Технология доступа к базам данных

ADO.NET

Урок №8

Модель сначала (Model First)

Содержание

Модель сначала (Model First)

На первом уроке мы создали приложение, демонстрирующее основные приемы работы с Entity Framework. В том приложении мы использовали подход «база данных сначала». У нас была приготовлена БД, и для нее мы создали Entity Data Model. Сейчас мы с вами создадим еще одно приложение, использующее Entity Framework. Но создадим его, используя другой подход. Мы с вами создадим Entity Data Model в специальном дизайнере Visual Studio, а затем, на основе созданной EDM, создадим БД. Вы должны помнить, что такой подход называется «модель сначала».

Снова создадим новое консольное приложение, например, с именем LibraryModelFirst. Теперь надо установить для создаваемого проекта последнюю версию Entity Framework. Вы уже знаете, что это делается с помощью утилиты NuGet. Давайте повторим действия по установке последней версии фреймворка. Для этого активируем меню Project — Managing NuGet packages (*Проект — Управление пакетами NuGet*). В левом верхнем углу появившегося окна введем имя EntityFramework в поле для поиска, а в вертикальной панели слева выберем место поиска — «В сети». Через какое-то время в центральной части окна будет отображен найденный пакет

EntityFramework, рядом с которым будет расположена кнопка Setup (*Установить*). В правой части окна будет выведено описание пакета: кем он создан, какая версия, дата релиза и другая информация. Нажимаем кнопку Setup (*Установить*). Установка занимает совсем немного времени. Вот теперь мы готовы приступить к работе с Entity Framework.

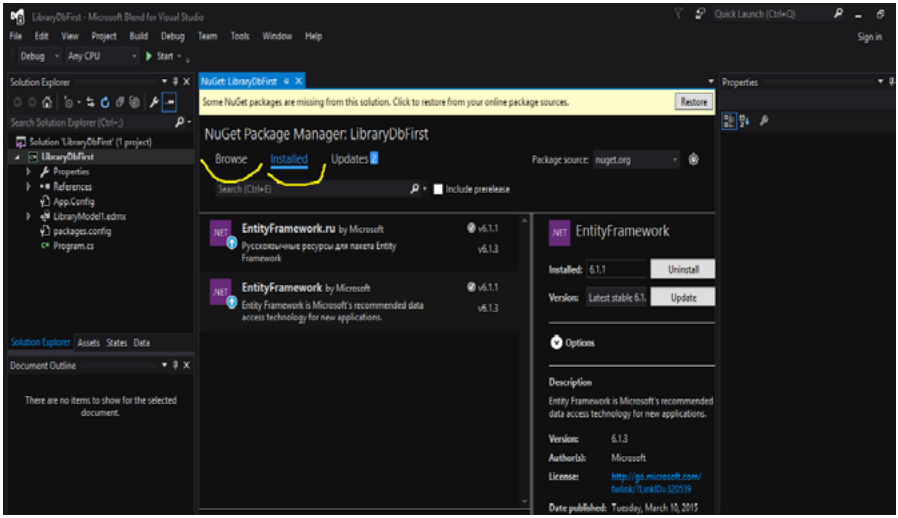

Рис. 1. Работа с NuGet

Давайте проверим, что установка последней версии Entity Framework действительно выполнена. Сделать это очень просто. В обозревателе решений разверните узел References, в нем выделите узел Entity Framework, нажмите правую кнопку мыши и выберите опцию Properties (*Свойства*). В окне, которое откроется под окном обозревателя решения, вы увидите версию установленного фреймворка (рис. 2).

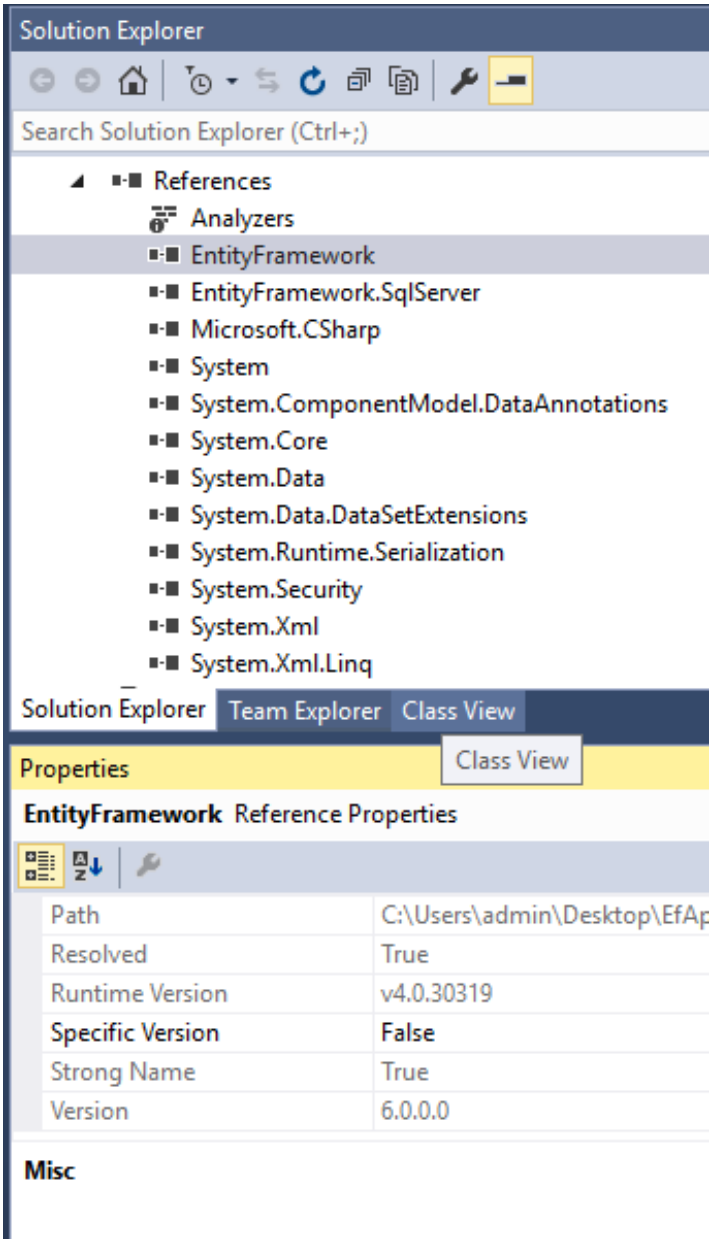

Рис. 2. Версия NuGet

Теперь перейдем к созданию Entity Data Model. Для этого надо выполнить следующие действия:

1. Перейти в обозреватель решения созданного проекта, выбрать там проект и активировать контекстное меню (нажав правую кнопку мыши);
2. Выбрать команду Add-New item (*Добавить-Создать элемент*);
3. В появившемся окне найти и выбрать элемент ADO.NET EDM Model (Модель ADO.NET EDM) с именем по умолчанию Model1.edmx;
4. Заменить имя на LibraryModel2.edmx и нажать кнопку Add (*Добавить*);

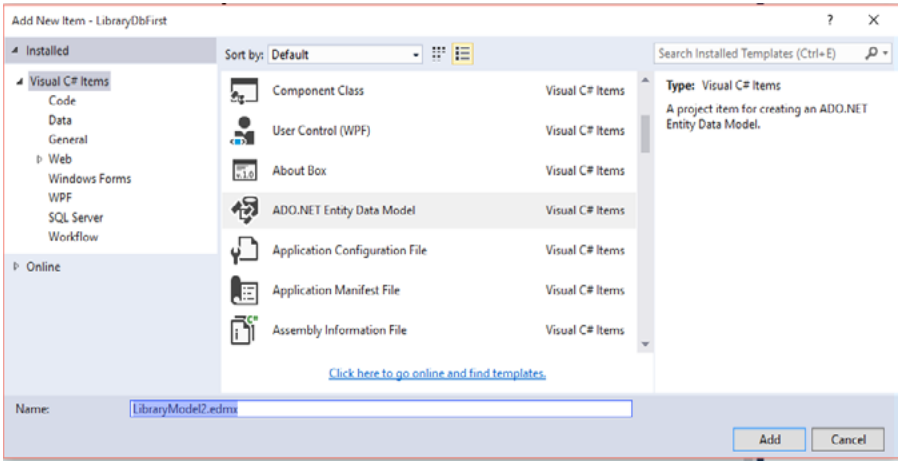

Рис. 3. Добавление модели данных

5. В появившемся мастере создания моделей EDM надо выбрать опцию Empty EF Designer model (*Пустая модель*) и нажать кнопку Finish (*Готово*).

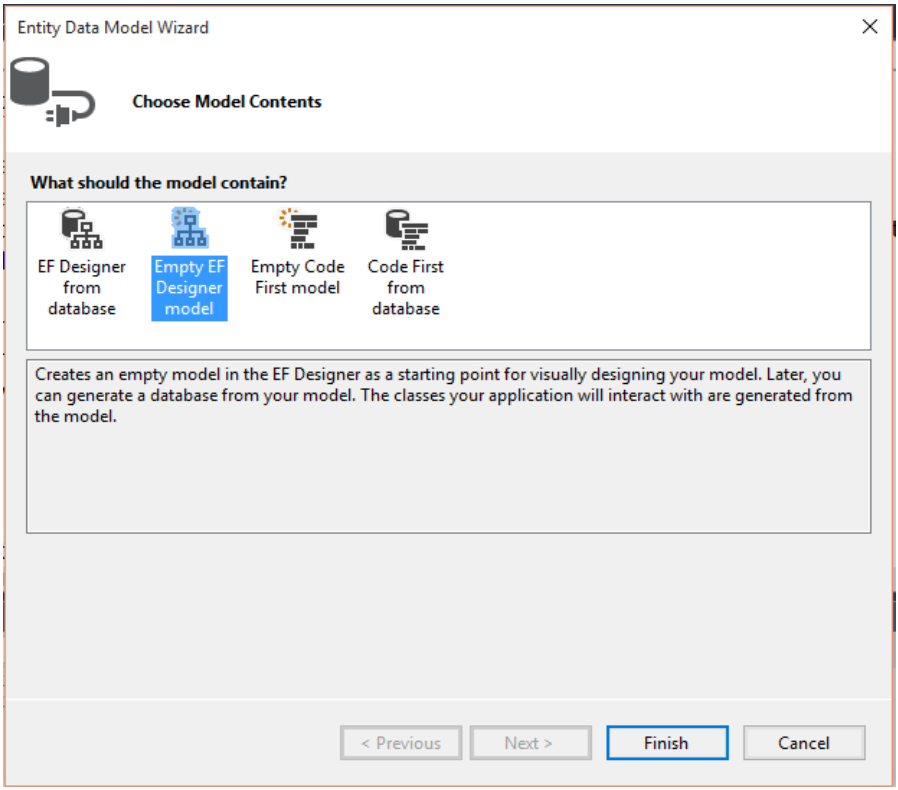

Рис. 4. Создание пустой Entity Data Model

Создание сущностей в модели

Теперь можно переходить к созданию сущностей, из которых будет состоять наша модель. Конструктор моделей позволяет создавать сущности разными способами. Можно установить мышку на свободное место в окне конструктора, нажать правую кнопку и выбрать команду Add New (*Добавить новый*). В следующем окне выбрать опцию Entity (*Сущность*). После выполнения этих действий появится мастер создания сущностей.

