Базы данных

ORM. Entity Framework. Report Viewer.

[Что такое ORM](#_2et92p0)

[Приложение с использованием Entity Framework](#_tyjcwt)

[Формирование отчётов в виде документов Word и Excel](#_1t3h5sf)

[Расширение функционала, работы с базой данных и формирования отчетов, приложения "Рассыльщик"](#_2s8eyo1)

[Домашнее задание](#_17dp8vu)

[Дополнительные материалы](#_3rdcrjn)

[Используемая литература](#_26in1rg)

# 

# Что такое ORM

Ранее уже разбиралась работа с базами данных с использованием ADO.Net. На данном уроке разбирается другой подход взаимодействия с БД, присутствующий в платформе .Net - Entity Framework, являющийся ORM решением.

Что такое ORM (Object-Relational Mapping)? Объектно-реляционное отображение - технология взаимодействия с БД, связывающая структуру БД и принципы ООП. Данный метод позволяет как бы отображать таблицы и структуру на классы в объектно-ориентированных языках программирования, а само взаимодействие с базой происходит посредством LINQ to Entities, а также Entity SQL.

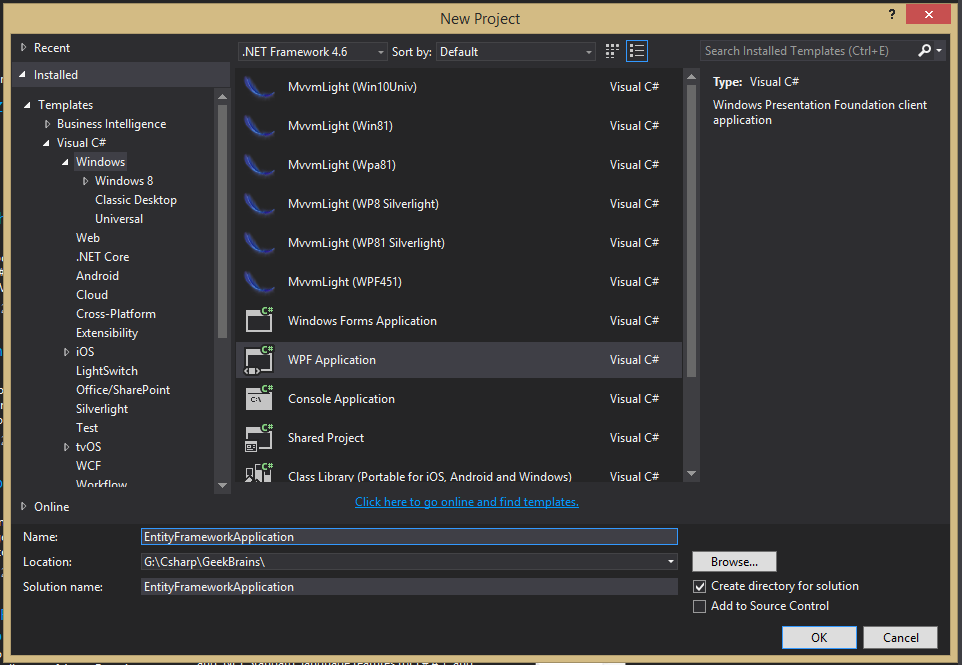
Также существуют различные подходы к самому формированию классов для взаимодействия:

* Database first, когда классы генерируются на основе уже существующей БД;
* Code first, когда сама БД и отношения между таблицами генерируется на основе классов.

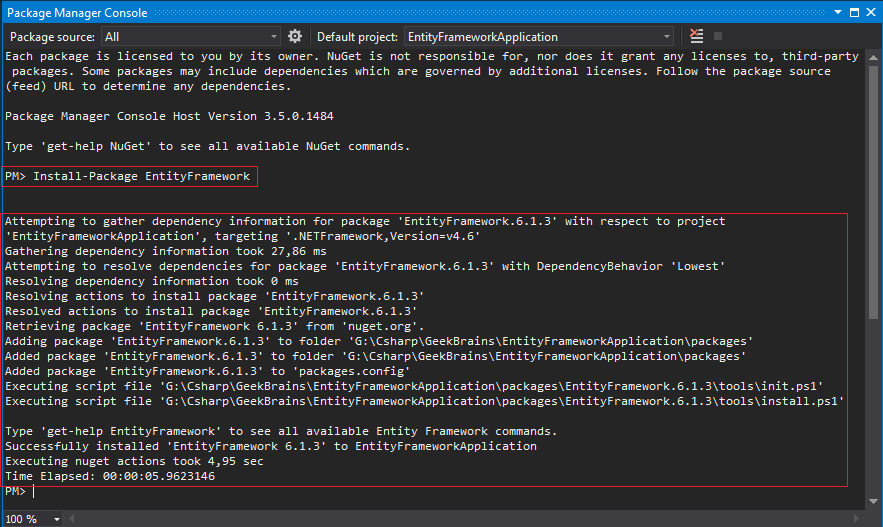
В нашем случае, будем использовать второй подход, как более удобный.

# Приложение с использованием Entity Framework

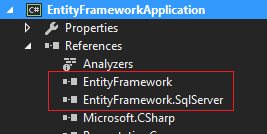
Чтобы разобраться с работой Entity Framework, создаём новое WPF приложение, назовём его - EntityFrameworkApplication.



Далее необходимо установить сам Entity Framework. Сделаем это через Package Manager Console, посредством вызова Install-Package EntityFramework:



Развернув Preferences, мы видим, что необходимые нам библиотеки установлены:



Как уже говорилось, в данном примере будут использовать подход - Code first, т.е. БД будет создаваться согласно написанному коду.

