**Подключение и работа с базой данных в C#.**

Здесь будет рассмотрен пример создания проекта WPF и подключения к Базе данных.

WPF (Windows Presentation Foundation) - часть экосистемы платформы .NET и представляет собой подсистему для построения графических интерфейсов. Ключевая особенность рендеринга графики в WPF: используя WPF, значительная часть работы по отрисовке графики, как простейших кнопочек, так и сложных 3D-моделей, ложиться на графический процессор на видеокарте, что также позволяет воспользоваться аппаратным ускорением графики.

Преимущества WPF

Что вам, как разработчику, предлагает WPF?

* Использование традиционных языков .NET-платформы - C# и VB.NET для создания логики приложения.
* Возможность декларативного определения графического интерфейса с помощью специального языка разметки XAML, основанном на xml и представляющем альтернативу программному созданию графики и элементов управления, а также возможность комбинировать XAML и C#/VB.NET.
* Независимость от разрешения экрана: поскольку в WPF все элементы измеряются в независимых от устройства единицах, приложения на WPF легко масштабируются под разные экраны с разным разрешением.
* Новые возможности, которых сложно было достичь в WinForms, например, создание трехмерных моделей, привязка данных, использование таких элементов, как стили, шаблоны, темы и др.
* Хорошее взаимодействие с WinForms, благодаря чему, например, в приложениях WPF можно использовать традиционные элементы управления из WinForms.
* Богатые возможности по созданию различных приложений: это и мультимедиа, и двухмерная и трехмерная графика, и богатый набор встроенных элементов управления, а также возможность самим создавать новые элементы, создание анимаций, привязка данных, стили, шаблоны, темы и многое другое.
* Аппаратное ускорение графики - вне зависимости от того, работаете ли вы с 2D или 3D, графикой или текстом, все компоненты приложения транслируются в объекты, понятные Direct3D, и затем визуализируются с помощью процессора на видеокарте, что повышает производительность, делает графику более плавной.
* Создание приложений под множество ОС семейства Windows - от Windows XP до Windows 10.

Первое, что требуется сделать – создать новый проект. В окне создания проекта выберем WPF App (.NET Core) как показано на рисунке 1.

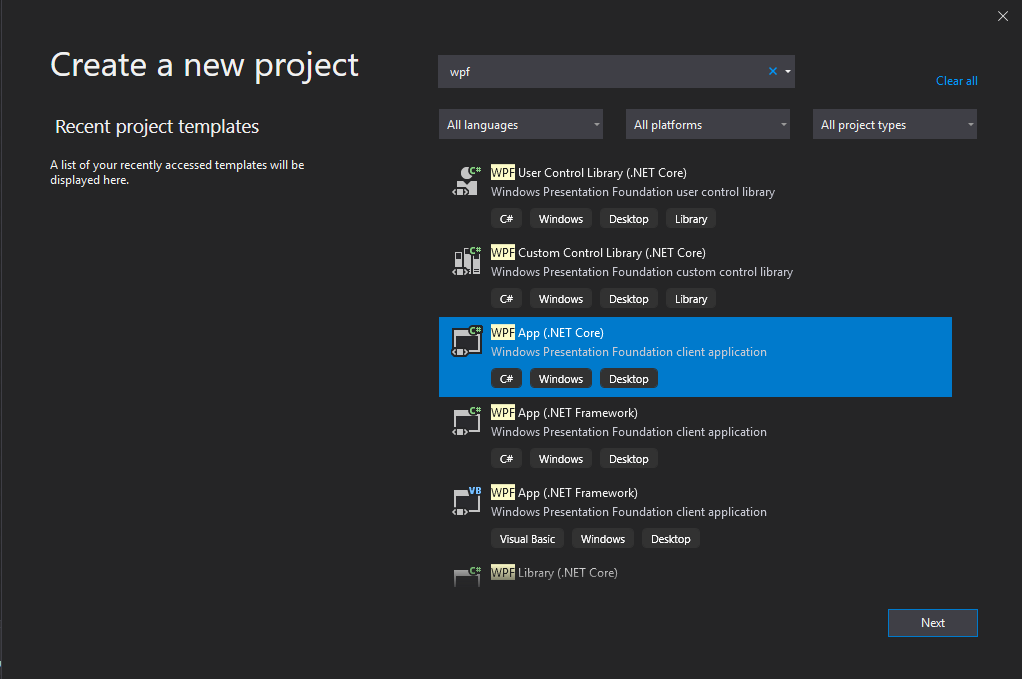


Рисунок 1 – Создание WPF приложения.

Далее необходимо дать имя проекту и выбрать путь (рис. 2).

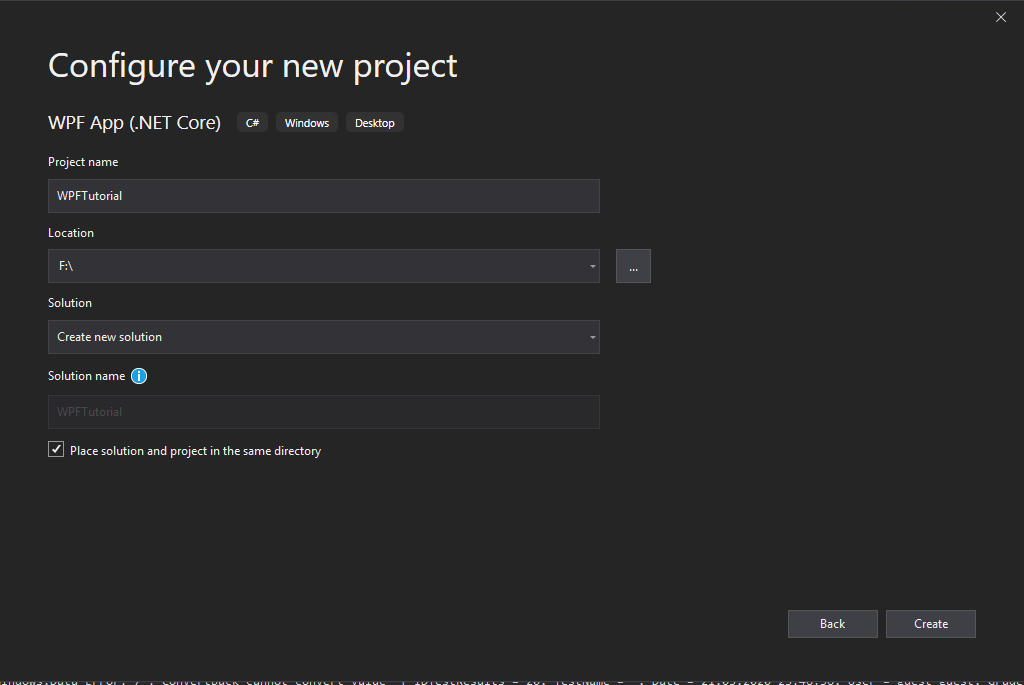


Рисунок 2 – Конфигурация проекта.

