Упражнение №4 по ПС

Работа с Бази данни и Entity Framework.

В упражнението се разглежда същността на Entity Framework, възможността да генерираме Модел на данните от вече създадена база и обратното с Code-First подхода.

Какво е Entity Framework

Писането и управлението на ADO.Net код за достъп до данни е еднообразна и монотонна работа. Microsoft предлага O/RM framework наречен "Entity Framework", за да се улесни и автоматизира дейностите в приложението ни свързани с бази данни.

Определение

Microsoft дава следното определение за Entity Framework:

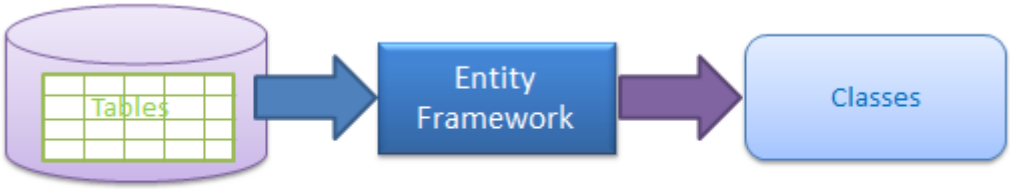
The Microsoft ADO.NET Entity Framework is an Object/Relational Mapping (ORM) framework that enables developers to work with relational data as domain-specific objects, eliminating the need for most of the data access plumbing code that developers usually need to write. Using the Entity Framework, developers issue queries using LINQ, then retrieve and manipulate data as strongly typed objects. The Entity Framework's ORM implementation provides services like change tracking, identity resolution, lazy loading, and query translation so that developers can focus on their application-specific business logic rather than the data access fundamentals.

Или накратко, Entity framework е фреймуърк за свързване на обекти/релации (Object/Relational Mapping (O/RM) framework). Той е разширение на ADO.NET, който дава на разработчиците автоматизиран механизъм за достъп и съхранение на данни в база данни.

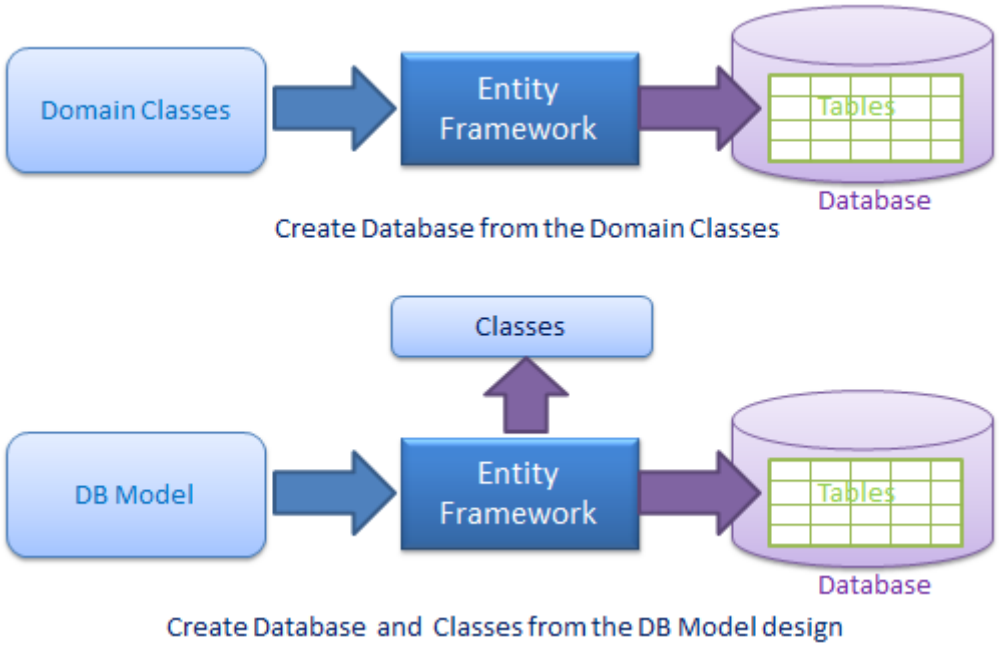
Entity framework (EF) е удобен в следните три случая:

1. Ако вече имате съществуваща база данни или смятате да проектирате базата преди останалите части на приложението.
2. Ако искате да се фокусирате на основните класове и след това от тях да генерирате базата данни.
3. Ако искате да проектирате схемата на базата данни с визуален дизайнер и след това да създадете самата база и класовете от схемата.

Фигура 1 илюстрира горните 3 сценария.



Generate Data Access Classes for Existing Database



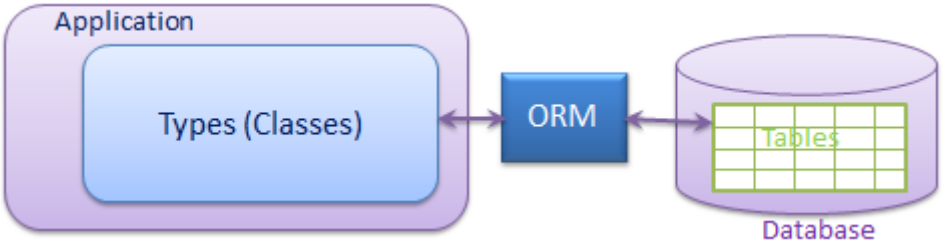
Фиг. 1. Сценарии за генериране на база и/или класове

Както е показано на фигурата горе, EF създава класовете за достъп до данните от съществуваща база данни, така че може да използвате тях, за да взаимодействате с базата вместо ADO.Net директно. EF може да създава база данни от описанието на основните класове с данни (Code-first).

Какво е O/RM?

ORM е инструмент за съхранение на данни от основни обекти на релационна база данни като MS SQL Server по автоматичен начин, без писане на много код. O/RM включва три основни части: обекти от основните класове (Domain class objects), релационни обекти (Relational database objects) и информация за връзките/напасването (Mapping information) за това как обектите се свързват с конкретните елементи на релационната база (таблици, изгледи и съхранение процедури). ORM позволява да отделим разработката на базата данни от тази на основните класове. Това прави приложението по-лесно за поддръжка и разширение. Също така ни позволява автоматизирани стандартни CRUD (Create, Read, Update & Delete) операции, създавани без разработчика да пише ръчно код.

Един типичен ORM инструмент генерира класове за взаимодействие с базата данни за приложението ви както е показано на фигура 2.

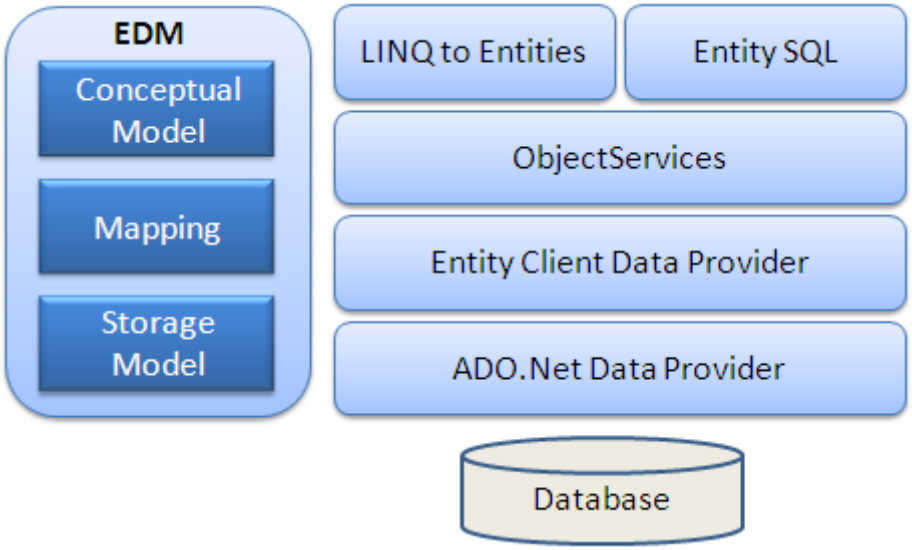


Фиг. 2. ORM

Entity Framework е open source ORM framework на Microsoft. Всеки може да допринесе към проекта Entity Framework project в codeplex. На пазара съществуват и много други ORM frameworks за .net като например DataObjects.Net, NHibernate, OpenAccess, SubSonic и др.