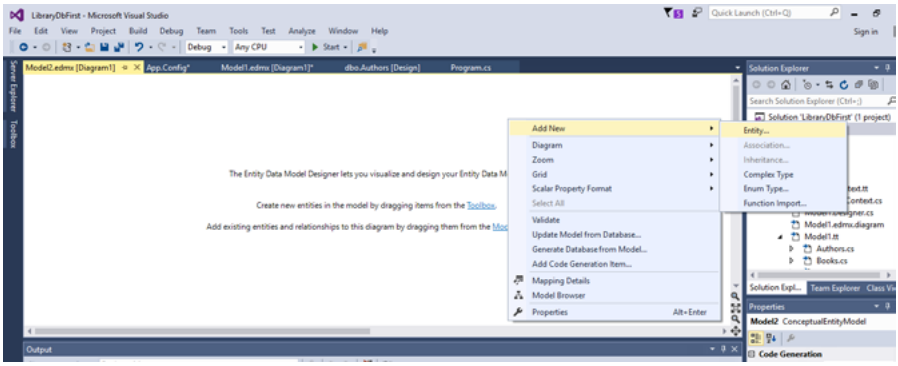

Рис. 5. Создание новой сущности

В появившемся окне мастера надо указать имя создаваемой сущности в поле Entity name — Author, в поле Entity Set тоже указать Author, а в поле Property name для ключевого свойства занести значение Id и нажать OK. Имя Id для ключевого свойства мы выбрали потому, что соглашения Entity Framework предлагают, чтобы ключевые свойства назывались либо просто Id либо «ИмяСущностиId». *(Смотрите об этих соглашениях раздел* ***Соглашения Entity Framework*** *нашего урока)*.

Т.е. мы могли бы назвать это свойство AuthorId. Регистр в данном случае роли не играет. Давайте при работе с Entity Framework будем придерживаться договоренностей о стандартах. Нажмите OK (рис. 6).

На диаграмме создаваемой модели появится графическое отображение созданной сущности с единственным пока свойством Id. Выделите эту сущность и снова выберите из контекстного меню команду Add New (*Добавить новый*). Теперь мы можем выполнять настройку

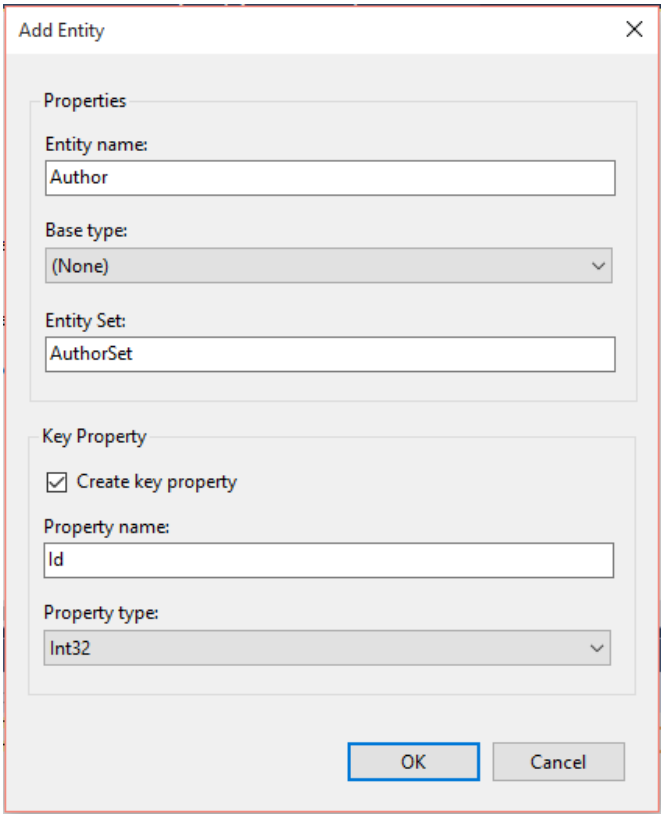

Рис. 6. Создание сущности Author

созданной сущности — добавлять скалярные свойства, свойства навигации, связи и т.п.

Чтобы правильно конструировать сущности, надо помнить, что скалярное свойство сущности соответствует полю таблицы, а навигационное свойство сущности соответствует связи. Поэтому выберите в появившемся окне опцию Scalar Property (*Скалярное свойство*), укажите имя создаваемого свойства FirstName и нажмите Enter.