После нажатия на кнопку «Создать», вы увидите MainWindow.xaml – пользовательский интерфейс, к которому вернемся чуть позже.

В данном примере мы будем использовать Entity FrameWork и подход Code-First – это означает, что мы будем просто описывать сущности, а EF создаст саму базу данных.

Для этого добавим папку Model в проект, чтобы бы туда создавать классы, описывающие сущности для базы данных (рис. 3).

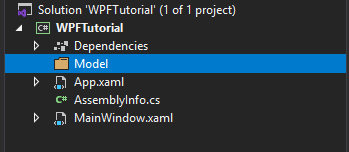


Рисунок 3 – Структура проекта.

Пусть наше приложение будет посвящено работе с книгами. Также будем считать, что у книги может быть только один автор. Добавим в проект новый класс Book (Правой кнопкой мыши по папке –> Добавить -> Класс) (рис. 4). По аналогии добавим класс Author.

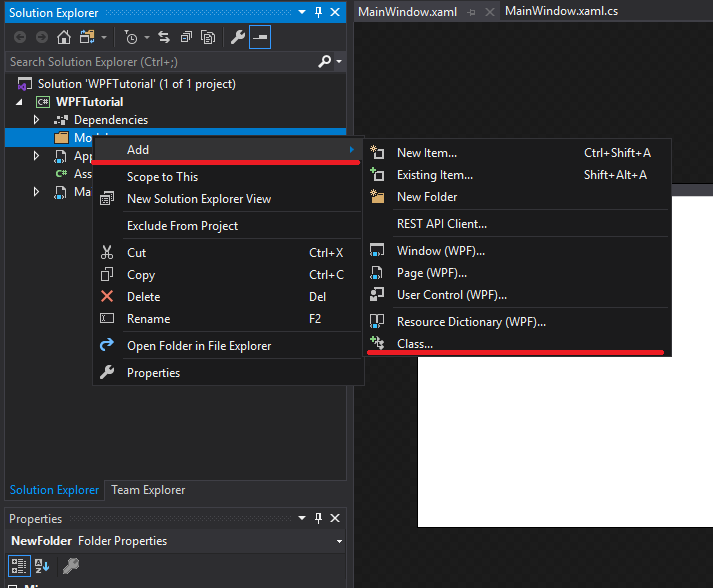


Рисунок 4 – Добавление классов.

Сначала опишем Author.cs:

1. Добавим модификатор public для класса
2. Пусть у автора будут следующие атрибуты: ИД, Фамилия и Имя
3. Соответственно, эти атрибуты необходимо добавить в класс

Вставьте следующий код в Author.cs:

public class Author

{

//ИД автора

public int IDAuthor { get; set; }

//Фамилия автора

public string LastName { get; set; }

//Имя автора

public string FirstName { get; set; }

}

Но теперь нужно указать, что IDAuthor является первичным ключом и установим автоинкремент. Для этого в пространство имён добавим:

**using** System**.**ComponentModel**.**DataAnnotations**;**

**using** System**.**ComponentModel**.**DataAnnotations**.**Schema**;**

И изменим класс следующим образом:

**public** class Author

**{**

//ИД автора

//Устанавливает Первичный ключ

**[**Key**]**

//Устанавливает автоинкремент

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**Identity**)]**

**public** int IDAuthor **{** **get;** **set;** **}**

//Фамилия автора

**public** string LastName **{** **get;** **set;** **}**

//Имя автора

**public** string FirstName **{** **get;** **set;** **}**

**}**

В Book.cs будут следующие атрибуты: ИД, название, жанр, автор, год выпуска.

**public** class Book

**{**

//ИД книги

//Устанавливает Первичный ключ

**[**Key**]**

//Устанавливает автоинкремент

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**Identity**)]**

**public** int IDBook **{** **get;** **set;** **}**

//Название книги

**public** string Name **{** **get;** **set;** **}**

//Жанр

**public** string Genre **{** **get;** **set;** **}**

//Год выпуска

**public** int Year **{** **get;** **set;** **}**

**}**

Осталось лишь отобразить связь между Автором и Книгой. Связь между ними будет один ко многим, где один – Автор, многие – Книга.

Нужно модифицировать наши классы следующим образом:

Author.cs целиком после всех операций

**using** System**.**Collections**.**Generic**;**

**using** System**.**ComponentModel**.**DataAnnotations**;**

**using** System**.**ComponentModel**.**DataAnnotations**.**Schema**;**

**namespace** WPFTutorial**.**Model

**{**

**public** class Author

**{**

//ИД автора

//Устанавливает Первичный ключ

**[**Key**]**

//Устанавливает автоинкремент

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**Identity**)]**

**public** int IDAuthor **{** **get;** **set;** **}**

//Фамилия автора

**public** string LastName **{** **get;** **set;** **}**

//Имя автора

**public** string FirstName **{** **get;** **set;** **}**

//Автор - один в связи, т.е. будет содержать коллекцию многих (Книги)

**public** ICollection**<**Book**>** Books **{** **get;** **set;** **}**

**}**

**}**

Book.cs целиком

**using** System**.**ComponentModel**.**DataAnnotations**;**

**using** System**.**ComponentModel**.**DataAnnotations**.**Schema**;**

**namespace** WPFTutorial**.**Model

**{**

**public** class Book

**{**

//ИД книги

//Устанавливает Первичный ключ

**[**Key**]**

//Устанавливает автоинкремент

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**Identity**)]**

**public** int IDBook **{** **get;** **set;** **}**

//Название книги

**public** string Name **{** **get;** **set;** **}**

//Жанр

**public** string Genre **{** **get;** **set;** **}**

//Год выпуска

**public** string Year **{** **get;** **set;** **}**

//Содержит ИД автора

**public** int IDAuthor **{** **get;** **set;** **}**

**public** Author Author **{** **get;** **set;** **}**

**}**

**}**

В качестве СУБД будем использовать SQLite - одна из наиболее используемых систем управления базами данных. Главным преимуществом SQLite является то, что для базы данных не нужно сервера. База данных представляет собой обычный локальный файл, который мы можем перемещать вместе с главным файлом приложения.

Теперь необходимо добавить функциональность SQLite в проект. Для этого нужно нажать правой кнопкой на узел Dependencies и выбрать в открывшемся контекстном меню пункт Manage NuGet Packages... В открывшемся окне выберем первую вкладку слева – Browse и введём sqlite в поиск. Нам нужно выбрать Microsoft.Data.Sqlite.Core.

Устанавливаем пакет. В примере будет использоваться версия 3.1.2 (рис. 5). Также **необходимо** установить пакет Microsoft.EntityFrameworkCore.Sqlite.

