

Содержание

Настройка БД	2
Разработка приложения	16

Настройка БД

Создание БД

1. Первым делом необходимо разработать БД в 3НФ и представить ERD в PDF формате.
2. Таблицы БД мы будем создавать на основе выданных excel-файлов.

Название	Поля
<p>Тип материала (MaterialTypes)</p> <p>Хранит информацию о типах материалов, используемых в производстве.</p>	<p>ID:</p> <pre>ID INT IDENTITY NOT NULL</pre> <p>Тип материала:</p> <pre>Title NVARCHAR 50 NOT NULL)</pre> <p>Процент брака:</p> <pre>ScrapRate DECIMAL(5,2) NOT NULL</pre>
<p>Тип продукции (ProductTypes)</p> <p>Хранит информацию о различных типах продукции, которые производит компания.</p>	<p>ID:</p> <pre>ID INT IDENTITY NOT NULL</pre> <p>Тип продукции:</p> <pre>Title NVARCHAR (50)</pre> <p>Коэффициент типа продукции:</p> <pre>Ratio DECIMAL (5,2)</pre>
<p>Продукция (Products)</p> <p>Хранит информацию о продуктах.</p>	<p>ID:</p> <pre>INT IDENTITY NOT NULL</pre>

	<p>Тип продукции:</p> <div>ProductTypeID INT NOT NULL</div> <p>Наименование продукции:</p> <div>Title NVARCHAR(200) NOT NULL</div> <p>Артикул:</p> <div>Article NCHAR(7) NOT NULL</div> <p>Минимальная стоимость д ля партнера:</p> <div>MinPartnerPrice (DECIMAL 11,2)</div>
<p>Партнеры (Partners)</p> <p>Хранит информацию о партнерах компании.</p>	<p>ID:</p> <div>ID INT IDENTITY NOT NULL</div> <p>Тип партнера:</p> <div>Type NCHAR(3) NOT NULL</div> <p>Наименование партнера:</p> <div>Title NVARCHAR(50) NOT NULL</div>

Директор:

```
Director  
NVARCHAR(100) NOT  
NULL
```

Электронная почта партнера:

```
Email NVARCHAR(100)  
NOT NULL
```

Телефон партнера:

```
Phone CHAR(13) NOT  
NULL
```

Юридический адрес партнера:

```
LegalAddress  
NVARCHAR(255) NOT  
NULL
```

ИНН:

```
INN CHAR(10) NOT NULL
```

Рейтинг:

```
Rating TINYINT NOT  
NULL
```

<p>Партнеры продукции (PartnerProducts)</p> <p>Хранит информацию о продажах продукции партнерам.</p>	<p>Продукция:</p> <div>ProductID INT NOT NULL</div> <p>Партнер:</p> <div>PartnerID INT NOT NULL</div> <p>Количество продукции:</p> <div>CountProduct BIGINT NOT NULL</div> <p>Дата продажи:</p> <div>DateSale DATE NOT NULL</div>
--	---

3. Для комплексного создания БД будем использовать запрос SQL:

```
CREATE TABLE [dbo].[MaterialTypes](
    [ID] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY,
    [Title] [nvarchar](50) NOT NULL,
    [ScrapRate] [decimal](5, 2) NOT NULL
)
GO

CREATE TABLE [dbo].[ProductTypes](
    [ID] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY,
    [Title] [nvarchar](50) NOT NULL,
    [Ratio] [decimal](5, 2) NOT NULL,
)
GO
```

```

CREATE TABLE [dbo].[Products](
    [ID] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY,
    [ProductTypeID] [int] NOT NULL,
    [Title] [nvarchar](200) NOT NULL,
    [Article] [nchar](7) NOT NULL UNIQUE,
    [MinPartnerPrice] [decimal](11, 2) NOT NULL
)
GO

```

```

CREATE TABLE [dbo].[Partners](
    [ID] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY,
    [Type] [nchar](3) NOT NULL CHECK
        ( [Type] IN ( 'ΠΑΟ', '3ΑΟ', 'ΟΑΟ', 'ΟΟΟ' ) ),
    [Title] [nvarchar](50) NOT NULL,
    [Director] [nvarchar](100) NOT NULL,
    [Email] [nvarchar](200) NOT NULL,
    [Phone] [char](13) NOT NULL,
    [LegalAddress] [nvarchar](255) NOT NULL,
    [INN] [char](10) NOT NULL,
    [Rating] [tinyint] NOT NULL CHECK
        ( [Rating] > 0 AND [Rating] < 11 )
)
GO

```

```

CREATE TABLE [dbo].[PartnerProducts](
    [ProductID] [int] NOT NULL FOREIGN KEY REFERENCES Products(Id)
        ON UPDATE CASCADE ON DELETE CASCADE,
    [PartnerID] [int] NOT NULL FOREIGN KEY REFERENCES Partners(Id)
        ON UPDATE CASCADE ON DELETE CASCADE,
    [CountProduct] [bigint] NOT NULL,
    [DateSale] [date] NOT NULL DEFAULT GETDATE(),
    PRIMARY KEY (ProductID, PartnerID)
)
GO

```

Импорт данных в БД

Импорт данных в независимую таблицу

Независимая таблица – это таблица, не содержащая внешних ключей (разберем импорт на таблице ProductTypes).

1. Добавим столбец ID в самое начало.

	A	B	C
1	ID	Тип продукции	Коэффициент типа продукции
2		Ламинат	2,35
3		Массивная доска	5,15
4		Паркетная доска	4,34
5		Пробковое покрытие	1,5

Рисунок 1 – Добавление колонки ID

2. Скопируем данные вместе с пустым столбцом ID (без заголовков).

	A	B	C
1	ID	Тип продукции	Коэффициент типа продукции
2		Ламинат	2,35
3		Массивная доска	5,15
4		Паркетная доска	4,34
5		Пробковое покрытие	1,5

Рисунок 2 – Копирование данных

3. В SSMS найдём таблицу ProductTypes, нажмем правой кнопкой мыши, далее – Изменить первые 200 строк.

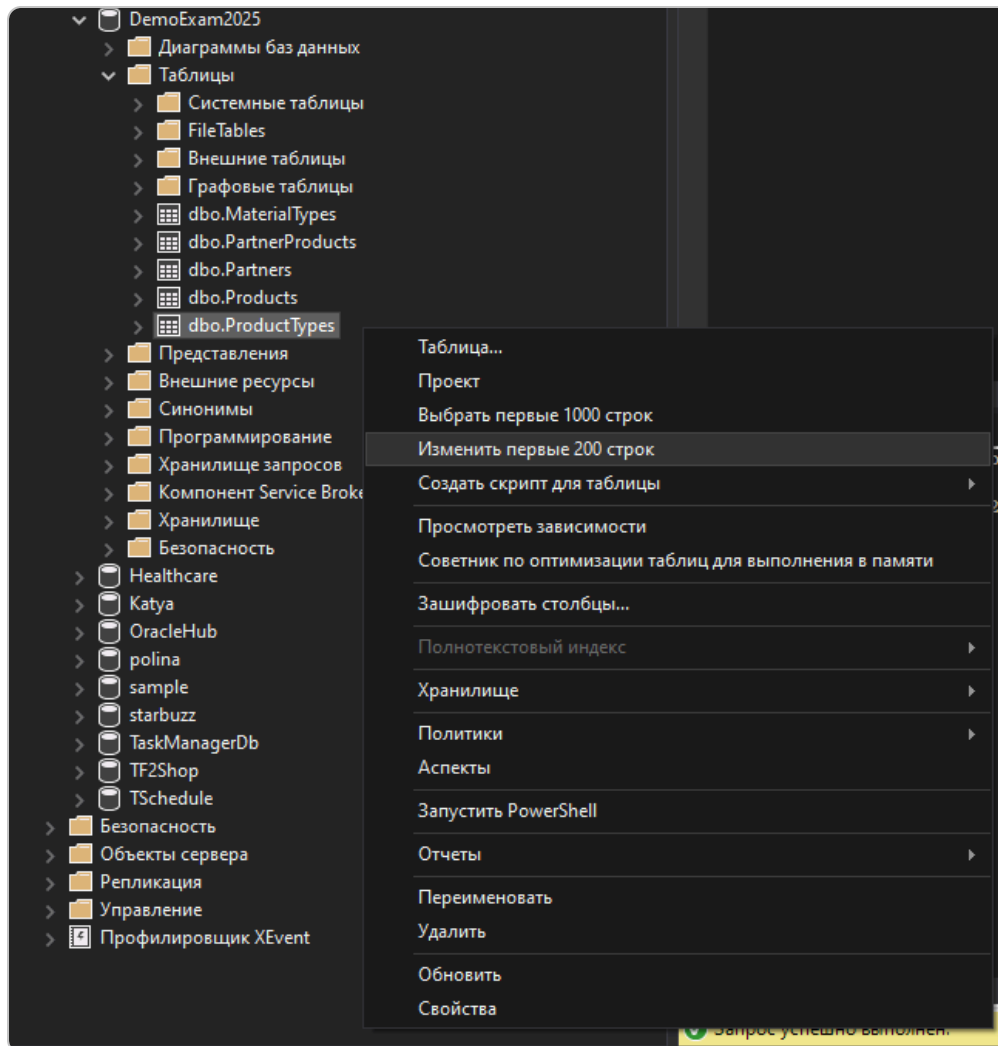


Рисунок 3 – Изменение данных таблицы

4. В открывшейся вкладке жмем на самую первую ячейку, а затем вставляем скопированные ячейки.

Импорт данных в зависимую таблицу

Зависимая таблица – это таблица, ограниченная внешними ключами (разберем пример импорта на таблице Products)

1. У таблицы Products существует зависимость от таблицы ProductTypes по полю ID, поэтому для начала нужно заполнить эту пустую колонку.

	A	B	C
1	ID	Тип продукции	Коэффициент типа продукции
2	1	Ламинат	2,35
3	2	Массивная доска	5,15
4	3	Паркетная доска	4,34
5	4	Пробковое покрытие	1,5

Рисунок 4 – Заполнение колонки ID

2. Выделим заголовок столбца «Тип продукции» и отсортируем его от А до Я.

Рисунок 5 – Сортировка колонки по алфавиту

⚠ Как можно заметить – ничего не изменилось. Пугаться не стоит, это лишь потому, что значения уже расположены в алфавитном порядке. В каждом Excel-листе необходимо сортировать зависимый столбец для правильной работы формулы ПРОСМОТР у зависимой таблицы.

3. Точно так же добавим столбец ID.

	A	B	C
1	ID	Тип продукции	Наименование продукции
2		Паркетная доска	Паркетная доска Ясень темный однополосная 14 мм
3		Паркетная доска	Инженерная доска Дуб Французская елка однополосная 12 мм
4		Ламинат	Ламинат Дуб дымчато-белый 33 класс 12 мм
5		Ламинат	Ламинат Дуб серый 32 класс 8 мм с фаской
6		Пробковое покрытие	Пробковое напольное клеевое покрытие 32 класс 4 мм

Рисунок 6 – Добавление колонки ID

4. Перенесем названия типов продукции в отдельную колонку для будущего сравнения с данными из ссылающейся таблицы.

	A	B	C	D	E	F	G
1	ID	Тип продукции	Наименование продукции	Артикул	Минимальная стоимость для партнера		
2			Паркетная доска Ясень темный однополосная 14 мм	8758385	4456,90		Паркетная доска
3			Инженерная доска Дуб Французская елка однополосная 12 мм	8858958	7330,99		Паркетная доска
4			Ламинат Дуб дымчато-белый 33 класс 12 мм	7750282	1799,33		Ламинат
5			Ламинат Дуб серый 32 класс 8 мм с фаской	7028748	3890,41		Ламинат
6			Пробковое напольное клеевое покрытие 32 класс 4 мм	5012543	5450,59		Пробковое покрытие

Рисунок 7 – Перенос данных ссылающейся таблицы

5. Далее в пустую ячейку колонки Тип продукции занесем формулу:

```
=ПРОСМОТР(G2;Product_type_import.xlsx!$B$2:$B$5;Product_type_import.xlsx!$A$2:$A$5)
```

⚠ Справочная информация: **G2** – ячейка с названием типа продукции. **Product_type_import.xlsx** – наименование книги Excel с данными для сравнения. **!\$B\$2:\$B\$5** – колонка с типом продукции из листа ProductType (значение для сравнения). **!\$A\$2:\$A\$5** – колонка с ID из листа ProductType (желаемое значение для подстановки).

6. Растянем полученное значение на весь столбик.

