

# Entity Framework



- ▶ **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 - 6.0 - 2013

## ▶ Entity Framework Core 2.0 -2017

объектно-ориентированная, легковесная ,  
расширяемая и кроссплатформенная

Поддержка провайдеров: для MS SQL  
Server, для SQLite, для PostgreSQL, для  
MySQL

# 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)

# 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

# Архитектура

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)**



Model1.edmx.sql    Model1.edmx [Diagram1]    Program.cs\*

```
graph LR
    User "User" -- "1" --- "*" Actor "Actor"
```

**User Entity Properties:**

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

**Actor Entity Properties:**

- Свойства
  - Id
  - Role
- Свойства навигации
  - User

**Обозреватель решений — поиск (Ctrl+ж)**

Решение "EE\_CF" (проектов: 1)

- EE\_CF
  - Properties
  - Ссылки
  - App.config
  - Customer.cs
    - Customer
    - CustomerContext
    - Do
  - DB\_layer.cs
  - Form1.cs
    - Form1.Designer.cs
  - Model1.edmx
  - Model1.Context.tt
    - Model1.Context.cs

```

<?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="3.0" xmlns:edmx="http://schemas.microsoft.com/ado/2009/11/edmx">
  <!-- EF Runtime content -->
  <edmx:Runtime>
    <!-- SSDL content -->
    <edmx:StorageModels>
      <Schema Namespace="Model1.Store" Alias="Self" Provider="System.Data.SqlClient" ProviderManifestToken="2012" xmlns="http://schemas.microsoft.com/ado/2009/11/ssdl">
        <!-- CSDL content -->
        <edmx:ConceptualModels>
          <Schema xmlns="http://schemas.microsoft.com/ado/2009/11/edm" xmlns:cg="http://schemas.microsoft.com/ado/2009/11/edm/Conceptual">
            <EntityContainer Name="Model1.Container" annotation:LazyLoadingEnabled="true">...</EntityContainer>
            <EntityType Name="User">...</EntityType>
            <EntityType Name="Actor">...</EntityType>
            <Association Name="UserActor">...</Association>
          </Schema>
        </edmx:ConceptualModels>
        <!-- C-S mapping content -->
        <edmx:Mappings>
          <Mapping Space="C-S" xmlns="http://schemas.microsoft.com/ado/2009/11/edm/Mapping">...</Mapping>
        </edmx:Mappings>
      </edmx:Runtime>
      <!-- EF Designer content (DO NOT EDIT MANUALLY BELOW HERE) -->
      <edmx:Designer xmlns="http://schemas.microsoft.com/ado/2009/11/edmx">
        <edmx:Connection>...</edmx:Connection>
        <edmx:Options>...</edmx:Options>
        <!-- Diagram content (shape and connector positions) -->
        <edmx:Diagrams>
          <!-- Diagram content -->
        </edmx:Diagrams>
      </edmx:Designer>
    </edmx:StorageModels>
  </edmx:Runtime>
</edmx:Edmx>

```

Обозреватель решений —

- Properties
- References
- App.config
- Customer.cs
- Form1.cs
- Form1.Designer.cs
- Form1
- Model1.edmx
  - Model1.Container.cs
  - Model1.Designer.cs
  - Model1.edmx.cs
  - Model1.tt
  - Actor.cs
  - Model1.cs
  - User.cs

Свойства

Документ XML



Разное

Вывод

Кодировка

Схемы

Таблица стилей

Открыть как xml

кодогенерация

```
<edmx:Runtime>
  <!-- SSDL content -->
  <edmx:StorageModels>
    <Schema Namespace="Model1.Store" Alias="Self" Provider="System.Data.SqlClient" ProviderManifestToken="2008" />
    <EntityContainer Name="Model1StoreContainer">
      <EntitySet Name="UserSet" EntityType="Model1.Store.UserSet" store:Type="Tables" Schema="dbo" />
      <EntitySet Name="ActorSet" EntityType="Model1.Store.ActorSet" store:Type="Tables" Schema="dbo" />
      <AssociationSet Name="UserActor" Association="Model1.Store.UserActor">
        <End Role="User" EntitySet="UserSet" />
        <End Role="Actor" EntitySet="ActorSet" />
      </AssociationSet>
    </EntityContainer>
    <EntityType Name="UserSet">
      <Key>
        <PropertyRef Name="Id" />
      </Key>
      <Property Name="Id" Type="int" StoreGeneratedPattern="Identity" Nullable="false" />
      <Property Name="Name" Type="nvarchar(max)" Nullable="false" />
      <Property Name="Login" Type="nvarchar(max)" Nullable="false" />
      <Property Name="Password" Type="nvarchar(max)" Nullable="false" />
    </EntityType>
    <EntityType Name="ActorSet">
      <Key>
        <PropertyRef Name="Id" />
      </Key>
      <Property Name="Id" Type="int" StoreGeneratedPattern="Identity" Nullable="false" />
      <Property Name="Role" Type="nvarchar(max)" Nullable="false" />
      <Property Name="User_Id" Type="int" Nullable="false" />
    </EntityType>
    <Association Name="UserActor">
      <End Role="User" Type="Model1.Store.UserSet" Multiplicity="1" />
      <End Role="Actor" Type="Model1.Store.ActorSet" Multiplicity="*" />
    </Association>
  </edmx:StorageModels>
</edmx:Runtime>
```