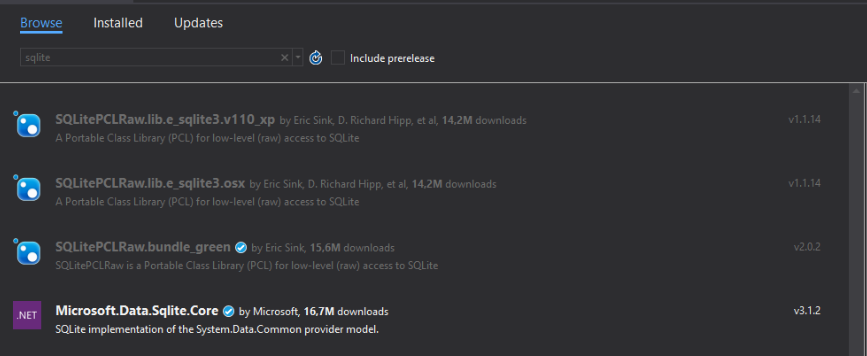


Рисунок 5 – Установка NuGet пакета.

Следующим шагом будет создание контекста. Для этого добавим в проект новый класс и назовём BookDBContext. Добавим модификатор public и этот класс должен наследовать класс DbContext, входящий в пространство имён -

**using** Microsoft**.**EntityFrameworkCore**;**

Теперь в теле класса BookDBContext необходимо объявить классы, представляющие сущности Книг и Авторов, попутно добавив в пространство имён ещё и папку Model:

**using** WPFTutorial**.**Model**;**

**using** Microsoft**.**EntityFrameworkCore**;**

**namespace** WPFTutorial

**{**

**public** class BookDBContext **:** DbContext

**{**

**public** DbSet**<**Author**>** Authors **{** **get;** **set;** **}**

**public** DbSet**<**Book**>** Books **{** **get;** **set;** **}**

**}**

**}**

Следующий шаг – добавление строки подключения:

**protected** **override** void OnConfiguring**(**DbContextOptionsBuilder optionsBuilder**)**

**{**

optionsBuilder**.**UseSqlite**(**"Filename=BookDB.db"**);**

**}**

Где Filename – название будущей базы данных.

Добавим конструктор класса BookDBContext и следующую команду в него:

**public** BookDBContext **()**

**{**

//Проверяет существует ли БД, если да - ничего не делает, иначе создаёт схему

Database**.**EnsureCreated**();**

**}**

Определим метод OnModelCreating, где следую из названия можно понять его суть – действия с сущностями при создании БД. Здесь мы можем явно определить связи и ключи между сущностями, их поведение при удалении главной записи, а также задать начальные значения для базы данных.

**protected** **override** void OnModelCreating**(**ModelBuilder modelBuilder**)**

**{**

/\*в данном случае поведение при удалении - каскадное,

т.е. если удалить запись об авторе, то все книги, связаные с ним, тоже удалятся\*/

modelBuilder**.**Entity**<**Author**>()**

**.**HasMany**(**b **=>** b**.**Books**).**WithOne**(**a **=>** a**.**Author**).**HasForeignKey**(**i **=>** i**.**IDAuthor**).**OnDelete**(**DeleteBehavior**.**Cascade**);**

modelBuilder**.**Entity**<**Book**>()**

**.**HasOne**(**a **=>** a**.**Author**).**WithMany**(**b **=>** b**.**Books**).**HasForeignKey**(**i **=>** i**.**IDAuthor**).**OnDelete**(**DeleteBehavior**.**Cascade**);**

//добавим некоторые значения, чтобы при создании база не была пустой

modelBuilder**.**Entity**<**Author**>().**HasData**(**

**new** Author**[]**

**{**

**new** Author**{**IDAuthor**=**1**,** FirstName**=**"Дэвид"**,** LastName**=**"Уоллес"**},**

**new** Author**{**IDAuthor**=**2**,** FirstName**=**"Говард"**,** LastName**=**"Лавкрафт"**}**

**});**

modelBuilder**.**Entity**<**Book**>().**HasData**(**

**new** Book**[]**

**{**

**new** Book**{**IDBook**=**1**,** IDAuthor**=**1**,** Name**=**"Бесконечная шутка"**,** Genre**=**"Роман"**,** Year**=**"1996г."**},**

**new** Book**{**IDBook**=**2**,** IDAuthor**=**2**,** Name**=**"Хребты безумия"**,** Genre**=**"Рассказ"**,** Year**=**"1931г."**},**

**new** Book**{**IDBook**=**3**,** IDAuthor**=**2**,** Name**=**"Цвет из иных миров"**,** Genre**=**"Рассказ"**,** Year**=**"1927г."**}**

**});**

**base.**OnModelCreating**(**modelBuilder**);**

**}**

Окончательный вид BookDBContext:

**using** Microsoft**.**EntityFrameworkCore**;**

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

**using** WPFTutorial**.**Model**;**

**namespace** WPFTutorial

**{**

**public** class BookDBContext **:** DbContext

**{**

**public** DbSet**<**Author**>** Authors **{** **get;** **set;** **}**

**public** DbSet**<**Book**>** Books **{** **get;** **set;** **}**

**protected** **override** void OnConfiguring**(**DbContextOptionsBuilder optionsBuilder**)**

**{**

optionsBuilder**.**UseSqlite**(**"Filename=BookDB.db"**);**

**}**

**public** BookDBContext **()**

**{**

//Проверяет существует ли БД, если да - ничего не делает, иначе создаёт схему

Database**.**EnsureCreated**();**

**}**

**protected** **override** void OnModelCreating**(**ModelBuilder modelBuilder**)**

**{**

/\*в данном случае поведение при удалении - каскадное,

т.е. если удалить запись об авторе, то все книги, связаные с ним, тоже удалятся\*/

modelBuilder**.**Entity**<**Author**>()**

**.**HasMany**(**b **=>** b**.**Books**).**WithOne**(**a **=>** a**.**Author**).**HasForeignKey**(**i **=>** i**.**IDAuthor**).**OnDelete**(**DeleteBehavior**.**Cascade**);**

modelBuilder**.**Entity**<**Book**>()**

**.**HasOne**(**a **=>** a**.**Author**).**WithMany**(**b **=>** b**.**Books**).**HasForeignKey**(**i **=>** i**.**IDAuthor**).**OnDelete**(**DeleteBehavior**.**Cascade**);**

//добавим некоторые значения, чтобы при создании база не была пустой

modelBuilder**.**Entity**<**Author**>().**HasData**(**

**new** Author**[]**

**{**

**new** Author**{**IDAuthor**=**1**,** FirstName**=**"Дэвид"**,** LastName**=**"Уоллес"**},**

**new** Author**{**IDAuthor**=**2**,** FirstName**=**"Говард"**,** LastName**=**"Лавкрафт"**}**

**});**

modelBuilder**.**Entity**<**Book**>().**HasData**(**

**new** Book**[]**

**{**

**new** Book**{**IDBook**=**1**,** IDAuthor**=**1**,** Name**=**"Бесконечная шутка"**,** Genre**=**"Роман"**,** Year**=**"1996г."**},**

**new** Book**{**IDBook**=**2**,** IDAuthor**=**2**,** Name**=**"Хребты безумия"**,** Genre**=**"Рассказ"**,** Year**=**"1931г."**},**

**new** Book**{**IDBook**=**3**,** IDAuthor**=**2**,** Name**=**"Цвет из иных миров"**,** Genre**=**"Рассказ"**,** Year**=**"1927г."**}**

**});**

**base.**OnModelCreating**(**modelBuilder**);**

**}**

**}**

**}**

Далее, чтобы создать БД, необходимо объявить переменную класса контекста и обратиться к её конструктору, для этого нужно раскрыть список MainWindow.xaml и открыть MainWindow.xaml.cs – он же Code-Behind (рис. 6).



