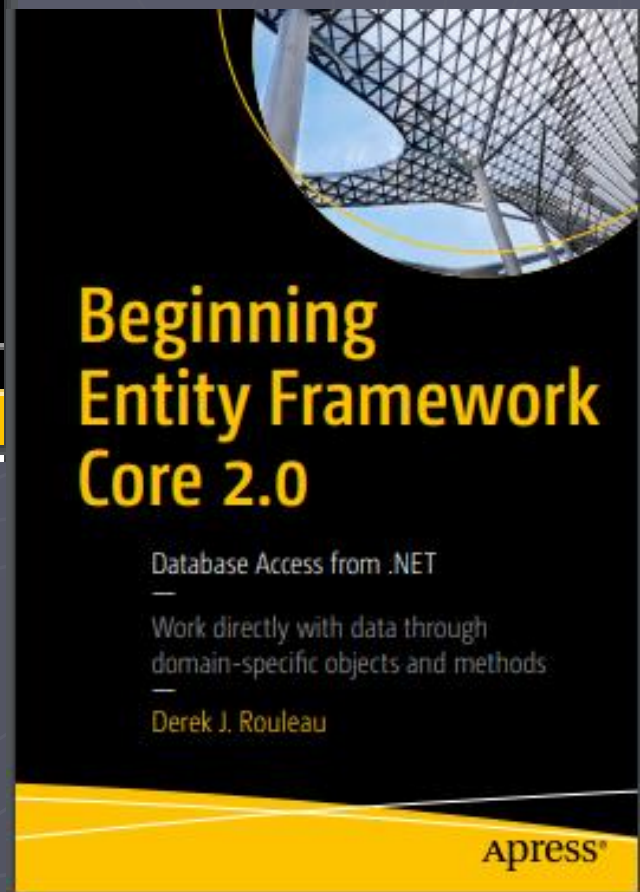
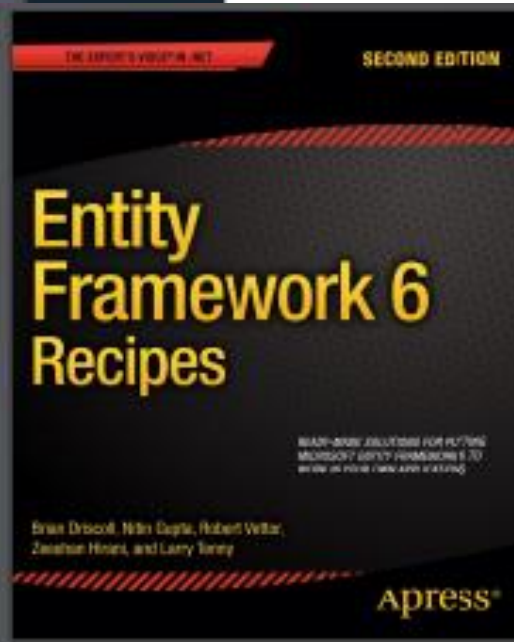
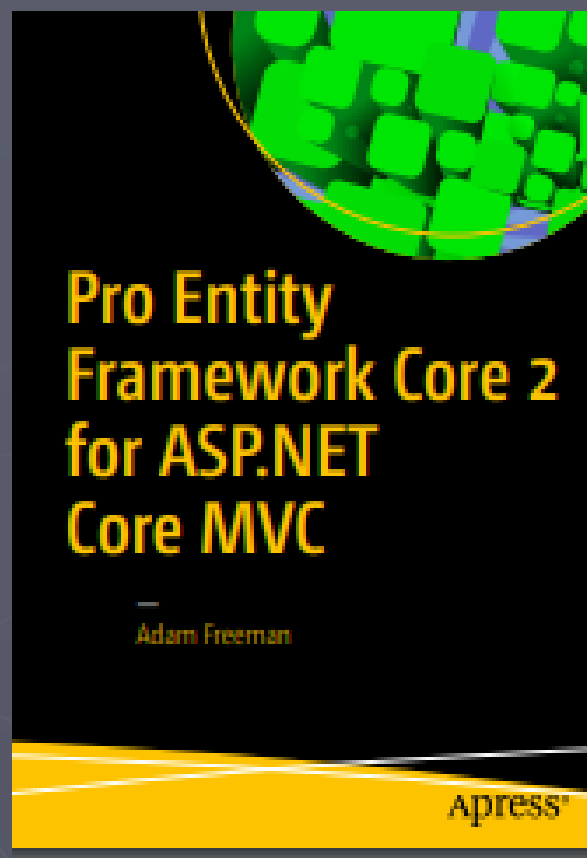
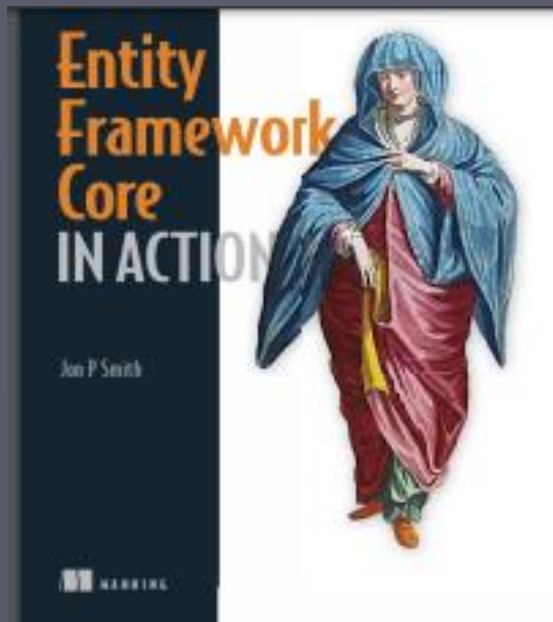


Entity Framework





Object-relational mapping

- Создание объектной модели по БД
 - Создание схемы БД по объектной модели
 - Выполнение запросов к БД с помощью OOAPI
 - CRUD – create, retrieve(read), update, delete
- ORM-системы автоматически генерируют SQL запросы для выполнения операций над данными при вызове OO

Преимущества

- ▶ Меньший объем кода
- ▶ Автоматическое использование паттернов проектирование (слой доступа данных) – улучшает дизайн
- ▶ Код хорошо протестирован (индустриальные стандарты – LINQ)

- ▶ **Entity Framework** - объектно-ориентированная технология на базе фреймворка .NET для работы с данными.
- ▶ Entity Framework - технология ORM - сопоставления сущностей C# с таблицами в базе данных.

ORM (object-relational mapping) - отображения данных на реальные объекты



- ▶ Entity Framework - 1.0 - 2008
- ▶ Entity Framework - 4.0 - 2010
- ▶ Entity Framework - 6.4 - 2013

▶ Entity Framework Core 3.1 -2019

объектно-ориентированная, легковесная ,
расширяемая и кроссплатформенная

(EF 6.4 vs Core - <https://docs.microsoft.com/en-us/ef/efcore-and-ef6/>)

Поддержка провайдеров: для MS SQL
Server/SQL Azure, Azure Cosmos DB, SQLite,
PostgreSQL, MySQL и др.

Сущность (entity)

- ▶ Набор данных, ассоциированных с определенным объектом
- ▶ Обладает свойствами
- ▶ Ключ – набор свойств, которые уникально определяют эту сущность.
- ▶ Связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим

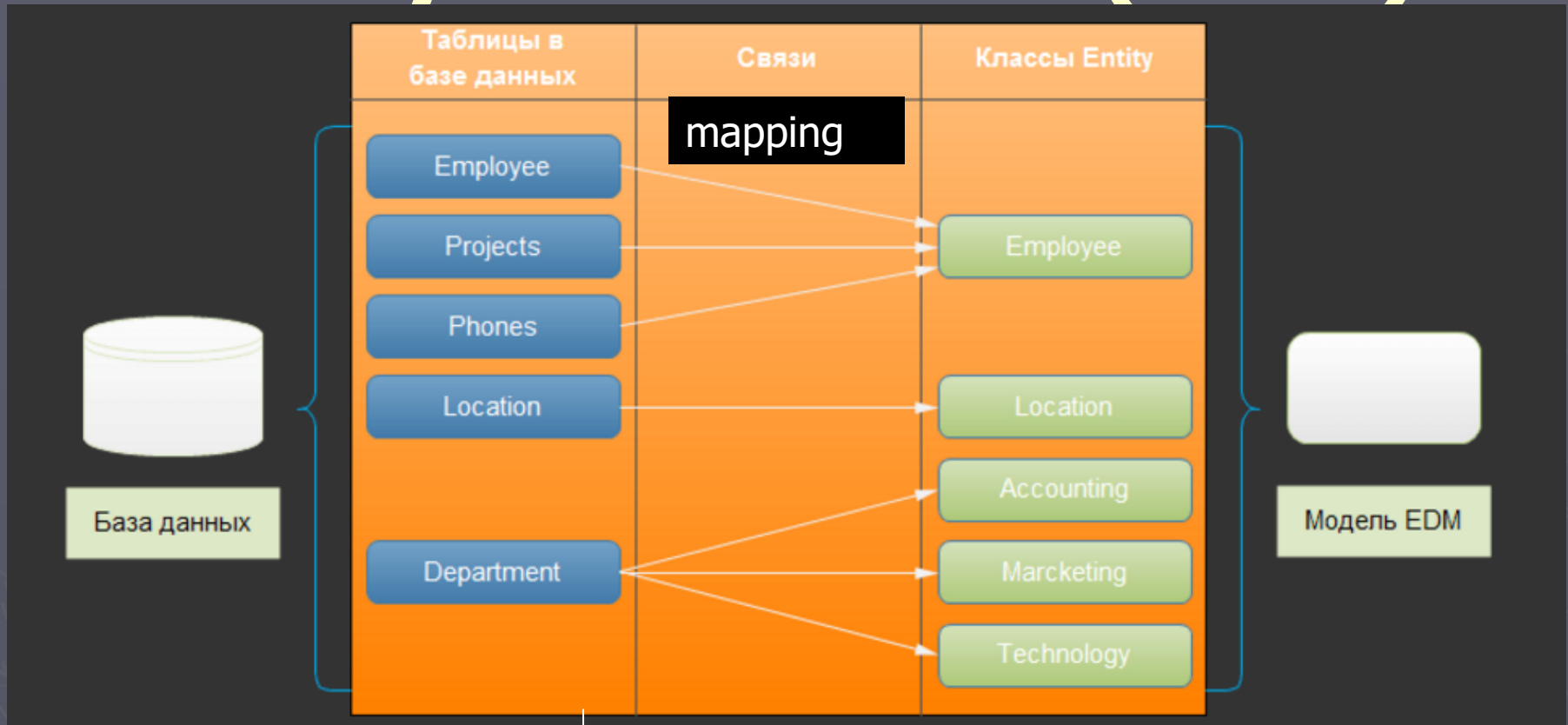
Entity Data Model

Модель предметной области
(концептуальная) – описание
объектов



Модель данных –
описание таблиц
и зависимостей

Entity Data Model (EDM)



