

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ**
**федеральное государственное бюджетное образовательное учреждение высшего
образования «Российский экономический университет имени Г.В. Плеханова»**
Московский приборостроительный техникум

ОТЧЕТ

по учебной практике

УП 01.01 «Прикладное программирование»
индекс по УП и наименование практики

Профессионального модуля ПМ.01 «Разработка модулей программного
обеспечения для компьютерных систем»
индекс по УП и наименование профессионального модуля

Специальность 09.02.07 «Информационные системы и программирование»
код и наименование специальности

Студент Ахвердиева Самира Ниязиевна

Группа П50-6-22

Руководитель по практической подготовке от техникума
Мысев Дмитрий Владимирович

«_____» _____ 2024 года

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	2
ЛАБОРАТОРНАЯ РАБОТА №1	3
Подключение базы данных. Чтение данных из БД	3
ЛАБОРАТОРНАЯ РАБОТА №2	14
Добавление, изменение, удаление данных в БД	14
ЛАБОРАТОРНАЯ РАБОТА №3	27
Отображение нескольких таблиц из БД	27
ЛАБОРАТОРНАЯ РАБОТА №4	33
Поиск и фильтрация данных	33
ЛАБОРАТОРНАЯ РАБОТА №5	41
Разработка информационной системы	41

ЛАБОРАТОРНАЯ РАБОТА №1

Подключение базы данных. Чтение данных из БД

Цель работы: реализовать базу данных из двух-трех таблиц, связанных между собой, и подключить к ней WPF проект, используя EntityFramework и DataSet, а также добавить окна или страницы для разных таблиц.

Создание базы данных.

Создаём базу данных с предметной областью «Клиника» и используем её. Далее создаём 3 таблицы «Doctors», «Patients» и «Appointments». В таблице «Appointments» используются внешние ключи для связи с таблицами «Patients» и «Doctors».

Doctors (Врачи):

ID_Doctor: это первичный ключ для таблицы.

DoctorSurname: Фамилия врача (не может быть пустой).

DoctorName: Имя врача (не может быть пустым).

DoctorMiddleName: Отчество врача (может быть пустым).

Specialization: Специализация врача (не может быть пустой).

Patients (Пациенты):

ID_Patient: Первичный ключ таблицы.

PatientSurname: Фамилия пациента (не может быть пустой).

PatientName: Имя пациента (не может быть пустым).

PatientMiddleName: Отчество пациента (может быть пустым).

Gender: Пол пациента (не может быть пустым).

DateOfBirth: Дата рождения пациента (не может быть пустой).

Appointments (Приемы):

ID_Appointment: Первичный ключ таблицы.

Patient_ID: Внешний ключ, связывающий с таблицей Patients по ID_Patient.

Doctor_ID: Внешний ключ, связывающий с таблицей Doctors по ID_Doctor.

DateAppointment: Дата и время приема (не может быть пустым).

DescriptionAppointment: Описание приема (не может быть пустым).

```
create database Clinic2;
go
use Clinic2;
go

create table Doctors (
    ID_Doctor int primary key identity(1,1),
    DoctorSurname varchar(50) not null,
    DoctorName varchar(50) not null,
    DoctorMiddleName varchar(50),
    Specialization varchar(100) not null
);
go

create table Patients (
    ID_Patient int primary key identity(1,1),
    PatientSurname varchar(50) not null,
    PatientName varchar(50) not null,
    PatientMiddleName varchar(50),
    Gender varchar(1) not null,
    DateOfBirth date not null
);
go

create table Appointments (
    ID_Appointment int primary key identity(1,1),
    Patient_ID int not null,
    Doctor_ID int not null,
    DateAppointment datetime not null,
    DescriptionAppointment text not null,
    foreign key (Patient_ID) references Patients(ID_Patient),
    foreign key (Doctor_ID) references Doctors(ID_Doctor)
);
go
```

Рисунок 1 – БД

Заполняем таблицы данными с помощью оператора insert и создаём представление.

```
insert into Patients (PatientSurname, PatientName, PatientMiddleName, DateOfBirth, Gender) values
('Кузнецова', 'Анастасия', 'Андреевна', '21.08.2004', 'Ж'),
('Смирнов', 'Егор', 'Романович', '29.11.1973', 'М'),
('Крылов', 'Даниил', 'Александрович', '05.08.1976', 'М'),
('Антонов', 'Август', 'Всеволодович', '31.08.2003', 'М'),
('Родионов', 'Аркадий', 'Федотович', '21.08.2002', 'М');
go

insert into Doctors (DoctorSurname, DoctorName, DoctorMiddleName, Specialization) values
('Путятин', 'Алексей', 'Владимирович', 'Терапевт'),
('Заперенчук', 'Илья', 'Сергеевич', 'Хирург'),
('Березина', 'Ксения', '', 'Окулист');
go

insert into Appointments (Patient_ID, Doctor_ID, DateAppointment, DescriptionAppointment) values
(1, 1, '15.03.2024 09:00', 'Общий осмотр'),
(2, 2, '16.03.2024 10:30', 'Плановая операция'),
(3, 3, '17.03.2024 15:45', 'Проверка зрения');
go

create view DoctorsView2 as
select Doctors.DoctorSurname + ' ' + SUBSTRING(Doctors.DoctorName, 1, 1) + '.' + SUBSTRING(Doctors.DoctorMiddleName, 1, 1) + '.' as 'ФИО',
    Doctors.Specialization as 'Специализация'
from Doctors
go
```

Рисунок 2 - БД

DataSet - подключение базы данных к WPF проекту.

Создаём проект «Приложение WPF (.NET Framework)» и подключаем базу данных SQL в проект.

Для того, чтобы подключиться к БД, нужно в Visual Studio выбрать «Средства» -> «Подключиться к базе данных». В окне «Сменить источник данных», нужно выбрать «Microsoft SQL Server». Далее появляется окно, где необходимо указать имя сервера. Его можно взять из MSSQL, в окне соединения с сервером (самое стартовое). В конце выбираем необходимую БД. После этого и наше подключение будет сохранено в «Обозревателе серверов».

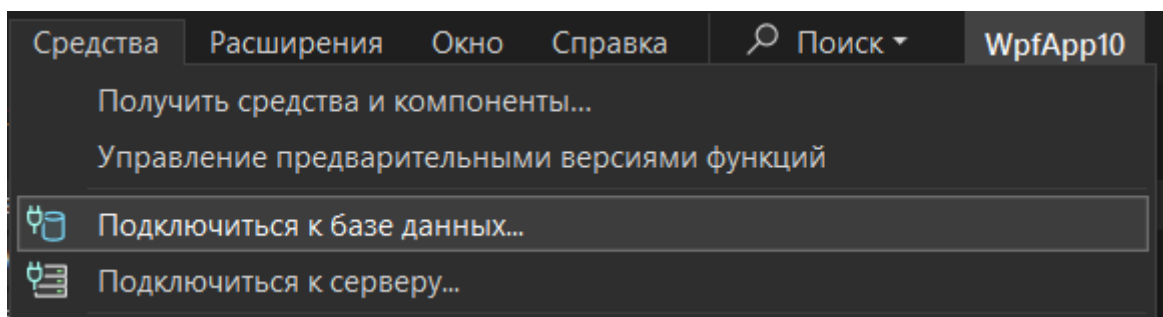


Рисунок 3 - Подключение к бд

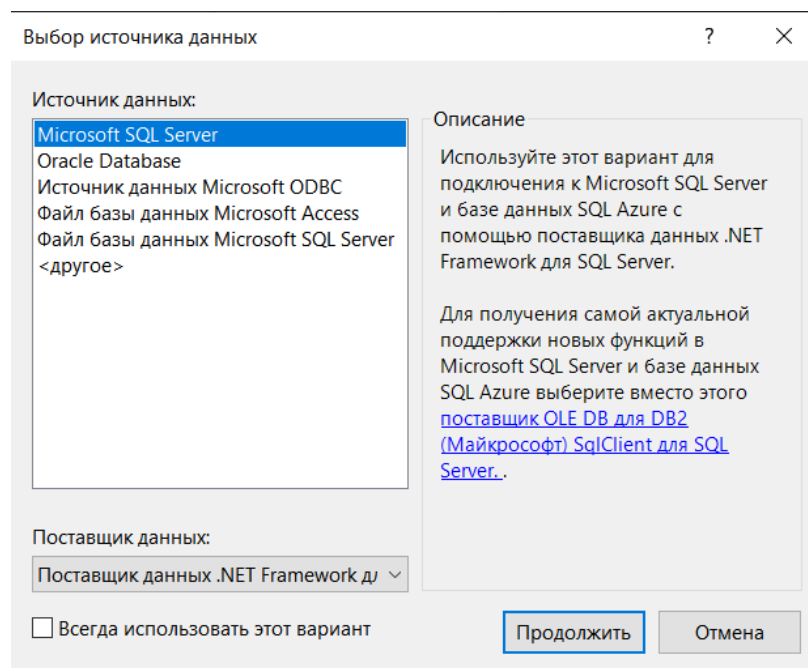


Рисунок 4 - Подключение к бд

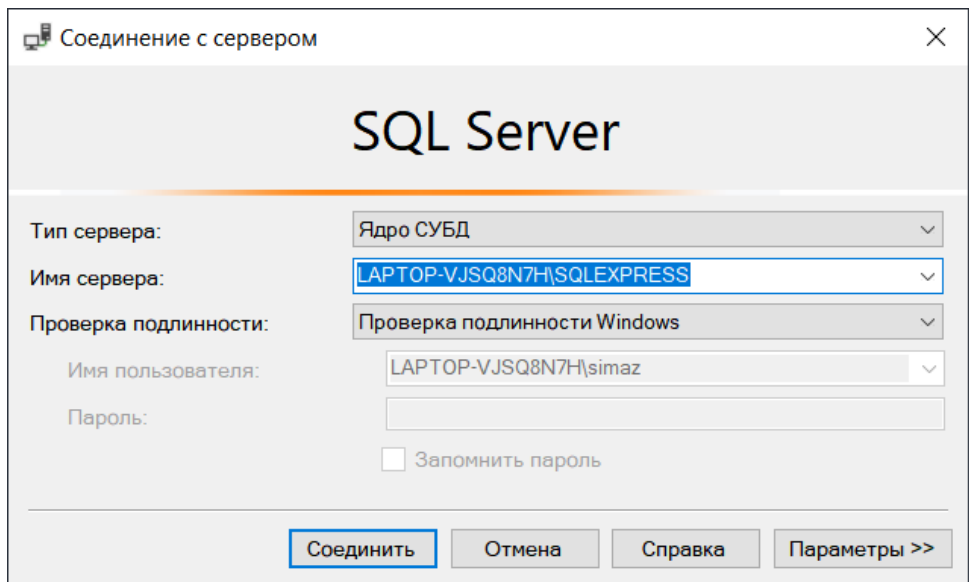


Рисунок 5 - Имя сервера

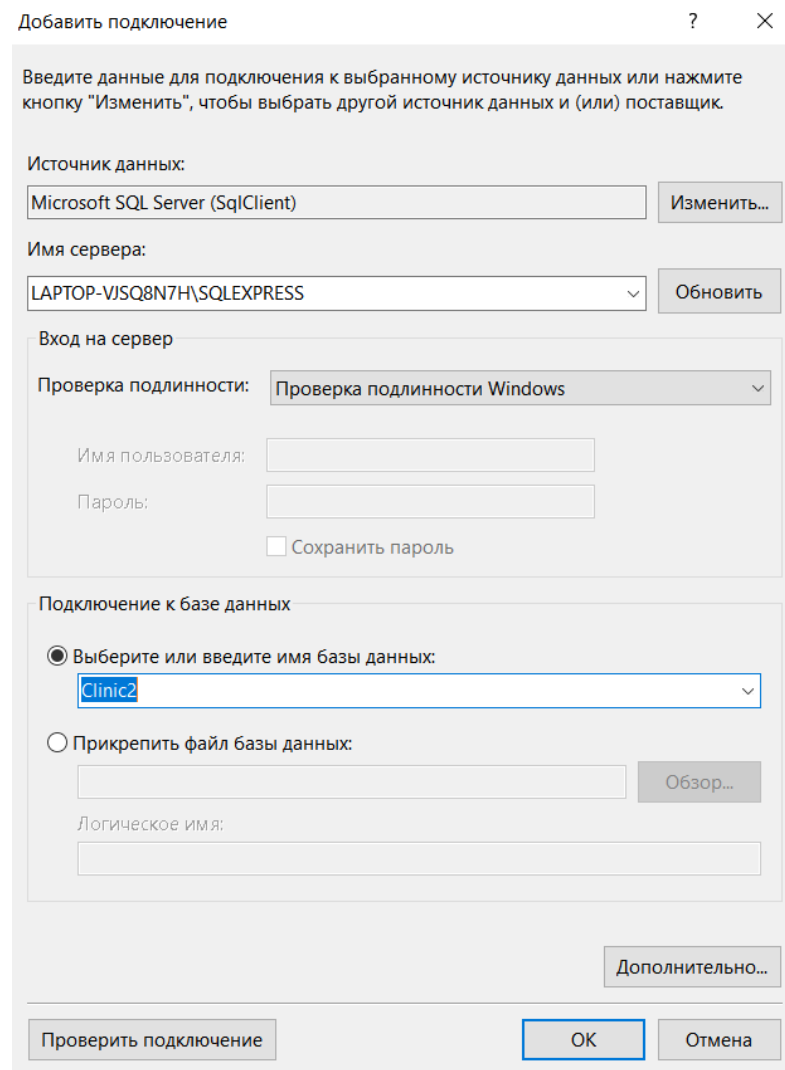


Рисунок 6 - Подключение к бд

Для создания объектов, при помощи которого мы в коде будем работать с бд – набор данных (DataSet). Открываем «Проект» -> «Добавить

НОВЫЙ ИСТОЧНИК ДАННЫХ».

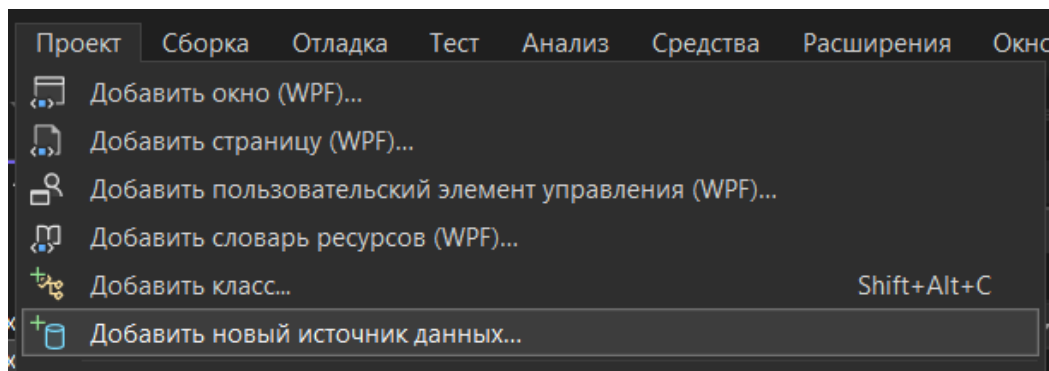


Рисунок 7 - Создание объектов

Перед нами откроется мастер настройки источника данных». Открывается окно, там выбираем, что источник данных – БД и нажимаем далее. При выборе объектов базы данных выбираем все то, с чем мы хотим работать. В нашем случае – это каждая таблица внутри БД. Нажимаем «Готово» и наш набор данных создастся.

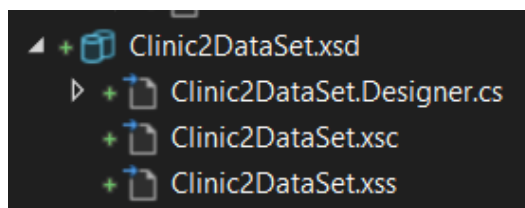


Рисунок 8 - DataSet

EF - подключение базы данных к WPF проекту.

Продельываем тоже самое, что и в DataSet до тех пор, пока наше подключение будет сохранено в «Обозревателе серверов».

Для создания объектов, при помощи которого мы в коде будем работать с бд – Модель ADO.NET. Создается она как обычный файл внутри проекта, через ПКМ по проекту -> «Добавить» -> «Создать элемент».

Во вкладке «Данные» выбираем необходимую модель и даем название.

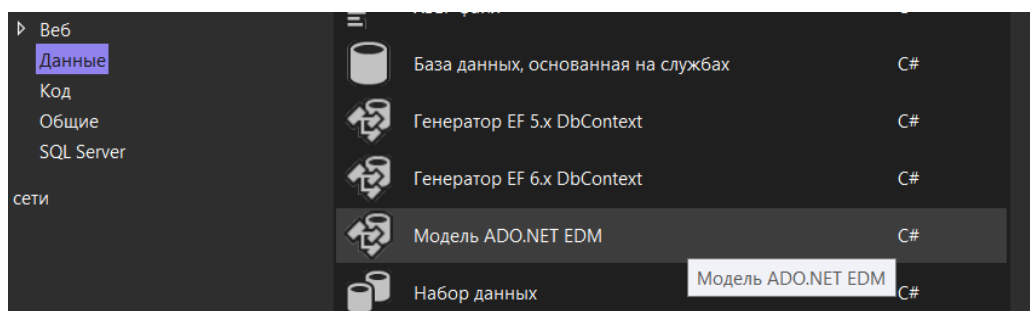


Рисунок 9 - Выбор модели

В появляющемся окне выбираем конструктор, затем выбираем то самое подключение, которое создавали ранее.

Далее выбираем все что нужно – таблички и представления.

