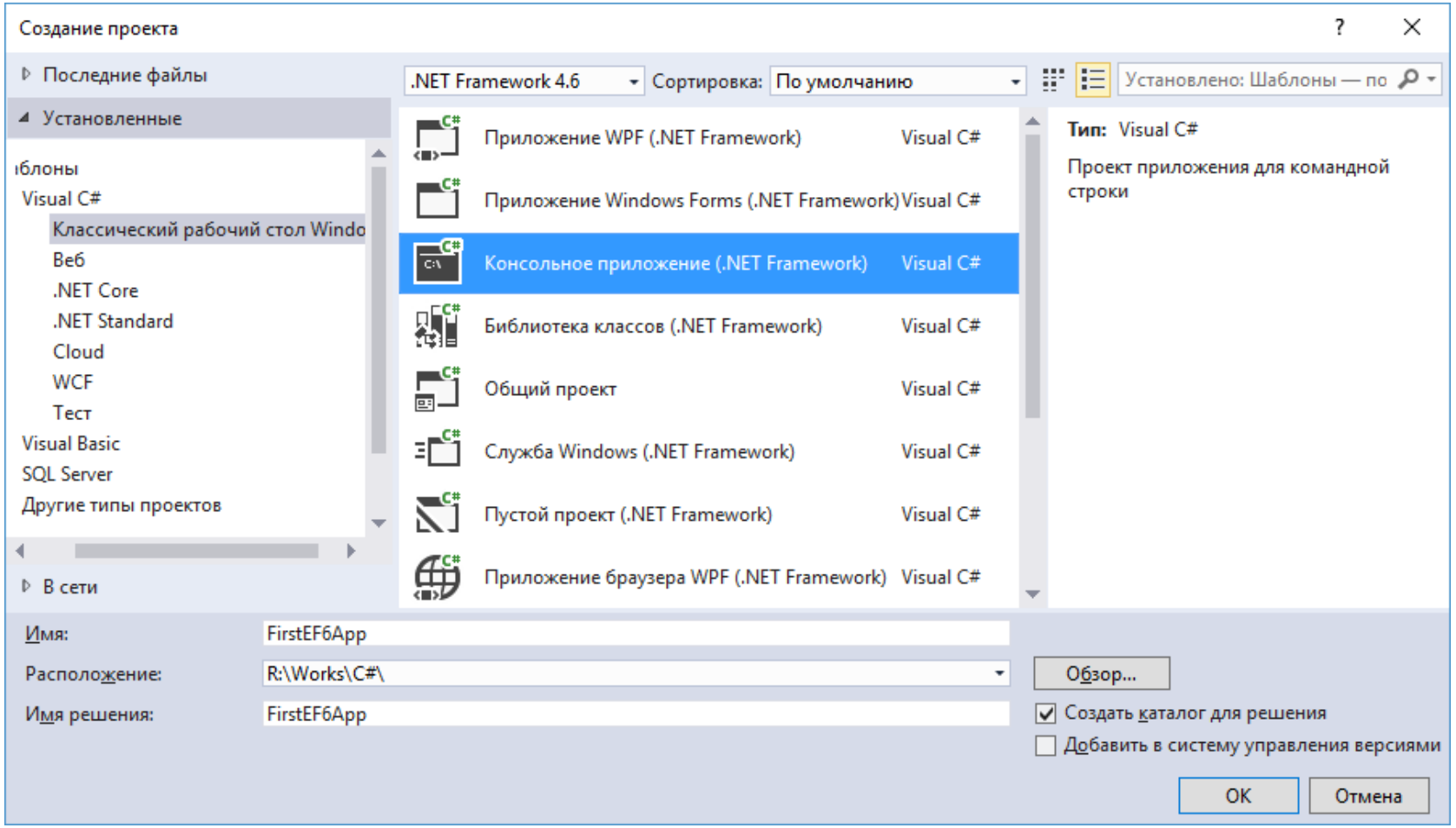
Подход Code First

Создадим новый проект по типу Console Application.



Подход Code First

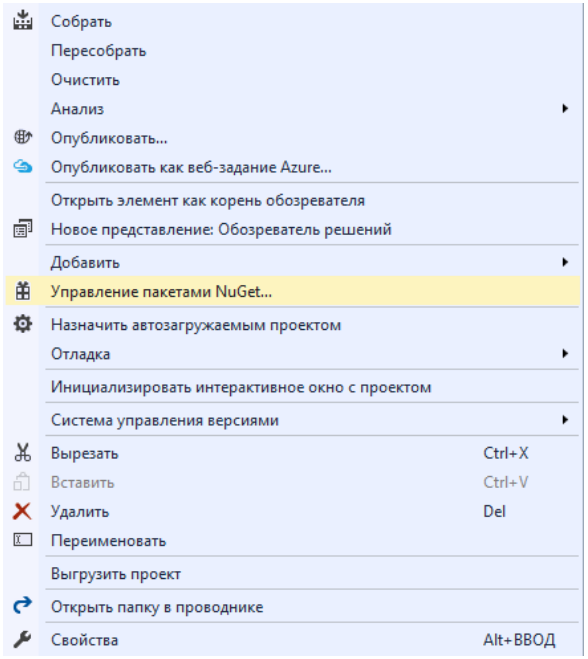
Пусть наше приложение будет посвящено работе с пользователями.
Добавим новый класс, который будет описывать данные.

