

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КОМИ
Государственное профессиональное образовательное учреждение
«Воркутинский политехнический техникум»

КУРСОВОЙ ПРОЕКТ

По дисциплине МДК.05.02 Разработка кода
информационных систем

Разработка информационной системы
«Поликлиника»

Выполнил студент гр. ИСП-20 / _____ / Райков С.В. /
(подпись) (Ф.И.О.)

ОЦЕНКА: _____

Дата: _____

ПРОВЕРИЛ

Научный руководитель _____ / Егоров Данил Павлович /
(подпись) (Ф.И.О.)

Воркута

2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
ГЛАВА 1. ВЫБОР ИНСТРУМЕНТАРИЯ	4
1.1 Платформа .NET	4
1.2 Язык программирования C#	6
1.3 Windows Presentation Foundation (WPF)	9
1.4 СУБД SQL server	13
1.5 Microsoft SQL Server Management Studio	15
1.6 Entity Framework	16
ГЛАВА 2. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ.....	19
2.1 Разработка диаграммы ERD	19
2.2 Разработка базы данных	20
ГЛАВА 3. РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ	26
3.1 Разработка интерфейса информационной системы	26
3.2 Программирование информационной системы.....	32
ЗАКЛЮЧЕНИЕ.....	39
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ	40

ВВЕДЕНИЕ

Сегодня управление предприятием без компьютера просто немыслимо. Компьютеры давно и прочно вошли в такие области управления, как бухгалтерский учет, управление складом, ассортиментом, а также и медициной. Однако современный бизнес, как и многие другие направления, требует гораздо более широкого применения информационных технологий в управлении предприятием. Жизнеспособность и развитие информационных технологий объясняется тем, что современный бизнес крайне чувствителен к ошибкам в управлении. Интуиции, личного опыта руководителя и размеров капитала уже мало для того, чтобы быть первым. Для принятия любого грамотного управленческого решения в условиях неопределенности и риска необходимо постоянно держать под контролем различные аспекты финансово-хозяйственной деятельности, будь то: торговля, медицина или предоставление каких-либо услуг. И чем крупнее предприятие, тем серьезнее должны быть подобные вложения сил и средств. Они являются жизненной необходимостью — в жесткой конкурентной борьбе одержать победу сможет лишь тот, кто лучше оснащен и наиболее эффективно организован.

Автоматизированная информационная система «Поликлиника» включает в себя данные о врачах, пациентах, кабинетах и вызовах, которые необходимы для работы поликлиники. База данных позволяет осуществлять добавление, изменение, поиск и удаление данных, а также просматривать эти данные.

Актуальность данной темы в том, что в наш век информационных технологий, стало реально все документы преобразовывать в электронный вид и регистратура в считанные минуты может найти сведения о принятых пациентах, вызовах, кабинетах.

Объект: информационная система “Поликлиника”.

Предмет: автоматизация бизнес процессов работы поликлиники.

Цель работы: разработать автоматизированную информационную систему для работы регистратуры поликлиники.

Задачи:

- создать систему регистрации клиентов/больных в очереди или записи к специалистам;
- создать в системе учёт медицинских карт;
- построить концептуальную информационную модель;
- сформировать физическую структуру базы данных;
- реализовать простое пользовательское приложение.

ГЛАВА 1. ВЫБОР ИНСТРУМЕНТАРИЯ

1.1 Платформа .NET

.NET Framework — программная платформа, выпущенная компанией Microsoft в 2002 году. Основой платформы является общезыковая среда исполнения Common Language Runtime (CLR), которая подходит для различных языков программирования. Функциональные возможности CLR доступны в любых языках программирования, использующих эту среду. В настоящее время .NET Framework развивается в виде .NET.

Считается, что платформа .NET Framework является ответом компании Microsoft на набравшую к тому времени большую популярность платформу Java компании Sun Microsystems (ныне принадлежит Oracle).

Хотя .NET Framework является патентованной технологией корпорации Microsoft и официально рассчитана на работу под операционными системами семейства Windows, существуют независимые проекты (прежде всего это Mono и Portable.NET), позволяющие запускать программы .NET Framework на некоторых других операционных системах.

Основной идеей при разработке .NET Framework являлось обеспечение свободы разработчика за счёт предоставления ему возможности создавать приложения различных типов, способные выполняться на различных типах устройств и в различных средах.

Вторым принципом стала ориентация на системы, работающие под управлением семейства операционных систем Microsoft Windows.

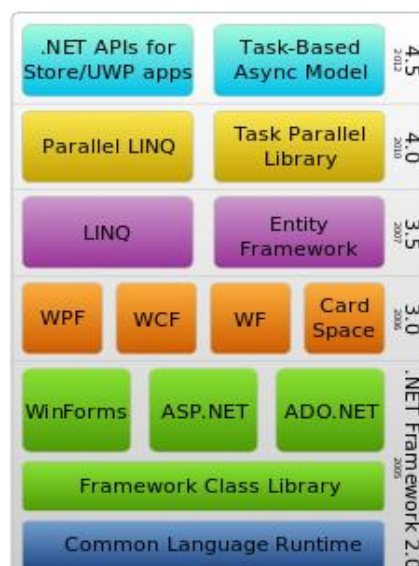


Рис. 1.1 Стек технологий .NET Framework

Программа для .NET Framework, написанная на любом поддерживаемом языке программирования, сначала переводится компилятором в единый для .NET

промежуточный байт-код Common Intermediate Language (CIL) (ранее назывался Microsoft Intermediate Language, MSIL). В терминах .NET получается сборка, англ. assembly. Затем код либо исполняется виртуальной машиной Common Language Runtime (CLR), либо транслируется утилитой NGen.exe в исполняемый код для конкретного целевого процессора. Использование виртуальной машины предпочтительно, так как избавляет разработчиков от необходимости заботиться об особенностях аппаратной части. В случае использования виртуальной машины CLR встроенный в неё JIT-компилятор «на лету» (just in time) преобразует промежуточный байт-код в машинные коды нужного процессора. Современная технология динамической компиляции позволяет достигнуть высокого уровня быстродействия. Виртуальная машина CLR также сама заботится о базовой безопасности, управлении памятью и системе исключений, избавляя разработчика от части работы.

Архитектура .NET Framework описана и опубликована в спецификации Common Language Infrastructure (CLI), разработанной Microsoft и утверждённой ISO и ECMA. В CLI описаны типы данных .NET, формат метаданных о структуре программы, система исполнения байт-кода и многое другое.

Объектные классы .NET, доступные для всех поддерживаемых языков программирования, содержатся в библиотеке Framework Class Library (FCL). В FCL входят классы Windows Forms, ADO.NET, ASP.NET, Language Integrated Query, Windows Presentation Foundation, Windows Communication Foundation и другие. Ядро FCL называется Base Class Library (BCL).

Среды разработки, поддерживающие .NET:

- Microsoft Visual Studio (C#, Visual Basic .NET, Managed C++, F#)
- SharpDevelop
- MonoDevelop
- Embarcadero RAD Studio (Delphi for .NET); ранее Borland Developer Studio (Delphi for .NET, C#)
- A#
- Zonnon
- PascalABC.NET
- JetBrains Rider

Приложения .NET также можно разрабатывать в текстовом редакторе, просто вызывая компилятор из командной строки.

Одной из основных идей Microsoft .NET является совместимость программных частей, написанных на разных языках. Например, служба, написанная на C++ для Microsoft .NET, может обратиться к методу класса из библиотеки, написанной на Delphi; на C# можно написать

класс, наследованный от класса, написанного на Visual Basic .NET, а исключение, созданное методом, написанным на C#, может быть перехвачено и обработано в Delphi. Каждая библиотека (сборка) в .NET имеет сведения о своей версии, что позволяет устранить возможные конфликты между разными версиями сборок.

Языки, поставляемые вместе с Microsoft Visual Studio:

- C#
- Visual Basic .NET
- JScript .NET
- C++/CLI — новая версия Managed C++
- F# — член семейства языков программирования ML, включён в VS2010/VS2012/ VS2015/VS2017/VS2019/VS2022
- J# — последний раз был включён в VS2005

1.2 Язык программирования C#

C# — объектно-ориентированный язык программирования общего назначения. Разработан в 1998—2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота как язык разработки приложений для платформы Microsoft .NET Framework и .NET Core. Впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, переменные, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Delphi, Модула, Smalltalk и, в особенности, Java — C#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественная реализация интерфейсов).

C# разрабатывался как язык программирования прикладного уровня для CLR и, как таковой, зависит, прежде всего, от возможностей самой CLR. Это касается, прежде всего, системы типов C#, которая отражает BCL. Присутствие или отсутствие тех или иных выразительных особенностей языка диктуется тем, может ли конкретная языковая

особенность быть транслирована в соответствующие конструкции CLR. Так, с развитием CLR от версии 1.1 к 2.0 значительно обогатился и сам C#; подобного взаимодействия следует ожидать и в дальнейшем (однако, эта закономерность была нарушена с выходом C# 3.0, представляющего собой расширения языка, не опирающиеся на расширения платформы .NET). CLR предоставляет C#, как и всем другим .NET-ориентированным языкам, многие возможности, которых лишены «классические» языки программирования. Например, сборка мусора не реализована в самом C#, а производится CLR для программ, написанных на C#, точно так же, как это делается для программ на VB.NET, J# и др.