Набор таблиц

Набор ассоциаций

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="UserSet" EntityType="Model1.User" />
      <EntitySet Name="ActorSet" EntityType="Model1.Actor" />
      <AssociationSet Name="UserActor" Association="Model1.UserActor">
        <End Role="User" EntitySet="UserSet" />
        <End Role="Actor" EntitySet="ActorSet" />
      </AssociationSet>
    </EntityContainer>
    <EntityType Name="User">
      <Key>
        <PropertyRef Name="Id" />
      </Key>
      <Property Name="Id" Type="Int32" Nullable="false" annotation:StoreGeneratedPattern="Identity" />
      <Property Name="Name" Type="String" Nullable="false" />
      <Property Name="Login" Type="String" Nullable="false" />
      <Property Name="Password" Type="String" Nullable="false" />
      <NavigationProperty Name="Actor" Relationship="Model1.UserActor" FromRole="User" ToRole="Actor" />
    </EntityType>
    <EntityType Name="Actor">
      <Key>
        <PropertyRef Name="Id" />
      </Key>
      <Property Name="Id" Type="Int32" Nullable="false" annotation:StoreGeneratedPattern="Identity" />
      <Property Name="Role" Type="String" Nullable="false" />
      <NavigationProperty Name="User" Relationship="Model1.UserActor" FromRole="Actor" ToRole="User" />
    </EntityType>
  </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

```
<!-- SSDL content -->
<edmx:StorageModels>...</edmx:StorageModels>
<!-- CSDL content -->
<edmx:ConceptualModels>...</edmx:ConceptualModels>
<!-- C-S mapping content -->
<edmx:Mappings>
  <Mapping Space="C-S" xmlns="http://schemas.microsoft.com/ado/2009/11/mapping/cs">
    <EntityContainerMapping StorageEntityContainer="Model1StoreContainer" CdmEntityContainer="Model1Container">
      <EntitySetMapping Name="UserSet">
        <EntityTypeMapping TypeName="IsTypeOf(Model1.User)">
          <MappingFragment StoreEntitySet="UserSet">
            <ScalarProperty Name="Id" ColumnName="Id" />
            <ScalarProperty Name="Name" ColumnName="Name" />
            <ScalarProperty Name="Login" ColumnName="Login" />
            <ScalarProperty Name="Password" ColumnName="Password" />
          </MappingFragment>
        </EntityTypeMapping>
      </EntitySetMapping>
      <EntitySetMapping Name="ActorSet">
        <EntityTypeMapping TypeName="IsTypeOf(Model1.Actor)">
          <MappingFragment StoreEntitySet="ActorSet">
            <ScalarProperty Name="Id" ColumnName="Id" />
            <ScalarProperty Name="Role" ColumnName="Role" />
          </MappingFragment>
        </EntityTypeMapping>
      </EntitySetMapping>
      <AssociationSetMapping Name="UserActor" TypeName="Model1.UserActor" StoreEntitySet="ActorSet">
        <EndProperty Name="User">
          <ScalarProperty Name="Id" ColumnName="User_Id" />
        </EndProperty>
        <EndProperty Name="Actor">

```

Имя свойства и  
имя колонки

context.tt Model1.edmx Model1.tt Model1.edmx.diagram Model1.edmx.sql

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

const string inputFile = @"Model1.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 loader = new EdmMetadataLoader(textTransform.Host, textTransform.Errors);
var itemCollection = loader.CreateEdmItemCollection(inputFile);
var modelNamespace = loader.GetModelNamespace(inputFile);
var codeStringGenerator = new CodeStringGenerator(code, typeMapper, ef);

var container = itemCollection.OfType<EntityContainer>().FirstOrDefault();
if (container == null)
{
    return string.Empty;
}
#>
//-----
// <auto-generated>
// <#=CodeGenerationTools.GetResourceString("Template_GeneratedCodeCommentLine1")#>
//
// <#=CodeGenerationTools.GetResourceString("Template_GeneratedCodeCommentLine2")#>
// <#=CodeGenerationTools.GetResourceString("Template_GeneratedCodeCommentLine3")#>
// </auto-generated>
//-----
<#
```

Генерирует  
контекст

```
var codeNamespace = code.VsNamespaceSuggestion();
if (!String.IsNullOrEmpty(codeNamespace))
{
    public partial class Actor
    {
        public int Id { get; set; }
        public string Role { get; set; }

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

Скрипт на t4 – выполнив  
который будут  
сгенерированы сущности

Обозреватель решений

Обозреватель решений — поиск (Ctrl+ж)

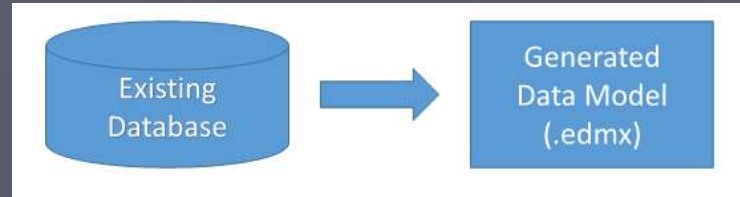
- Customer
- CustomerContext
- Do
- DB\_layer.cs
- Form1.cs
- Form1.Designer.cs
- Model1.edmx
- Model1.Context.tt**
- Model1.Context.cs
- Model1Container
- Model1.Designer.cs
- Model1.edmx.diagram
- Model1.tt
- Model1.edmx.sql
- Model2.edmx

Обозреватель решений Team Explorer

Свойства

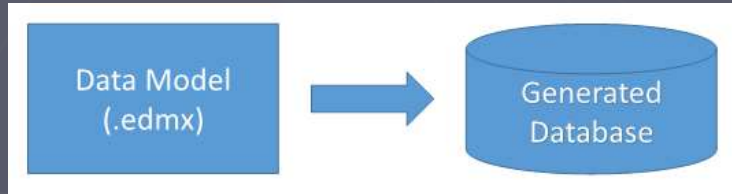
# Подходы к проектированию

## ► *Database-First*



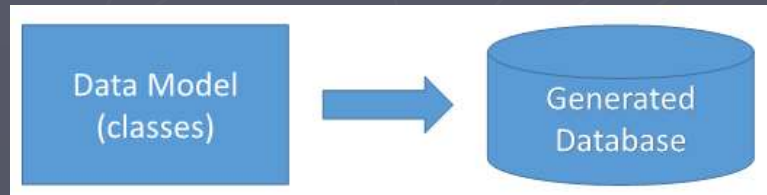
создание базы данных -> генерация EDMX-модель