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

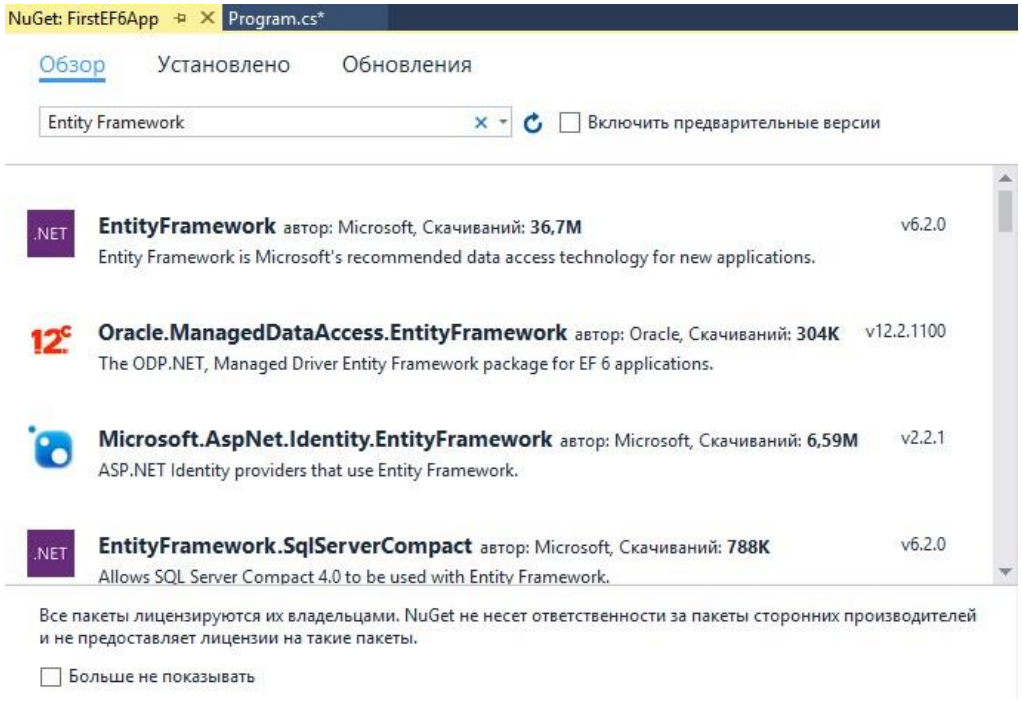
- Это обычный класс, который содержит некоторое количество автосвойств.
- Каждое свойство будет сопоставляться с отдельным столбцом в таблице из БД.
- Entity Framework при работе с Code First требует определения ключа элемента для создания первичного ключа в таблице в БД.
- По умолчанию при генерации БД EF в качестве первичных ключей будет рассматривать свойства с именами Id или [Имя_класса]Id (то есть UserId).
- Если мы хотим назвать ключевое свойство иначе, то нужно будет внести дополнительную логику на C#.

Подход Code First

По умолчанию библиотека для EF еще не добавлена.
Чтобы ее добавить, нажмем на проект правой кнопкой мыши и выберем в контекстном меню Manage NuGet Packages...:



В появившемся окне управления NuGet-пакетами выберем пакет Entity Framework и установим его:



Подход Code First

После установки пакета добавим в проект новый класс UserContext:

```
using System.Data.Entity;

namespace FirstEF6App
{
        class UserContext : DbContext
        {
                public UserContext() : base("DbConnection")
                { }

                public DbSet<User> Users { get; set; }
        }
}
```

Основу функциональности Entity Framework составляют классы, находящиеся в пространстве имен *System.Data.Entity*.

Среди них следует выделить следующие:

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

Подход Code First

- В любом приложении, работающим с БД через Entity Framework, нам нужен будет контекст (класс производный от DbContext) и набор данных DbSet, через который мы сможем взаимодействовать с таблицами из БД.
- В примере таким контекстом является класс UserContext.
- В конструкторе этого класса вызывается конструктор базового класса, в который передается строка "DbConnection" - это имя будущей строки подключения к базе данных.
- Если не использовать конструктор, то строка подключения будет носить имя самого класса контекста данных.
- В классе также определено свойство Users, которое будет хранить набор объектов User.
- В классе контекста данных набор объектов представляет класс DbSet<T>. Через это свойство будет осуществляться связь с таблицей объектов User в БД.

Подход Code First

Для установки подключения обычно используется файл конфигурации приложения. В проектах для десктопных приложений файл конфигурации называется *App.config*, в проектах веб-приложений - *web.config*.

После добавления Entity Framework ***App.config*** выглядит примерно следующим образом:

App.config* | User.cs | UserContext.cs | Program.cs

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <configuration>
3    <configSections>
4      <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=2
5      <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, Entity
6    </configSections>
7    <startup>
8      <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
9    </startup>
10   <entityFramework>
11     <defaultConnectionFactory type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory, EntityFramework
12       <parameters>
13         <parameter value="mssqllocaldb" />
14       </parameters>
15     </defaultConnectionFactory>
16     <providers>
17       <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, E
18     </providers>
19   </entityFramework>
20 </configuration>
```

Подход Code First

Установим подключение к базе данных.
После закрывающего тега **</configSections>** добавим следующий элемент:

```
<connectionStrings>
      <add name="DBConnection" connectionString="data
            source=(localdb)\MSSQLLocalDB;Initial
            Catalog=userstore.mdf;Integrated Security=True;"
       providerName="System.Data.SqlClient"/>
 </connectionStrings>
```

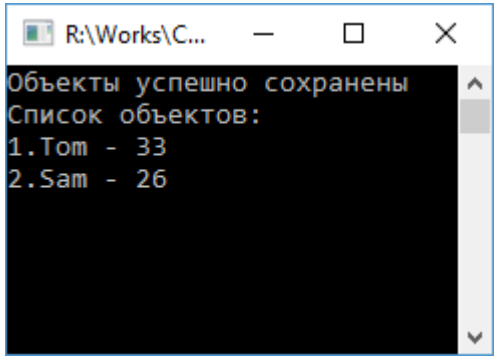
Все подключения к источникам данных устанавливаются в секции **connectionStrings**, а каждое отдельное подключение представляет элемент add.

В конструкторе класса контекста UserContext мы передаем в качестве названия подключения строку "DbConnection", поэтому данное название указывается в атрибуте **name="DBConnection"**.

Подход Code First

Перейдем к файлу *Program.cs* и изменим его содержание следующим образом:

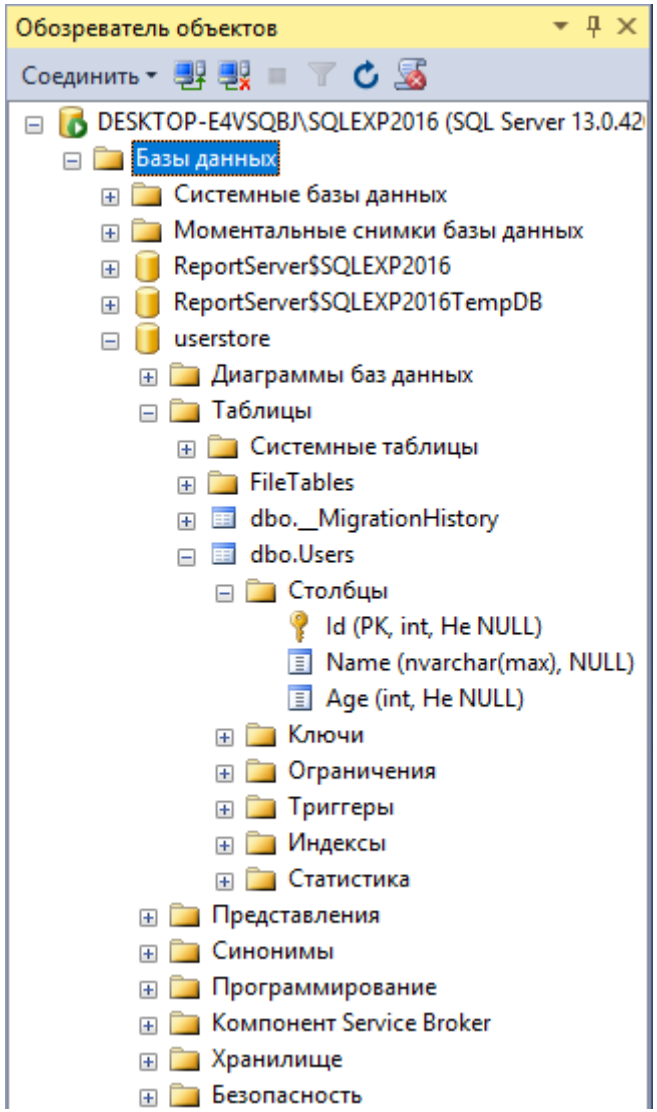
```
static void Main(string[] args)
{
    using (UserContext db = new UserContext())
    {
        // создаем два объекта User
        User user1 = new User { Name = "Tom", Age = 33 };
        User user2 = new User { Name = "Sam", Age = 26 };

        // добавляем их в БД
        db.Users.Add(user1);
        db.Users.Add(user2);
        db.SaveChanges();
        Console.WriteLine("Объекты успешно сохранены");