Архитектура на Entity Framework

Фигура 3 по-долу показва общата архитектура Entity Framework.



Фиг. 3. Архитектура на Entity Framework

Ще разгледаме всеки един от компонентите на архитектурата по отделно:

EDM (Entity Data Model): EDM се състои от три основни части - Conceptual model, Mapping и Storage model.

- **Conceptual Model:** Концептуалният модел съдържа класовете модели и техните връзки. Това е независимо от дизайна на вашата таблица.
- **Storage Model:** Модел на съхранение е модела на структурата на базата данни, включващ таблици, изгледи (views), съхранени процедури, релации и ключове.
- **Mapping:** Връзките (напасването) се състоят от информация за това, кой концептуален модел е свързан с даден модел на съхранение.

LINQ to Entities: LINQ to Entities е език за заявки, използван за писането на заявки към обектния модел. Той връща entities, които са дефинирани в концептуалния модел. Необходими са допълнителни познания за LINQ.

Entity SQL: Entity SQL е друг език за заявки, подобно на LINQ to Entities. По труден е от L2E и разработчиците трябва да го изучват отделно.

Object Service: Object service е отговорен за „материализацията“, което е процеса по преобразуване на данните върнати от entity client data provider (долния слой) към обектна структура на entity.

Entity Client Data Provider: Основната задача на този слой е да преобразува L2E или Entity SQL заявки в SQL заявки, които са „разбираеми“ от базата данни. Той си комуникира с ADO.Net data

provider, който от своя страна изпраща и получава данните от базата.

ADO.Net Data Provider: Този слой комуникира с базата данни, използвайки стандартния ADO.Net.

Конфигуриране на средата за работа с Entity Framework

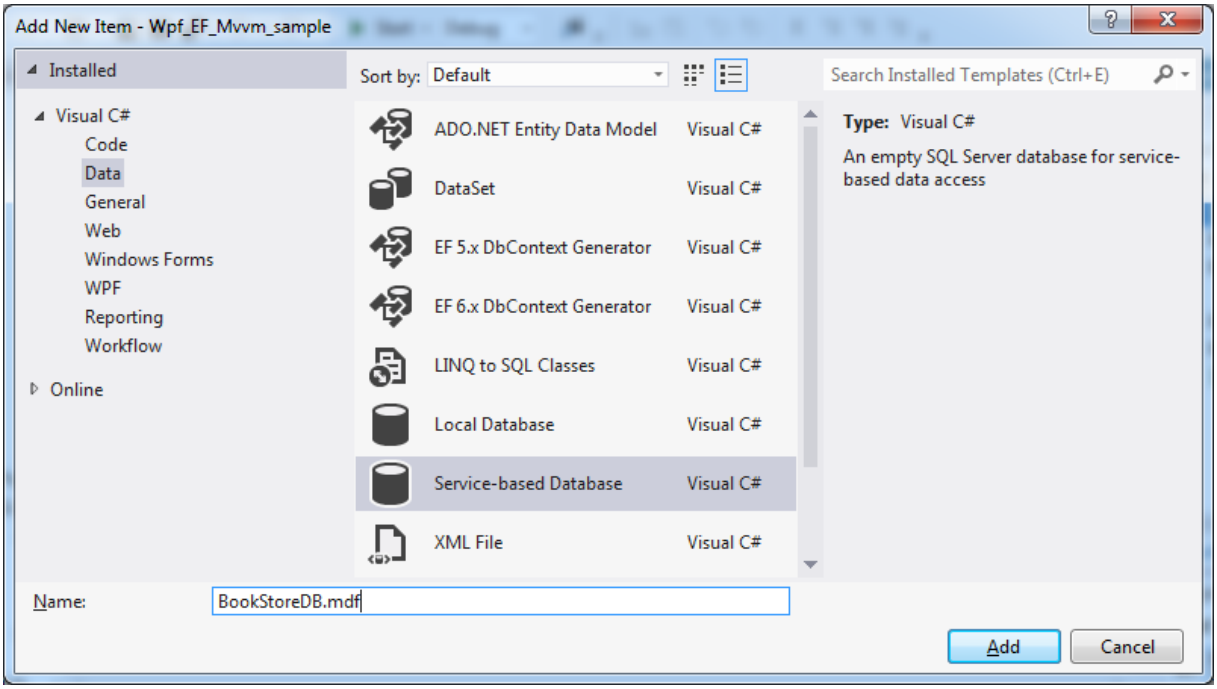
Ако работите на собствен компютър най-вероятно ще трябва да изтеглите и инсталирате Entity Framework 6 Tools for Visual Studio 2012 & 2013 (източник 1). Ако ползвате компютрите в залата, то те вече са конфигурирани и може да пропуснете тази стъпка от упражнението.

Създаване на локална база данни

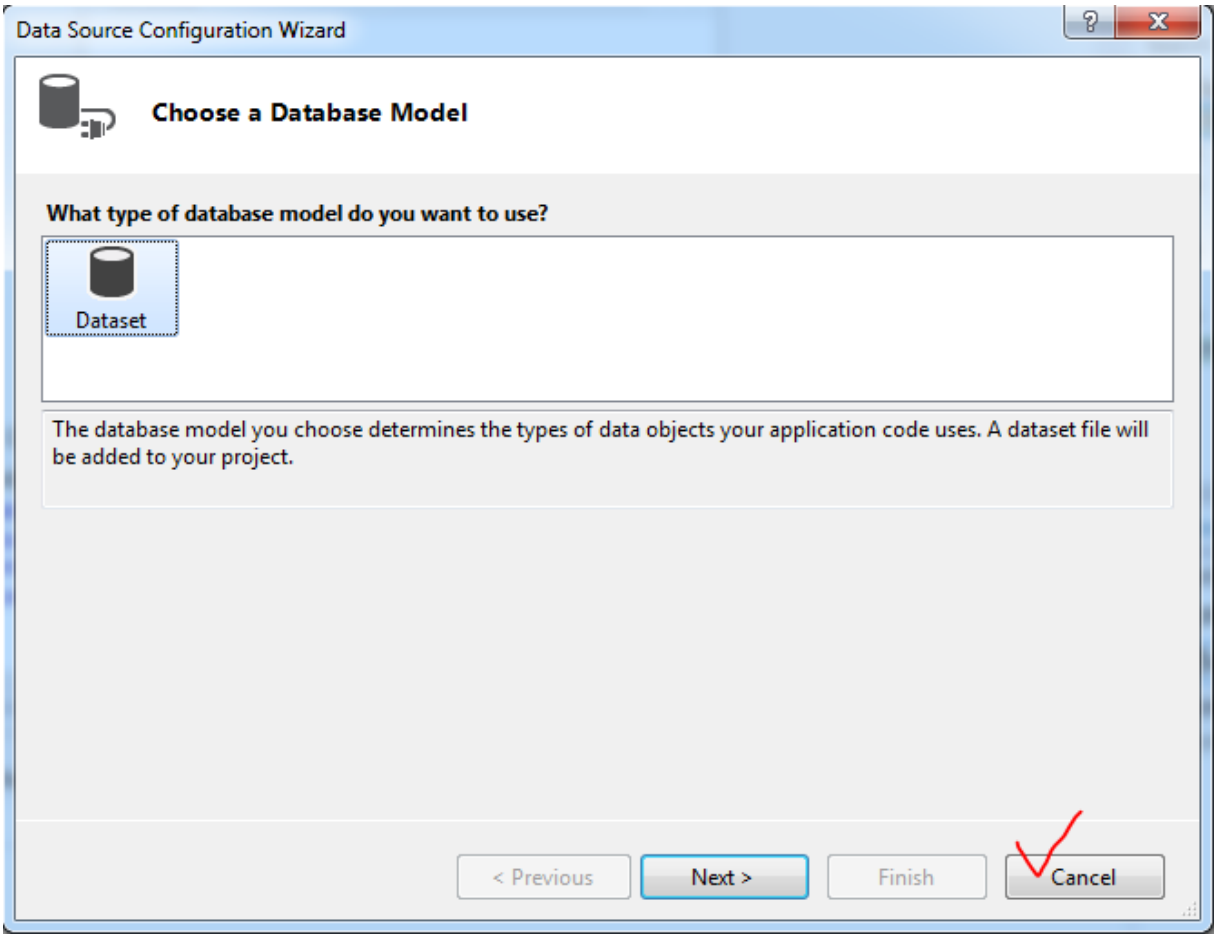
В този пример ще създадем просто WPF – MVVM приложение, което използва Entity Framework (EF) за връзка към локална за приложението база данни.