Уровень хранилища

Удаленный
описывается **Store
Schema Definition
Language (SSDL)**

Уровень
сопоставления

Связующий
**Mapping
Schema
Language (MSL)**

Концептуальный
уровень

Рабочий
описывается
**Conceptual Schema
Definition Language
(CSDL)**

► Manage NuGet Packages...: управление пакетами

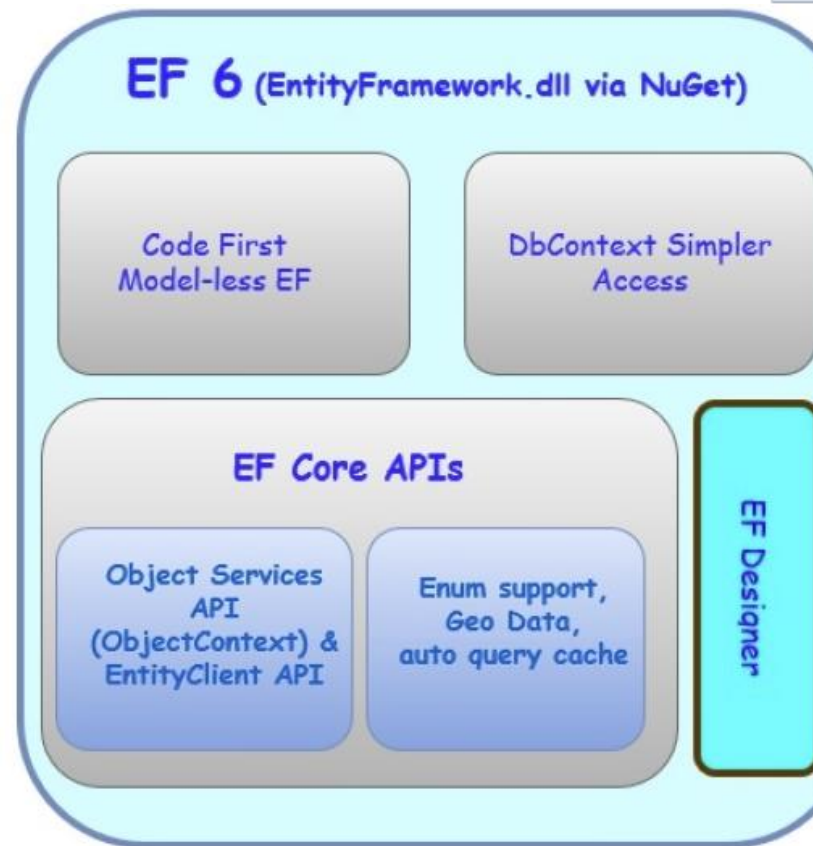
NuGet — решение → X MainWindow.xaml MainWindow.xaml.cs

Обзор Установлено Обновления Консолидировать

Entity frame X ↕ ☐ Включить предварительные версии

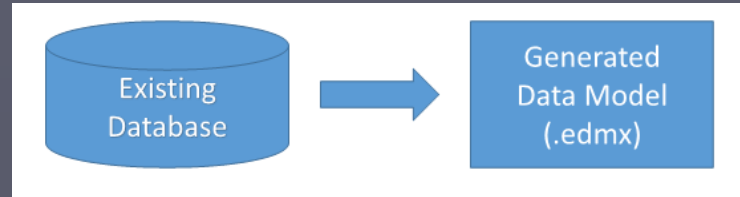
Источник пакета: nuget.org

- Microsoft.EntityFrameworkCore** ✓ автор: Microsoft, Скачиваний: 96,1M
Entity Framework Core is a lightweight and extensible version of the popular Entity Framework access technology.
- Microsoft.EntityFrameworkCore.Relational** ✓ автор: Microsoft, Скачиваний: 88,8M
Shared Entity Framework Core components for relational database providers.
- EntityFramework** ✓ автор: Microsoft, Скачиваний: 88,8M
Entity Framework 6 (EF6) is a tried and tested object-relational mapper for .NET with many feature development and stabilization.
- Microsoft.EntityFrameworkCore.Abstractions** ✓ автор: Microsoft, Скачиваний: 88,8M
Provides abstractions and attributes that are used to configure Entity Framework Core
- Microsoft.EntityFrameworkCore.Analyzers** ✓ автор: Microsoft, Скачиваний: 88,8M
CSharp Analyzers for Entity Framework Core.



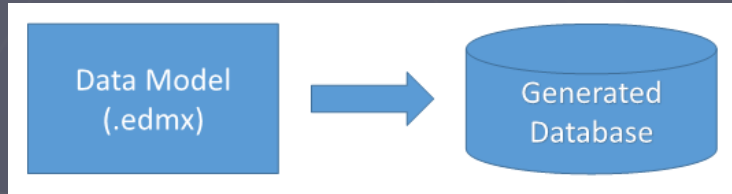
Подходы к проектированию

► *Database-First*



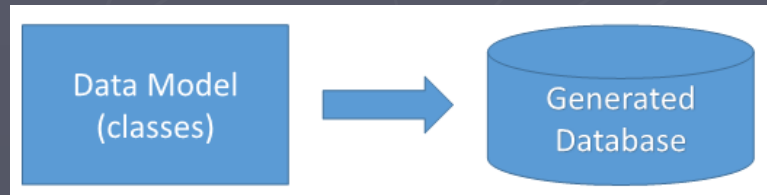
создание базы данных -> генерация EDMX-модель

► *Model-First* (только EF 6)



создание графической модели EDMX -> генерация базы данных

► *Code-First*



настройка классов C# объектной модели

- 1) генерация сущностных классов из существующей базы данных
- 2) создание базы данных из созданной вручную модели объектов C#

подход Model-First (EF6)

Добавить новый элемент - EE_CF

Установленные

- Элементы Visual C#
 - Windows Forms
 - WPF
 - Веб
 - Данные
 - Код
 - Общий
 - Reporting
 - SQL Server
 - Workflow
 - Графика
- В сети

Сортировка по: По умолчанию

XML-файл	Элементы Visual C#
XSLT-файл	Элементы Visual C#
База данных, основанная на службах	Элементы Visual C#
Генератор EF 5.x DbContext	Элементы Visual C#
Генератор EF 6.x DbContext	Элементы Visual C#
Классы LINQ to SQL	Элементы Visual C#
Модель ADO.NET EDM	Элементы Visual C#
Набор данных	Элементы Visual C#
Схема XML	Элементы Visual C#

[Щелкните здесь для поиска шаблонов в Интернете.](#)

Установлено: Шаблоны — поиск (Ctrl F)

Тип: Элементы Visual C#

Элемент проекта для создания модели EDM ADO.NET.

Мастер моделей EDM



Выбор содержимого модели

Что должна содержать модель?

Конструктор EF из базы данных

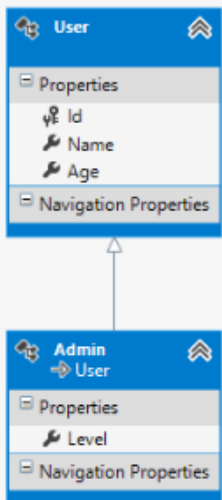
Пустая модель конструктора EF

Пустая модель Code First

Code First из базы данных

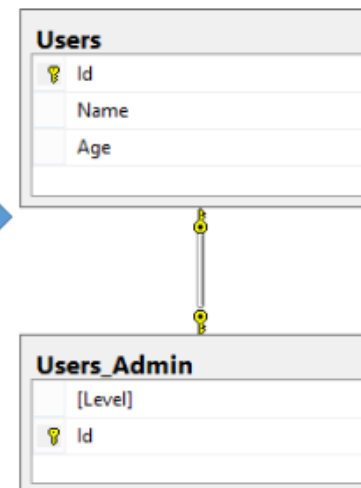
Создает пустую модель в конструкторе EF в качестве отправной точки для визуального проектирования модели. Затем из модели можно будет сформировать базу данных. Классы, с которыми будет взаимодействовать ваше приложение, формируются из этой модели.

Добавляем данные к проекту

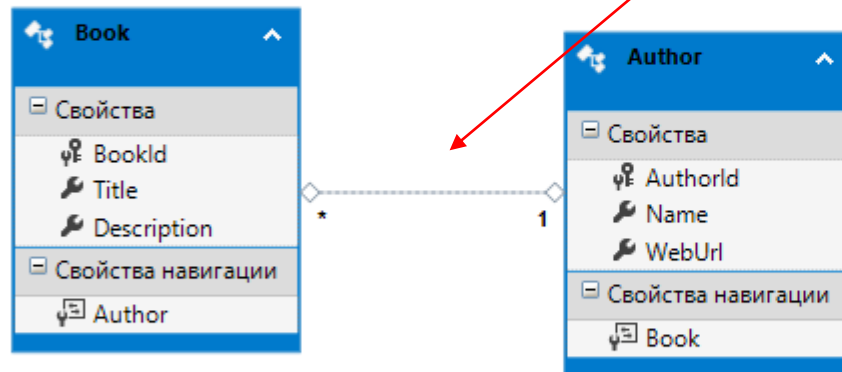


```
public partial class User
{
    public int Id { get; set; }
    public string Name { get; set; }
    public DateTime? Age { get; set; }
}

public partial class Admin : User
{
    public string Level { get; set; }
}
```



Определяем сущности и
отношения между ними



Построение модели
в дизайнере

Обозреватель решений

Обозреватель решений — поиск (Ctrl+;)

Решение "EF6Demo" (проекты: 1 из 1)

EF6Demo

Properties

Ссылки

App.config

App.xaml

MainWindow.xaml

Model1.edmx

Model1.Designer.cs

Model1.edmx.diagram

Панель элементов

Поиск по панели элементов

Entity Framework

Указатель

Ассоциация

Сущность

Наследование

Общие

В этой группе нет элементов управления. Перетащите элемент в эту область, чтобы добавить его в панель элементов.

