16. Опишите операторы цикла.

Для выполнения повторяющихся операций в T-SQL применяются циклы. В частности, в T-SQL есть цикл WHILE. Этот цикл выполняет определенные действия, пока некоторое условие истинно.

|  |  |
| --- | --- |
| 1  2 | WHILE условие      {инструкция|BEGIN...END} |

Если в блоке WHILE необходимо разместить несколько инструкций, то все они помещаются в блок BEGIN...END.

Например, вычислим факториал числа:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | DECLARE @number INT, @factorial INT  SET @factorial = 1;  SET @number = 5;    WHILE @number > 0      BEGIN          SET @factorial = @factorial \* @number          SET @number = @number - 1      END;    PRINT @factorial |

То есть в данном случае пока переменная @number не будет равна 0, будет продолжаться цикл WHILE. Так как @number равна 5, то цикл сделает пять проходов. Каждый проход цикла называется итерацией. В каждой итерации будет переустанавливаться значение переменных @factorial и @number.

Другой пример - рассчитаем баланс счета через несколько лет с учетом процентной ставки:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | USE productsdb;    CREATE TABLE #Accounts ( CreatedAt DATE, Balance MONEY)    DECLARE @rate FLOAT, @period INT, @sum MONEY, @date DATE  SET @date = GETDATE()  SET @rate = 0.065;  SET @period = 5;  SET @sum = 10000;    WHILE @period > 0      BEGIN          INSERT INTO #Accounts VALUES(@date, @sum)          SET @period = @period - 1          SET @date = DATEADD(year, 1, @date)          SET @sum = @sum + @sum \* @rate      END;    SELECT \* FROM #Accounts |

Здесь создается временная таблица #Accounts, в которую добавляется в цикле пять строк с данными.

Операторы BREAK и CONTINUE

Оператор BREAK позволяет завершить цикл, а оператор CONTINUE - перейти к новой итерации.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | DECLARE @number INT  SET @number = 1    WHILE @number < 10      BEGIN          PRINT CONVERT(NVARCHAR, @number)          SET @number = @number + 1          IF @number = 7              BREAK;          IF @number = 4              CONTINUE;          PRINT 'Конец итерации'      END; |

Когда переменная @number станет равна 4, то с помощью оператора CONTINUE произойдет переход к новой итерации, поэтому последующая строка PRINT 'Конец итерации' не будет выполняться, хотя цикл продолжится.

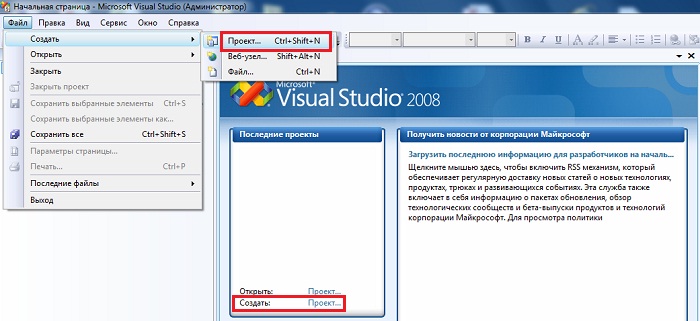
Когда переменная @number станет равна 7, то оператор BREAK произведет выход из цикла, и он завершится.

37 Изложите процесс создания отчетов.

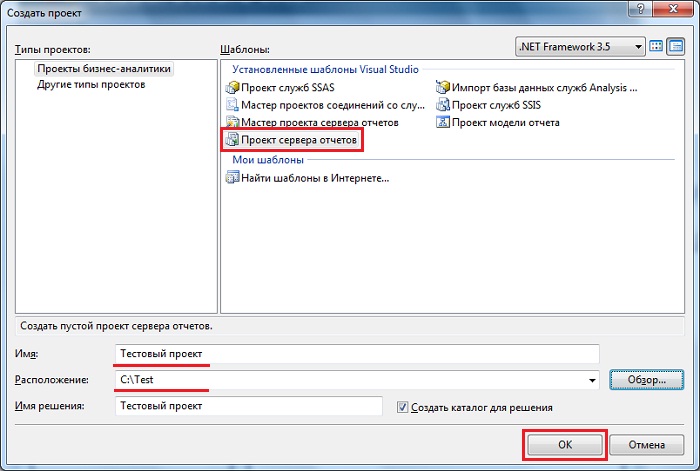
Переходим к разработке отчета, открываем Business Intelligence Development Studio (BIDS) «Пуск ->Все программы-> Microsoft SQL Server 2008 R2-> Среда SQL Server Business Intelligence Development Studio».

Создание проекта сервера отчетов

Далее нам необходимо вызвать окно создания проекта, это мы можем сделать на начальной странице, используя кнопку «Создать проект» или через меню «Файл ->Создать -> Проект».



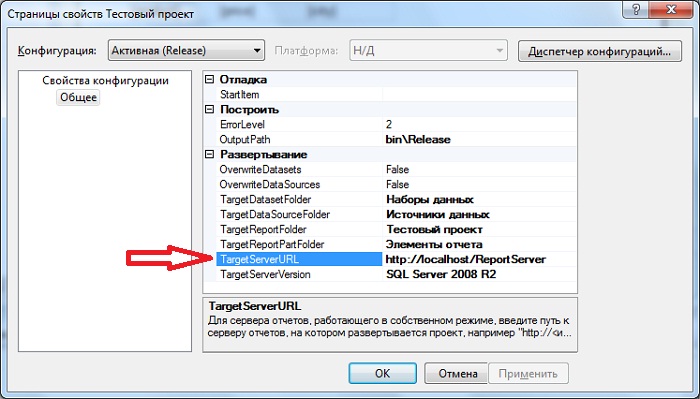
После нам необходимо создать «Проект сервера отчетов», мы можем задать имя этого проекта и указать путь к каталогу, в котором будут лежать все файлы проекта.



Затем давайте сразу зададим адрес сервера отчетов для нашего проекта, сам адрес можно посмотреть в диспетчере конфигурации служб Reporting Services в пункте «URL-адрес веб-службы», по умолчанию на локальном компьютере это — http://localhost/ReportServer.

В BIDS выбираем меню «Проект -> Свойства (конфигурация Release)» и в пункте TargetServerURL указываем наш адрес.

Примечание! Адрес сервера отчетов необходимо указывать для каждой конфигурации проекта.

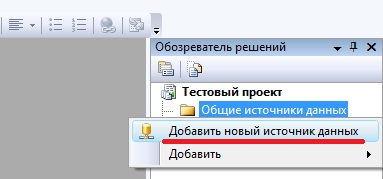


Теперь мы можем сразу из BIDS опубликовывать отчеты на сервере отчетов.