1. Създайте WPF приложение с име `Wpf_EF_Mvvm_sample` .

2. Към проекта добавете файл от тип Service-based Database, с име **BookStoreDB.mdf** . Това ще е файла на локалната ни база.



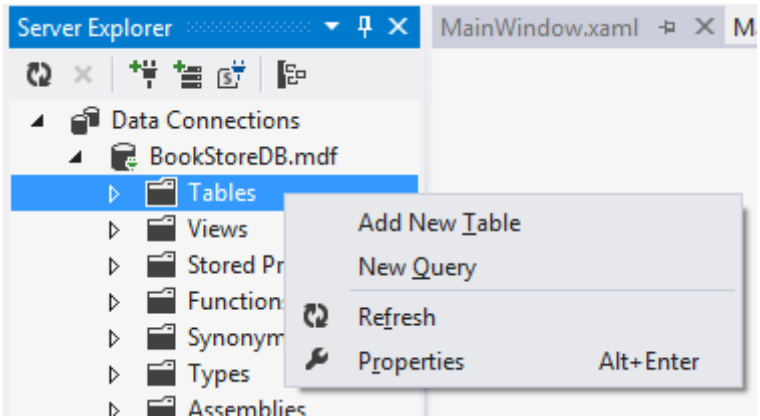
В отворилия ви се прозорец за „Data Source Configuration Wizard“ изберете Cancel. Ще създадем ентити модела после (след като си направим базата и нейните таблици).



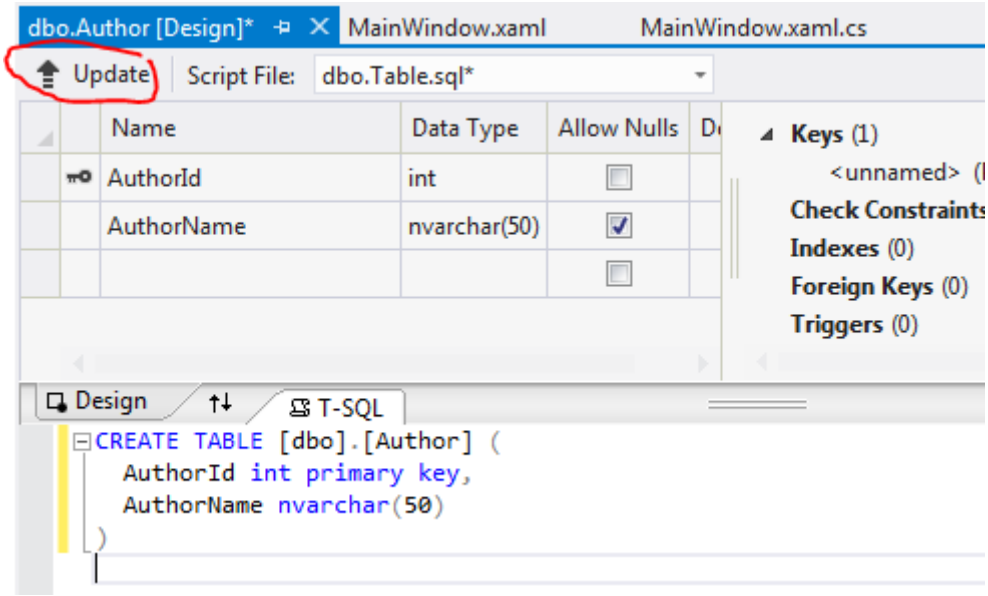
3. Добавяне на таблици Author и Book към базата. Следвайте следните стъпки и въвеждайте информацията от screenshot-овете:

3.1. Натиснете десен бутон върху BookStoreDB.mdf и изберете Open… Ще ви се отвори структурата на базата данни в прозорец наречен Server Explorer. За сега базата ни е празна.

3.2. В Server Explorer, върху Tables изберете Add New Table.

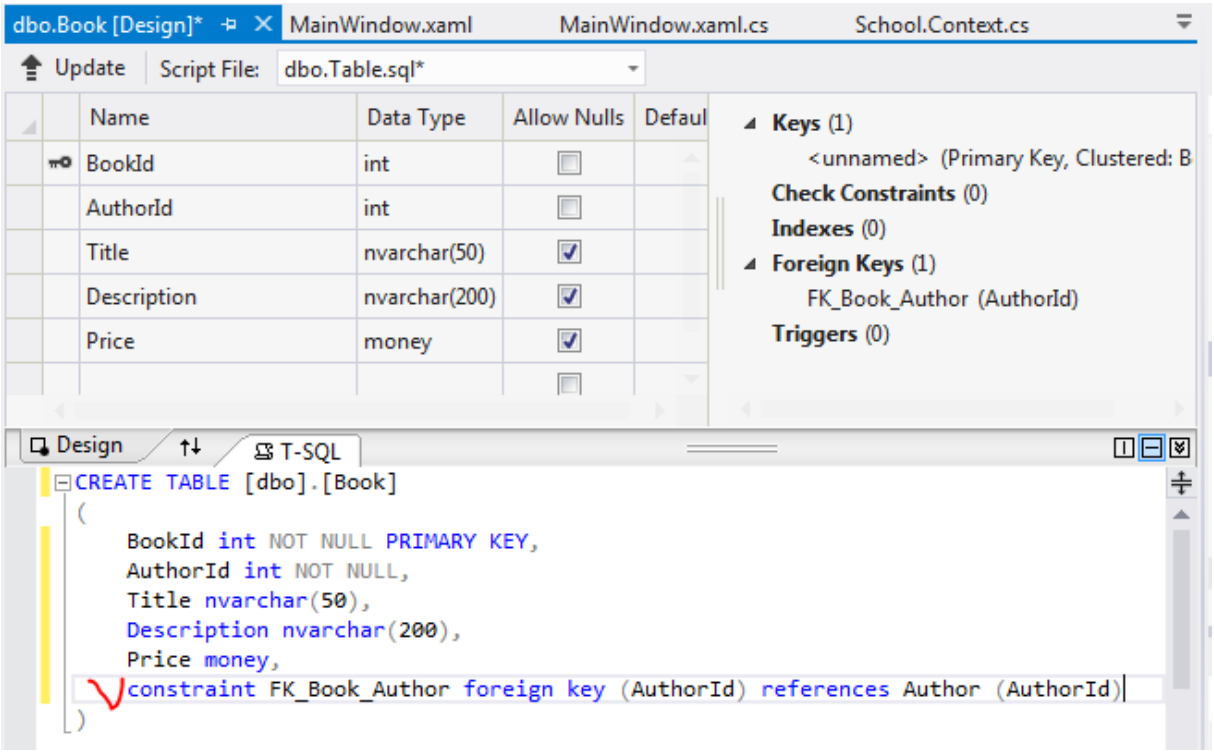


3.3. В прозореца въведете имена на колоните и типовете както са показани на долния скрийншот. След като сте готови с описанието, за да създадете реално таблицата натиснете бутона “Update Database”.