B2							=ПРОСМОТР(G2;Product_type_import.xlsx!\$B\$2:\$B\$5;Product_type_import.xlsx!\$A\$2:\$A\$5)
	A	B	C	D	E	F	G
1	ID	Тип продукции	Наименование продукции	Артикул	Минимальная стои		
2			Паркетная доска Ясень темный однополосная 14 мм	8758385			
3			Инженерная доска Дуб Французская елка однополосная 12 мм	8858958			
4			Ламинат Дуб дымчато-белый 33 класс 12 мм	7750282			
5			Ламинат Дуб серый 32 класс 8 мм с фаской	7028748			
6			Пробковое напольное клеевое покрытие 32 класс 4 мм	5012543			

Рисунок 8 – Заполнение всех ячеек в колонке Тип продукции

7. Вставим полученную таблицу в SSMS (копируем без заголовков и вспомогательного столбца G).

	A	B	C	D	E
1	ID	Тип продукции	Наименование продукции	Артикул	Минимальная стоимость для партнера
2			Паркетная доска Ясень темный однополосная 14 мм	8758385	4456,90
3			Инженерная доска Дуб Французская елка однополосная 12 мм	8858958	7330,99
4			Ламинат Дуб дымчато-белый 33 класс 12 мм	7750282	1799,33
5			Ламинат Дуб серый 32 класс 8 мм с фаской	7028748	3890,41
6			Пробковое напольное клеевое покрытие 32 класс 4 мм	5012543	5450,59

Рисунок 9 – Копирование нужных ячеек

SQLQuery1.sql - lo...m2025 (roman (60))*		DESKTOP-LTVFD42\S...25 - dbo.Products			
	ID	ProductTypeID	Title	Article	MinPartnerPrice
	1	3	Паркетная дос...	8758385	4456,90
	2	3	Инженерная д...	8858958	7330,99
	3	1	Ламинат Дуб д...	7750282	1799,33
	4	1	Ламинат Дуб с...	7028748	3890,41
	5	4	Пробковое на...	5012543	5450,59
	NULL	NULL	NULL	NULL	NULL

Рисунок 10 – Вставка в таблицу MS SQL

8. Точно также заполняются остальные таблицы.

Создание резервной копии

После создания БД и импорта данных необходимо создать её резервную копию.

1. Кликнуть на БД правой кнопкой мыши и выбрать Задачи | Создать резервную копию.

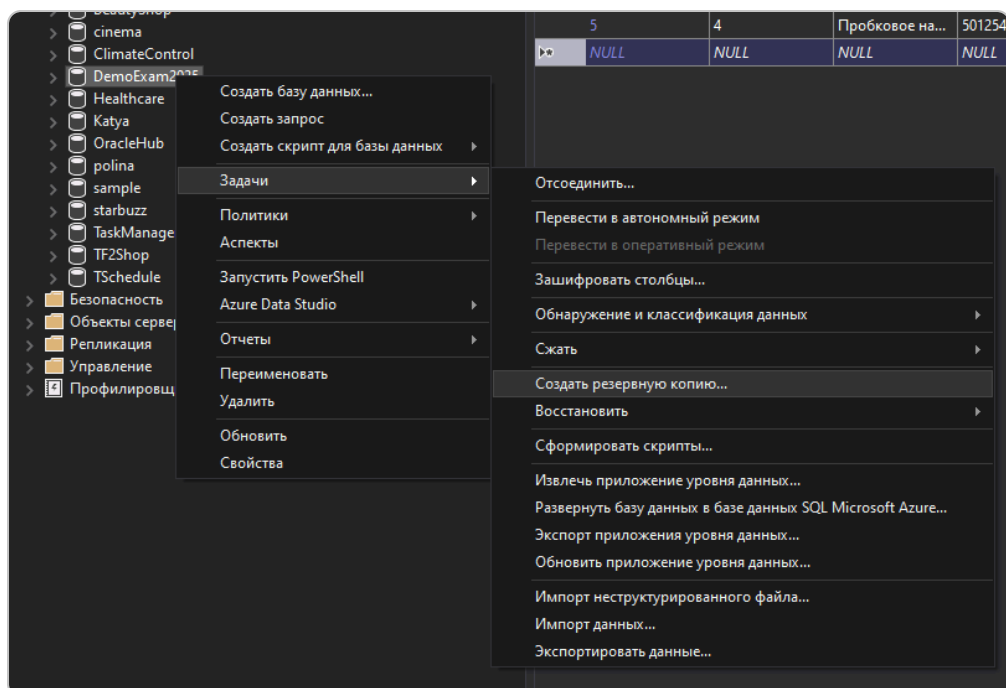


Рисунок 11 – Создание резервной копии БД

2. Выбираем место для сохранения резервной копии БД: нажимаем на кнопку Добавить, затем на ..., после указания папки вписываем название копии БД и нажимаем ОК, ещё раз на ОК и в главном окне тоже нажимаем на ОК. Далее мастер уведомит об успехе, либо об ошибке копирования.