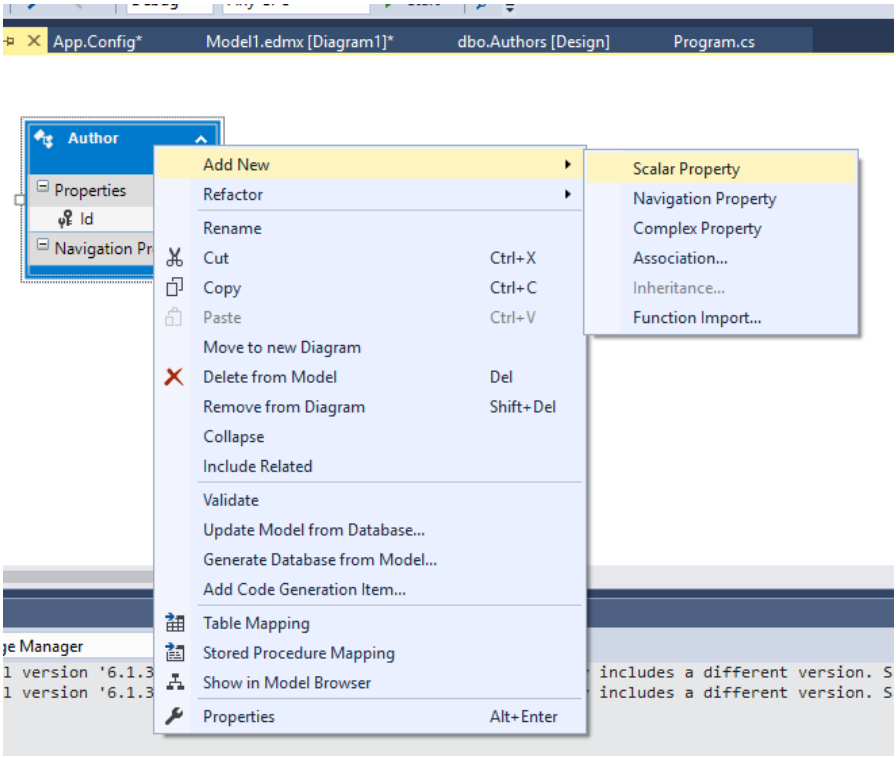

Рис. 7. Добавление свойств сущности Author

Выделите созданное свойство FirstName и выберите из контекстного меню команду Properties (*Свойства*). Под окном обозревателя решения откроется окно свойств, в котором можно указать тип свойства сущности и другие его характеристики. По умолчанию созданные скалярные свойства имеют тип string. Поэтому тип созданного свойства FirstName изменять не надо, а вот максимальную длину можно ограничить, например, 50 символами. Посмотрите внимательно на другие характеристики, которые можно изменять в этом окне свойств.

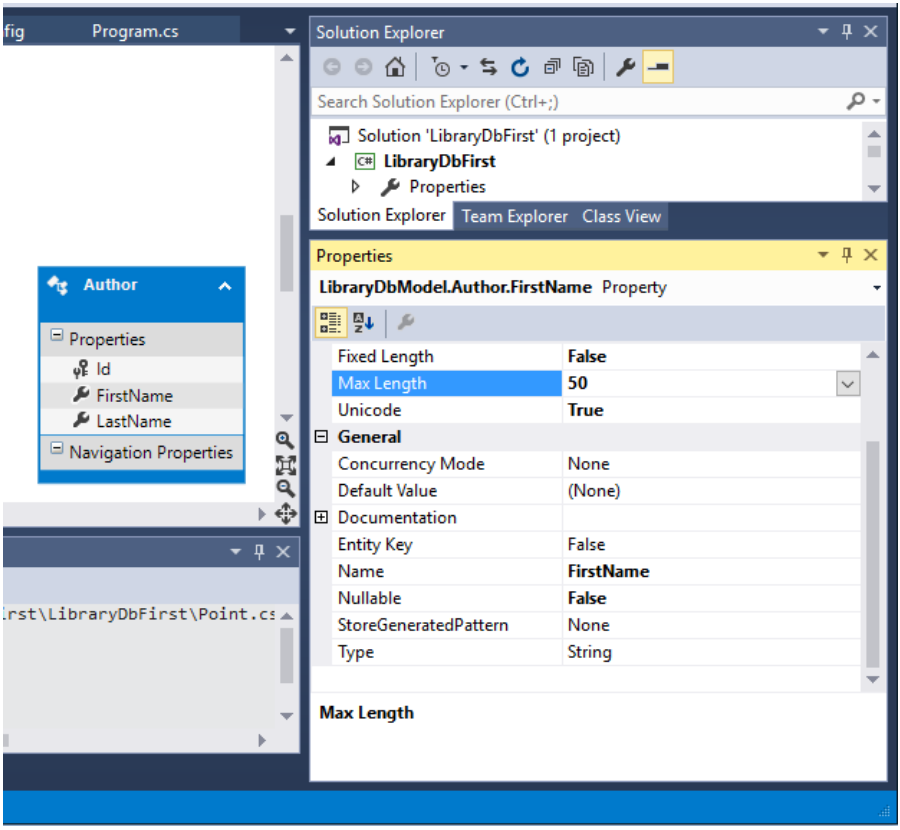

Рис. 8. Настройка свойств сущности Author

Используя только что рассмотренный подход, можно создать и настроить все требуемые сущности. Но конструктор моделей EDM предоставляет еще один способ создания структурных элементов модели. Если у вас в Visual Studio сейчас открыта диаграмма модели (узел LibraryModel2.edmx), то вы можете открыть в Visual Studio окно Toolbox, и там, в группе Entity Framework, увидеть необходимые для создания модели элементы: pointer (*указатель*), association (*ассоциация*), entity (*сущность*),

inheritance (*наследование*). Дальнейшая работа с панелью элементов проходит, как и всегда. Вы можете перетаскивать требуемые элементы из панели на диаграмму и работать с ними в графическом режиме.