User

- Свойства
 - Id
 - Name
 - Login
 - Password
- Свойства навигации
 - Actor

- Добавить новый
- Диаграмма
- Масштаб
- Сетка
- Формат скалярных свойств
- Выбрать все
- Проверить
- Обновить модель из базы данных...
- Сформировать базу данных на основе модели...
- Добавить элемент создания кода...
- Сведения о сопоставлении
- Браузер моделей
- Включить загрузку упрощенного решения
- Свойства

Обозреватель решений — поиск (Ctrl+ж)

(проектов: 1)

g
cs
mer
merContext

Designer.cs

dmx

dmx.sql

dmx

Team Explorer

Alt+BВOD

Model1 ConceptualEntityModel



Имя схемы базы данных
Рабочий процесс создания базы данных
Шаблон создания DDL

Схема

Доступ к контейнеру сущностей

Доступ к контейнеру сущностей

Модификатор доступа контейнера сущностей EDM.

граничения

Генерация базы данных

Мастер формирования базы данных

Выберите версию

Какую версию Entity Framework вы хотите использовать?

☒ Entity Framework 6.x

☐ Entity Framework 5.0

!

Можно также установить и использовать другие версии Entity Framework.
[Получить дополнительные сведения об этом](#)

Мастер формирования базы данных

Выбор подключения к данным

Какое подключение к данным будет использоваться приложением для подключения к базе данных?

books.mdf

Создать соединение...

возможно, эта строка подключения содержит конфиденциальные данные (например, пароль), которые требуются для подключения к базе данных. Хранение конфиденциальных данных в строке подключения может представлять угрозу безопасности. Включить конфиденциальные данные в строку подключения?

☐ Нет, исключить конфиденциальные данные из строки подключения. Они будут заданы в коде приложения.

☐ Да, включить конфиденциальные данные в строку подключения.

Строка подключения:

metadata=res://*/Model1.csdl|res://*/Model1.ssdl|res://*/Model1.msl;provider=System.Data.SqlClient;provider connection string="data source=(LocalDB)\MSSQLLocalDB;attachdbfilename=C:\Users\npats\Documents\books.mdf;integrated security=True;connect timeout=30;MultipleActiveResultSets=True;App=EntityFramework"

☒ Сохранить параметры соединения в App.Config как:

Model1Container

< Назад

Далее >

Готово

Отмена

Выбираем
базу данных

Мастер формирования базы данных



Сводка и настройки

Сохранить DDL как: Model1.edmx.sql

DDL

```
-----  
-- Entity Designer DDL Script for SQL Server 2005, 2008, 2012 and  
-----  
-- Date Created: 05/03/2020 20:17:05  
-- Generated from EDMX file: C:\NATALLIA\лекции\ООПС#\ОО  
\Proj_EF2020\EF6Demo\EF6Demo\Model1.edmx  
-----  
  
SET QUOTED_IDENTIFIER OFF;  
GO  
USE [books];  
GO  
IF SCHEMA_ID(N'dbo') IS NULL EXECUTE(N'CREATE SCHEMA [dbo]');  
GO
```

< Назад

Далее >

Просматриваем скрипт,
который позволит
создать таблицы и
наложение
ограничений на ключи

```
<connectionStrings>  
  <add name="Model1Container"  
— connectionString="metadata=res://*/M  
odel1.csdl|res://*/Model1.ssdl|res://  
/*/Model1.msl;provider=System.Data.S  
qlClient;provider connection  
string=&quot;data  
source=(LocalDB)\MSSQLLocalDB;attach  
dbfilename=C:\Users\npats\Documents\  
books.mdf;integrated  
security=True;connect  
timeout=30;MultipleActiveResultSets=  
True;App=EntityFramework&quot;;"  
providerName="System.Data.EntityClie  
nt" />  
</connectionStrings>
```



```
g="false" hostspecific="true"#>
S.ttinclude"#><#@

l1.edmx";
Transformation.Create(this);
ols(this);
);
(code, ef, textTransform.Errors);
der(textTransform.Host, textTransform.Errors);
ateEdmItemCollection(inputFile);
ModelNamespace(inputFile);
odeStringGenerator(code, typeMapper, ef);

OfType<EntityContainer>().FirstOrDefault();
```

Генерирует
контекст

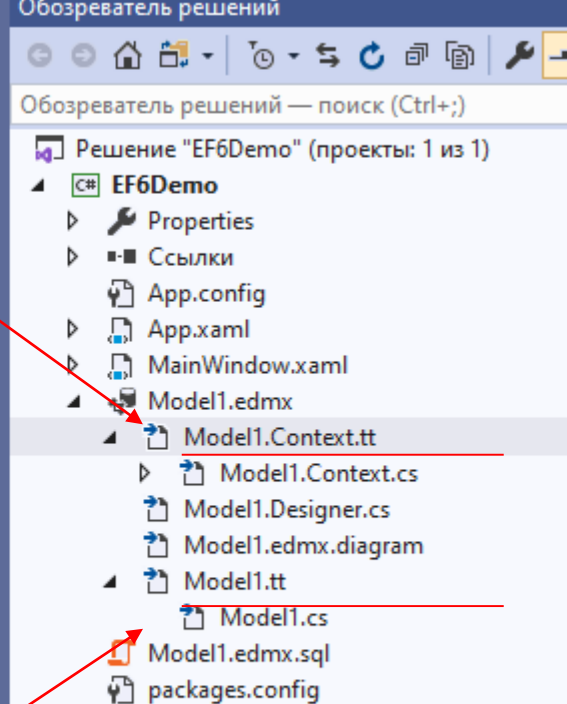
```
sourceString("Template_GeneratedCodeCommentLine1")#>
```

```
sourceString("Template_GeneratedCodeCommentLine2")#>
```

```
public partial class Model1Container : DbContainer
```

```
{
    public Model1Container()
        : base("name=Model1Container")
    {
    }
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        throw new UnintentionalCodeFirstException();
    }
    public virtual DbSet<Book> BookSet { get; set; }
    public virtual DbSet<Author> AuthorSet { get; set; }
}
```

Скрипт на t4 – выполнив
который будут
сгенерированы сущности



Model1.Designer.cs Model1.tt Model1.cs Model1.Context.cs Model1.Context.tt

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <edmx:Edmx Version="3.0" xmlns:edmx="http://schemas.microsoft.com/ado/2009/11/edmx">
3   <!-- EF Runtime content -->
4   <edmx:Runtime>
5     <!-- SSDL content -->
6     <edmx:StorageModels>
7       <Schema Namespace="Model1.Store" Alias="Self" Provider="System.Data.SqlClient" ProviderManifestToken="2012" xmlns:store=
8     <EntityContainer Name="Model1StoreContainer">
9       <EntitySet Name="BookSet" EntityType="Model1.Store.BookSet" store:Type="Tables" Schema="dbo" />
10      <EntitySet Name="AuthorSet" EntityType="Model1.Store.AuthorSet" store:Type="Tables" Schema="dbo" />
11      <AssociationSet Name="AuthorBook" Association="Model1.Store.AuthorBook">
12        <End Role="Author" EntitySet="AuthorSet" />
13        <End Role="Book" EntitySet="BookSet" />
14      </AssociationSet>
15    </EntityContainer>
16    <EntityType Name="BookSet">
17      <Key>
18        <PropertyRef Name="BookId" />
19      </Key>
20      <Property Name="BookId" Type="int" StoreGeneratedPattern="Identity" Nullable="false" />
21      <Property Name="Title" Type="nvarchar(max)" Nullable="false" />
22      <Property Name="Description" Type="nvarchar(max)" Nullable="false" />
23      <Property Name="Author_AuthorId" Type="int" Nullable="false" />
24    </EntityType>
25    <EntityType Name="AuthorSet">

```

Обозреватель решений

Обозреватель решений

Решение "EF6Demo"

- EF6Demo
 - Properties
 - Ссылки
 - App.config
 - App.xaml
 - MainWindow.x
 - Model1.edmx
 - Model1.Co
 - Model1
 - Mod
 - Model1.Des
 - Model1.ed
 - Model1.tt
 - Model1
 - Model1.edmx.
 - packages.conf

Открыть как xml
кодогенерация

```

3   <!-- EF Runtime content -->
4   <edmx:Runtime>
5     <!-- SSDL content -->
6     <edmx:StorageModels>...</edmx:StorageModels>
7     <!-- CSDL content -->
8     <edmx:ConceptualModels>
9       <Schema xmlns="http://schemas." Namespace="Model1" Alias="Se
10     </edmx:ConceptualModels>
11     <!-- C-S mapping content -->
12     <edmx:Mappings>...</edmx:Mappings>
13   </edmx:Runtime>
14   <!-- EF Designer content (DO NOT EDIT MANUALLY BELOW HERE) -->
15   <edmx:Designer xmlns="http://schemas.">...</edmx:Designer>
16 </edmx:Edmx>

```