## ► *Model-First*



создание графической модели EDMX -> генерация базы данных

## ► *Code-First*

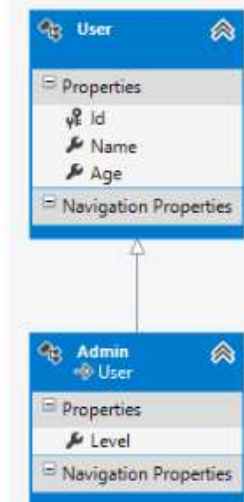
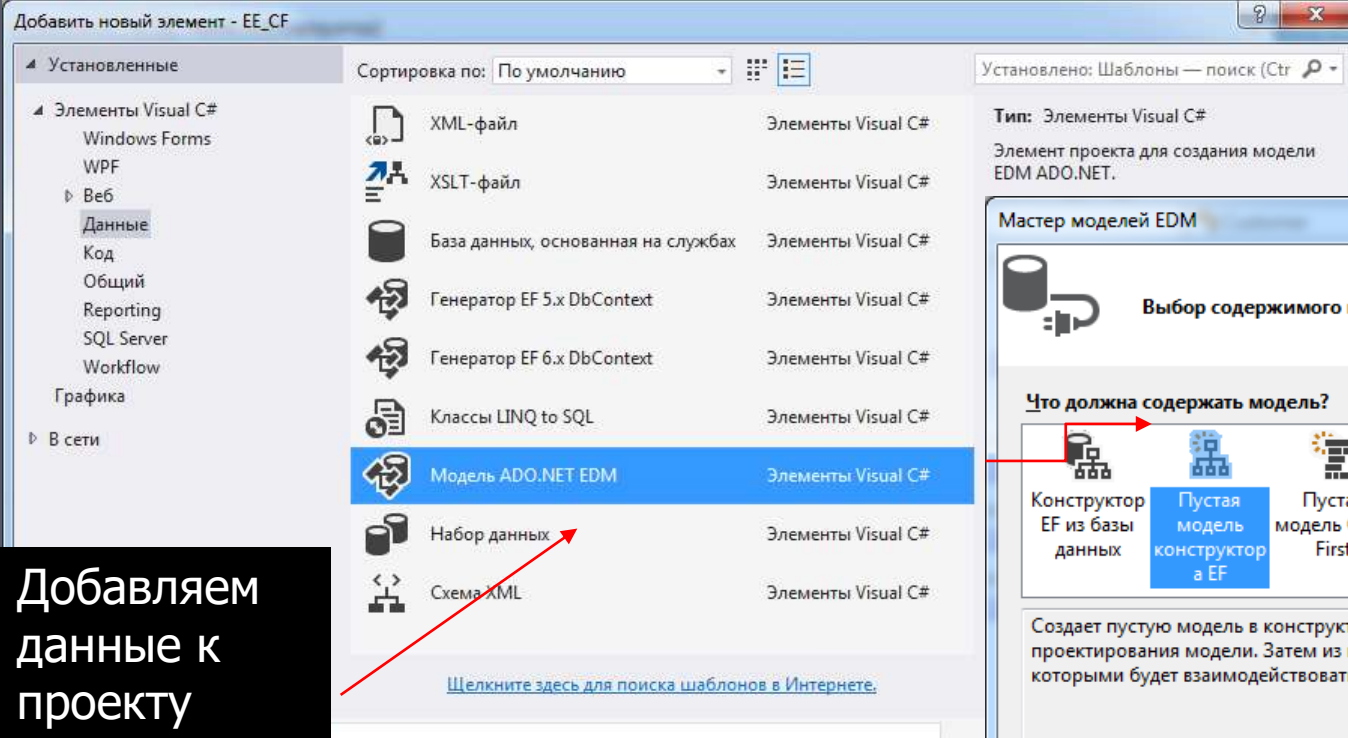