Сначала определимся, какими данными мы будем оперировать. Пускай наше приложение будет предоставлять операции с коллекцией аудиотреков.

Следующий класс будет представлять трек:

|  |
| --- |
| public class Track  {  [Key]  public int TrackId { get; set; }  public string ArtistName { get; set; }  public string TrackName { get; set; }  } |

Атрибут [Key] говорит о том, что переменная TrackId является первичным ключом для данной таблицы. Существуют и другие атрибуты, некоторые из них:

* [Required], означает обязательное свойство, т.е. в БД будет NOT NULL;
* [NotMapped], данное свойство не будет отображаться на таблицу БД и, соответственно, не будет присутствовать столбец в таблице.

В общем случае можно напрямую указывать отношение классов к таблицам, а свойств к столбцам.

|  |
| --- |
| [Table("Tracks")]  public class Track  {  [Key]  public int TrackId { get; set; }  [Required]  [Column("ArtistName")]  public string ArtistName { get; set; }  [Required]  [Column("TrackName")]  public string TrackName { get; set; }  } |

Здесь непосредственно указано, что класс относится к таблице Tracks, а свойство ArtistName к одноимённому полю.

После создания таблицы по данному классу она будет выглядеть следующим образом:

|  |
| --- |
| CREATE TABLE [dbo].[Tracks]  (  [TrackId] INT NOT NULL,  [ArtistName] NVARCHAR(MAX) NOT NULL,  [TrackName] NVARCHAR(MAX) NOT NULL,  CONSTRAINT [PK\_Tracks] PRIMARY KEY CLUSTERED ([TrackId] ASC)  ) |

Также нам понадобится само подключение к базе. Будем использовать MS SQL LocalDB, которая предоставляет наиболее удобный вариант использования БД для разработчиков, при это сохраняя полную совместимость с с полноценной MS SQL.

Внесём в App.config, в раздел configuration следующие изменения:

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

Этими действиями создаётся подключение к БД с именем - SqlLocalDatabaseConnection и база данных - Database.mdf

Ещё одним немаловажным элементом является контекст БД, являющийся наследником DbContext.

Именно с помощью него будут доступны операции с базой данных.

|  |
| --- |
| public class SqlLocalDbContext : DbContext  {  public SqlLocalDbContext() : base("SqlLocalDatabaseConnection")  {  Database.SetInitializer(new DropCreateDatabaseIfModelChanges<SqlLocalDbContext>());  }  public virtual DbSet<Track> Tracks { get; set; }  } |

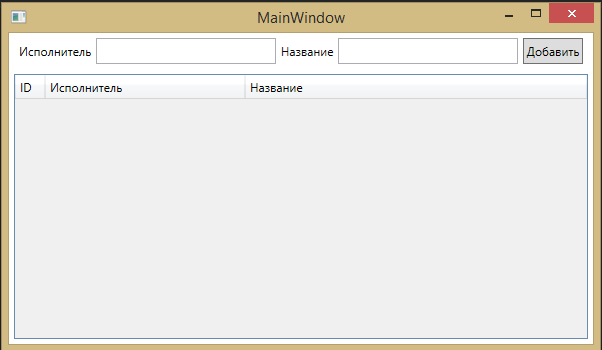
В конструкторе происходит инициализация нашей БД в случае изменения модели. Свойство Tracks, представляет собой коллекцию сущностей, которые мы можем запросить из БД, и над которыми возможно проводить операции, такие как добавление, удаление, изменение и т.д.

Интерфейс приложения будет максимально простым:

|  |
| --- |
| <Window x:Class="EntityFrameworkApplication.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:EntityFrameworkApplication"  mc:Ignorable="d"  Title="MainWindow"  Height="350"  Width="600"  Loaded="MainWindow\_OnLoaded">  <Grid>  <Grid.RowDefinitions>  <RowDefinition Height="Auto"/>  <RowDefinition />  </Grid.RowDefinitions>  <Grid Grid.Row="0"  Margin="5">  <Grid.ColumnDefinitions>  <ColumnDefinition Width="Auto"/>  <ColumnDefinition/>  <ColumnDefinition Width="Auto"/>  <ColumnDefinition/>  <ColumnDefinition Width="70"/>  </Grid.ColumnDefinitions>  <Label Grid.Column="0"  Content="Исполнитель" />  <TextBox Grid.Column="1"  x:Name="ArtistNameTxt"  VerticalContentAlignment="Center"/>  <Label Grid.Column="2"  Content="Название" />  <TextBox Grid.Column="3"  x:Name="TrackNameTxt"  VerticalContentAlignment="Center"/>  <Button Grid.Column="4"  Margin="5 0 5 0"  Name="AddButton"  Content="Добавить"  Click="AddButton\_OnClick"/>  </Grid>  <DataGrid x:Name="Grid"  Grid.Row="1"  Margin="5"  AutoGenerateColumns="False"  IsReadOnly="True"  ItemsSource="{Binding TracksList}">  <DataGrid.Columns>  <DataGridTextColumn Header="ID"  Width="30"  Binding="{Binding TrackId}" />  <DataGridTextColumn Header="Исполнитель"  Width="200"  Binding="{Binding ArtistName}" />  <DataGridTextColumn Header="Название"  Width="\*"  Binding="{Binding TrackName}" />  </DataGrid.Columns>  </DataGrid>  </Grid>  </Window> |

На форме располагается два Label, два TextBox и кнопка для добавления нового трека, а также DataGrid для отображения.

