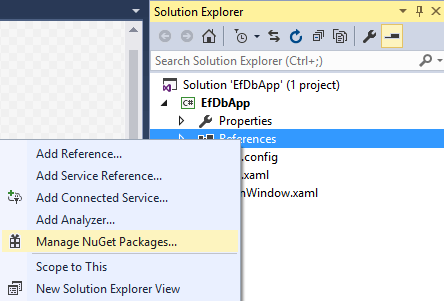
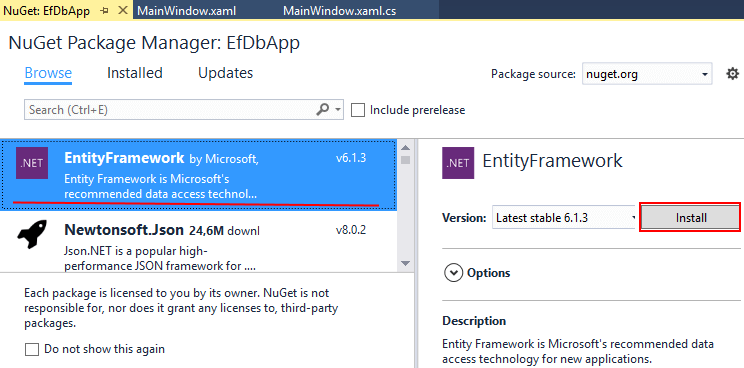
## Работа с Entity Framework

Entity Framework представляет ORM-технологию, которая позволяет абстрагироваться от структуры базы данных и может выполнять автоматически сопоставление таблиц и их данных с моделями классов, определенных разработчиком. Более подробно про эту технологию можно прочитать в руководстве по [Entity Framework](https://metanit.com/sharp/entityframework/), здесь же мы рассмотрим лишь, как сделать простейшее приложение с использованием этого инструмента.

Вначале создадим новый проект. Пусть он называется **EfDbApp**. Первым делом нам надо добавить все необходимые пакеты Entity Framework в наш проект. Для этого в структуре проекта нажмем правой кнопкой мыши на узел **References** (Библиотеки) и в контекстном меню выберем пункт **Manage NuGet Packages...**:



В открывшемся менеджере пакетов NuGet выберем пакет Entity Framework и установим его:



Затем определим строку подключения в файле *App.config*:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | <?xml version="1.0" encoding="utf-8"?>  <configuration>    <configSections>      <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->      <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />    </configSections>    <startup>      <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />    </startup>      <connectionStrings>      <add name="DefaultConnection"           connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=mobiledb;Integrated Security=True"           providerName="System.Data.SqlClient" />    </connectionStrings>      <entityFramework>      <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework" />      <providers>        <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />      </providers>    </entityFramework>  </configuration> |

В данном случае мы будем использовать ту же базу данных **mobiledb**, которую создали в предыдущих темах. Однако даже если у нас нет базы данных mobiledb, то при первом обращении к базе данных Entity Framework создаст ее автоматически.

Теперь добавим в проект новую папку, которую назовем *Models* и в которой определим новый класс Phone:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public class Phone  {      public int Id { get; set; }      public string Title { get; set; }      public string Company { get; set; }      public int Price { get; set; }  } |

Для взаимодействия с базой данных через Entity Framework нам нужен контекст данных, поэтому добавим в папку Models еще один класс, который назовем MobileContext:

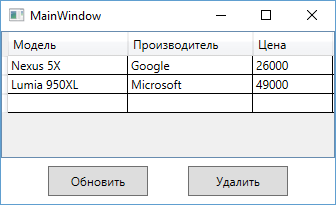
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | using System.Data.Entity;    public class MobileContext : DbContext  {      public MobileContext(): base("DefaultConnection")      {        }      public DbSet<Phone> Phones { get; set; }  } |

Класс контекста наследуется от класса DbContext. В своем конструкторе он передает в конструктор базового класса название строки подключения из файла *App.config*. Также в контексте данных определяется свойство по типу DbSet<Phone> - через него мы будем взаимодействовать с таблицей, которая хранит объекты Phone.