        // получаем объекты из БД и выводим на консоль
        var users = db.Users;
        Console.WriteLine("Список объектов:");
        foreach (User u in users)
        {
            Console.WriteLine($"{u.Id}.{u.Name} - {u.Age}");
        }
    }
    Console.Read();
}
```



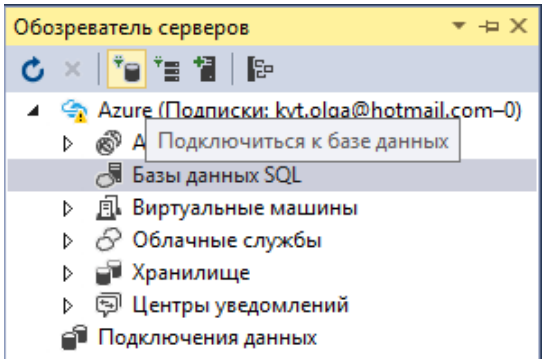
Так как класс UserContext через родительский класс DbContext реализует интерфейс IDisposable, то для работы с UserContext с автоматическим закрытием данного объекта мы можем использовать конструкцию using.

Подход Code First

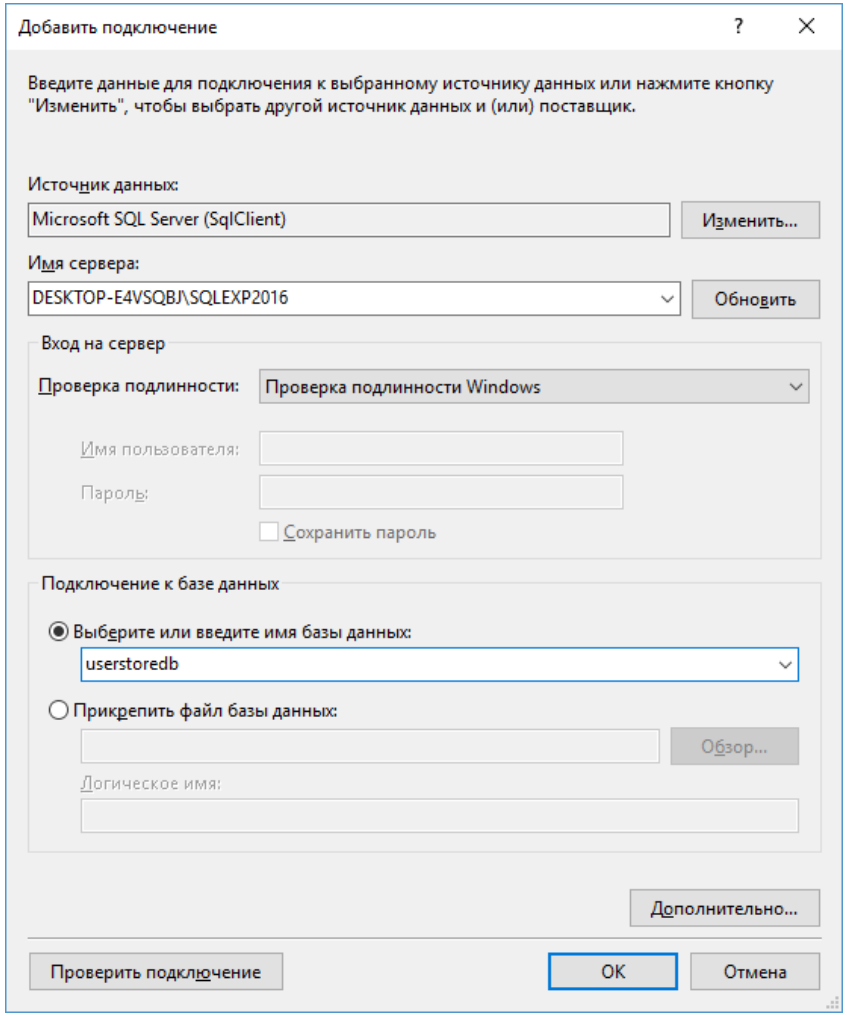
- EF обеспечивает простое и удобное управление объектами из БД.
- В данном примере не надо даже создавать БД и определять в ней таблицы.
- EF все сделает на основе определения класса контекста данных и классов моделей.
- Если база данных уже имеется, то EF не будет повторно создавать ее.
- Наша задача - только определить модель, которая будет храниться в базе данных, и класс контекста.
- Поэтому данный подход называется **Code First** - сначала пишется код, а потом по нему создается база данных и ее таблицы.



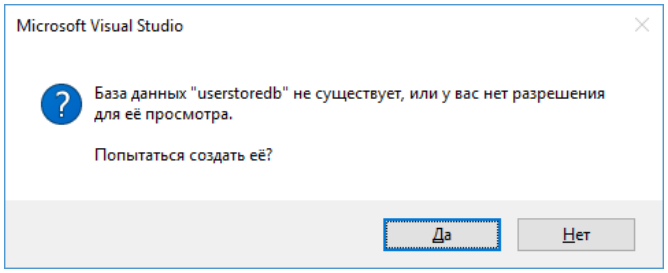
1. Создадим тестовую базу данных. В Visual Studio выберем в меню пункт View->Server Explorer. В открывшемся окне Server Explorer подключимся к новой базе данных, выбрав Connect to Database.