настройка классов C# объектной модели

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

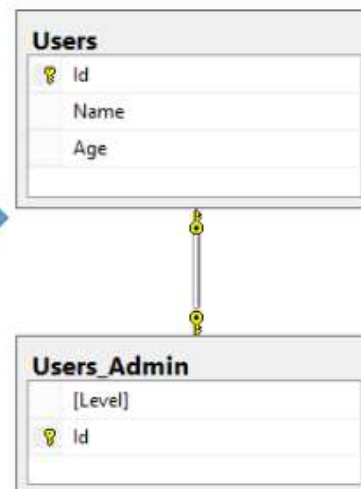


# подход Model-First

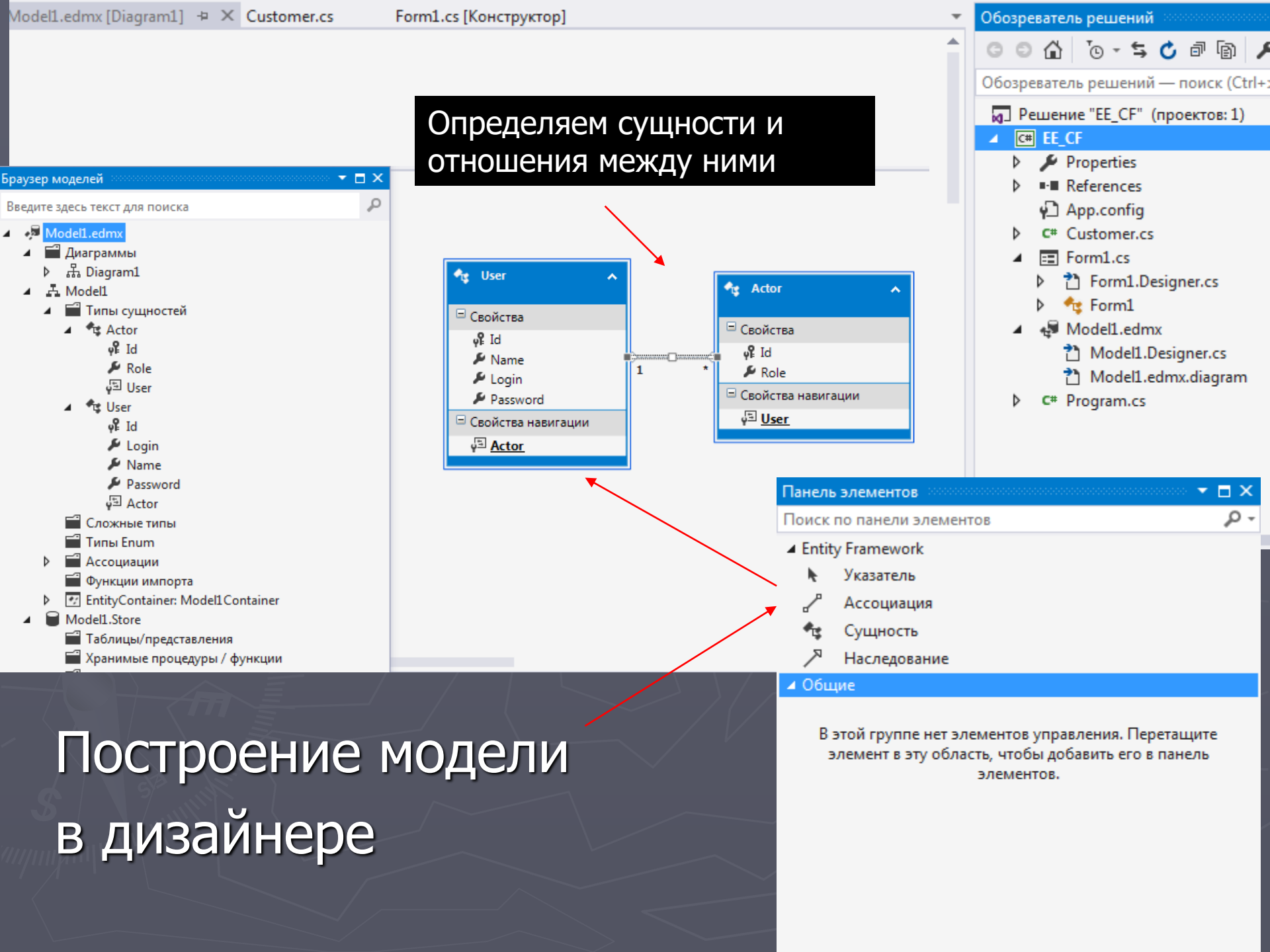


```
public partial class User
{
    public int Id { get; set; }
    public string Name { get; set; }
    public DateTime? Age { get; set; }
}

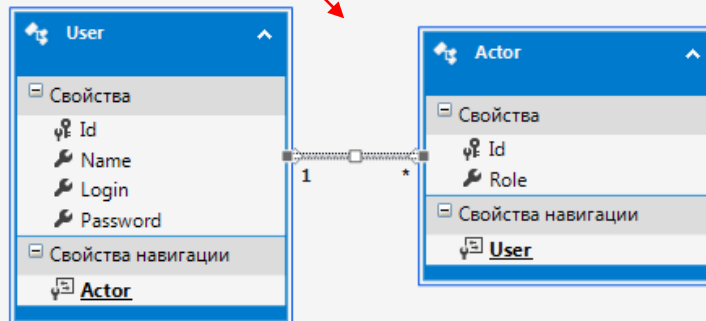
public partial class Admin : User
{
    public string Level { get; set; }
}
```