Рисунок 6 – Структура MainWindow.

Объявим новую переменную класса BookDBContext и вызовем сразу конструктор:

BookDBContext db **=** **new** BookDBContext**();**

**public** MainWindow**()**

**{**

InitializeComponent**();**

**}**

Вот и всё! Можем запускать проект, и, как только он скомпилируется, мы увидим белое окно MainWindow.xaml, но перейдя по пути, где лежит проект /bin/Debug/netcoreapp3.1 мы увидим файл BookDB.db. Давайте откроем в SQLiteStudio – визуальной среды для создания БД в СУБД SQLite. (<https://sqlitestudio.pl/index.rvt?act=download>).

Откроем SQLiteStudio выберем Database -> Add a database, где указываем путь к созданной БД (рис. 7).

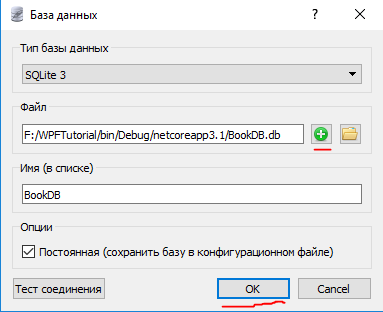


Рисунок 7 – SQLiteStudio добавление БД.

Слева можно увидеть нашу БД, и чтобы убедиться, что создание прошло успешно, раскроем структуру таблиц. Как видно на рисунке 8 – все описанные атрибуты стали полями таблиц.

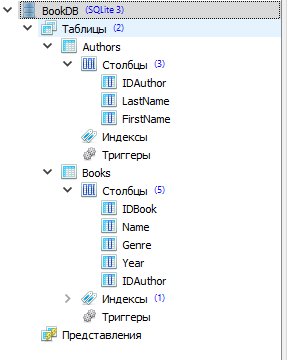


Рисунок 8 – Структура БД.

Теперь выполним запрос SELECT к таблицам, чтобы посмотреть на их содержание (рис. 9 и 10).

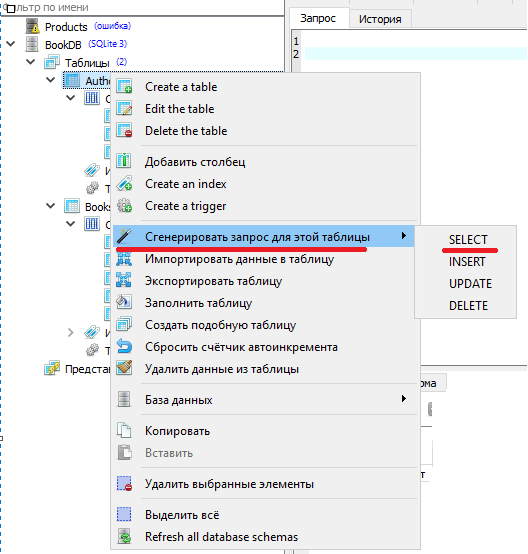


Рисунок 9 – Генерация запроса SELECT.

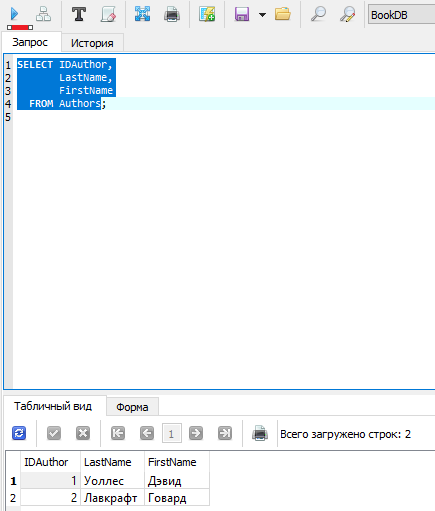


Рисунок 10 – Результат запроса SELECT.

Аналогично можете проделать и со второй таблицей.

Как видно из скриншотов – база данных создана, а таблицы заполнены начальными данными.

Теперь можно переходить к проектированию пользовательского интерфейса. Для написания интерфейса используется язык разметки xaml. Для более удобной работы рекомендую установить расширение XAML Styler (Extensions -> Manage Extensions вкладка Online и в поиске XAML Styler).

Замените код на следующий:

<Window

x:Class=**"WPFTutorial.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:local=**"clr-namespace:WPFTutorial"**

xmlns:mc=**"http://schemas.openxmlformats.org/markup-compatibility/2006"**

Title=**"Книги"**

Width=**"800"**

Height=**"600"**

WindowStartupLocation=**"CenterScreen"**

mc:Ignorable=**"d"**>

<!-- Цвет выберите на свой вкус -->

<Grid Background=**"Beige"**>

<!-- Определяет количество колонок и их атрибуты -->

<Grid.ColumnDefinitions>

<ColumnDefinition Width=**"2.5\*"** />

<ColumnDefinition Width=**"\*"** />

</Grid.ColumnDefinitions>

<!-- Определяет количество строк и их атрибуты -->

<Grid.RowDefinitions>

<RowDefinition Height=**"3\*"** />

<RowDefinition Height=**"\*"** />

</Grid.RowDefinitions>

</Grid>

</Window>

Здесь задается название, высота и ширина окна, а также задаётся Grid – наиболее часто используемый контейнер, представляющий обычную таблицу.

Для отображения данных из таблиц используется элемент DataGrid. Добавим его в тело Grid.

<DataGrid

x:Name=**"dataGridBook"**

Grid.RowSpan="2"

Grid.Column=**"0"**

Margin=**"5"**

Background=**"LightGray"** />

Grid.RowSpan – указывает количество строк для объединения в Grid, Grid.Column – номер колоны для расположения элемента. x:Name необходимо чтобы обращаться к элементу в Code-Behind (MainWindow.xaml.cs).

Для того чтобы заполнить данными DataGrid, откроемм MainWindow.xaml.cs и в конструктор MainWindow добавим следующий код:

dataGridBook**.**ItemsSource **=** db**.**Books**.**ToList**();**

Здесь мы свойству ItemSource объекта DataGrid присваиваем данные из таблицы Books и сразу конвертируем в List, т.к. только данные типа коллекции можно присваивать для ItemSource.

Запустим приложение. Как видно на рисунке 11, данные отображаются, но для удобства пользователя нужно преобразить внешний вид. Сейчас мы поменяем имена столбцов, уберём не нужные, и вместо ИД Авторов выведем их Имя и Фамилию.