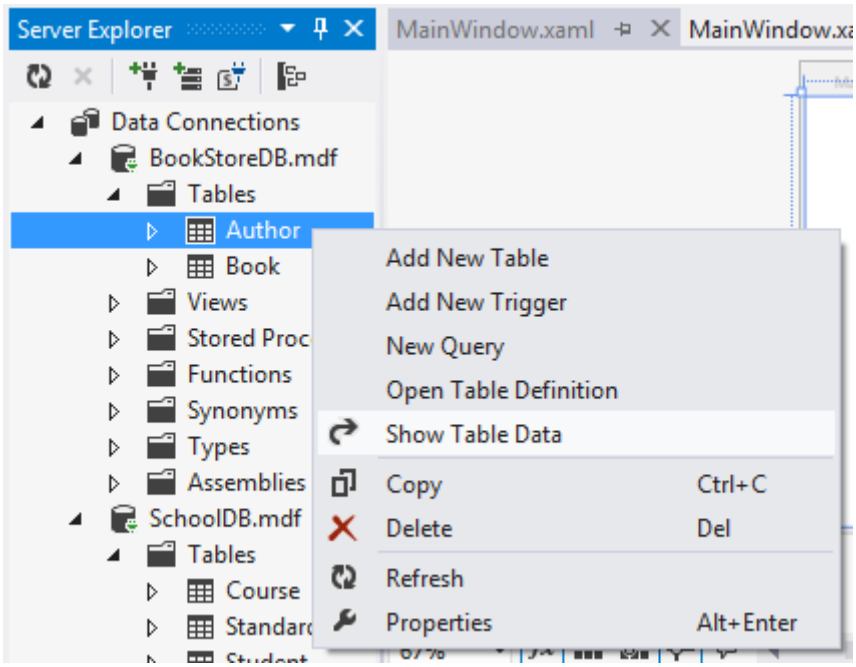
В отворилия се прозорец натиснете Update Database, за да се изпълни току що генерирания SQL скрипт.

3.4. Аналогично направете втора таблица със следните полета:

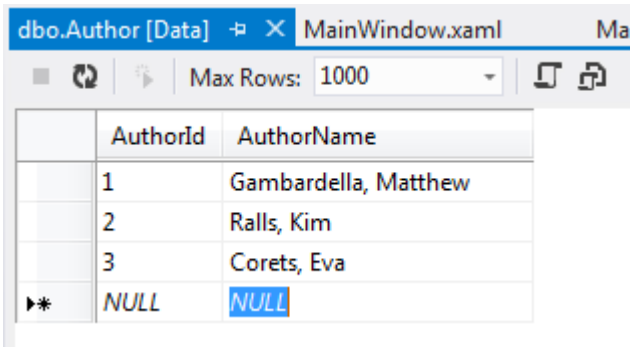


4. Добавяне на данни в таблиците.

4.1. Единия начин да добавим данни в таблицата е ръчно. Изберете с десен бутон върху таблицата Author → Show Table Data.

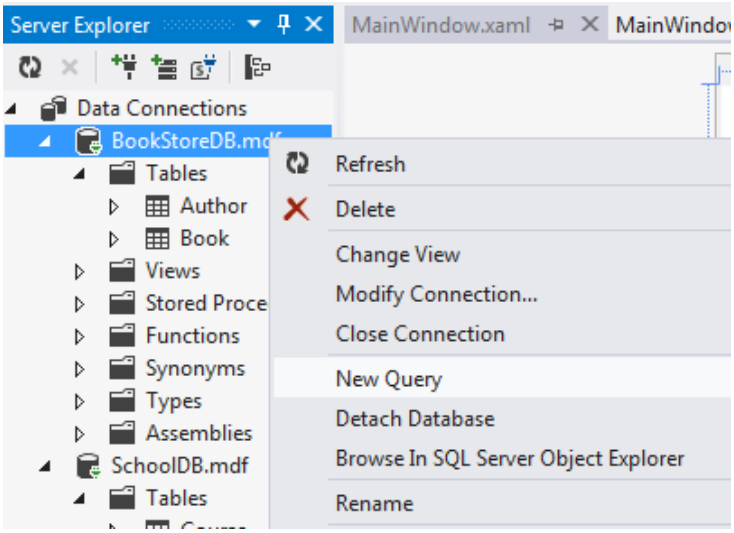


4.2. В новоотворилия се прозорец попълнете следните данни.



Важно: *Тъй като този визуален интерфейс изпълнява SQL команди, е добре да го затваряте веднага след като приключите работа с него.... защото иначе лесно се „чупи“, губи си връзката с базата данни и други весели неща.*

4.3. Данните в другата таблица ще добавим със скрипт, както си знаете от БД. С десен бутон върху името на базата данни в Server Explorer изберете New Query…



4.4. В отворилия се таб (най-вероятно с име SQLQuery1.sql) поставете следния код:

```
insert into Book values (1, 1, 'XML Developers Guide', 'An in-depth look at creating
applications with XML.', 4.95)

insert into Book values (2, 1, 'Foods of the world.', 'Simply a book about food.', 5.95)

insert into Book values (3, 1, 'Cars', 'A book about cars.', 8.95)

insert into Book values (4, 2, 'Scarecrows', 'This be could horror or agriculture.', 4.15)

insert into Book values (5, 3, 'Book of blue', 'First in a series of books about colors',
6.30)

insert into Book values (6, 3, 'EF', 'Some tips and trics on Entity Frameworks', 3.45)
```

Натиснете бутона за **Execute** (триъгълник горе в ляво на прозореца). След това проверете дали данните са успешно записани в таблица Book, като отворите съдържанието ѝ с **Show Table Data**. Ако всичко е наред, можете да затворите Query файла без да го запаметявате.

Имплементация на приложение с локална база данни

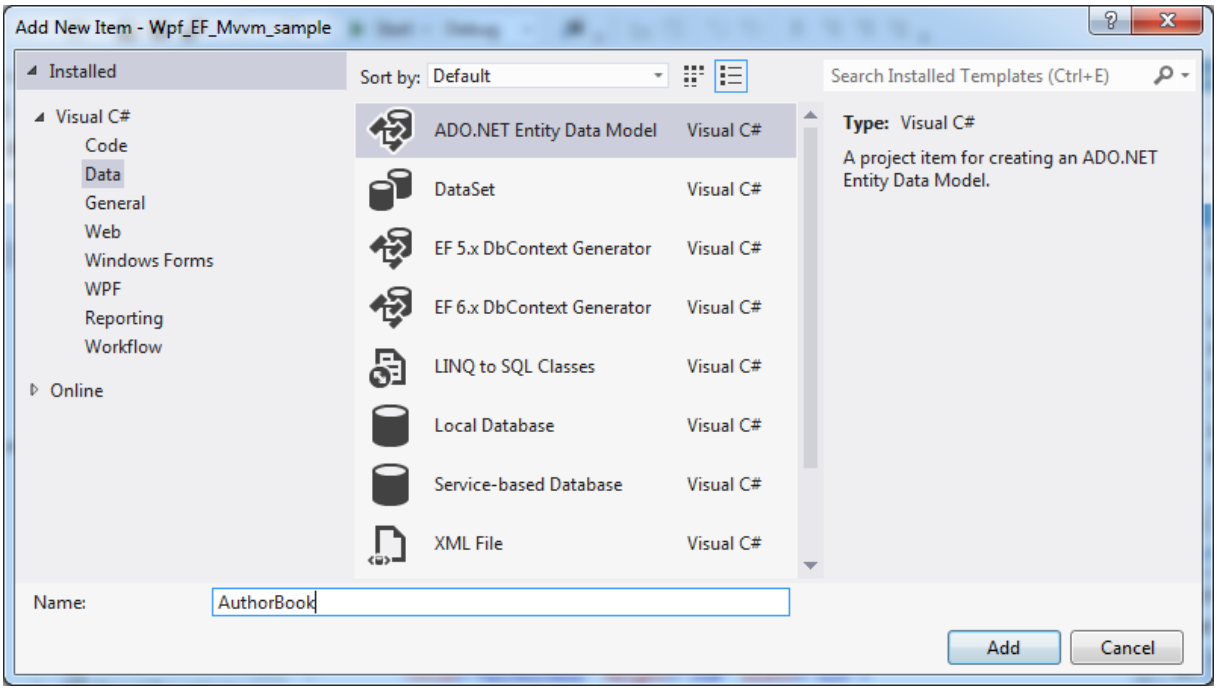
Вече създадохме проекта, но със следващите точки ще го изградим в MVVM стила. Трите изграждащи го слоя ще бъдат следните:

- Презентационен слой – ще съдържат съответно два combobox-а, един textbox (в MainWindow.xaml).
- Бизнес слой – ViewModel с име на файла MainWindowViewModel.cs
- Слой на данните – ADO.NET Entity Model, който съдържа две таблици (Author and Book)

При стартиране на приложението MainWindowViewModel ще зареди ентити таблицата Author, което е източника данни (datasource) за първия combobox (ще го наричаме за по-кратко combo1). След избиране на елемент от combo1, втория комбобокс (на кратко combo2) се зарежда с данни от съответните редове от ентитито на таблица Book (книгите на избрания автор). А след избор на елемент в combo2 в текстбокс полето ще се зареди съдържанието на колоната „Description“ за избрания запис.

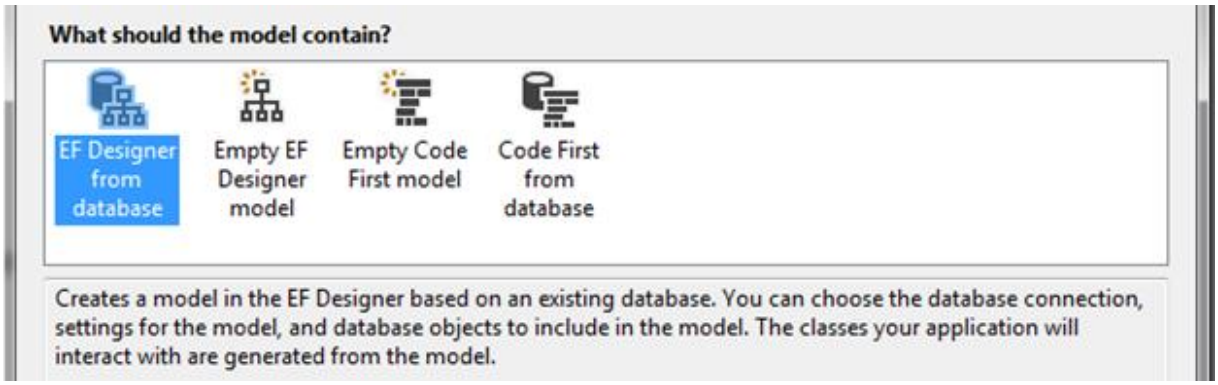
Създаване на Entity Model

5. Първо ще добавим ADO.NET Entity Mode към проекта (десен бутон → Add New Item) и кръстете файла "**AuthorBook**". Това ентити ще играе ролята на Модел в нашето MVVM. Създаденият файл трябва да е с разширение „.edmx“.

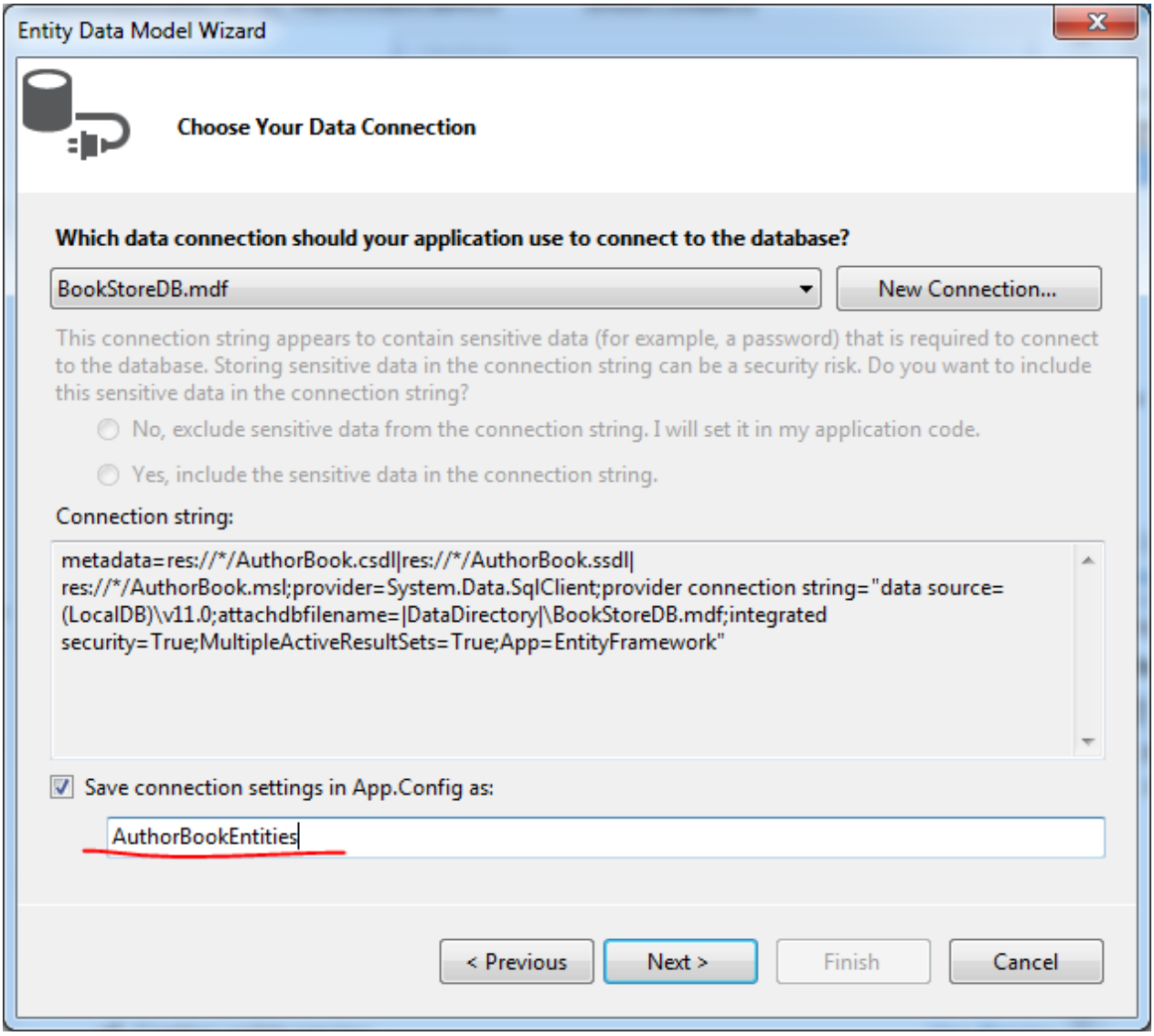


**Отново няма да спазим разделянето на слоевете в отделни папки, за целите на максимално опростените примери. Когато разработвате собствени приложения не забравяйте да го имплементирате!*