Всё это мы можем увидеть, запустив приложение:



В code behind пропишем код на событие нажатия кнопки, загрузки нашей формы, а также инициализации списка треков TracksList и контекста БД Сontext:

|  |
| --- |
| public partial class MainWindow : Window  {  private SqlLocalDbContext Сontext;  public List<Track> TracksList;  public MainWindow()  {  InitializeComponent();  Сontext = new SqlLocalDbContext();  }  private void MainWindow\_OnLoaded(object sender, RoutedEventArgs e)  {  ReloadTracksList();  }  private void AddButton\_OnClick(object sender, RoutedEventArgs e)  {  if (!string.IsNullOrWhiteSpace(ArtistNameTxt.Text) && !string.IsNullOrWhiteSpace(TrackNameTxt.Text))  {  AddNewTrack();  ReloadTracksList();  }  }  private void AddNewTrack()  {  Сontext.Tracks.Add(  new Track  {  ArtistName = ArtistNameTxt.Text,  TrackName = TrackNameTxt.Text  });  Сontext.SaveChanges();  }  private void ReloadTracksList()  {  TracksList = Сontext.Tracks.ToList();  Grid.ItemsSource = TracksList;  }  } |

Разберём этот код. Основной интерес вызывают два метода - AddNewTrack и ReloadTracksList, один из которых добавляет новый трек в БД, а второй запрашивает все треки из базы. Делается все очень просто и прозрачно.

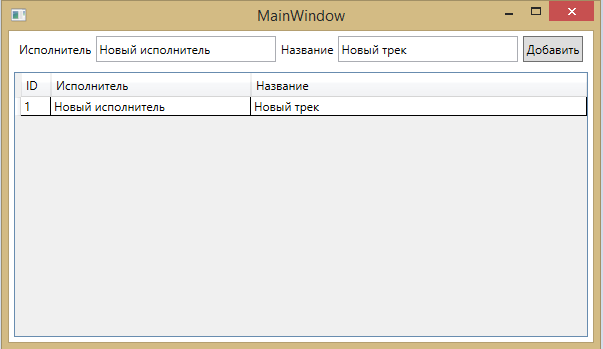
|  |
| --- |
| ...  TracksList = Сontext.Tracks.ToList();  ... |

В данном случае происходит выборка всех треков из контекста и присвоение этого значения свойству TracksList.

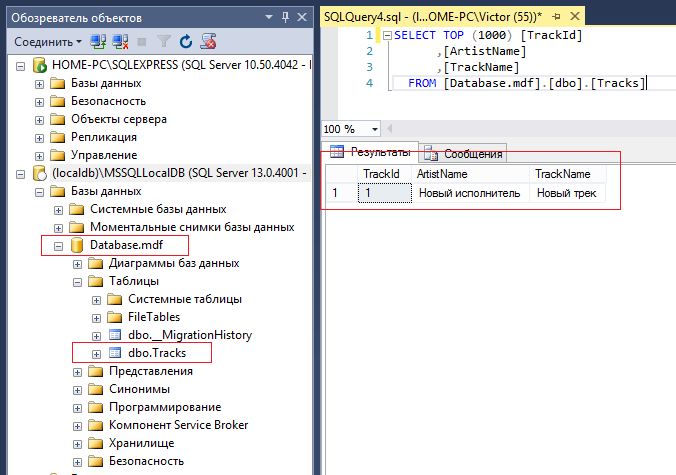
|  |
| --- |
| Сontext.Tracks.Add(  new Track  {  ArtistName = ArtistNameTxt.Text,  TrackName = TrackNameTxt.Text  });  Сontext.SaveChanges(); |

Добавляем к контексту новый объект типа Track, метод SaveChanges сохраняет все изменения, зафиксированные в контексте в БД.

Попробуем добавить новый трек - введём в текстбоксы имя исполнителя и название трека, в нашем случае это будут - “Новый исполнитель” и “Новый трек”. Нажмём кнопку добавить и увидим, что новые данные отобразились в DataGrid:



Также можем убедится, что данные присутствуют и в БД, также как и созданная база Database и таблица Tracks:



# Формирование отчётов в виде документов Word и Excel

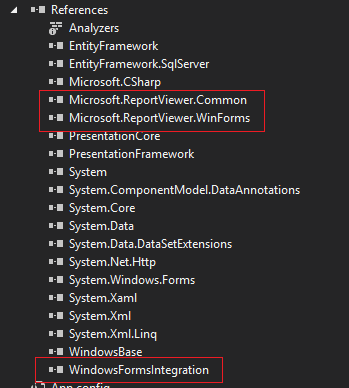
Рассмотрим создание отчётов в нашем приложении. В качестве генератора отчётов будем использовать - Microsoft Report Viewer. Для его использования необходимо добавить несколько библиотек в проект:

* Microsoft.ReportViewer.Common;
* Microsoft.ReportViewer.WinForms;
* WindowsFormsIntegration.