Теперь определим разметку интерфейса:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | <Window x:Class="EfDbApp.MainWindow"          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"          xmlns:d="http://schemas.microsoft.com/expression/blend/2008"          xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"          xmlns:local="clr-namespace:EfDbApp"          mc:Ignorable="d"          Title="MainWindow" Height="250" Width="300">      <Window.Resources>          <Style TargetType="Button">              <Setter Property="Margin" Value="20 8 20 8" />              <Setter Property="Width" Value="100" />              <Setter Property="Height" Value="30" />          </Style>      </Window.Resources>      <Grid>          <Grid.RowDefinitions>              <RowDefinition Height="\*" />              <RowDefinition Height="Auto" />          </Grid.RowDefinitions>          <DataGrid AutoGenerateColumns="False" x:Name="phonesGrid">              <DataGrid.Columns>                  <DataGridTextColumn Binding="{Binding Title}" Header="Модель" Width="100"/>                  <DataGridTextColumn Binding="{Binding Company}" Header="Производитель" Width="110"/>                  <DataGridTextColumn Binding="{Binding Price}" Header="Цена" Width="70"/>              </DataGrid.Columns>          </DataGrid>            <StackPanel HorizontalAlignment="Center"  Grid.Row="1" Orientation="Horizontal">              <Button x:Name="updateButton" Content="Обновить" Click="updateButton\_Click" />              <Button x:Name="deleteButton" Content="Удалить" Click="deleteButton\_Click" />          </StackPanel>      </Grid>  </Window> |

Как и в прошлой теме, на форме будет один элемент DataGrid и две кнопки для обновления и удаления.



Теперь определим в файле кода c# привязку данных и обработчики кнопок:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | using System.Windows;  using EfDbApp.Models;  using System.Data.Entity;    namespace EfDbApp  {      public partial class MainWindow : Window      {          MobileContext db;          public MainWindow()          {              InitializeComponent();                db = new MobileContext();              db.Phones.Load(); // загружаем данные              phonesGrid.ItemsSource = db.Phones.Local.ToBindingList(); // устанавливаем привязку к кэшу                this.Closing += MainWindow\_Closing;          }            private void MainWindow\_Closing(object sender, System.ComponentModel.CancelEventArgs e)          {              db.Dispose();          }            private void updateButton\_Click(object sender, RoutedEventArgs e)          {              db.SaveChanges();          }            private void deleteButton\_Click(object sender, RoutedEventArgs e)          {              if (phonesGrid.SelectedItems.Count>0)              {                  for (int i = 0; i < phonesGrid.SelectedItems.Count; i++)                  {                      Phone phone = phonesGrid.SelectedItems[i] as Phone;                      if (phone != null)                      {                          db.Phones.Remove(phone);                      }                  }              }              db.SaveChanges();          }      }  } |

С помощью следующих строк здесь мы загружаем данные из бд и выполняем привязку к ним элемента DataGrid:

|  |  |
| --- | --- |
| 1  2 | db.Phones.Load();  phonesGrid.ItemsSource = db.Phones.Local.ToBindingList(); |

После окончания работы с контекстом данных нам нужно его утилизовать, а именно вызвать его метод Dispose(). Подходящим местом для этого является обработчик события Closing класса MainWindow.

И как и в прошлой теме, здесь нам достаточно ввести данные в строку грида, чтобы создать новый объект. Также мы можем изменять уже существующие значения в ячейках грида. И чтобы сохранить все изменения, в обработчике кнопки обновления вызывается метод **db.SaveChanges()**.

Для удаления данных сначала вызывается метод db.Phones.Remove(phone) для всех удаляемых объектов, а затем также метод db.SaveChanges.