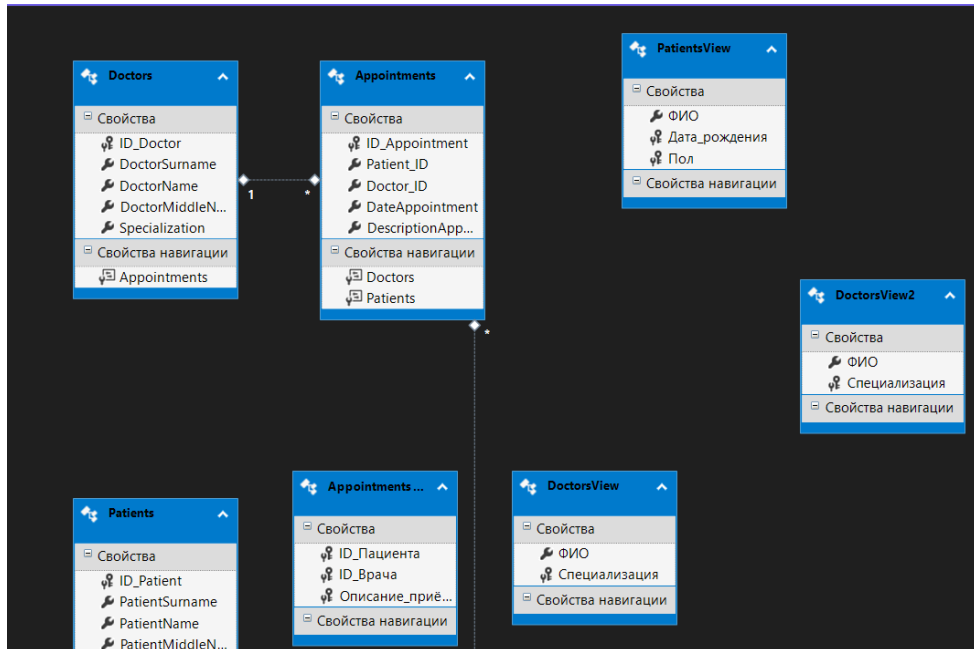


Рисунок 10 - EF

Создание кнопок для перехода на другие окна.

В основном хамл создаём 3 кнопки: «Просмотр пациентов», «Просмотр врачей», «Просмотр приемов».

```
ms:Ignore="0"
Title="Клиника" Height="250" Width="300">
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>

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

  <Button x:Name="Patients" Content="Просмотр пациентов" Click="Patients_Click" Width="200px" Height="50px" />
  <Button x:Name="Doctors" Content="Просмотр врачей" Click="Doctors_Click" Width="200px" Height="50px" />
  <Button x:Name="Appointments" Content="Просмотр приемов" Click="Appointments_Click" Width="200px" Height="50px" />
</Grid>
```

Рисунок 11 - Создание кнопок

В основном cs пишем логику для этих кнопок, чтобы при нажатии открывалось новое окно.


```

namespace WpfApp11
{
    // Ссылка: 2
    public partial class MainWindow : Window
    {
        // Ссылка: 0
        public MainWindow()
        {
            InitializeComponent();
        }

        // Ссылка: 1
        private void Patients_Click(object sender, RoutedEventArgs e)
        {
            PatientsW window1 = new PatientsW();
            window1.ShowDialog();
        }

        // Ссылка: 1
        private void Doctors_Click(object sender, RoutedEventArgs e)
        {
            DoctorsW window2 = new DoctorsW();
            window2.ShowDialog();
        }

        // Ссылка: 1
        private void Appointments_Click(object sender, RoutedEventArgs e)
        {
            AppointmentsW window3 = new AppointmentsW();
            window3.ShowDialog();
        }
    }
}

```

Рисунок 12 – Класс MainWinidow.xaml.cs

Чтение данных из бд для DataSet.

Создадим в xaml таблицу, куда мы будем выгружать все данные. В данном случае, DataGrid настроен таким образом, что автоматическое генерирование столбцов отключено (AutoGenerateColumns="False"), и столбцы указаны в разделе DataGrid.Columns. Это сделано для отображения названий столбцов на русском языке.

```

<Grid
    Title="Пациенты" Height="450" Width="800">
    <DataGrid AutoGenerateColumns="False" x:Name="PatientsView2">
        <DataGrid.Columns>
            <DataGridTextColumn Header="Фамилия" Binding="{Binding PatientSurname}" />
            <DataGridTextColumn Header="Имя" Binding="{Binding PatientName}" />
            <DataGridTextColumn Header="Отчество" Binding="{Binding PatientMiddleName}" />
            <DataGridTextColumn Header="Дата рождения" Binding="{Binding DateOfBirth, StringFormat={}{0:dd/MM/yyyy}}" />
            <DataGridTextColumn Header="Пол" Binding="{Binding Gender}" />
        </DataGrid.Columns>
    </DataGrid>
</Grid>

```

Рисунок 13 - Окно с информацией о пациентах

```

<Grid>
  <DataGrid AutoGenerateColumns="False" x:Name="AppointmentsView">
    <DataGrid.Columns>
      <DataGridTextColumn Header="ID Пациента" Binding="{Binding Patient_ID}" />
      <DataGridTextColumn Header="ID Врача" Binding="{Binding Doctor_ID}" />
      <DataGridTextColumn Header="Дата приема" Binding="{Binding DateAppointment, StringFormat={}{0:dd/MM/yyyy HH:mm:ss}}"/>
      <DataGridTextColumn Header="Описание" Binding="{Binding DescriptionAppointment}" />
    </DataGrid.Columns>
  </DataGrid>
</Grid>
</Window>

```

Рисунок 14 - Окно с информацией о приемах

Далее в cs пишем:

`PatientsView2.ItemsSource = null;` Устанавливается источник данных элемента управления `PatientsView2` в значение `null`. Это делается для очистки данных, которые могли быть связаны с этим элементом ранее.

`PatientsView2.Items.Clear();` Очищается коллекция элементов, отображаемых в `PatientsView2`.

`PatientsView2.ItemsSource = patient.GetData();` Устанавливается источник данных элемента управления `PatientsView2` в результат выполнения метода `GetData()` адаптера `patient`. Метод `GetData()` возвращает набор, содержащий информацию о приемах и пациентах, из базы данных. Эти данные затем отображаются в `DataGrid PatientsView2`.

```

using System.Windows.Shapes;
using WpfApp11.Clinic2DataSetTableAdapters;

namespace WpfApp11
{
    Ссылка: 4
    public partial class PatientsW : Window
    {
        PatientsTableAdapter patient = new PatientsTableAdapter();
        Ссылка: 1
        public PatientsW()
        {
            InitializeComponent();
            PatientsView2.ItemsSource = null;
            PatientsView2.Items.Clear();
            PatientsView2.ItemsSource = patient.GetData();
        }
    }
}

```

Рисунок 15 - Класс PatientW

```

using System.Windows.Shapes;
using WpfApp11.Clinic2DataSetTableAdapters;

namespace WpfApp11
{
    Ссылка 4
    public partial class AppointmentsW : Window
    {
        AppointmentsTableAdapter priems = new AppointmentsTableAdapter();
        Ссылка 1
        public AppointmentsW()
        {
            InitializeComponent();
            AppointmentsView.ItemsSource = null;
            AppointmentsView.Items.Clear();
            AppointmentsView.ItemsSource = priems.GetData();
        }
    }
}

```

Рисунок 16 - Класс AppointmentsW

Чтение данных из бд для EW.

Для того, чтобы подтянуть данные из таблицы, возьмем представление из бд: «DoctorsView2». Для ее отображения, тоже сделаем представление в xaml, назвав ее «DoctorsView2».

```

mc:Ignorable="d"
Title="Врачи" Height="450" Width="800">
<Grid>
    <DataGrid x:Name="DoctorsView2"/>
</Grid>
</Window>

```

Рисунок 17 - Xaml с информацией о врачах

Чтобы подтянуть все данные из таблицы, нужна одна переменная, которая условно будет являться базой данных. Из нее я могу подтянуть любую табличку, представить ее как коллекцию, и уже оттуда брать данные. Делается это вот так: создается переменная типа данных <название бд>Entities и название, в данном случае context

Через эту переменную context берется представление context.<DoctorsView>. Представление преобразовывается в лист и пихается в датагрид для отображения.

```

namespace WpfApp1
{
    Ссылка: 4
    public partial class DoctorsW : Window
    {
        private Clinic2Entities context = new Clinic2Entities();
        Ссылка: 1
        public DoctorsW()
        {
            InitializeComponent();
            DoctorsView2.ItemsSource = context.DoctorsView2.ToList();
        }
    }
}

```

Рисунок 18 – класс DoctorsW.xaml.cs

Запускаем программу.



Рисунок 19 - Результат вывода окна с пациентами

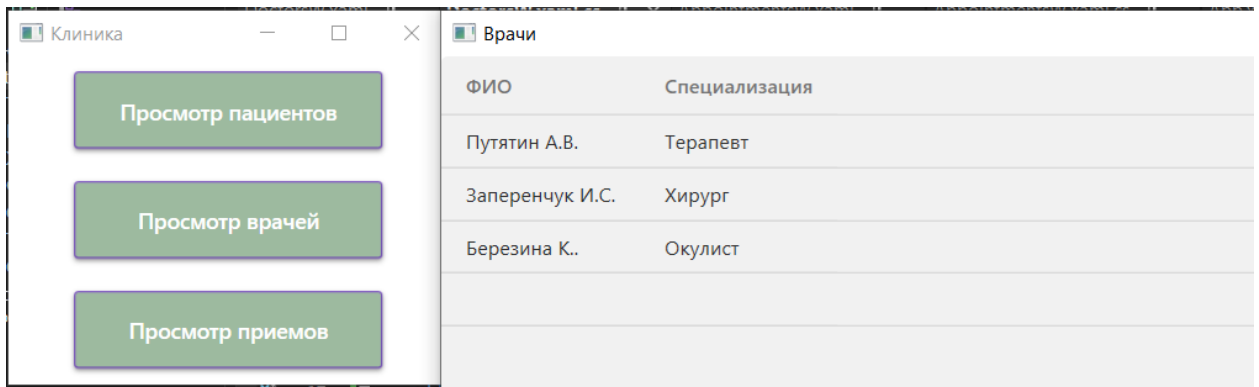


Рисунок 20 - Результат вывода окна с врачами

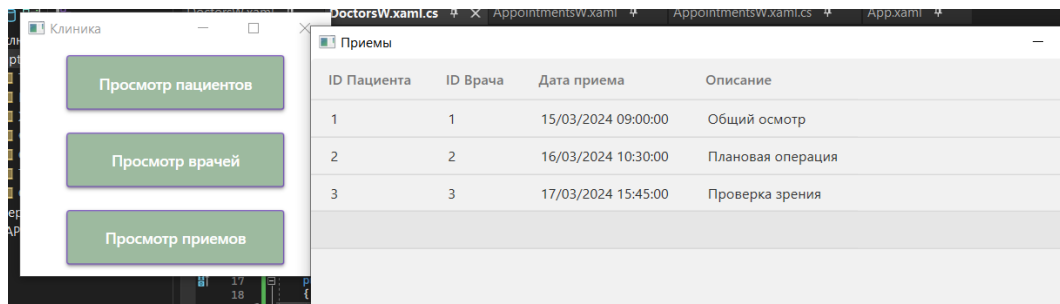


Рисунок 21 - Результат вывода окна с приемами

Вывод: реализовали базу данных из трех таблиц, связанных между собой, и подключили к ней WPF проект, используя EntityFramework и DataSet, а также добавили окна для разных таблиц.

ЛАБОРАТОРНАЯ РАБОТА №2

Добавление, изменение, удаление данных в БД

Цель работы: реализовать базу данных из двух-трех таблиц, связанных между собой, и подключить к ней WPF проект, используя EntityFramework и DataSet, добавить возможность добавлять, изменять и удалять данные в таблицу.

1. Создание бд и таблиц.

Создаём 3 таблицы: «Students» с информацией о студентах, «Courses» с информацией о курсах и «StudentsCourses», где связь многие ко многим между таблицами: «Students» и «Courses».

```
create database University;
go
use University;
go

create table Students(
    ID_Student int primary key identity(1,1),
    StudentSurname varchar(50) not null,
    StudentName varchar(50) not null,
    StudentMiddleName varchar(50),
    Age int not null,
    Email varchar(100) not null
);
go

create table Courses (
    ID_Course int primary key identity(1,1),
    Title varchar(100) not null,
    DescriptionCourse text
);
go

create table StudentsCourses(
    ID_StudentsCourses int primary key identity(1,1),
    Student_ID int not null,
    Course_ID int not null,
    foreign key (Student_ID) references Students(ID_Student),
    foreign key (Course_ID) references Courses(ID_Course)
);
go
```

Рисунок 22 - Создание бд и таблиц

2. Заполнение таблиц.

```

INSERT INTO Students (StudentSurname, StudentName, StudentMiddleName, Age, Email)
VALUES
('Иванов', 'Иван', 'Иванович', 20, 'ivan@yandex.ru'),
('Петров', 'Петр', 'Петрович', 22, 'petr@mail.ru'),
('Мурлаева', 'Екатерина', 'Олеговна', 21, 'katapulta@gmail.com');
go

INSERT INTO Courses (Title, DescriptionCourse)
VALUES
('Введение в программирование', 'Основы программирования на языке Python'),
('Математика для инженеров', 'Основные математические понятия и методы в инженерных науках'),
('Английский язык', '');
go

INSERT INTO StudentsCourses (Student_ID, Course_ID)
VALUES
(5, 1),
(6, 2),
(7, 3);
go

```

Рисунок 23 - Заполнение таблиц

3. Создание представления.

Для создания представления используем оператор inner join для объединения соответствующих данных из нескольких таблиц.

```

CREATE VIEW StudentsCoursesView2 AS
SELECT
  StudentsCourses.ID_StudentsCourses AS 'ID',
  Students.StudentSurname + ' ' + SUBSTRING(Students.StudentName, 1, 1) + '.' + SUBSTRING(Students.StudentMiddleName, 1, 1) + '.' AS 'StudentFullName',
  Courses.Title AS 'CourseTitle'
FROM StudentsCourses
INNER JOIN
  Students ON StudentsCourses.Student_ID = Students.ID_Student
INNER JOIN
  Courses ON StudentsCourses.Course_ID = Courses.ID_Course;
go

```

Рисунок 24 - Создание представления

4. Создание кнопок для основного меню.

В основном xaml создаём три кнопки: «Students» для отображения данных из таблицы Students, «Courses» для отображения данных из таблицы Courses и «StudentsCourses» для отображения данных из представления.

```

<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>

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

  <Button x:Name="Students" Content="Просмотр студентов" Click="Students_Click" Width="250px" Height="50px" Grid.Row="0" Grid.Column="0" />
  <Button x:Name="Courses" Content="Просмотр курсов" Click="Courses_Click" Width="250px" Height="50px" Grid.Row="1" Grid.Column="0" />
  <Button x:Name="StudentsCourses" Content="Просмотр студентов и их курсов" Click="StudentsCourses_Click" Width="250px" Height="50px" Grid.Row="2" Grid.Column="0" />
</Grid>
</Window>

```

Рисунок 25 - MainWindow.xaml

Пропишем логику в основном CS для создания новых окон при нажатии:

```
Ссылка 2
public partial class MainWindow : Window
{
    Ссылка 0
    public MainWindow()
    {
        InitializeComponent();
    }
    Ссылка 1
    private void Students_Click(object sender, RoutedEventArgs e)
    {
        StudentsW window1 = new StudentsW();
        window1.ShowDialog();
        if (window1.DialogResult == false)
        {
            window1.Close();
        }
    }

    Ссылка 1
    private void Courses_Click(object sender, RoutedEventArgs e)
    {
        CoursesW window2 = new CoursesW();
        window2.ShowDialog();
        if (window2.DialogResult == false)
        {
            window2.Close();
        }
    }

    Ссылка 1
    private void StudentsCourses_Click(object sender, RoutedEventArgs e)
    {
        StudentsCoursesW window2 = new StudentsCoursesW();
        window2.ShowDialog();
        if (window2.DialogResult == false)
        {
            window2.Close();
        }
    }
}
```

Рисунок 26 - MainWindow.xaml.cs

5. Добавление, изменение и удаление данных с использованием EF.

Добавляем в StudentsW.xaml.cs такие методы как: StudentsGrd_SelectionChanged(object sender, SelectionChangedEventArgs e): обработчик события изменения выбора в StudentsGrd. Обновляет данные в текстовых полях окна на основе выбранного студента в StudentsGrd.

AddButton_Click(object sender, RoutedEventArgs e): обработчик нажатия кнопки "Добавить". Проверяет, заполнены ли все текстовые поля, и выводит сообщение об ошибке, если они не заполнены. Создает нового студента на основе введенных данных и добавляет его в базу данных. Обновляет данные в StudentsGrd и вызывает метод UpdateStudentsCourses().


```

Ссылка: 4
public partial class StudentsW : Window
{
    private UniversityEntities context = new UniversityEntities();
    Ссылка: 1
    public StudentsW()
    {
        InitializeComponent();
        StudentsGrd.ItemsSource = context.Students.ToList();
        StudentsGrd.SelectionChanged += StudentsGrd_SelectionChanged;
    }
    Ссылка: 1
    private void StudentsGrd_SelectionChanged(object sender, SelectionChangedEventArgs e)
    {
        if (StudentsGrd.SelectedItem != null)
        {
            Students selectedStudent = (Students)StudentsGrd.SelectedItem;

            TextBoxSurname.Text = selectedStudent.StudentSurname;
            TextBoxName.Text = selectedStudent.StudentName;
            TextBoxMiddleName.Text = selectedStudent.StudentMiddleName;
            TextBoxAge.Text = selectedStudent.Age.ToString();
            TextBoxEmail.Text = selectedStudent.Email;
        }
    }
    Ссылка: 1
    private void AddButton_Click(object sender, RoutedEventArgs e)
    {
        if (string.IsNullOrEmpty(TextBoxSurname.Text) || string.IsNullOrEmpty(TextBoxName.Text) ||
            string.IsNullOrEmpty(TextBoxAge.Text) || string.IsNullOrEmpty(TextBoxEmail.Text))
        {
            MessageBox.Show("Пожалуйста, заполните все текстовые поля.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
            return;
        }

        Students newStudent = new Students()
        {
            StudentSurname = TextBoxSurname.Text,
            StudentName = TextBoxName.Text,
            StudentMiddleName = TextBoxMiddleName.Text,
            Age = Convert.ToInt32(TextBoxAge.Text),
            Email = TextBoxEmail.Text
        };

        context.Students.Add(newStudent);
        context.SaveChanges();

        StudentsGrd.ItemsSource = context.Students.ToList();

        UpdateStudentsCourses();
    }
}

```

Рисунок 27 - StudentsW.xaml.cs

Далее добавляем метод `UpdateButton_Click(object sender, RoutedEventArgs e)`: обработчик нажатия кнопки "Изменить". Проверяет, заполнены ли все текстовые поля, и выводит сообщение об ошибке, если они не заполнены. Обновляет данные выбранного студента на основе введенных данных и сохраняет изменения в базе данных. Обновляет данные в `StudentsGrd` и вызывает метод `UpdateStudentsCourses()`.

`DeleteButton_Click(object sender, RoutedEventArgs e)`: обработчик нажатия кнопки "Удалить". Получает выбранного студента и удаляет его из базы данных. Удаляет студента из всех курсов, используя метод `RemoveStudentFromCourses()`. Обновляет данные в `StudentsGrd`.