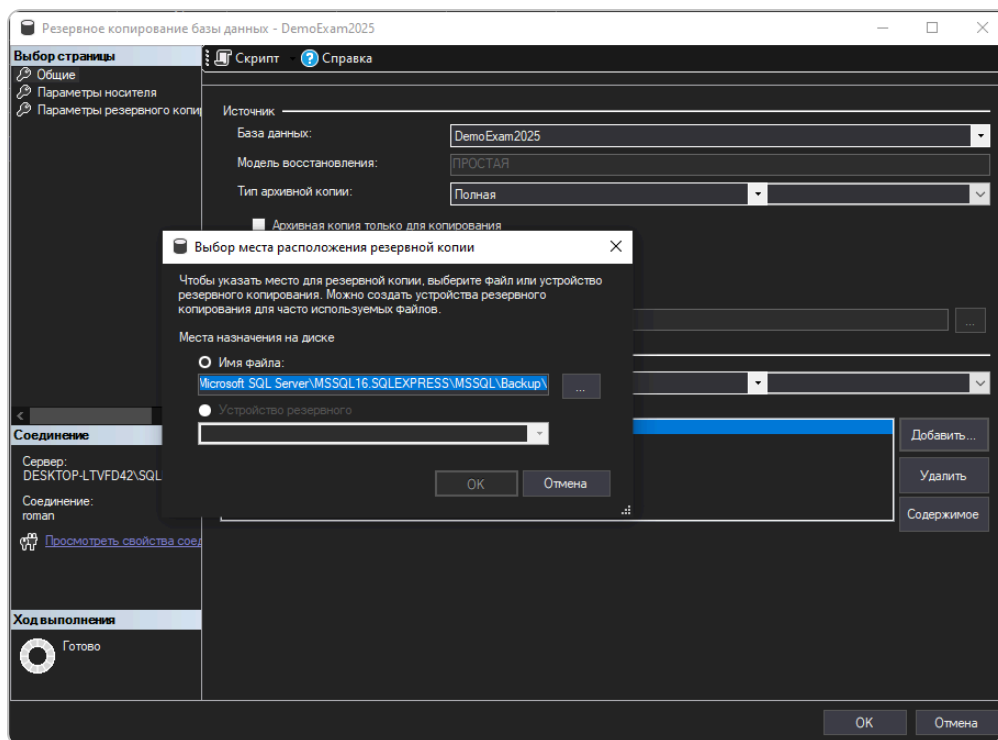


Рисунок 12 – Настройка сохранения резервной копии БД

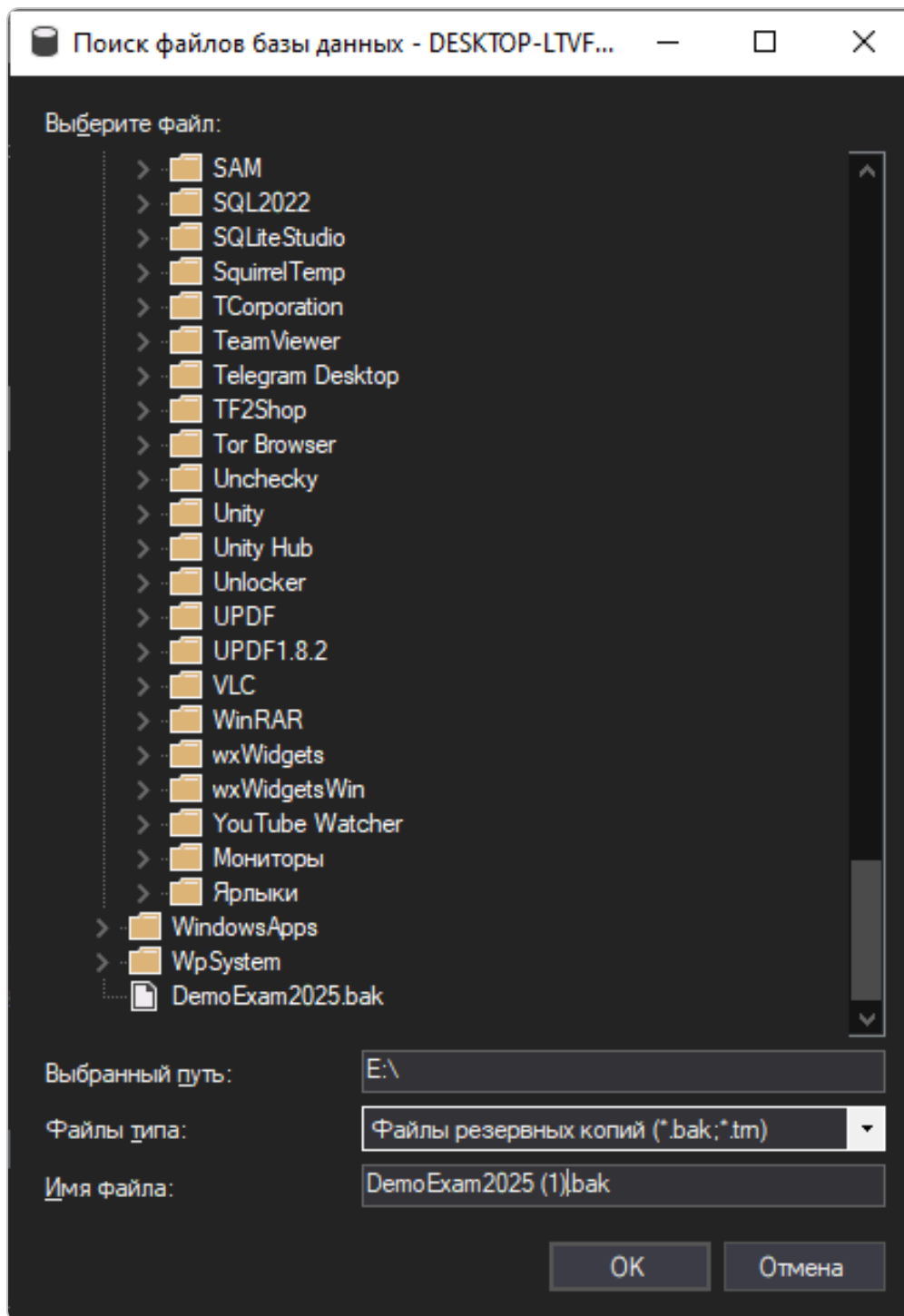


Рисунок 13 – Выбор места сохранения резервной копии БД

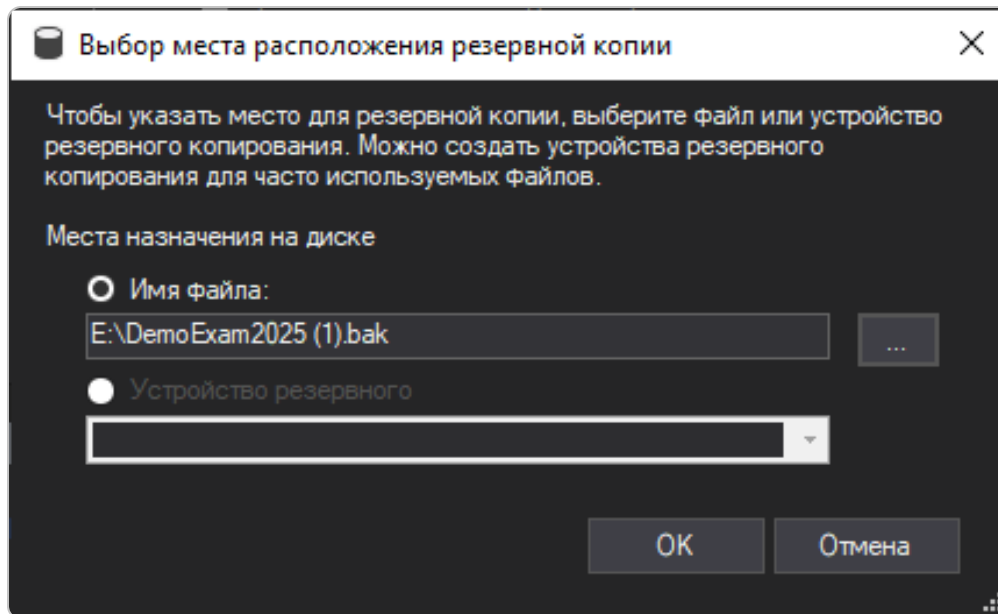


Рисунок 14 – Выбранное место для хранения сохранилось

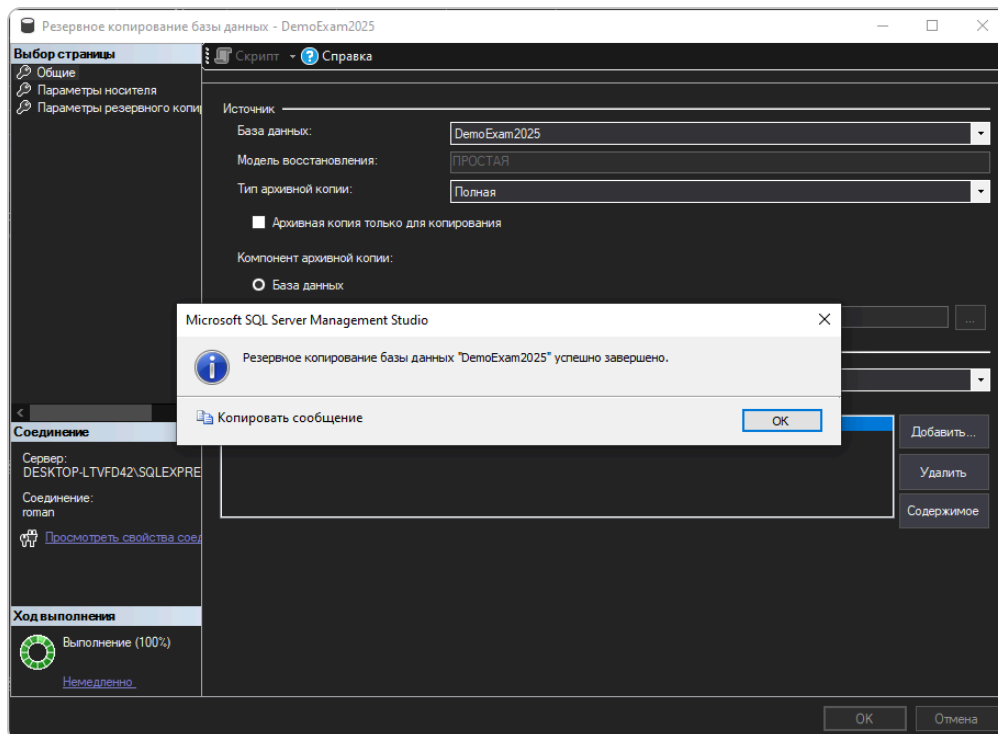


Рисунок 15 – Успешное создание резервной копии БД

Разработка приложения

Создание проекта

1. Открываем Visual Studio и нажимаем Создание проекта.
2. Выбираем шаблон Приложение WPF (.NET Framework).
3. Указываем подходящее название, расположение, ставим галочку над условием Поместить решение и проект в одном каталоге и выбираем последнюю версию .NET Framework.

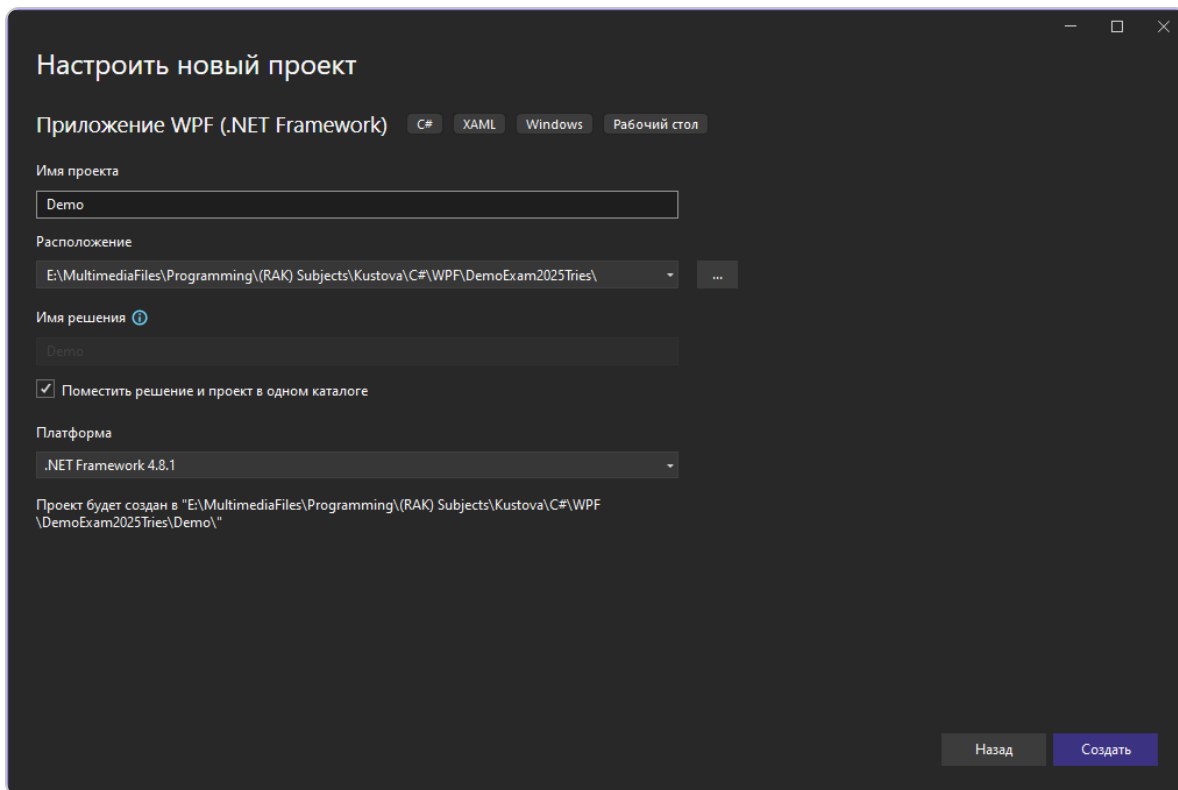


Рисунок 1 – Настройка нового проекта

4. Для установки иконки приложения нужно открыть свойства проекта и указать путь к иконке.