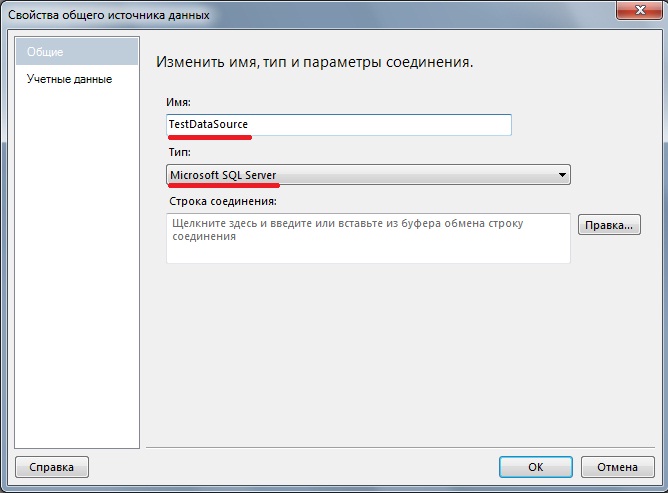
Создание общего источника данных

Далее давайте создадим один источник данных для подключения к SQL серверу, для того чтобы не хранить этот источник в каждом отчете, а хранить только ссылку на него. Это делаем для того, чтобы в случае изменения адреса SQL сервера нам не пришлось править все отчеты, т.е. мы исправим в одном месте и все.

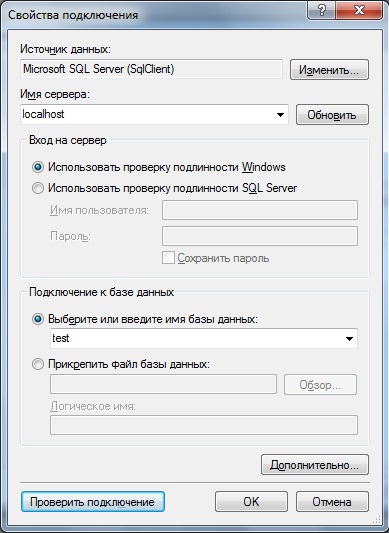
Для этого в «Обозревателе решений» в пункте «Общие источники данных» нажимаем правой кнопкой «Добавить новый источник данных».



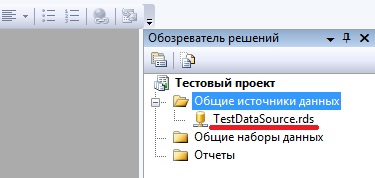
Затем указываем название нашего источника и, соответственно, выбираем, что это за источник. В нашем примере это «Microsoft SQL Server».



Для указания строки подключения нажимаем Правка, и заполняем соответствующие данные, т.е. в нашем случае это localhost, база данных test. Также здесь необходимо задать тип проверки подлинности, если указать тип «Использовать проверку подлинности Windows», то вход на SQL сервер будет выполнен от имени учетной записи, под которой Вы загрузили операционную систему. Если указать «Использовать проверку подлинности SQL Server», то, соответственно, на сервере должна быть заведена специальная учетная запись и, конечно же, сам сервер должен быть настроен на такой способ проверки подлинности. Для примера я буду использовать проверку подлинности Windows.

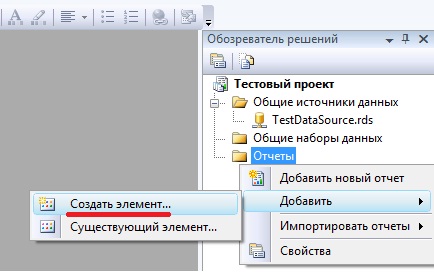


Можете нажать «Проверить подключение», и если все хорошо нажать «ОК». После чего строка подключения у нас отобразится, и мы можем нажать «ОК» для создания общего источника данных. В обозревателе решений у нас появится только что созданный источник данных.



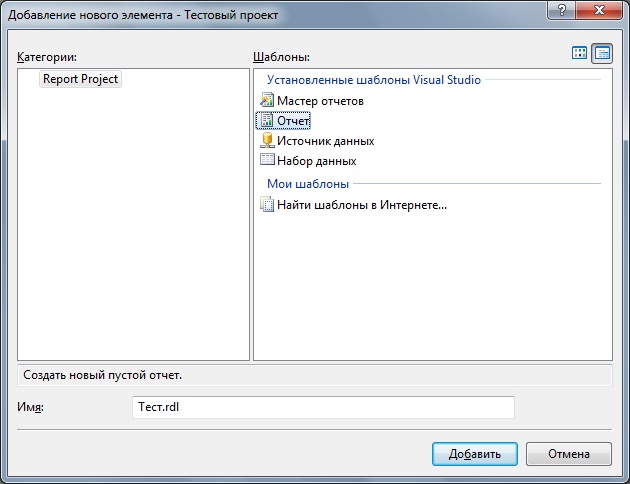
Создание отчета

Теперь давайте создадим сам отчет, для этого в обозревателе решений щелкаем правой кнопкой по пункту «Отчеты ->Добавить ->Создать элемент»



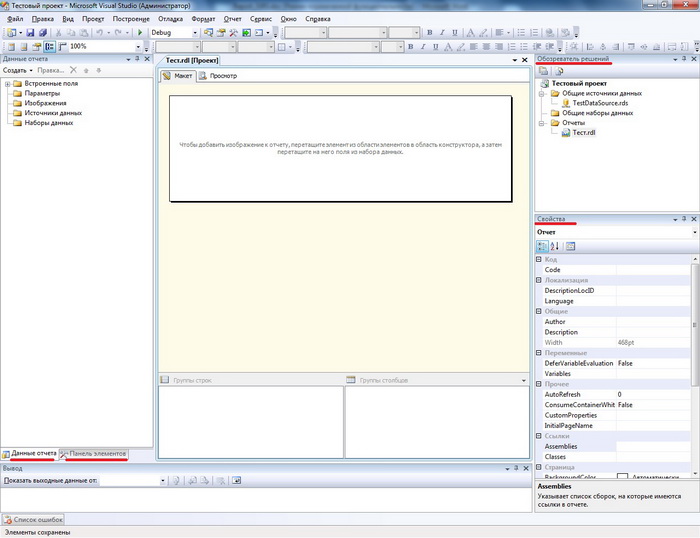
Примечание! Если Вы заметили, есть пункт «Добавить новый отчет», но если Вы его нажмете, то у Вас запустится мастер создания отчета, лично мне удобнее создавать отчеты с чистого листа без использования мастера.

Затем в окне «Добавление нового элемента» в шаблонах мы выбираем «Отчет» и в пункте имя задаем имя нашего отчета.