Определяем сущности и отношения между ними



Построение модели  
в дизайнере

**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)

9  
cs  
ner  
merContext

Designer.cs

dmx  
dmx.sql  
dmx

Team Explorer

Alt+ВВОД

Model1 ConceptualEntityModel



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

Схема

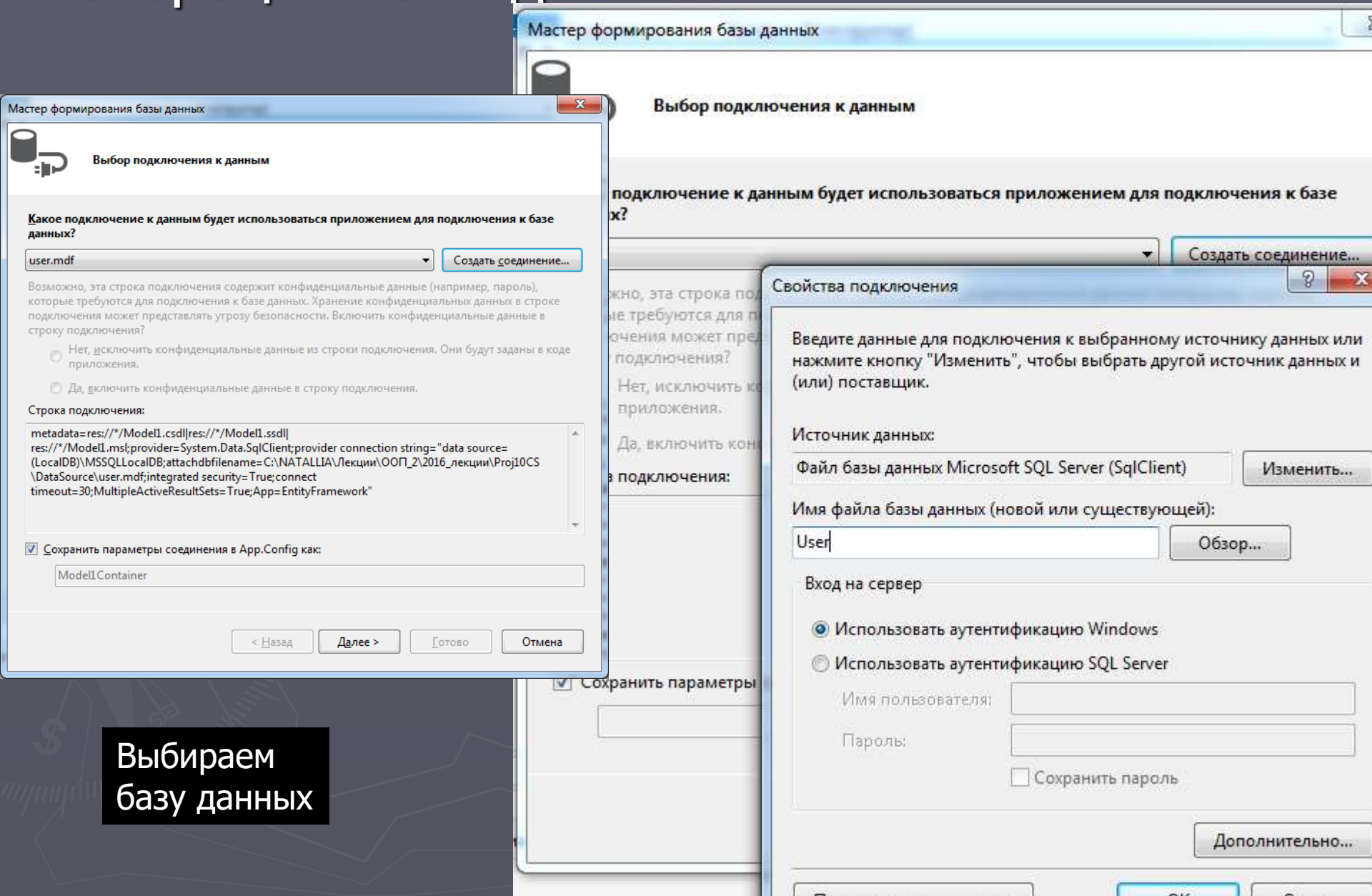
Доступ к контейнеру сущностей

Доступ к контейнеру сущностей

Модификатор доступа контейнера сущностей EDM.

граничения

# Генерация базы данных



Выбираем  
базу данных



Сводка и настройки

Сохранить DDL как: Model1.edmx.sql

DDL

-- Creating all tables

-- Creating table 'UserSet'

```
CREATE TABLE [dbo].[UserSet] (
  [Id] int IDENTITY(1,1) NOT NULL,
  [Name] nvarchar(max) NOT NULL,
  [Login] nvarchar(max) NOT NULL,
  [Password] nvarchar(max) NOT NULL
);
GO
```

-- Creating table 'ActorSet'

```
CREATE TABLE [dbo].[ActorSet] (
  [Id] int IDENTITY(1,1) NOT NULL,
  [Role] nvarchar(max) NOT NULL,
  [User_Id] int NOT NULL
);
GO
```

< Назад

Далее >

Готово

Отмена

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

```
<connectionStrings>
  <add name="Model1Container"
connectionString="metadata=res://*/Model1.csdl|res://*/Model1.s
sd1|res://*/Model1.msl;provider=System.Data.SqlClient;provider
connection string=&quot;data
source=(LocalDB)\MSSQLLocalDB;attachdbfilename=.\\user.mdf;integ
rated security=True;connect
timeout=30;MultipleActiveResultSets=True;App=EntityFramework&qu
ot;" providerName="System.Data.EntityClient" />
</connectionStrings>
```

Сгенерированные  
классы

КОД

```
public partial class Actor
{
    public int Id { get; set; }
    public string Role { get; set; }

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

```
public partial class User
{
    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
"CA2214:DoNotCallOverridableMethodsInConstructors")]
    public User()
    {
        this.Actor = new HashSet<Actor>();
    }

    public int Id { get; set; }
    public string Name { get; set; }
    public string Login { get; set; }
    public string Password { get; set; }

    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
"CA2227:CollectionPropertiesShouldBeReadOnly")]
    public virtual ICollection<Actor> Actor { get; set; }
}
```

```

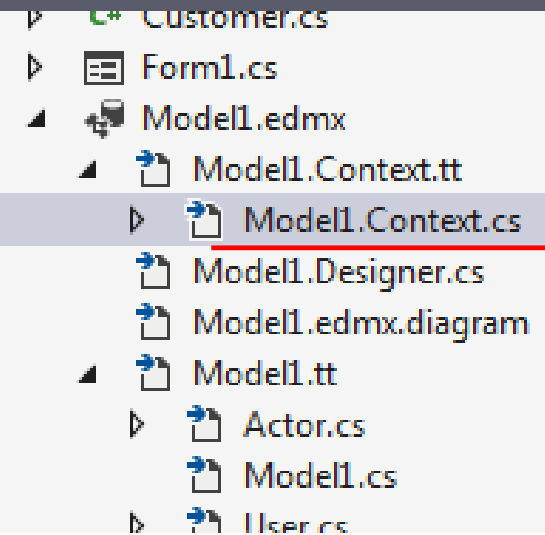
namespace EE_CF
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;

    public partial class Model1Container : DbContext
    {
        public Model1Container()
            : base("name=Model1Container")
        {
        }