Версия 8.0

- Модификатор `readonly`. Был создан для обозначения члена, который не изменит состояние.
- Методы интерфейсов по умолчанию. Теперь при создании метода интерфейса можно объявить его реализацию по умолчанию, которую можно переопределить в классе, который реализует этот интерфейс.
- Сопоставление шаблонов. Возможность позволяет работать с шаблонами в зависимости от формата в связанных, но различных типах данных.
- Рекурсивные шаблоны. Является выражением шаблона, которое применяется к результатам другого выражения шаблона.
- Выражения `switch` позволяют сократить количество `case` и `break`, а также фигурных скобок.
- Шаблоны свойств. Позволяет сопоставлять свойства исследуемого объекта с помощью `{ variable : value } => ...`.
- Шаблоны кортежей. Используется, если нужно работать с несколькими наборами входных данных. `(value1, value2,...) => ...`
- Объявление `using`. Это объявление переменной, которому предшествует ключевое слово `using`. Оно сообщает компилятору, что объявляемая переменная должна быть удалена в конце области видимости.
- Статический локальный метод. Теперь можно убедиться в том, что метод не охватывает какие-либо переменные из области видимости с помощью добавления к нему модификатора `static`.
- Удаляемые ссылочные структуры. Ссылочные структуры не могут реализовать `IDisposable` (как и любые другие интерфейсы). Поэтому чтобы удалить `ref struct`, необходим доступный `void Dispose()`.
- Типы значений, допускающие значение `null`. Теперь, чтобы указать, что переменная типа значений допускает значение `null`, необходимо поставить к имени типа ?

- Асинхронные потоки. Это во-первых интерфейс `IAsyncEnumerable<T>`. А во-вторых конструкция `foreach` с `await`.
- Асинхронные высвобождаемые типы. Начиная с C# 8.0 язык поддерживает асинхронные освобождаемые типы, реализующие интерфейс `System.IAsyncDisposable`. Операнд выражения `using` может реализовывать `IDisposable` или `IAsyncDisposable`. В случае `IAsyncDisposable` компилятор создает код для `await`, возвращенного `Task` из `IAsyncDisposable.DisposeAsync`.
- Индексы и диапазоны. Диапазоны и индексы обеспечивают лаконичный синтаксис для доступа к отдельным элементам или диапазонам в последовательности. Нововведение включает в себя операторы `^` и `..`, а также `System.Index` и `System.Range`
- Оператор присваивания объединения с `null`. Оператор `??=` можно использовать для присваивания значения правого операнда левому операнду только в том случае, если левый операнд принимает значение `null`.

```

• List<int> numbers = null;
• int? i = null;
•
• numbers ??= new List<int>();
• numbers.Add(i ??= 17);
• numbers.Add(i ??= 20);
•
• Console.WriteLine(string.Join(" ", numbers)); // output: 17 17
• Console.WriteLine(i); // output: 17

```

Рис. 1.2 Фрагмент кода языка программирования C#

- Неуправляемые сконструированные типы. Начиная с C# 8.0, сконструированный тип значения является неуправляемым, если он содержит поля исключительно неуправляемых типов (например универсальный тип `<T>`).
- Выражение `stackalloc` во вложенных выражениях. Теперь если результат выражения `stackalloc` имеет тип `System.Span<T>` или `System.ReadOnlySpan<T>`, то его можно использовать в других выражениях.

```

• Span<int> numbers = stackalloc[] { 1, 2, 3, 4, 5, 6 };
• var ind = numbers.IndexOfAny(stackalloc[] { 2, 4, 6, 8 });
• Console.WriteLine(ind); // output: 1

```

Рис. 1.3 Фрагмент кода для создания массива

- Порядок маркеров `$` и `@` в интерполированных строках `verbatim` теперь может быть любым.

1.3 Windows Presentation Foundation (WPF)

Windows Presentation Foundation (WPF) — аналог WinForms, система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая (презентационная) подсистема в составе .NET Framework (начиная с версии 3.0), использующая язык XAML.

WPF предустановлена в Windows Vista (.NET Framework 3.0), Windows 7 (.NET Framework 3.5 SP1), Windows 8 (.NET Framework 4.0 и 4.5), Windows 8.1 (.NET Framework 4.5.1) и Windows 10 (.NET Framework 4.7). С помощью WPF можно создавать широкий спектр как автономных, так и запускаемых в браузере приложений.

В основе WPF лежит векторная система визуализации, не зависящая от разрешения устройства вывода и созданная с учётом возможностей современного графического оборудования. WPF предоставляет средства для создания визуального интерфейса, включая язык XAML (eXtensible Application Markup Language), элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление.

Графической технологией, лежащей в основе WPF, является DirectX, в отличие от Windows Forms, где используется GDI/GDI+. Производительность WPF выше, чем у GDI+ за счёт использования аппаратного ускорения графики через DirectX.

Также существует урезанная версия CLR, называемая WPF/E, она же известна как Silverlight.

XAML представляет собой язык декларативного описания интерфейса, основанный на XML. Также реализована модель разделения кода и дизайна, позволяющая кооперироваться программисту и дизайнеру. Кроме того, есть встроенная поддержка стилей элементов, а сами элементы легко разделить на элементы управления второго уровня, которые, в свою очередь, разделяются до уровня векторных фигур и свойств/действий. Это позволяет легко задать стиль для любого элемента, например, Button (кнопка).

Для работы с WPF требуется любой .NET-совместимый язык. В этот список входит множество языков: C#, F#, VB.NET, C++, Ruby, Python, Delphi (Prism), Lua и многие другие. Для полноценной работы может быть использована как Visual Studio, так и Expression Blend. Первая ориентирована на программирование, а вторая — на дизайн и позволяет делать многие вещи, не прибегая к ручному редактированию XAML. Примеры этому — анимация, стилизация, состояния, создание элементов управления и так далее.

WPF предоставляет широкий спектр возможностей по созданию интерактивных настольных приложений:

Привязка данных

Это гибкий механизм, который позволяет через расширения разметки XAML связывать различные данные (от значений свойств элементов управления до общедоступных свойств, реализующих поля базы данных через Entity Framework). Привязка данных представлена классом Binding, который в свою очередь унаследован от MarkupExtension, что позволяет использовать привязки не только в коде, но и в разметке (рис. 1.4).

```
<StackPanel Orientation="Horizontal">
    <Slider x:Name="slider" Width="200" Minimum="1" Maximum="100"
    Value="60"/>
    <TextBox Text="{Binding ElementName=slider, Path=Value, Mode=TwoWay,
    UpdateSourceTrigger=PropertyChanged}"/>
</StackPanel>
```

Рис. 1.4 Фрагмент .xaml кода с программированием значений

Помимо основного класса Binding в WPF реализовано еще несколько механизмов привязок:

- MultiBinding — позволяет создавать множественные привязки, указывая несколько элементов
- TemplateBinding — используется в шаблонах для связывания свойства элемента внутри шаблона со свойством элемента, к которому применен шаблон
- PriorityBinding — ранжирует список привязок и выбирает из них свойство (согласно приоритету) к которому будет применена привязка. Если привязка, имеющая наивысший приоритет успешно возвращает значение, то нет необходимости обрабатывать другие привязки в списке.

Стили

Позволяют создавать стилевое оформление элементов и, как правило, используются только в разметке:

```
<Button>
    <Button.Style>
        <Style TargetType="Button">
            <Setter Property="FontSize" Value="20"/>
            <Setter Property="Foreground" Value="LimeGreen"/>
        </Style>
    </Button.Style>
</Button>
```

Рис. 1.5 Фрагмент .xaml кода со стилями

Если стиль задается в ресурсах (например в словаре ресурсов), то можно использовать атрибут x:Key для указания уникального ключа. Затем в элементе управления, к которому необходимо применить стиль, нужно использовать расширение разметки StaticResource для связывания с этим стилем. Если использовать этот прием, то стили не будут нагромождать

разметку.

Шаблоны элементов управления

Позволяют менять графическое оформление элементов и представлены классом `ControlTemplate`. В отличие от стилей, можно менять не только графическое представление элемента, но и его структуру. При этом шаблон элемента управления задается через свойство `Template`.

Простой пример круглой кнопки (рис. 1.6).

```
<Button Content="Hey!" Background="LimeGreen" Foreground="White">
  <Button.Template>
    <ControlTemplate TargetType="Button">
      <Grid>
        <Ellipse Fill="{TemplateBinding Background}"
Stroke="{TemplateBinding BorderBrush}" Stretch="Fill"/>
        <ContentPresenter VerticalAlignment="Center"
HorizontalAlignment="Center"/>
      </Grid>
    </ControlTemplate>
  </Button.Template>
</Button>
```

Рис. 1.6 Фрагмент кода .xaml для создания круглой кнопки

Шаблоны данных

В отличие от шаблонов элементов управления, задаются для определенного контекста данных (который в блочных элементах управления задается через свойство `DataContext`, а в списковых через `ItemsSource`). Сам шаблон данных представлен классом `DataTemplate`. Для обозначения типа данных, к которому необходимо применить шаблон, используется свойство `DataType`.

Ресурсы

Система ресурсов позволяет объединять шаблоны, стили, кисти, анимацию и многие другие интерактивные элементы, что существенно упрощает работу с ними. Ресурсы задаются в свойстве `Resources` класса `FrameworkElement`, от которого унаследованы все элементы управления, панели компоновки и даже класс `Application`. Это позволяет создавать многоуровневую систему ресурсов:

- ресурсы внутри объекта — действительны только для этого объекта
- ресурсы внутри панели компоновки (например `Grid`) — позволяет задать границу контекста ресурсов на уровне этой панели

- ресурсы внутри окна Window — если в приложении используется несколько окон, то ресурсы одного окна не будут доступны ресурсам другого окна

```
<Window.Resources>
    <SolidColorBrush x:Key="SampleBrush" Color="LimeGreen"/>
</Window.Resources>

...

<Button Content="Hey!" Background="{StaticResource SampleBrush}" />
```

Рис. 1.7 Фрагмент кода .xaml для создания заднего фона

- ресурсы приложения — доступны повсеместно (как правило задаются в отдельном словаре ресурсов)

Графика

WPF представляет обширный, масштабируемый и гибкий набор графических возможностей:

- Графика, не зависящая от разрешения и устройства. Основной единицей измерения в графической системе WPF является аппаратно-независимый пиксель, который составляет 1/96 часть дюйма независимо от фактического разрешения экрана.
- Дополнительная поддержка графики и анимации. WPF упрощает программирование графики за счет автоматического управления анимацией. Разработчик не должен заниматься обработкой сцен анимации, циклами отрисовки и билинейной интерполяцией
- Аппаратное ускорение. Графическая система WPF использует преимущества графического оборудования, чтобы уменьшить использование ЦП.