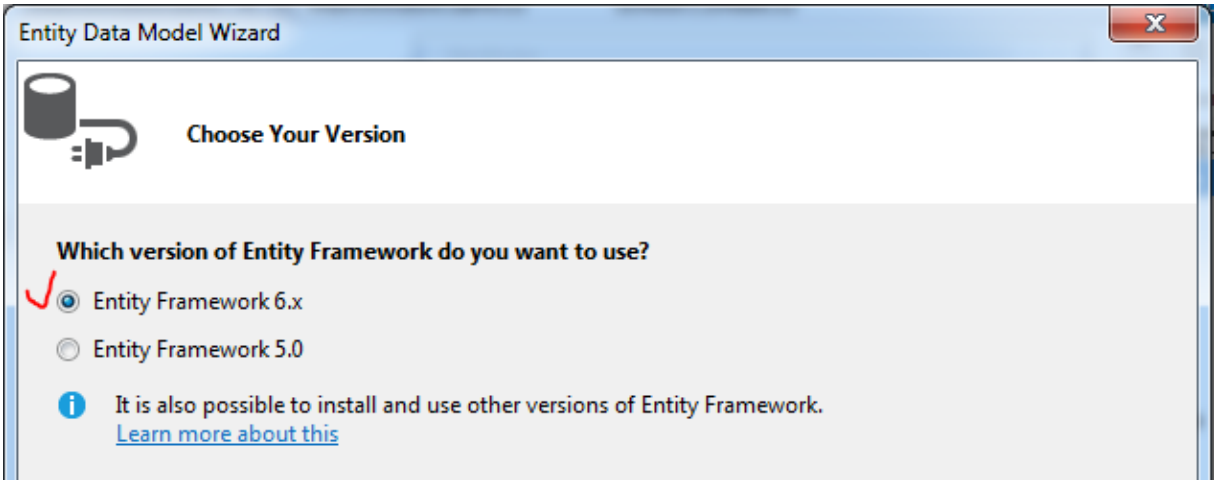
6. При конфигурирането създаваното ентити, в отворилия се wizard избираме "**Generate from Database**", т.е. ще създадем ентити модел към вече готова база.



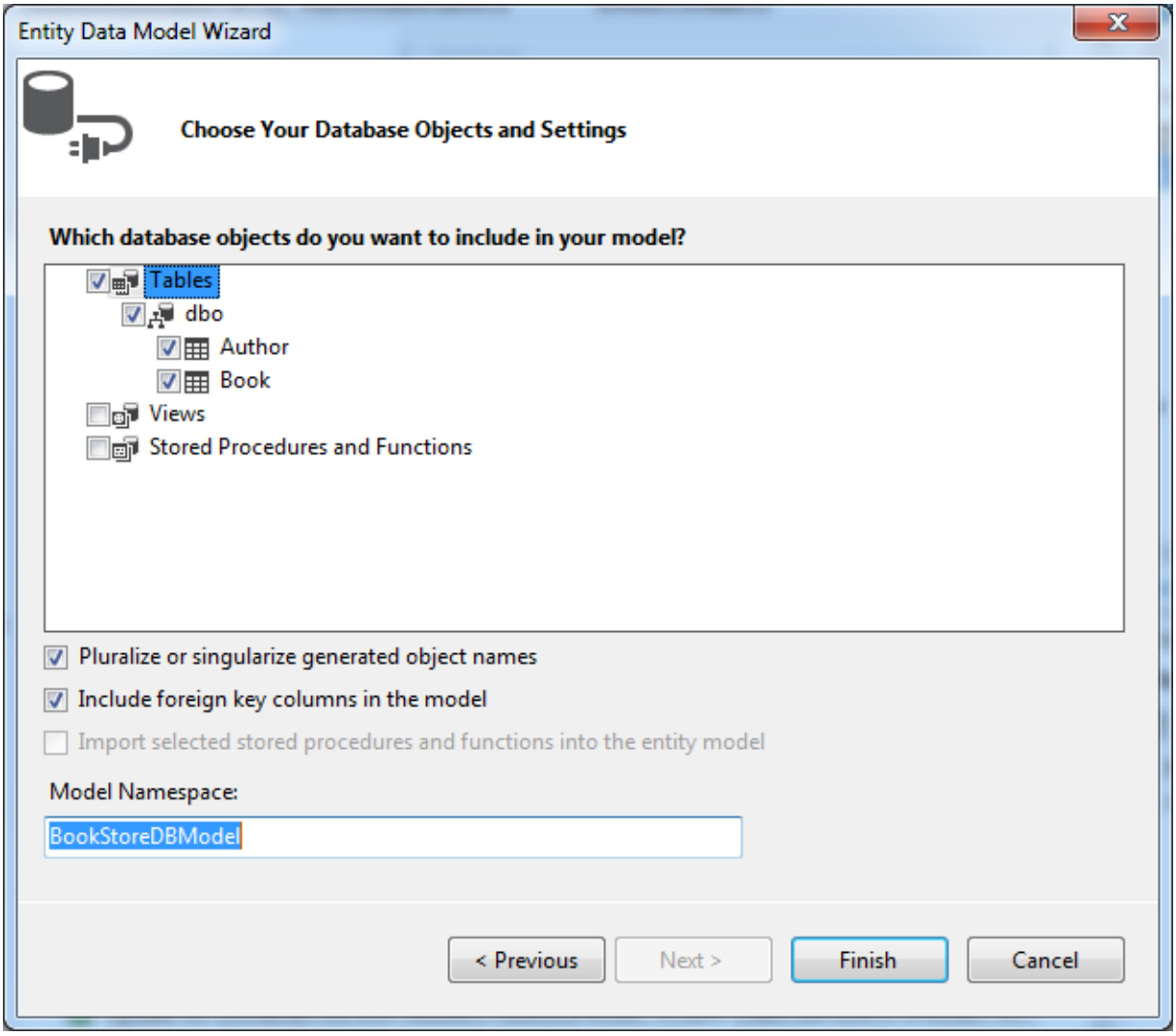
6.1. В следващия прозорец изберете от падащия списък името на вашата база. Бутона „New Connection“ се ползва, когато искате да се вържете към външна база данни.



6.2. Задайте име AuthorBookEntities, а в следващия прозорец (той не е задължително да се появи, тъй като е възможно версията да е настроена предварително) изберете версия 6.x .



6.3.В долния прозорец е показано как да маркирате таблиците, за които искате да се генерират класове в ентити модела. Е конкретния случай това са Author и Book. В същия диалог сменяме името на модела на **AuthorBookDBModel** и натискаме **Finish**.



С това нашия Entity Model е вече генериран. Това представлява нашия слой на данните (Data layer) или така наречения Модел в MVVM.

Създаване на ViewModel

7. Добавете към проекта клас файл с име **MainWindowViewModel.cs**.

7.1. Във файла е необходимо да добавим референцията **using System.ComponentModel;** , за да е достъпен интерфейса **INotifyPropertyChanged**. От предното упражнение знаем, че трябва да го наследим и имплементираме, за да работи двупосочния binding.

7.2. Добавете и референцията **using System.Runtime.CompilerServices;** . В този пример ще реализираме метода NotifyPropertyChanged("somePropertyName") (този, който викаме в SET методите на свойствата и приемаше параметър името на свойството), по малко „по-красив“ начин, без досадния параметър - NotifyPropertyChanged(). Посредством етикета **[CallerMemberName]** , който ще видите как се използва по-надолу в кода ще реализираме автоматично вземане на името на свойството.

7.3. Единствения сорс код, който ще пишем в проекта, ще се намира във файла **MainWindowViewModel.cs** , който вече създадохме. Това е така наречения „Бизнес слой“ или логика на програмата.