```

<edmx:Runtime>
  <!-- SSDL content -->
  <edmx:StorageModels>
    <Schema Namespace="Model1.Store" Alias="Self" Provider="System.Data.SqlClient" ProviderManifest="Model1.Store.Model1Manifest" />
  <EntityContainer Name="Model1StoreContainer">
    <EntityType Name="BookSet" EntityType="Model1.Store.BookSet" store:Type="Tables" Schema="dbo" />
    <EntityType Name="AuthorSet" EntityType="Model1.Store.AuthorSet" store:Type="Tables" Schema="dbo" />
    <AssociationSet Name="AuthorBook" Association="Model1.Store.AuthorBook">
      <End Role="Author" EntitySet="AuthorSet" />
      <End Role="Book" EntitySet="BookSet" />
    </AssociationSet>
  </EntityContainer>
  <EntityType Name="BookSet">
    <Key>
      <PropertyRef Name="BookId" />
    </Key>
    <Property Name="BookId" Type="int" StoreGeneratedPattern="Identity" Nullable="false" />
    <Property Name="Title" Type="nvarchar(max)" Nullable="false" />
    <Property Name="Description" Type="nvarchar(max)" Nullable="false" />
    <Property Name="Author_AuthorId" Type="int" Nullable="false" />
  </EntityType>
  <EntityType Name="AuthorSet">
    <Key>
      <PropertyRef Name="AuthorId" />
    </Key>
    <Property Name="AuthorId" Type="int" StoreGeneratedPattern="Identity" Nullable="false" />
    <Property Name="Name" Type="nvarchar(max)" Nullable="false" />
    <Property Name="WebUrl" Type="nvarchar(max)" Nullable="false" />
  </EntityType>
  <Association Name="AuthorBook">
    <End Role="Author" Type="Model1.Store.AuthorSet" Multiplicity="1" />
    <End Role="Book" Type="Model1.Store.BookSet" Multiplicity="*" />
  </Association>

```

Набор ассоциаций

Набор таблиц

Набор свойств

```

<!-- CSDL content -->
<edmx:ConceptualModels>
  <Schema xmlns="http://schemas.microsoft.com/ado/2009/11/edm" xmlns:cg="http://schemas.microsoft.com/ado/2009/11/edm"
    <EntityContainer Name="Model1Container" annotation:LazyLoadingEnabled="true">
      <EntitySet Name="BookSet" EntityType="Model1.Book" />
      <EntitySet Name="AuthorSet" EntityType="Model1.Author" />
      <AssociationSet Name="AuthorBook" Association="Model1.AuthorBook">
        <End Role="Author" EntitySet="AuthorSet" />
        <End Role="Book" EntitySet="BookSet" />
      </AssociationSet>
    </EntityContainer>
    <EntityType Name="Book">
      <Key>
        <PropertyRef Name="BookId" />
      </Key>
      <Property Name="BookId" Type="Int32" Nullable="false" annotation:StoreGeneratedPattern="Identity" />
      <Property Name="Title" Type="String" Nullable="false" />
      <Property Name="Description" Type="String" Nullable="false" />
      <NavigationProperty Name="Author" Relationship="Model1.AuthorBook" FromRole="Book" ToRole="Author" />
    </EntityType>
    <EntityType Name="Author">
      <Key>
        <PropertyRef Name="AuthorId" />
      </Key>
      <Property Name="AuthorId" Type="Int32" Nullable="false" annotation:StoreGeneratedPattern="Identity" />
      <Property Name="Name" Type="String" Nullable="false" />
      <Property Name="WebUrl" Type="String" Nullable="false" />
      <NavigationProperty Name="Book" Relationship="Model1.AuthorBook" FromRole="Author" ToRole="Book" />
    </EntityType>
    <Association Name="AuthorBook">
      <End Type="Model1.Author" Role="Author" Multiplicity="1" />
      <End Type="Model1.Book" Role="Book" Multiplicity="*" />
    </Association>
  </Schema>
</edmx:ConceptualModels>

```

Набор сущностей

Набор ассоциаций

Набор свойств

связывает сущности, указанные в разделах SSDL и CSDL, и определяет как будут отображаться данные из базы данных на классы .NET

```
<!-- C-S mapping content -->
<edmx:Mappings>
  <Mapping Space="C-S" xmlns="http://schemas.microsoft.com/ado/edmx/mapping/cs" />
  <EntityContainerMapping StorageEntityContainer="AdventureWorksLT" Model="AdventureWorksLT" />
  <EntitySetMapping Name="BookSet">
    <EntityTypeMapping TypeName="IsTypeOf(Model1.Book)">
      <MappingFragment StoreEntitySet="BookSet">
        <ScalarProperty Name="BookId" ColumnName="BookId" />
        <ScalarProperty Name="Title" ColumnName="Title" />
        <ScalarProperty Name="Description" ColumnName="Description" />
      </MappingFragment>
    </EntityTypeMapping>
  </EntitySetMapping>
  <EntitySetMapping Name="AuthorSet">
    <EntityTypeMapping TypeName="IsTypeOf(Model1.Author)">
      <MappingFragment StoreEntitySet="AuthorSet">
        <ScalarProperty Name="AuthorId" ColumnName="AuthorId" />
        <ScalarProperty Name="Name" ColumnName="Name" />
        <ScalarProperty Name="WebUrl" ColumnName="WebUrl" />
      </MappingFragment>
    </EntityTypeMapping>
  </EntitySetMapping>
  <AssociationSetMapping Name="AuthorBook" TypeName="Model1.AuthorBook" StoreEntitySet="BookSet" />
  <EndProperty Name="Author">
    <ScalarProperty Name="AuthorId" ColumnName="Author_AuthorId" />
  </EndProperty>
  <EndProperty Name="Book">
    <ScalarProperty Name="BookId" ColumnName="BookId" />
  </EndProperty>
</edmx:Mappings>
```



Имя свойства и
имя колонки

```
[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",  
"CA2214:DoNotCallOverridableMethodsInConstructors")]
```

```
public Author()  
{  
    this.Book = new HashSet<Book>();  
}
```

```
    public int AuthorId { get; set; }  
    public string Name { get; set; }  
    public string WebUrl { get; set; }
```

```
[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",  
"CA2227:CollectionPropertiesShouldBeReadOnly")]
```

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

```
public partial class Book
```

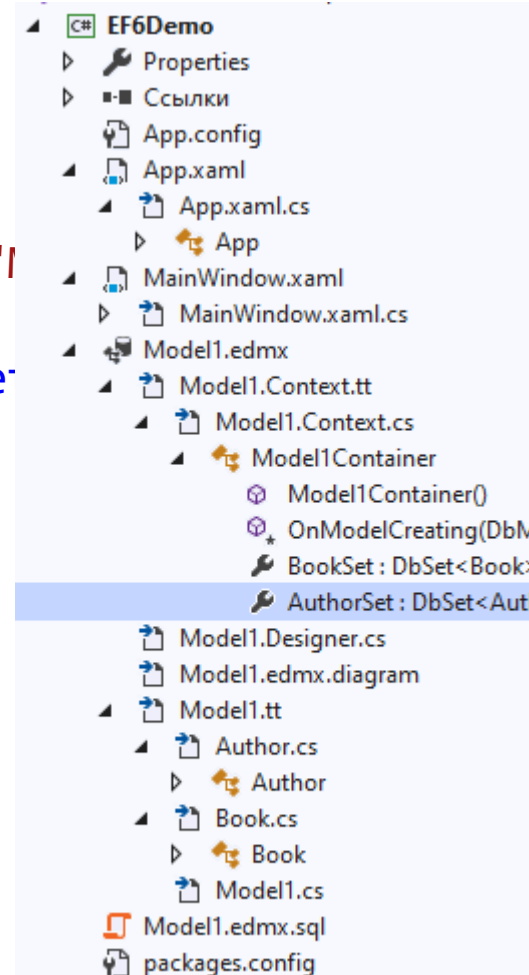
```
{
```

```
    public int BookId { get; set; }  
    public string Title { get; set; }  
    public string Description { get; set; }
```

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

```
}
```

Сгенерированные
классы




```
namespace EF6Demo
```

```
{
```

```
    using System;
```

```
    using System.Data.Entity;
```

```
    using System.Data.Entity.Infrastructure;
```

```
    public partial class Model1Container : DbContext
```

```
    {
```

```
        public Model1Container()
```

```
            : base("name=Model1Container")
```

```
        {
```

```
        }
```

```
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
```

```
        {
```

```
            throw new UnintentionalCodeFirstException();
```

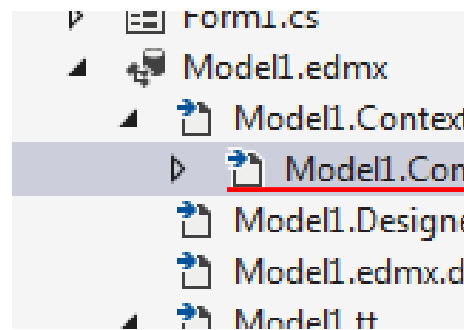
```
        }
```

```
        public virtual DbSet<Book> BookSet { get; set; }
```

```
        public virtual DbSet<Author> AuthorSet { get; set; }
```

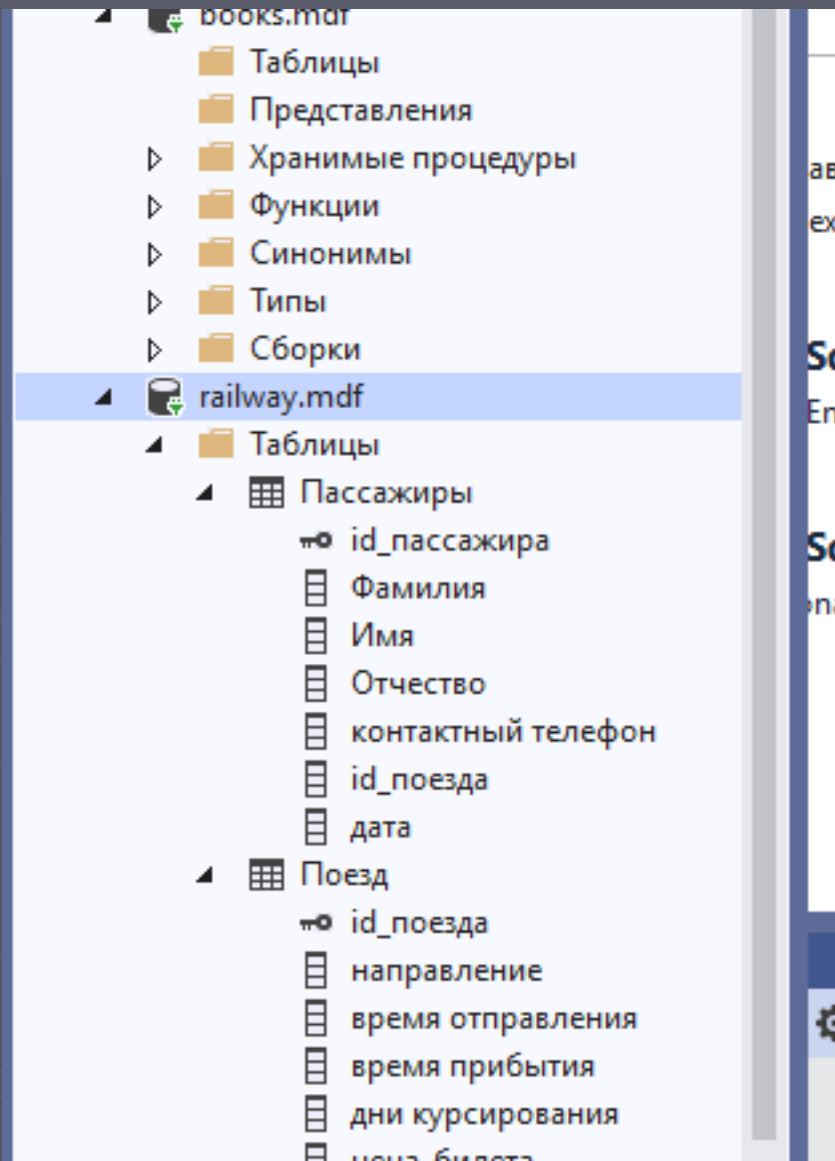
```
    }
```