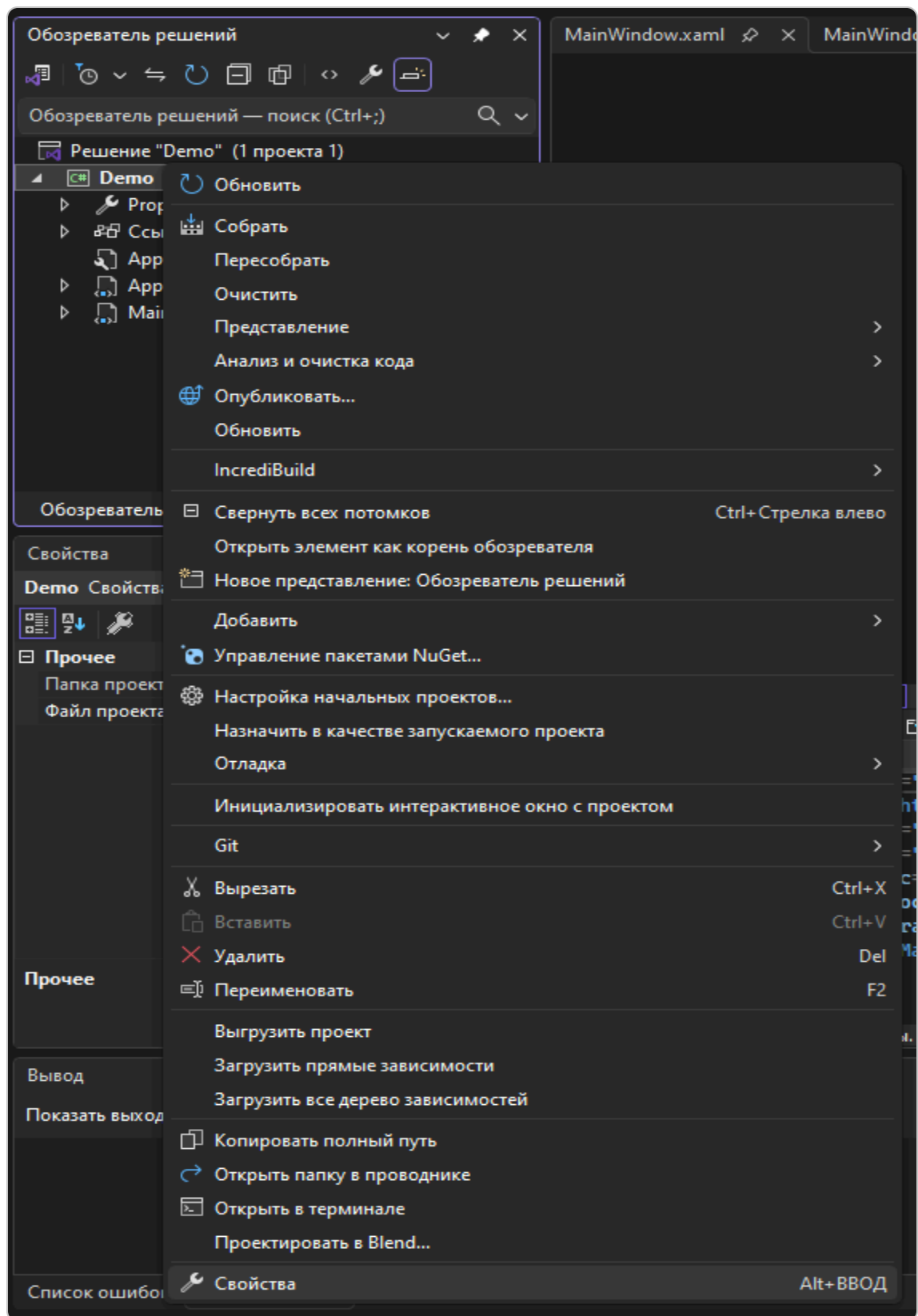


Рисунок 2 – Путь к свойствам проекта

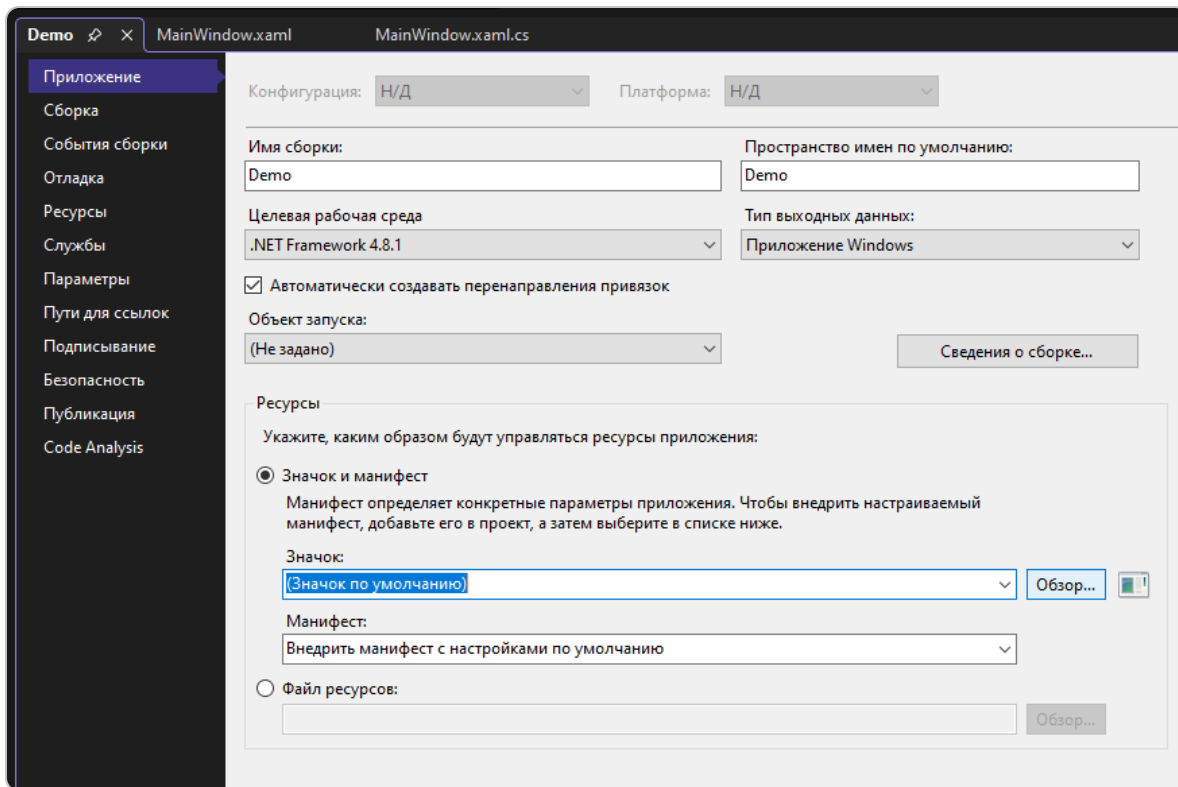


Рисунок 3 – Установление иконки приложения

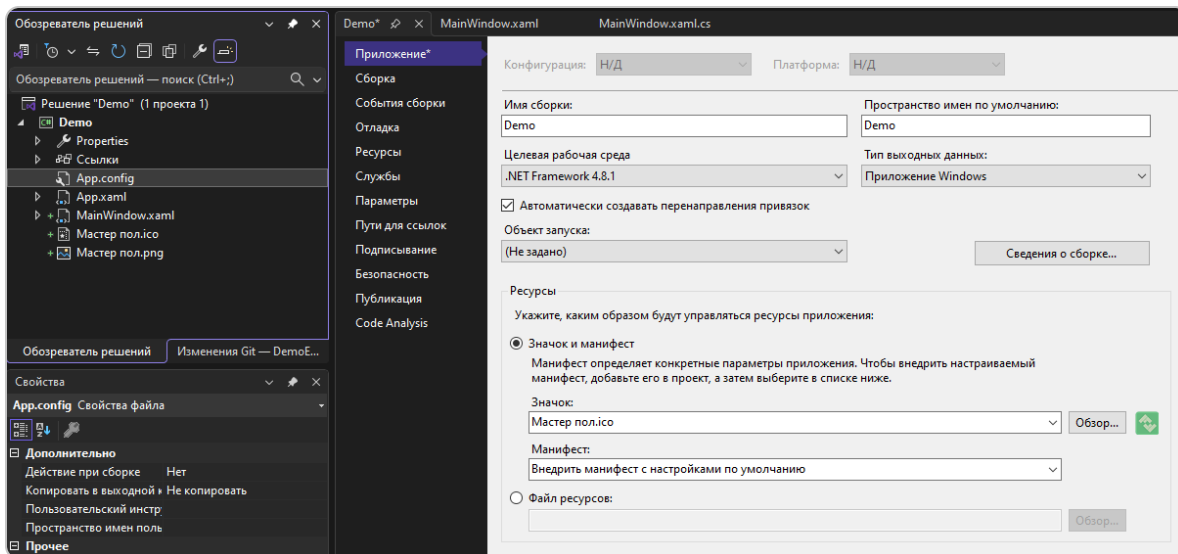


Рисунок 4 – Иконка приложения установлена

Подключение БД

Мы воспользуемся жестким фреймворком Entity Framework для подключения к MS SQL БД.

1. Создадим папку Data.
2. Внутри папки кликаем правой кнопкой мыши и выбираем Создать элемент, из списка выбираем Модель ADO.NET EDM и даем ей незамысловатое название.

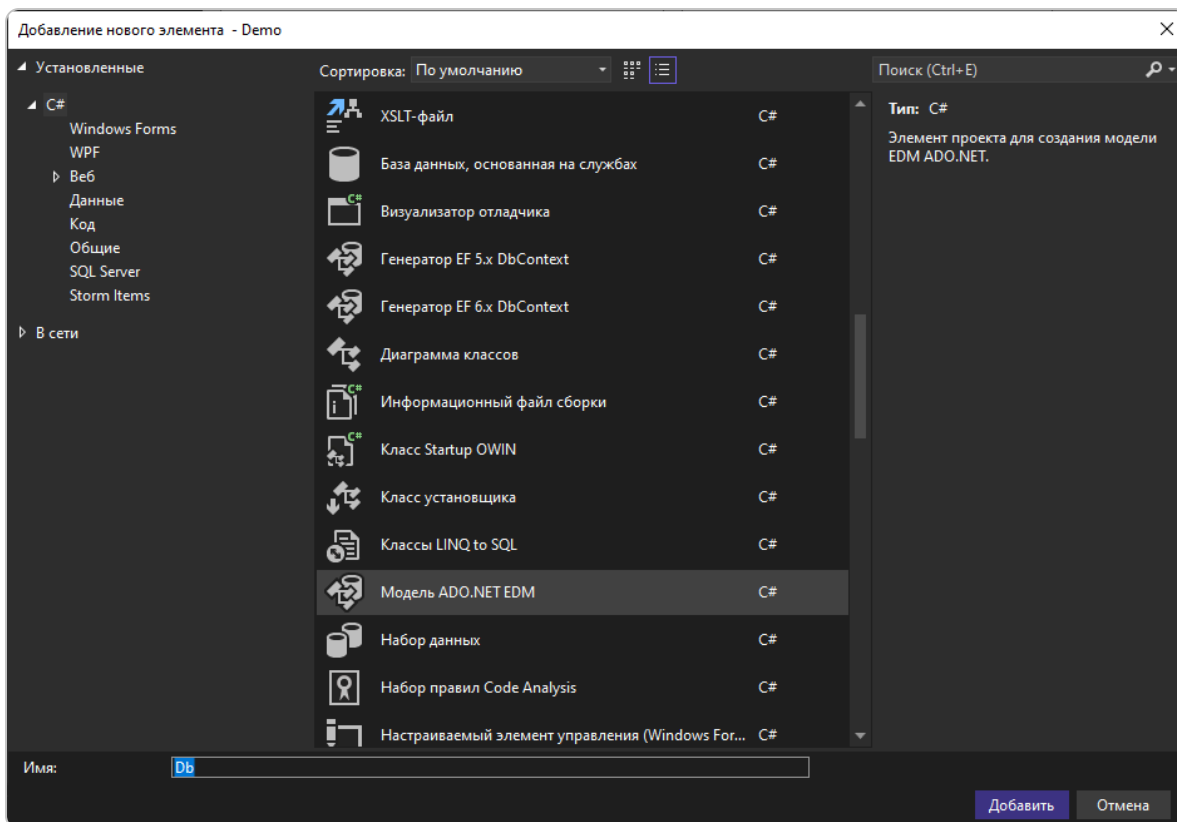


Рисунок 5 – Создание EDM модели

3. В Мастере выбираем Конструктор EF из базы данных.

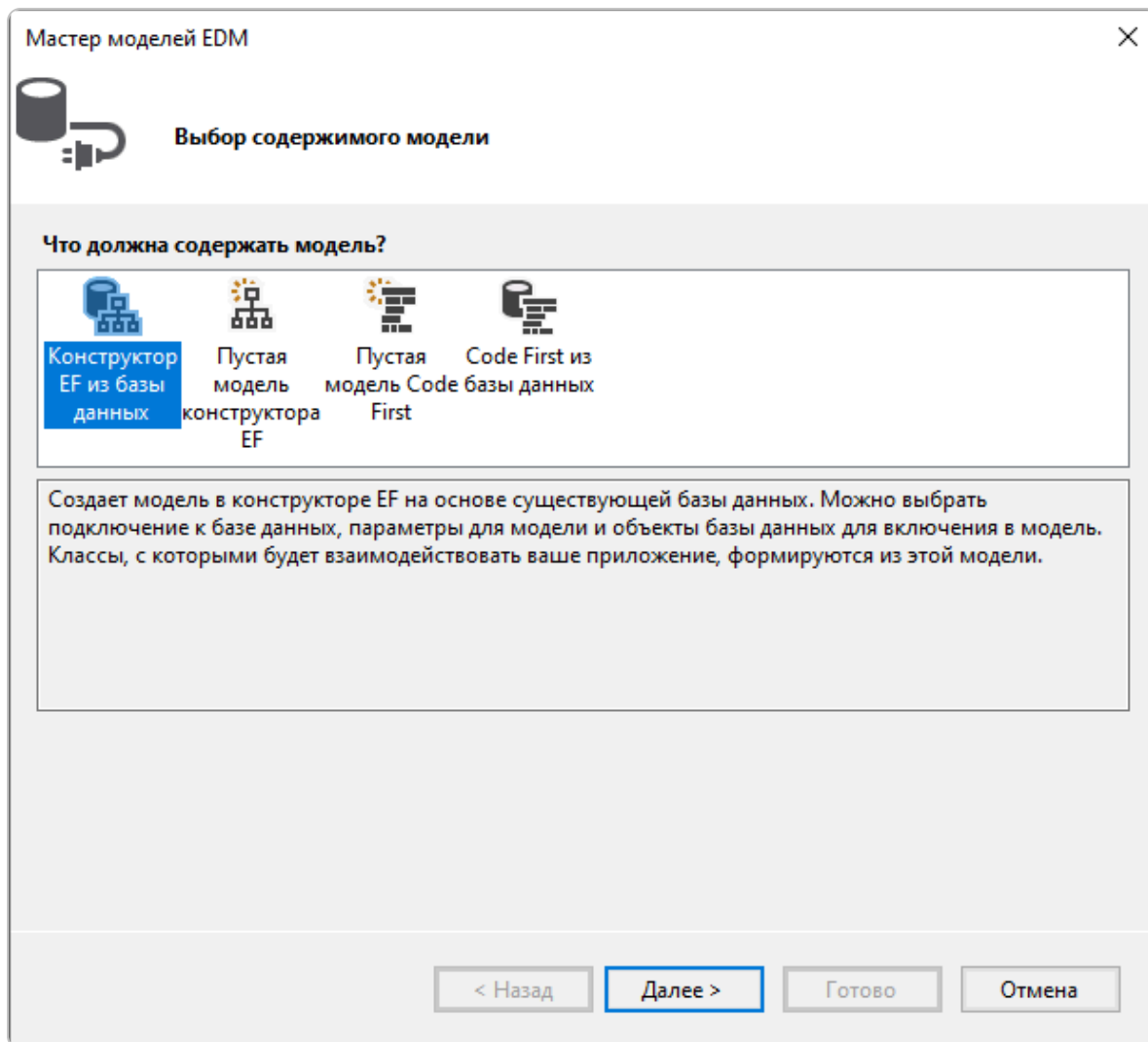


Рисунок 6 – Выбор содержимого модели

4. Следом нажимаем на кнопку Создать соединение.

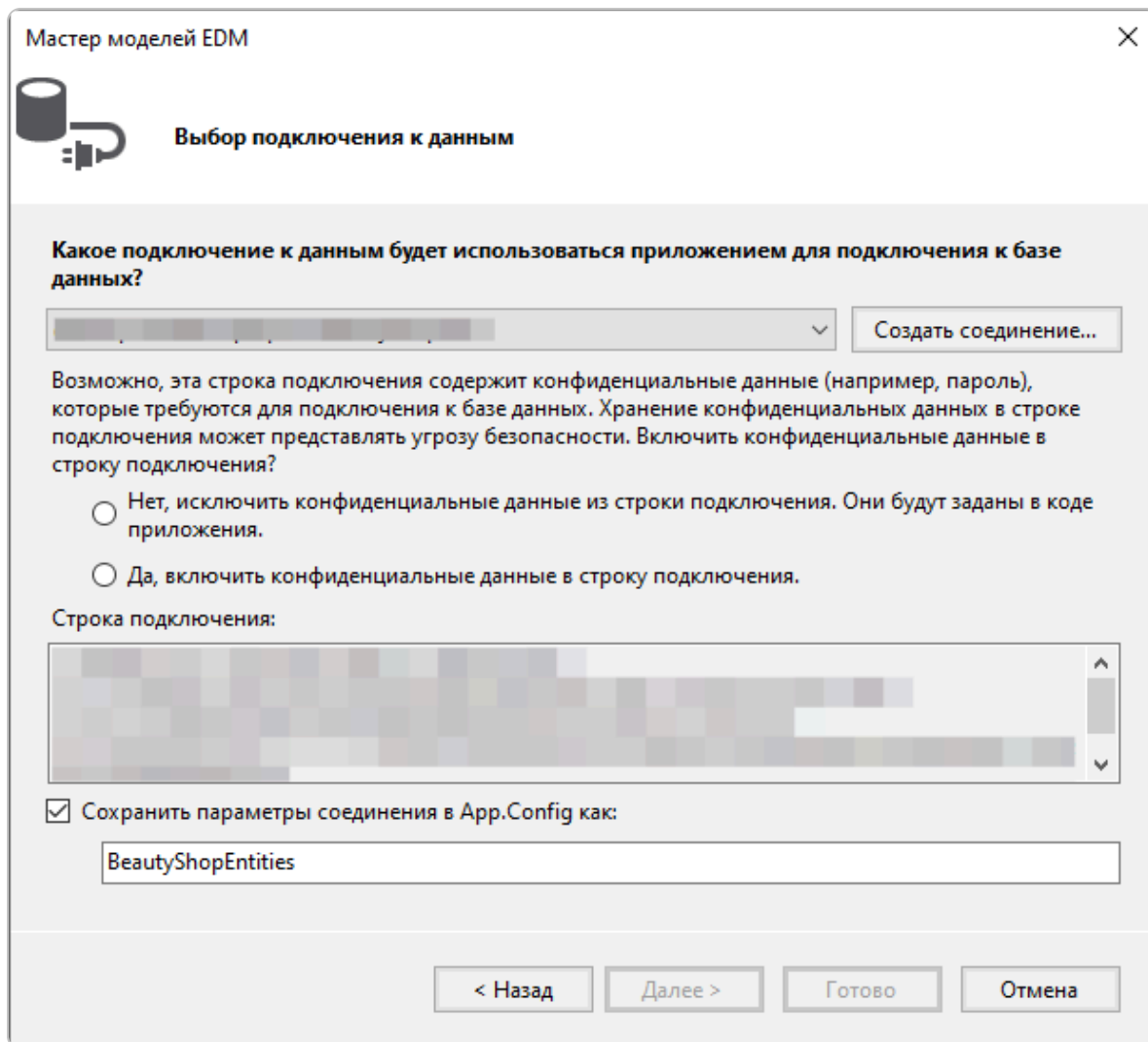


Рисунок 7 – Выбор подключения к данным

5. Здесь вводим Имя сервера (можно получить из SSMS открыв свойства сервера), на всякий случай поставим галочку у Сертификат сервера.
6. Если все успешно, то заработает список с Выбором базы данных – выбираем нужную нам БД

Свойства подключения

Введите данные для подключения к выбранному источнику данных или нажмите кнопку "Изменить", чтобы выбрать другой источник данных и (или) поставщик.