7.3. а) Добавете имплементацията на метода NotifyPropertyChanged().

```
public event PropertyChangedEventHandler PropertyChanged;
private void NotifyPropertyChanged([CallerMemberName] String propName = "")
{
   if (PropertyChanged != null)
   {
      PropertyChanged(this, new PropertyChangedEventArgs(propName));
   }
}
```

7.3. б) В класа добавете следните свойства и техните private полета:

```
        private List<Author> _authors;
        public List<Author> Authors
        {
            get
            {
                return _authors;
            }
            set
            {
                _authors = value;
                NotifyPropertyChanged();
            }
        }

        private List<Book> _books;
        public List<Book> Books
        {
            get
            {
                return _books;
            }
            set
            {
                _books = value;
                NotifyPropertyChanged();
            }
        }

        private Book _selectedBook;
        public Book SelectedBook
        {
            get
            {
                return _selectedBook;
            }
            set
            {
                _selectedBook = value;
                NotifyPropertyChanged();
            }
        }

        private Author _selectedAuthor;
        public Author SelectedAuthor
```

```
{
    get
    {
        return _selectedAuthor;
    }
    set
    {
        _selectedAuthor = value;
        NotifyPropertyChanged();
    }
}
```

7.3. в) В началото на класа създайте и инициализирайте обекта **ctx**, който ще прави връзката с нашите Entity класове в модела. Това е нашия **контекстен** обект.

```
AuthorBookEntities ctx = new AuthorBookEntities();
```

7.3. г) Добавете следния метод, който ще извика ентити заявка към базата данни за да вземе записите за всички автори. Резултатът от нея се записва в свойството **Authors**:

```
private void FillAuthors()
{
    var q = (from a in ctx.Authors
             select a).ToList();
    this.Authors = q;
}
```

Удебеленият код представлява LINQ заявка. Тя връща DBSet, който не може да бъде консумиран директно от контролите, затова се налага да го преобразуваме в списък (list). Не случайно зададохме свойството Authors да бъде от тип List<Author>. То ще се използва като източник на списъка от автори за combo1 (ще байндем свойството ItemSource за него). Свойството SelectedAuthor пък ще бъде свързано със атрибута SelectedItem на combo1. (ще го направим в точка 8)

7.3. д) В конструктора на класа извикайте метода **FillAuthors(),** за да се направи началното зареждане на всички автори в първия комбобокс.

7.3. е) Добавете метод **FillBook()**, който ще селектира от базата книгите на избрания автор.

```
private void FillBook()
{
    Author author = this.SelectedAuthor;

    var q = (from book in ctx.Books
             orderby book.Title
             where book.AuthorId == author.AuthorId
             select book).ToList();
    this.Books = q;
}
```

7.3. ж) Добавете в **set** метода на **SelectedAuthor** извикването на **FillBook()**, за да може след промяна на избрания автор да се случи автоматично зареждане на неговите книги.

** Една удобна функция на Visual Studio е, че можете да се отървете от излишните using декларации във вашия код. Преместете курсора някъде в кода на using клаузите и натиснете десен бутон. Изберете "Organize Usings" от контекстното меню и след това*

изберете "Remove unused usings".

7.4. Компилирайте (Build → Build Solution). Би трябвало да няма никакви компилационни грешки, тъй като долните слоеве са независими от изгледа и могат да бъдат тествани без да сме написали нашия XAML.

Създаване на изгледа

8. Файлът **MainWindow.xaml** за презентационния слой беше създаден още с проекта и сега остава да развием неговия дизайн и връзки към долния слой.

8.1. В отварящия таг Window добавете връзката към namespace-а на приложението ви. Вместо Wpf_EF_Mvvm_sample сложете името на вашия проект ако в началото не сте го кръстили така.

```
xmlns:vm="clr-namespace:Wpf_EF_Mvvm_sample"
```

8.2. Добавете следния data context в XAML за прозореца.

```
<Window.DataContext>
    <vm:MainWindowViewModel />
</Window.DataContext>
```

9. В грида добавете следните дефиниции:

```
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="180" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="25" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
```

10. Добавете следните визуални контроли в грида:

10.1. Комбобокс combo1, който ще зарежда списъка с автори, като показва техните имена. Имаме binding на ItemsSource със свойството Authors, и на SelectedItem с SelectedAuthor.

```
<ComboBox Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="1" Name="combo1"
          DisplayMemberPath="AuthorName" ItemsSource="{Binding Authors}"
          SelectedItem="{Binding SelectedAuthor}" />
```

10.2. Втория комбобокс combo2, ще извежда списък с книги на даден автор. Тук binding-а е между ItemsSource и свойството Books, и SelectedItem – SelectedBook.

```
<ComboBox Grid.Row="0" Grid.Column="1" Grid.ColumnSpan="1" Name="combo2"
          DisplayMemberPath="Title" ItemsSource="{Binding Books}"
          SelectedItem="{Binding SelectedBook}" />
```

10.3. Накрая поставяме TextBlock за извеждане на информацията за избраната книга. Атрибута Text е свързан със полето Description на свойството SelectedBook.

```
<TextBlock Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="2" Name="tbDesc"
           Text="{Binding SelectedBook.Description}" />
```

11. Тествайте дали програмата работи както трябва.

Прост пример за Code First подхода

Ще създадем простичък пример за програма, в която първо ще опишем класовете описващи данните за едно училище – например Ученици, Оценки, Учители и Класове, а след това по техния модел автоматично ще се създаде база данни, в която те ще се съхраняват.

0. Към Solution-а ви от първата част добавете нов проект с име SchoolCodeFirst.

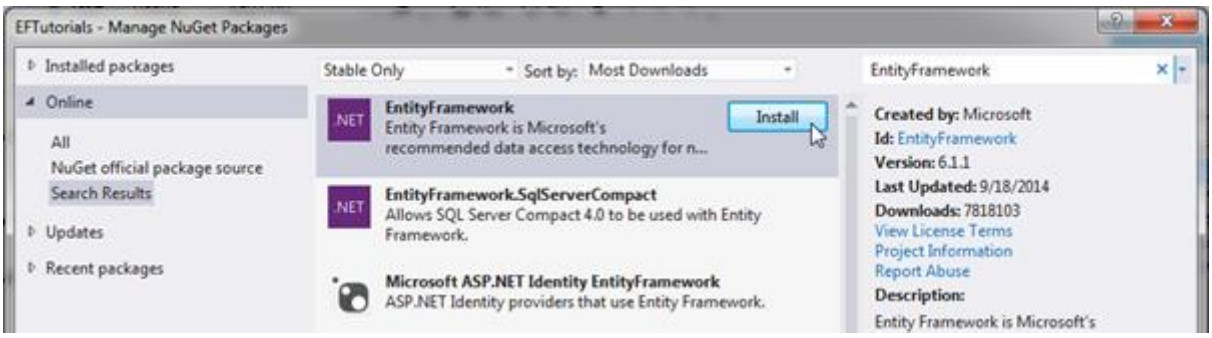
Добавяне на референции към Entity Framework в даден проект

В новия си проект трябва да добавим референция към библиотеките за Entity Framework. Някой от стъпките надолу могат да варират в зависимост от нивото на инсталация на компютъра, на който седите.

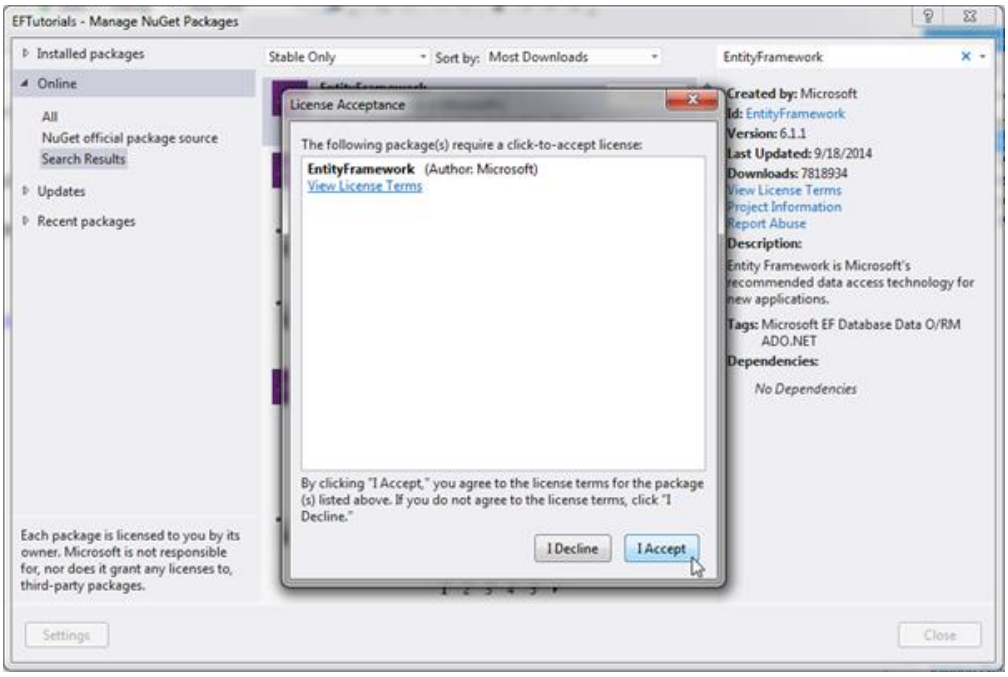
1. Натиснете с десния бутон на мишката върху името на вашия проект в solution explorer и от контекстното меню изберете **Manage NuGet Packages**… Това ще отвори за вас диалоговия прозорец **Manage NuGet packages**.

