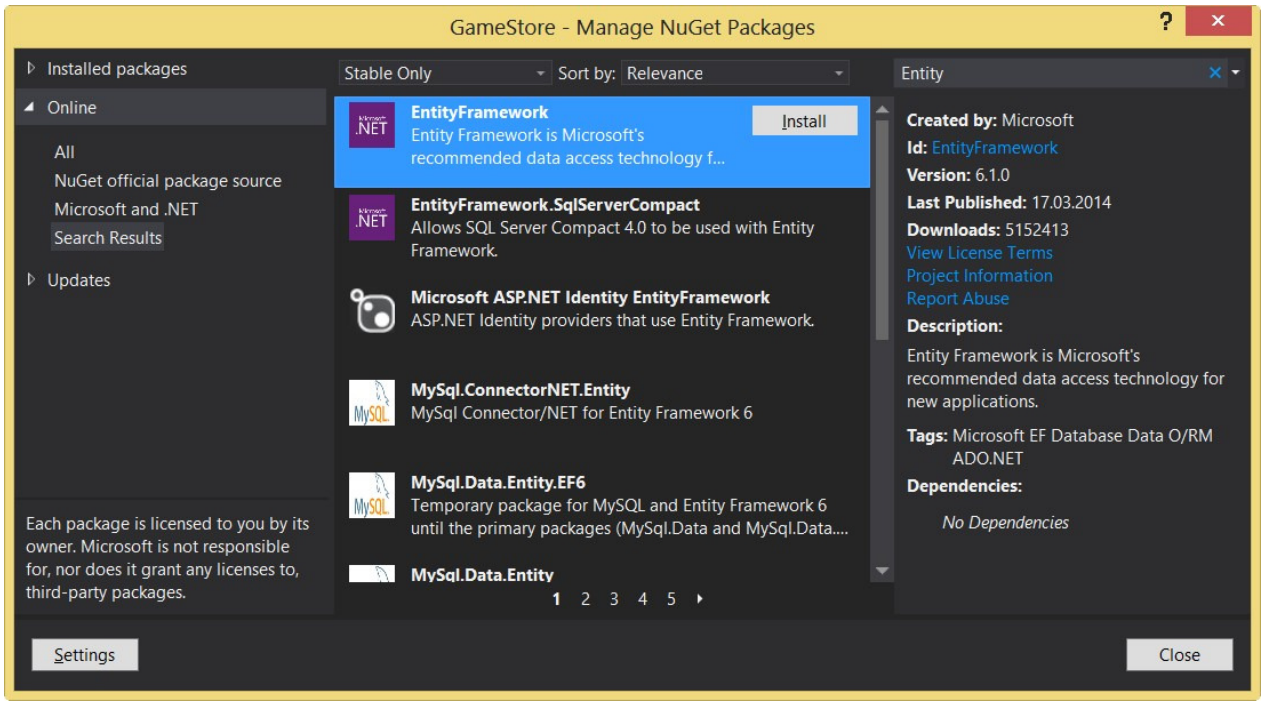
В этом файле определен простой класс Game с автоматически реализуемыми свойствами, которые соответствуют столбцам в созданной ранее таблице базы данных.

Обратите внимание, что класс Game определен в пространстве имен GameStore.Models. Мы предпочитаем включать папки, используемые для организации файлов проекта, в применяемые пространства имен. Тем не менее, многие приложения Web Forms написаны так, что все созданные внутри них классы находятся в единственном пространстве имен. Ни один из подходов не дает какого-либо преимущества, и вы должны избрать для себя тот вариант, который подходит больше. Однако если вы решите использовать единственное пространство имен, то должны корректировать объявление namespace, добавляемое средой Visual Studio в новые файлы классов, когда они создаются внутри папки.