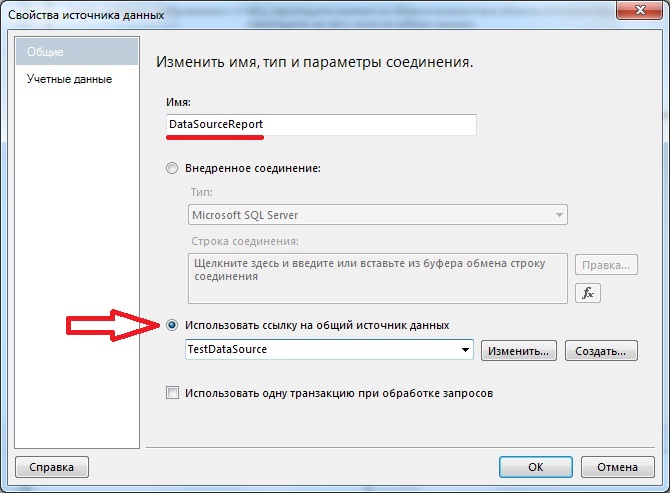
Далее Вы можете настроить среду, как Вам удобно, но так что бы под рукой у Вас были следующие окна: обозреватель решений, панель элементов, свойства и данные отчета, так как эти окна будут требоваться достаточно часто.

Например, мне удобно вот так



Теперь создаем источник данных для отчета, т.е. как договорились, создаем ссылку на общий источник данных. Для этого в окне «Данные отчета» жмем «Создать ->Источник данных» или также можно щелкнуть правой кнопкой по пункту источники данных.

Где мы задаем имя источника данных и выбираем пункт «Использовать ссылку на общий источник данных», жмем «ОК».

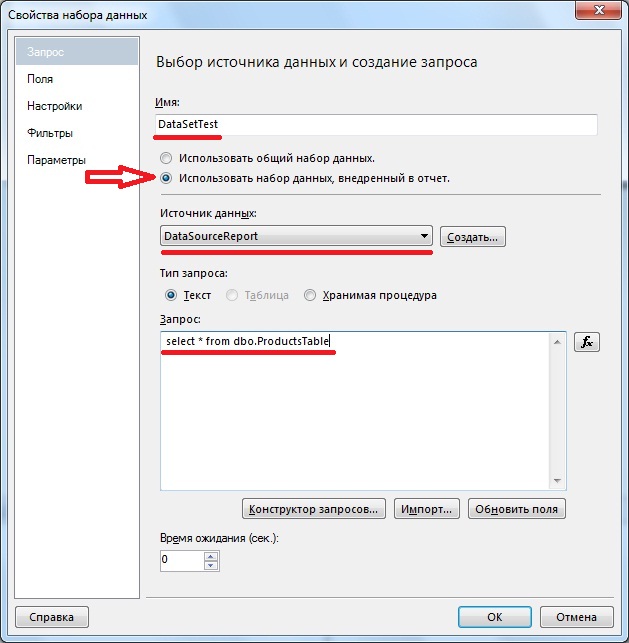


Затем добавляем набор данных, предлагаю использовать простой запрос, который будет выбирать все данные из нашей тестовой таблицы

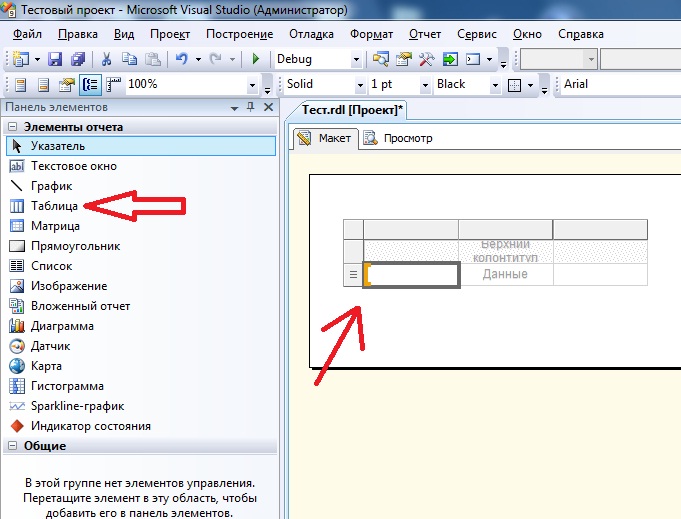
SELECT \* FROM dbo.ProductsTable

Жмем «Создать ->Набор данных» или снова через правую кнопку.

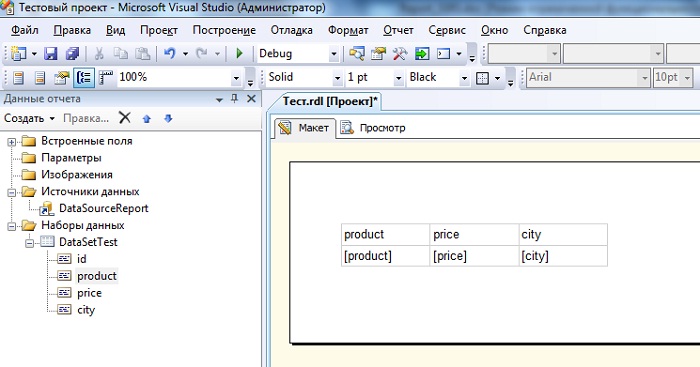
В окне «Свойства набора данных» мы задаем имя набора данных, выбираем пункт «Использовать набор данных, внедренный в отчет» и в текст запроса соответственно вставляем наш запрос, жмем «ОК»



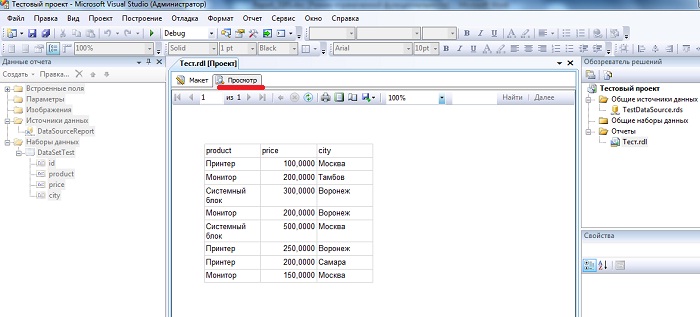
Нам осталось спроектировать сам шаблон вывода информации, для этого на макет с «Панели элементов» перетащите элемент «Таблица». Этот элемент отлично подходит для отображения табличных данных.



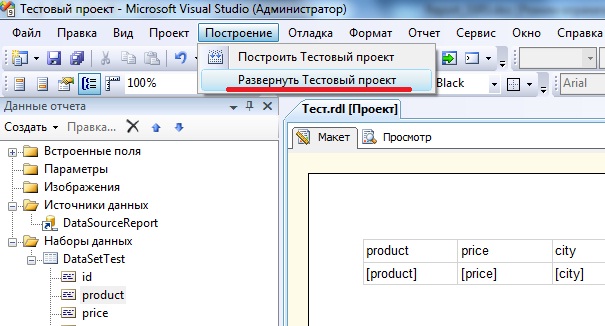
Теперь можно перетащить поля с набора данных в эту таблицу, для отображения.