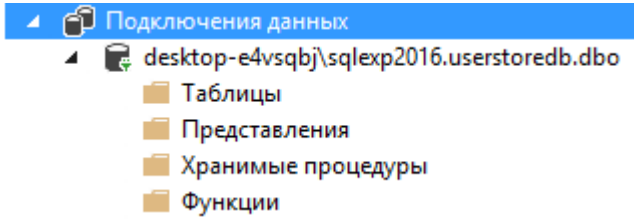
2. Выберем сервер, в качестве имени базы данных введем userstoredb



3. Если база данных не существует, нам отобразится окно с подтверждением ее создания.

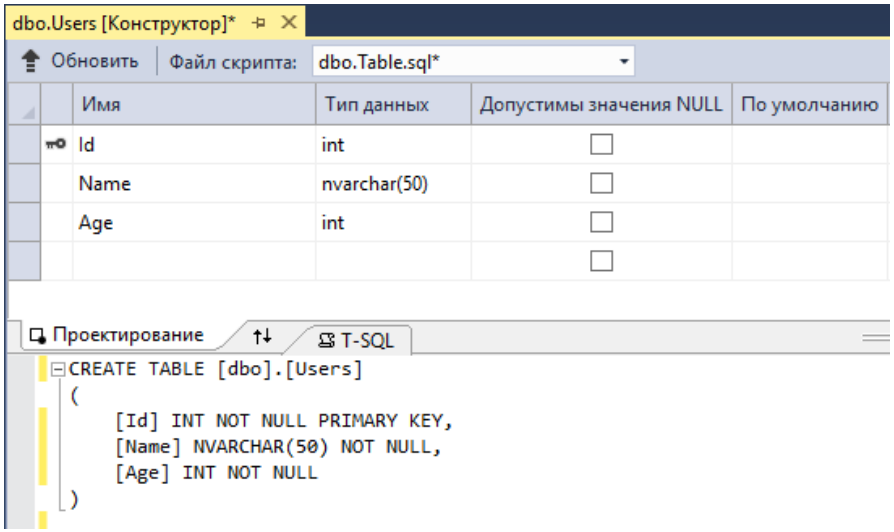


4. Созданная база данных отобразится в окне Server Explorer

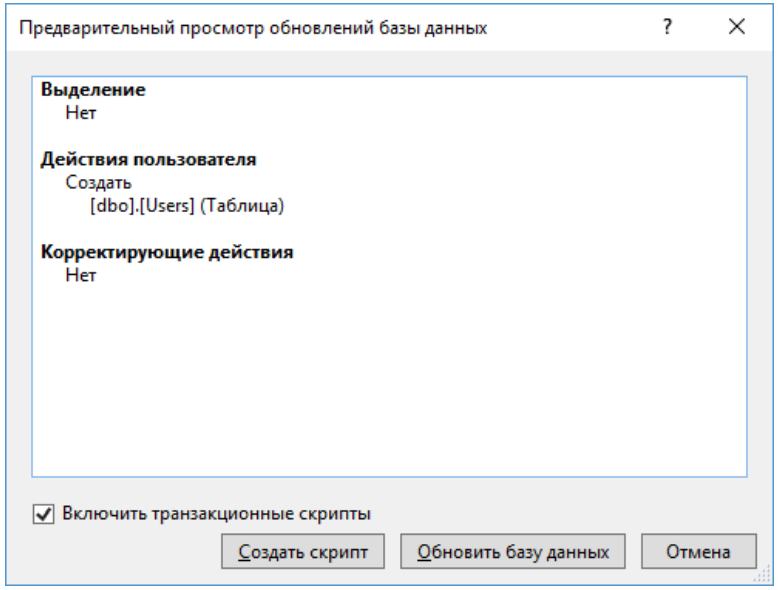


Подход Code First

5. База данных еще пуста, поэтому добавим в нее таблицу. Нажмем правой кнопкой мыши на узел Tables и в появившемся контекстном меню выберем Add New Table. Затем в центральном поле в режиме дизайнера создадим следующее определение таблицы:



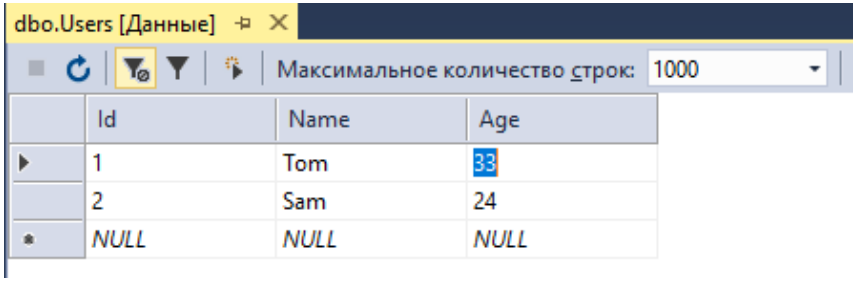
6. Нажмем в верхнем левом углу на кнопку Update. В новом окне нам будет выдана некоторая информация об изменениях, производимых в БД.



7. Нажмем на кнопку Update Database. И после этого будет создана таблица Users.

Подход Code First

8. Мы можем добавить некоторые данные в таблицу. Для этого нажмем на таблицу в окне Server Explorer правой кнопкой мыши и выберем пункт Show Table Data. У нас откроется форма для работы с данными, в которую введем пару строк:



База данных готова.

Добавим подключение в файл конфигурации приложения:

```
<connectionStrings>
    <add name="UserDB" connectionString="Data Source=DESKTOP-
        E4VSQBJ\SQLEXP2016;Initial Catalog=userstoredb;Integrated
        Security=True;" providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Подход Code First

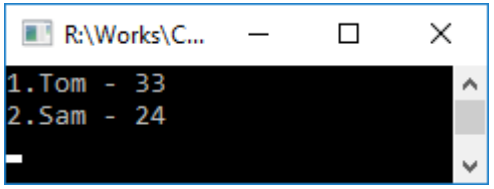
Определим классы модели данных и контекста.

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

class UserContext : DbContext
{
    public UserContext() : base("UserDB")
    { }

    public DbSet<User> Users { get; set; }
}
```

Подход Code First

Для получения данных определим следующий код в консольном приложении.

```
using (UserContext db = new UserContext())
{
    var users = db.Users;
    foreach (User u in users)
    {
        Console.WriteLine($"{u.Id}.{u.Name} - {u.Age}");
    }
}
```



Соглашения по наименованию в Code First

При создании таблиц и их столбцов в базе данных в Entity Framework по умолчанию действуют некоторые соглашения по именованию, которые указывают, какие имена должны быть у таблиц, столбцов, какие типы и т.д.

Типы SQL Server и C# сопоставляются следующим образом:

- int : int
- bit : bool
- char : string
- date : DateTime
- datetime : DateTime
- datetime2 : DateTime
- decimal : decimal
- float : double
- money : decimal
- nchar : string
- ntext : string
- numeric : decimal
- nvarchar : string
- real : float
- smallint : short
- text : string
- tinyint : byte
- varchar : string

Соглашения по наименованию в Code First

NULL и NOT NULL

- Все первичные ключи по умолчанию имеют определение NOT NULL.