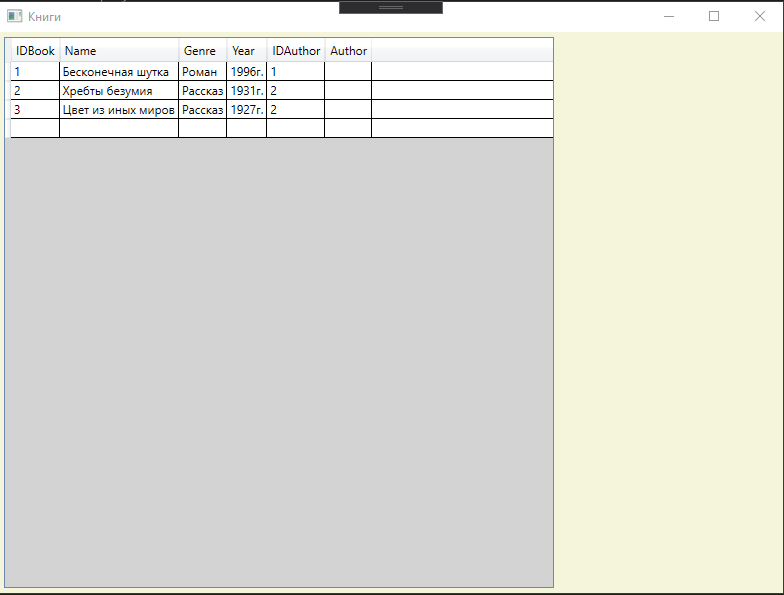


Рисунок 11 – Приложение «Книги».

Для начала изменим код DataGrid на следующий:

<DataGrid

x:Name="dataGridBook"

Grid.RowSpan="2"

Grid.Column="0"

Margin="5"

Background="LightGray"

AutoGenerateColumns="False"

IsReadOnly="True"

CanUserDeleteRows="True"

CanUserAddRows="False"

FontSize="16">

<DataGrid.Columns>

<DataGridTextColumn Header="Название книги" Width="\*" Binding="{Binding Name}">

<DataGridTextColumn.ElementStyle>

<Style TargetType="TextBlock">

<Setter Property="TextWrapping" Value="Wrap" />

</Style>

</DataGridTextColumn.ElementStyle>

</DataGridTextColumn>

<DataGridTextColumn Header="Жанр" Width="0.6\*" Binding="{Binding Genre}">

<DataGridTextColumn.ElementStyle>

<Style TargetType="TextBlock">

<Setter Property="TextWrapping" Value="Wrap" />

</Style>

</DataGridTextColumn.ElementStyle>

</DataGridTextColumn>

<DataGridTextColumn Header="Год выхода" Width="0.6\*" Binding="{Binding Year}">

<DataGridTextColumn.ElementStyle>

<Style TargetType="TextBlock">

<Setter Property="TextWrapping" Value="Wrap" />

</Style>

</DataGridTextColumn.ElementStyle>

</DataGridTextColumn>

<DataGridTextColumn Header="Автор" Width="\*" Binding="{Binding Author}">

<DataGridTextColumn.ElementStyle>

<Style TargetType="TextBlock">

<Setter Property="TextWrapping" Value="Wrap" />

</Style>

</DataGridTextColumn.ElementStyle>

</DataGridTextColumn>

</DataGrid.Columns>

</DataGrid>

Здесь мы отключаем свойство автогенерации столбцов, затем сами определяем их количество, заголовок и ширину, а также путь для привязки. В параметрах стиля задано свойство для переноса текста.

Теперь измени код файла MainWindow.xaml.cs на предложенный ниже:

**using** System**.**Linq**;**

**using** System**.**Windows**;**

**namespace** WPFTutorial

**{**

**public** **partial** class MainWindow **:** Window

**{**

BookDBContext db **=** **new** BookDBContext**();**

**public** MainWindow**()**

**{**

InitializeComponent**();**

/\*Для удобства создадим метод, который ничего не возвращает и

не требует параметров для передачи\*/

SetDataGridValue**();**

**}**

**private** void SetDataGridValue**()**

**{**

var books **=** **(from** b **in** db**.**Books

**join** a **in** db**.**Authors on b**.**IDAuthor equals a**.**IDAuthor

**select** **new**

**{**

IDBook**=**b**.**IDBook**,**

Name **=** b**.**Name**,**

Genre **=** b**.**Genre**,**

Year **=** b**.**Year**,**

Author **=** $"{a.FirstName} {a.LastName}"

**}).**ToList**();**

dataGridBook**.**ItemsSource **=** books**;**

**}**

**}**

**}**

В методе SetDataGridValue мы объявляем переменную books ключевым словом var, т.к. мы не знаем её конечный после выполнения запроса. Затем в данную переменную мы помещаем результат запроса, написанный на языке запросов LINQ, который входит в пространство имён System.Linq. Его синтаксис схож с SQL. Переменную books конвертируем в List и присваиваем свойству ItemSource.

Теперь при запуске приложения получим следующий результат (рис. 12).

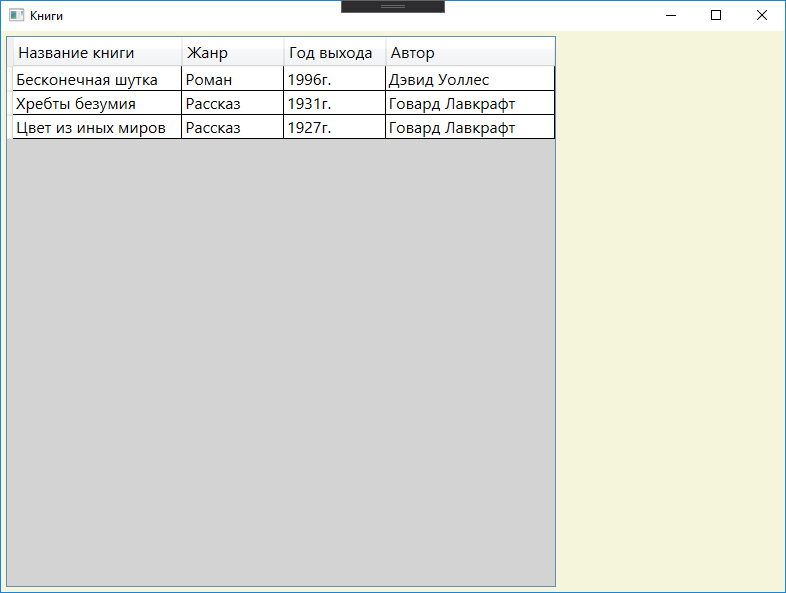


Рисунок 12 – Приложение «Книги».

Сейчас внесём возможность редактирования, удаления и добавления новых записей о книгах, поиск по автору и названию книги.