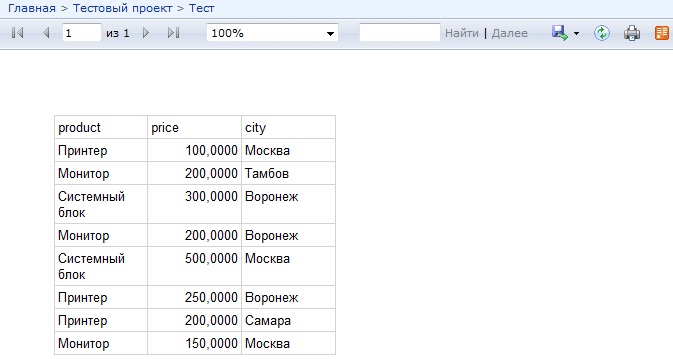
В принципе можно сразу проверить работу данного отчета, для этого перейдите на вкладку «Просмотр», т.е. таким образом можно производить отладку без опубликования отчета на сервере.



А для опубликования отчета и всего проекта на сервере отчетов, необходимо в пункте меню «Построение» нажать «Развернуть Тестовый проект»



После чего, если Вы зайдете в «Диспетчер отчетов», по умолчанию это http://localhost/Reports, то Вы увидите, что у Вас там появился каталог «Тестовый проект», в котором лежит отчет «Тест», а также появился соответствующий источник данных. Для запуска отчета перейдите в каталог и нажмите на отчет.



На этом предлагаю заканчивать, в следующих материалах мы рассмотрим процесс создания более сложных отчетов с использованием группировок, параметров, интерактивных сортировок и других крутых возможностей SQL Server Reporting Services.

Практическое задание

73 «Композиторы и их произведения»

1. Был запущен Sql Server management studio и создана база данных KR2Poet

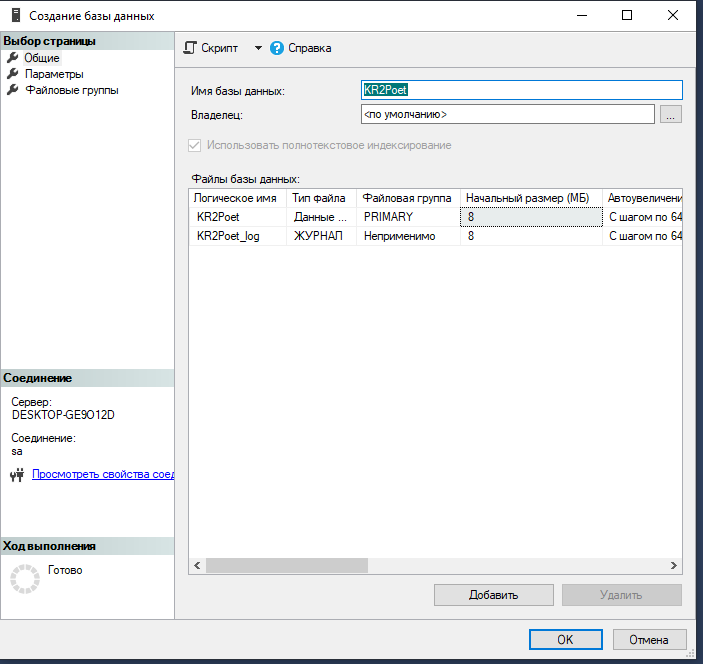


Рисунок 1 – Создание базы данных

Далее была создана таблица ComposerList имеющая следующую структуру

Таблица 1. Данные таблицы ComposerList.

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Пояснение | Тип данных | Ключ поля |
| ComposerKey | Ключ | bigint | Первичный |
| Name | Имя | ntext |  |
| Surname | Фамилия | ntext |  |
| MiddleName | Отчество | ntext |  |
| Genre | Жанр написанной музыки | ntext |  |
| Numberofworks | Количество произведений | bigint |  |
| BirthDate | Дата рождения | ntext |  |
| DeathDate | Дата смерти | ntext |  |
| BirthYear | Год Рождения | bigint |  |
| DeathYear | Год Cмерти | bigint |  |