Двухмерная графика

WPF предоставляет библиотеку общих двухмерных фигур, нарисованных с помощью векторов, таких, как прямоугольники и эллипсы, а также графические пути. И в своей функциональности фигуры реализуют многие возможности, которые доступны обычным элементам управления.

Двухмерная графика в WPF включает визуальные эффекты, такие как градиенты, точечные рисунки, чертежи, рисунки с видео, поворот, масштабирование и наклон.

Трёхмерная графика

WPF также включает возможности трёхмерной отрисовки, интегрированные с двухмерной графикой, что позволяет создавать более яркий и интересный пользовательский интерфейс.

1.4 СУБД SQL server

Microsoft SQL Server является одной из наиболее популярных систем управления базами данных (СУБД) в мире. Данная СУБД подходит для самых различных проектов: от небольших приложений до больших высоконагруженных проектов.

SQL Server был создан компанией Microsoft. Первая версия вышла в 1987 году. А текущей версией является версия 2022, которая вышла в ноябре 2022 году и которая будет использоваться в текущем руководстве.

SQL Server долгое время был исключительно системой управления базами данных для Windows, однако начиная с версии 16 эта система доступна и на Linux.

SQL Server характеризуется такими особенностями как:

Производительность. SQL Server работает очень быстро.

Надежность и безопасность. SQL Server предоставляет шифрование данных.

Простота. С данной СУБД относительно легко работать и вести администрирование.

Центральным аспектом в MS SQL Server, как и в любой СУБД, является база данных. База данных представляет хранилище данных, организованных определенным способом. Нередко физически база данных представляет файл на жестком диске, хотя такое соответствие необязательно. Для хранения и администрирования баз данных применяются системы управления базами данных (database management system) или СУБД (DBMS). И как раз MS SQL Server является одной из такой СУБД.

Для организации баз данных MS SQL Server использует реляционную модель. Эта модель баз данных была разработана еще в 1970 году Эдгаром Коддом. А на сегодняшний день она фактически является стандартом для организации баз данных.

Реляционная модель предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта.

Для идентификации каждой строки в рамках таблицы применяется первичный ключ (primary key). В качестве первичного ключа может выступать один или несколько столбцов. Используя первичный ключ, мы можем сослаться на определенную строку в таблице. Соответственно две строки не могут иметь один и тот же первичный ключ.

Через ключи одна таблица может быть связана с другой, то есть между двумя таблицами могут быть организованы связи. А сама таблица может быть представлена в виде отношения ("relation").

Для взаимодействия с базой данных применяется язык SQL (Structured Query Language). Клиент (например, внешняя программа) отправляет запрос на языке SQL посредством

специального API. СУБД должным образом интерпретирует и выполняет запрос, а затем посылает клиенту результат выполнения.

Изначально язык SQL был разработан в компании IBM для системы баз данных, которая называлась System/R. При этом сам язык назывался SEQUEL (Structured English Query Language). Хотя в итоге ни база данных, ни сам язык не были впоследствии официально опубликованы, по традиции сам термин SQL нередко произносят как "сиквел".

В 1979 году компания Relational Software Inc. разработала первую систему управления баз данных, которая называлась Oracle и которая использовала язык SQL. В связи с успехом данного продукта компания была переименована в Oracle.

Впоследствии стали появляться другие системы баз данных, которые использовали SQL. В итоге в 1989 году Американский Национальный Институт Стандартов (ANSI) кодифицировал язык и опубликовал его первый стандарт. После этого стандарт периодически обновлялся и дополнялся. Последнее его обновление состоялось в 2011 году. Но несмотря на наличие стандарта нередко производители СУБД используют свои собственные реализации языка SQL, которые немного отличаются друг от друга.

Выделяются две разновидности языка SQL: PL-SQL и T-SQL. PL-SQL используется в таких СУБД как Oracle и MySQL. T-SQL (Transact-SQL) применяется в SQL Server. Собственно поэтому, в рамках текущего руководства будет рассматриваться именно T-SQL.

В зависимости от задачи, которую выполняет команда T-SQL, он может принадлежать к одному из следующих типов:

DDL (Data Definition Language / Язык определения данных). К этому типу относятся различные команды, которые создают базу данных, таблицы, индексы, хранимые процедуры и т.д. В общем определяют данные.

В частности, к этому типу мы можем отнести следующие команды:

CREATE: создает объекты базы данных (саму базу данных, таблицы, индексы и т.д.)

ALTER: изменяет объекты базы данных

DROP: удаляет объекты базы данных

TRUNCATE: удаляет все данные из таблиц

DML (Data Manipulation Language / Язык манипуляции данными). К этому типу относят команды на выбор данных, их обновление, добавление, удаление - в общем все те команды, с помощью которыми мы можем управлять данными.

К этому типу относятся следующие команды:

SELECT: извлекает данные из БД

UPDATE: обновляет данные

INSERT: добавляет новые данные

DELETE: удаляет данные

DCL (Data Control Language / Язык управления доступа к данным). К этому типу относят команды, которые управляют правами по доступу к данным. В частности, это следующие команды:

GRANT: предоставляет права для доступа к данным

REVOKE: отзывает права на доступ к данным

1.5 Microsoft SQL Server Management Studio

SQL Server Management Studio (SSMS) — утилита из Microsoft SQL Server 2005 и более поздних версий для конфигурирования, управления и администрирования всех компонентов Microsoft SQL Server. Утилита включает скриптовый редактор и графическую программу, которая работает с объектами и настройками сервера.

Главным инструментом SQL Server Management Studio является Object Explorer, который позволяет пользователю просматривать, извлекать объекты сервера, а также полностью ими управлять.

Также есть SQL Server Management Studio Express для Express версии сервера, которая является бесплатной. Однако в ней нет поддержки ряда компонентов (Analysis Services, Integration Services, Notification Services, Reporting Services) и SQL Server 2005 Mobile Edition.

Начиная с версии 16.5.3 пакет SSMS выделен в отдельный обновляемый продукт, доступный для скачивания на сайте Microsoft. Текущая доступная версия SSMS 18.4 (15.0.18206.0) (general availability) поддерживает MS SQL server начиная с версии 2008 по 2019.

Среда SQL Server Management Studio – это основной, стандартный и полнофункциональный инструмент для работы с Microsoft SQL Server, разработанный компанией Microsoft, который предназначен как для разработчиков, так и для администраторов SQL Server.

С помощью SSMS Вы можете разрабатывать базы данных, выполнять инструкции T-SQL, а также администрировать Microsoft SQL Server.

Если в Ваши задачи входит полное сопровождение Microsoft SQL Server, начиная от создания баз данных, написания SQL запросов, создания хранимых процедур и функций, и заканчивая администрированием SQL Server, включая управление безопасностью, то основным Вашим инструментом будет как раз среда SQL Server Management Studio.

Среда SQL Server Management Studio реализована только под Windows, поэтому если Вам нужен инструмент для работы с Microsoft SQL Server, который будет работать на других

платформах, например, на Linux или macOS, то Вам следует использовать инструмент Azure Data Studio, который также является официальным инструментом, разработанным компанией Microsoft.

Основной функционал SQL Server Management Studio

Теперь давайте рассмотрим функционал и возможности среды SQL Server Management Studio, иными словами, какие именно действия и операции мы можем выполнять, используя данный инструмент.

Сначала давайте посмотрим на общий перечень возможностей, которые нам предоставляет среда SQL Server Management Studio, а затем более подробно рассмотрим каждый пункт из этого перечня.

Основной функционал SSMS

- Подключение к службам SQL Server
- Обзорщик объектов
- Обзорщик шаблонов
- Редактор SQL кода
- Просмотр плана выполнения запроса
- Обзорщик решений
- Конструктор таблиц
- Конструктор баз данных (Диаграммы баз данных)
- Конструктор запросов и представлений
- Просмотр свойств объектов
- Мастер создания скриптов
- Управление безопасностью SQL Server
- Присоединение и отсоединение баз данных
- Создание резервных копий баз данных и восстановление баз данных из архива
- Создание связанных серверов (Linked Server)
- Монитор активности SQL Server
- Настройка репликации баз данных
- Профилировщик XEvent

1.6 Entity Framework

ADO.NET Entity Framework (EF) — объектно-ориентированная технология доступа к данным, является object-relational mapping (ORM) решением для .NET Framework от Microsoft.

Предоставляет возможность взаимодействия с объектами как посредством LINQ в виде LINQ to Entities, так и с использованием Entity SQL. Для облегчения построения web-решений используется как ADO.NET Data Services (Astoria), так и связка из Windows Communication Foundation и Windows Presentation Foundation, позволяющая строить многоуровневые приложения, реализуя один из шаблонов проектирования MVC, MVP или MVVM.

Entity SQL представляет собой язык, подобный языку SQL, который позволяет выполнять запросы к концептуальным моделям в Entity Framework.

LINQ to Entities – это альтернативный интерфейс LINQ API, используемый для обращения к базе данных. Он отделяет сущностную объектную модель данных от физической базы данных, вводя логическое отображение между ними. Так, например, схемы реляционных баз данных не всегда подходят для построения объектно-ориентированных приложений и в результате мы имеем объектную модель приложения, существенно отличающуюся от логической модели данных, в этом случае используется LINQ to Entities, который использует модель EDM (Entity Data Model). То есть, если вам нужно ослабить связь между вашей сущностной объектной моделью данных и физической моделью данных, например, если ваши сущностные объекты конструируются из нескольких таблиц или вам нужна большая гибкость в моделировании ваших сущностных объектов используйте LINQ to Entities

Релиз ADO.NET Entity Framework состоялся 11 августа 2008 года в составе .NET Framework 3.5 Service Pack 1 и Visual Studio 2008 Service Pack 1. В VS 2008 вошёл EDM Wizard для реверс-инжиниринга существующих баз данных и EDM Designer для редактирования сгенерированных моделей или создания их с нуля.