Начнём с поиска. Дополним тело Grid после тега </DataGrid> следующим кодом:

<StackPanel Grid.Row=**"0"** Grid.Column=**"1"**>

<Label Content=**"Поиск по названию"** FontSize=**"16"**/>

<TextBox x:Name=**"textBoxSearchName"** Height=**"35"** Margin=**"5"** TextChanged=**"textBoxSearchName\_TextChanged"**/>

<Label Content=**"Поиск по автору"** FontSize=**"16"**/>

<TextBox x:Name=**"textBoxSearchAuthor"** Height=**"35"** Margin=**"5"** TextChanged=**"textBoxSearchAuthor\_TextChanged"**/>

</StackPanel>

StackPanel – контейнер, Label – метка, а элемент TextBox – поле, куда пользователь сможет вводить текст, в свою очередь событие TextChanged будет на это реагировать. Напишем следующий код в события:

private void textBoxSearchName\_TextChanged(object sender, System.Windows.Controls.TextChangedEventArgs e)

{

var searchName = (from b in db.Books

join a in db.Authors on b.IDAuthor equals a.IDAuthor

select new

{

IDBook = b.IDBook,

Name = b.Name,

Genre = b.Genre,

Year = b.Year,

Author = $"{a.FirstName} {a.LastName}"

}).ToList();

if (textBoxSearchAuthor.Text==null)

{

dataGridBook.ItemsSource = searchName.Where(n => n.Name.ToUpper().Contains(textBoxSearchName.Text.ToUpper())).ToList();

}

else

{

var res = searchName.Where(n => n.Name.ToUpper().Contains(textBoxSearchName.Text.ToUpper()));

dataGridBook.ItemsSource = res.Where(n=>n.Author.ToUpper().Contains(textBoxSearchAuthor.Text.ToUpper())).ToList();

}

}

private void textBoxSearchAuthor\_TextChanged(object sender, System.Windows.Controls.TextChangedEventArgs e)

{

var searchAuthor = (from b in db.Books

join a in db.Authors on b.IDAuthor equals a.IDAuthor

select new

{

IDBook = b.IDBook,

Name = b.Name,

Genre = b.Genre,

Year = b.Year,

Author = $"{a.FirstName} {a.LastName}"

}).ToList();

if (textBoxSearchName.Text == null)

{

dataGridBook.ItemsSource = searchAuthor.Where(n => n.Author.ToUpper().Contains(textBoxSearchAuthor.Text.ToUpper())).ToList(); ;

}

else

{

var res = searchAuthor.Where(n => n.Author.ToUpper().Contains(textBoxSearchAuthor.Text.ToUpper()));

dataGridBook.ItemsSource = res.Where(n => n.Name.ToUpper().Contains(textBoxSearchName.Text.ToUpper()));

}

}

Теперь создадим новое окно, которое будет служить для редактирования имеющихся и добавления новых записей. Правой кнопкой по проекту слева -> Добавить -> Окно(WPF). Пусть будет называться AddEditWin. Сразу создадим ещё одно окна – список авторов, которые есть в базе данных. Назовём его AuthorList.

В MainWindow добавим четыре кнопки: Добавить книгу, Редактировать книгу, Удалить книгу, Список авторов. Теперь код StackPanel выглядит следующим образом:

<StackPanel Grid.Row=**"0"** Grid.Column=**"1"**>

<Label Content=**"Поиск по названию"** FontSize=**"16"** />

<TextBox

x:Name=**"textBoxSearchName"**

Height=**"35"**

Margin=**"5"**

TextChanged=**"textBoxSearchName\_TextChanged"** />

<Label Content=**"Поиск по автору"** FontSize=**"16"** />

<TextBox

x:Name=**"textBoxSearchAuthor"**

Height=**"35"**

Margin=**"5"**

TextChanged=**"textBoxSearchAuthor\_TextChanged"** />

<Button

x:Name=**"buttonAddBook"**

Width=**"Auto"**

Height=**"35"**

Margin=**"5,50,5,5"**

Click=**"buttonAddBook\_Click"**

Content=**"Добавить книгу"**

FontSize=**"16"** />

<Button

x:Name=**"buttonEditBook"**

Width=**"Auto"**

Height=**"35"**

Margin=**"5"**

Click=**"buttonEditBook\_Click"**

Content=**"Редактировать книгу"**

FontSize=**"16"** />

<Button

x:Name=**"buttonDeleteBook"**

Width=**"Auto"**

Height=**"35"**

Margin=**"5"**

Click=**"buttonDeleteBook\_Click"**

Content=**"Удалить книгу"**

FontSize=**"16"** />

<Button

x:Name=**"buttonAuthorList"**

Width=**"Auto"**

Height=**"35"**

Margin=**"5,50,5,5"**

Click=**"buttonAuthorList\_Click"**

Content=**"Список авторов"**

FontSize=**"16"** />

</StackPanel>

Событие для кнопки «Список авторов» нужно заменить на следующие строки:

**private** void buttonAuthorList\_Click**(object** sender**,** RoutedEventArgs e**)**

**{**

//AuthorList - название окна, которые мы создали и хотим открыть

AuthorList authorList **=** **new** AuthorList**();**

authorList**.**Show**();**

**}**

Для кнопки «Удалить книгу» событие будет иметь следующий вид:

**private** void buttonDeleteBook\_Click**(object** sender**,** RoutedEventArgs e**)**

**{**

/\*Опять выполним запрос, конечно правильней вынести его в отдельный метод и

возвращать объект\*/

var books **=** **(from** b **in** db**.**Books

**join** a **in** db**.**Authors on b**.**IDAuthor equals a**.**IDAuthor

**select** **new**

**{**

IDBook **=** b**.**IDBook**,**

Name **=** b**.**Name**,**

Genre **=** b**.**Genre**,**

Year **=** b**.**Year**,**

Author **=** $"{a.FirstName} {a.LastName}"

**}).**ToList**();**

//Проверяем, пустой ли объект

**if** **(**dataGridBook**.**SelectedItem **!=** **null)**

**{**

//Находим объект, который равен выбранному

var IDb **=** books**.**Where**(**a **=>** a**.**Equals**(**dataGridBook**.**SelectedItem**)).**ToList**();**

//Получаем его ИД

int Id **=** IDb**[**0**].**IDBook**;**

//Находим объект в БД по его ИД и присваиваем его переменной класса Book

Book book **=** db**.**Books**.**Find**(**Id**);**

//Удаляем объект из таблицы

db**.**Books**.**Remove**(**book**);**

//Сохраняем изменения

db**.**SaveChanges**();**

//Обноваляем таблицу

SetDataGridValue**();**

MessageBox**.**Show**(**"Операция успешно выполнена!"**,** ""**,** MessageBoxButton**.**OK**,** MessageBoxImage**.**Information**);**

**}**

//Выводим диалоговое окно с предупреждением

**else** MessageBox**.**Show**(**"Вы не выбрали запись для удаления!"**,** ""**,** MessageBoxButton**.**OK**,** MessageBoxImage**.**Warning**);**