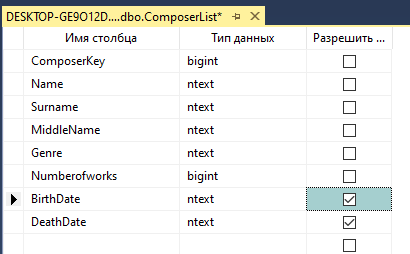


Рисунок 2 – Таблица ComposerList

Далее была создана таблица ComposerWorksList имеющая следующую структуру

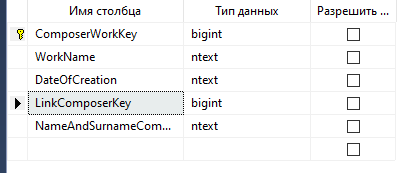


Рисунок 3 – Таблица ComposerWorksList

Далее была создана связь между таблицами ComposerList и ComposerWorksList

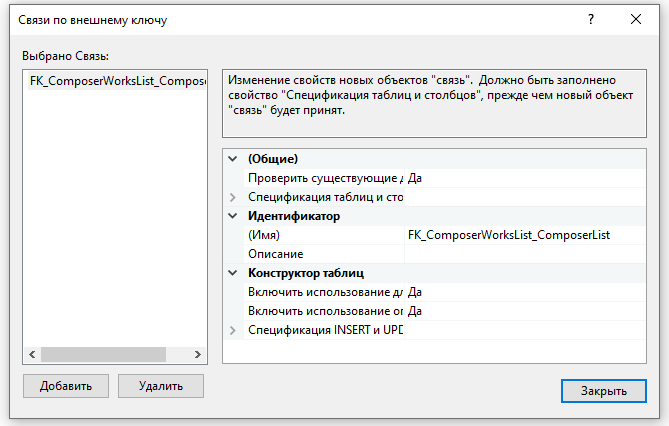


Рисунок 4 – Создание связи между таблицами ComposerList и ComposerWorksList

Далее была создана таблица MusicGenresList имеющая следующую структуру

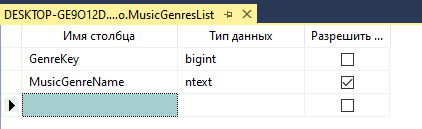


Рисунок 5 – Таблица MusicGenresList

Далее таблица ComposerList была заполнена данными

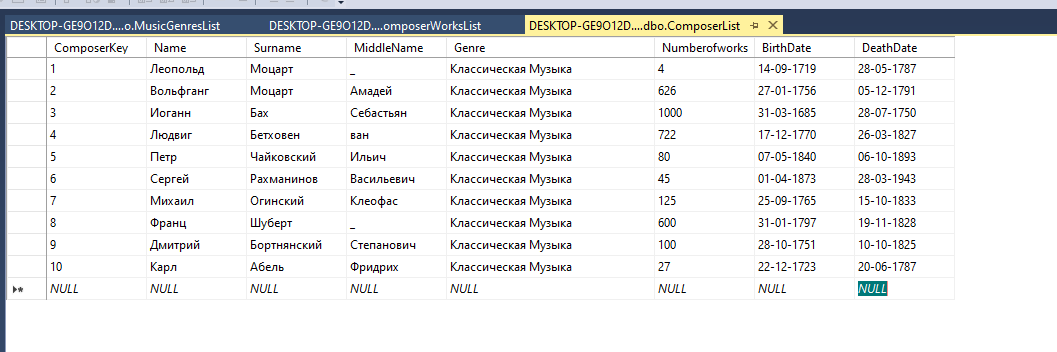


Рисунок 6 – Данные в таблице ComposerList

Далее таблица ComposerWorksList была заполнена данными

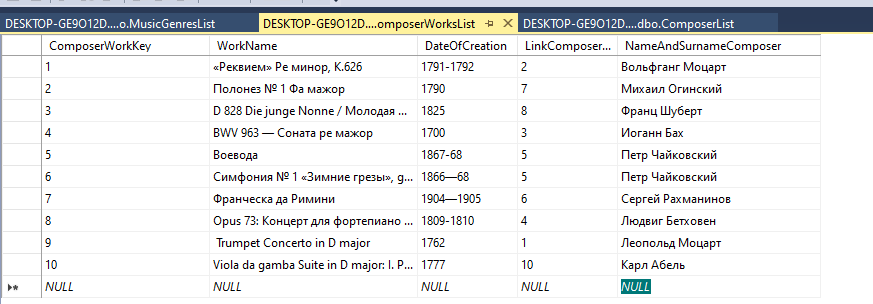


Рисунок 7 – Данные в таблице ComposerWorksList

Далее таблица MusicGenresList была заполнена данными

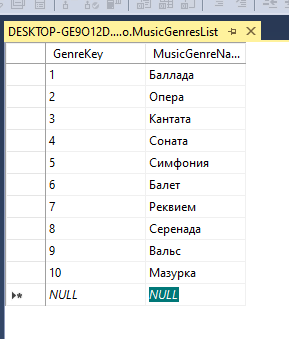


Рисунок 8 – Данные в таблице MusicGenresList

Далее были написаны запросы на языке SQL

- на вывод некоторых полей из двух таблиц;

**Определение:**

SELECT используется для создания выборок из базы данных MySQL.

С помощью SELECT можно:

выводить данные из одной или нескольких таблиц и колонок.

вывести все данные из таблицы используя оператор   \* .

выводить данные с условием, используя команду WHERE.

ограничить количество выводимых записей, используя команду LIMIT.

задать смещение колонок, используя команду OFFSET.

**Сравнение**. Значение одного выражения сравнивается со значением дру­гого выражения. Например, такое условие отбора используется для вы­бора всех офисов, находящихся в восточном регионе, или всех служащих, фактические объемы продаж которых превышают плановые.

SELECT ComposerList.Name,ComposerList.Surname, ComposerWorksList.WorkName, ComposerWorksList.LinkComposerKey

FROM ComposerList, ComposerWorksList

WHERE ComposerList.ComposerKey = ComposerWorksList.LinkComposerKey

- на вывод данных по условию, представляющему выражение

типа сравнения;

**Определение:**

SELECT используется для создания выборок из базы данных MySQL.

С помощью SELECT можно:

выводить данные из одной или нескольких таблиц и колонок.

вывести все данные из таблицы используя оператор   \* .

выводить данные с условием, используя команду WHERE.

ограничить количество выводимых записей, используя команду LIMIT.

задать смещение колонок, используя команду OFFSET.

**Сравнение**. Значение одного выражения сравнивается со значением дру­гого выражения. Например, такое условие отбора используется для вы­бора всех офисов, находящихся в восточном регионе, или всех служащих, фактические объемы продаж которых превышают плановые.

с логическим оператором «И»;

SELECT ComposerList.Name,ComposerList.Surname, ComposerWorksList.WorkName, ComposerWorksList.LinkComposerKey

FROM ComposerList, ComposerWorksList

WHERE ComposerList.ComposerKey = ComposerWorksList.LinkComposerKey AND ComposerList.Surname LIKE 'Моцарт'

с логическим оператором «ИЛИ»;

SELECT ComposerList.Name,ComposerList.Surname, ComposerWorksList.WorkName, ComposerWorksList.LinkComposerKey

FROM ComposerList, ComposerWorksList

WHERE ComposerList.ComposerKey = ComposerWorksList.LinkComposerKey AND (ComposerList.Surname LIKE 'Моцарт' OR ComposerList.Surname LIKE 'Огинский')

- с вычислениями над полями БД;

Определение: Запросы с вычисляемыми полями - поле, содержимое которого является результатом расчета по содержимому других полей, оно существует только в результирующей таблице запроса. В запросе над полями можно производить вычисления. Результат вычисления образует вычисляемое поле в таблице, создаваемой по запросу. При каждом выполнении запроса производится вычисление с использованием текущих значений полей.

Пример команды

SELECT ComposerKey,BirthYear, DeathYear, (DeathYear- BirthYear) AS Age FROM ComposerList

- параметрический.

Определение: Параметрический запрос- это запрос, при выполнении которого пользователю необходимо ввести значение требуемого ему параметра.

Пример команды

INSERT INTO ComposerList (ComposerKey, Name, Surname, MiddleName, Genre, Numberofworks, BirthDate, DeathDate, BirthYear, DeathYear) VALUES (insertedkey, insname, inssurname, insmiddlename, insgenre, insnumworks, insbirthdate, insdeathdate, insbirthyear, insdeathyear)

Далее была разработана форма c#

При открытии программы вам предстает главное окно программы и страница «Работы»

Страница «Работы» cлужит для просмотра информации о произведениях композиторов

Главное окно программы содержит следующие компоненты:

- NavigationView служит для перехода между страницами

- ContentPresenter служит для показа содержимого страниц

- TitleBar служит для показа названия программы и кнопок управления

Страница «Работы» содержит следующие компоненты:

- Label -служит для показа названия страницы;

- DataGrid – служит для показа информации находящийся в базе данных;

- Textbox – служит для ввода данных добавляемых в базу данных или обновления существующих данных а также для ввода запроса для поиска по базе данных;

- Button – отправка новых данных в базу данных или обновление существующих;