Добавление инфраструктуры Entity Framework

Добавить инфраструктуру Entity Framework в проект GameStore проще всего с помощью диспетчера пакетов NuGet. Выберите пункт Manage NuGet Packages (Управлять пакетами NuGet) в меню Project (Проект) и введите "entity" в поле поиска для категории Online, как показано на рисунке ниже:





Выберите пакет `EntityFramework` (он не содержит пробелов в своем названии). На момент написания этих строк текущей версией `Entity Framework` была `6.0`, но вы можете столкнуться и с более новой версией, если она станет доступной на тот момент. Щелкните на кнопке `Install` (Установить); диспетчер пакетов `NuGet` предложит ознакомиться и принять условия лицензии, после чего загрузит и установит нужные сборки. Видимых изменений структуры проекта в окне `Solution Explorer` не произойдет, но если вы раскроете элемент `References` (Ссылки), то увидите, что были установлены новые сборки.

Создание контекста Entity Framework

Мы должны создать класс, который будет ассоциировать модель данных `Game` с созданной ранее базой данных. Одной из причин, по которым мы отдаем предпочтение инфраструктуре `Entity Framework`, является то, что она значительно упрощает этот процесс. Добавим в папку `Models\Repository` новый файл класса по `EFDbContext.cs` с содержимым, представленным в примере ниже:

```
using System.Data.Entity;

namespace GameStore.Models.Repository
{
    public class EFDbContext : DbContext
    {
        public DbSet<Game> Games { get; set; }
    }
}
```

Чтобы ассоциировать класс `Game` с базой данных, понадобится создать класс,

производный от System.Data.Entity.**DbContext** и имеющий свойство для каждой таблицы базы данных, с которой планируется работать.

Имя свойства указывает таблицу, а параметр типа результата DbSet — модель, которую инфраструктура Entity Framework должна использовать для представления строк в этой таблице. В рассматриваемом случае именем свойства является Games, а параметром типа — Game, что сообщает Entity Framework о необходимости применения типа модели Game для представления строк в таблице Games.

Нужно также указать Entity Framework, каким образом подключаться к базе данных; это делается путем добавления строки подключения к базе данных в файл Web.config. Файл Web.config содержит конфигурационную информацию для приложения ASP.NET Framework. Ниже показано содержимое файла Web.config вместе с добавленным определением строки подключения к базе данных:

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFi
             EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken
             requirePermission="false"/>
  </configSections>
  <connectionStrings>
    <add name="EFDbContext"
         connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=GameStore;
         providerName="System.Data.SqlClient"/>
  </connectionStrings>
  <system.web>
    <compilation debug="true" targetFramework="4.5"/>
    <pages controlRenderingCompatibilityVersion="4.0"/>
  </system.web>
  <entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConne
  </entityFramework>
</configuration>
```

Атрибут name соответствует имени класса, определенного в предыдущем разделе что позволяет Entity Framework автоматически обнаруживать информацию о подключении к базе данных. Чтобы выяснить необходимые значения для атрибутов connectionString и providerName в проекте, щелкните правой кнопкой мыши на подключении к базе данных в окне Server Explorer среды Visual Studio и выберите в контекстном меню пункт Properties (Свойства). Откроется диалоговое окно, содержащее всю нужную информацию.

Создание хранилища товаров

Осталось еще добавить класс хранилища, который оперирует на созданном ранее классе `EFDbContext` и действует в качестве шлюза между бизнес-логикой приложения и базой данных. Мы создали новый файл класса `Repository.cs` в папке `Models\Repository` с содержимым, представленным в примере ниже:

```
using System.Collections.Generic;

namespace GameStore.Models.Repository
{
    public class Repository
    {
        private EFDbContext context = new EFDbContext();

        public IEnumerable<Game> Games
        {
            get { return context.Games; }
        }
    }
}
```

В классе `Repository` определено свойство по имени `Games`, которое возвращает результаты чтения свойства с таким же именем в классе `EFDbContext`. Скоро мы добавим к этому классу дополнительную функциональность, но в настоящий момент мы достигли точки, где можем извлекать все строки из таблицы базы данных и представлять каждую из них в виде объекта `Game`.

Excel in Silverlight

Read/Edit Excel File in Silverlight Professional Silverlight Component

Alexandr Erohin ✯ alexerohinzzz@gmail.com © 2011 - 2016