23 июня 2008 года, ещё до релиза первой версии, на стадии финальной доводки Entity Framework V1, начался процесс разработки Entity Framework V2.0. По словам англ. Tim Mallalieu, менеджера программы LINQ to SQL и EF, в .NET Framework 4.0 именно Entity Framework станет рекомендуемой технологией доступа к реляционным СУБД посредством LINQ.

12 апреля 2010 года в составе релиза Visual Studio 2010 и .NET Framework 4.0 был представлена Entity Framework 4.0. Позже уже отдельно от фреймворка были представлены версии: 4.1 (апрель 2011), 4.2 (октябрь 2011), 4.3 (февраль 2012).

11 августа 2012 года была представлена версия 5.0.0, которая была предназначена для .NET Framework 4.5. А 17 октября 2013 года была представлена версия 6.0, которая вышла под лицензией Apache License v2, тем самым став open-source проектом.

Версия 6.0 была выпущена 17 октября 2013 года и сейчас это проект с открытым исходным кодом под лицензией Apache License v2. В версии 6.0 был сделан ряд улучшений в поддержке метода работы Code First.

Изначально с самой первой версии Entity Framework поддерживал подход Database First, который позволял по готовой базе данных сгенерировать модель edmx. Затем эта модель использовалась для подключения к базе данных. Позже был добавлен подход Model First. Он позволял создать вручную с помощью визуального редактора модель edmx, и по ней создать базу данных. Начиная с 5.0 предпочтительным подходом становится Code First. Его суть - сначала пишется код модели на C#, а затем по нему генерируется база данных. При этом модель edmx уже не используется.

ГЛАВА 2. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

2.1 Разработка диаграммы ERD

Диаграмма — (ER-модель данных) обеспечивает стандартный способ определения данных и отношений между ними. Она включает сущности и взаимосвязи, отражающие основные бизнес-правила предметной области. Диаграммы «сущность— связь» в отличие от функциональных диаграмм определяют спецификации структур данных программного обеспечения.

Базовыми понятиями ER-модели данных (ER — Entity— Relationship) являются сущность, атрибут и связь.

Сущность — это класс однотипных реальных или абстрактных объектов (людей, событий, состояний, предметов и т.п.), информация о которых имеет существенное значение для рассматриваемой предметной области.

Атрибут — любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности. Атрибут, таким образом, представляет собой некоторый тип характеристик или свойств, ассоциированных с множеством реальных или абстрактных объектов.

Атрибуты делятся на ключевые, т. е. входящие в состав уникального идентификатора ключа, и описательные — прочие.

Первичный ключ — это атрибут или совокупность атрибутов и связей, предназначенная для уникальной идентификации каждого экземпляра сущности.

Описательные атрибуты могут быть обязательными или необязательными. Обязательные атрибуты для каждой сущности всегда имеют конкретное значение, необязательные могут быть не определены.

Связь — это отношение одной сущности к другой или к самой себе. Если любой экземпляр одной сущности связан хотя бы с одним экземпляром другой сущности, то связь является обязательной. Необязательная связь представляет собой условное отношение между сущностями. Каждая сущность может обладать любым количеством связей с другими сущностями модели. Различают три типа отношений «один-к-одному»; «один-ко-многим»; «многие-ко-многим».

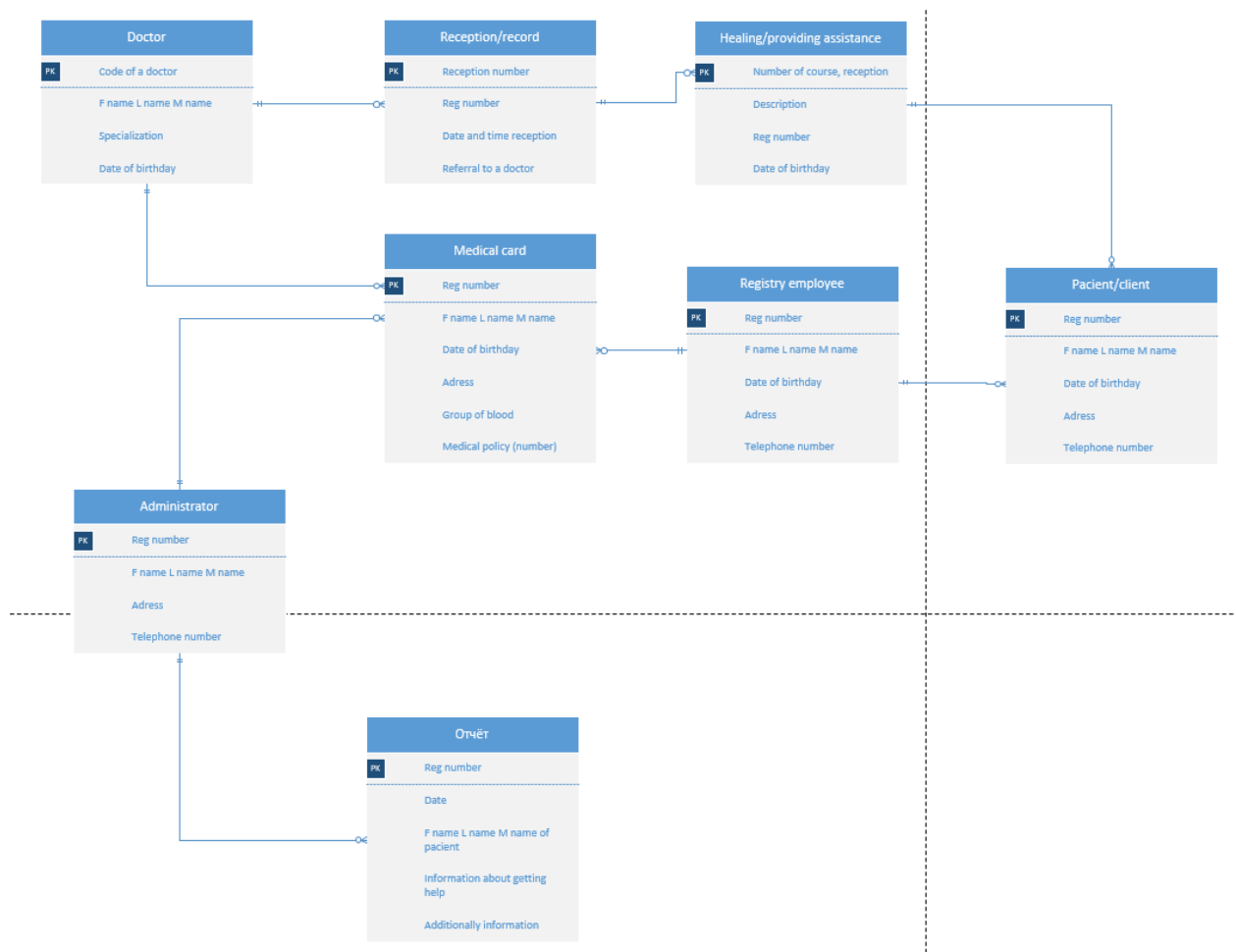


Рис 2.1 Диаграмма ER

Основные сущности для данной диаграммы это: клиент/пациент, врач, администратор и относящиеся сущности к системе. Второстепенными сущностями являются: работник регистратуры, медицинская карта, рецепт/лечение и лечение. Возникают связи между сущностями — клиент, который может подать много заявок, а работник регистратуры один. Такое отношение “многие к одному”. Также у сущностей есть атрибуты, например у клиента есть ФИО, дата рождения и т.д. Каждый запрос содержит уникальный идентификатор (личный номер)

2.2 Разработка базы данных

Первое, что нужно сделать, это запустить среду SQL Server Management Studio и подключиться к SQL серверу.