2. В ляво изберете „Online“

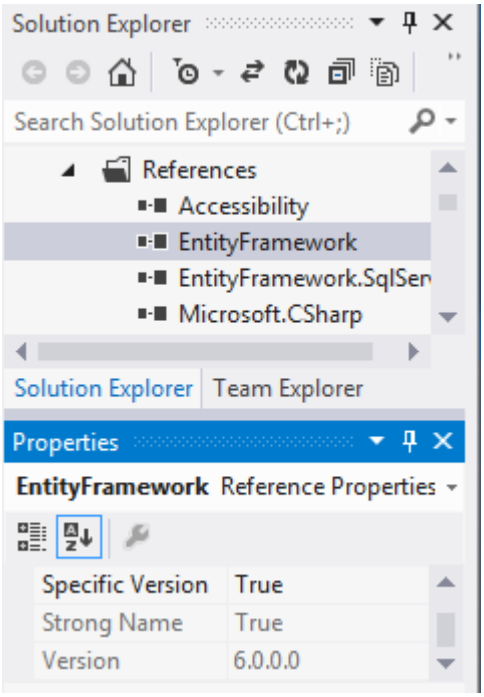
3. Намерете EntityFramework пакета и натиснете Install



4. Натиснете на бутона **I Accept** за потвърждение на лиценза. Това ще стартира инсталацията.



След като тя завърши успешно, в проекта ви ще бъде добавен автоматично EntityFramework.dll файла (не се вижда в Solution Explorer, а само в реалните файлове). Под категория References трябва да виждате нещо подобно.



След тази стъпка сме готови да ползваме Entity Framework в конкретния проект!

Добавяне на класове описващи данните

Вместо да разработваме първо базата данни, ще създадем първо класовете за училището.

First, we will create two simple Student and Standard classes where every Student is associated with one Standard as shown below.

5. Добавете клас файл Student.cs в проекта и сложете в него кода, описващ полетата (свойствата) на класа.

```
public class Student
 {
     public Student()
     {

     }
     public int StudentID { get; set; }
     public string StudentName { get; set; }
     public DateTime? DateOfBirth { get; set; }
     public byte[] Photo { get; set; }
     public decimal Height { get; set; }
     public float Weight { get; set; }

     public Standard Standard { get; set; }

     public override string ToString() { return this.StudentName; }
 }
```

6. Добавете клас файл Standard.cs в проекта и сложете в него кода, описващ полетата (свойствата) на класа. Това е клас, който ще представлява училищен клас съдържащ в себе си списък от ученици.

```
public class Standard
{
    public Standard()
    {

    }
    public int StandardId { get; set; }
    public string StandardName { get; set; }

    public ICollection<Student> Students { get; set; }

}
```

Добавяне на DbContext

След като приключим с описването на класовете (за сега ще разработим само тези 2-та, за да не усложняваме допълнително примера), трябва да напишем и контекстен клас, който наследява DbContext. Това се изисква от Code-First подхода.

7. Създайте клас SchoolContext, както е показано по-долу. Той наследява класа DBContext и предоставя свойства от типа DbSet за типовете, които искаме да са част от модела (в нашия случай Student и Standard класовете). DbSet е колекция от ентити класове, затова на свойствата от този тип даваме имена в множествено число (Students и Standards). Необходимо е да добавите референцията **using System.Data.Entity;**

```
public class SchoolContext : DbContext
{
    public SchoolContext() : base()
    {       }
    public DbSet<Student> Students { get; set; }
    public DbSet<Standard> Standards { get; set; }
}
```

Създаване на визуален интерфейс за тестване

В програмата си ще поставим 2 бутона и един ListBox. Първия бутон ще служи за добавяне на нов ученик, а при натискане на втория в ListBox-а ще се показват въведените в базата ученици. (посочения по-долу код е СПАГЕТИ!)

8. В грида на вашия MainWindow добавете следните дефиниции и елементи.

```
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="100"/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
<Button Grid.Row="0" Height="100" Click="Button_Click">Add students</Button>
<Button Grid.Row="1" Height="100" Click="ShowStudentsClick">Show students</Button>
<ListBox x:Name="StudentsList" Grid.Column="1" Grid.RowSpan="2" />
```

9. В Code-behind файла MainWindow.xaml.cs добавете метода за добавяне на ученик. За да правим разлика между добавянията, в името се записва времето на добавяне:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    using (var ctx = new SchoolContext())
    {
        Student stud = new Student()
        {
            StudentName = "New Student " + DateTime.Now.ToLongTimeString()
        };
        ctx.Students.Add(stud);
        ctx.SaveChanges();
    }
}
```

10. В Code-behind файла MainWindow.xaml.cs добавете метода прочитане и показване на учениците в базата.

```
private void ShowStudentsClick(object sender, RoutedEventArgs e)
{
    using (var ctx = new SchoolContext())
    {
      StudentsList.ItemsSource = ctx.Students.ToList();
    }
}
```

11. Тествайте програмата като я стартирате няколко пъти и всеки път добавяте нови хора. Вижте как дори и след затваряне на програмата, старите записи се пазят.

Важно: Компютрите в зала 1115 нямат администраторски права и базата данни не може да бъде създадена поради тази причина. Отворете Visual Studio с администраторски права (десен бутон → Run as Administrator) като повикате преподавателя, за да въведе администраторската парола.

Задача за самостоятелна работа в часа:

Добавете MainWindowViewModel и преместете в него кода от Code-behind, добавете необходимите свойства и команди, за да се получи приложение спазващо MVVM модела:

1. Дефинирайте свойство за списъка от ученици, направете binding с него в контролата ListBox; (виж пример 1 от това упр.)
2. Направете команда, която да вика метода за добавяне на ученик, а не това да става с „onclick“ (виж пример 2 от упр. 3)
3. Направете автоматично обновяване на ListBox след добавяне.
4. Добавете TextBox поле, в което да се въвежда името на новия ученик. Направете подходящ Binding, така че вместо "New Student " в базата да се записва въведеното от потребителя име.
5. Структурирайте кода в папки View, ViewModel и Model. Не забравяйте, че като ги преместите няма автоматично да се зададе правилен под-namespace. (пр. 3 - упр. 3)

Задача за домашно:

Намерете къде всъщност е базата данни, която се създава. Можете да прочетете повече информация за това например тук https://msdn.microsoft.com/en-us/data/jj193542.aspx

Източници

1. Entity Framework 6 Tools for Visual Studio 2012 & 2013, (download) https://www.microsoft.com/en-us/download/confirmation.aspx?id=40762
2. http://www.entityframeworktutorial.net
3. Microsoft MSDN, Entity Framework Overview, https://msdn.microsoft.com/en-us/library/bb399567%28v=vs.110%29.aspx
4. Microsoft MSDN, Walkthrough: Binding WPF Controls to an Entity Data Model, https://msdn.microsoft.com/en-us/library/dd465159%28v=vs.110%29.aspx
5. Richard Protzel, Tutorial for a Basic WPF – MVVM Project Using Entity Framework, 6 Feb 2015, http://www.codeproject.com/Articles/873592/Tutorial-for-a-Basic-WPF-MVVM-Project-Using-Entity
6. Adventure Works for SQL Server 2012, http://msftdbprodsamples.codeplex.com/releases/view/55330
7. Amarnath Kashyap, CodePlex, WPF MVVM with Entity Framework, https://vherp.codeplex.com/