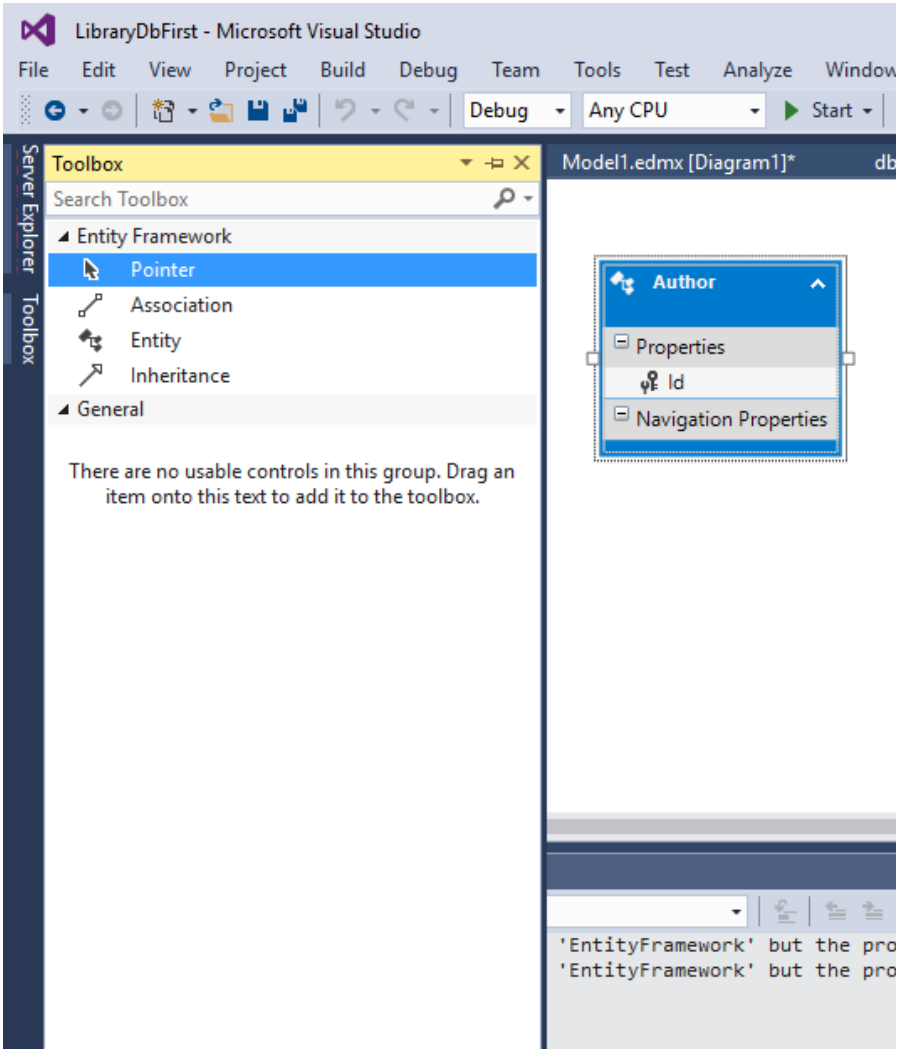

Рис. 9. Работа с Toolbox

Ознакомьтесь внимательно с конструктором моделей EDM и создайте три сущности: Author, Publisher и Book. Создавайте эти сущности с теми же свойствами, что и в первом уроке, но с двумя исключениями. Не создавайте внешние ключи AuthorId и PublisherId в сущности Book и свойства навигации. Хотя конструктор и позволяет создавать свойства навигации, вручную это делать не стоит. Внешние ключи и свойства навигации будут созданы другим способом, а именно — при создании связей.

Создание связей в модели

Когда все три сущности будут созданы и отображены на диаграмме, мы перейдем к созданию связей между ними. Выделите созданную сущность Author и снова выберите из контекстного меню команду Add New (*Добавить новый*). В появившемся окне выберите опцию Assosiation (*Ассоциация*).

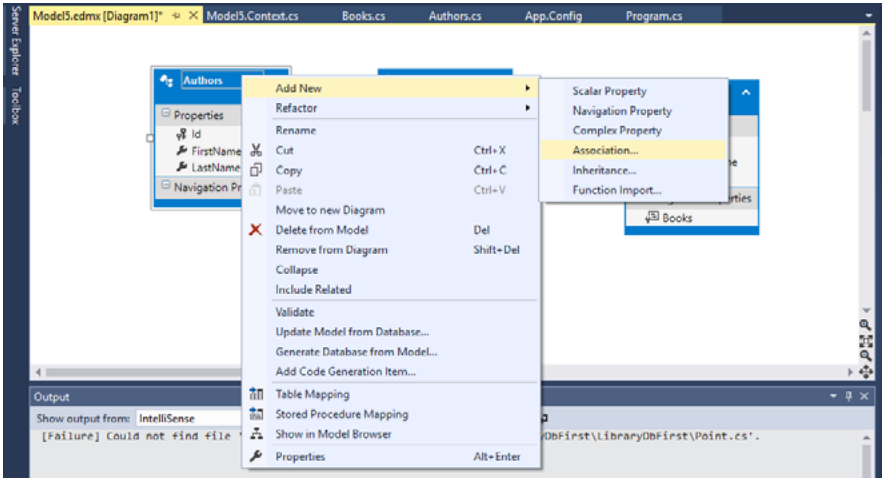

Рис. 10. Добавление связей

После выполнения этих действий перед вами откроется окно, позволяющее создавать связи между сущностями. В этом окне вы можете указать, между какими сущностями надо создать связь и какой тип связи вы хотите получить и, кроме этого, можете создать требуемый для этой связи внешний ключ. Будьте внимательны при создании связи: точно указывайте, какого типа связь вам необходима. Дело