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

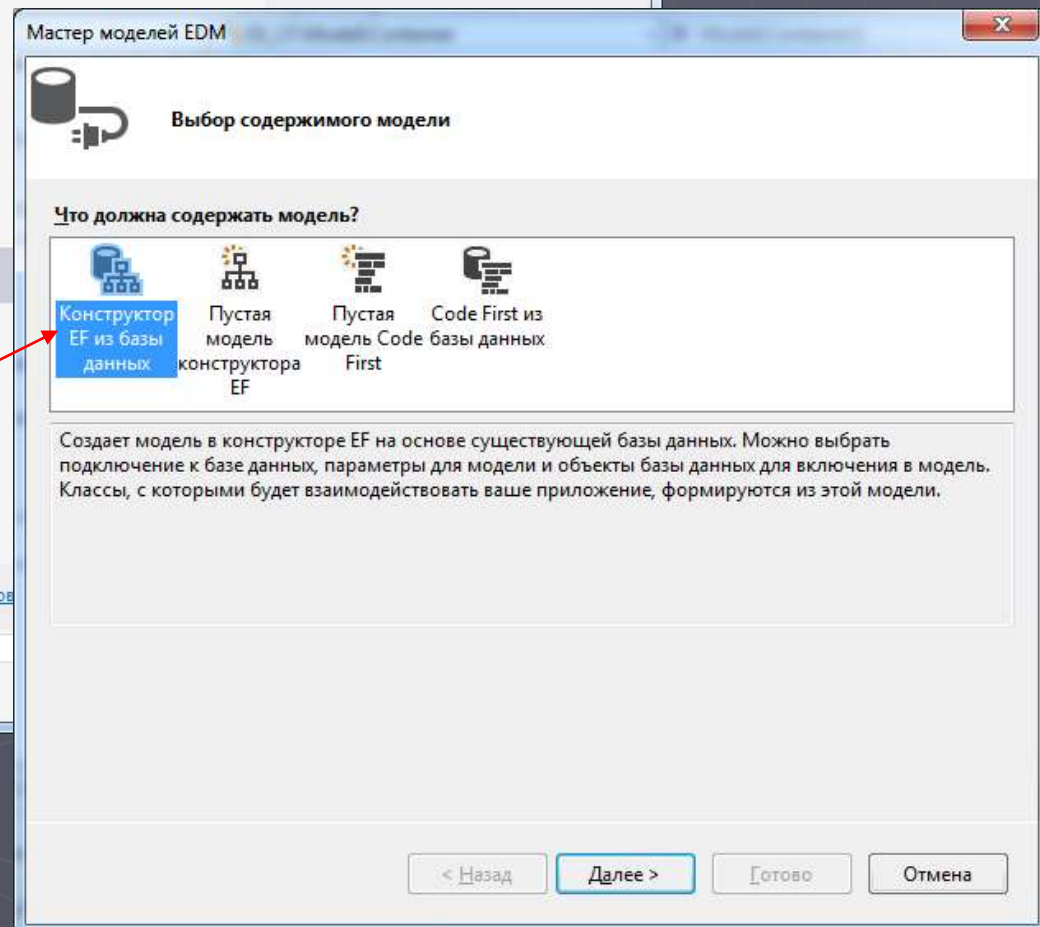
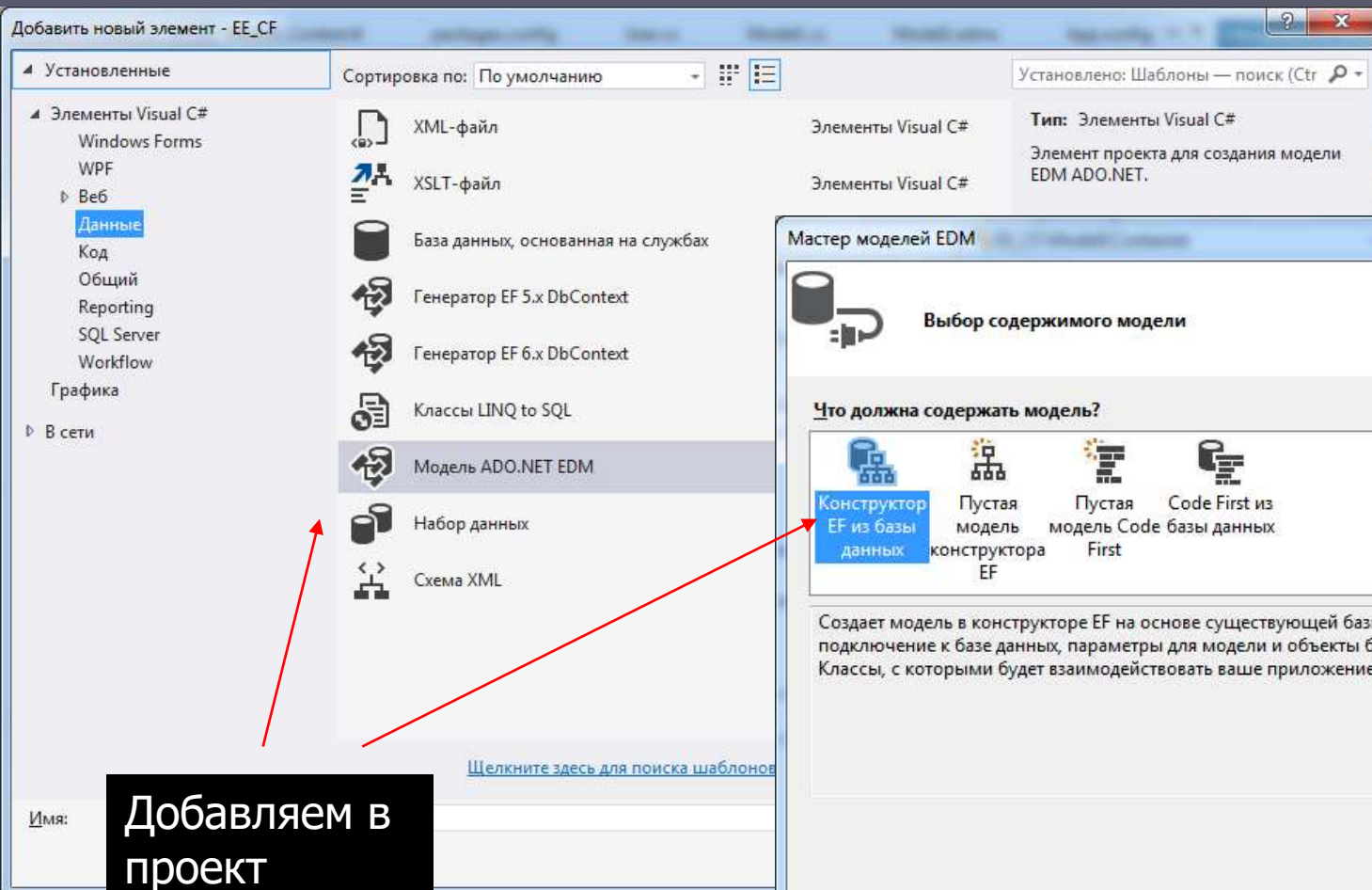
        public virtual DbSet<User> UserSet { get; set; }
        public virtual DbSet<Actor> ActorSet { get; set; }
    }
}

```




Сочетание паттернов  
Unit Of Work и Repository

# Database First





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. Хотите ли вы включить эту 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;providerAssembly=System.Data;attachdbfilename=C:\NATALIA\Railway\Railway.mdf;integrated security=True;connect timeout=30