```
}
```

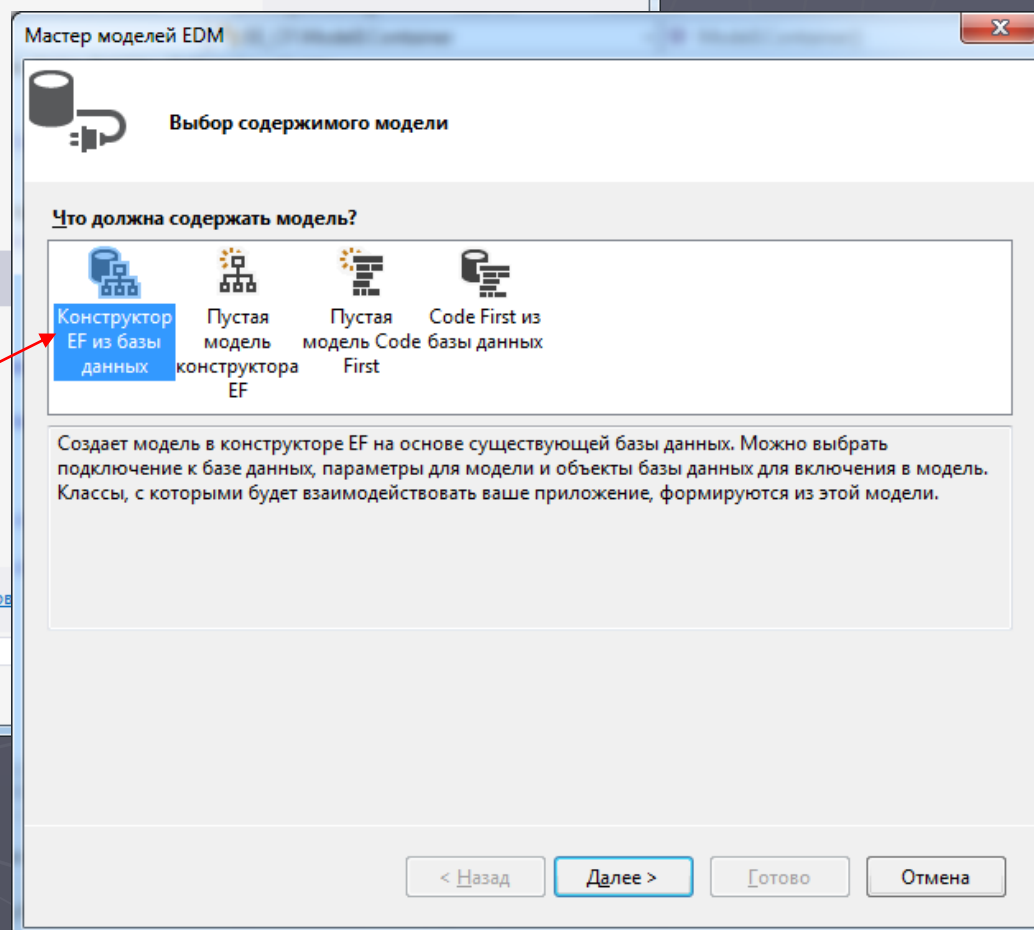
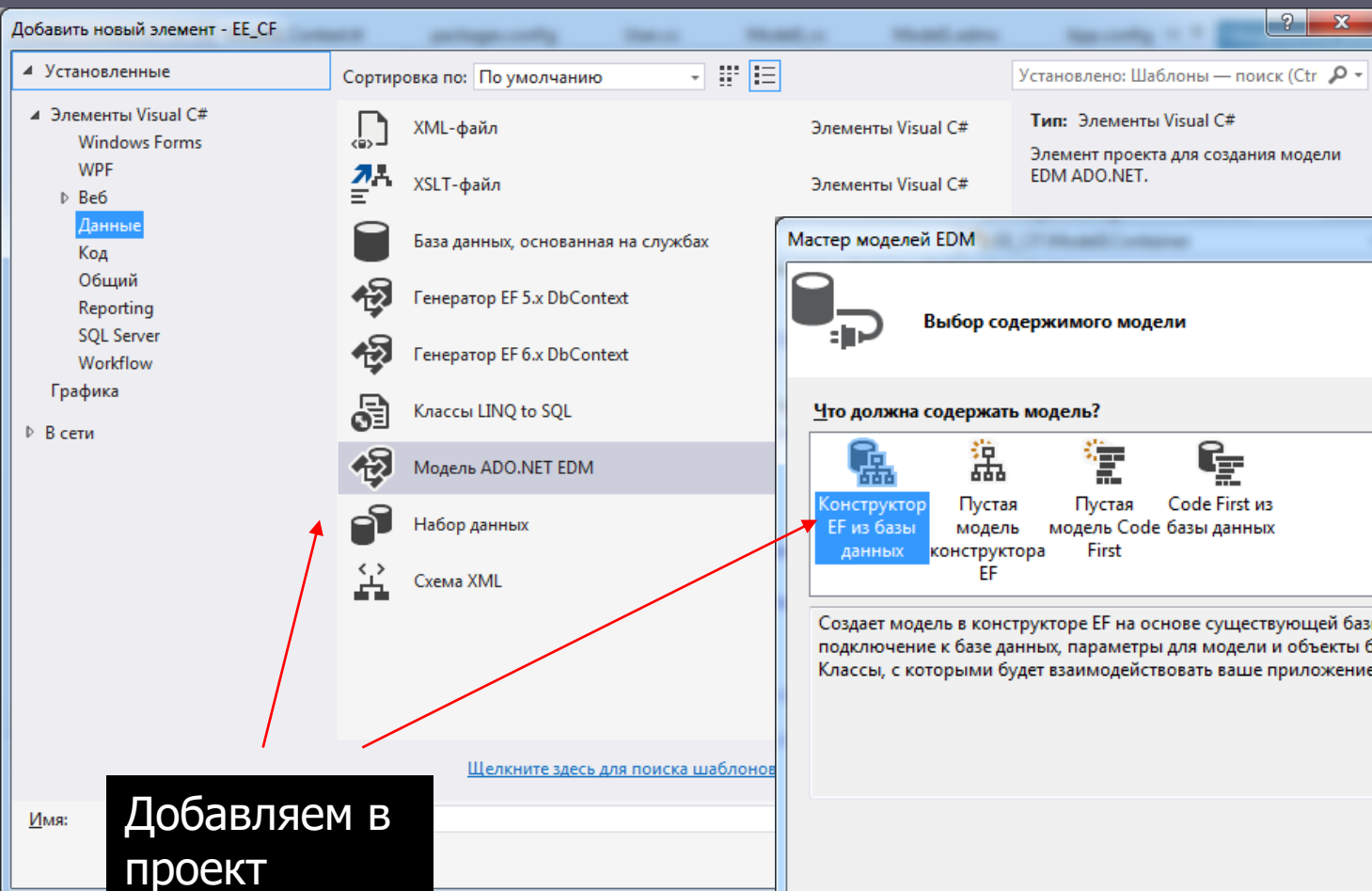


Сочетание паттернов
Unit Of Work и Repository


Database First



EF6



Мастер моделей EDM

 **Выбор подключения к данным**

Какое подключение к данным будет использоваться приложением для подключения к базе данных?

railway.mdf

Возможно, эта строка подключения содержит конфиденциальные данные, которые требуются для подключения к базе данных. Хранение строки подключения может представлять угрозу безопасности. Вы хотите включить эти данные в строку подключения?

☐ Нет, исключить конфиденциальные данные из строки подключения.

☐ Да, включить конфиденциальные данные в строку подключения.

Строка подключения:


metadata=res://*/Model2.csdl|res://*/Model2.ssdl|res://*/Model2.msl;provider=System.Data.SqlClient;provider=(LocalDB)\MSSQLLocalDB;attachdbfilename=C:\NATALIA\...\DataSource\railway.mdf;integrated security=True;connect

☒ Сохранить параметры соединения в App.Config как:

railwayEntities

< Назад

Мастер моделей EDM

 **Выберите параметры и объекты базы данных**

Какие объекты базы данных нужно включить в модель?

☒ Таблицы

- ☒ dbo
 - ☐ sysdiagrams
 - ☒ Пассажиры
 - ☒ Поезд
- ☐ Представления
- ☐ Хранимые процедуры и функции

☐ Формировать имена объектов во множественном или единственном числе

☒ Включить столбцы внешних ключей в модель

☒ Импортировать выбранные хранимые процедуры и функции в модель сущностей

Пространство имен модели:

railwayModel

< Назад Далее > Готово Отмена

Определяем
данные
и объекты

Меню: Подключения серверов, Подключения SharePoint, Подключения данных, railway.mdf, Таблицы, Пассажиры, Поезд, Представления, Хранимые проц, Функции, Синонимы, Типы, Сборки.

Меню: Свойства, Свойства навигации.

Меню: Свойства, Свойства навигации.

Меню: Браузер моделей, Введите здесь текст для поиска, Model2.edmx, Диаграммы, Diagram1, railwayModel, Типы сущностей, Пассажиры, Поезд, Сложные типы, Типы Enum, Ассоциации, Функции импорта, EntityContainer: railwayEntities, Хранилище railwayModel, Таблицы/представления, Хранимые процедуры / функции, Ограничения.

Меню: Обзор решений, Обзор решений, Model1, User.cs, Model1.edmx, Model2.edmx, Model2.Co, Model2, Model2.De, Model2.ed, Model2.tt, Model2, Пасса, Поезд, packages.con, Program.cs, railway.mdf.

Меню: Свойства, EE_CF Свойства проекта, ReSharper.

Сущности, которые
станут классами

Context.tt

Model2.tt