RemoveStudentFromCourses(int studentId): метод для удаления студента из всех курсов. Получает все записи о курсах, в которых участвует данный студент, и удаляет их из базы данных.

```
Ссылка 1
private void UpdateButton_Click(object sender, RoutedEventArgs e)
{
    if (string.IsNullOrEmpty(TextBoxSurname.Text) || string.IsNullOrEmpty(TextBoxName.Text) ||
        string.IsNullOrEmpty(TextBoxAge.Text) || string.IsNullOrEmpty(TextBoxEmail.Text))
    {
        MessageBox.Show("Пожалуйста, заполните все текстовые поля.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
        return;
    }

    Students selectedStudent = (Students)StudentsGrd.SelectedItem;

    if (selectedStudent != null)
    {
        selectedStudent.StudentSurname = TextBoxSurname.Text;
        selectedStudent.StudentName = TextBoxName.Text;
        selectedStudent.StudentMiddleName = TextBoxMiddleName.Text;
        selectedStudent.Age = Convert.ToInt32(TextBoxAge.Text);
        selectedStudent.Email = TextBoxEmail.Text;

        context.SaveChanges();

        StudentsGrd.ItemsSource = context.Students.ToList();

        UpdateStudentsCourses();
    }
}

Ссылка 1
private void DeleteButton_Click(object sender, RoutedEventArgs e)
{
    Students selectedStudent = (Students)StudentsGrd.SelectedItem;

    if (selectedStudent != null)
    {
        RemoveStudentFromCourses(selectedStudent.ID_Student);

        context.Students.Remove(selectedStudent);
        context.SaveChanges();

        StudentsGrd.ItemsSource = context.Students.ToList();
    }
}

Ссылка 1
private void RemoveStudentFromCourses(int studentId)
{
    var studentCourses = context.StudentsCourses.Where(sc => sc.Student_ID == studentId).ToList();
    context.StudentsCourses.RemoveRange(studentCourses);
    context.SaveChanges();
}
```

Рисунок 28 - StudentsW.xaml.cs

Метод UpdateStudentsCourses() необходим для обновления связей между студентами и курсами. Проверяет, существуют ли все связи между студентами и курсами, и добавляет отсутствующие связи в базу данных. Удаляет связи, если студент или курс были удалены.

```

private void UpdateStudentsCourses()
{
    foreach (var studentCourse in context.StudentsCourses.ToList())
    {
        if (!context.Students.Any(s => s.ID_Student == studentCourse.Student_ID) ||
            !context.Courses.Any(c => c.ID_Course == studentCourse.Course_ID))
        {
            context.StudentsCourses.Remove(studentCourse);
        }
    }

    foreach (var student in context.Students)
    {
        var studentCourses = context.StudentsCourses
            .Where(sc => sc.Student_ID == student.ID_Student)
            .Select(sc => sc.Course_ID)
            .ToList();

        foreach (var courseId in studentCourses)
        {
            if (!context.StudentsCourses.Any(sc => sc.Student_ID == student.ID_Student && sc.Course_ID == courseId))
            {
                context.StudentsCourses.Add(new StudentsCourses()
                {
                    Student_ID = student.ID_Student,
                    Course_ID = courseId
                });
            }
        }
    }

    context.SaveChanges();
}

```

Рисунок 29 - StudentsW.xaml.cs

TextBoxAge_PreviewTextInput(object sender, TextCompositionEventArgs e): обработчик события предварительного ввода текста в поле TextBoxAge. Разрешает ввод только цифр.

TextBoxSurname_PreviewTextInput(object sender, TextCompositionEventArgs e): обработчик события предварительного ввода текста в поле TextBoxSurname. Разрешает ввод только букв и пробелов и ограничивает количество символов.

TextBoxEmail_PreviewTextInput(object sender, TextCompositionEventArgs e): обработчик события предварительного ввода текста в поле TextBoxEmail. Ограничивает количество символов.

Back_Click(object sender, RoutedEventArgs e): обработчик нажатия кнопки "Назад". Устанавливает свойство DialogResult окна на false.

```

Ссылка 1
private void TextBoxAge_PreviewTextInput(object sender, TextCompositionEventArgs e)
{
    if (!System.Text.RegularExpressions.Regex.IsMatch(e.Text, "[0-9]+$"))
    {
        e.Handled = true;
    }
}

Ссылка 3
private void TextBoxSurname_PreviewTextInput(object sender, TextCompositionEventArgs e)
{
    TextBox textBox = sender as TextBox;
    if (textBox != null)
    {
        int maxLength = 50;

        if (textBox.Text.Length >= maxLength)
        {
            e.Handled = true;
            return;
        }

        foreach (char c in e.Text)
        {
            if (!char.IsLetter(c) && !char.IsWhiteSpace(c))
            {
                e.Handled = true;
                return;
            }
        }
    }
}

Ссылка 1
private void TextBoxEmail_PreviewTextInput(object sender, TextCompositionEventArgs e)
{
    TextBox textBox = sender as TextBox;
    if (textBox != null)
    {
        int maxLength = 100;
        if (textBox.Text.Length >= maxLength)
        {
            e.Handled = true;
        }
    }
}

Ссылка 1
private void Back_Click(object sender, RoutedEventArgs e)
{
    this.DialogResult = false;
}

```

Рисунок 30 - StudentsW.xaml.cs

В xaml добавляем TextBox и необходимые методы внутри тегов.

Используем Header и для связки - Binding для отображения названий столбцов на русском.

```

RowDefinition Height="*"/>
</Grid.RowDefinitions>
<DataGrid x:Name="StudentsGrid" AutoGenerateColumns="False" Grid.Row="0" Grid.RowSpan="4">
    <DataGrid.Columns>
        <DataGridTextColumn Header="Фамилия" Binding="{Binding StudentSurname}" />
        <DataGridTextColumn Header="Имя" Binding="{Binding StudentName}" />
        <DataGridTextColumn Header="Отчество" Binding="{Binding StudentMiddleName}" />
        <DataGridTextColumn Header="Возраст" Binding="{Binding Age}" />
        <DataGridTextColumn Header="Почта" Binding="{Binding Email}" />
    </DataGrid.Columns>
</DataGrid>
<Button Content="Добавить" Click="AddButton_Click" VerticalAlignment="Top" Grid.Row="0" Grid.Column="1" Width="100" HorizontalAlignment="Left" Background="#FF6CB464"/>
<Button Content="Изменить" Click="UpdateButton_Click" VerticalAlignment="Top"
Grid.Row="0" Grid.Column="1" Width="100"
HorizontalAlignment="Center" Background="#FF6CB464" />
<Button Content="Удалить" Click="DeleteButton_Click" VerticalAlignment="Top" Grid.Row="0" Grid.Column="2" Width="100" HorizontalAlignment="Right" Background="#FF6CB464" />
<TextBox x:Name="TextBoxSurname" PreviewTextInput="TextBoxSurname_PreviewTextInput" Grid.Row="1" Grid.Column="1" Width="100px" Height="30" Background="#3389B498" HorizontalAlignment="Left"/>
<TextBox x:Name="TextBoxName" PreviewTextInput="TextBoxName_PreviewTextInput" Grid.Row="1" Grid.Column="1" Width="100px" Height="30" Background="#3389B498" HorizontalAlignment="Left"/>
<TextBox x:Name="TextBoxMiddleName" PreviewTextInput="TextBoxMiddleName_PreviewTextInput" Grid.Row="1" Grid.Column="1" Width="100px" Height="30" Background="#3389B498" HorizontalAlignment="Left"/>
<TextBox PreviewTextInput="TextBoxAge_PreviewTextInput" x:Name="TextBoxAge" Grid.Row="2" Grid.Column="1" Width="100px" Height="30" Background="#3389B498" HorizontalAlignment="Left"/>
<TextBox PreviewTextInput="TextBoxEmail_PreviewTextInput" x:Name="TextBoxEmail" Grid.Row="2" Grid.Column="1" Width="100px" Height="30" Background="#3389B498" HorizontalAlignment="Left"/>
<TextBlock Text="Фамилия" Grid.Column="1" Grid.Row="1" VerticalAlignment="Top" HorizontalAlignment="Left"/>
<TextBlock Text="Имя" VerticalAlignment="Top" Grid.Column="1" Grid.Row="1" HorizontalAlignment="Center"/></TextBlock>
<TextBlock Text="Отчество" HorizontalAlignment="Right" VerticalAlignment="Top" Grid.Column="1" Grid.Row="1"/></TextBlock>
<TextBlock Text="Возраст" HorizontalAlignment="Left" VerticalAlignment="Top" Grid.Column="1" Grid.Row="2"/></TextBlock>
<TextBlock Text="Почта" VerticalAlignment="Top" Grid.Column="1" Grid.Row="2" HorizontalAlignment="Center"/></TextBlock>
<Button Content="Назад" Click="Back_Click" Grid.Column="1" Grid.Row="3" Width="100" HorizontalAlignment="Right" VerticalAlignment="Bottom" Background="#FF6CB464" BorderBrush="#FF415927"/>

```

Рисунок 31 - StudentsW.xaml

Аналогично необходимо добавить методы для добавления, изменения и удаления в StudentsCoursesW, но в этом классе мы будем использовать комбобоксы. В конструкторе создаем их:

```
Ссылка: 1
public StudentsCoursesW()
{
    InitializeComponent();
    StudentsCoursesGrd.ItemsSource = context.StudentsCoursesView2.ToList();

    ComboBoxStudents.ItemsSource = context.Students.ToList();
    ComboBoxStudents.DisplayMemberPath = "StudentSurname";

    ComboBoxCourses.ItemsSource = context.Courses.ToList();
    ComboBoxCourses.DisplayMemberPath = "Title";
}
```

Рисунок 32 - StudentsCoursesW.xaml.cs

Также добавляем в xaml:

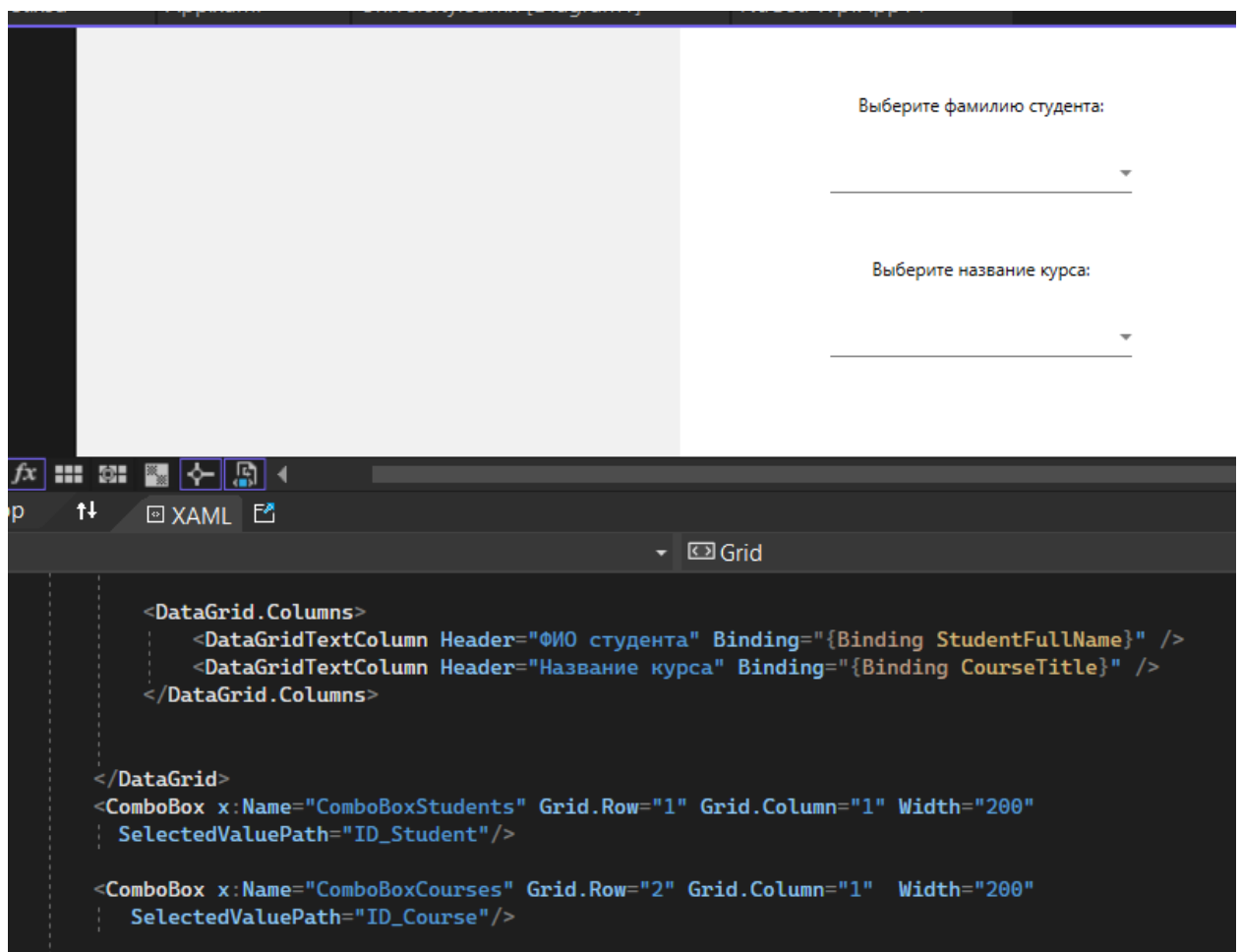


Рисунок 33 - StudentsCoursesW.xaml

Далее прописываем добавление и изменение данных.

AddButton_Click(object sender, RoutedEventArgs e): обработчик нажатия кнопки "Добавить". Проверяет, выбраны ли студент и курс, и выводит

сообщение об ошибке, если они не выбраны. Создает новую запись о связи студента и курса и добавляет её в базу данных. Обновляет данные в StudentsCoursesGrd.

UpdateButton_Click(object sender, RoutedEventArgs e): обработчик нажатия кнопки "Изменить". Получает выбранную связь студента и курса и обновляет её данные на основе выбранных студента и курса из ComboBox. Обновляет данные в StudentsCoursesGrd.

```
Ссылка 1
private void AddButton_Click(object sender, RoutedEventArgs e)
{
    if (ComboBoxStudents.SelectedItem == null || ComboBoxCourses.SelectedItem == null)
    {
        MessageBox.Show("Пожалуйста, выберите студента и курс.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
        return;
    }

    Students selectedStudent = (Students)ComboBoxStudents.SelectedItem;
    Courses selectedCourse = (Courses)ComboBoxCourses.SelectedItem;

    if (selectedStudent == null || selectedCourse == null)
    {
        MessageBox.Show("Пожалуйста, выберите студента и курс.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
        return;
    }

    StudentsCourses newStudentCourse = new StudentsCourses()
    {
        Student_ID = selectedStudent.ID_Student,
        Course_ID = selectedCourse.ID_Course
    };
    context.StudentsCourses.Add(newStudentCourse);
    context.SaveChanges();

    StudentsCoursesGrd.ItemsSource = context.StudentsCoursesView2.ToList();
}

Ссылка 1
private void UpdateButton_Click(object sender, RoutedEventArgs e)
{
    StudentsCoursesView2 selectedStudentCourseView = (StudentsCoursesView2)StudentsCoursesGrd.SelectedItem;

    if (selectedStudentCourseView != null)
    {
        Students selectedStudent = (Students)ComboBoxStudents.SelectedItem;
        Courses selectedCourse = (Courses)ComboBoxCourses.SelectedItem;

        StudentsCourses selectedStudentCourse = context.StudentsCourses.Find(selectedStudentCourseView.ID);

        if (selectedStudent != null && selectedCourse != null)
        {
            selectedStudentCourse.Student_ID = selectedStudent.ID_Student;
            selectedStudentCourse.Course_ID = selectedCourse.ID_Course;
            context.SaveChanges();

            StudentsCoursesGrd.ItemsSource = context.StudentsCoursesView2.ToList();
        }
    }
}
```

Рисунок 34 - StudentsCoursesW.xaml.cs

DeleteButton_Click(object sender, RoutedEventArgs e): обработчик нажатия кнопки "Удалить". Получает выбранную связь студента и курса и удаляет её из базы данных. Обновляет данные в StudentsCoursesGrd.

```

Ссылка 1
private void DeleteButton_Click(object sender, RoutedEventArgs e)
{
    StudentsCoursesView2 selectedStudentCourseView = (StudentsCoursesView2)StudentsCoursesGrd.SelectedItem;

    if (selectedStudentCourseView != null)
    {
        StudentsCourses selectedStudentCourse = context.StudentsCourses.Find(selectedStudentCourseView.ID);

        if (selectedStudentCourse != null)
        {
            context.StudentsCourses.Remove(selectedStudentCourse);
            context.SaveChanges();

            StudentsCoursesGrd.ItemsSource = context.StudentsCoursesView2.ToList();
        }
    }
}

```

Рисунок 35 - StudentsCoursesW.xaml.cs

6. Добавление, изменение и удаление данных с использованием DataSet.
Добавляем запрос для удаления:

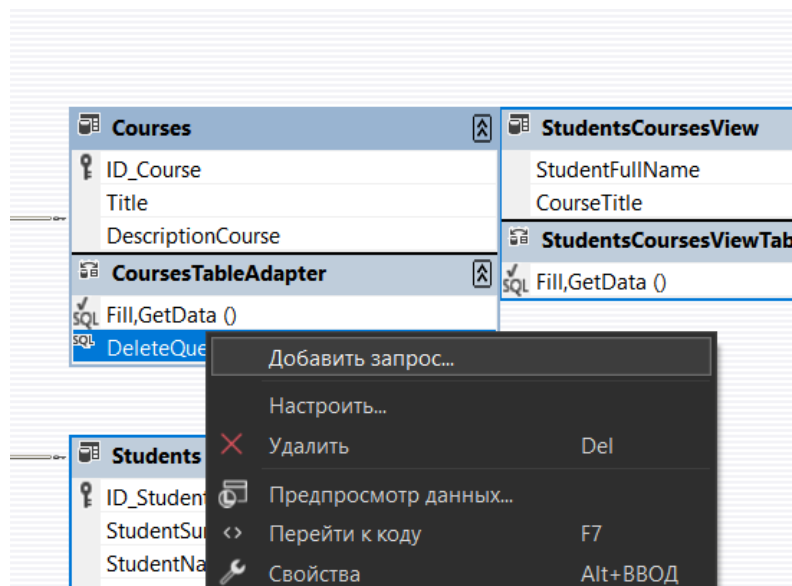


Рисунок 36 - Добавление запроса

Нажимаем далее использовать инструкции sql->delete

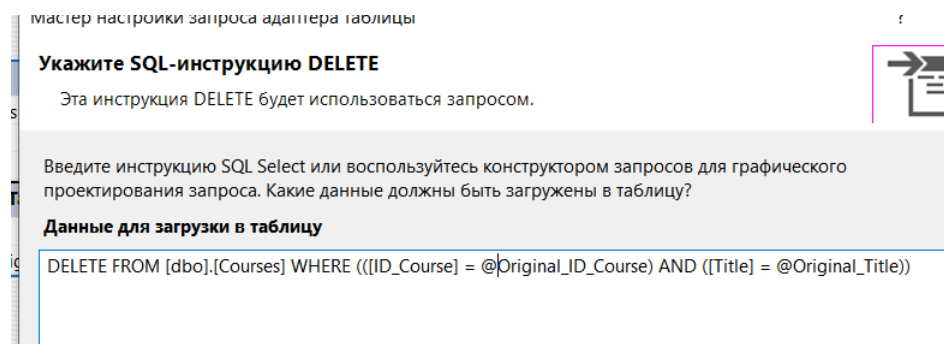


Рисунок 37 - Запрос

В классе Courses создаём методы для добавления, изменение и удаления:

CoursesDataSet_SelectionChanged(object sender, SelectionChangedEventArgs e): обработчик события изменения выбора в CoursesDataSet. Обновляет данные в текстовых полях окна на основе выбранного курса. CoursesGrd_SelectionChanged(object sender, SelectionChangedEventArgs e):

AddButton_Click(object sender, RoutedEventArgs e): обработчик нажатия кнопки "Добавить". Проверяет, заполнено ли поле с названием курса, и выводит сообщение об ошибке, если оно не заполнено. Добавляет новый курс в базу данных с использованием метода Insert и обновляет данные в CoursesDataSet.

```
Ссылка 4
public partial class CoursesW : Window
{
    private CoursesTableAdapter crs = new CoursesTableAdapter();
    private StudentsCoursesTableAdapter stdcrs = new StudentsCoursesTableAdapter();
    Ссылка 1
    public CoursesW()
    {
        InitializeComponent();
        CoursesDataSet.ItemsSource = crs.GetData();
    }
    Ссылка 1
    private void CoursesDataSet_SelectionChanged(object sender, SelectionChangedEventArgs e)
    {
        if (CoursesDataSet.SelectedItem != null)
        {
            DataRowView rowView = (DataRowView)CoursesDataSet.SelectedItem;
            DataRow row = rowView.Row;

            TextBoxTitle.Text = row["Title"].ToString();
            TextBoxDescription.Text = row["DescriptionCourse"].ToString();
        }
    }

    Ссылка 1
    private void AddButton_Click(object sender, RoutedEventArgs e)
    {
        if (string.IsNullOrEmpty(TextBoxTitle.Text))
        {
            MessageBox.Show("Пожалуйста, введите название курса.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
            return;
        }

        crs.Insert(TextBoxTitle.Text, TextBoxDescription.Text);
        CoursesDataSet.ItemsSource = crs.GetData();
    }
}
```

Рисунок 38 - CoursesW.xaml.cs

UpdateButton_Click(object sender, RoutedEventArgs e): обработчик нажатия кнопки "Изменить". Проверяет, заполнено ли поле с названием курса, и выводит сообщение об ошибке, если оно не заполнено. Также проверяет выбрана ли вообще строка, которую надо изменить, если не выбрана, то выводится сообщение об ошибке. Получает выбранный курс и обновляет его данные на основе введенных данных в текстовых полях. Обновляет данные в CoursesDataSet.

DeleteButton_Click(object sender, RoutedEventArgs e): обработчик нажатия кнопки "Удалить". Получает выбранный курс и удаляет его из базы данных с использованием метода DeleteQuery. Удаляет связи курса со студентами с помощью метода DeleteByCourseID из таблицы связей StudentsCourses. Обновляет данные в CoursesDataSet.

TextBoxTitle_PreviewTextInput(object sender, TextCompositionEventArgs e): ограничивает количество символов.

```
Ссылка 1
private void UpdateButton_Click(object sender, RoutedEventArgs e)
{
    if (CoursesDataSet.SelectedItem == null)
    {
        MessageBox.Show("Пожалуйста, выберите курс для изменения.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
        return;
    }
    if (string.IsNullOrWhiteSpace(TextBoxTitle.Text))
    {
        MessageBox.Show("Пожалуйста, введите название курса.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
        return;
    }
    var selectedCourse = (CoursesRow)((DataRowView)CoursesDataSet.SelectedItem).Row;
    selectedCourse.Title = TextBoxTitle.Text;
    selectedCourse.DescriptionCourse = TextBoxDescription.Text;
    crs.Update(selectedCourse);
    CoursesDataSet.ItemsSource = crs.GetData();
}

Ссылка 1
private void DeleteButton_Click(object sender, RoutedEventArgs e)
{
    var selectedCourse = (CoursesRow)((DataRowView)CoursesDataSet.SelectedItem).Row;
    stdcrs.DeleteByCourseID(selectedCourse.ID_Course);
    crs.DeleteQuery(selectedCourse.ID_Course, selectedCourse.Title);

    CoursesDataSet.ItemsSource = crs.GetData();
}

Ссылка 1
private void TextBoxTitle_PreviewTextInput(object sender, TextCompositionEventArgs e)
{
    TextBox textBox = sender as TextBox;
    if (textBox != null)
    {
        int maxLength = 100;
        if (textBox.Text.Length >= maxLength)
        {
            e.Handled = true;
        }
    }
}

Ссылка 1
private void Back_Click(object sender, RoutedEventArgs e)
{
    this.DialogResult = false;
}
```

Рисунок 39 - CoursesW.xaml.cs

Этот метод предназначен для удобного удаления всех записей о связях студентов и курсов по идентификатору курса из таблицы базы данных.

```

Ссылка 6
partial class StudentsCoursesTableAdapter
{
    Ссылка 1
    public int DeleteByCourseID(int courseID)
    {
        using (var connection = new SqlConnection(this.Connection.ConnectionString))
        {
            string deleteStatement = "DELETE FROM StudentsCourses WHERE Course_ID = @CourseID";
            SqlCommand command = new SqlCommand(deleteStatement, connection);
            command.Parameters.AddWithValue("@CourseID", courseID);

            connection.Open();
            return command.ExecuteNonQuery();
        }
    }
}

```

Рисунок 40 - Метод для удаления

Вывод: реализовали базу данных из двух-трех таблиц, связанных между собой, и подключили к ней WPF проект, используя EntityFramework и DataSet, добавили возможность добавлять, изменять и удалять данные в таблицу.

ЛАБОРАТОРНАЯ РАБОТА №3

Отображение нескольких таблиц из БД

Цель работы: реализовать базу данных из двух-трех таблиц, связанных между собой, и подключить к ней WPF проект, используя EntityFramework и DataSet, продемонстрировать все данные в одной таблице, также данные из нескольких таблиц должны отображаться в адекватном варианте.

1. Создание бд и таблиц.

Создаём базу данных с предметной областью «Клиника» и используем её. Далее создаём 5 таблиц: «Doctors», «Patients», «Appointments», «Gender» и «Specializations». В таблице «Appointments» используются внешние ключи для связи с таблицами «Patients» и «Doctors». В таблице «Patients» используется внешний ключ Gender_ID для связи с таблицей «Gender». В таблице «Doctors» используется внешний ключ Specialization_ID для связи с таблицей «Specializations».

```
create table Gender (
    ID_Gender int primary key identity(1,1),
    GenderValue varchar(1) not null
);
insert into Gender (GenderValue) values
('M'),
('X');

create table Specializations (
    ID_Specialization int primary key identity(1,1),
    SpecializationName varchar(100) not null
);

create table Doctors (
    ID_Doctor int primary key identity(1,1),
    DoctorSurname varchar(50) not null,
    DoctorName varchar(50) not null,
    DoctorMiddleName varchar(50),
    Specialization_ID int not null,
    foreign key (Specialization_ID) references Specializations(ID_Specialization)
);

create table Patients (
    ID_Patient int primary key identity(1,1),
    PatientSurname varchar(50) not null,
    PatientName varchar(50) not null,
    PatientMiddleName varchar(50),
    Gender_ID int not null,
    DateOfBirth date not null,
    foreign key (Gender_ID) references Gender(ID_Gender)
);

create table Appointments (
    ID_Appointment int primary key identity(1,1),
    Patient_ID int not null,
    Doctor_ID int not null,
    DateAppointment datetime not null,
    DescriptionAppointment text not null,
    foreign key (Patient_ID) references Patients(ID_Patient),
    foreign key (Doctor_ID) references Doctors(ID_Doctor)
);
```

Рисунок 41 - Создание БД и таблиц

2. Заполнение таблиц данными.

```
insert into Patients (PatientSurname, PatientName, PatientMiddleName, DateOfBirth, Gender_ID) values
('Кузнецова', 'Анастасия', 'Андреевна', '21.08.2004', '2'),
('Смирнов', 'Егор', 'Романович', '29.11.1973', '1'),
('Крылов', 'Даниил', 'Александрович', '05.08.1976', '1'),
('Антонов', 'Август', 'Всеволодович', '31.08.2003', '1'),
('Родионов', 'Аркадий', 'Федотович', '21.08.2002', '1');
go

insert into Specializations (SpecializationName) values
('Терапевт'),
('Хирург'),
('Офтальмолог'),
('Уролог'),
('Гинеколог'),
('Оториноларинголог'),
('Невролог');
go

insert into Doctors (DoctorSurname, DoctorName, DoctorMiddleName, Specialization_ID) values
('Путятин', 'Алексей', 'Владимирович', 1),
('Заперенчук', 'Илья', 'Сергеевич', 2),
('Березина', 'Ксения', '', 3);
go

insert into Appointments (Patient_ID, Doctor_ID, DateAppointment, DescriptionAppointment) values
(1, 1, '15.03.2024 09:00', 'Общий осмотр'),
(2, 2, '16.03.2024 10:30', 'Плановая операция'),
(3, 3, '17.03.2024 15:45', 'Проверка зрения');
go
```

Рисунок 42 - Заполнение таблиц

3. Создание представления.

Для создания представления используем оператор inner join для объединения соответствующих данных из нескольких таблиц.

```
CREATE VIEW ClinicView
AS
SELECT
    Patients.PatientSurname,
    Patients.PatientName,
    Patients.PatientMiddleName,
    Gender.GenderValue,
    Patients.DateOfBirth,
    Doctors.DoctorSurname,
    Doctors.DoctorName,
    Doctors.DoctorMiddleName,
    Specializations.SpecializationName,
    Appointments.DateAppointment,
    Appointments.DescriptionAppointment
FROM Appointments
INNER JOIN Patients ON Appointments.Patient_ID = Patients.ID_Patient
INNER JOIN Doctors ON Appointments.Doctor_ID = Doctors.ID_Doctor
INNER JOIN Gender ON Patients.Gender_ID = Gender.ID_Gender
INNER JOIN Specializations ON Doctors.Specialization_ID = Specializations.ID_Specialization;
go
```

Рисунок 43 - Создание представления

4. Создание кнопок для основного меню.

В основном xaml создаём две кнопки: «EF» для отображения данных с использованием EF и «DataSet» для отображения данных с использованием DataSet.

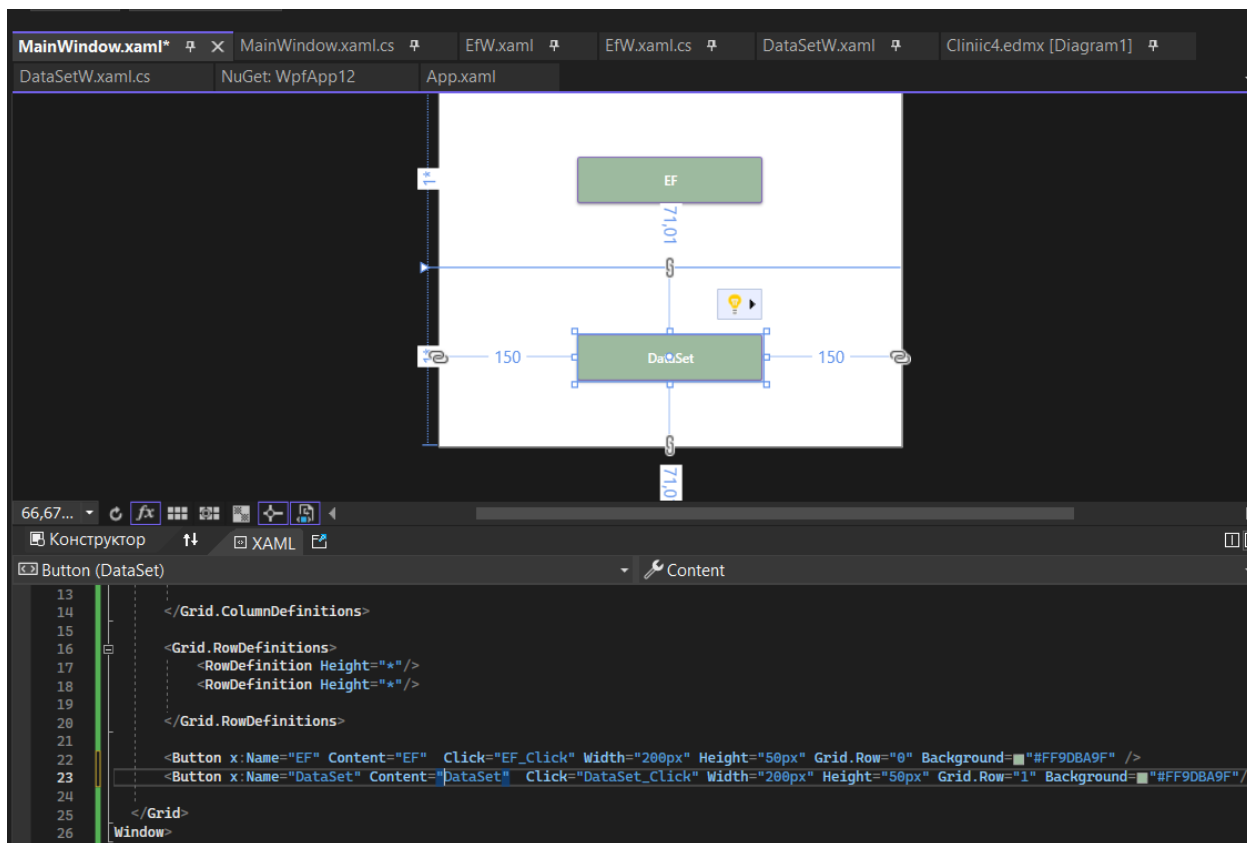


Рисунок 44 - Создание кнопок

Пропишем логику в основном cs для создания новых окон при нажатии:

```

Ссылка 2
public partial class MainWindow : Window
{
    Ссылка 0
    public MainWindow()
    {
        InitializeComponent();
    }

    Ссылка 1
    private void EF_Click(object sender, RoutedEventArgs e)
    {
        EfW window1 = new EfW();
        window1.ShowDialog();
        if (window1.DialogResult == false)
        {
            window1.Close();
        }
    }

    Ссылка 1
    private void DataSet_Click(object sender, RoutedEventArgs e)
    {
        DataSetW window2 = new DataSetW();
        window2.ShowDialog();
        if (window2.DialogResult == false)
        {
            window2.Close();
        }
    }
}

```

Рисунок 45 - Класс MainWindow.xaml.cs

5. Чтение данных из нескольких таблиц с использованием EF.

Для корректного отображения названий столбцов и даты используем `AutoGenerateColumns="False"` для отключения автоматической генерации и `Binding` для связывания. В `Header` записываем как хотим, чтобы отображалось на интерфейсе, а в `Binding` названия столбцов из представления. Также используем `StringFormat` для корректного отображения даты и времени.

```

Title="Клиника" Height="450" Width="1000">
<Grid>
<DataGrid x:Name="ClinicViewGrd" AutoGenerateColumns="False" >
<DataGrid.Columns>
<DataGridTextColumn Header="Фамилия пациента" Binding="{Binding PatientSurname}" />
<DataGridTextColumn Header="Имя пациента" Binding="{Binding PatientName}" />
<DataGridTextColumn Header="Отчество пациента" Binding="{Binding PatientMiddleName}" />
<DataGridTextColumn Header="Дата рождения пациента" Binding="{Binding DateOfBirth, StringFormat=\{0:dd.MM.yyyy\}}" />
<DataGridTextColumn Header="Пол пациента" Binding="{Binding GenderValue}" />
<DataGridTextColumn Header="Фамилия врача" Binding="{Binding DoctorSurname}" />
<DataGridTextColumn Header="Имя врача" Binding="{Binding DoctorName}" />
<DataGridTextColumn Header="Отчество врача" Binding="{Binding DoctorMiddleName}" />
<DataGridTextColumn Header="Специализация" Binding="{Binding SpecializationName}" />
<DataGridTextColumn Header="Дата приема" Binding="{Binding DateAppointment, StringFormat=\{0:dd.MM.yyyy HH:mm\}}" />
<DataGridTextColumn Header="Описание приема" Binding="{Binding DescriptionAppointment}" />
</DataGrid.Columns>
</DataGrid>
</Grid>

```

Рисунок 46 - Класс EfW.xaml

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.Windows;
7  using System.Windows.Controls;
8  using System.Windows.Data;
9  using System.Windows.Documents;
10 using System.Windows.Input;
11 using System.Windows.Media;
12 using System.Windows.Media.Imaging;
13 using System.Windows.Shapes;
14
15 namespace WpfApp12
16 {
17     Ссылка 4
18     public partial class EfW : Window
19     {
20         private Clinic4Entities context = new Clinic4Entities();
21         Ссылка 1
22         public EfW()
23         {
24             InitializeComponent();
25             ClinicViewGrd.ItemsSource = context.ClinicView.ToList();
26         }
27     }
28 }

```

Рисунок 47 - Класс EfW.xaml.cs

6. Чтение данных из нескольких таблиц с использованием DataSet.

Здесь мы делаем то же самое, что и в EF.