☒ 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 непре
    <#=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>

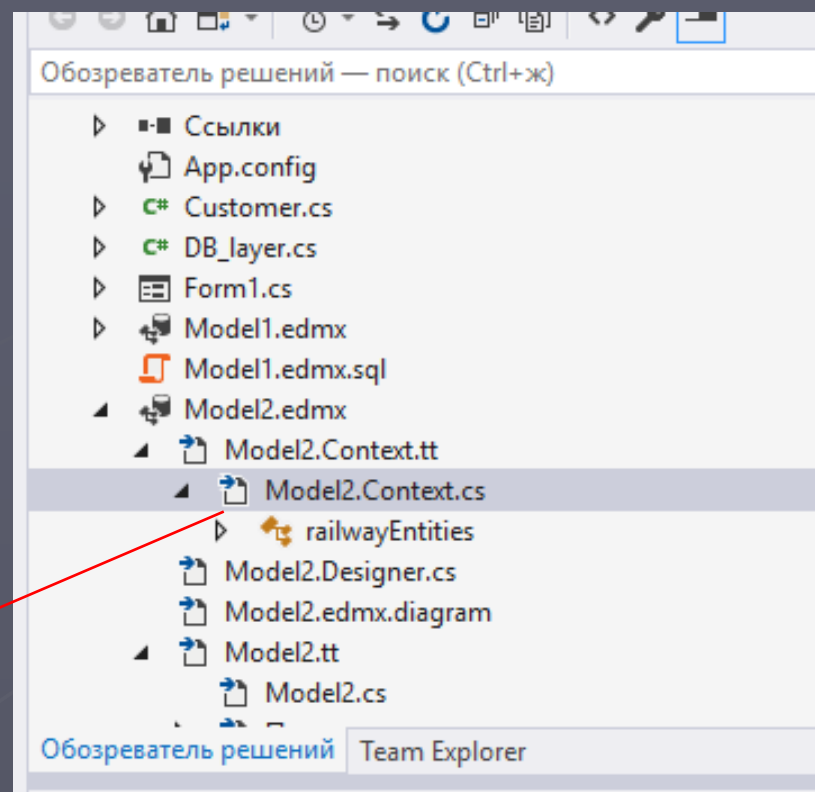
namespace EE_CF
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;

    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();
        }

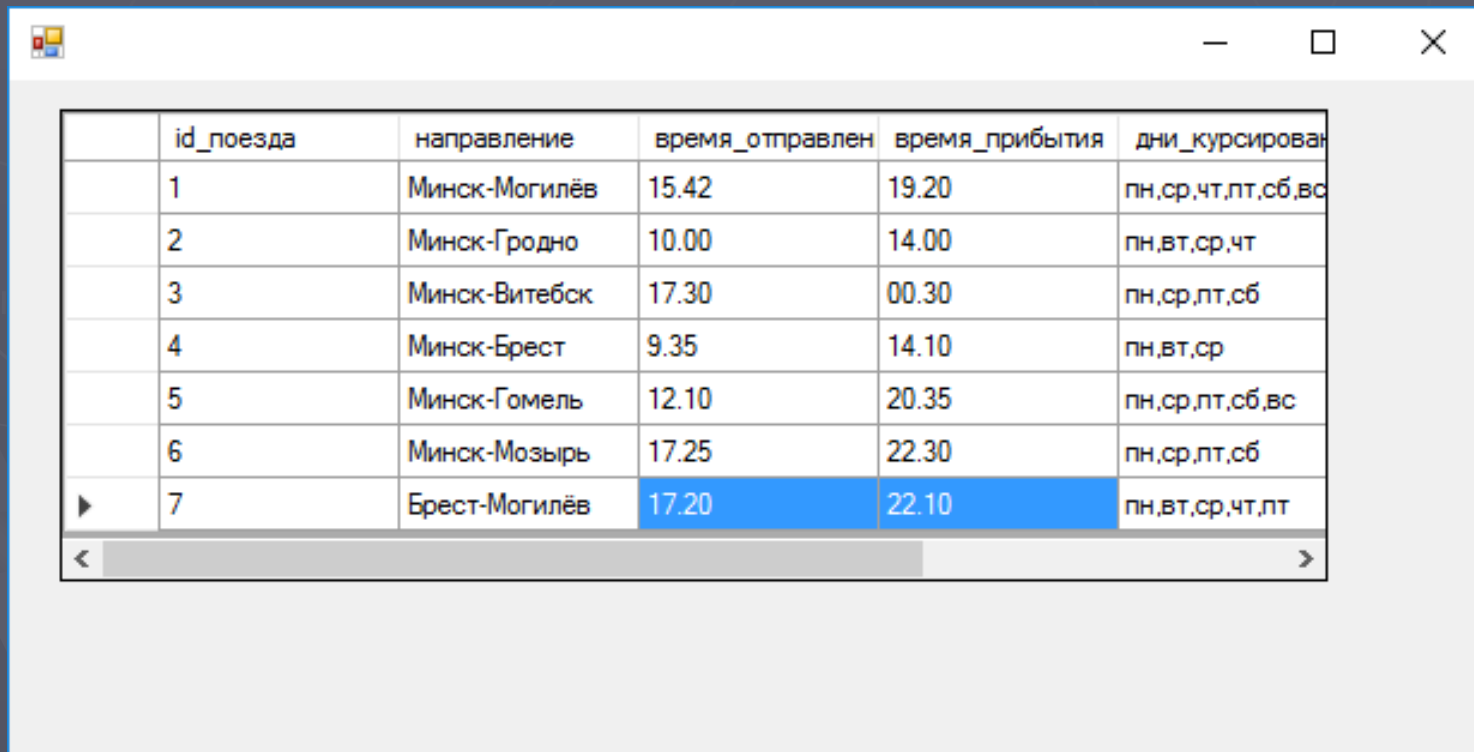
        ссылка: 0
        public virtual DbSet<Пассажиры> Пассажиры { get; set; }

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



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