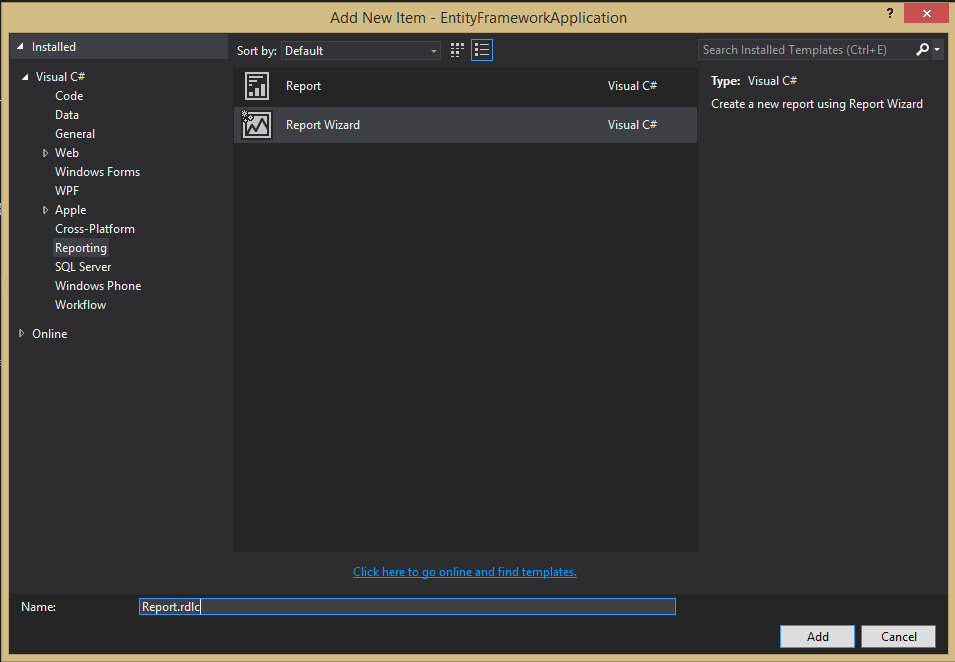
Также нам будет необходимо несколько изменить верстку нашего приложения. Будем использовать табы, на одном будет размещаться наша форма с гридом, а на другом контрол с генератором отчётов.

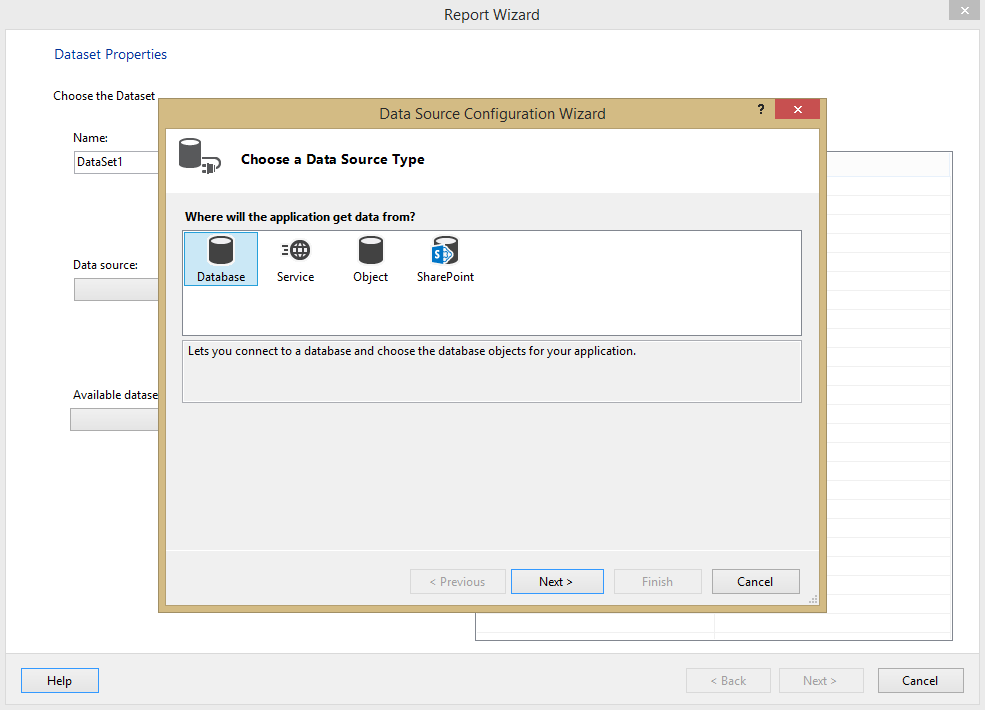
|  |
| --- |
| <TabControl>  <TabItem Header="Треки">  <!-- использовать предыдущий код -->  </TabItem>  <TabItem Header="Отчеты">  <WindowsFormsHost>  <winForms:ReportViewer x:Name="ReportViewer"/>  </WindowsFormsHost>  </TabItem>  </TabControl> |

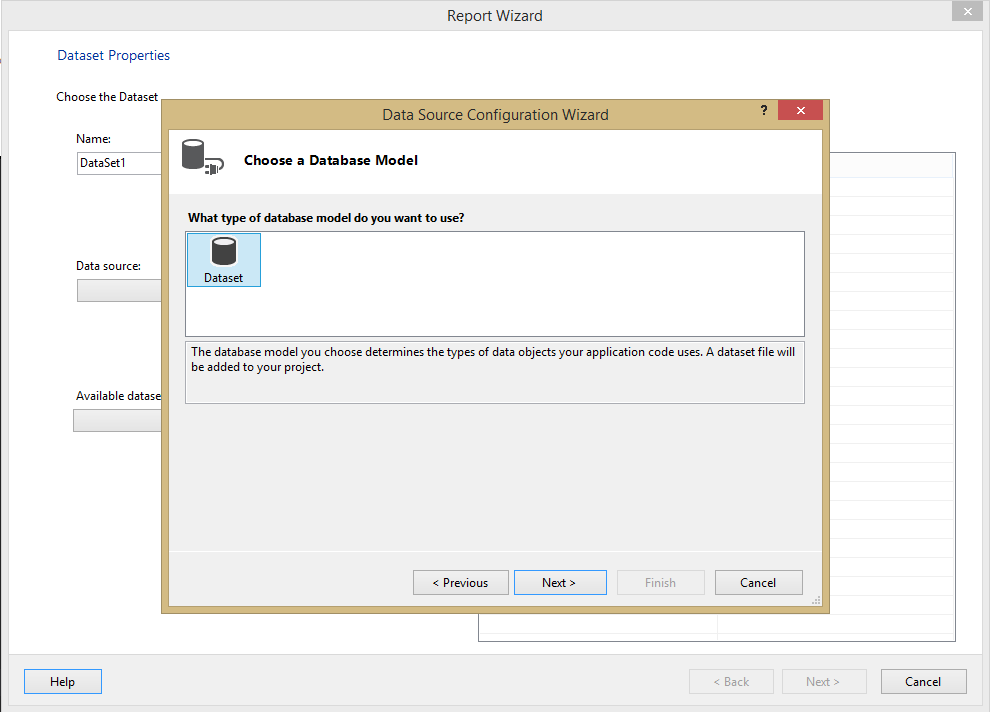
Не забудьте добавить ссылку на сам WinForms!