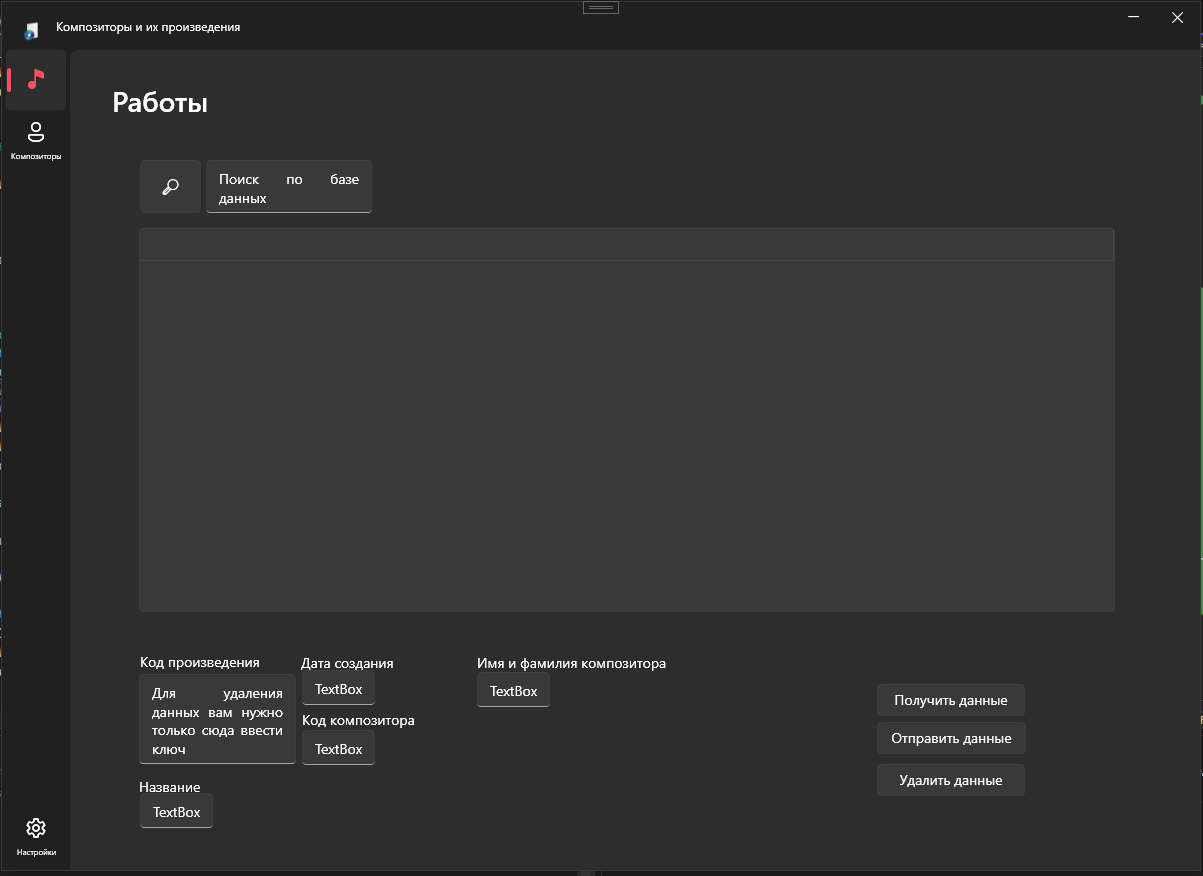


Рисунок 9. Главное окно программы и страница «Работы»

Также присутствует страница «Композиторы»

Страница «Композиторы» cлужит для просмотра списка композиторов и информации о них

Страница «Композиторы» содержит следующие компоненты:

- Label -служит для показа названия страницы;

- DataGrid – служит для показа информации находящийся в базе данных;

- Textbox – служит для ввода данных добавляемых в базу данных или обновления существующих данных а также для ввода запроса для поиска по базе данных;

- Button – отправка новых данных в базу данных или обновление существующих;

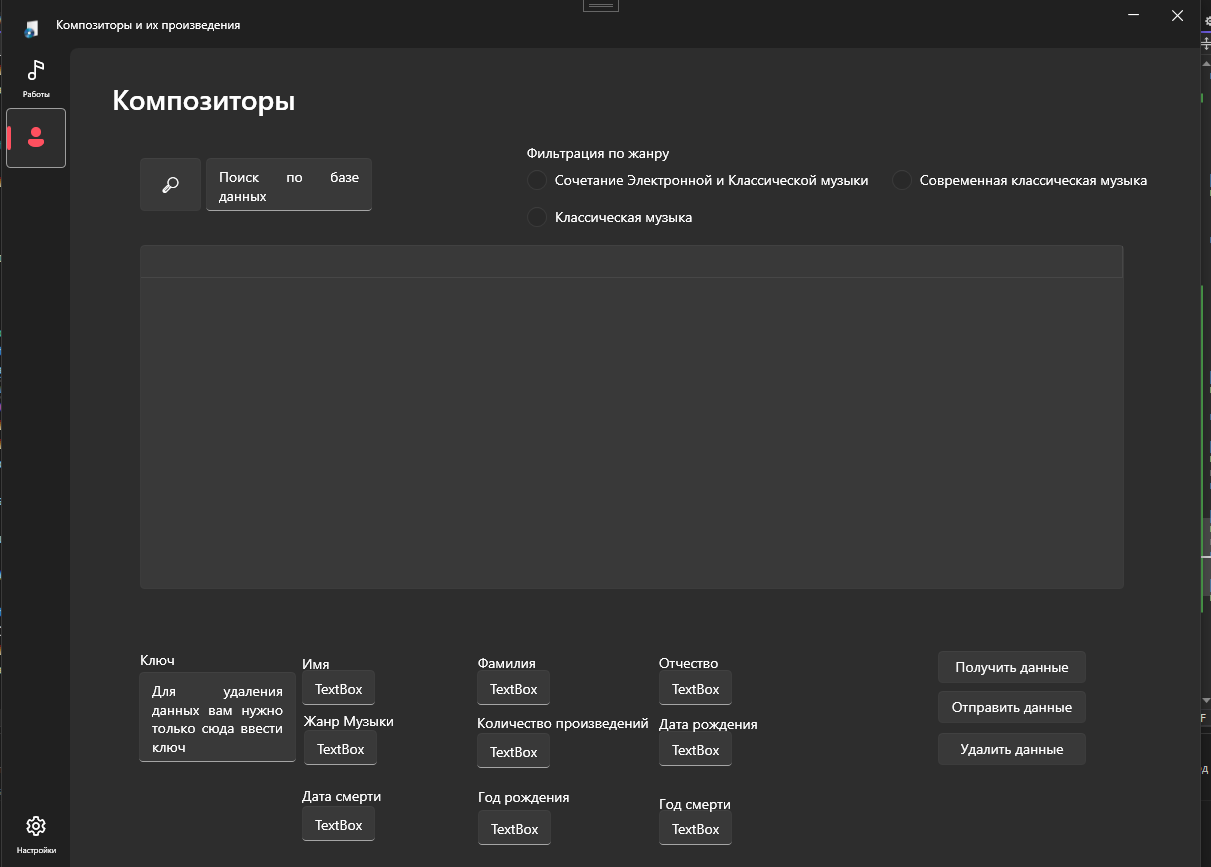


Рисунок 10. Страница «Композиторы»