```

Grid>
<DataGrid x:Name="ClinicDataSetGrd">
  <DataGrid.Columns>
    <DataGridTextColumn Header="Фамилия пациента" Binding="{Binding PatientSurname}" />
    <DataGridTextColumn Header="Имя пациента" Binding="{Binding PatientName}" />
    <DataGridTextColumn Header="Отчество пациента" Binding="{Binding PatientMiddleName}" />
    <DataGridTextColumn Header="Дата рождения пациента" Binding="{Binding DateOfBirth, StringFormat=\{0:dd.MM.yyyy\}}" />
    <DataGridTextColumn Header="Пол пациента" Binding="{Binding GenderValue}" />
    <DataGridTextColumn Header="Фамилия врача" Binding="{Binding DoctorSurname}" />
    <DataGridTextColumn Header="Имя врача" Binding="{Binding DoctorName}" />
    <DataGridTextColumn Header="Отчество врача" Binding="{Binding DoctorMiddleName}" />
    <DataGridTextColumn Header="Специализация" Binding="{Binding SpecializationName}" />
    <DataGridTextColumn Header="Дата приема" Binding="{Binding DateAppointment, StringFormat=\{0:dd.MM.yyyy HH:mm\}}" />
    <DataGridTextColumn Header="Описание приема" Binding="{Binding DescriptionAppointment}" />
  </DataGrid.Columns>
</DataGrid>
Grid<

```

Рисунок 48 - DataSet.xaml

```
7 using System.Windows.Controls;
8 using System.Windows.Data;
9 using System.Windows.Documents;
10 using System.Windows.Input;
11 using System.Windows.Media;
12 using System.Windows.Media.Imaging;
13 using System.Windows.Shapes;
14 using WpfApp12.Clinic4DataSetTableAdapters;
15
16
17 namespace WpfApp12
18 {
19     Ссылка: 4
20     public partial class DataSetW : Window
21     {
22         private ClinicViewTableAdapter cl = new ClinicViewTableAdapter();
23
24         Ссылка: 1
25         public DataSetW()
26         {
27             InitializeComponent();
28             ClinicDataSetGrd.ItemsSource = cl.GetData();
29         }
30     }
31 }
```

Рисунок 49 - DataSetW.xaml.cs

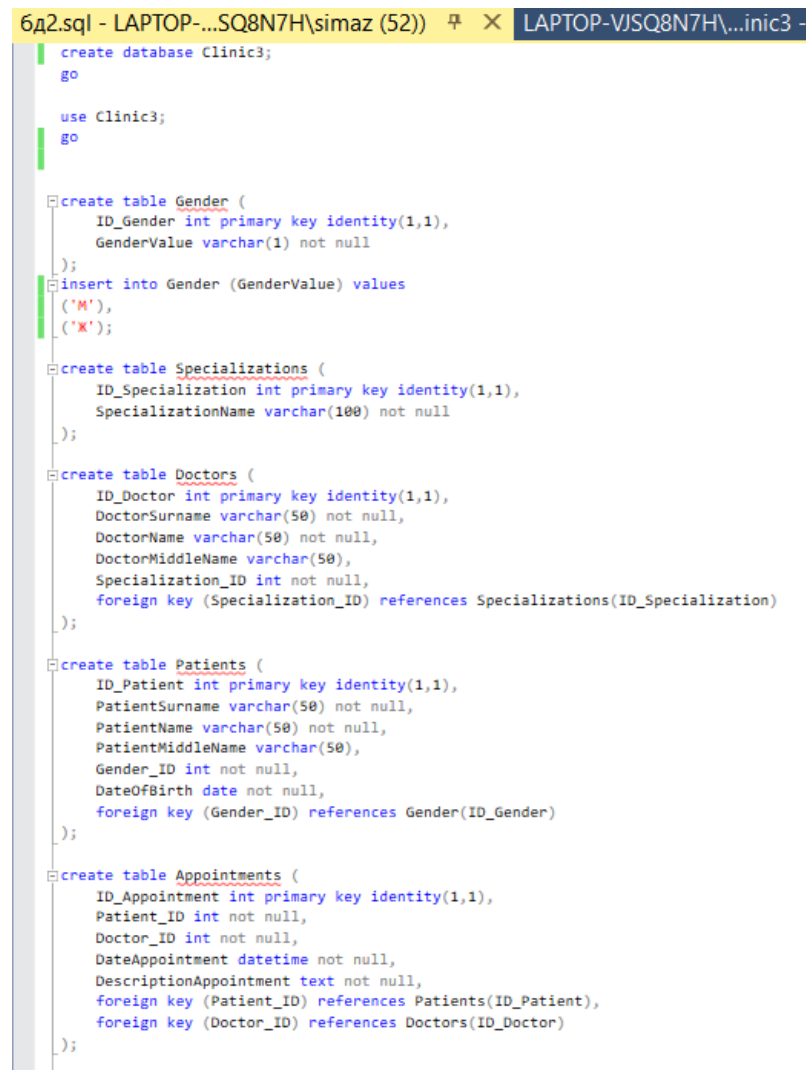
Вывод: реализовали базу данных из 5 таблиц, связанных между собой, и подключили к ней WPF проект, используя EntityFramework и DataSet, продемонстрировали все данные в одной таблице, также данные из нескольких таблиц отображаются в адекватном варианте.

ЛАБОРАТОРНАЯ РАБОТА №4

Поиск и фильтрация данных

Цель работы: реализовать базу данных из двух-трех таблиц, связанных между собой, и подключить к ней WPF проект, используя EntityFramework и DataSet, данные должны читаться в DataGrid, также необходимо реализовать поиск и фильтрацию хотя бы по одному столбцу.

1. Создание бд и таблиц.



```
create database Clinic3;
go

use Clinic3;
go

create table Gender (
    ID_Gender int primary key identity(1,1),
    GenderValue varchar(1) not null
);
insert into Gender (GenderValue) values
('M'),
('Ж');

create table Specializations (
    ID_Specialization int primary key identity(1,1),
    SpecializationName varchar(100) not null
);

create table Doctors (
    ID_Doctor int primary key identity(1,1),
    DoctorSurname varchar(50) not null,
    DoctorName varchar(50) not null,
    DoctorMiddleName varchar(50),
    Specialization_ID int not null,
    foreign key (Specialization_ID) references Specializations(ID_Specialization)
);

create table Patients (
    ID_Patient int primary key identity(1,1),
    PatientSurname varchar(50) not null,
    PatientName varchar(50) not null,
    PatientMiddleName varchar(50),
    Gender_ID int not null,
    DateOfBirth date not null,
    foreign key (Gender_ID) references Gender(ID_Gender)
);

create table Appointments (
    ID_Appointment int primary key identity(1,1),
    Patient_ID int not null,
    Doctor_ID int not null,
    DateAppointment datetime not null,
    DescriptionAppointment text not null,
    foreign key (Patient_ID) references Patients(ID_Patient),
    foreign key (Doctor_ID) references Doctors(ID_Doctor)
);
```

Рисунок 50 - Создание бд и таблиц

2. Заполнение таблиц данными.

```

insert into Patients (PatientSurname, PatientName, PatientMiddleName, DateOfBirth, Gender_ID) values
('Кузнецова', 'Анастасия', 'Андреевна', '21.08.2004', '2'),
('Смирнов', 'Егор', 'Романович', '29.11.1973', '1'),
('Крылов', 'Даниил', 'Александрович', '05.08.1976', '1'),
('Антонов', 'Август', 'Всеволодович', '31.08.2003', '1'),
('Родионов', 'Аркадий', 'Федотович', '21.08.2002', '1');
go

insert into Specializations (SpecializationName) values
('Терапевт'),
('Хирург'),
('Офтальмолог'),
('Уролог'),
('Гинеколог'),
('Оториноларинголог'),
('Невролог');
go

insert into Doctors (DoctorSurname, DoctorName, DoctorMiddleName, Specialization_ID) values
('Путятин', 'Алексей', 'Владимирович', 1),
('Заперенчук', 'Илья', 'Сергеевич', 2),
('Березина', 'Ксения', '', 3);
go

insert into Appointments (Patient_ID, Doctor_ID, DateAppointment, DescriptionAppointment) values
(1, 1, '15.03.2024 09:00', 'Общий осмотр'),
(2, 2, '16.03.2024 10:30', 'Плановая операция'),
(3, 3, '17.03.2024 15:45', 'Проверка зрения');
go

```

Рисунок 51 - Заполнение таблиц

3. Создание представлений.

```

CREATE VIEW PatientsView2
AS
SELECT
    Patients.ID_Patient,
    Patients.PatientSurname,
    Patients.PatientName,
    Patients.PatientMiddleName,
    Patients.DateOfBirth,
    Patients.Gender_ID,
    Gender.GenderValue
FROM Patients
INNER JOIN Gender ON Patients.Gender_ID = Gender.ID_Gender;
go

CREATE VIEW AppointmentsView2 AS
SELECT
    Patients.PatientSurname + ' ' + SUBSTRING(Patients.PatientName, 1, 1) + '.' + SUBSTRING(Patients.PatientMiddleName, 1, 1) + '.' AS 'PatientFullName',
    Doctors.DoctorSurname + ' ' + SUBSTRING(Doctors.DoctorName, 1, 1) + '.' + SUBSTRING(Doctors.DoctorMiddleName, 1, 1) + '.' AS 'DoctorFullName',
    Appointments.DateAppointment,
    Appointments.DescriptionAppointment,
    Patients.ID_Patient,
    Doctors.ID_Doctor
FROM
    Appointments
INNER JOIN
    Patients ON Appointments.Patient_ID = Patients.ID_Patient
INNER JOIN
    Doctors ON Appointments.Doctor_ID = Doctors.ID_Doctor;
go

create view DoctorsView2 as
select Doctors.DoctorSurname,
    Doctors.DoctorName,
    Doctors.DoctorMiddleName,
    Specializations.SpecializationName
from Doctors
inner join Specializations on Doctors.Specialization_ID = Specializations.ID_Specialization;
go

```

Рисунок 52 - Создание представлений

4. Реализация поиска данных с использованием EF.

Создаём текстовое поле (SearchTextBox) с серым текстом в качестве подсказки, который исчезает при фокусировке на поле или вводе текста. Если текстовое поле не в фокусе и не содержит текста, отображается серый текст подсказки «Введите фамилию пациента».

```

</DataGrid>
<TextBox x:Name="SearchTextBox" Grid.Column="1" Grid.Row="0" Height="32" Width="348" HorizontalAlignment="Left" VerticalAlignment="Top">
  <TextBox.Style>
    <Style TargetType="TextBox">
      <Setter Property="Foreground" Value="Black"/>
      <Setter Property="Template">
        <Setter.Value>
          <ControlTemplate TargetType="{x:Type TextBox}">
            <Grid>
              <ScrollViewer x:Name="PART_ContentHost" />
              <TextBlock x:Name="Placeholder" Text="Введите фамилию пациента"
                Foreground="Gray" Visibility="Collapsed"/>
            </Grid>
            <ControlTemplate.Triggers>
              <MultiTrigger>
                <MultiTrigger.Conditions>
                  <Condition Property="IsFocused" Value="False"/>
                  <Condition Property="Text" Value=""/>
                </MultiTrigger.Conditions>
                <Setter TargetName="Placeholder" Property="Visibility" Value="Visible"/>
              </MultiTrigger>
            </ControlTemplate.Triggers>
          </ControlTemplate>
        </Setter.Value>
      </Setter>
    </Style>
  </TextBox.Style>
</TextBox>

```

Рисунок 53 - Создание текстового поля

Добавляем кнопки «Поиск» и «Очистка».

```

</TextBox>
<Button Content="Поиск" Click="Search_Click" Grid.Column="1" Grid.Row="1" />
<Button Content="Очистка" Click="Clear_Click" Grid.Column="1" Grid.Row="2" />

```

Рисунок 54 - добавление кнопок

Прописываем логику для этих кнопок. Чтобы при нажатии на кнопку «Поиск», отображались только данные, содержащие введенное значение в TextBox напомним условие для поиска: табличка.Where(item => item.[Название столбца].Contains(текстбокс. Text)).

Чтобы при нажатии на кнопку «Очистить», данные после поиска возобновлялись внутри нее нужно просто снова отправить запрос на получение информации.

```

private void Search_Click(object sender, RoutedEventArgs e)
{
    string searchText = SearchTextBox.Text;
    PatientsCbx.SelectedItem = null;
    PatientsViewGrd.ItemsSource = context.PatientsView2.ToList().Where(item => item.PatientSurname.Contains(SearchTextBox.Text));
    SearchTextBox.Text = searchText;
}

Ссылка 1
private void Clear_Click(object sender, RoutedEventArgs e)
{
    SearchTextBox.Text = null;
    PatientsViewGrd.ItemsSource = context.PatientsView2.ToList();
    PatientsCbx.SelectedItem = null;
}

```

Рисунок 55 - методы Search_Click и Clear_Click

То же самое делаем для таблицы DoctorsView2:

```
private void Search_Click(object sender, RoutedEventArgs e)
{
    string searchText = SearchTextBox.Text;
    DoctorsCbx.SelectedItem = null;
    DoctorsViewGrd.ItemsSource = context.DoctorsView2.ToList().Where(item => item.SpecializationName.Contains(SearchTextBox.Text));
    SearchTextBox.Text = searchText;
}

Ссылка 1
private void Clear_Click(object sender, RoutedEventArgs e)
{
    SearchTextBox.Text = null;
    DoctorsViewGrd.ItemsSource = context.DoctorsView2.ToList();
    DoctorsCbx.SelectedItem = null;
}
```

Рисунок 56 - методы Search_Click и Clear_Click

5. Реализация фильтрации данных с использованием EF.

Создаем ComboBox с фамилиями пациентов.

В этот список нужно выгрузить все данные из таблички PatientsView2 + чтобы у нас отображался текст, в качестве DisplayMemberPath укажем название столбца.

```
<ComboBox x:Name="PatientsCbx" DisplayMemberPath="PatientSurname" SelectionChanged="PatientsCbx_SelectionChanged" />
<Button Content="Назад" Click="Back_Click" Grid.Column="1" Grid.Row="3" Width="100" HorizontalAlignment="Right" />
```

Рисунок 57 - Создание комбобокса

```
Ссылка 1
public PatientsW()
{
    InitializeComponent();
    PatientsViewGrd.ItemsSource = context.PatientsView2.ToList();
    PatientsCbx.ItemsSource = context.PatientsView2.ToList();
}

Ссылка 1
```

Рисунок 58 - Выгрузка данных в комбобокс

При изменении выбора в ComboBox (событие SelectionChanged, которое надо создать, дважды нажав по списку), нужно взять объект, который написан выше, и также при помощи LINQ запросов сделать выборку через Where.

Проверим, точно ли у нас есть выбор через SelectionItem != null. Если есть, возьмем объект. По этому объекту делаем выборку – где item.PatientSurname (фамилия пациента) равен выбранной фамилии. Результат поместим в датагрид.

```
Ссылка 1
private void PatientsCbx_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    SearchTextBox.Text = null;
    var selectedPatient = PatientsCbx.SelectedItem as PatientsView2;
    if (selectedPatient != null)
    {
        PatientsViewGrd.ItemsSource = context.PatientsView2.ToList().Where(item => item.PatientSurname == selectedPatient.PatientSurname);
    }
}
```

Рисунок 59 – SelectionChanged

То же самое делаем для таблицы DoctorsView2:

```
<ComboBox x:Name="DoctorsCbx" DisplayMemberPath="SpecializationName" SelectionChanged="DoctorsCbx_SelectionChanged"/>
```

Рисунок 60 - Создание комбобокса

```
public DoctorsW()  
{  
    InitializeComponent();  
    DoctorsCbx.ItemsSource = context.DoctorsView2.ToList();  
    DoctorsViewGrd.ItemsSource = context.DoctorsView2.ToList();  
}
```

Рисунок 61 - Выгрузка данных в комбобокс

```
private void DoctorsCbx_SelectionChanged(object sender, SelectionChangedEventArgs e)  
{  
    SearchTextBox.Text = null;  
    var selectedDoctor = DoctorsCbx.SelectedItem as DoctorsView2;  
    if (selectedDoctor != null)  
    {  
        DoctorsViewGrd.ItemsSource = context.DoctorsView2.ToList().Where(item => item.SpecializationName == selectedDoctor.SpecializationName)  
    }  
}
```

Рисунок 62 - SelectionChanged

6. Результат поиска данных с использованием EF.

Фамилия	Имя	Отчество	Дата рождения
Крылов	Даниил	Александрович	05.08.1976

Рисунок 63 – Поиск

Фамилия	Имя	Отчество	Дата рождения
Родионов	Аркадий	Федотович	21.08.2002

Рисунок 64 – Фильтрация

7. Реализация поиска и фильтрации данных с использованием DataSet.

Создаём запрос в таблице AppointmentsView2. Поиск будет по столбцу с фамилией и инициалами врача. Используем оператор поиска LIKE:

Введите инструкцию SQL Select или воспользуйтесь конструктором проектирования запроса. Какие данные должны быть загружены?

Данные для загрузки в таблицу

```
SELECT *  
FROM [AppointmentsView2]  
WHERE [DoctorFullName] LIKE '%'+@toSearch+'%'
```

Рисунок 65 - Создание запроса

Далее создаем метод и даем ему название:

Имя метода:

☒ **Вернуть таблицу данных (DataTable)**

Создает метод, который возвращает новую таблицу данных, заполненную результатами исполнения инструкции SQL или хранимой процедуры SELECT, введенной на предыдущей странице.

Имя метода:

SearchByDoctor

Рисунок 66 - метод SearchByDoctor

При нажатии на кнопку «Поиск» обновим таблицу AppointmentsViewDgr этим методом. Внутри напишем текст из текстового поля.

При нажатии на кнопку «Очистить» внутри нее нужно просто снова отправить запрос на получение информации.

```
Ссылка 1  
private void Search_Click(object sender, RoutedEventArgs e)  
{  
    AppointmentsViewGrd.ItemsSource = priems.SearchByDoctor(SearchTextBox.Text);  
}  
  
Ссылка 1  
private void Clear_Click(object sender, RoutedEventArgs e)  
{  
    SearchTextBox.Text = null;  
    AppointmentsViewGrd.ItemsSource = priems.GetData();  
}
```

Рисунок 67 - методы Search_Click и Clear_Click

8. Реализация фильтрации данных с использованием DataSet.

Создаём новый запрос:

проектирования запроса. Какие данные должны быть загружены?

Данные для загрузки в таблицу

```
SELECT *  
FROM [AppointmentsView2]  
WHERE [DoctorFullName] = @DoctorFullName
```

Рисунок 68 - Создание запроса

Далее создаем метод и даем ему название:

Имя метода:

☒ **Вернуть таблицу данных (DataTable)**

Создает метод, который возвращает новую таблицу данных, заполненную результатами исполнения инструкции SQL или хранимой процедуры SELECT, введенной на предыдущей странице.

Имя метода:

Рисунок 69 - метод FilterByDoc

Создаем комбобокс:

```
<Button Content="Очистка" Click="Clear_Click" Grid.Column="1" Grid.Row="1" Width="100" Background="#FF9DBA9F"
<ComboBox x:Name="DoctorCbx" DisplayMemberPath="DoctorFullName" SelectionChanged="DoctorCbx_SelectionChanged"
<Button Content="Назад" Grid.Column="2" Grid.Row="3" Click="Back_Click" Width="100" HorizontalAlignment="Right"/>
```

Рисунок 70 - Создание комбобокса

Выгружаем данные в него:

```
public AppointmentsW()
{
    InitializeComponent();
    AppointmentsViewGrd.ItemsSource = priems.GetData();
    DoctorCbx.ItemsSource = priems.GetData();
}
```

Рисунок 71 - Выгрузка данных в комбобокс

При изменении выбора в ComboBox (событие SelectionChanged) получаем полное имя выбранного врача и передаем его в метод "FilterByDoc" для фильтрации записей в гриде "AppointmentsViewGrd".