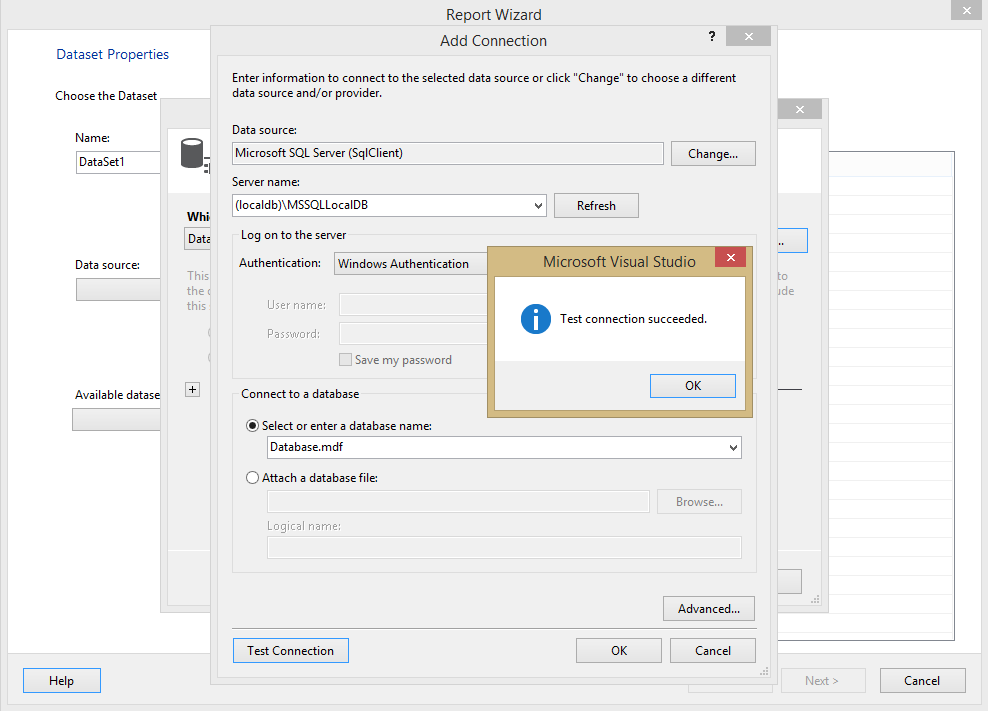


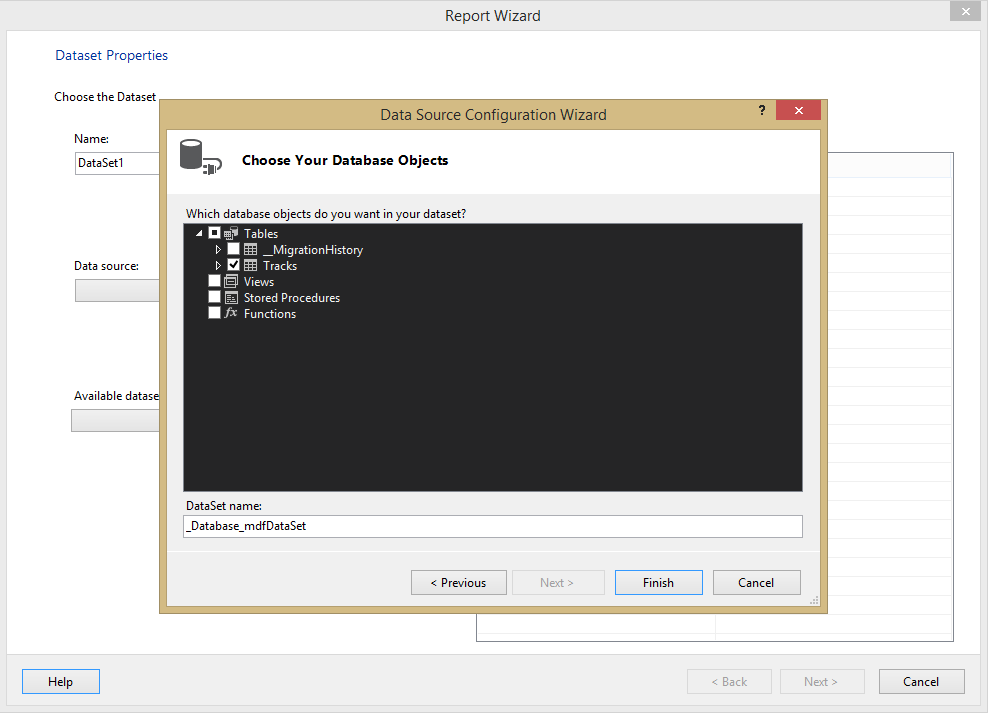
В тэг WindowsFormsHost поместим контрол отчётов, т.к. Его использование обусловлено необходимостью наличия WinFоrms. Добавим новый отчёт:

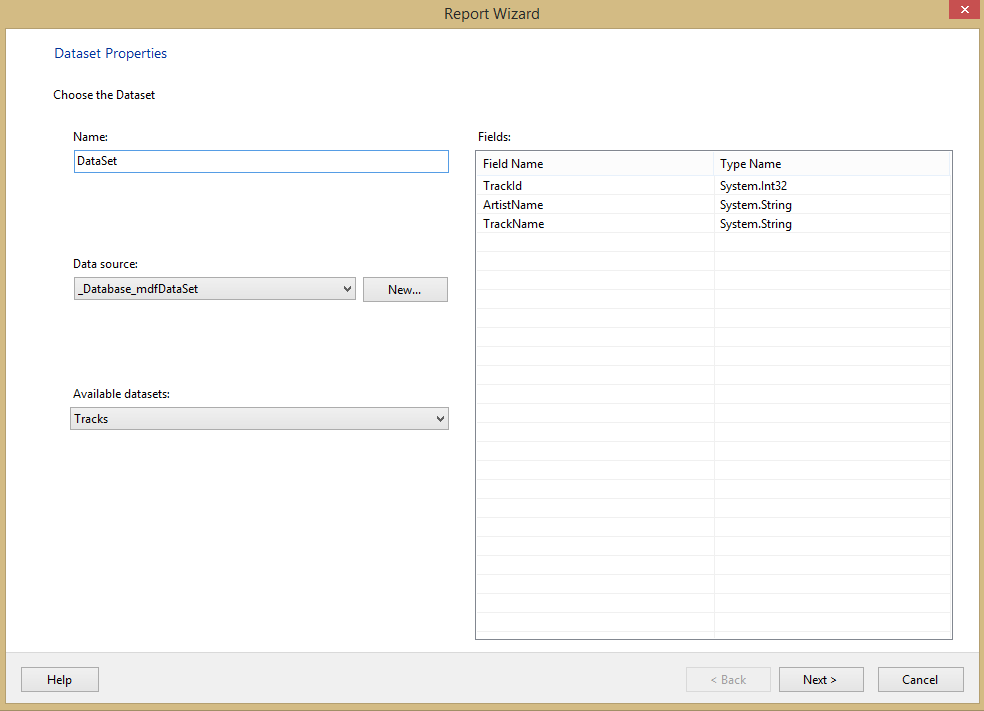


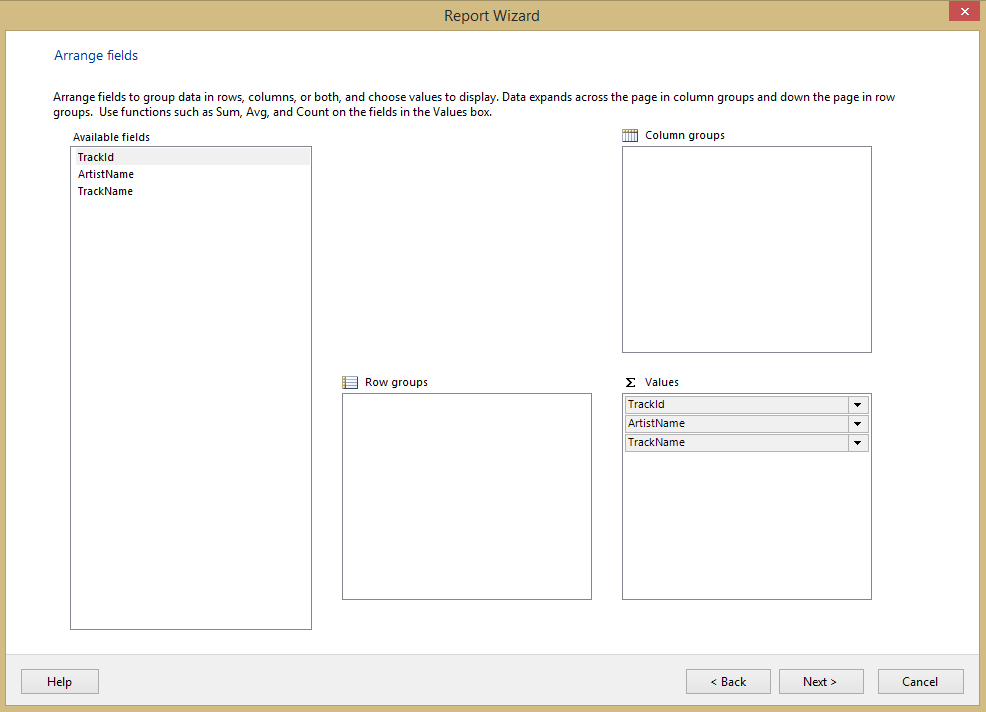


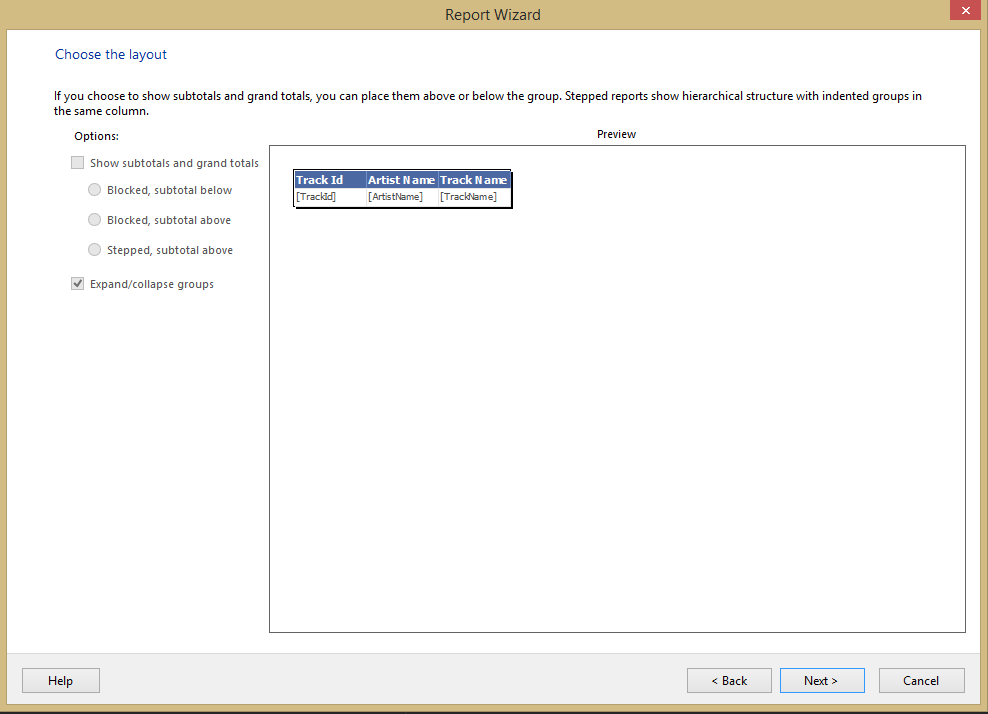










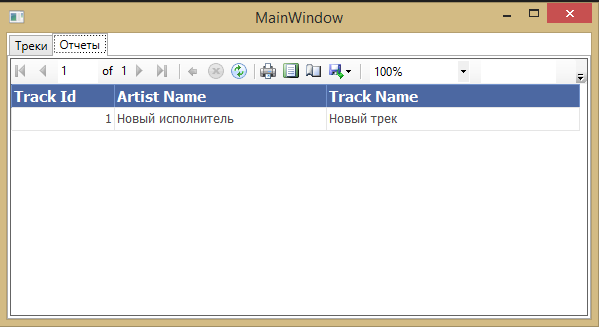


Отчёт создан. Теперь пропишем в code behind метод для его генерации.

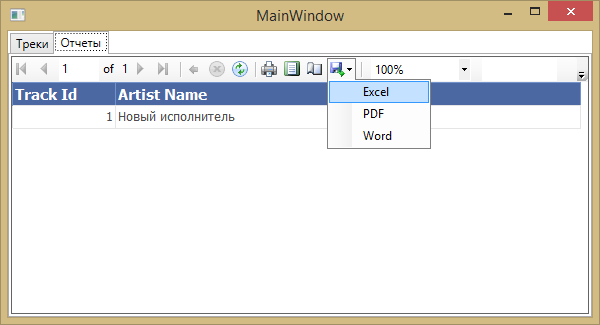
|  |
| --- |
| ...  public MainWindow()  {  InitializeComponent();  Сontext = new SqlLocalDbContext();  ReportViewer.Load += ReportViewerOnLoad;  }  ...  private void ReportViewerOnLoad(object sender, EventArgs eventArgs)  {  ReportDataSource reportDataSource = new ReportDataSource();  \_Database\_mdfDataSet dataset = new \_Database\_mdfDataSet();  dataset.BeginInit();  reportDataSource.Name = "DataSet";  reportDataSource.Value = dataset.Tracks;  ReportViewer.LocalReport.DataSources.Add(reportDataSource);  ReportViewer.LocalReport.ReportPath = "../../Report.rdlc";  dataset.EndInit();  TracksTableAdapter tracksTableAdapter = new TracksTableAdapter {ClearBeforeFill = true};  tracksTableAdapter.Fill(dataset.Tracks);  ReportViewer.RefreshReport();  } |

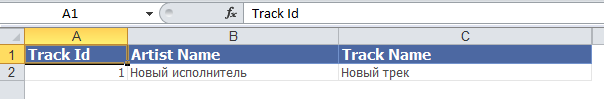
В методе ReportViewerOnLoad происходит привязка, сгенерированного нами DataSource и DataSet к нашему отчёту и заполнение его данными.

При запуске приложения получает сгенерированный отчёт:



Также есть возможность экспортировать отчёт в различные форматы, такие как Exсel, PDF и Word.