Полный код форм приведен ниже

using ComposerAndTheirWorks.ViewModels.Pages;

using Wpf.Ui.Controls;

using System.Data.SqlClient;

using ComposerAndTheirWorks.Views.Windows;

using System.Data;

using System.Windows.Controls;

using System.Windows;

namespace ComposerAndTheirWorks.Views.Pages

{

public partial class DataPage : INavigableView<DataViewModel>

{

public DataViewModel ViewModel { get; }

public DataPage(DataViewModel viewModel)

{

ViewModel = viewModel;

DataContext = this;

InitializeComponent();

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

try

{

SqlConnection cnn;

var conStr = MainWindow.formations;

cnn = new SqlConnection(conStr);

cnn.Open();

Console.Write("OPENING DB CONNECTION!!! \n");

Console.Write("[MENU] Connect db clicked - test \n");

var select = "SELECT \* FROM ComposerList";

var commandBuilder = new SqlCommand(select, cnn);

commandBuilder.ExecuteNonQuery();

var dataAdapter = new SqlDataAdapter(commandBuilder);

var ds = new DataTable("Avtobusx");

dataAdapter.Fill(ds);

datagrid1.ItemsSource = ds.DefaultView;

Console.WriteLine("Connection established and the datagrid filled!");

cnn.Close();

}

catch (Exception ex)

{

Console.Write("Error");

}

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

try

{

var CompKey = Edit1.Text;

var NameOfWork = Edit2.Text;

var CreationDate = Edit3.Text;

var LinkCompKey = Edit4.Text;

var CompNameSurname = Edit5.Text;

var WorksCount = Edit6.Text;

var BirthDate = Edit7.Text;

var DeathDate = Edit8.Text;

var BirthYear = Edit9.Text;

var DeathYear = Edit10.Text;

var query =

"INSERT INTO ComposerList(ComposerKey, Name, Surname, MiddleName,Genre,Numberofworks,BirthDate,DeathDate,BirthYear,DeathYear) " +

"Values('" + CompKey + "', '" + NameOfWork + "', '" + CreationDate + "', '" + LinkCompKey + "','" +

CompNameSurname + "','" + WorksCount + "','" + BirthDate + "','" + DeathDate + "', '" + BirthYear + "','" + DeathYear + "' )";

SqlConnection cnn;

var conStr = MainWindow.formations;

cnn = new SqlConnection(conStr);

cnn.Open();

var command = new SqlCommand(query, cnn);

command.ExecuteNonQuery();

Console.WriteLine("Data inserted!\n Closing connection ");

cnn.Close();

Console.WriteLine("Connection has been closed , database ready for next operation!");

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

}

private void Button\_Click\_2(object sender, RoutedEventArgs e)

{

SqlConnection cnn;

var conStr = MainWindow.formations;

cnn = new SqlConnection(conStr);

cnn.Open();

Console.Write("OPENING DB CONNECTION!!! \n");

var select = "DELETE FROM ComposerList WHERE ComposerKey=@a2";

var commandBuilder = new SqlCommand(select, cnn);

commandBuilder.Parameters.AddWithValue("a2", Edit1.Text);

try

{

commandBuilder.ExecuteNonQuery();

Console.WriteLine("Delete operation successful!");

}

catch (SqlException ex)

{

Console.WriteLine(ex.ToString());

}

}

private void Button\_Click\_3(object sender, RoutedEventArgs e)

{

try

{

SqlConnection cnn;

var conStr = MainWindow.formations;

cnn = new SqlConnection(conStr);

cnn.Open();

Console.Write("OPENING DB CONNECTION!!! \n");

Console.Write("[MENU] Connect db clicked - test \n");

var select = "Select \* FROM ComposerList WHERE Surname Like '%" + searchbox.Text + "%'";

var commandBuilder = new SqlCommand(select, cnn);

commandBuilder.ExecuteNonQuery();

var dataAdapter = new SqlDataAdapter(commandBuilder);

var ds = new DataTable("Avtobusx");

dataAdapter.Fill(ds);

datagrid1.ItemsSource = ds.DefaultView;

Console.WriteLine("Connection established and the datagrid filled!");

cnn.Close();

}

catch (Exception ex)

{

Console.Write("Error");

}

}

private void RadioButton\_Click(object sender, RoutedEventArgs e)

{

try

{

SqlConnection cnn;

var conStr = MainWindow.formations;

var filterquery= "Cочетание Электронной и Классической музыки";

cnn = new SqlConnection(conStr);

cnn.Open();

Console.Write("OPENING DB CONNECTION!!! \n");

Console.Write("[MENU] Connect db clicked - test \n");

var select = "Select \* FROM ComposerList WHERE Genre Like '%" + filterquery + "%'";

var commandBuilder = new SqlCommand(select, cnn);

commandBuilder.ExecuteNonQuery();

var dataAdapter = new SqlDataAdapter(commandBuilder);

var ds = new DataTable("Avtobusx");

dataAdapter.Fill(ds);

datagrid1.ItemsSource = ds.DefaultView;

Console.WriteLine("Connection established and the datagrid filled!");

cnn.Close();

}

catch (Exception ex)

{

Console.Write("Error");

}

}

private void genre3\_Checked(object sender, RoutedEventArgs e)

{

try

{

SqlConnection cnn;

var conStr = MainWindow.formations;

var filterquery = "Современная классическая музыка";

cnn = new SqlConnection(conStr);

cnn.Open();

Console.Write("OPENING DB CONNECTION!!! \n");

Console.Write("[MENU] Connect db clicked - test \n");

var select = "Select \* FROM ComposerList WHERE Genre Like '%" + filterquery + "%'";

var commandBuilder = new SqlCommand(select, cnn);

commandBuilder.ExecuteNonQuery();

var dataAdapter = new SqlDataAdapter(commandBuilder);

var ds = new DataTable("Avtobusx");

dataAdapter.Fill(ds);

datagrid1.ItemsSource = ds.DefaultView;

Console.WriteLine("Connection established and the datagrid filled!");

cnn.Close();

}

catch (Exception ex)

{

Console.Write("Error");

}

}

private void genre2\_Click(object sender, RoutedEventArgs e)

{

try

{

SqlConnection cnn;

var conStr = MainWindow.formations;

var filterquery = "Классическая музыка";

cnn = new SqlConnection(conStr);

cnn.Open();

Console.Write("OPENING DB CONNECTION!!! \n");