```
<#@ template language="C#" debug="false" hostspecific="true" #>
<#@ include file="EF6.Utility.CS.ttinclude" #><#@
output extension=".cs" #><#

const string inputFile = @"Model2.edmx";
var textTransform = DynamicTextTransformation.Create(this);
var code = new CodeGenerationTools(this);
var ef = new MetadataTools(this);
var typeMapper = new TypeMapper(code, ef, textTransform.Errors);
var fileManager = EntityFrameworkTemplateFileManager.Create(this);
var itemCollection = new EdmMetadataLoader(textTransform.Host, textTransform.Errors).CreateEdmItemCollection(inputFile);
var codeStringGenerator = new CodeStringGenerator(code, typeMapper, ef);
```

```
if (!typeMapper.VerifyCaseInsensitiveTypeUniqueness(typeMapper.GetAllGlobalItems(itemCollection), inputFile))
{
    return string.Empty;
}
```

```
WriteHeader(codeStringGenerator, fileManager);
```

```
foreach (var entity in typeMapper.GetItemsToGenerate<EntityType>(itemCollection))
```

```
{
    //-----
    file // <auto-generated>
    Beg: // Этот код создан по шаблону.
    //
    #> // Изменения, вносимые в этот файл вручную, могут привести к непра
    <#=code: // Изменения, вносимые в этот файл вручную, будут перезаписаны при
    <#=code: // </auto-generated>
    {
    <#
```

```
namespace EE_CF
{
    using System;
    using System.Collections.Generic;
```

```
ссылка: 1
public partial class Пассажиры
{
    ссылка: 0
    public string id_пассажира { get; set; }
    ссылка: 0
    public string фамилия { get; set; }
    ссылка: 0
    public string имя { get; set; }
    ссылка: 0
    public string отчество { get; set; }
    ссылка: 0
    public string контактный_телефон { get; set; }
    ссылка: 0
    public int id_поезда { get; set; }
    ссылка: 0
    public string дата { get; set; }
}
```

```
faultValues(entity);
igationProperties(entity);
```

Шаблон, на t4

Обозреватель решений

Обозреватель решений — поиск (Ctrl+ж)

- Ссылки
- App.config
- Customer.cs
- DB_layer.cs
- Form1.cs
- Model1.edmx
- Model1.edmx.sql
- Model2.edmx
 - Model2.Context.tt
 - Model2.Designer.cs
 - Model2.edmx.diagram
 - Model2.tt
 - Model2.cs
 - Пассажиры.cs
 - Поезд.cs

Обозреватель решений Team Explorer

Свойства



```
<auto-generated>
    Этот код создан по шаблону.

    Изменения, вносимые в этот файл вручную, могут привести к непредвиденной работе
    Изменения, вносимые в этот файл вручную, будут перезаписаны при повторном созда
</auto-generated>

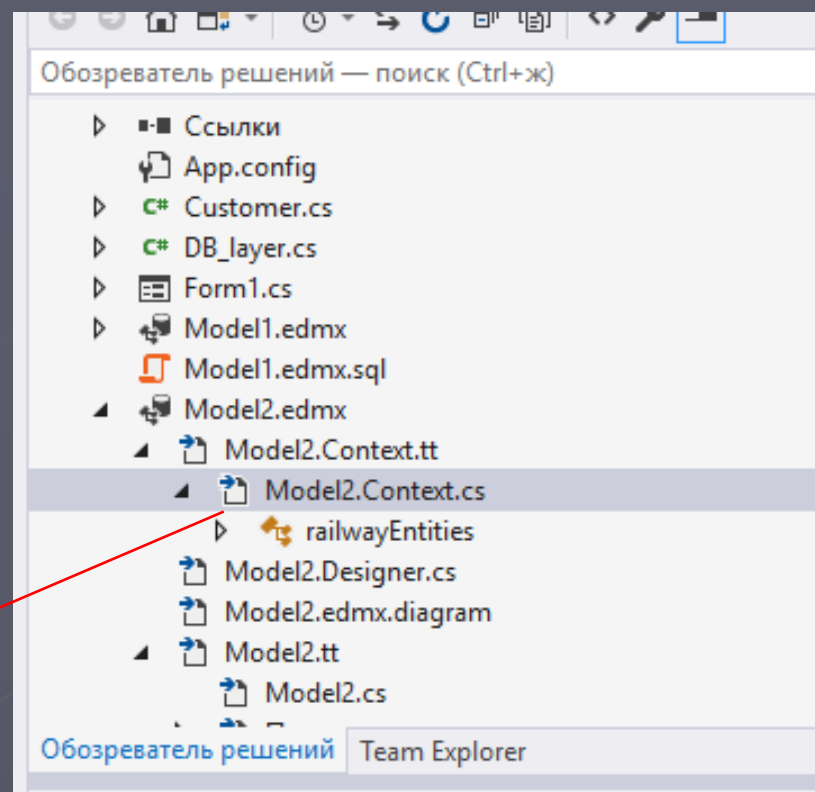
namespace EE_CF
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;

    ссылка: 1
    public partial class railwayEntities : DbContext
    {
        ссылка: 0
        public railwayEntities()
            : base("name=railwayEntities")
        {
        }

        ссылка: 1
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        ссылка: 0
        public virtual DbSet<Пассажиры> Пассажиры { get; set; }

        ссылка: 0
        public virtual DbSet<Поезд> Поезд { get; set; }
    }
}
```



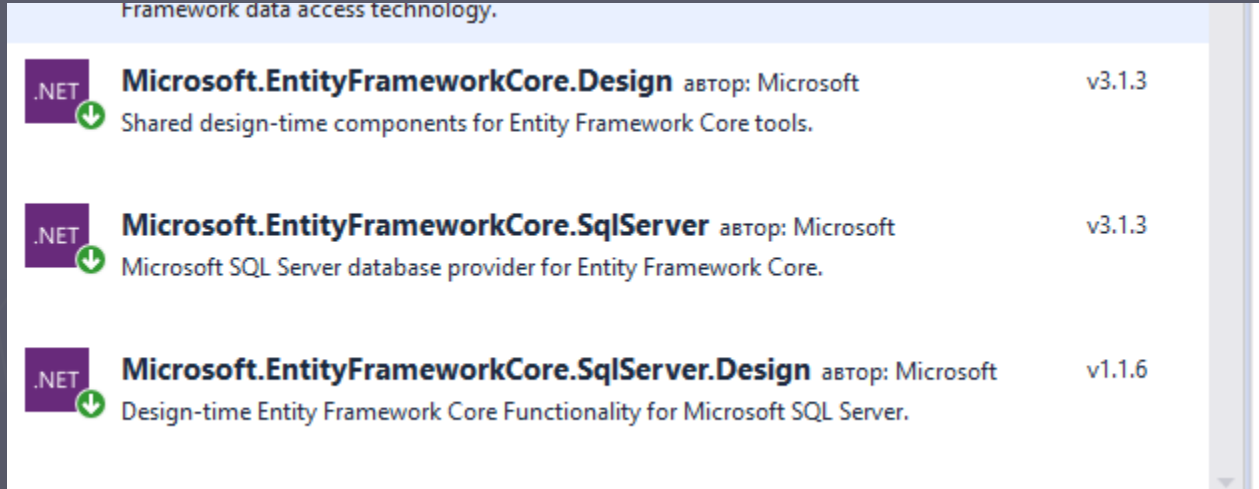
```
string a;
var context = new railwayEntities();
foreach (var p in context.Пассажиры)
    a = p.Имя;
```



idпоезда	направление	времяотправления	времяприбытия	дникурсирования	ценабилета	во
1	Минск-Могилёв	15.42	19.20	пн,ср,чт,пт,сб,вс	14000	15
2	Минск-Гродно	10.00	14.00	пн,вт,ср,чт	12000	75
3	Минск-Витебск	17.30	00.30	пн,ср,пт,сб	20000	10
4	Минск-Брест	9.35	14.10	пн,вт,ср	15000	12
5	Минск-Гомель	12.10	20.35	пн,ср,пт,сб,вс	26000	15
6	Минск-Мозырь	17.25	22.30	пн,ср,пт,сб	20000	15
7	Брест-Могилёв	17.20	22.10	пн,вт,ср,чт,пт	20000	15

```
var context = new railwayEntities();  
dataGrid1.ItemsSource = context.Поезд.ToList();
```

EF core



► Scaffold-DbContext Command

Tools -> NuGet Package Manager -> Package Manager Console

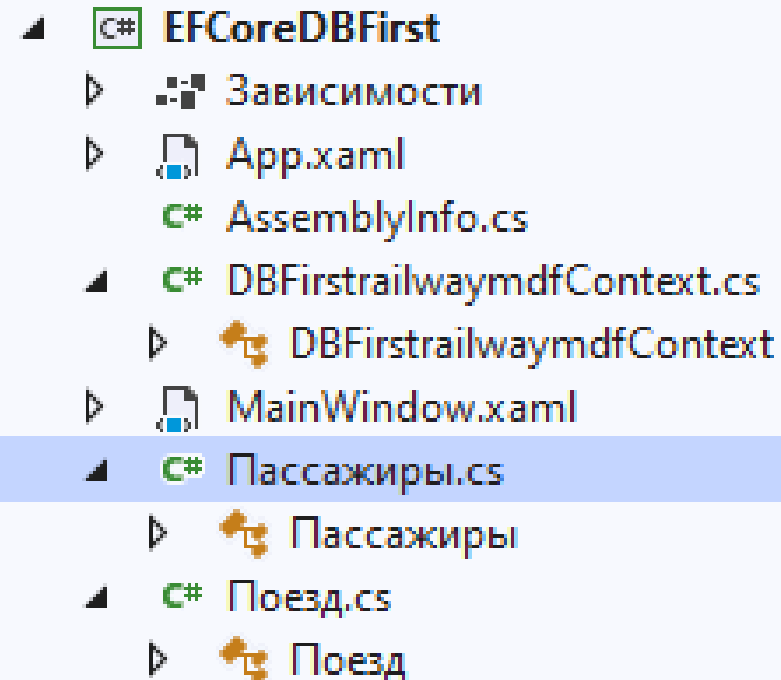
```
Scaffold-DbContext [-Connection] [-Provider] [-OutputDir] [-Context] [-Schemas>] [-Tables>] [-DataAnnotations] [-Force] [-Project] [-StartupProject] [<CommonParameters>]
```