- Столбцы, сопоставляемые со свойствами ссылочных типов (string, array), в базе данных имеют определение NULL, а все значимые типы (DateTime, bool, char, decimal, int, double, float) - NOT NULL.
- Если свойство имеет тип Nullable<T>, то оно сопоставляется со столбцом с определением NULL.

Ключи

- Entity Framework требует наличия первичного ключа, так как это позволяет ему отслеживать объекты.
- По умолчанию в качестве ключей EF рассматривает свойства с именем *Id* или *[Название_типа]Id* (например, PostId в классе Post).
- Как правило, ключи имеют тип int или GUID, но также могут представлять и любой другой примитивный тип.

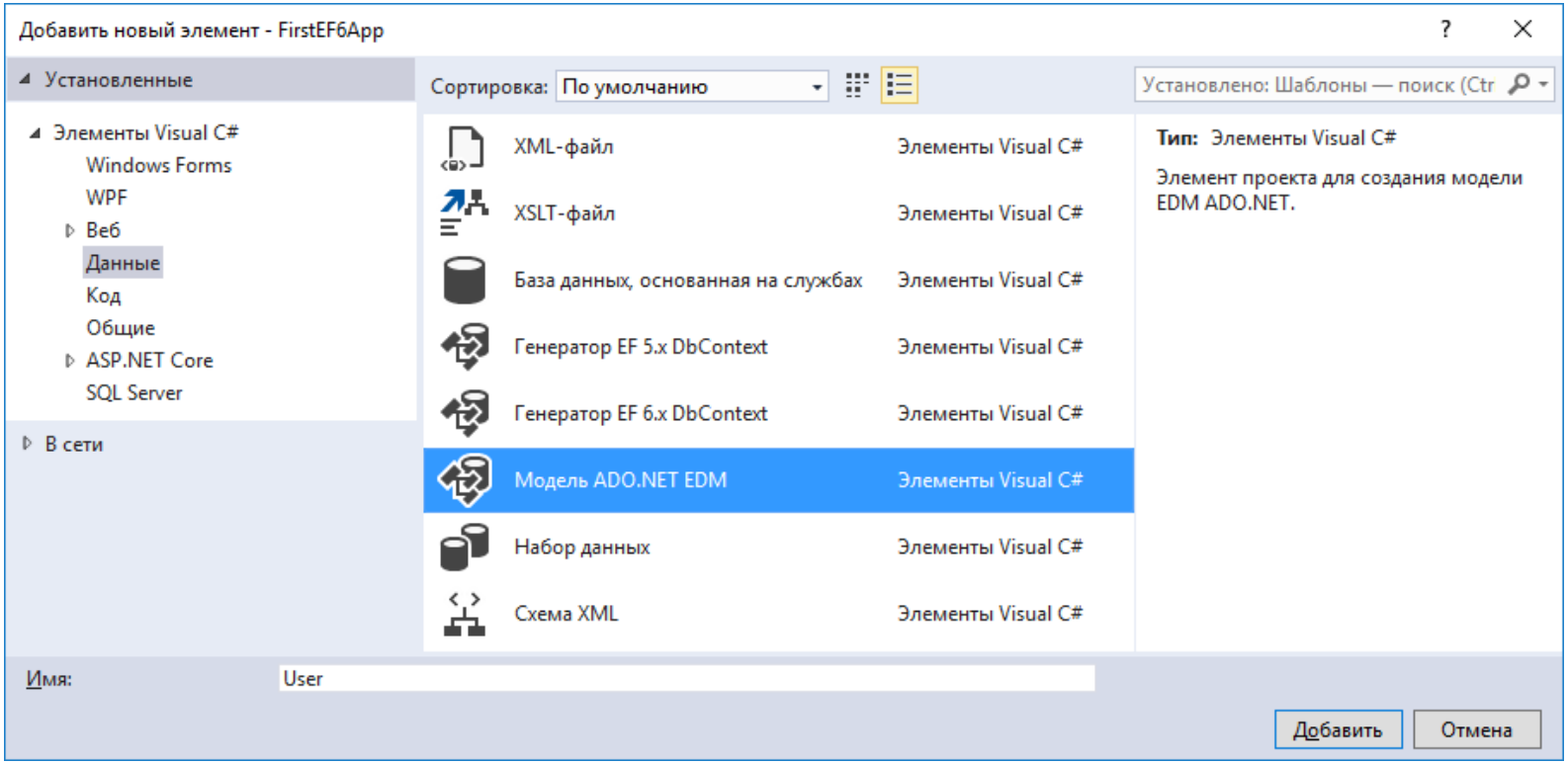
Соглашения по наименованию в Code First

Названия таблиц и столбцов

- С помощью специального класса PluralizationService Entity Framework проводит сопоставление между именами классов моделей и именами таблиц.
- При этом таблицы получают по умолчанию в качестве названия множественное число в соответствии с правилами английского языка, например, класс User - таблица Users, класс Person - таблица People (но не Persons!).
- Названия столбцов получают названия свойств модели.
- Если нас не устраивают названия таблиц и столбцов по умолчанию, то мы можем переопределить данный механизм с помощью Fluent API или аннотаций.

Автоматизация Code First

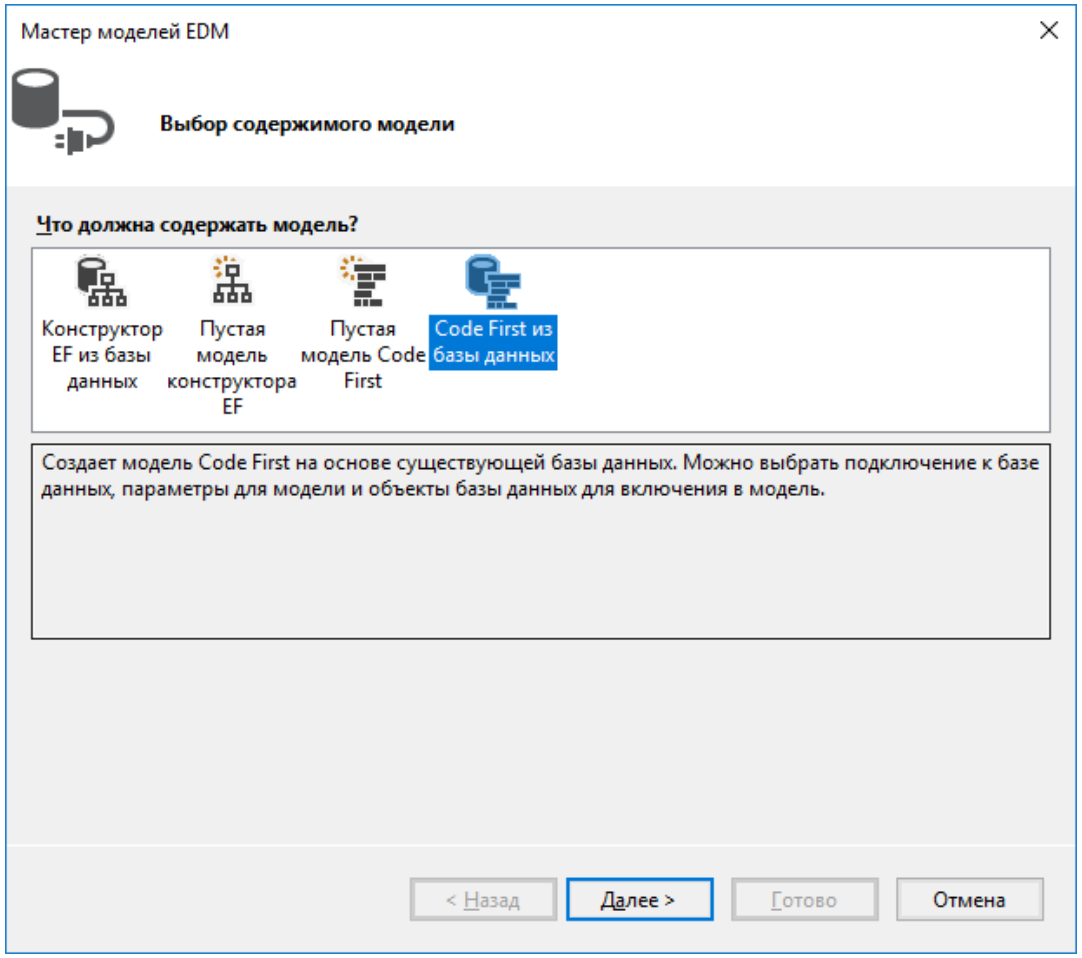
Вручную создавать классы по уже готовой БД со всеми полями и связями между собой довольно утомительно, особенно если таблиц в БД очень много.
Можно автоматизировать этот процесс.
Для этого добавим в проект новый элемент ADO.NET Entity Data Model.



Нажмем «Добавить».

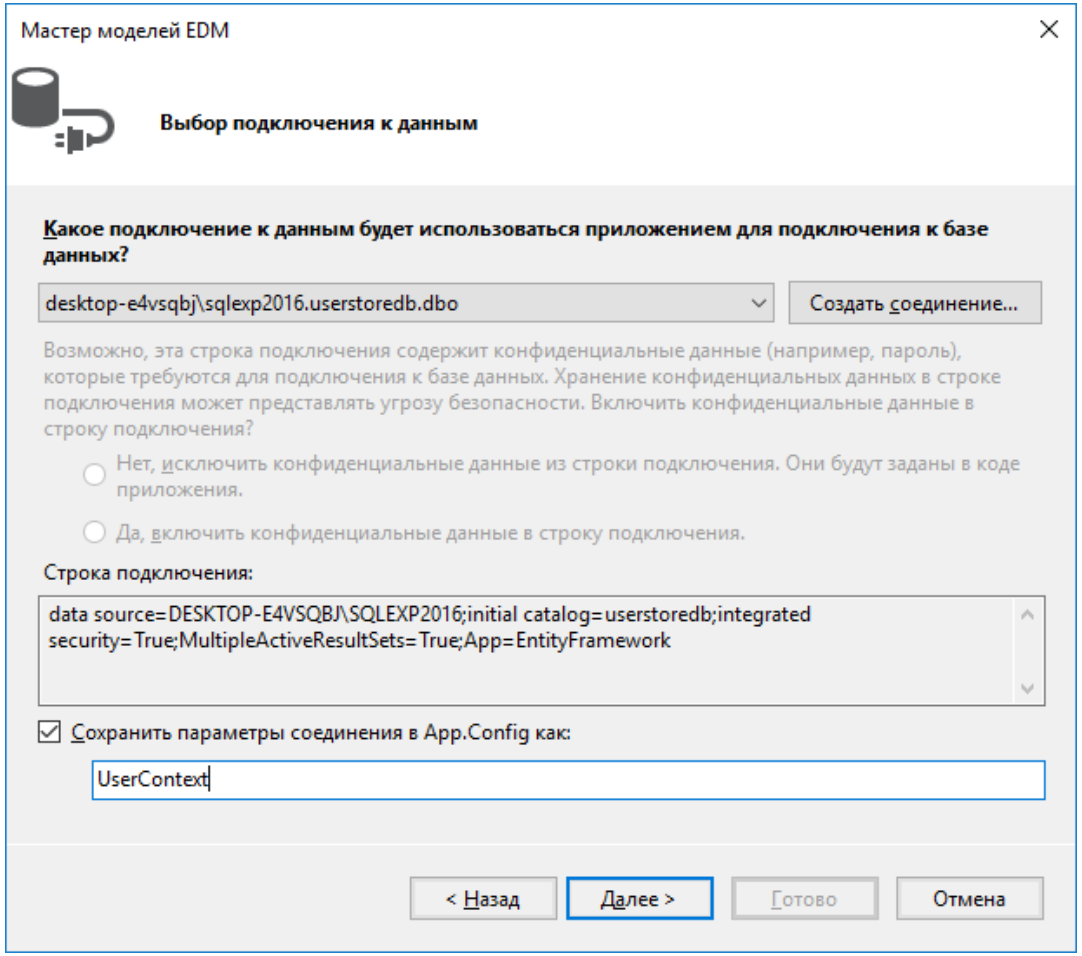
Автоматизация Code First