Источник данных:
Microsoft SQL Server (SqlClient) Изменить...

Имя сервера:
localhost Обновить

Вход на сервер

Проверка подлинности: Проверка подлинности Windows

Имя пользователя:

Пароль:

Шифровать: Mandatory (True)

☒ Сертификат сервера:
☐ Сохранить мой пароль

Подключение к базе данных

☒ Выберите или введите имя базы данных:
DemoExam2025

☐ Прикрепить файл базы данных:
 Обзор...

Логическое имя:


Дополнительно...

Проверить подключение OK Отмена

Рисунок 8 – Свойства подключения

7. Далее указываем понравившееся название для сущностей.

Мастер моделей EDM

 Выбор подключения к данным

Какое подключение к данным будет использоваться приложением для подключения к базе данных?

desktop-ltvfd42\sqlexpress.DemoExam2025.dbo1 Создать соединение...

Возможно, эта строка подключения содержит конфиденциальные данные (например, пароль), которые требуются для подключения к базе данных. Хранение конфиденциальных данных в строке подключения может представлять угрозу безопасности. Включить конфиденциальные данные в строку подключения?

☐ Нет, исключить конфиденциальные данные из строки подключения. Они будут заданы в коде приложения.

☐ Да, включить конфиденциальные данные в строку подключения.

Строка подключения:

```
metadata=res://*/Data.Db.csdl|res://*/Data.Db.ssdl|
res://*/Data.Db.msl;provider=System.Data.SqlClient;provider connection string="data
source=localhost;initial catalog=DemoExam2025;integrated
security=True;trustservercertificate=True;MultipleActiveResultSets=True;App=EntityFramework"
```

☒ Сохранить параметры соединения в App.Config как:

DemoExam2025Entities

< Назад Далее > Готово Отмена

Рисунок 9 – Настройка подключения 2

8. Выбираем новейшую версию Entity Framework.

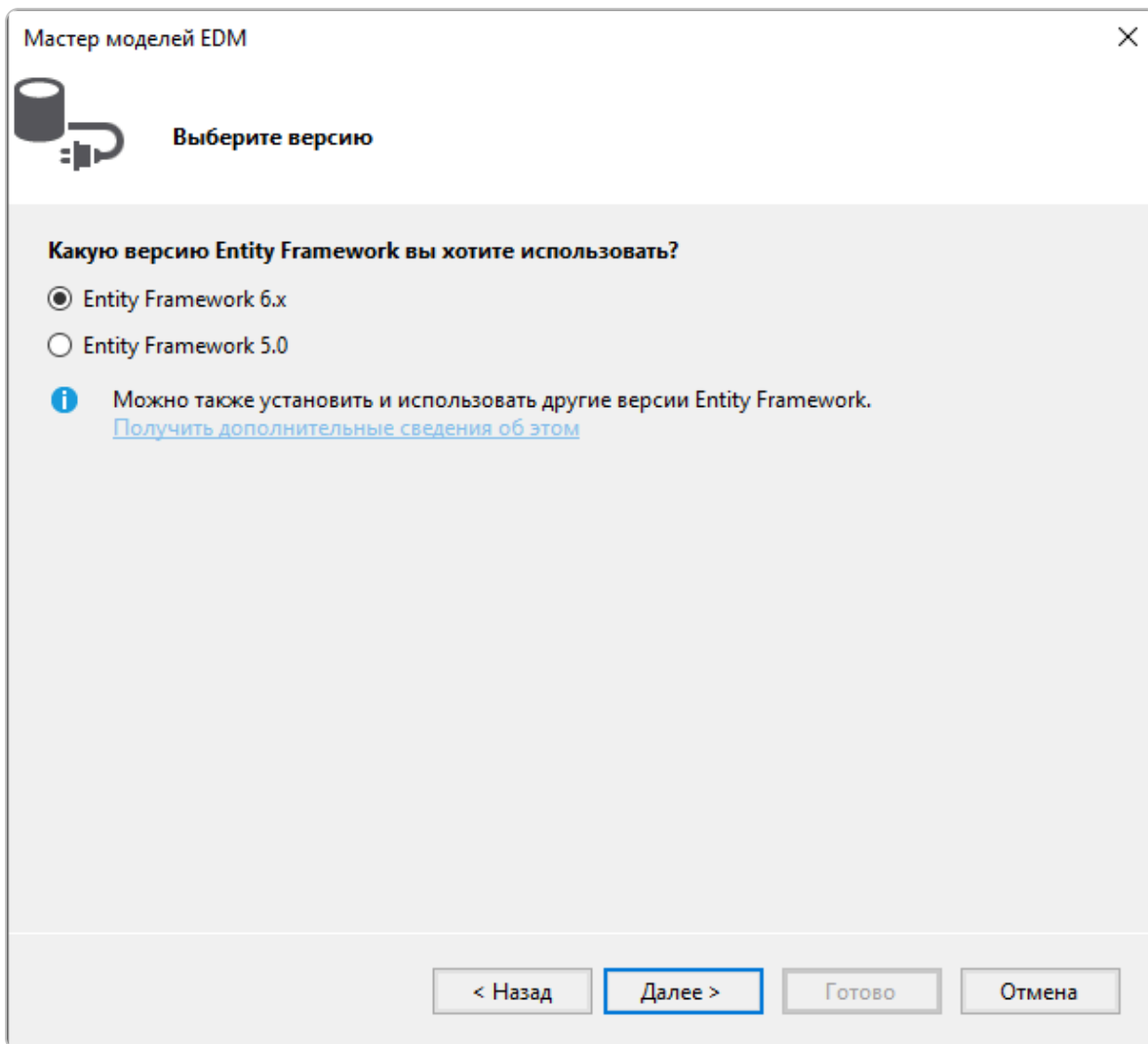


Рисунок 10 – Выбор версии

9. Ставим галочку у Таблицы и у Формировать имена объектов во множественном или единственном числе, другие параметры оставляем без изменений и нажимаем Готово

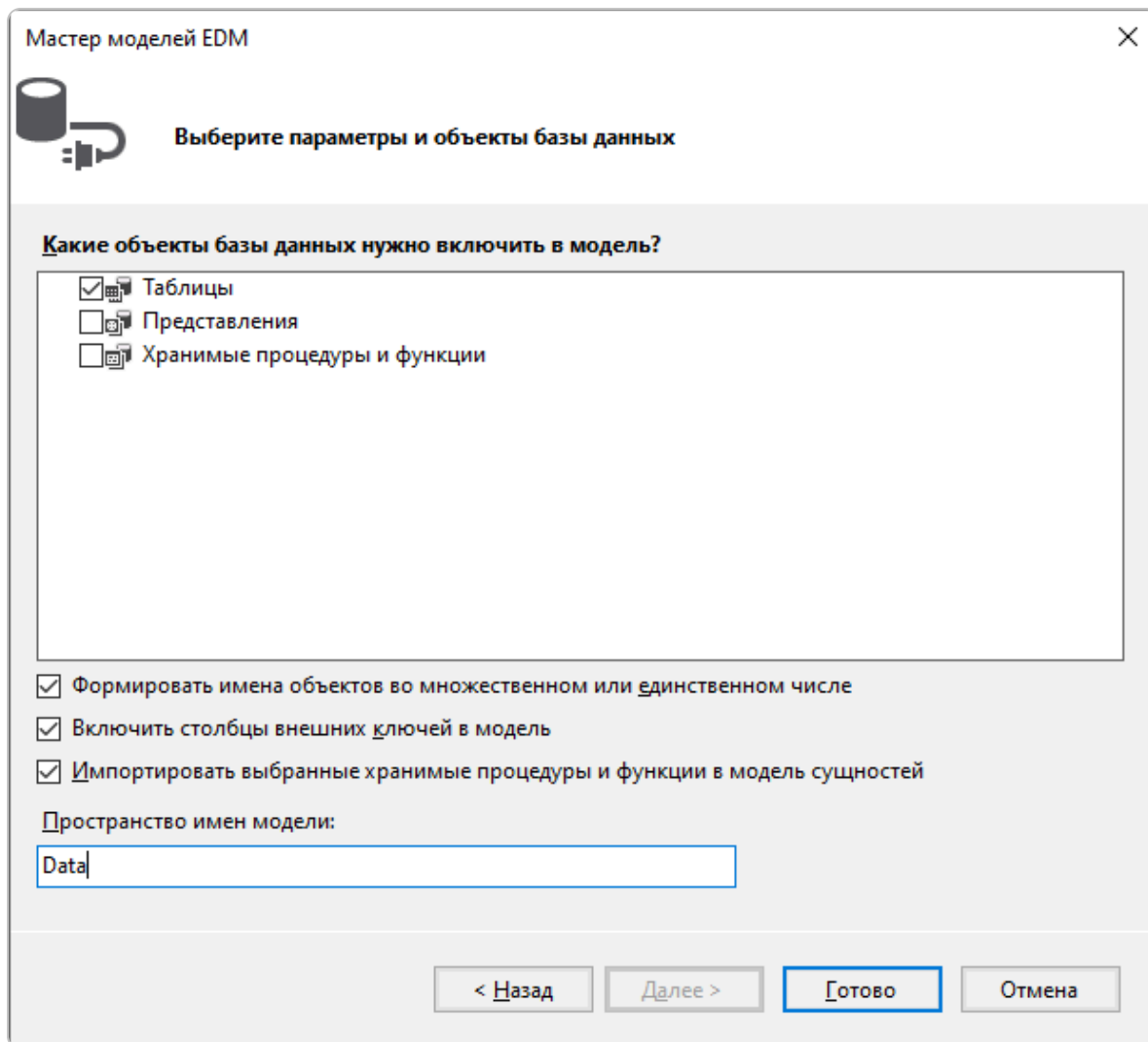


Рисунок 11 – Выбор параметров и объектов БД

10. Если появятся какие-то окошки с предупреждениями об опасности, соглашаемся со всем.
11. Через некоторое время все будет готово и появится диаграмма с созданными моделями.