```
PM> Scaffold-DbContext
```

```
"Server=(LocalDB)\MSSQLLocalDB;Database=railway;integrated security=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

```
namespace EFCoreDBFirst
{
    public partial class Пассажиры
    {
        public string IdПассажира { get; set; }
        public string Фамилия { get; set; }
        public string Имя { get; set; }
        public string Отчество { get; set; }
        public string КонтактныйТелефон { get; set; }
    }

    public int IdПоезда { get; set; }
    public string Дата { get; set; }
}
}
```




```
namespace EFCoreDBFirst
{
    public partial class Поезд
    {
        public int IdПоезда { get; set; }
        public string Направление { get; set; }
        public string ВремяОтправления { get; set; }

        public string ВремяПрибытия { get; set; }
        public string ДниКурсирования { get; set; }

        public int ЦенаБилета { get; set; }
        public int ВсегоМест { get; set; }
    }
}
```

```

public partial class DBFirstrailwaymdfContext : DbContext
{
    public DBFirstrailwaymdfContext()
    {
    }
    public DBFirstrailwaymdfContext(DbContextOptions<DBFirstrailwaymdfContext> options
        : base(options)
    {
    }
    public virtual DbSet<Пассажиры> Пассажиры { get; set; }
    public virtual DbSet<Поезд> Поезд { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        if (!optionsBuilder.IsConfigured)
        {
#warning To protect potentially sensitive information in your connection string, you should
source code. See http://go.microsoft.com/fwlink/?LinkId=723263 for guidance on storing con

optionsBuilder.UseSqlServer("Server=(localdb)\\mssqllocaldb;Database=railway.mdf;Trusted_C
");
        }
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Пассажиры>(entity =>
        {
            entity.HasKey(e => e.IdПассажира);

            entity.Property(e => e.IdПассажира)
                HasColumnName("id пассажира")

```

```
modelBuilder.Entity<Поезд>(entity =>
{
    entity.HasKey(e => e.IdПоезда);

    entity.Property(e => e.IdПоезда)
        .HasColumnName("id_поезда")
        .ValueGeneratedNever();

    entity.Property(e => e.ВремяОтправления)
        .IsRequired()
        .HasColumnName("время отправления")
        .HasMaxLength(5)
        .IsUnicode(false);

    entity.Property(e => e.ВремяПрибытия)
        .IsRequired()
        .HasColumnName("время прибытия")
        .HasMaxLength(50)
        .IsUnicode(false);

    entity.Property(e => e.ВсегоМест).HasColumnName("всего_мест");

    entity.Property(e => e.ДниКурсирования)
        .IsRequired()
        .HasColumnName("дни курсирования")
        .HasMaxLength(20)
        .IsUnicode(false);

    entity.Property(e => e.Направление)
        .IsRequired()
        .HasColumnName("направление")
        .HasMaxLength(30)
        .IsUnicode(false);

    entity.Property(e => e.ЦенаБилета).HasColumnName("цена_билета");
});

OnModelCreatingPartial(modelBuilder);
}
```

DbContext

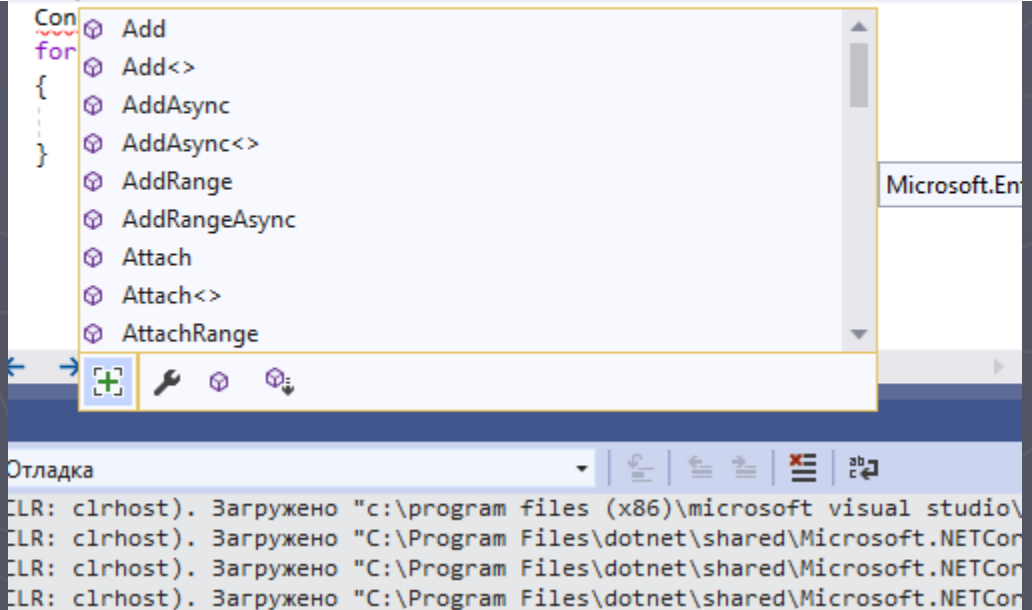
- ▶ DbContext представляет сеанс с базой данных, который можно использовать для запросов и сохранения экземпляров ваших сущностей в базе данных.
- ▶ DbContext представляет собой комбинацию шаблонов Unit of Work и Repository.

Задачи:

- ▶ Управление подключением к базе данных
- ▶ Настройка модели и отношения
- ▶ Запросы к базе данных
- ▶ Сохранение данных
- ▶ Настройка отслеживания изменений
- ▶ Кэширование
- ▶ Управление транзакциями

```
using (DBFirstrailwaymdfContext db = new
DBFirstrailwaymdfContext())
{
    // получаем объекты из бд и выводим на консоль
    var passanger = db.Пассажиры.ToList();
    Console.WriteLine("Список объектов:");
    foreach (Пассажиры p in passanger)
    {
        Console.WriteLine($"{p.Имя} -
{p.КонтактныйТелефон}");
    }
}
```

```
// получаем объекты из бд и выводим на консоль
var passanger = db.Пассажиры.ToList();
db.
```



Подход Code-First

объект POCO (Plain Old CLR Object)

```
public class Customer
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Company { get; set; }
}
```

автоматически находит такое поле с помощью механизма рефлексии - в его имени должна содержаться строка "Id"

посредник между бд и классами, описывающими данные

добавить класс контекста базы данных

```
class CustomerContext : DbContext
{
    public CustomerContext()
        : base("DbConnection")
    { }

    public DbSet<Customer> Customer { get; set; }
}
```

имя строки подключения к базе данных

System.Data.Entity

- ▶ **DbContext**: определяет контекст данных, используемый для взаимодействия с базой данных.
- ▶ **DbModelBuilder**: сопоставляет классы на языке C# с сущностями в базе данных.
- ▶ **DbSet/DbSet<TEntity>**: представляет набор сущностей, хранящихся в базе данных

Загрузить в память
локальную копию

Операции

```
context.Customer.Load();
```

```
dataGrid1.ItemsSource = context.Customer.Local.ToBindingList();
```

```
Customer fdel = context.Customer.Find(2);
```

Найти

```
context.Customer.Remove(fdel);
```

Удалить

```
context.SaveChanges();
```

Сохранить
изменения


```
public void InsertCustomer()
{
    // Создать объект для записи в БД
    Customer customer = new Customer
    {
        Id = 23,
        Name = "Nik",
        Company = "IBM",
    };

    // Создать объект контекста
    CustomerContext context = new CustomerContext();

    // Вставить объект в БД и сохранить изменения
    context.Customer.Add(customer);
    context.SaveChanges();
}
```

Настройка конфигураций при Code First

- ▶ Соглашения (conventions)
- ▶ Аннотации
- ▶ Fluent API



Соглашение конфигураций

► Соглашение для ключевого свойства

Свойство с именем **Id**



```
public partial class Item
{
    public Guid Id { get; set; }
}
```

Свойство с именем

[имя_класса]Id



```
public partial class Item
{
    public Guid ItemId { get; set; }
}
```

► Если нет ключевого свойства, то надо определить

```
public partial class Item
{
    [Key]
    public Guid GlobalItemKey { get; set; }
}
```

имеют тип `int` **ИЛИ** `GUID`

Сопоставление типов

C# Data Type	Mapping to SQL Server Data Type
int	int
string	nvarchar(Max)
decimal	decimal(18,2)
float	real
byte[]	varbinary(Max)
datetime	datetime
bool	bit
byte	tinyint
short	smallint
long	bigint
double	float
char	No mapping
sbyte	No mapping (throws exception)
object	No mapping

- ▶ Все первичные ключи - NOT NULL
- ▶ Столбцы, сопоставляемые со свойствами ссылочных типов - NULL
- ▶ все значимые типы - NOT NULL

- ▶ **PluralizationService** Entity Framework проводит сопоставление между именами классов моделей и именами таблиц.
- ▶ таблицы получают по умолчанию в качестве названия множественное число
- ▶ Названия столбцов получают названия свойств модели.
- ▶ Один —ко-многим

```
public class Student {  
    public int StudentId { get; set; }  
    public string StudentName { get; set; }  
}
```

```
public class Grade {  
    public int GradeId { get; set; }  
    public string GradeName { get; set; }  
    public string Section { get; set; }  
  
}
```

► ОДИН-КО-МНОГИМ

```
public class Student {  
    public int StudentId { get; set; }  
    public string StudentName { get; set; }  
}
```

```
public class Grade {  
    public int GradeId { get; set; }  
    public string GradeName { get; set; }  
    public string Section { get; set; }  
    public ICollection<Student> Students { get; set; }  
}
```