Нам откроется мастер создания модели. Здесь нам надо выбрать пункт Code First from database:



Автоматизация Code First

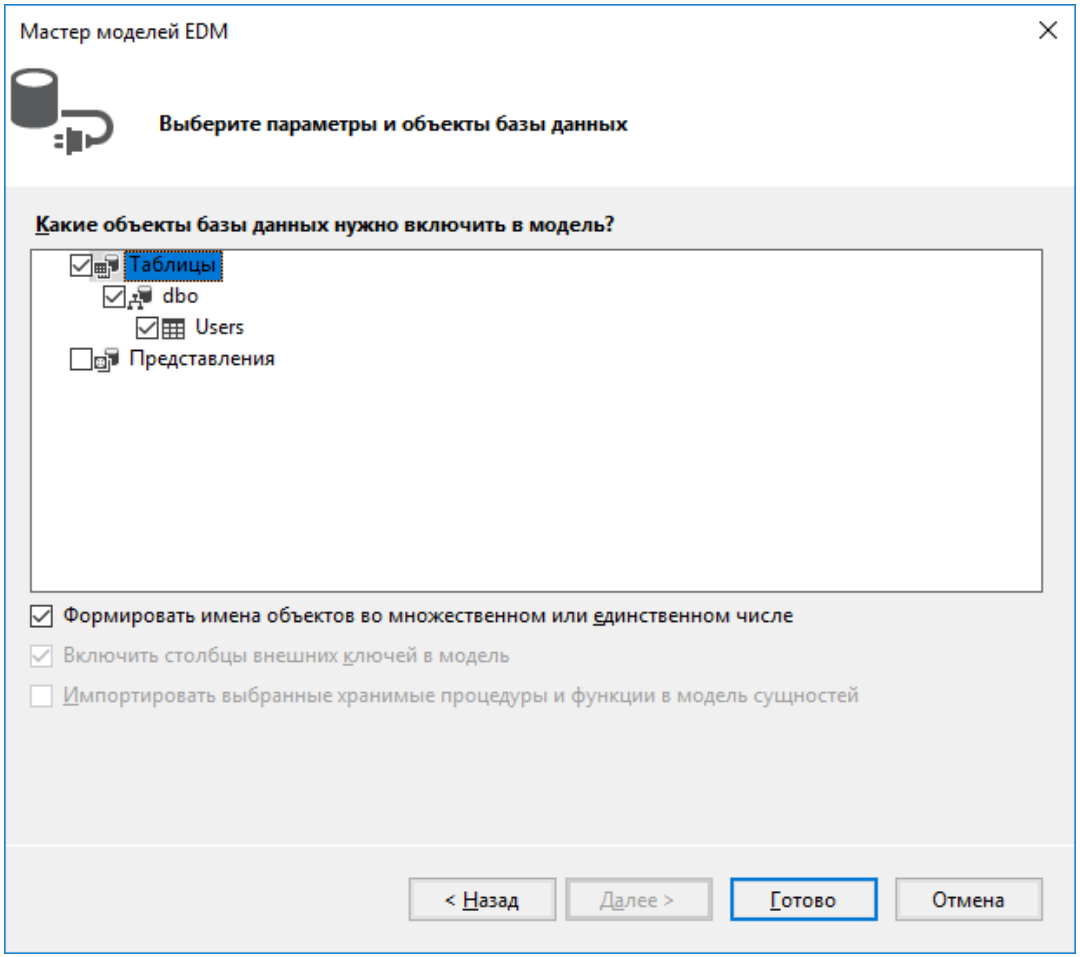
Далее на следующем шаге настройки модели надо будет установить подключение к имеющейся базе данных.



Также здесь можно установить название подключения, которое будет использоваться в файле конфигурации App.config.

Автоматизация Code First

На последнем шаге будет предложено выбрать те таблицы из БД, по которым надо создать модели.



Автоматизация Code First

После этого будут сгенерированы классы моделей.
Например, в примере по единственной таблице в БД будет сгенерирован следующий класс:

```
namespace FirstEF6App
{
    using System;
    using System.Collections.Generic;
    using System.ComponentModel.DataAnnotations;
    using System.ComponentModel.DataAnnotations.Schema;
    using System.Data.Entity.Spatial;

    public partial class User
    {
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int Id { get; set; }

        [Required]
        [StringLength(50)]
        public string Name { get; set; }

        public int Age { get; set; }
    }
}
```

Автоматизация Code First

В файле *App.config* появилось определение подключения.

```
<connectionStrings>
        <add name="UserContext" connectionString="data
        source=DESKTOP-E4VSQBJ\SQLEXP2016;initial
        catalog=userstoredb;integrated security=True;
        MultipleActiveResultSets=True; App=EntityFramework"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

Для полноценной работы осталось добавить класс контекста данных и начать взаимодействовать с базой данных.