Console.Write("[MENU] Connect db clicked - test \n");

var select = "Select \* FROM ComposerList WHERE Genre Like '%" + filterquery + "%'";

var commandBuilder = new SqlCommand(select, cnn);

commandBuilder.ExecuteNonQuery();

var dataAdapter = new SqlDataAdapter(commandBuilder);

var ds = new DataTable("Avtobusx");

dataAdapter.Fill(ds);

datagrid1.ItemsSource = ds.DefaultView;

Console.WriteLine("Connection established and the datagrid filled!");

cnn.Close();

}

catch (Exception ex)

{

Console.Write("Error");

}

}

}

}

using ComposerAndTheirWorks.ViewModels.Pages;

using ComposerAndTheirWorks.Views.Windows;

using System.Data.SqlClient;

using System.Data;

using Wpf.Ui.Controls;

using System;

using System.Windows.Controls;

namespace ComposerAndTheirWorks.Views.Pages

{

public partial class DashboardPage : INavigableView<DashboardViewModel>

{

public DashboardViewModel ViewModel { get; }

public DashboardPage(DashboardViewModel viewModel)

{

ViewModel = viewModel;

DataContext = this;

InitializeComponent();

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

try

{

SqlConnection cnn;

var conStr = MainWindow.formations;

cnn = new SqlConnection(conStr);

cnn.Open();

Console.Write("OPENING DB CONNECTION!!! \n");

Console.Write("[MENU] Connect db clicked - test \n");

var select = "SELECT \* FROM ComposerWorksList";

var commandBuilder = new SqlCommand(select, cnn);

commandBuilder.ExecuteNonQuery();

var dataAdapter = new SqlDataAdapter(commandBuilder);

var ds = new DataTable("Avtobusx");

dataAdapter.Fill(ds);

datagrid1.ItemsSource = ds.DefaultView;

Console.WriteLine("Connection established and the datagrid filled!");

cnn.Close();

}

catch (Exception ex)

{

Console.Write("Error");

}

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

try

{

var CompKey = Edit1.Text;

var NameOfWork = Edit2.Text;

var CreationDate = Edit3.Text;

var LinkCompKey = Edit4.Text;

var CompNameSurname = Edit5.Text;

var query =

"INSERT INTO ComposerWorksList(ComposerWorkKey, WorkName, DateOfCreation, LinkComposerKey,NameAndSurnameComposer) " +

"Values('" + CompKey + "', '" + NameOfWork + "', '" + CreationDate + "', '" + LinkCompKey + "','" +

CompNameSurname + "')";

SqlConnection cnn;

var conStr = MainWindow.formations;

cnn = new SqlConnection(conStr);

cnn.Open();

var command = new SqlCommand(query, cnn);

command.ExecuteNonQuery();

Console.WriteLine("Data inserted!\n Closing connection ");

cnn.Close();

Console.WriteLine("Connection has been closed , database ready for next operation!");

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

}

private void Button\_Click\_2(object sender, RoutedEventArgs e)

{

SqlConnection cnn;

var conStr = MainWindow.formations;

cnn = new SqlConnection(conStr);

cnn.Open();

Console.Write("OPENING DB CONNECTION!!! \n");

var select = "DELETE FROM ComposerWorksList WHERE ComposerWorkKey=@a2";

var commandBuilder = new SqlCommand(select, cnn);

commandBuilder.Parameters.AddWithValue("a2", Edit1.Text);

try

{

commandBuilder.ExecuteNonQuery();

Console.WriteLine("Delete operation successful!");

}

catch (SqlException ex)

{

Console.WriteLine(ex.ToString());

}

}

private void Button\_Click\_3(object sender, RoutedEventArgs e)

{

try

{

SqlConnection cnn;

var conStr = MainWindow.formations;

cnn = new SqlConnection(conStr);

cnn.Open();

Console.Write("OPENING DB CONNECTION!!! \n");

Console.Write("[MENU] Connect db clicked - test \n");

var select = "Select \* FROM ComposerWorksList WHERE NameAndSurnameComposer Like '%" + searchbox.Text + "%'";

var commandBuilder = new SqlCommand(select, cnn);

commandBuilder.ExecuteNonQuery();

var dataAdapter = new SqlDataAdapter(commandBuilder);

var ds = new DataTable("Avtobusx");

dataAdapter.Fill(ds);

datagrid1.ItemsSource = ds.DefaultView;

Console.WriteLine("Connection established and the datagrid filled!");

cnn.Close();

}

catch (Exception ex)

{

Console.Write("Error");

}

}

}

}

using ComposerAndTheirWorks.ViewModels.Windows;

using Wpf.Ui.Controls;

using System.Data.SqlClient;

namespace ComposerAndTheirWorks.Views.Windows

{

public partial class MainWindow

{

public MainWindowViewModel ViewModel { get; }

public static string formations =

"Data Source=localhost;" +

"Initial Catalog=KR2Poet;" +

"User id=sa;" +

"Password=ctrt55xx;";

public MainWindow(

MainWindowViewModel viewModel,

INavigationService navigationService,

IServiceProvider serviceProvider,

ISnackbarService snackbarService,

IContentDialogService contentDialogService

)

{

Wpf.Ui.Appearance.Watcher.Watch(this);

ViewModel = viewModel;

DataContext = this;

InitializeComponent();

navigationService.SetNavigationControl(NavigationView);

snackbarService.SetSnackbarPresenter(SnackbarPresenter);

contentDialogService.SetContentPresenter(RootContentDialog);

NavigationView.SetServiceProvider(serviceProvider);

}

}

}

using System.Collections.ObjectModel;

using Wpf.Ui.Common;

using Wpf.Ui.Controls;

namespace ComposerAndTheirWorks.ViewModels.Windows

{

public partial class MainWindowViewModel : ObservableObject

{

[ObservableProperty]

private string \_applicationTitle = "Композиторы и их произведения";

[ObservableProperty]

private ObservableCollection<object> \_menuItems = new()

{

new NavigationViewItem()

{

Content = "Работы",

Icon = new SymbolIcon { Symbol = SymbolRegular.MusicNote124 },

TargetPageType = typeof(Views.Pages.DashboardPage)

},

new NavigationViewItem()

{

Content = "Композиторы",

Icon = new SymbolIcon { Symbol = SymbolRegular.Person24 },

TargetPageType = typeof(Views.Pages.DataPage)

}

};

[ObservableProperty]

private ObservableCollection<object> \_footerMenuItems = new()

{

new NavigationViewItem()

{

Content = "Настройки",

Icon = new SymbolIcon { Symbol = SymbolRegular.Settings24 },

TargetPageType = typeof(Views.Pages.SettingsPage)

}

};

[ObservableProperty]

private ObservableCollection<MenuItem> \_trayMenuItems = new()

{

new MenuItem { Header = "Home", Tag = "tray\_home" }

};

}

}