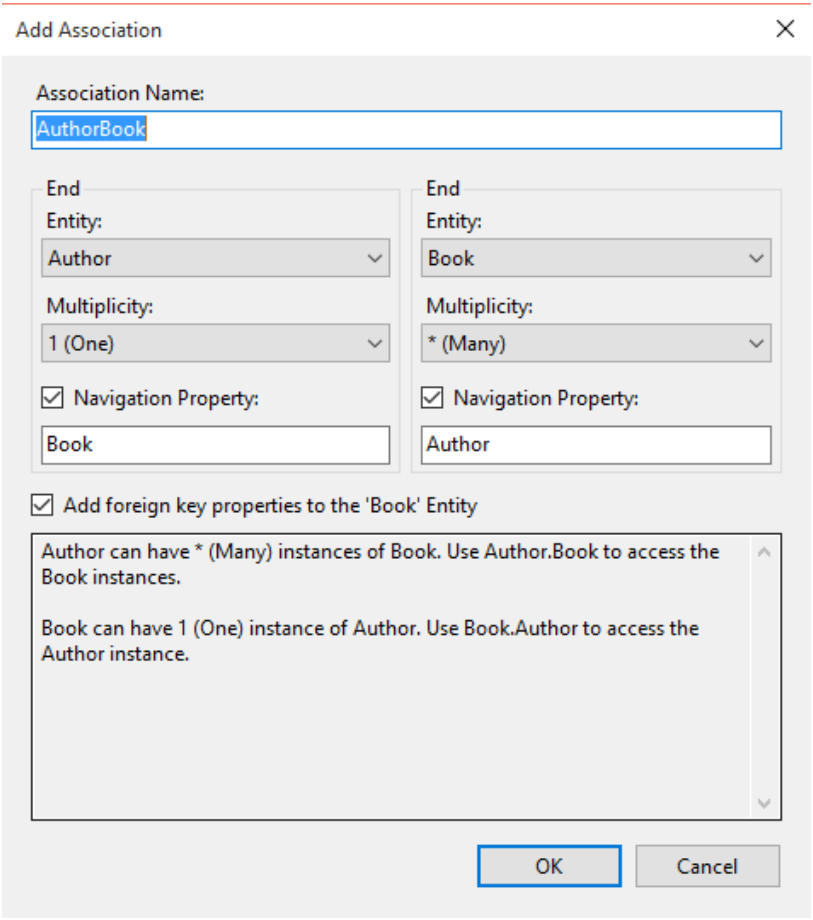

Рис. 11. Окно создания связи

в том, что свойства навигации в сущностях будут создаваться на основе связей. В нашем случае нам нужна связь типа «один ко многим»: один автор — много книг. Для создания внешнего ключа в соответствующей сущности надо выставить галочку в чекбоксе «Добавить свойства внешнего ключа…». Этот чекбокс выделен по умолчанию. Окно может выглядеть таким образом (рис. 11).

После выполнения всех этих действий будет полезно посмотреть определения созданных сущностей. Вы увидите, что в созданных сущностях появились и свойства навигации, которые мы не создавали явным образом. Свойства навигации были созданы при создании связей между сущностями. В сущности Author свойство навигации Book имеет тип коллекции, потому что мы создали связь «один ко многим» и должны иметь возможность хранить для одного автора несколько связанных сущностей Book:

```
public virtual ICollection<Book> Book { get; set; }
```

В сущности же Book свойство навигации Author имеет такой вид:

```
public virtual Author Author { get; set; }
```

Как вы видите, свойства навигации были созданы строго в соответствии со связью между сущностями. На стороне связи «один» свойство навигации представляет собой просто значение, а на стороне связи «много» это уже коллекция.

Если вы посмотрите на определение класса контекста БД, то увидите уже знакомую вам картину. Я думаю, вы не забыли, что класс контекста БД используется для подключения к БД и для передачи в БД запросов, которые мы будем создавать и выполнять. В моем приложении контекст БД выглядит таким образом:

```
public partial class LibraryModel2Container : DbContext
{
    public LibraryModel2Container()
        : base("name=LibraryModel2Container")
    {
    }

    protected override void
      OnModelCreating(DbModelBuilder modelBuilder)
    {
        throw new UnintentionalCodeFirstException();
    }

    public DbSet<Author> Author { get; set; }
    public DbSet<Book> Book { get; set; }
}
```

Вы снова видите наследование от класса DbContext, указание строки подключения в конструкторе и свойства типа DbSet<T> для всех сущностей модели. Можете посмотреть в обозревателе решения на файл конфигурации приложения, чтобы убедиться, что в нем создана соответствующая строка подключения к БД, с именем, указанным в конструкторе. Еще пару слов надо сказать об имени класса контекста БД. Это имя создается автоматически, но если оно вам не нравится, вы можете всегда

изменить его. Для этого надо снова перейти к диаграмме модели, активировать контекстное меню на свободной области диаграммы, выбрать опцию Properties (*Свойства*) и изменить значение поля Entity Container Name (*Имя контейнера сущностей*).

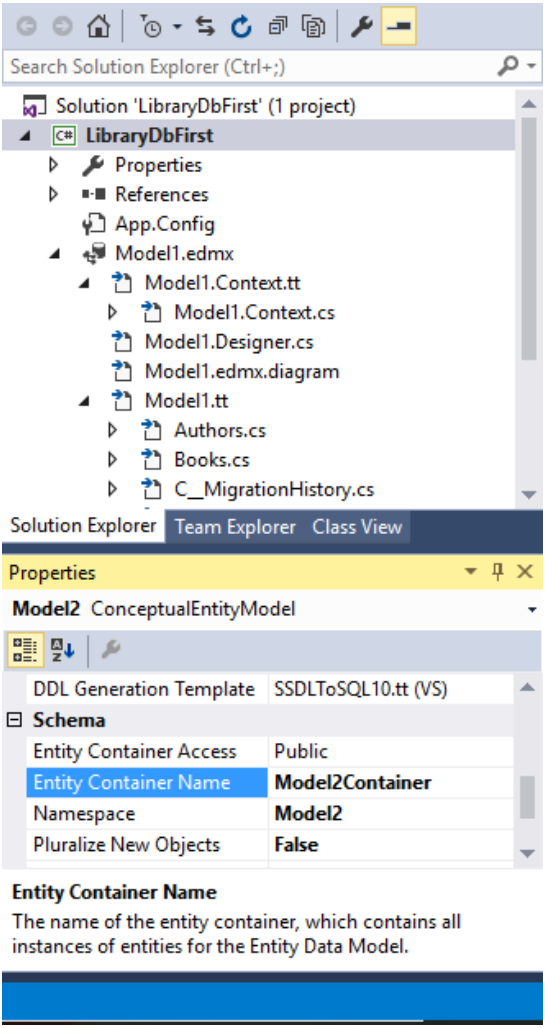

Рис. 12. Изменение имени контекста БД

Создание базы данных на основе модели

После того, как будут созданы все необходимые сущности и связи между ними, можно переходить к созданию БД на основе созданной модели. Снова установите курсор мыши на свободное пространство конструктора моделей и активируйте контекстное меню. Выберите из контекстного меню команду Generate Database from Model (*Сформировать базу данных на основе модели*). В появившемся окне мастера формирования базы данных нажмите кнопку New Connection (*Создать соединение*).