```
private void DoctorCbx_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    SearchTextBox.Text = null;
    string doctorFullName = (DoctorCbx.SelectedItem as DataRowView)["DoctorFullName"].ToString();
    AppointmentsViewGrd.ItemsSource = priems.FilterByDoc(doctorFullName);
}
```

Рисунок 72 – SelectionChanged

9. Результат поиска и фильтрации данных с использованием DataSet.

Приемы				Бере
ФИО пациента	ФИО врача	Дата приема	Опис	
Крылов Д.А.	Березина К..	17/03/2024 15:45	Пров	

Рисунок 73 – Поиск

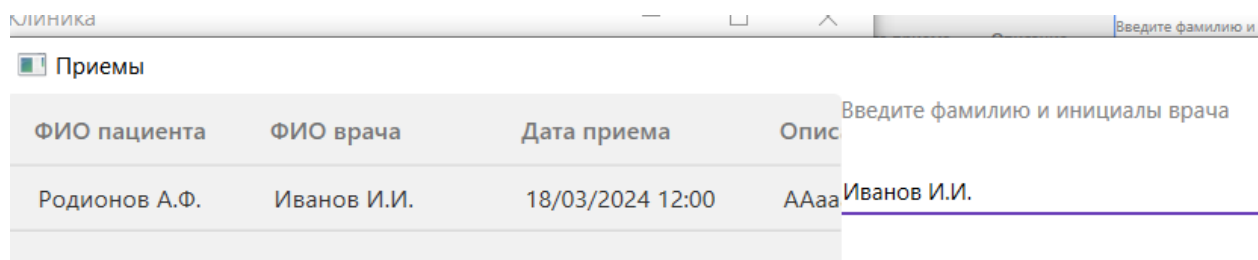


Рисунок 74 – Фильтрация

Вывод: реализовали базу данных из 5 таблиц, связанных между собой, и подключили к ней WPF проект, используя EntityFramework и DataSet, данные читаются в DataGridView, также реализовали поиск и фильтрацию по одному столбцу в каждой таблице.

ЛАБОРАТОРНАЯ РАБОТА №5

Разработка информационной системы

Цель работы: создать базу данных по предметной области «Магазин мебели», содержащую в себе 10-15 таблиц и нормализованную согласно 3 НФ. Также создать WPF-проект согласно техническому заданию и присоединить к нему базу данных для их взаимодействия.

1. Создание базы данных.

1.1. Создаем 12 таблиц, связанных между собой.

Таблица "Roles" (Роли):

- Эта таблица создается для хранения различных ролей пользователей в системе. У каждой роли есть уникальный идентификатор (ID_Role) и название роли (RoleName).

Таблица "Users" (Пользователи):

- В данной таблице хранится информация о пользователях. Каждый пользователь имеет свой уникальный идентификатор (ID_User), фамилию, имя, отчество, логин, хэш пароля, а также связь с ролью через поле Role_ID, которое ссылается на ID роли в таблице "Roles".

Таблица "Discounts" (Скидки):

- Эта таблица предназначена для хранения информации о скидках. У скидки есть уникальный идентификатор (ID_Discount), процент скидки (PercentageDiscount) и название скидки (DiscountName).

Таблица "FurnitureCategories" (Категории мебели):

- В этой таблице содержатся различные категории мебели. Каждая категория имеет уникальный идентификатор (ID_Category) и название категории (CategoryName).

Таблица "Manufacturers" (Производители):

- Данная таблица предназначена для хранения информации о производителях мебели. У каждого производителя есть свой уникальный

идентификатор (ID_Manufacturer), название производителя (ManufacturerName) и страна происхождения (Country).

```
-- Создание базы данных
CREATE DATABASE FurnitureStore3;
GO

-- Использование базы данных
USE FurnitureStore3;
GO

-- Таблица Ролей
CREATE TABLE Roles (
    ID_Role INT PRIMARY KEY IDENTITY(1,1),
    RoleName VARCHAR(50) NOT NULL
);

-- Таблица Пользователей
CREATE TABLE Users (
    ID_User INT PRIMARY KEY IDENTITY(1,1),
    UserSurname VARCHAR(50) NOT NULL,
    UserName VARCHAR(50) NOT NULL,
    UserMiddleName VARCHAR(50),
    UserLogin VARCHAR(50) NOT NULL,
    PasswordHash VARCHAR(300) NOT NULL,
    Role_ID INT NOT NULL,
    FOREIGN KEY (Role_ID) REFERENCES Roles(ID_Role)
);

-- Таблица Скидок
CREATE TABLE Discounts (
    ID_Discount INT PRIMARY KEY IDENTITY(1,1),
    PercentageDiscount INT NOT NULL,
    DiscountName VARCHAR(50)
);

-- Таблица Категорий мебели
CREATE TABLE FurnitureCategories (
    ID_Category INT PRIMARY KEY IDENTITY(1,1),
    CategoryName VARCHAR(100) NOT NULL
);

-- Таблица Производителей
CREATE TABLE Manufacturers (
    ID_Manufacturer INT PRIMARY KEY IDENTITY(1,1),
    ManufacturerName VARCHAR(100) NOT NULL,
    Country VARCHAR(100) NOT NULL
);
```

Рисунок 75 - Создание таблиц

Таблица "Furniture" (Мебель):

- Данная таблица используется для хранения информации о различных предметах мебели. У каждой мебели есть уникальный идентификатор (ID_Furniture), название, описание, цена, ссылки на категорию (Category_ID), производителя (Manufacturer_ID) и скидку (Discount_ID) через внешние ключи, которые связываются с соответствующими таблицами FurnitureCategories, Manufacturers и Discounts.

Таблица "Services" (Услуги):

- В этой таблице хранится информация о различных услугах, которые могут быть предоставлены мебельным магазином. Каждая услуга имеет уникальный идентификатор (ID_Service), название, описание и цену.

Таблица "Orders" (Заказы):

- В данной таблице записываются данные о заказах. Каждый заказ имеет уникальный идентификатор (ID_Order), идентификатор пользователя (Userr_ID), дату заказа (OrderDate), общую сумму заказа (SumMoneyClient) и сумму сдачи (Changee). Поле Userr_ID связано с таблицей пользователей (Users) через внешний ключ.

Таблица "OrderDetails" (Детали заказа):

- Эта таблица используется для хранения деталей о товарах в каждом заказе. Каждая запись содержит уникальный идентификатор (ID_OrderDetail), идентификатор заказа (Order_ID), количество товара, цену за единицу товара. Поле Order_ID связано с таблицей заказов (Orders) через внешний ключ.

```

-- Таблица Мебели
CREATE TABLE Furniture (
    ID_Furniture INT PRIMARY KEY IDENTITY(1,1),
    FurnitureName VARCHAR(100) NOT NULL,
    DescriptionFurniture TEXT,
    Price DECIMAL(10,2) NOT NULL,
    Category_ID INT,
    Manufacturer_ID INT,
    Discount_ID INT,
    FOREIGN KEY (Category_ID) REFERENCES FurnitureCategories(ID_Category),
    FOREIGN KEY (Manufacturer_ID) REFERENCES Manufacturers(ID_Manufacturer),
    FOREIGN KEY (Discount_ID) REFERENCES Discounts(ID_Discount)
);

-- Таблица Услуг
CREATE TABLE Services (
    ID_Service INT PRIMARY KEY IDENTITY(1,1),
    ServiceName VARCHAR(100) NOT NULL,
    DescriptionService TEXT,
    Price DECIMAL(10,2) NOT NULL
);

-- Таблица Заказов
CREATE TABLE Orders (
    ID_Order INT PRIMARY KEY IDENTITY(1,1),
    Userr_ID INT,
    OrderDate DATETIME NOT NULL,
    SumMoneyClient DECIMAL(10,2) NOT NULL,
    Changee DECIMAL(10,2),
    FOREIGN KEY (Userr_ID) REFERENCES Users(ID_User)
);

-- Таблица Детали заказа
CREATE TABLE OrderDetails (
    ID_OrderDetail INT PRIMARY KEY IDENTITY(1,1),
    Order_ID INT,
    Quantity INT NOT NULL,
    Price DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (Order_ID) REFERENCES Orders(ID_Order)
);

```

Рисунок 76 - Создание таблиц

Таблица "Sklad" (Склад):

- Данная таблица используется для отслеживания количества товаров на складе. Включает в себя уникальный идентификатор склада (ID_Sklad), количество товара (Quantity), идентификатор товара (Furniture_ID) и идентификатор пользователя, который отвечает за управление складом (Userr_ID). Внешние ключи связаны с таблицами Furniture (Мебель) и Users (Пользователи).

Таблица "OrderFurniture" (Заказанная мебель):

- Эта таблица хранит информацию о заказанной мебели в рамках заказа. Каждая запись содержит уникальный идентификатор (ID_OrderFurniture), идентификатор детали заказа (OrderDetail_ID), идентификатор товара (Furniture_ID), количество заказанной мебели. Внешние ключи связаны с таблицами OrderDetails (Детали заказа) и Furniture (Мебель).

Таблица "OrderServices" (Заказанные услуги):

- В данной таблице хранится информация о заказанных услугах в рамках заказа. Каждая запись содержит уникальный идентификатор (ID_OrderServices), идентификатор детали заказа (OrderDetail_ID), идентификатор услуги (Service_ID), количество заказанных услуг. Внешние ключи связаны с таблицами OrderDetails (Детали заказа) и Services (Услуги).

```
-- Таблица Склада
CREATE TABLE Sklad (
    ID_Sklad INT PRIMARY KEY IDENTITY(1,1),
    Quantity INT NOT NULL,
    Furniture_ID INT NOT NULL,
    Userr_ID INT NOT NULL,
    FOREIGN KEY (Furniture_ID) REFERENCES Furniture(ID_Furniture),
    FOREIGN KEY (Userr_ID) REFERENCES Users(ID_User)
);

CREATE TABLE OrderFurniture (
    ID_OrderFurniture INT PRIMARY KEY IDENTITY(1,1),
    OrderDetail_ID INT,
    Furniture_ID INT,
    Quantity INT NOT NULL,
    FOREIGN KEY (OrderDetail_ID) REFERENCES OrderDetails(ID_OrderDetail),
    FOREIGN KEY (Furniture_ID) REFERENCES Furniture(ID_Furniture)
);

CREATE TABLE OrderServices (
    ID_OrderServices INT PRIMARY KEY IDENTITY(1,1),
    OrderDetail_ID INT,
    Service_ID INT,
    Quantity INT NOT NULL,
    FOREIGN KEY (OrderDetail_ID) REFERENCES OrderDetails(ID_OrderDetail),
    FOREIGN KEY (Service_ID) REFERENCES Services(ID_Service)
);
```

Рисунок 77 - Создание таблиц

1.2. Создание физической и логической модели данных.

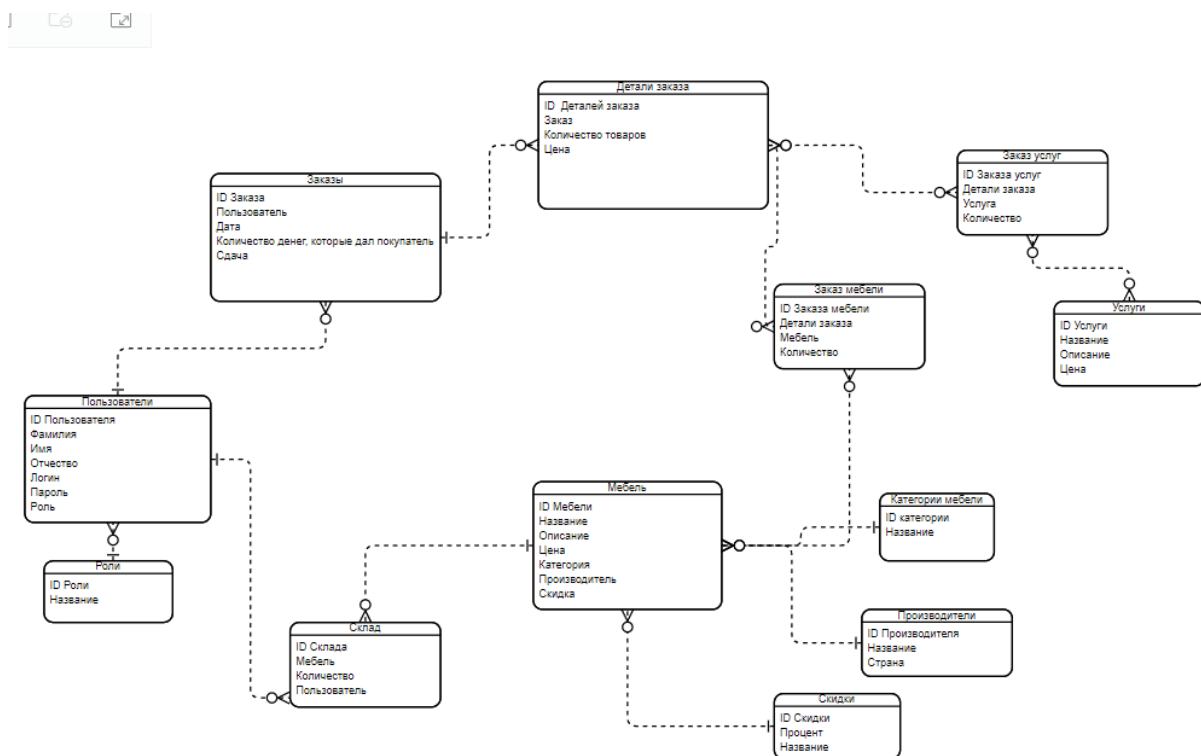
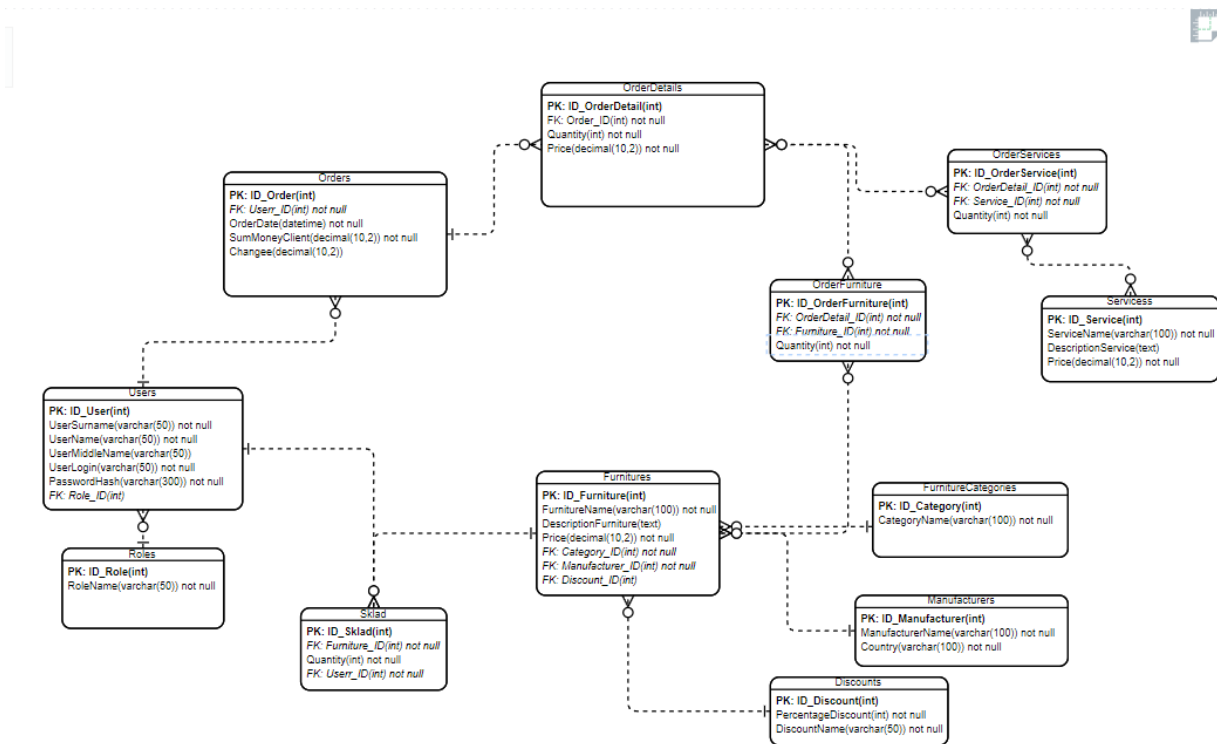


Рисунок 78 - Логическая модель данных



```

INSERT INTO Roles(RoleName)
VALUES
('Администратор'),
('Кассир'),
('Склад-менеджер');
GO

INSERT INTO Users(UserSurname, UserName, UserMiddleName, UserLogin, PasswordHash, Role_ID)
VALUES
('Иванов', 'Иван', 'Иванович', 'admin', 'qqq', 1),
('Петров', 'Петр', 'Петрович', 'kassir', 'www', 2),
('Мурлаева', 'Екатерина', '', 'sklad', 'eee', 3);
GO

INSERT INTO Discounts(PercentageDiscount, DiscountName)
VALUES
(50, 'Товар с витрины'),
(25, 'Акция');
GO

INSERT INTO FurnitureCategories(CategoryName)
VALUES
('Столы'),
('Стулья'),
('Шкафы'),
('Диваны');

INSERT INTO Manufacturers(ManufacturerName, Country)
VALUES
('IKEA', 'Швеция'),
('ООО "Кресла и диваны"', 'Россия'),
('Фабрика "Шкафы-Люкс"', 'Россия'),
('ООО "Уютная мебель"', 'Россия');