Навигационные
свойства

```
public class Student {  
    public int Id { get; set; }  
    public string Name { get; set; }  
    public Grade Grade { get; set; }  
}
```

```
public class Grade {  
    public int GradeID { get; set; }  
    public string GradeName { get; set; }  
  
    public ICollection<Student> Students { get; set; }  
}
```

Аннотации

настройка сопоставления моделей и таблиц с помощью атрибутов

```
[Key]  
public int Ident { get; set; }
```

Обязательность значения

```
[Required]  
public string Name { get; set; }
```

Задание допустимой длины

```
[MaxLength(20)]  
public string Name { get; set; }
```

```
[Table("М")]  
public class Actor
```

Задание допустимой длины

```
[NotMapped]  
public int Role { get; set; }
```

Поле не сохраняется в БД.

```
[ForeignKey("CompId")]  
public Company Company { get; set; }
```

```
set; }
```

```
public int Id { get; set; }  
[Column("MName")]  
public string Name { get; set; }  
}
```

Соглашение конфигураций Fluent API

набор методов, которые определяются сопоставлением между классами и их свойствами и таблицами и их столбцами

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

Конфигурация
контекста

Многословно

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Entity<User>().Property(p=>p.Name).HasMaxLength(30);
    modelBuilder.Entity<User>().HasKey(it => it.Login);
    // throw new UnintentionalCodeFirstException();
}
```


Способы получения связанных данных

► "жадная загрузка" или **eager loading**

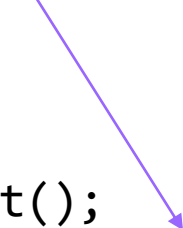
```
using(DB db = new DB())
{
    IEnumerable<User> users = db.User.Include(p => p.Actor);
    foreach( User p in users)
    {
        MessageBox.Show(p.Actor.Role);
    }
}
```

► "ленивая загрузка" или **lazy loading**

при первом обращении к объекту, если связанные данные не нужны, то они не подгружаются. Однако при первом же обращении к навигационному свойству эти данные автоматически подгружаются из бд.

► explicit loading("явная загрузка")

```
using (DB db = new DB())  
{  
  
    var t = db.Users.FirstOrDefault();  
    db.Entry(t).Collection("Actors").Load();  
  
}
```




Управление транзакциями

```
using (DBt db = new DB())
{
    using (var transaction = db.Database.BeginTransaction())
    {
        try
        {
            User p1 = db.Users.FirstOrDefault(p => p.Name == "Pol")
                // .....
                db.SaveChanges();
                transaction.Commit();
        }
        catch (Exception ex)
        {
            transaction.Rollback();
        }
    }
}
```

Repository

- ▶ паттерн, задача - управление доступом к источнику данных (содержит операции над данными или реализует CRUD-интерфейс)

может работать с разными сущностями



```
public interface IGenericRepository<TEntity> where TEntity : class
{
    void Create(TEntity item);
    TEntity FindById(int id);
    IEnumerable<TEntity> Get();
    IEnumerable<TEntity> Get(Func<TEntity, bool> predicate);
    void Remove(TEntity item);
    void Update(TEntity item);
}
```

позволяет абстрагироваться от конкретных подключений к источникам данных, с которыми работает программа, и является промежуточным звеном между классами, непосредственно взаимодействующими с данными, и остальной программой.


► базовая реализация для репозитория

```
public class EFGenericRepository<TEntity> : IGenericRepository<TEntity> where TEntity :  
class
```

```
{
```

```
    DbContext _context;  
    DbSet<TEntity> _dbSet;
```

ссылка на КОНТЕКСТ
набор DbSet



```
    public EFGenericRepository(DbContext context)  
    {  
        _context = context;  
        _dbSet = context.Set<TEntity>();  
    }
```

```
    public IEnumerable<TEntity> Get()  
    {  
        return _dbSet.AsNoTracking().ToList();  
    }
```

```
    public IEnumerable<TEntity> Get(Func<TEntity, bool> predicate)  
    {  
        return _dbSet.AsNoTracking().Where(predicate).ToList();  
    }
```

```
    public TEntity FindById(int id)  
    {  
        return _dbSet.Find(id);  
    }
```

```
    public void Create(TEntity item)  
    {
```

Преимущества

- ▶ гибкость при работе с разными типами подключений
- ▶ слой абстракции поверх слоя распределения данных
- ▶ сокращение дублирования кода запросов

```
EFGenericRepository<User> userRepo =  
    new EFGenericRepository<User>(new MyDBContext());
```

- ▶ Если репозитории используют одно и то же подключение, то для организации доступа к одному подключению для всех репозиториях приложения используется паттерн - **Unit Of Work**

содержит набор репозиториях и ряд некоторых общих для них функций

```
public class MyDBContext : DbContext  
{  
    public DbSet<User> Users { get; set; }  
    public DbSet<Company> Companies { get; set; }  
}
```

Repository + Unit Of Work

► 1) определяем модели

```
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
}

public class Company
{
    public int Id { get; set; }
    public string CName { get; set; }
    public Student Student { get; set; }
}
```


► 2) контекст + паттерн Репозиторий

```
public class StudentContext : DbContext
{
    public DbSet<Student> Students { get; set; }
    public DbSet<Company> Companies { get; set; }
}

interface IRepository<T> where T : class
{
    IEnumerable<T> GetAll();
    T Get(int id);
    void Create(T item);
    void Update(T item);
    void Delete(int id);
}
```

► 3) Реализация репозитория

```
public class StudentRepository : IRepository<Student>
{
    private StudentContext db;

    public StudentRepository(StudentContext context)
    {
        this.db = context;
    }

    public IEnumerable<Student> GetAll()
    {
        return db.Students;
    }

    public Student Get(int id)
    {
        return db.Students.Find(id);
    }

    public void Create(Student student)
    {
        db.Students.Add(student);
    }

    public void Update(Student student)
    {
        db.Entry(student).State = EntityState.Modified;
    }

    public void Delete(int id)
    {
        Student student = db.Students.Find(id);
        if (student != null)
            db.Students.Remove(student);
    }
}
```

► 4) предоставляет доступ к репозиториям через отдельные свойства и определяет общий контекст для обоих репозитриев

```
public class UnitOfWork : IDisposable
{
    private StudentContext db = new StudentContext();
    private StudentRepository studentRepository;
    private CompanyRepository companyRepository;

    public StudentRepository Students
    {
        get
        {
            if (studentRepository == null)
                studentRepository = new StudentRepository(db);
            return studentRepository;
        }
    }

    public CompanyRepository Company
    {
        get
        {
            if (companyRepository == null)
                companyRepository = new CompanyRepository(db);
            return companyRepository;
        }
    }

    public void Save()
    {
        db.SaveChanges();
    }

    public void Dispose()
```

Хэширование паролей

```
{  
  
    public class SaltedHash  
    {  
  
        public string Hash { get; private set; }  
  
        public string Salt { get; private set; }  
  
        public SaltedHash(string password)  
        {  
            var saltBytes = new byte[32];  
  
            new Random().NextBytes(saltBytes);  
  
            Salt = Convert.ToBase64String(saltBytes);  
  
            var passwordAndSaltBytes = Concat(password, saltBytes);  
  
            Hash = ComputeHash(passwordAndSaltBytes);  
  
        }  
    }  
}
```

```
static string ComputeHash(byte[] bytes)
```

```
{
```

```
    using (var sha256 = SHA256.Create())
```

```
    {        return
```

```
Convert.ToBase64String(sha256.ComputeHash(bytes));
```

```
    }
```

```
}
```

Salt	Password
U9hgysRNNIUP3dbaXwMmsiEgbrE4qqGdDYwQatzAu20=	98JEex/cQgeC5SiB/WmosK7sCR84
U9hgysRNNIUP3dbaXwMmsiEgbrE4qqGdDYwQatzAu20=	WYi36U5doBKg+Kbi9SvO4SR2Kk6

```
static byte[] Concat(string password, byte[] saltBytes)
```

```
{
```

```
    var passwordBytes = Encoding.UTF8.GetBytes(password);
```

```
    return passwordBytes.Concat(saltBytes).ToArray();
```

```
}
```

```
public static bool Verify(string salt, string hash, string password)
{
    var saltBytes = Convert.FromBase64String(salt);
    var passwordAndSaltBytes = Concat(password, saltBytes);
    var hashAttempt = ComputeHash(passwordAndSaltBytes);
    return hash == hashAttempt;
}
```

```
}
```

```
}
```

LINQ to Entities

	Enumerable	Queryable
Выполнение	В памяти	Удаленно
Реализация	Объекты итераторы	Дерево выражений
Интерфейс	IEnumerable<T>	IQueryable<T>
Провайдеры	System.Collections LINQ 2 Objects	System.Linq LINQ 2 SQL LINQ 2 Entities

максимальная скорость
Для всего набора

оптимизация запроса
тратится меньше памяти
меньше пропускной способности сети,
обработаться чуть медленнее

LINQ to Entities

создает интерфейс для взаимодействия

ADO.NET ← EntityClient

EntityConnection

EntityCommand

EntityDataReader

```
var user = from p in db.Customers
            where p.Id == 1
            select p;
```

Запросы в итоге транслируются в
одной выражение sql

```
using (CustomerContext db = new CustomerContext())
{
    var forDel = db.Customer.Where(p => p.Id == 2);
}
```

операторы LINQ и методы расширения LINQ


- ▶ **First()/FirstOrDefault()**
- ▶ **Select()**
- ▶ **OrderBy() ThenBy()**
- ▶ **Join()**
- ▶ **GroupBy()**
- ▶ **Union() и т.д.**

Работа с SQL

Выборка

прямые sql-запросы к базе данных

```
var comps = db.Database.SqlQuery<Customer>("SELECT * FROM Customers");
```



позволяет получать информацию о базе данных, подключении и осуществлять запросы к БД.

ExecuteSqlCommand()

```
int num = db.Database  
    .ExecuteSqlCommand  
    ("DELETE FROM Customers WHERE Id=3");
```

Асинхронные операции

- ▶ **SaveChangesAsync**
- ▶ **FindAsync**
- ▶ **FirstOrDefaultAsync**
- ▶ **И т.д.** все методы возвращают объект задачи **Task** или **Task<T>**