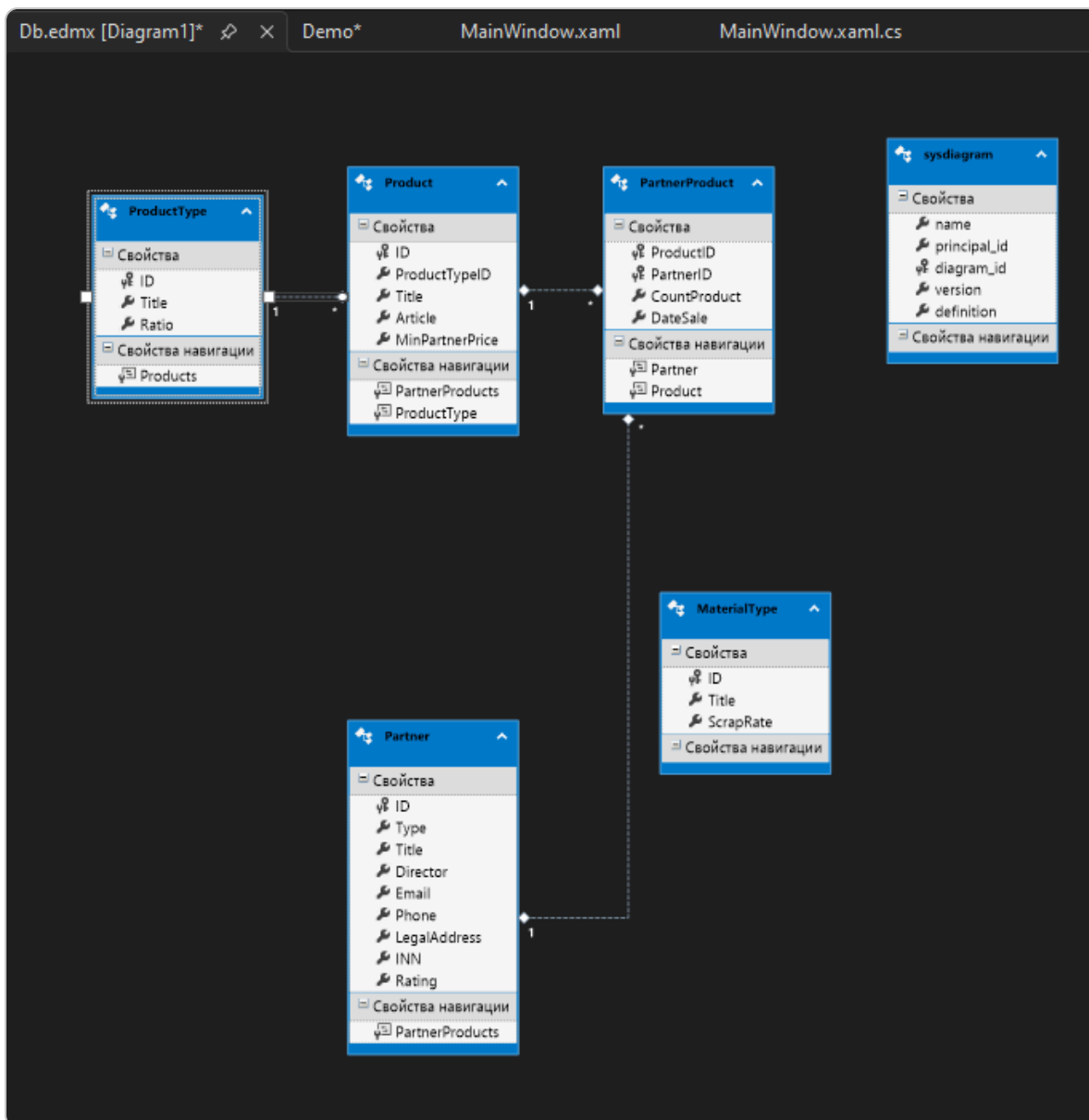


Рисунок 12 – Диаграмма БД

12. Доступ к контексту сделаем с помощью статического поля в классе App (файл App.xaml.cs). Так мы сможем получать контекст отовсюду с помощью **App.Context**

```
using Demo.Data;

namespace Demo
{
    public partial class App
    {
```

```

    public static readonly DemoExam2025Entities Context
        = new DemoExam2025Entities();
    }
}

```

Стили

Вот что нам дано в Руководстве по стилю:

Руководство по стилю

Все экранные формы пользовательского интерфейса должны иметь заголовок. Кроме того, на главной форме должен быть установлен логотип (представлен в ресурсах). Логотип не искажать: не менять изображение, пропорции изображения, цвет.

Для приложения должна быть установлена иконка (дана в ресурсах), если это реализуемо в рамках платформы.

Использовать шрифт Segoe UI.

В качестве основного фона используется белый цвет, дополнительного фона используется цвет #F4E8D3. Для акцентирования внимания пользователя на целевом действии интерфейса используется цвет #67BA80.

Основной фон	Дополнительный фон	Акцентирование внимания
#FFFFFF	#F4E8D3	#67BA80

Рисунок 13 – Руководство по стилю

Для начала реализуем эти стили:

1. Создадим папку Styles.
2. В меню выберем Добавить|Словарь ресурсов и назовем его Style.

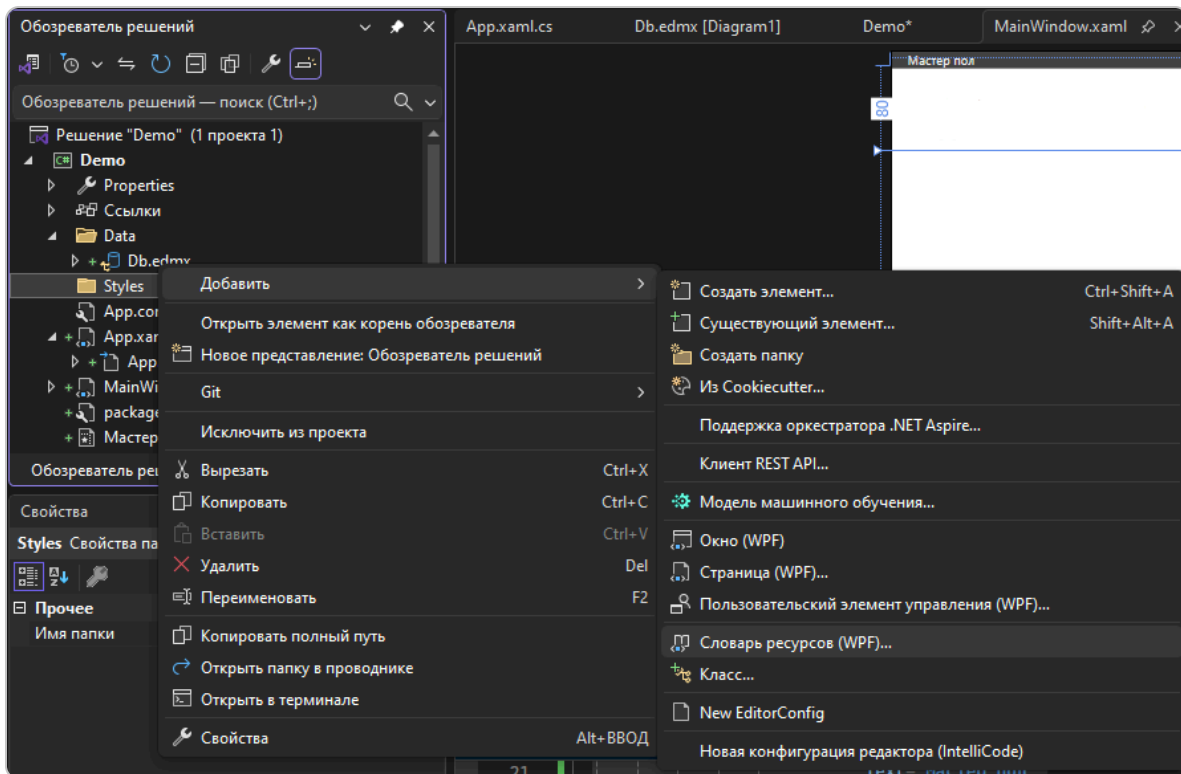


Рисунок 14 – Создание словаря ресурсов

3. Далее нужно указать здесь стили:

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <SolidColorBrush x:Key="PrimaryColor" Color="White"/>
    <SolidColorBrush x:Key="SecondaryColor" Color="#F4E8D3"/>
    <SolidColorBrush x:Key="AccentColor" Color="#67BA80"/>

    <Style TargetType="TextBlock">
        <Setter Property="FontFamily" Value="Segoe UI"/>
    </Style>

    <Style TargetType="Border">
        <Setter Property="CornerRadius" Value="4"/>
    </Style>
```

```

<Style TargetType="TextBox">
    <Setter Property="Margin" Value="4"/>
    <Setter Property="Padding" Value="8,4"/>
</Style>

<Style TargetType="ComboBox">
    <Setter Property="Margin" Value="4"/>
    <Setter Property="Padding" Value="8,4"/>
</Style>

<Style TargetType="Button">
    <Setter Property="Cursor" Value="Hand"/>
    <Setter Property="Padding" Value="20,8"/>
    <Setter Property="Background" Value="{DynamicResource
AccentColor}"/>
</Style>

<Style TargetType="ListViewItem">
    <Setter Property="Margin" Value="4"/>
    <Setter Property="BorderThickness" Value="1"/>
    <Setter Property="BorderBrush" Value="#131517"/>
</Style>

</ResourceDictionary>

```

⚠ **SolidColorBrush** это кисть, которая содержит цвет, к ней можно будет обратиться через {DynamicResource НазваниеКисти}. Границы будут чуть-чуть округлые. Шрифт элементов **TextBlock** будет Segoe UI. У кнопок **Button** будет акцентный цвет и особый курсор. **TextBox** и **ComboBox** будут иметь внешние и внутренние отступы. У элементов списка **ListViewItem** будет внешний отступ и темная граница.

4. Теперь нужно подключить этот файл: заходим в App.xaml и прописываем **ResourceDictionary**:

```

<Application x:Class="Demo.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:Demo"
    StartupUri="MainWindow.xaml">
    <Application.Resources>
        <ResourceDictionary Source="/Styles/Style.xaml"/>
    </Application.Resources>
</Application>

```

Окна

Главное окно

Наш список партнеров должен быть схож с предоставленным дизайном.

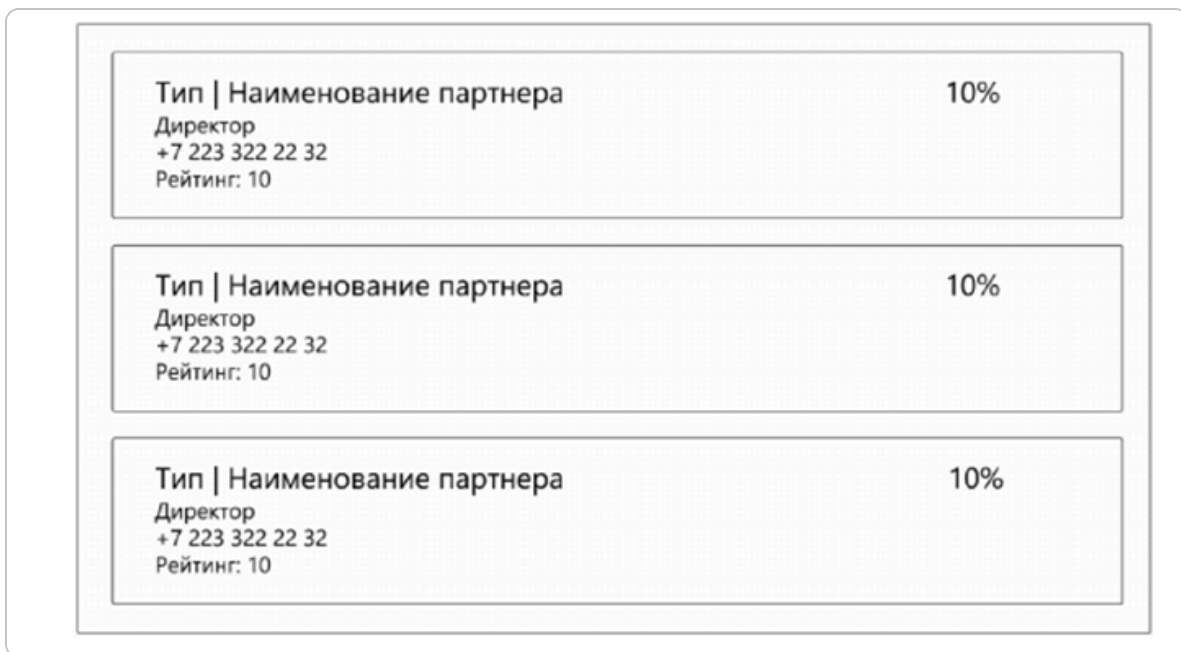


Рисунок 15 – Макет списка партнеров

Первым делом изменим размеры и заголовок окна:

```

<Window x:Class="Demo.MainWindow"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

```

```

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
    Width="800"
    Height="450"
    MinWidth="600"
    MinHeight="300"
    mc:Ignorable="d"
    Title="Мастер пол">
    <Grid>

    </Grid>
</Window>

```

Создадим разбиения на строки с помощью **Grid.RowDefinitions**.

```

<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="80"/>
        <RowDefinition/>
    </Grid.RowDefinitions>
</Grid>

```

Теперь, когда есть 2 строки мы можем создать заголовок:

```

<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="80"/>
        <RowDefinition/>
    </Grid.RowDefinitions>

    <Border Background="{DynamicResource SecondaryColor}">
        <StackPanel Margin="12"
            Orientation="Horizontal">
            <Image Margin="0,0,12,0"
                Source="/Мастер пол.png"/>

```

```

        <TextBlock FontSize="32"
                  Text="Мастер пол"
                  FontWeight="SemiBold"
                  VerticalAlignment="Center"/>
    </StackPanel>
</Border>
</Grid>

```

⚠ **Border** нужен для цвета заднего фона, который мы получаем из Style.xaml. **StackPanel** нужен для расположения элементов в линию. **Image** содержит изображения по пути *Source*. **TextBlock** выровняли по центру вертикали и дали полужирный шрифт.

После заголовка следует создать контент. Воспользуемся элементом **ListView**

⚠ **ListView** – это элемент в виде списка, предназначенный для отображения данных одного типа.

Будем делать настройку **ListView** последовательно:

1. Создадим **ListView** и укажем простые стили.