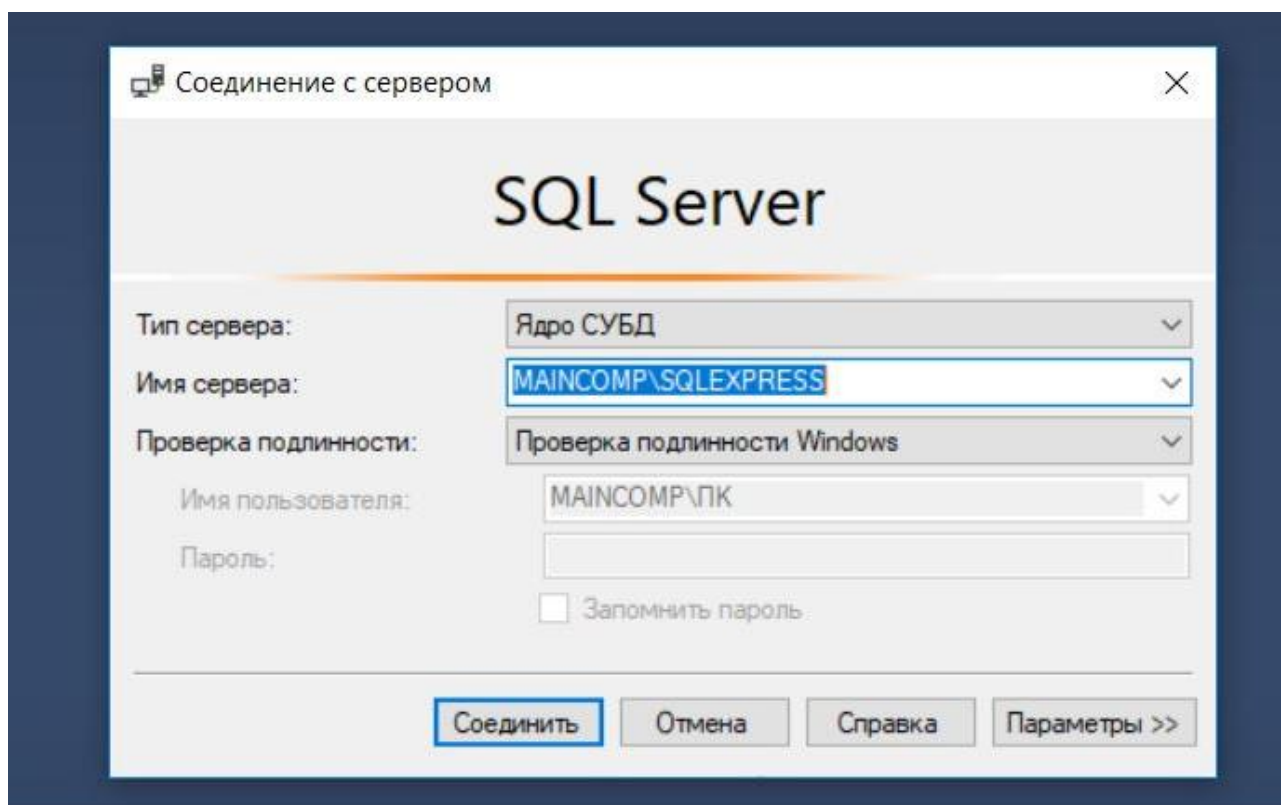


Рис. 2.1 Подключение к серверу базы данных

Затем в обозревателе объектов щелкнуть по контейнеру «Базы данных» правой кнопкой мыши и выбрать пункт «Создать базу данных».

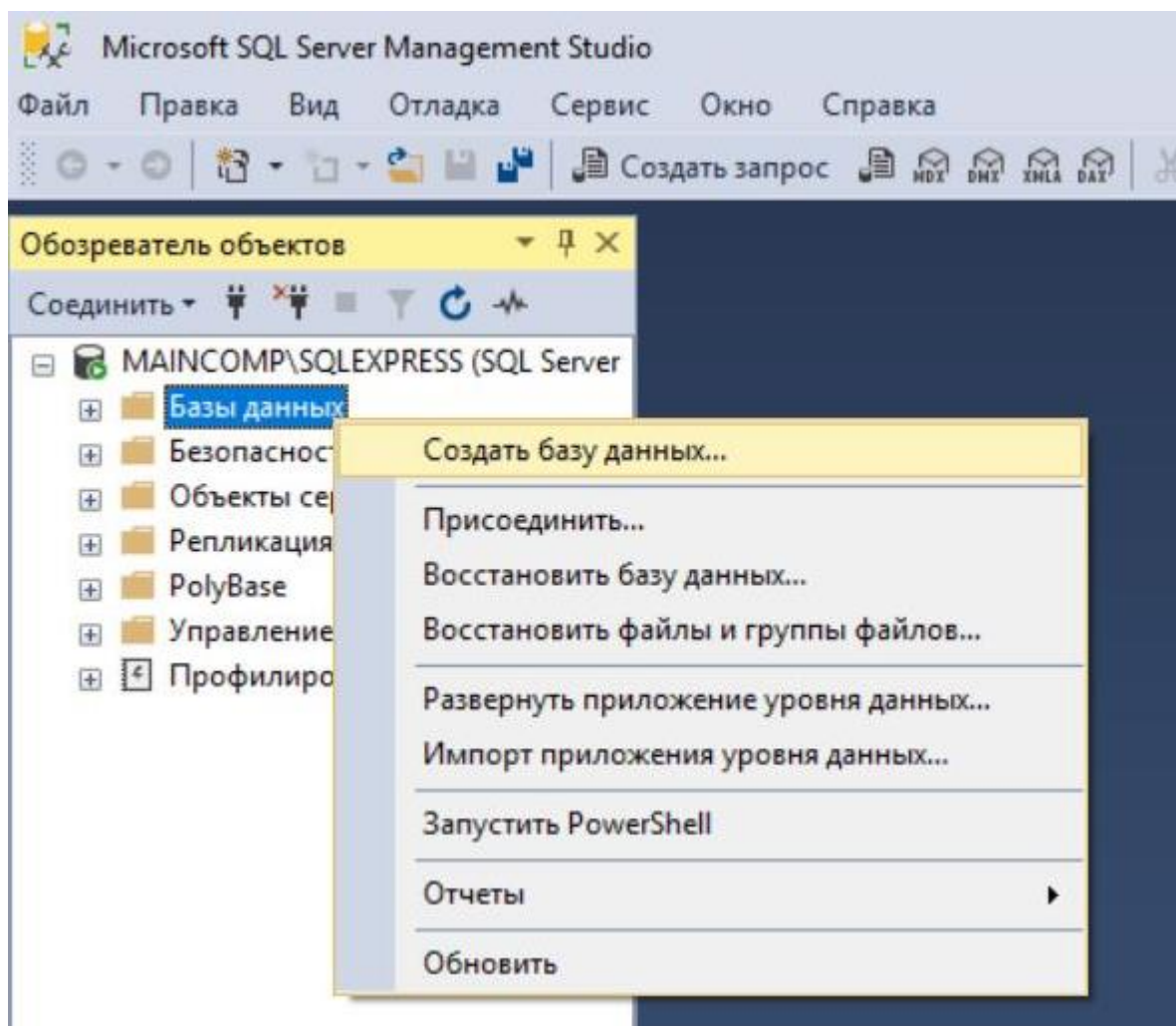


Рис. 2.2 Окно создание базы данных

В результате откроется окно «Создание базы данных». Здесь обязательно нужно заполнить только поле «Имя базы данных», остальные параметры настраиваются по необходимости. После того, как Вы ввели имя БД, нажимайте «ОК».

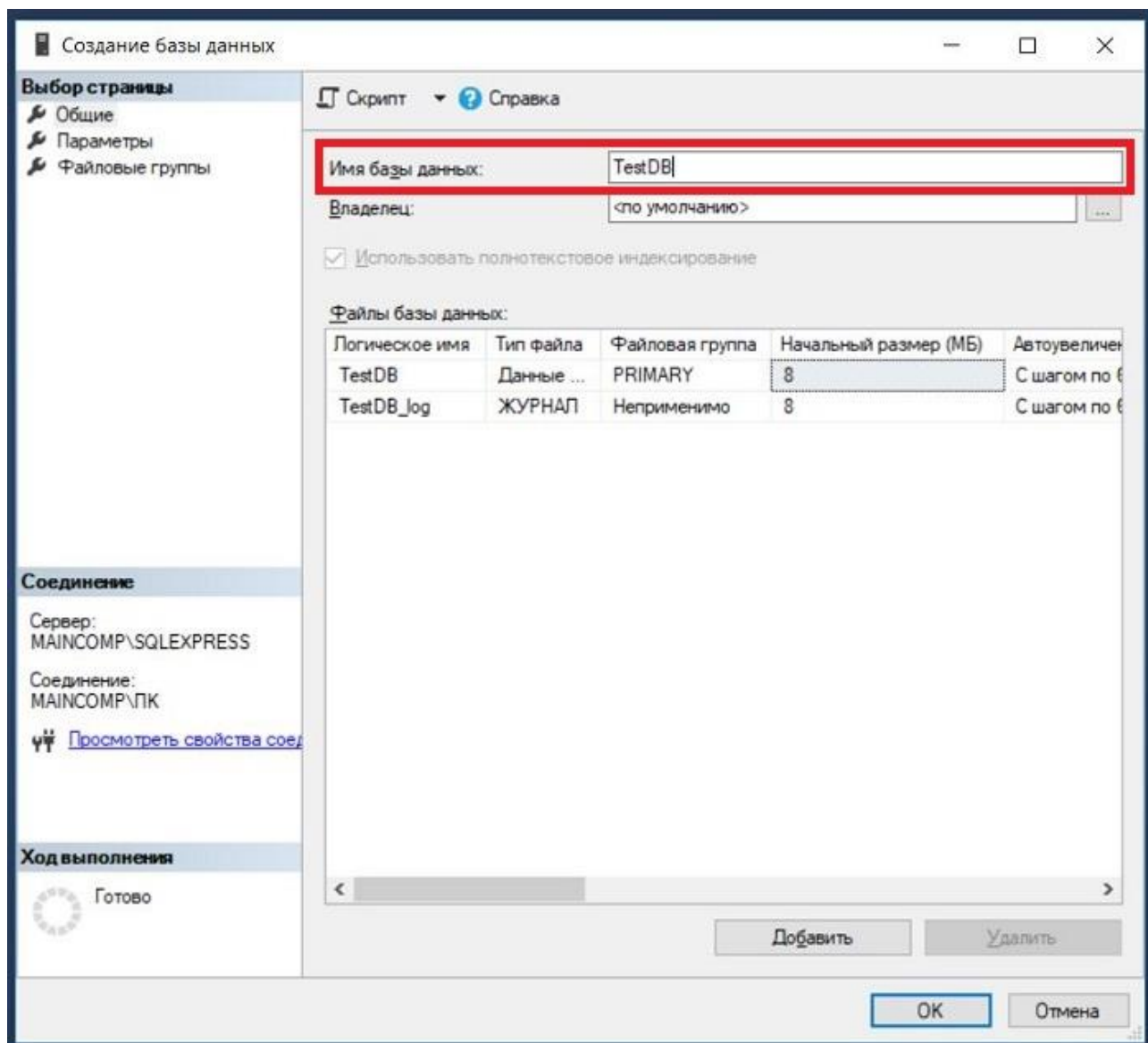


Рис. 2.3 Окно создания базы данных

Если БД с таким именем на сервере еще нет, то она будет создана, в обозревателе объектов она сразу отобразится.