INSERT INTO Furniture(FurnitureName, DescriptionFurniture, Price, Category_ID, Manufacturer_ID, Discount_ID)
VALUES
('Стол письменный', 'Стол письменный с ящиком', 15000.00, 1, 1, 1),
('Стол обеденный', 'Деревянный стул с обивкой', 5000.00, 4, 2, NULL),
('Шкаф для одежды', 'Двустворчатый шкаф с зеркалом', 25000.00, 1, 3, NULL),
('Диван угловой', 'Угловой диван с подушками', 35000.00, 2, 1, 2);

```

Рисунок 80 - Заполнение таблиц

```

INSERT INTO Servicess(ServiceName, DescriptionService, Price)
VALUES
('Доставка', 'Доставка до двери', 2000.00),
('Сборка', 'Сборка мебели на месте', 3000.00);

INSERT INTO Orders(Userr_ID, OrderDate, SumMoneyClient, Changee)
VALUES
(2, '15.03.2024 09:00', 50000.00, 0),
(2, '16.03.2024 10:30', 35000.00, 5000.00),
(2, '17.03.2024 15:45', 25000.00, 0);

INSERT INTO OrderDetails(Order_ID, Quantity, Price)
VALUES
(4, 20, 30000.00),
(5, 10, 35000.00),
(6, 30, 75000.00);

INSERT INTO OrderFurniture(OrderDetail_ID, Furniture_ID, Quantity)
VALUES
(3, 1, 2),
(4, 4, 1),
(5, 3, 2);

INSERT INTO OrderServices(OrderDetail_ID, Service_ID, Quantity)
VALUES
(3, 1, 1),
(4, 2, 1),
(5, 1, 1);

INSERT INTO Sklad(Quantity, Furniture_ID, Userr_ID)
VALUES
(100, 1, 3),
(100, 2, 3),
(100, 3, 3),
(100, 4, 3);

```

Рисунок 81 - Заполнение таблиц

```

INSERT INTO Orders (Userr_ID, OrderDate, SumMoneyClient, Changee)
VALUES
(2, '20.03.2024 09:00', 45000.00, 0),
(2, '21.03.2024 10:30', 30000.00, 0),
(2, '25.03.2024 15:45', 20000.00, 0);

INSERT INTO OrderDetails (Order_ID, Quantity, Price)
VALUES
(10, 2, 15000.00),
(11, 3, 5000.00),
(10, 1, 10000.00),
(12, 2, 8000.00);

INSERT INTO OrderFurniture (OrderDetail_ID, Furniture_ID, Quantity)
VALUES
(7, 1, 2),
(8, 2, 3),
(9, 3, 1),
(10, 4, 2);

INSERT INTO OrderServices (OrderDetail_ID, Service_ID, Quantity)
VALUES
(7, 1, 1),
(7, 1, 1),
(8, 1, 1),
(9, 1, 1);

```

Рисунок 82 - Заполнение таблиц

2. Подключаем бд к WPF.

2.1. Создаем разные окна и страницы для разных таблиц.

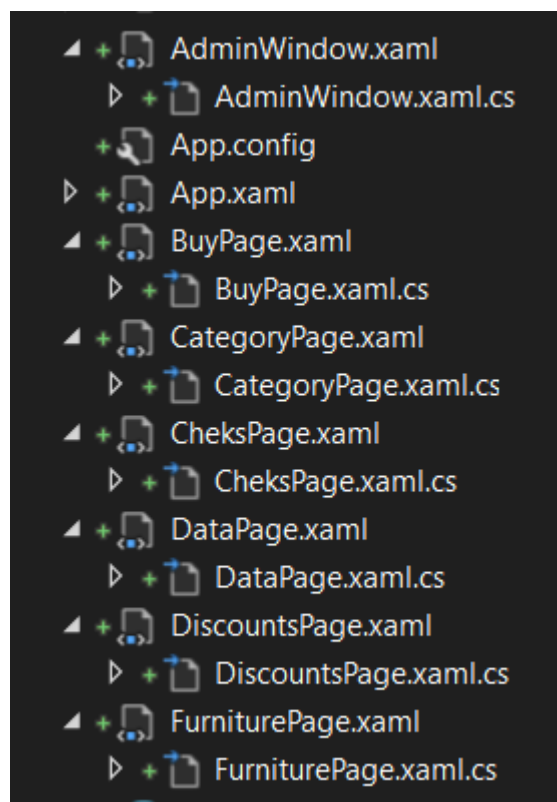


Рисунок 83 - Создание окон и страниц

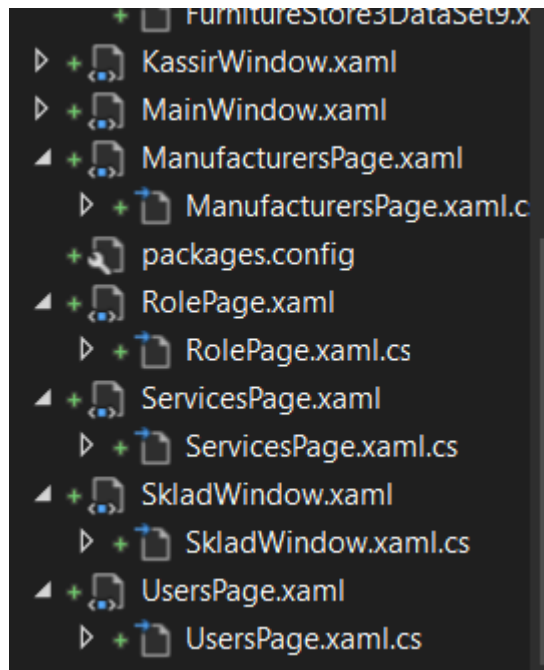


Рисунок 84 - Создание окон и страниц

2.2. Данные из БД должны читаться в DataGrid.

```
<DataGrid x:Name="RoleGrd" AutoGenerateColumns="False" SelectionChanged="RoleGrd_SelectionChanged"
    <DataGrid.Columns>
        <DataGridTextColumn Header="Название роли" Binding="{Binding RoleName}" /
    </DataGrid.Columns>
</DataGrid>
```

Рисунок 85 – DataGrid

Используем DataSet для чтения данных из таблицы.

```
Ссылка: 4
public partial class RolePage : Page
{
    private RolesTableAdapter role = new RolesTableAdapter();

    Ссылка: 2
    public RolePage()
    {
        InitializeComponent();
        RoleGrd.ItemsSource = role.GetData();
    }
}
```

Рисунок 86 - Использование DataSet

2.3. В программе должна быть авторизация по ролям. Создаем 3 роли: администратор, кассир, склад-менеджер. У разных ролей доступ к разным

таблицам - условный пользователь не может посмотреть таблицы администратора.

В MainWindow.xaml создаем окно для авторизации.

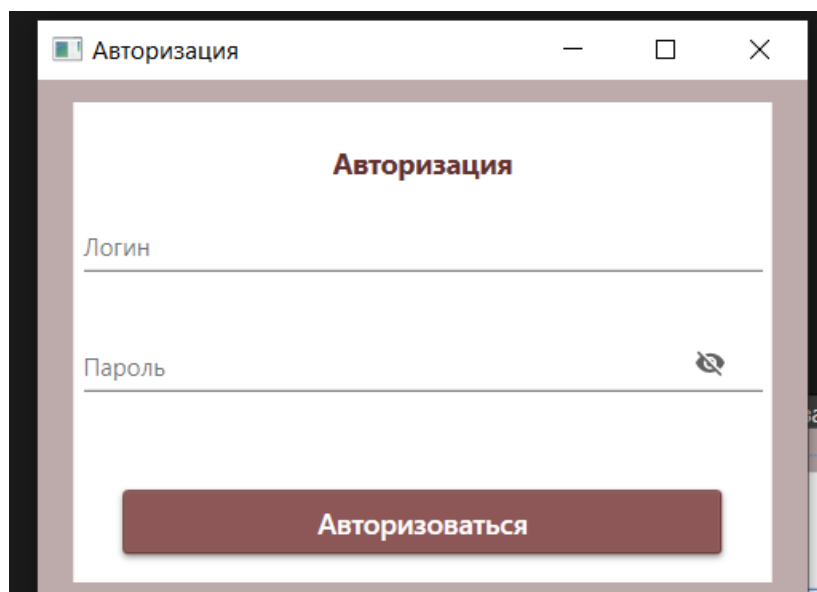


Рисунок 87 - Окно для авторизации

Прописываем логику для кнопки «Авторизоваться».

Этот фрагмент кода представляет метод, который выполняется при нажатии на кнопку.

```
`var alluser = user.GetData().Rows;`
```

- Переменная `alluser` получает данные о пользователях из базы данных в виде коллекции строк (Rows).

```
`for (int i = 0; i < alluser.Count; i++)`
```

- Запускается цикл перебора всех строк данных о пользователях.

```
`if      (alluser[i][4].ToString()      ==      LoginTbx.Text      &&  
ComputeHash>PasswordTbx.Password) == alluser[i][5].ToString())`
```

- В данном условии проверяется, соответствует ли введенный пользователем логин (хранится в элементе 4 строки данных) и пароль каким-либо учетным данным пользователя в базе данных.

```
`int roleid = (int)alluser[i][6];`
```

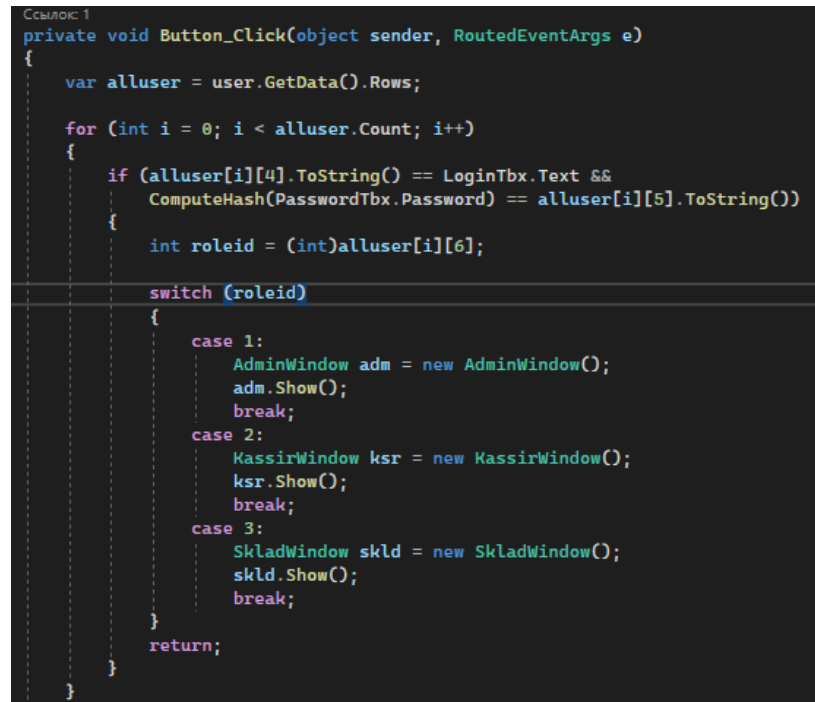
- Вытаскивается идентификатор роли пользователя из данных about данного пользователя.

```
`switch (roleid) { case 1: ... case 2: ... case 3: ... }`
```

- В зависимости от идентификатора роли пользователя программа определяет, какое окно (AdminWindow, KassirWindow или SkladWindow) нужно показать пользователю.

```
`return;`
```

При успешной авторизации и определении роли пользователя выводится соответствующее окно, и метод завершается.

The image shows a snippet of C# code within a method named Button_Click. It starts with a variable 'alluser' assigned to 'user.GetData().Rows'. A 'for' loop iterates through 'alluser.Count'. Inside the loop, an 'if' statement checks if 'alluser[i][4].ToString()' equals 'LoginTbx.Text' and 'ComputeHash>PasswordTbx.Password)' equals 'alluser[i][5].ToString()'. If true, 'roleid' is set to '(int)alluser[i][6]'. A 'switch' statement follows, with three cases: 'case 1' creates and shows 'AdminWindow', 'case 2' creates and shows 'KassirWindow', and 'case 3' creates and shows 'SkladWindow'. Each case ends with 'break;'. The method concludes with 'return;' and a closing brace. The code is highlighted with a dark background and light text, with some lines underlined.

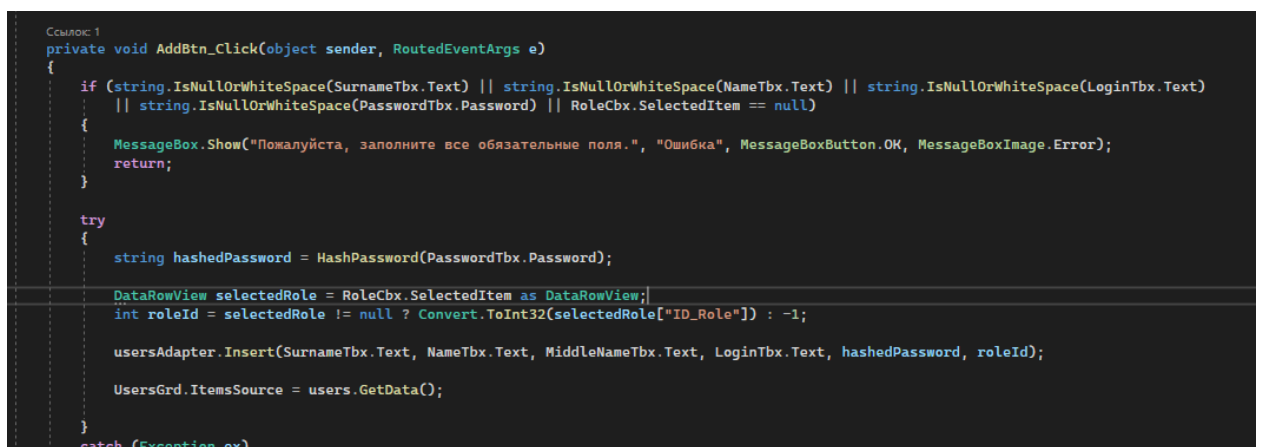
```
Ссылка 1
private void Button_Click(object sender, RoutedEventArgs e)
{
    var alluser = user.GetData().Rows;

    for (int i = 0; i < alluser.Count; i++)
    {
        if (alluser[i][4].ToString() == LoginTbx.Text &&
            ComputeHash>PasswordTbx.Password)' == alluser[i][5].ToString())
        {
            int roleid = (int)alluser[i][6];

            switch (roleid)
            {
                case 1:
                    AdminWindow adm = new AdminWindow();
                    adm.Show();
                    break;
                case 2:
                    KassirWindow ksr = new KassirWindow();
                    ksr.Show();
                    break;
                case 3:
                    SkladWindow skld = new SkladWindow();
                    skld.Show();
                    break;
            }
        }
    }
    return;
}
```

Рисунок 88 - Метод для кнопки "Авторизоваться"

2.4. Регистрировать пользователя должен администратор. Данные пользователя идут в БД.

The image shows a snippet of C# code within a method named AddBtn_Click. It begins with an 'if' statement checking for null or white space in 'SurnameTbx.Text', 'NameTbx.Text', 'LoginTbx.Text', and 'PasswordTbx.Password'. If any are null, a 'MessageBox.Show' is called with an error message, and the method returns. A 'try' block follows, where 'hashedPassword' is calculated using 'HashPassword'. Then, 'selectedRole' is assigned to 'RoleCbx.SelectedItem as DataRowView', and 'roleId' is converted from 'selectedRole["ID_Role"]' to an integer. The 'usersAdapter.Insert' method is called with the user details and 'roleId'. 'UsersGrd.ItemsSource' is set to 'users.GetData()'. The 'try' block ends with a 'catch' statement for 'Exception ex'. The code is highlighted with a dark background and light text, with some lines underlined.

```
Ссылка 1
private void AddBtn_Click(object sender, RoutedEventArgs e)
{
    if (string.IsNullOrEmpty(SurnameTbx.Text) || string.IsNullOrEmpty(NameTbx.Text) || string.IsNullOrEmpty(LoginTbx.Text)
        || string.IsNullOrEmpty>PasswordTbx.Password)' || RoleCbx.SelectedItem == null)
    {
        MessageBox.Show("Пожалуйста, заполните все обязательные поля.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
        return;
    }

    try
    {
        string hashedPassword = HashPassword>PasswordTbx.Password);

        DataRowView selectedRole = RoleCbx.SelectedItem as DataRowView;
        int roleId = selectedRole != null ? Convert.ToInt32(selectedRole["ID_Role"]) : -1;

        usersAdapter.Insert(SurnameTbx.Text, NameTbx.Text, MiddleNameTbx.Text, LoginTbx.Text, hashedPassword, roleId);

        UsersGrd.ItemsSource = users.GetData();
    }
    catch (Exception ex)
    {
    }
}
```

Рисунок 89 - Регистрация пользователя

2.5. Все пароли должны быть скрыты точками.

Используем PasswordBox.

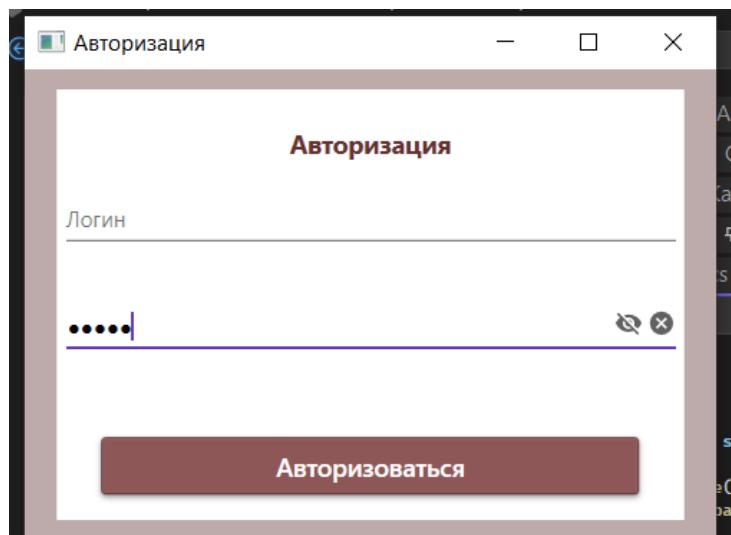


Рисунок 90 - Скрытие пароля точками

Данные для заполнения

Введите фамилию

Введите имя

Введите отчество

Выберите роль

Логин

.....

Рисунок 91 - Скрытие пароля точками

2.6. Должно быть предусмотрено хеширование паролей для безопасности хранения информации в БД.

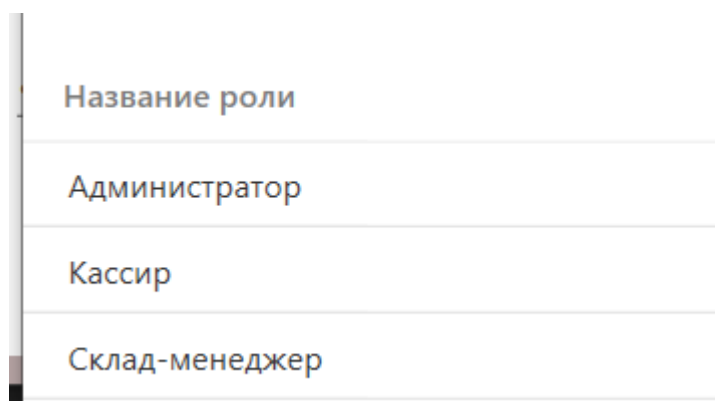
Создаём хеш-функцию.