```

<ListView Grid.Row="1"
           x:Name="PartnersListView"
           HorizontalContentAlignment="Stretch">

</ListView>

```

2. Добавим **ItemTemplate**

⚠ **ItemTemplate** изменяет отображение элемента **ListView**: по умолчанию это простой **TextBlock**


```

<ListView Margin="8"
    Grid.Row="1"
    x:Name="PartnersListView"
    d:ItemsSource="{d:SampleData}"
    HorizontalContentAlignment="Stretch">
    <ListView.ItemTemplate>
        <DataTemplate>

        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>

```

⚠ d:ItemsSource помогает посмотреть, как бы выглядел наполненный данными **ListView**.

3. Для похожего дизайна используем Grid с двумя колонками:

```

<Grid Margin="10"
    HorizontalAlignment="Stretch">
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition Width="Auto"/>
    </Grid.ColumnDefinitions>
</Grid>

```

4. Для отображения типа и наименования партнера создадим новый класс в папке Data, назовем его *Partners*, сделаем его *public partial* и переименуем в *Partner*, затем добавим новое свойство *TypeName*, которое будет предоставлять нужную нам информацию о партнере (Тип | Наименование), а также рассчитаем его скидку:

```

using System.Linq;

namespace Demo.Data

```

```

{
    public partial class Partner
    {
        public string TypeName => $"{Type} | {Title}";
        public byte Discount
        {
            get
            {
                var productCount = PartnerProducts.Sum(product =>
product.CountProduct);
                if (productCount < 9999)
                    return 0;
                if (productCount < 49999)
                    return 5;
                if (productCount < 299999)
                    return 10;
                return 15;
            }
        }
    }
}

```

5. Расположим данные о скидке во второй колонке, а все остальное оформим вертикальным списком:

```

<Grid Margin="10"
    HorizontalAlignment="Stretch">
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition Width="Auto"/>
    </Grid.ColumnDefinitions>

    <StackPanel>
        <TextBlock Text="{Binding TypeName}"/>
        <TextBlock Text="{Binding Director}"/>
        <TextBlock Text="{Binding Phone}"/>
    </StackPanel>

```

```

        <TextBlock Text="{Binding Rating, StringFormat={}}Рейтинг:
{0}"/>
    </StackPanel>

    <TextBlock Text="{Binding Discount, StringFormat={}}{0}%"
Grid.Column="1"/>
</Grid>

```



StringFormat полезен для удобного представления. Например:
StringFormat={}Рейтинг: {0}, при Rating равном 5 вернет *Рейтинг: 5*

6. Чтобы указать, откуда **ListView** должен получить элементы, укажем список партнеров в его свойстве `ItemsSource` – сделаем это с помощью доп. метода `UpdateListView()` (это понадобится позже):

```

using System.Linq;
using System.Windows;

namespace Demo
{
    public partial class MainWindow
    {
        public MainWindow()
        {
            InitializeComponent();
            UpdateListView();
        }

        private void UpdateListView()
        => PartnersListView.ItemsSource = App.Context.Partners
            .ToList();
    }
}

```

7. Вот, как на данный момент выглядит приложение.

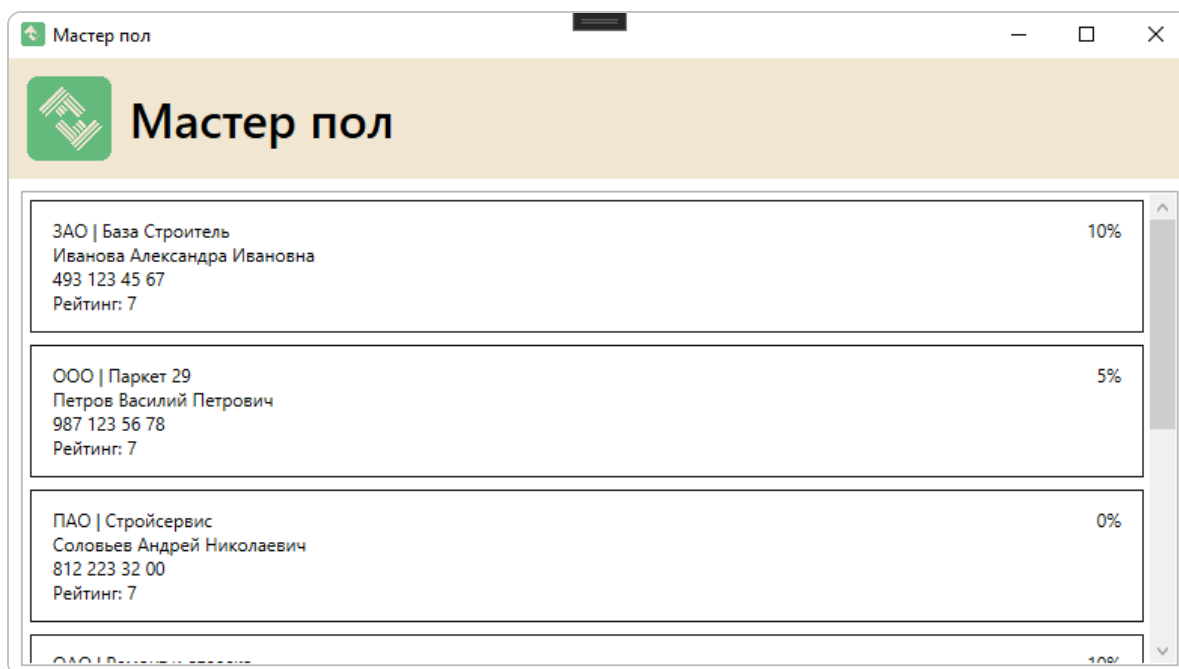


Рисунок 16 – Отображение списка партнеров в приложении

8. Немного модернизируем **Border** и создадим кнопку для добавления партнера:

```
<Border Background="{DynamicResource SecondaryColor}">
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition/>
      <ColumnDefinition Width="Auto"/>
    </Grid.ColumnDefinitions>

    <StackPanel Margin="12"
      Orientation="Horizontal">
      <Image Margin="0,0,12,0"
        Source="/Мастер пол.png"/>
      <TextBlock FontSize="32"
        Text="Мастер пол"
        FontWeight="SemiBold"
        VerticalAlignment="Center"/>
    </StackPanel>
  </Grid>
</Border>
```

```

        <Button Margin="8"
            FontSize="18"
            Padding="20,8"
            Grid.Column="1"
            x:Name="AddButton"
            Click="AddButton_Click"
            VerticalAlignment="Center"
            Content="Добавление партнера"/>
    </Grid>
</Border>

```

9. Для Редактирования, Просмотра историй реализаций и удаления создадим контекстное меню для **ListViewItem**:

```

<Grid Margin="10" HorizontalAlignment="Stretch">
    <Grid.ContextMenu>
        <ContextMenu>
            <MenuItem x:Name="Edit"
                Header="Изменить"
                Click="Edit_Click"/>
            <MenuItem Header="История"
                x:Name="History"
                Click="History_Click"/>
            <MenuItem x:Name="Delete"
                Header="Удалить"
                Foreground="Red"
                Click="Delete_Click"/>
        </ContextMenu>
    </Grid.ContextMenu>

    <!--...-->
</Grid>

```

Окно создания/редактирования партнера

1. Создадим новое окно с названием **AddEditWindow**.
2. В обработчике окна **MainWindow.AddButton_Click** пропишем логику открытия окна и обновления списка партнеров:

```
private void AddButton_Click(object sender, RoutedEventArgs e)
{
    new AddEditWindow().ShowDialog();
    UpdateListView();
}
```

4. Доработаем конструктор **AddEditWindow**, чтобы тот принимал в качестве аргумента необязательный параметр *Partner partner*.

```
using Demo.Data;

namespace Demo
{
    public partial class AddEditWindow
    {
        private Partner _partner;
        private readonly string[] _types = { "ПАО", "ООО", "ЗАО", "ОАО" };

        public AddEditWindow(Partner partner = null)
        {
            InitializeComponent();
            TypeComboBox.ItemsSource = _types;
            _partner = partner ?? new Partner { Type = _types[0] };
            DataContext = _partner;
        }
    }
}
```

5. Для **EditButton_Click** практически то же самое, только в качестве параметра

передаем партнера, а для **Delete_Click** добавим дополнительное подтверждение:

```
private void Edit_Click(object sender, RoutedEventArgs e)
{
    if (sender is MenuItem mi && mi.DataContext is Partner partner)
    {
        new AddEditWindow(partner).ShowDialog();
        UpdateListView();
    }
}

private void Delete_Click(object sender, RoutedEventArgs e)
{
    if (sender is MenuItem mi && mi.DataContext is Partner partner
        && MessageBox.Show("Вы уверены?", "Подтверждение",
        MessageBoxButton.YesNo, MessageBoxImage.Question) is
        MessageBoxResult.Yes)
    {
        App.Context.Partners.Remove(partner);
        App.Context.SaveChanges();
        UpdateListView();
    }
}
```

6. Сделаем разметку для **AddEditWindow** с помощью *Grid*:

```
<Window x:Class="Demo.AddEditWindow"

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"
    mc:Ignorable="d"
    Title="AddEdit" Height="450" Width="800">
    <Grid>
```

```
<Grid.RowDefinitions>
    <RowDefinition Height="80"/>
    <RowDefinition/>
</Grid.RowDefinitions>

<Border Background="{DynamicResource SecondaryColor}">
    <StackPanel Margin="12"
        Orientation="Horizontal">
        <Image Margin="0,0,12,0"
            Source="/Мастер пол.png"/>
        <TextBlock FontSize="32"
            Text="Мастер пол"
            FontWeight="SemiBold"
            VerticalAlignment="Center"/>
    </StackPanel>
</Border>

<Grid Grid.Row="1"
    Margin="15">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="160"/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>

    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>

    <TextBlock Text="Наименование"/>
```



```

<TextBox Grid.Column="1"
    Text="{Binding Title}"
    x:Name="TitleBox"
    MaxLength="100"
    ToolTip="Введите наименование партнёра"/>

<TextBlock Text="Тип партнера"
    Grid.Row="1"/>
<ComboBox x:Name="TypeComboBox"
    Text="{Binding Type}"
    Grid.Row="1" Grid.Column="1"
    ToolTip="Выберите тип партнёра"/>

<TextBlock Text="Рейтинг компании"
    Grid.Row="2"/>
<TextBox Text="{Binding Rating}" MaxLength="3"
    Grid.Row="2" Grid.Column="1"
    x:Name="RatingBox"
    ToolTip="Введите сюда рейтинг партнёра в формате
XX"/>

<TextBlock Text="Адрес компании"
    Grid.Row="3"/>
<TextBox Text="{Binding LegalAddress}"
    MaxLength="255"
    Grid.Row="3"
    Grid.Column="1"
    x:Name="AddressBox"
    ToolTip="Введите сюда адрес партнёра"/>

<TextBlock Text="ФИО Директора"
    Grid.Row="4"/>
<TextBox Text="{Binding Director}"
    MaxLength="255"
    Grid.Row="4"
    Grid.Column="1"
    x:Name="DirectorBox"
    ToolTip="Введите сюда ФИО директора"/>

```

```

<TextBlock Text="Телефон компании"
            Grid.Row="5"/>
<TextBox Text="{Binding Phone}" MaxLength="13"
          Grid.Row="5" Grid.Column="1"
          x:Name="PhoneBox"
          ToolTip="Введите сюда номер телефона партнёра в
формате 000 000 00 00"/>

<TextBlock Text="Почта компании"
            Grid.Row="6"/>
<TextBox Text="{Binding Email}" MaxLength="255"
          Grid.Row="6" Grid.Column="1"
          x:Name="EmailBox"
          ToolTip="Введите сюда адрес электронной почты
партнёра"/>

<TextBlock Text="ИНН"
            Grid.Row="7"/>
<TextBox Text="{Binding INN}" MaxLength="10"
          Grid.Row="7" Grid.Column="1"
          x:Name="InnBox"
          ToolTip="Введите сюда ИНН партнёра в формате
XXXXXXXXXX"/>