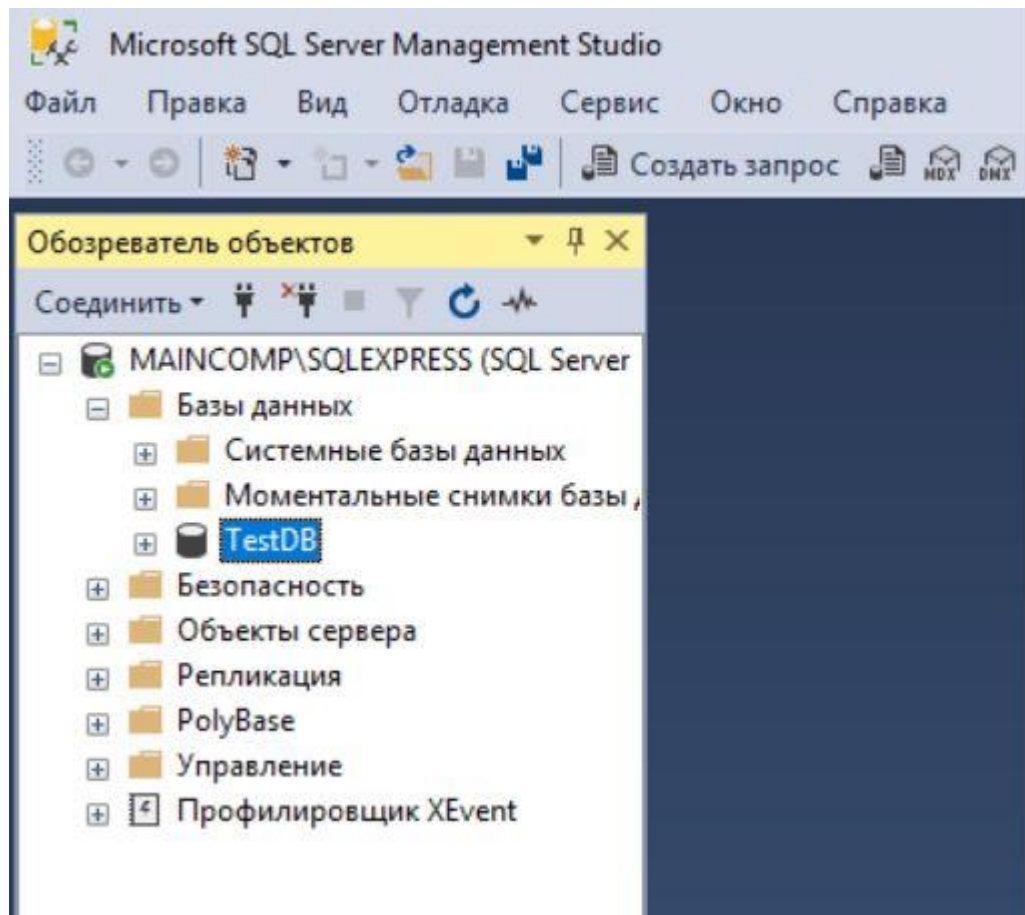


Рис. 2.4 Создание БД

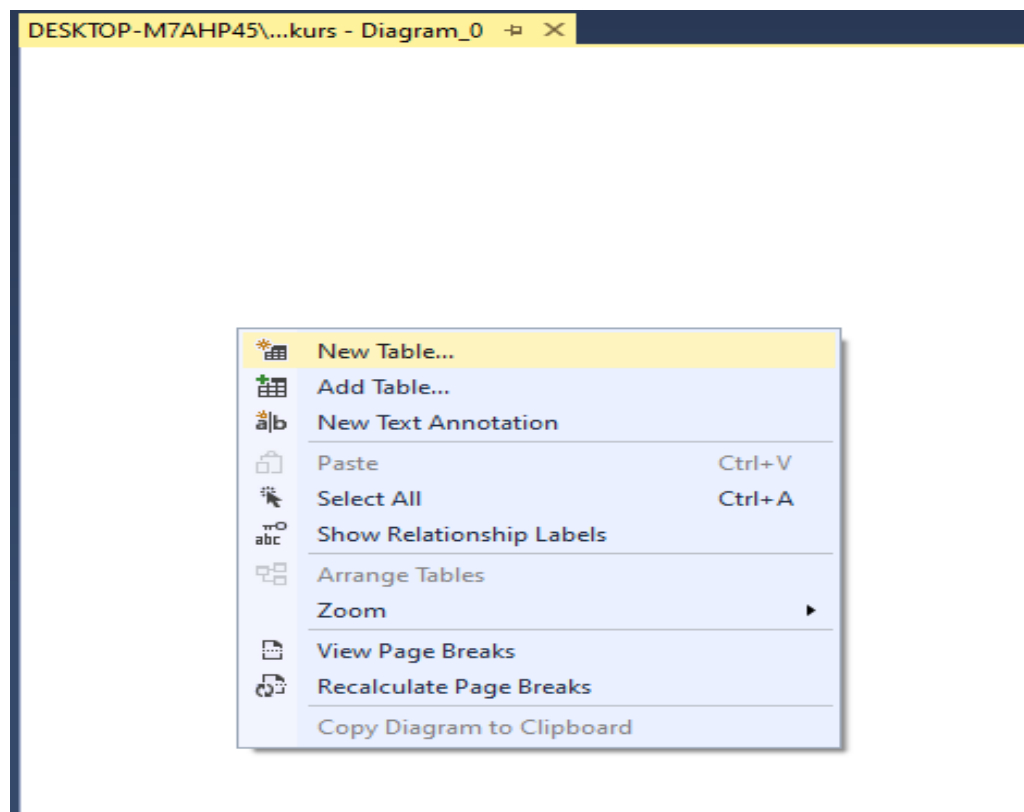


Рис. 2.5 создание таблицы в SSMS

User			
	Column Name	Data Type	Allow Nulls
🔑	ID	int	<input type="checkbox"/>
	Login	nvarchar(50)	<input type="checkbox"/>
	Password	nvarchar(50)	<input type="checkbox"/>
	RoleID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Рис. 2.6 Созданная «сущность» пользователя в БД

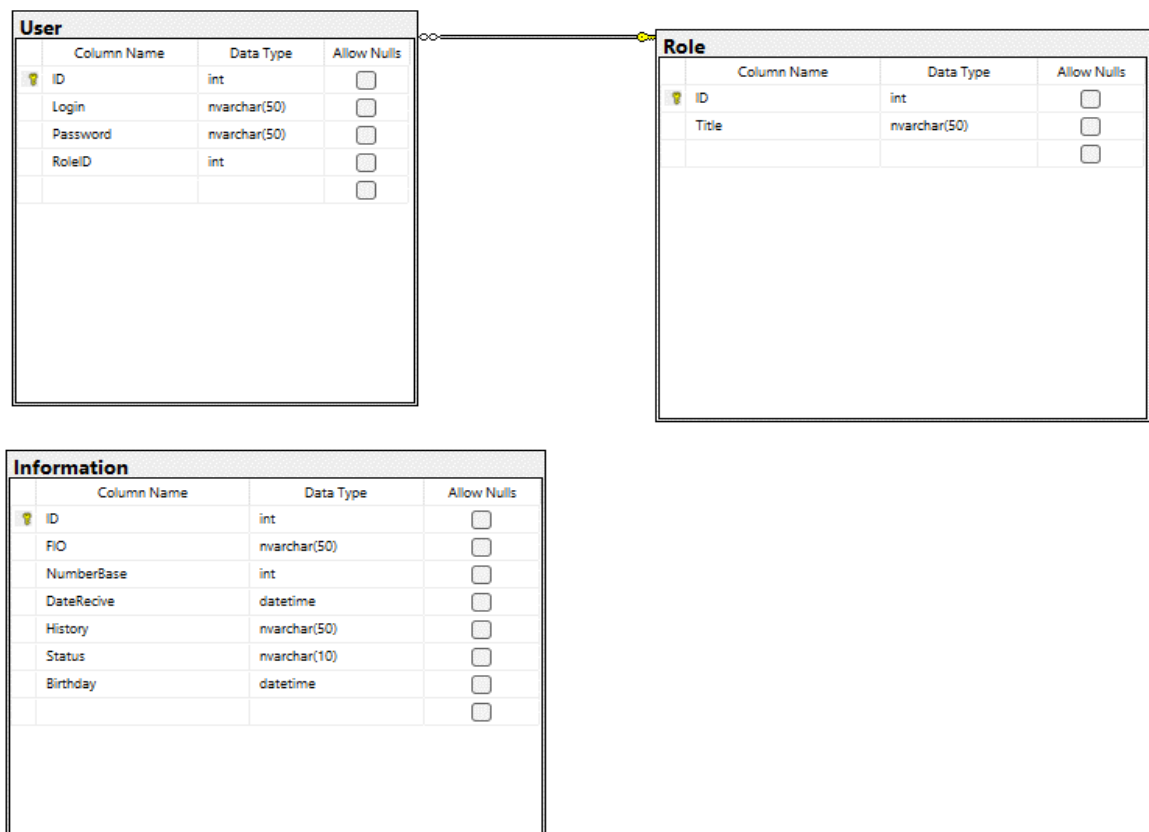


Рис. 2.7 Созданная таблица информационной системы “поликлиника”

	ID	Login	Password	RoleID
	1	Admin	111	1
	2	User	222	2
	3	423	123234	1

Рис. 2.8 Сущность “пользователь” базы данных с внесёнными данными

ГЛАВА 3. РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ

3.1 Разработка интерфейса информационной системы

Для разработки интерфейса информационной системы я использовал программу Visual studio. Язык программирования C# + wpf. В моей информационной системе “Поликлиника” я использовал 10 окон, а именно: вкладка авторизации, главное окно, окно редактирования, добавления для главного окна, окно сотрудников, окно редактирования сотрудников, окно добавления сотрудников, окно новостей и окно настроек.