Два других способа создания отчетов в Excel:

1.Использование Office COM Interop.

|  |
| --- |
| Excel.Application excelApp;  Excel.Workbook book;  Excel.Worksheet sheet;  Excel.Range range;  try  {  excelApp = new Excel.Application();  excelApp.Visible = true;  book = (Excel.Workbook)(excelApp.Workbooks.Add(Missing.Value));  sheet = (Excel.Worksheet)book.ActiveSheet;  sheet.Cells[1, 1] = "Track Id";  sheet.Cells[1, 2] = "Author";  sheet.Cells[1, 3] = "Name";  sheet.get\_Range("A1", "C1").Font.Bold = true;  sheet.get\_Range("A1", "C1").VerticalAlignment = Excel.XlVAlign.xlVAlignCenter;  string[,] Tracks = new string[3, Сontext.Tracks.Count()];  var tracks = (from t in Сontext.Tracks where t.TrackId == 1 select t);  for (int i = 1; i < Сontext.Tracks.Count() + 1; i++)  {  Track Item = Сontext.Tracks.First(j => j.TrackId == i);  Tracks[i - 1, 0] = Item.TrackId.ToString();  Tracks[i - 1, 1] = Item.ArtistName.ToString();  Tracks[i - 1, 2] = Item.TrackName.ToString();  }  sheet.get\_Range("A2", "C4").Value2 = Tracks;  range = sheet.get\_Range("D2", "D6");  range.Formula = "=B2 & \" \" & C2";  //AutoFit columns A:D.  range = sheet.get\_Range("A1", "D1");  range.EntireColumn.AutoFit();  }  catch (Exception theException)  {  String errorMessage;  errorMessage = "Error: ";  errorMessage = String.Concat(errorMessage, theException.Message);  errorMessage = String.Concat(errorMessage, " Line: ");  errorMessage = String.Concat(errorMessage, theException.Source);  MessageBox.Show(errorMessage, "Error");  } |

2. Использовать готовые библиотеки, в данном случае библиотеку EPPlus (на основе стандарта Open XML). Этот способ содержит гораздо больше преимуществ, чем предыдущий.

|  |
| --- |
| public void GenerateExcel()  {  using (ExcelPackage excelPackage = new ExcelPackage())  {  //Создаем лист  ExcelWorksheet Worksheet = excelPackage.Workbook.Worksheets.Add("Trakcs");  //Загружаем БД на лист, начиная с ячейки А2  Worksheet.Cells["A2"].LoadFromCollection(Сontext.Tracks);  Worksheet.Cells["A1"].Value = "Id";  Worksheet.Cells["B1"].Value = "Artist";  Worksheet.Cells["C1"].Value = "Name";  Worksheet.Cells.AutoFitColumns();  //Изменяем стиль всего диапозона ячеек (первый ряд)  using (ExcelRange range = Worksheet.Cells["A1:XFD1"])  {  range.Style.Font.Bold = true;  range.Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;  range.Style.VerticalAlignment = ExcelVerticalAlignment.Center;  range.Style.Fill.PatternType = ExcelFillStyle.LightGray;  }  excelPackage.SaveAs(new System.IO.FileInfo("test.xlsx"));  }  } |

# Расширение функционала, работы с базой данных и формирования отчетов, приложения "Рассыльщик"

Использование Entity Framework в приложении “Рассыльщик”, проще, на мое усмотрение, нежели использование Linq to SQL.

Приведём на примере таблицы Emails. Создаём класс модели:

|  |
| --- |
| public class Email  {  [Key]  public int EmailId { get; set; }  public string Email { get; set; }  public string Name { get; set; }  } |

Создаём контекст базы данных:

|  |
| --- |
| public class SqlLocalDbContext : DbContext  {  public SqlLocalDbContext() : base("SqlLocalDatabaseConnection")  {  Database.SetInitializer(new DropCreateDatabaseIfModelChanges<SqlLocalDbContext>());  }  public virtual DbSet<Email> Emails { get; set; }  } |

Вносим изменения в DBClass:

|  |
| --- |
| /// <summary>  /// Класс, который отвечает за работу с базой данных  /// </summary>  public class DBclass  {  private SqlLocalDbContext \_сontext = new SqlLocalDbContext();  public List<Email> Emails => \_сontext.Emails.ToList();  } |

Также исправим метод SendMails класса EmailSendServiceClass:

|  |
| --- |
| public void SendMails(List<Email> emails)  {  foreach (var email in emails)  {  SendMail(email.Email, email.Name);  }  } |

Теперь данные по адресам будут считываться из БД посредствам Entity Framework.

# Домашнее задание

1. Реализовать в приложении “Рассыльщик” отчет, выводящий существующих адресатов.
2. \*Реализовать в приложении “Рассыльщик” функциональность позволяющую добавлять, удалять и редактировать адресатов посредством Entity Framework.
3. \*Написать программу для продажи билетов в кино. Продажа выполняется следующим образом: кассир выбирает киносеанс и указывает количество покупаемых билетов. Информация о продажах сохраняется базу данных для последующего анализа. Для заказа сохраняется количество билетов и время продажи. Дополнительно может быть предусмотрено редактирования списка киносеансов. Для киносеанса заполняется время начала и название фильма.
4. \* Есть CSV-файл с таким содержанием:

|  |
| --- |
| Иванов И.И., [ivanov@mail.ru](mailto:ivanov@mail.ru), +7(111) 123-45-67  Петров П.П.,petrov[@mail.ru](mailto:ivanov@mail.ru), +7(222) 123-45-67  Федоров Ф.Ф., fedorov[@mail.ru](mailto:ivanov@mail.ru), +7(333) 123-45-67 |

То есть записи представляют собой значения: ФИО, почта, телефон.

Необходимо написать приложение, которое:

a. импортирует данный файл в базу данных;

b. позволяет редактировать данные.

# Дополнительные материалы

1. <https://msdn.microsoft.com/ru-ru/default.aspx>
2. <https://metanit.com>
3. <http://www.gotreportviewer.com>
4. http://www.c-sharpcorner.com/uploadfile/nipuntomar/report-viewer-control-in-wpf/

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Рихтер Дж. “CLR via C#”
2. Эндрю Троелсен - “ЯЗЫК ПРОГРАММИРОВАНИЯ C#5.0 И ПЛАТФОРМА .NET 4.5”