**}**

Кнопка «Редактировать книгу» имеет следующий код у события:

**private** void buttonEditBook\_Click**(object** sender**,** RoutedEventArgs e**)**

**{**

var books **=** **(from** b **in** db**.**Books

**join** a **in** db**.**Authors on b**.**IDAuthor equals a**.**IDAuthor

**select** **new**

**{**

IDBook **=** b**.**IDBook**,**

Name **=** b**.**Name**,**

Genre **=** b**.**Genre**,**

Year **=** b**.**Year**,**

Author **=** $"{a.FirstName} {a.LastName}"

**}).**ToList**();**

var IDb **=** books**.**Where**(**a **=>** a**.**Equals**(**dataGridBook**.**SelectedItem**)).**ToList**();**

int Id **=** IDb**[**0**].**IDBook**;**

//Передаём ИД выбранной записи

AddEditWin addEditWin **=** **new** AddEditWin**(**Id**);**

addEditWin**.**Show**();**

**}**

Повторюсь, код запроса можно вынести в отдельный метод.

Наконец, код события у кнопки «Добавить книгу»:

**private** void buttonAddBook\_Click**(object** sender**,** RoutedEventArgs e**)**

**{**

//Так как мы будем добавлять новую запись, то и передавать нам ничего не надо

AddEditWin addEditWin **=** **new** AddEditWin**(null);**

addEditWin**.**Show**();**

**}**

Перейдём к AddEditWin.xaml. Код разметки измените на приведенный ниже:

<Window

x:Class=**"WPFTutorial.AddEditWin"**

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:local=**"clr-namespace:WPFTutorial"**

xmlns:mc=**"http://schemas.openxmlformats.org/markup-compatibility/2006"**

Title=**"Добавление и редактирование книг"**

MinWidth=**"500"**

MinHeight=**"450"**

MaxWidth=**"500"**

MaxHeight=**"450"**

WindowStartupLocation=**"CenterScreen"**

mc:Ignorable=**"d"**>

<Grid Background=**"Beige"**>

<Grid.RowDefinitions>

<RowDefinition Height=**"\*"** />

<RowDefinition Height=**"0.2\*"** />

</Grid.RowDefinitions>

<StackPanel Grid.Row=**"0"** Grid.Column=**"0"**>

<Label Content=**"Название"** FontSize=**"16"** />

<TextBox

x:Name=**"textBoxName"**

Width=**"Auto"**

Height=**"35"**

Margin=**"5"**

FontSize=**"16"** />

<Label Content=**"Жанр"** FontSize=**"16"** />

<TextBox

x:Name=**"textBoxGenre"**

Height=**"35"**

Margin=**"5"**

FontSize=**"16"** />

<Label Content=**"Год выхода"** FontSize=**"16"** />

<TextBox

x:Name=**"textBoxYear"**

Height=**"35"**

Margin=**"5"**

FontSize=**"16"** />

<Label Content=**"Автор"** FontSize=**"16"** />

<ComboBox

x:Name=**"comboBoxAuthor"**

Height=**"35"**

Margin=**"5"**

DisplayMemberPath=**"Name"**

FontSize=**"16"**

SelectedValuePath=**"IDAuthor"** />

<Button

x:Name=**"buttonAuthor"**

Width=**"120"**

Height=**"35"**

Margin=**"5,0,5,0"**

Click=**"buttonAuthor\_Click"**

Content=**"Авторы"**

FontSize=**"16"** />

</StackPanel>

<StackPanel

Grid.Row=**"1"**

Grid.Column=**"0"**

Orientation=**"Horizontal"**>

<Button

x:Name=**"buttonAddUpdate"**

Width=**"120"**

Height=**"35"**

Margin=**"52,5,5,5"**

Click=**"buttonAddUpdate\_Click"**

Content=**"Добавить"**

FontSize=**"16"** />

<Button

x:Name=**"buttonClear"**

Width=**"120"**

Height=**"35"**

Margin=**"5"**

Click=**"buttonClear\_Click"**

Content=**"Очистить"**

FontSize=**"16"** />

<Button

x:Name=**"buttonClose"**

Width=**"120"**

Height=**"35"**

Margin=**"5"**

Click=**"buttonClose\_Click"**

Content=**"Закрыть"**

FontSize=**"16"** />

</StackPanel>

</Grid>

</Window>

AddEditeWin.xaml.cs в итоге выглядит следующим образом:

**using** System**;**

**using** System**.**Linq**;**

**using** System**.**Windows**;**

**using** WPFTutorial**.**Model**;**

**namespace** WPFTutorial

**{**

**public** **partial** class AddEditWin **:** Window

**{**

int IDb **=** 0**;**

BookDBContext db **=** **new** BookDBContext**();**

**public** AddEditWin**(object** id**)**

**{**

InitializeComponent**();**

//Медот для установки значений, если редактируем книгу

IDb **=** Convert**.**ToInt32**(**id**);**

**if** **(**IDb **!=** 0**)** SetBookData**(**Convert**.**ToInt32**(**IDb**));**

//Устанавливаем занчения для ComboBox

var authors **=** **(from** a **in** db**.**Authors

**select** **new**

**{**

IDAuthor **=** a**.**IDAuthor**,**

Name **=** $"{a.FirstName} {a.LastName}"

**}).**ToList**();**

comboBoxAuthor**.**ItemsSource **=** authors**;**

**}**

**private** void SetBookData**(**int id**)**

**{**

//Находим книгу по ИД

var book**=** db**.**Books**.**Where**(**i **=>** i**.**IDBook **==** id**).**ToList**();**

Book selectedBook **=** book**[**0**];**

//Устанавливаем занчения книги

textBoxName**.**Text **=** selectedBook**.**Name**;**

textBoxGenre**.**Text **=** selectedBook**.**Genre**;**

textBoxYear**.**Text **=** selectedBook**.**Year**;**

var authors **=** **(from** a **in** db**.**Authors

**select** **new**

**{**

IDAuthor **=** a**.**IDAuthor**,**

Name **=** $"{a.FirstName} {a.LastName}"

**}).**ToList**();**

var selectedAuthor **=** authors**.**Where**(**i **=>** i**.**IDAuthor **==** selectedBook**.**IDAuthor**).**ToList**();**

comboBoxAuthor**.**SelectedItem **=** selectedAuthor**[**0**];**

**}**

**private** void buttonAuthor\_Click**(object** sender**,** RoutedEventArgs e**)**

**{**

//Показывает окно со списком авторов

AuthorList authorList **=** **new** AuthorList**();**

authorList**.**Show**();**

**}**

**private** void buttonAddUpdate\_Click**(object** sender**,** RoutedEventArgs e**)**

**{**

var authors **=** **(from** a **in** db**.**Authors

**select** **new**

**{**

IDAuthor **=** a**.**IDAuthor**,**

Name **=** $"{a.FirstName} {a.LastName}"