На “рис. 3.1” находится окно авторизации в информационную систему “поликлиника”. На данном окне поля логин и код заблокированы до той поры, пока пользователь не введёт пароль, затем после нажатия на “пробел” включается поле пароль и блокируется поле логина, далее идёт проверка логина и пароля, после успешной проверки становится доступно поле “код”. Код для входа генерируется автоматически один раз в 10 секунд с выводом сообщения, также можно обновлять код вручную.

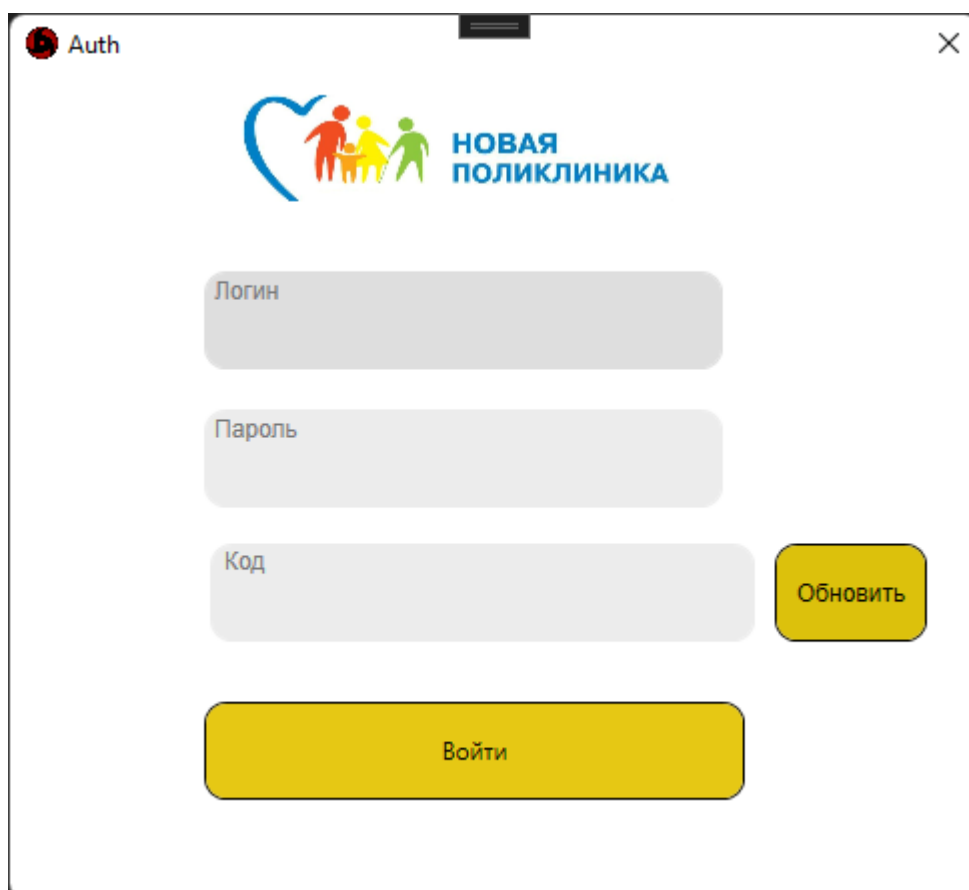


Рис. 3.1 Окно входа

Код, который отвечает за авторизацию, а именно поля ввода логина, пароля и кода представлен ниже в формате “.xaml”:

```

<Grid>
  <Button Style="{DynamicResource ButtonStyle1}" Foreground="Black" Content="Войти"
HorizontalAlignment="Left" Margin="97,313,0,0" VerticalAlignment="Top" Background="#FFE6C814"
BorderBrush="Black" Height="49" Width="270" Click="Sign_In"/>
  <Label FontFamily="Arial" Content="Вход в &#xD;&#xA;АИС &#xD;&#xA;Поликлиника"
FontSize="24" HorizontalAlignment="Left" Margin="144,0,0,0" VerticalAlignment="Top" Height="91"
Width="176"/>
  <PasswordBox x:Name="Password" Style="{DynamicResource PasswordBoxStyle1}"
FontFamily="Arial" FontSize="22" KeyUp="Password_KeyUp" IsEnabled="False"
HorizontalContentAlignment="Left" VerticalContentAlignment="Bottom" Margin="97,167,128,0"
VerticalAlignment="Top" Height="49"/>
  <TextBox x:Name="Login" Style="{DynamicResource TextBoxStyle1}" FontFamily="Arial"
FontSize="22" KeyUp="Login_KeyUp" IsEnabled="True" HorizontalContentAlignment="Left"
VerticalContentAlignment="Bottom" Margin="97,98,128,0" TextWrapping="NoWrap" VerticalAlignment="Top"
TextChanged="TextBox_TextChanged" Height="49"/>
  <Label FontFamily="Arial" Content="Логин" HorizontalAlignment="Left" Foreground="#787878"
Margin="97,96,0,0" VerticalAlignment="Top" Height="23"/>
  <Label FontFamily="Arial" Content="Пароль" HorizontalAlignment="Left" Foreground="#787878"
Margin="97,165,0,0" VerticalAlignment="Top" Height="27"/>
  <Button x:Name="RefreshKnopka" Style="{DynamicResource ButtonStyle2}" FontFamily="Arial"
Content="Обновить" HorizontalAlignment="Left" Background="#FFDCC10C" BorderBrush="Black"
Foreground="Black" Margin="382,234,0,0" VerticalAlignment="Top" Height="49" Width="76" Click="Refresh"/>
  <TextBox x:Name="CodeBox" Style="{DynamicResource TextBoxStyle2}" FontFamily="Arial"
MaxLength="4" KeyUp="CodeBox_KeyUp" IsEnabled="False" FontSize="22" HorizontalContentAlignment="Left"
VerticalContentAlignment="Bottom" HorizontalAlignment="Left" Margin="100,234,0,0" TextWrapping="Wrap"
VerticalAlignment="Top" Width="272" Height="49" TextChanged="Codek"/>
  <Label FontFamily="Arial" Content="Код" HorizontalAlignment="Left" Foreground="#787878"
Margin="102,231,0,0" VerticalAlignment="Top" Height="26"/>
</Grid>

```

На “рис. 3.2” представлено главное окно, которое появляется после успешного прохождения авторизации. На главном окне представлена база данных пациентов, которую может редактировать, удалять записи, изменять конкретную запись пользователь, чей уровень доступа является “администратор”. Для обычного пользователя имеется доступ к просмотру базы данных пациентов, без возможности изменять её, но с возможностью формировать отчёт в формате “.docx” с дальнейшей возможностью сохранения отчёта в любое удобное место. С главного окна возможно переключаться на другие окна системы, такие как “Сотрудники”, “Новости” или “Настройки”. Выполнить переход на другие окна возможно с каждого окна системы, кроме окна “Авторизация”, “Добавление” пользователя или пациента, “Редактирование” пользователя или пациента. Так же по аналогии выполнено окно “Сотрудники”, на которой находится база данных с пользователями системы.