```
var context = new railwayEntities();  
dataGridView1.DataSource = context.Поезд.ToList();
```



The screenshot shows a Windows application window with a standard title bar (minimize, maximize, close buttons). Inside the window is a DataGridView control displaying a table of train schedules. The table has 6 columns: an empty header cell, 'id\_поезда', 'направление', 'время\_отправлен', 'время\_прибытия', and 'дни\_курсирован'. There are 7 data rows. The 7th row is selected, highlighting the cells for 'время\_отправлен' (17.20) and 'время\_прибытия' (22.10). A horizontal scrollbar is visible at the bottom of the table.

	id_поезда	направление	время_отправлен	время_прибытия	дни_курсирован
	1	Минск-Могилёв	15.42	19.20	пн,ср,чт,пт,сб,вс
	2	Минск-Гродно	10.00	14.00	пн,вт,ср,чт
	3	Минск-Витебск	17.30	00.30	пн,ср,пт,сб
	4	Минск-Брест	9.35	14.10	пн,вт,ср
	5	Минск-Гомель	12.10	20.35	пн,ср,пт,сб,вс
	6	Минск-Мозырь	17.25	22.30	пн,ср,пт,сб
▶	7	Брест-Могилёв	17.20	22.10	пн,вт,ср,чт,пт

# 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

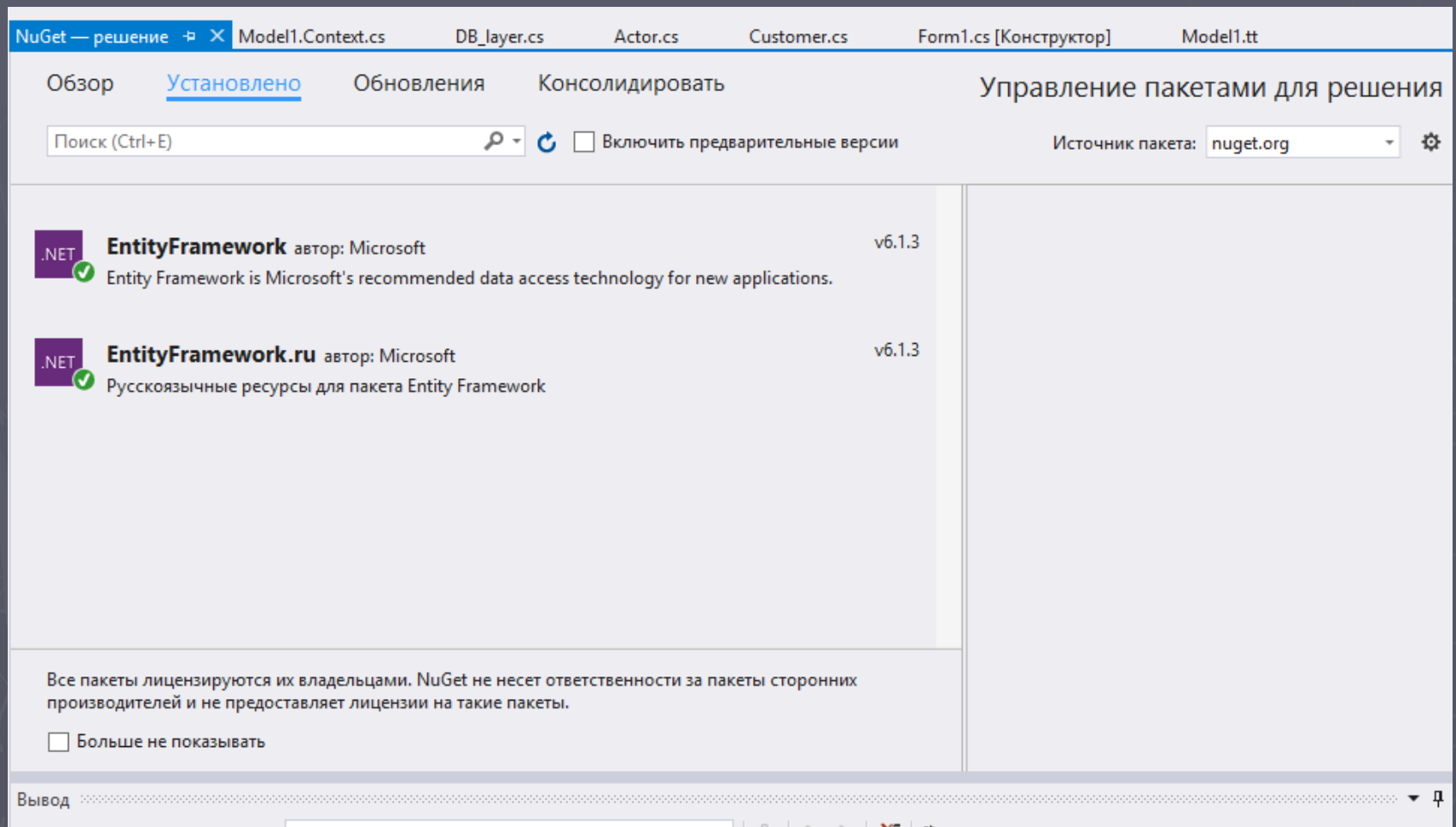
## добавить класс контекста базы данных

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

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

имя строки подключения к базе данных

# ► Manage NuGet Packages...: управление пакетами



# *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();
```

```
dataGridView1.DataSource = 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();
}
```

# 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

- ▶ int : int
- ▶ bit : bool
- ▶ char : string

И т.д.

- ▶ 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.

# Настройка конфигураций при Code First

- ▶ Аннотации
- ▶ Fluent API



# 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();
}
```

# Создание связи между таблицами

## ОДИН-КО-МНОГИМ (one-to-many)

```
public partial class Actor
{
    public int Id { get; set; }
    public string Role { get; set; }

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

```
public partial class User
{
    public User()
    {
        this.Actors = new HashSet<Actor>();
    }

    public int Id { get; set; }
    public string Name { get; set; }
    public string Login { get; set; }
    public string Password { get; set; }

    public virtual ICollection<Actor> Actors { get; set; }
}
```

Навигационные  
свойства



# Способы получения связанных данных

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

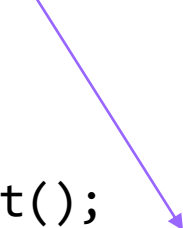
```
using (DB db = new DB())
{
    IEnumerable<User> users = db.User.Include(p => p.Actor);
    foreach (User p in users)
    {
        MessageBox.Show(p.Actor.Role);
    }
}
```

## ► "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; }  
}
```

# LINQ to Entities

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