<StackPanel Grid.Row="9"
            Grid.ColumnSpan="2"
            Orientation="Horizontal"
            HorizontalAlignment="Right"
            Margin="0,8">
    <Button x:Name="DeleteBtn"
            Content="Очистить"
            Visibility="Visible"
            Margin="0,0,10,0"
            Click="DeleteBtn_Click"/>
    <Button x:Name="SaveBtn"
            Content="Сохранить"
            Click="SaveBtn_Click"/>

```

```

        <Button x:Name="backBtn"
            Content="Назад"
            Margin="10,0,0,0"
            Click="BackBtn_Click"/>
    </StackPanel>
</Grid>
</Grid>
</Window>

```

7. Добавим реализацию закрытия и очищения в **AddEditWindow**:

```

private void DeleteBtn_Click(object sender, RoutedEventArgs e)
    => DataContext = _partner = new Partner
    {
        ID = _partner.ID,
        Type = _types[0]
    };

private void BackBtn_Click(object sender, RoutedEventArgs e)
    => Close();

```

8. Для добавления/редактирования мы сначала совершим валидацию данных, а потом будем использовать ну очень удобный метод под названием **AddOrUpdate()**:

```

private void SaveBtn_Click(object sender, RoutedEventArgs e)
{
    var message = ValidatePartner();

    if (!string.IsNullOrEmpty(message))
    {
        MessageBox.Show(message, "Ошибка", MessageBoxButton.OK,
        MessageBoxImage.Error);
        return;
    }
}

```

```

        if (MessageBox.Show("Вы уверены, что указали все, что хотели?",
            "Подтверждение", MessageBoxButton.YesNo,
            MessageBoxImage.Question) is MessageBoxResult.Yes)
        {
            App.Context.Partners.AddOrUpdate(_partner);
            App.Context.SaveChanges();
            Close();
        }
    }

    private string ValidatePartner()
    {
        try
        {
            if (string.IsNullOrEmpty(_partner.Title))
                throw new InvalidOperationException("Введите название
партнера");

            if (string.IsNullOrEmpty(_partner.LegalAddress))
                throw new InvalidOperationException("Введите адрес
партнёра");

            if (string.IsNullOrEmpty(_partner.Director))
                throw new InvalidOperationException("Введите ФИО
директора");

            if (string.IsNullOrEmpty(_partner.Phone))
                throw new InvalidOperationException("Введите телефон
партнёра");

            if (string.IsNullOrEmpty(_partner.Email))
                throw new InvalidOperationException("Введите почту
партнёра");

            if (string.IsNullOrEmpty(_partner.INN))
                throw new InvalidOperationException("Введите ИНН партнёра");

```

```

        if (_partner.Director.Length < 4)
            throw new InvalidOperationException("ФИО директора слишком
Короткое");

        if (_partner.Phone.Length != 13)
            throw new InvalidOperationException("Телефон партнера
слишком длинный");

        // Проверка что номер телефона не содержит букв
        if (_partner.Phone.Any(char.IsLetter))
            throw new InvalidOperationException("Телефон партнера не
должен содержать буквы");

        // Проверка что email содержит символ @
        if (!_partner.Email.Contains("@"))
            throw new InvalidOperationException("Email введен неверно");

        // Проверка что ИНН имеет необходимое количество цифр
        if (_partner.INN.Length != 10)
            throw new InvalidOperationException("ИНН партнера введен
неверно");

        // Проверка что ИНН является числом
        if (_partner.INN.Any(char.IsLetter))
            throw new InvalidOperationException("ИНН партнера должен
содержать только цифры");
    }
    catch (InvalidOperationException ioe)
    {
        return ioe.Message;
    }

    return string.Empty;
}

```

9. Вот как выглядит это окно:

Добавление и редактирование партнеров

Мастер пол

Наименование	<input type="text" value="База Строитель"/>
Тип партнера	<input type="text" value="ЗАО"/>
Рейтинг компании	<input type="text" value="7"/>
Адрес компании	<input type="text" value="652050, Кемеровская область, город Юрга, ул. Лесная, 15"/>
ФИО Директора	<input type="text" value="Иванова Александра Ивановна"/>
Телефон компании	<input type="text" value="493 123 45 67"/>
Почта компании	<input type="text" value="aleksandraivanova@ml.ru"/>
ИНН	<input type="text" value="2222455179"/>

Рисунок 16 – Добавление и редактирование партнеров

Окно истории реализаций продукции

1. Создадим новое окно **ProductSalesHistoryWindow**.
2. Изменим его конструктор на следующий, чтобы он мог принимать партнера:

```
using Demo.Data;  
using System.Collections.Generic;  
using System.Linq;  
using System.Windows;
```

```

namespace Demo
{
    public partial class ProductSalesHistoryWindow
    {
        private readonly Partner _partner;
        private readonly List<PartnerProduct> _deals;

        public ProductSalesHistoryWindow(Partner partner)
        {
            InitializeComponent();
            _partner = partner;
            _deals = App.Context.PartnerProducts
                .Where(pp => pp.PartnerID == _partner.ID)
                .ToList();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            if (!_deals.Any())
            {
                MessageBox.Show("Реализации у партнера отсутствуют",
                    "Сообщение", MessageBoxButton.OK, MessageBoxImage.Information);
                Close();
            }

            ListHistoryProductsSales.ItemsSource = _deals;
        }

        private void BackBtn_Click(object sender, RoutedEventArgs e)
            => Close();
    }
}

```

3. Создадим разметку с помощью **ListView**:

```

<Window x:Class="Demo.ProductSalesHistoryWindow"

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"
    mc:Ignorable="d"
    Loaded="Window_Loaded"
    Title="История реализации продукции"
    WindowStartupLocation="CenterScreen"
    Height="400" Width="800" MinHeight="400" MinWidth="640">

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="80"/>
            <RowDefinition/>
        </Grid.RowDefinitions>

        <Border Background="{DynamicResource SecondaryColor}">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition/>
                    <ColumnDefinition Width="Auto"/>
                </Grid.ColumnDefinitions>

                <StackPanel Margin="12"
                    Orientation="Horizontal">
                    <Image Margin="0,0,12,0"
                        Source="/Мастер пол.png"/>
                    <TextBlock FontSize="32"
                        Text="Мастер пол"
                        FontWeight="SemiBold"
                        VerticalAlignment="Center"/>
                </StackPanel>

                <Button Margin="20"

```



```

        Content="Назад"
        Grid.Column="1"
        x:Name="backBtn"
        Click="BackBtn_Click"/>
    </Grid>
</Border>

<ListView x:Name="ListHistoryProductsSales" Grid.Row="1"
    Width="{Binding ActualWidth, RelativeSource=
{RelativeSource AncestorType=Grid}}"
    d:ItemsSource="{d:SampleData ItemCount=5}">
    <ListView.ItemContainerStyle>
        <Style TargetType="ListViewItem">
            <Setter Property="HorizontalContentAlignment"
Value="Stretch"/>
            <Setter Property="Margin" Value="10,5"/>
        </Style>
    </ListView.ItemContainerStyle>
    <ListView.ItemTemplate>
        <DataTemplate>

            </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
</Grid>
</Window>

```

4. В **MainWindow** реализуем открытие этого окна:

```

private void History_Click(object sender, RoutedEventArgs e)
{
    if (sender is MenuItem mi && mi.DataContext is Partner partner)
        new ProductSalesHistoryWindow(partner).ShowDialog();
}

```

5. Создадим шаблон **ListViewItem** для истории реализаций:

```
<DataTemplate>
    <Border BorderThickness="1"
        Background="{DynamicResource AccentColor}"
        BorderBrush="{DynamicResource SecondaryColor}">
        <Grid Margin="10"
            HorizontalAlignment="Stretch"
            VerticalAlignment="Center">

            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="2*"/>
                <ColumnDefinition Width="Auto"/>
                <ColumnDefinition Width="Auto"/>
            </Grid.ColumnDefinitions>

            <Grid.Resources>
                <Style TargetType="TextBlock">
                    <Setter Property="Margin" Value="10,0,0,0"/>
                    <Setter Property="FontStyle" Value="Italic"/>
                    <Setter Property="Foreground" Value="{DynamicResource LightTextColor}"/>
                </Style>
            </Grid.Resources>

            <TextBlock Margin="0"
                FontSize="18"
                TextWrapping="Wrap"
                Text="{Binding Product.Title}"/>
            <TextBlock Grid.Column="1"
                HorizontalAlignment="Center"
                Text="{Binding CountProduct, StringFormat={}{}{0} шт.}"
                TextWrapping="Wrap"/>
            <TextBlock Grid.Column="2"
                HorizontalAlignment="Center"
                Text="{Binding DateSale, StringFormat={}{}{0:d MMMM
```

```
yyyy}}"/>
    </Grid>
  </Border>
</DataTemplate>
```

⚠ Здесь для удобства используем StringFormat, чтобы дата была в удобном формате: 7 Декабря 2024

6. Вот как выглядит это окно:

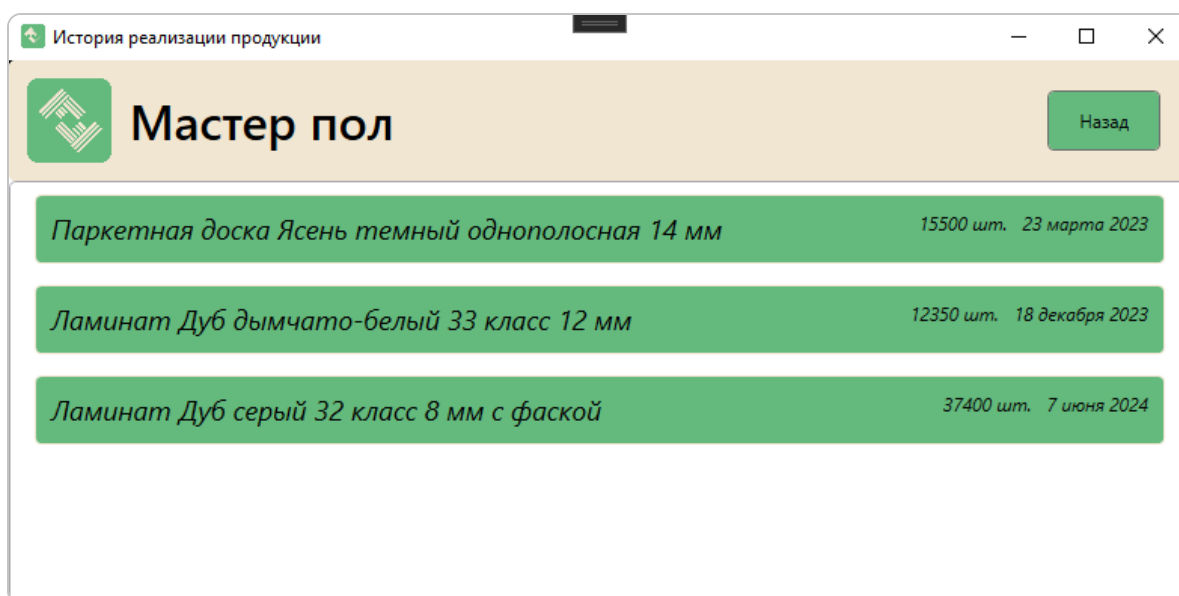


Рисунок 17 – История реализации продукции

Бизнес-логика

Расчет количества материала

Чтобы получить количество материала на 1 еб. товара нужно:

- Принять входные параметры: ID продукта и материала.
- В случае исключений и несоответствий возвращать -1.
- Рассчитать по формуле: Длина * Ширина * Коэффициент типа продукта * (1 + процент дефекта)

- Вернуть результат, округленный в большую сторону.

1. Создадим папку **Business**.
2. Создадим внутри нее класс **MaterialModule**.
3. Добавим в него следующий метод:

```
public static int CalculateMaterialCount(
    int productId,
    int materialTypeId,
    int productsCount,
    double length,
    double width)
{
    var productType = App.Context.ProductTypes.SingleOrDefault(
        pt => pt.ID == productId);
    var materialType = App.Context.MaterialTypes.SingleOrDefault(
        mt => mt.ID == materialTypeId);

    if (productType is null || materialType is null
        || productsCount <= 0 || length <= 0 || width <= 0)
        return -1;

    var productTypeCoefficient = (double)productType.Ratio;
    var scrapRate = (double)materialType.ScrapRate;

    if (productTypeCoefficient <= 0 || scrapRate <= 0)
        return -1;

    return (int)Math.Ceiling(length * width * productTypeCoefficient
        * (1 + scrapRate) * productsCount);
}
```

⚠ По ID мы ищем **ProductType** и **MaterialType**: если ничего не найдено, либо параметры метода ≤ 0 , возвращаем -1. Следом мы получаем **ProductType.Ratio** и **MaterialType.ScrapRate**: если хоть что-то ≤ 0 ,

возвращаем -1. Возвращаем округленный в большую сторону (с помощью Math.Ceiling) и приведенный к целочисленному типу результат формулы.