**}).**ToList**();**

var selectedAuthor **=** authors**.**Where**(**i **=>** i**.**Equals**(**comboBoxAuthor**.**SelectedItem**)).**ToList**();**

//Создаем переменную класса Book

Book book **=** **new** Book **{** Name **=** textBoxName**.**Text**,** Genre **=** textBoxGenre**.**Text**,** Year**=**textBoxYear**.**Text**,** IDAuthor**=**selectedAuthor**[**0**].**IDAuthor**};**

//Проверяем, существует ли объект с таким ИД

**if** **(**db**.**Books**.**Find**(**IDb**)!=null)**

**{**

//Т.к. переменная есть, то просто обновляем её

db**.**Books**.**Update**(**book**);**

db**.**SaveChanges**();**

MessageBox**.**Show**(**"Операция успешно выполнена!"**,** ""**,** MessageBoxButton**.**OK**,** MessageBoxImage**.**Information**);**

**}**

**else**

**{**

db**.**Books**.**Add**(**book**);**

db**.**SaveChanges**();**

MessageBox**.**Show**(**"Операция успешно выполнена!"**,** ""**,** MessageBoxButton**.**OK**,** MessageBoxImage**.**Information**);**

**}**

**}**

**private** void buttonClear\_Click**(object** sender**,** RoutedEventArgs e**)**

**{**

//Очищаем значения

textBoxGenre**.**Text **=** **null;**

textBoxName**.**Text **=** **null;**

textBoxYear**.**Text **=** **null;**

comboBoxAuthor**.**SelectedIndex **=** **-**1**;**

**}**

**private** void buttonClose\_Click**(object** sender**,** RoutedEventArgs e**)**

**{**

//Закрывает это окно

**this.**Close**();**

**}**

**}**

**}**

Теперь в MainWindow.xaml для Window добавим событие Activated, чтобы оперативно обновлять таблицу.

Xaml код:

<Window

x:Class=**"WPFTutorial.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:local=**"clr-namespace:WPFTutorial"**

xmlns:mc=**"http://schemas.openxmlformats.org/markup-compatibility/2006"**

Title=**"Книги"**

Width=**"800"**

Height=**"600"**

Activated=**"Window\_Activated"**

WindowStartupLocation=**"CenterScreen"**

mc:Ignorable=**"d"**>

Код события Activated:

**private** void Window\_Activated**(object** sender**,** System**.**EventArgs e**)**

**{**

SetDataGridValue**();**

**}**

Теперь отредактируем AuthorList.xaml. Заменим код разметки на следующий:

<Window

x:Class=**"WPFTutorial.AuthorList"**

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:local=**"clr-namespace:WPFTutorial"**

xmlns:mc=**"http://schemas.openxmlformats.org/markup-compatibility/2006"**

Title=**"AuthorList"**

Width=**"500"**

Height=**"450"**

WindowStartupLocation=**"CenterScreen"**

mc:Ignorable=**"d"**>

<Grid Background=**"Beige"**>

<Grid.RowDefinitions>

<RowDefinition Height=**"\*"** />

<RowDefinition Height=**"0.15\*"** />

</Grid.RowDefinitions>

<DataGrid

x:Name=**"dataGridAuthors"**

Grid.Row=**"0"**

Grid.Column=**"0"**

Margin=**"5"**

AutoGenerateColumns=**"False"**

Background=**"LightGray"**

CanUserAddRows=**"True"**

CanUserDeleteRows=**"True"**

FontSize=**"16"**

IsReadOnly=**"False"**>

<DataGrid.Columns>

<DataGridTextColumn

Width=**"\*"**

Binding=**"{Binding FirstName}"**

Header=**"Имя"**>

<DataGridTextColumn.ElementStyle>

<Style TargetType=**"TextBlock"**>

<Setter Property=**"TextWrapping"** Value=**"Wrap"** />

</Style>

</DataGridTextColumn.ElementStyle>

</DataGridTextColumn>

<DataGridTextColumn

Width=**"\*"**

Binding=**"{Binding LastName}"**

Header=**"Фамилия"**>

<DataGridTextColumn.ElementStyle>

<Style TargetType=**"TextBlock"**>

<Setter Property=**"TextWrapping"** Value=**"Wrap"** />

</Style>

</DataGridTextColumn.ElementStyle>

</DataGridTextColumn>

</DataGrid.Columns>

</DataGrid>

<StackPanel

Grid.Row=**"1"**

Grid.Column=**"0"**

Orientation=**"Horizontal"**>

<Button

x:Name=**"buttonAddUpdate"**

Width=**"170"**

Height=**"35"**

Margin=**"75,0,5,0"**

Click=**"buttonAddUpdate\_Click"**

Content=**"Добавить/Обновить"**

FontSize=**"16"** />

<Button

x:Name=**"buttonDelete"**

Width=**"170"**

Height=**"35"**

Margin=**"10,0,5,0"**

Click=**"buttonDelete\_Click"**

Content=**"Удалить"**

FontSize=**"16"** />

</StackPanel>

</Grid>

</Window>

В AuthorList.xaml.cs поместим код ниже:

**using** Microsoft**.**EntityFrameworkCore**;**

**using** System**.**Windows**;**

**using** WPFTutorial**.**Model**;**

**namespace** WPFTutorial

**{**

**public** **partial** class AuthorList **:** Window

**{**

BookDBContext db **=** **new** BookDBContext**();**

**public** AuthorList**()**

**{**

InitializeComponent**();**

SetDataGridValue**();**

**}**

**private** void SetDataGridValue**()**

**{**

db**.**Authors**.**Load**();**

/\*Здесь мы конвертируем к ObservableCollection, т.к. OC реализует интерфейс оповещения об изменении коллекции,

а по скольку мы будем добавлять данные из DataGrid, то нам как-раз таки и пригодится оповещение об изменении

\*/

dataGridAuthors**.**ItemsSource **=** db**.**Authors**.**Local**.**ToObservableCollection**();**

**}**

**private** void buttonAddUpdate\_Click**(object** sender**,** RoutedEventArgs e**)**

**{**

db**.**SaveChanges**();**

MessageBox**.**Show**(**"Операция успешно выполнена!"**,** ""**,** MessageBoxButton**.**OK**,** MessageBoxImage**.**Information**);**

SetDataGridValue**();**

**}**

**private** void buttonDelete\_Click**(object** sender**,** RoutedEventArgs e**)**

**{**

Author author **=** dataGridAuthors**.**SelectedItem **as** Author**;**

db**.**Authors**.**Remove**(**author**);**

db**.**SaveChanges**();**

MessageBox**.**Show**(**"Операция успешно выполнена!"**,** ""**,** MessageBoxButton**.**OK**,** MessageBoxImage**.**Information**);**

**}**

**}**

**}**

Подытожим. На данном примере был показан способ создания базы данных на СУБД SQLite, её подключение к приложению, основные действия с записями в БД (создание, чтение, обновление, удаление). Также был предложен один из способов фильтрации.