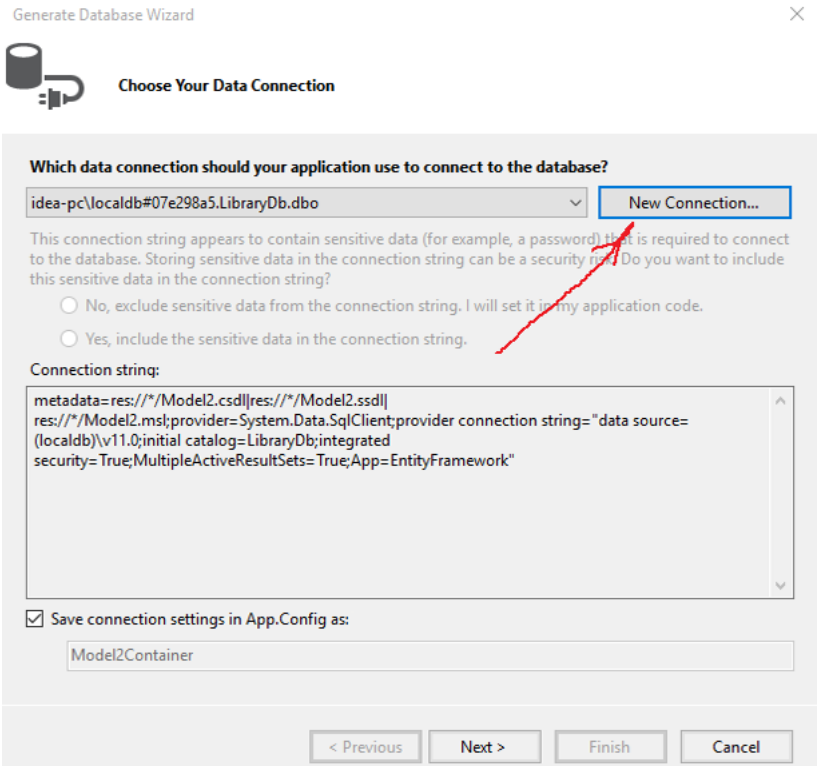

Рис. 13. Создание подключения

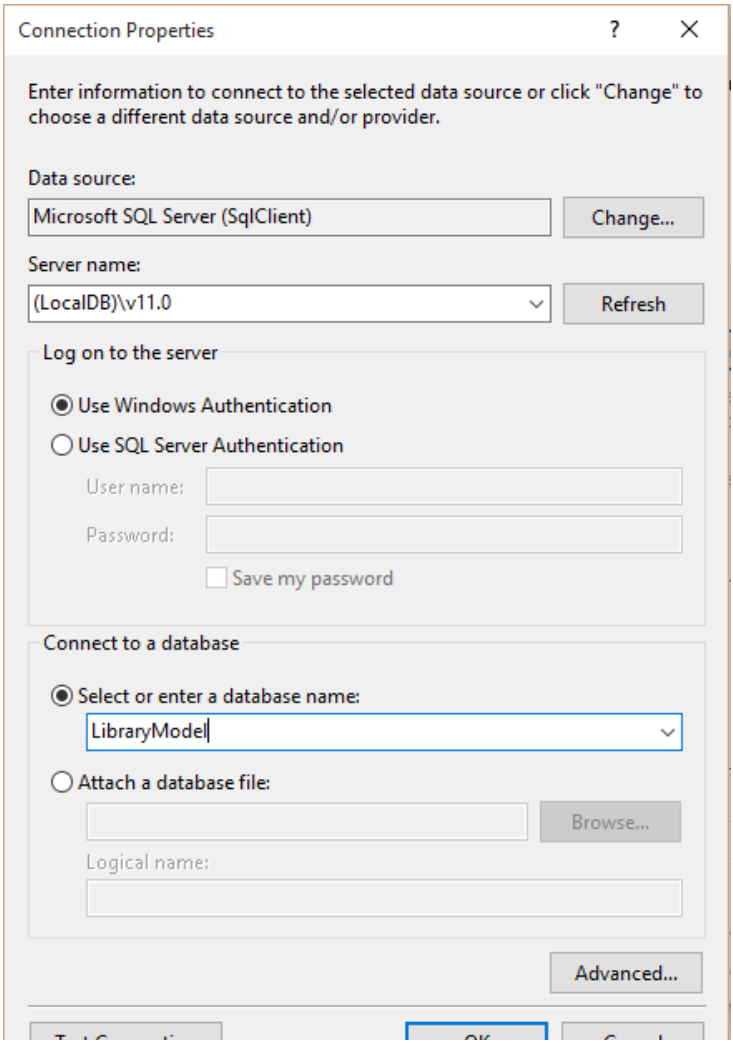

Рис. 14. Выбор сервера и имени базы данных

В следующем окне «свойства подключения» выберите необходимый источник данных. В нашем случае это будет Microsoft SQL Server. Если у вас указан другой источник, нажмите кнопку «изменить» и выберите требуемое значение.

Если в качестве источника данных выбрать Microsoft SQL Server (SqlClient), то в следующем окне «свойства подключения» надо указать имя сервера — (LocalDB)\v11.0, и имя создаваемой БД (рис. 14).

Теперь нажмите OK и в ответ на вопрос, хотите ли вы создать базу данных, ответьте утвердительно. Теперь нажмите кнопку Next (*Далее*). После этих действий конструктор моделей EDM построит скрипт, выполнение которого приведет к созданию БД на указанном сервере. Нажмите кнопку Finish (*Готово*), и скрипт будет сохранен в составе проекта и отображен в конструкторе. Теперь остается только выбрать из контекстного меню команду Execute (*Выполнить*), которая и запустит скрипт. После выполнения этого скрипта наша БД будет создана.