```
Ссылка: 1
private string ComputeHash(string input)
{
    using (SHA256 sha256 = SHA256.Create())
    {
        byte[] bytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(input));

        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < bytes.Length; i++)
        {
            builder.Append(bytes[i].ToString("x2"));
        }
        return builder.ToString();
    }
}
```

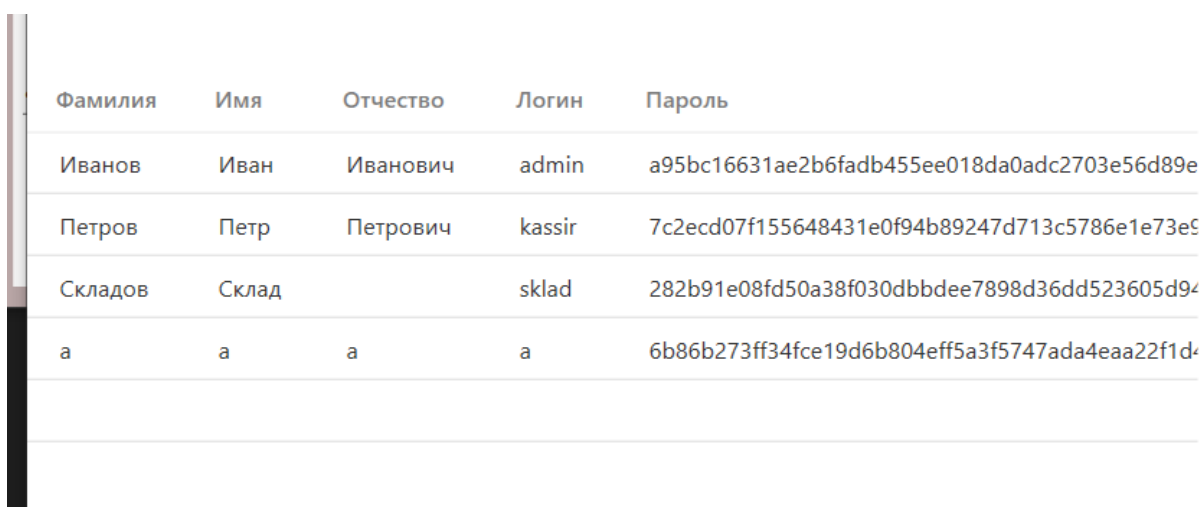
Рисунок 92 - Хеш-функция

2.7. Должна быть возможность читать данные из любой таблицы.



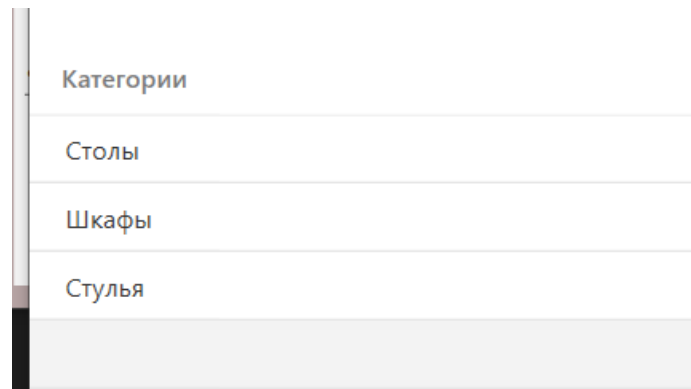
Название роли
Администратор
Кассир
Склад-менеджер

Рисунок 93 - Чтение данных из таблицы Roles



Фамилия	Имя	Отчество	Логин	Пароль
Иванов	Иван	Иванович	admin	a95bc16631ae2b6fadb455ee018da0adc2703e56d89e
Петров	Петр	Петрович	kassir	7c2ecd07f155648431e0f94b89247d713c5786e1e73e9
Складов	Склад		sklad	282b91e08fd50a38f030dbbdee7898d36dd523605d94
а	а	а	а	6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d4

Рисунок 94 - Чтение данных из таблицы Users

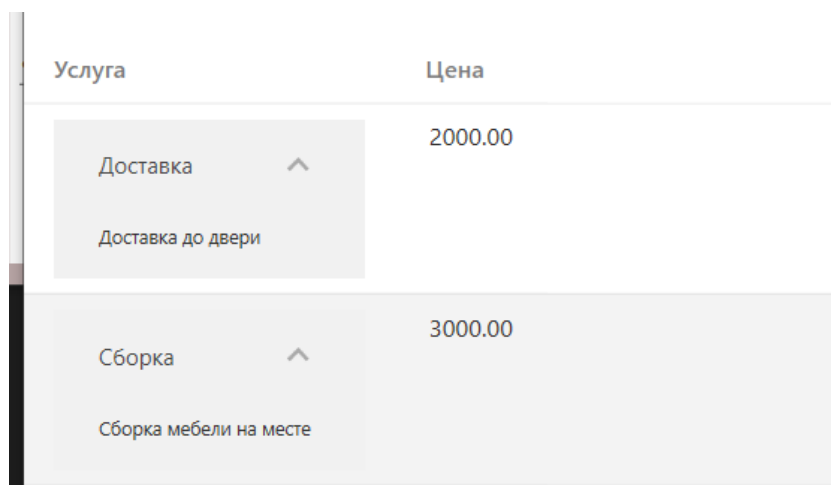


Категории
Стол
Шкаф
Стул

Рисунок 95 - Чтение данных из таблицы FurnitureCategories

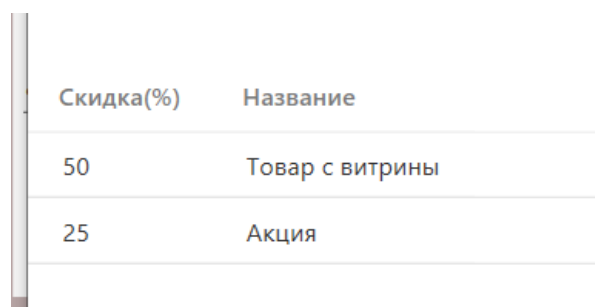
Производитель	Страна
ИКЕА	Швеция
ООО "Кресла и диваны"	Россия
Фабрика "Шкафы-Люкс"	Россия
ООО "Уютная мебель"	Россия

Рисунок 96 - Чтение данных из таблицы Manufacturers



Услуга	Цена
Доставка	2000.00
Доставка до двери	
Сборка	3000.00
Сборка мебели на месте	

Рисунок 97 - Чтение данных из таблицы Servicess



Скидка(%)	Название
50	Товар с витрины
25	Акция

Рисунок 98 - Чтение данных из таблицы Discounts

Производитель	Название мебели	Количество	Цена	Категория
IKEA	<div> <div>Стол письменный</div> <div>Стол письменный с ящиком</div> </div>	100	15000.00	Столы
Фабрика "Шкафы-Люкс"	<div> <div>Шкаф для одежды</div> <div>Двухстворчатый шкаф с зеркалом</div> </div>	100	25000.00	Столы

Рисунок 99 - Чтение данных из таблиц Furniture и Sklad

Номер заказа	Дата заказа	Название мебели	Количество мебели	Цена мебели (руб)	Скидка (%)	Количество услуг	Название услуги	Итоговая цена (руб)	Внесено (руб)	Сдача
11	3/21/2024 10:30:00 AM	Стул обеденный	3	5000.00	0	1	Доставка	17000.00	30000.00	13000.00
10	3/20/2024 9:00:00 AM	Шкаф для одежды	1	25000.00	0	1	Доставка	27000.00	45000.00	18000.00

Рисунок 100 - Чтение данных из таблиц: Orders, OrderDetails, OrderServices, OrderFurniture

2.8. Должна быть возможность добавлять данные в любую таблицу.

Для добавления данных в таблицы используем встроенный запрос INSERT.

```

Ссылка 1
private void AddBtn_Click(object sender, RoutedEventArgs e)
{
    string roleName = RoleTbx.Text.Trim();
    if (!string.IsNullOrEmpty(roleName))
    {
        try
        {
            role.Insert(roleName);
            RoleGrd.ItemsSource = role.GetData();
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Ошибка при добавлении роли: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }
    else
    {
        MessageBox.Show("Введите название роли", "Предупреждение", MessageBoxButton.OK, MessageBoxImage.Warning);
    }
}

```

Рисунок 101 - Добавление в таблицу

2.9. Должна быть возможность изменять данные из любой таблицы.

Для изменения данных в таблицах используем встроенный запрос UPDATE.

```

Ссылка: 1
private void UpdateBtn_Click(object sender, RoutedEventArgs e)
{
    string roleName = RoleTbx.Text.Trim();
    if (!string.IsNullOrEmpty(roleName) && RoleGrd.SelectedItem != null)
    {
        try
        {
            DataRowView rowView = (DataRowView)RoleGrd.SelectedItem;
            FurnitureStore3DataSet.RolesDataTable rolesDataTable = role.GetData();
            FurnitureStore3DataSet.RolesRow rowToUpdate = rolesDataTable.FindByID_Role(Convert.ToInt32(rowView["ID_Role"]));
            if (rowToUpdate != null)
            {
                rowToUpdate.RoleName = roleName;
                role.Update(rowToUpdate);
                RoleGrd.ItemsSource = role.GetData();
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Ошибка при обновлении роли: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }
    else
    {
        MessageBox.Show("Выберите роль для изменения и введите новое название роли", "Предупреждение", MessageBoxButton.OK, MessageBoxImage.Warning);
    }
}

```

Рисунок 102 - Изменение данных в таблице

2.10. Должна быть возможность удаления данных из таблиц.

Для удаления данных из таблиц используем встроенный запрос DELETE.

```

Ссылка: 1
private void DeleteBtn_Click(object sender, RoutedEventArgs e)
{
    if (RoleGrd.SelectedItem != null)
    {
        try
        {
            DataRowView rowView = (DataRowView)RoleGrd.SelectedItem;
            int roleId = Convert.ToInt32(rowView["ID_Role"]);
            string roleName = rowView["RoleName"].ToString();
            role.Delete(roleId, roleName);
            RoleGrd.ItemsSource = role.GetData();
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Ошибка при удалении роли: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }
    else
    {
        MessageBox.Show("Выберите роль для удаления", "Предупреждение", MessageBoxButton.OK, MessageBoxImage.Warning);
    }
}

```

Рисунок 103 - Удаление данных из таблиц

2.11. Данные из таблиц должны выгружаться в текстовые поля, чтобы их можно было изменить.

Используем SelectionChanged.

```

Ссылка: 1
private void RoleGrd_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (RoleGrd.SelectedItem != null)
    {
        DataRowView rowView = (DataRowView)RoleGrd.SelectedItem;
        DataRow row = rowView.Row;

        RoleTbx.Text = row["RoleName"].ToString();
    }
}

```

Рисунок 104 - Заполнение текстовых полей при нажатии на строку в DataGrid

2.12. Если поле внешний ключ, то данные должны добавляться через выпадающие списки.

Добавляем ComboBox.

```
Ссылка: 1
public UsersPage()
{
    InitializeComponent();
    UsersGrd.ItemsSource = users.GetData();
    RoleCbх.ItemsSource = role.GetData();
}
```

Рисунок 105 - Добавление выпадающего списка

2.14. Должна быть нормальная навигация между окнами и страницами.

Создаем кнопки.

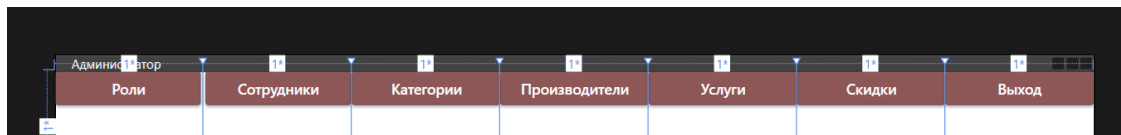


Рисунок 106 - Создание кнопок

Прописываем логику.

```
Ссылка: 1
private void RoleBtn_Click(object sender, RoutedEventArgs e)
{
    PageFrame.Content = new RolePage();
}

Ссылка: 1
private void UsersBtn_Click(object sender, RoutedEventArgs e)
{
    PageFrame.Content = new UsersPage();
}

Ссылка: 0
private void DataBtn_Click(object sender, RoutedEventArgs e)
{
    PageFrame.Content = new DataPage();
}

Ссылка: 1
private void BackBtn_Click(object sender, RoutedEventArgs e)
{
    this.Close();
}

Ссылка: 1
private void DiscountsBtn_Click(object sender, RoutedEventArgs e)
{
    PageFrame.Content = new DiscountsPage();
}

Ссылка: 1
private void CategoryBtn_Click(object sender, RoutedEventArgs e)
{
    PageFrame.Content = new CategoryPage();
}

Ссылка: 1
private void ManufacturersBtn_Click(object sender, RoutedEventArgs e)
{
    PageFrame.Content = new ManufacturersPage();
}

Ссылка: 1
private void ServiceBtn_Click(object sender, RoutedEventArgs e)
{
    PageFrame.Content = new ServicesPage();
}
}
```

Рисунок 107 - Методы для кнопок

Используем `PreviewTextInput`.

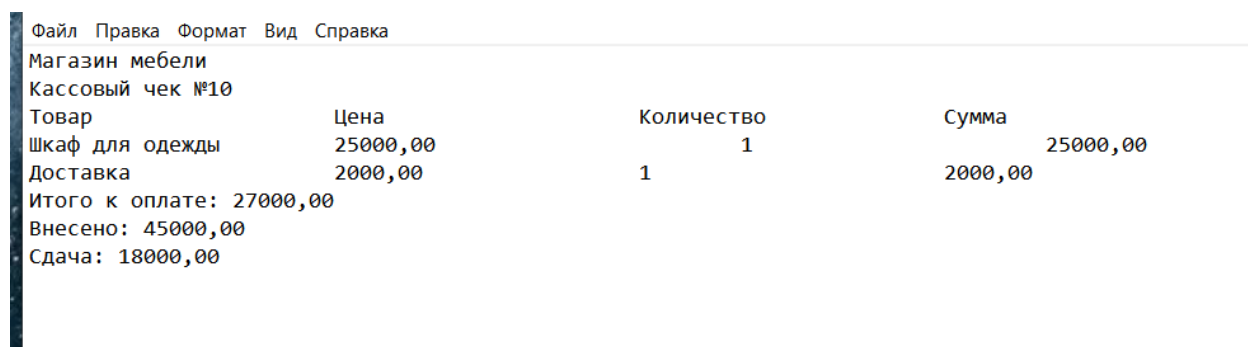
Используем `PreviewTextInput`.

Рисунок 108 – PreviewTextInput

2.19. Должна быть возможность просмотреть сохраненные в БД чеки и выгрузить их в файл.

Рисунок 109 - Выгрузка чеков в файл

При выборе строки в DataGrid и нажатии на кнопку «CheckToFile» создается чек на рабочем столе.



Магазин мебели			
Кассовый чек №10			
Товар	Цена	Количество	Сумма
Шкаф для одежды	25000,00	1	25000,00
Доставка	2000,00	1	2000,00
Итого к оплате: 27000,00			
Внесено: 45000,00			
Сдача: 18000,00			

Рисунок 110 – Чек

Вывод: создали базу данных по предметной области «Магазин мебели», содержащую в себе 12 таблиц и нормализованную согласно 3 НФ. Также создали WPF-проект согласно техническому заданию и присоединили к нему базу данных для их взаимодействия.

Список литературы

1. Бессмертный, И. А. Системы искусственного интеллекта : учеб. пособие для СПО / И. А. Бессмертный. — 2-е изд., испр. и доп. — М. : Издательство Юрайт, 2018. — 130 с.
2. Гниденко, И. Г. Технология разработки программного обеспечения : учеб. пособие для СПО / И. Г. Гниденко, Ф. Ф. Павлов, Д. Ю. Федоров. — М. : Издательство Юрайт, 2017. — 235 с.
3. Гордеев, С. И. Организация баз данных в 2 ч. Часть 2 : учебник для вузов / С. И. Гордеев, В. Н. Волошина. — 2-е изд., испр. и доп. — М. : Издательство Юрайт, 2019. — 501 с.
4. Жмудь, В. А. Моделирование замкнутых систем автоматического управления : учеб. пособие для академического бакалавриата / В. А. Жмудь. — 2-е изд., испр. и доп. — М. : Издательство Юрайт, 2019. — 128 с.
5. Зыков, С. В. Программирование. Объектно-ориентированный подход : учебник и практикум для академического бакалавриата / С. В. Зыков. — М. : Издательство Юрайт, 2019. — 155 с.
6. Иванов, В. М. Интеллектуальные системы : учеб. пособие для СПО / В. М. Иванов ; под науч. ред. А. Н. Сесекина. — М. : Издательство Юрайт, 2019. — 93 с.
7. Иванов, В. М. Интеллектуальные системы : учеб. пособие для вузов / В. М. Иванов ; под науч. ред. А. Н. Сесекина. — М. : Издательство Юрайт, 2017. — 91 с.
8. Кубенский, А. А. Функциональное программирование : учебник и практикум для академического бакалавриата / А. А. Кубенский. — М. : Издательство Юрайт, 2019. — 348 с.
9. Кудрина, Е. В. Основы алгоритмизации и программирования на языке `c#` : учеб. пособие для СПО / Е. В. Кудрина, М. В. Огнева. — М. : Издательство Юрайт, 2019. — 322 с.

10. Кудрина, Е. В. Основы алгоритмизации и программирования на языке `c#` : учеб. пособие для бакалавриата и специалитета / Е. В. Кудрина, М. В. Огнева. — М. : Издательство Юрайт, 2019. — 322 с.
11. Кудрявцев, К. Я. Методы оптимизации : учеб. пособие для вузов / К. Я. Кудрявцев, А. М. Прудников. — 2-е изд. — М. : Издательство Юрайт, 2019. — 140 с.
12. Лаврищева, Е. М. Программная инженерия и технологии программирования сложных систем : учебник для вузов / Е. М. Лаврищева. — 2-е изд., испр. и доп. — М. : Издательство Юрайт, 2019. — 432 с.