Проверим наше приложение и созданную БД на работоспособность. Добавим рядом с методом Main() два метода, таких же, как и в первом приложении:

```
static void AddAuthor(Author author)
{
    using (ModelFirstContainer db = new
ModelFirstContainer())
    {
        Author a = db.Author.Where((x) =>
          x.FirstName == author.FirstName &&
          x.LastName == author.LastName).
          FirstOrDefault();
        if (a == null)
        {
            db.Author.Add(author);
            db.SaveChanges();
            Console.WriteLine("New author added:" +
              author.LastName);
         }
```

```
      }
}

static void GetAllAuthors()
{
    using (ModelFirstContainer db =
       new ModelFirstContainer())
    {
        var au = db.Author.ToList();
        foreach (var a in au)
        {
             Console.WriteLine(a.FirstName + "  " +
               a.LastName);
        }
    }
}
```

Внесем соответствующие изменения в метод Main():

```
static void Main(string[] args)
          {
                        Author author = new Author {
FirstName = "Isaac", LastName = "Azimov" };
                        AddAuthor(author);
                        GetAllAuthors();
   .}
```

Теперь перестроим приложение и выполним его. Вы увидите в консольном окне результат выполнения метода GetAllAuthors(), что подтверждает также и работоспособность нашей БД, созданной по технологии «модель сначала».

Изменение модели и базы данных

Мы уже говорили о том, что технологию «модель сначала» надо использовать в том случае, когда вы точно знаете, из каких сущностей должна состоять ваша БД и какие связи между сущностями должны быть созданы, чтобы иметь возможность сконструировать требуемую БД в конструкторе модели. Но вы прекрасно понимаете, что рано или поздно возникнет необходимость изменить созданную модель. А это приведет к необходимости изменить существующую БД. Давайте рассмотрим, как надо поступать в таких случаях. Добавим в нашу БД сущность City и свяжем ее с сущностью Publisher (как будто, нам необходимо учитывать, в каком городе расположено каждое издательство). Конечно же, нам надо будет создать и связь между сущностями Publisher и City.

Добавляем новую сущность City. Для этого выделяем в обозревателе решений узел ModelFirst.edmx, чтобы отобразить диаграмму нашей БД. На диаграмме вызываем контекстное меню и выбираем команду Add New (*Добавить новый*), а далее выбираем опцию Entity (*Сущность*). Указываем имя сущности City, имя первичного ключа Id и нажимаем кнопку Add (*Добавить*). Сущность будет создана и отображена на диаграмме. Уже известным способом добавим в сущность скалярное свойство с именем CityName типа string с длиной поля 50 символов. Чтобы настроить характеристики созданного скалярного свойства, используйте опцию контекстного меню Properties (*Свойства*).

Теперь надо создать связь между новой созданной сущностью City и сущностью Publisher. Выделите сущность City и активируйте контекстное меню, затем выберите опцию Add New (Добавить новый), а далее выбирайте опцию Assosiation (*Ассоциацию*). В появившемся окне указываем, что нам нужна связь между сущностями City и Publisher, кратность со стороны сущности City должна быть «один», а со стороны Publisher — «много». Обязательно укажите на необходимость создания внешнего ключа в сущности Publisher и нажмите кнопку OK. Теперь обязательно выполните команду «сохранить», чтобы сохранить выполненные изменения в определениях сущностей. Связь будет создана, и сущности будут обновлены. Чтобы убедиться в этом, откройте определение сущности Publisher, убедитесь, что там появился внешний ключ CityId и свойство навигации City, а в определении сущности City — свойство навигации Publisher в виде коллекции:

```
public virtual ICollection<Author> Publisher { get; set; }
```

Модель мы успешно изменили. Теперь надо изменить БД, чтобы она соответствовала измененной модели. Для этого снова перейдем к диаграмме базы данных и выберем из контекстного меню опцию Generate Database from Model (*Сформировать базу данных на основе модели*). Вновь повторятся все действия, которые мы наблюдали при создании нашей БД. Вновь утвердительно отвечайте на все вопросы в ходе создания скрипта. Когда скрипт будет создан, выполните его. Но вы должны понимать

следующее: ***при изменении БД DDL скрипт, созданный Entity Framework, удаляет старую БД и взамен создает новую***. При этом, если в БД уже были данные, они будут уничтожены. Если вам необходимо сохранить данные при изменении БД, вы должны позаботиться об этом самостоятельно. Можно, например, отредактировать созданный скрипт или где-нибудь сохранить данные.

Оценка технологии «модель сначала»

Попробуем проанализировать проделанную работу и полученные результаты. Мы создали работоспособное приложение и базу данных, не написав ни одного SQL запроса. Здорово? Давайте обсудим. Технология «модель сначала» может применяться только в случае простых приложений и простых баз данных. В тех случаях, когда вы уверены в том, что вам не придется изменять созданную БД.

Вы уже увидели «узкие места» такого подхода: отсутствие полного контроля над процессом создания сущностей, сложность изменения сущностей и базы данных. А что же можно отнести к плюсам этого подхода? Возможно, тот факт, что можно создать БД, не умея писать SQL запросы, кто-то отнесет к плюсам? Но неумение писать SQL запросы вряд ли можно считать достоинством.

Домашнее задание

Модифицируйте Windows Forms приложение, созданное в качестве домашнего задания в уроке 1, добавив в него возможность импортировать данные о книгах из какого-либо файла. Это может быть xml-файл, или даже просто текстовый файл. Подготовьте или найдите такой файл, чтобы он содержал информацию минимум о нескольких сотнях книг. Импортируйте информацию из этого файла в БД приложения. Теперь создайте два метода, выбирающие из БД все данные разными способами: один — с помощью lazy loading, второй — eager loading. Сравните время работы обоих методов, создав какой-либо простой механизм профилирования.

Для нашей задачи достаточной будет точность на уровне милисекунд. Поэтому можно использовать такой способ вычисления числа милисекунд, прошедших от момента start до момента end:

```
try
{
    long start = DateTime.Now.Ticks;
    //some actions are performed here
    long end = DateTime.Now.Ticks;
    long tick = end - start; //number of ticks
    long milliseconds = tick / TimeSpan.
     TicksPerMillisecond; //number of milliseconds
}
catch (Exception ex)
{
    //handling exceptions
}
```