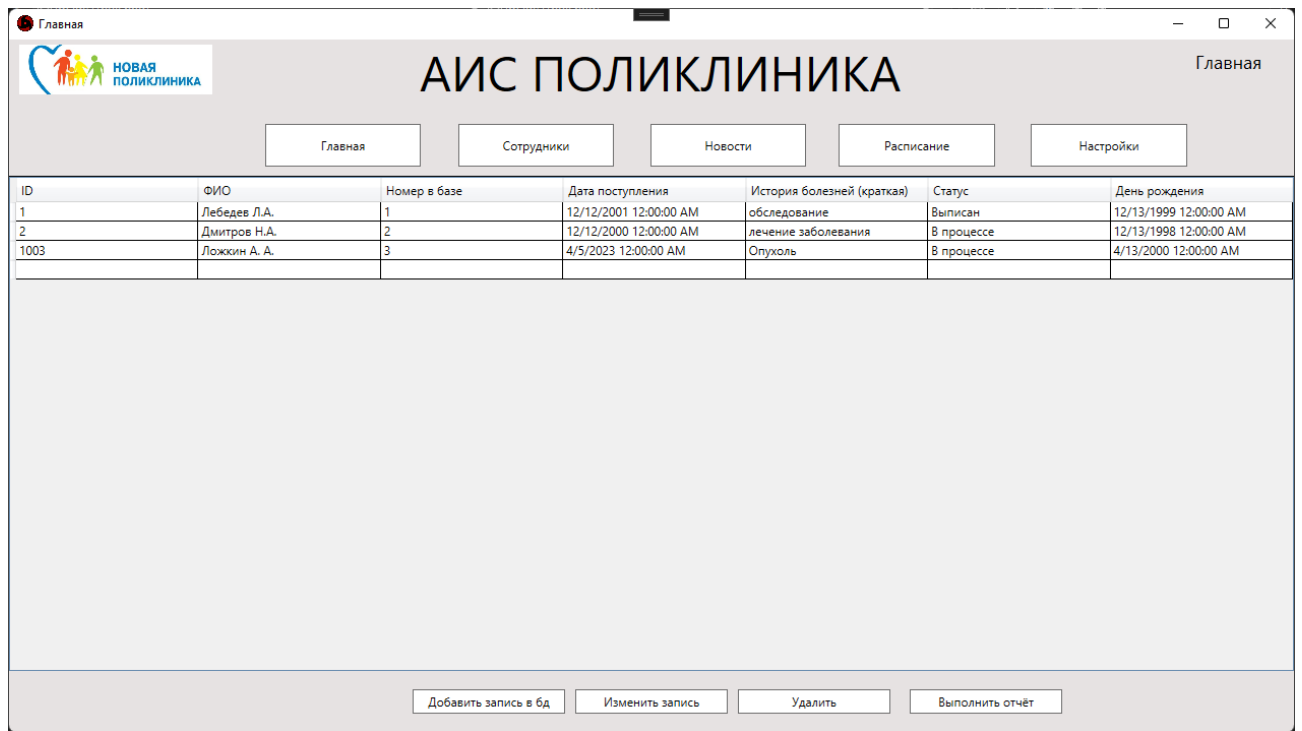


Рис. 3.2 Главное окно с базой данных пациентов

Код, который отвечает за главное окно, вывод и редактирование базы данных пациентов представлен ниже в формате “.xaml”:

```
<Grid Background="#FFE6E2E2">
  <DataGrid Grid.Row="1"
    AutoGenerateColumns="False"
    x:Name="UsersGrid">
    <DataGrid.Columns>
      <DataGridTextColumn Header="ID" Width="*" Binding="{Binding ID}"/>
      <DataGridTextColumn Header="ФИО" Width="*" Binding="{Binding FIO}"/>
      <DataGridTextColumn Header="Номер в базе" Width="*" Binding="{Binding NumberBase}"/>
      <DataGridTextColumn Header="Дата поступления" Width="*" Binding="{Binding
DateRecive}"/>
      <DataGridTextColumn Header="История болезней (краткая)" Width="*" Binding="{Binding
History}"/>
      <DataGridTextColumn Header="Статус" Width="*" Binding="{Binding Status}"/>
      <DataGridTextColumn Header="День рождения" Width="*" Binding="{Binding Birthday}"/>
    </DataGrid.Columns>
  </DataGrid>
  <StackPanel Grid.Row="2"
    HorizontalAlignment="Center"
    Orientation="Horizontal"
    VerticalAlignment="Center">
    <Button Height="24"
      Click="Add_Btn_Click"
      x:Name="AddBtn"
      Content="Добавить запись в бд"
      Width="150"
      Background="White"/>
    <Button Height="24"
      Click="Add_Btn_Click11"
      x:Name="EditBtn"
      Margin="10 0"
      Content="Изменить запись"/>
  </StackPanel>
</Grid>
```

```

        Width="150"
        Background="White"/>
    <Button Height="24"
        Click="RemoveBtn_Click"
        x:Name="RemoveBtn"
        Content="Удалить"
        Width="150"
        Background="White"/>
</StackPanel>
<Image Source="/left.png" Stretch="Fill" Margin="0,0,0,425"/>
<Button x:Name="glavnaya" Content="Главная" Margin="252,83,0,10" Background="White"
HorizontalAlignment="Left" Width="153"/>
    <Button x:Name="sotrudnichki" Content="Сотрудники" Margin="442,83,0,10"
Click="sotrudnichki_Click" Background="White" HorizontalAlignment="Left" Width="153"/>
    <Button x:Name="prikazy" Content="Новости" Margin="631,83,0,10" Click="prikazy_Click"
Background="White" HorizontalAlignment="Left" Width="153"/>
    <Button x:Name="otchety" Content="Отчёты" Margin="1280,0,0,134" Click="otchety_Click"
Background="White" HorizontalAlignment="Left" Width="0"/>
    <Button x:Name="raspisanye" Content="Расписание" Margin="816,83,0,10" Click="raspisanye_Click"
Background="White" HorizontalAlignment="Left" Width="154"/>
    <Button x:Name="settings" Content="Настройки" Margin="1005,83,0,10" Click="settings_Click"
Background="White" HorizontalAlignment="Left" Width="154"/>
    <Button Content="" HorizontalAlignment="Center" Margin="0,134,0,0" VerticalAlignment="Top"
Width="1280" Height="1" BorderBrush="Black" Background="Black" RenderTransformOrigin="0.5,0.5">
        <Button.RenderTransform>
            <TransformGroup>
                <ScaleTransform ScaleY="-1"/>
                <SkewTransform/>
                <RotateTransform/>
                <TranslateTransform/>
            </TransformGroup>
        </Button.RenderTransform>
    </Button>
    <TextBlock Margin="404,-1,382,72" TextWrapping="Wrap" FontSize="48"><Run Language="ru-ru"
Text="АИС ПОЛИКЛИНИКА"/></TextBlock>
    <TextBlock Margin="1167,10,23,96" TextWrapping="Wrap" FontSize="18"><Run Language="ru-ru"
Text="Главная"/></TextBlock>
    <Button Content="Выполнить отчёт" HorizontalAlignment="Left" Margin="886,0,0,0" Grid.Row="2"
VerticalAlignment="Center" Click="Export_Click" Height="24" Width="150" Background="White"/>
    <Grid.RowDefinitions>
        <RowDefinition Height="135"/>
        <RowDefinition/>
        <RowDefinition Height="60"/>
    </Grid.RowDefinitions>
</Grid>

```

На “рис. 3.3” представлено окно добавления нового пользователя в систему. На данном окне реализованы поля ввода логина, пароля и роли нового пользователя, которого возможно будет добавить в систему. Ввод роли реализован через “ComboBox” – элемент, который позволяет выбирать из списка готов вариантов нужный вариант в той или иной ситуации. В данном элементе находится выбор роли пользователя: “администратор” или “пользователь”. От выбора роли меняется возможности работника при использовании автоматизированной информационной системы. Кнопка “Сохранить” сохраняет нового пользователя в базу данных, а кнопка “Назад” возвращает пользователя на окно назад.

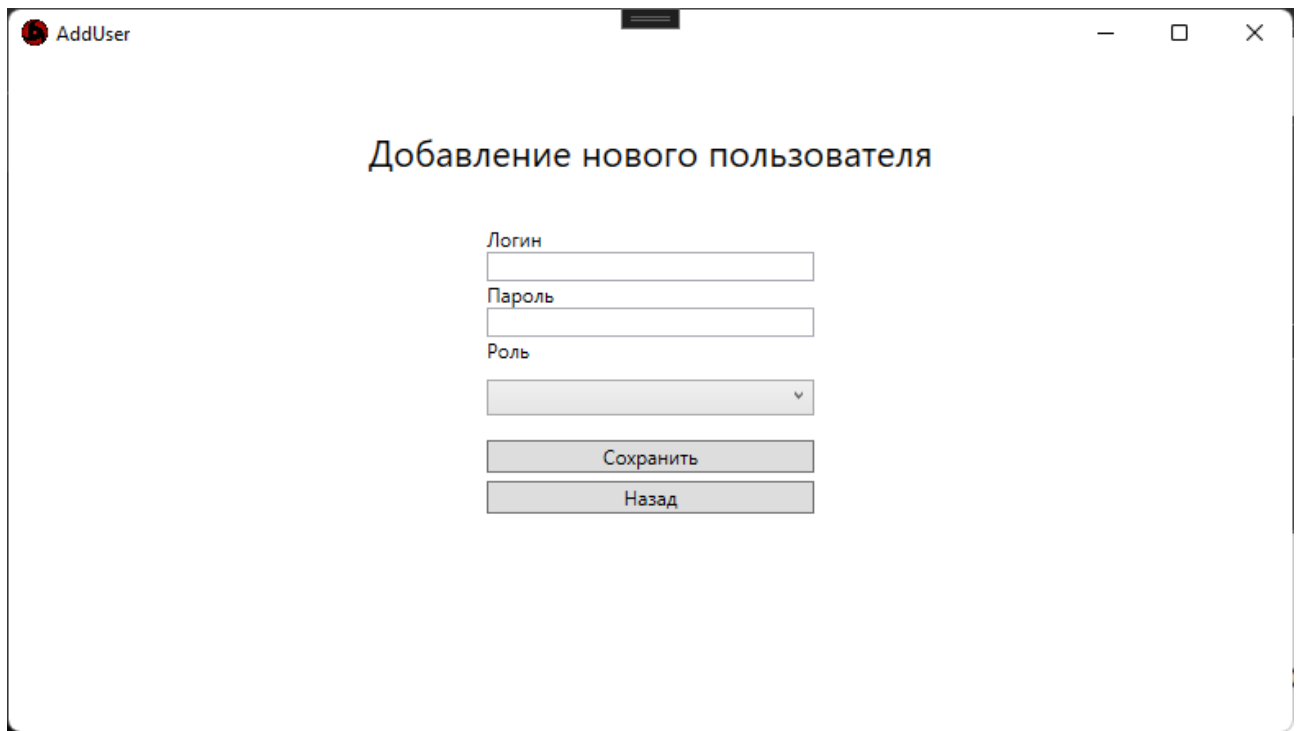


Рис. 3.3 Окно добавления нового пользователя в базу данных

Код, который отвечает за окно добавления нового пользователя в базу данных представлен ниже в формате “.xaml”:

```
<Grid>
<StackPanel VerticalAlignment="Top"
HorizontalAlignment="Center"
MinWidth="200" Margin="0,102,0,0">

<TextBlock Text="Логин"/>
<TextBox x:Name="LoginTxb1"/>

<TextBlock Text="Пароль"/>
<TextBox x:Name="PassTxb1"/>

<TextBlock Text="Роль"/>
<ComboBox x:Name="RoleTxb1"
DisplayMemberPath="Title"
Margin="0 10"/>

<Button Content="Сохранить"
Click="SaveClick" Margin="0 5 0 0"/>

<Button Content="Назад"
Click="Goba_Click" Margin="0 5 0 0"/>

</StackPanel>
<TextBlock HorizontalAlignment="Center" Margin="0,29,0,0" TextWrapping="Wrap"
VerticalAlignment="Top" FontSize="22"><Run Language="ru-ru" Text="Редактирование
пользователей"/></TextBlock>
</Grid>
```

Окно, позволяющее редактировать текущего пользователя показано на “рис. 3.4”. На данном окне поля, похожие на “рис. 3.3”, но в них уже есть информация, которую можно отредактировать. Вместо элемента “ComboBox” размещено поле ввода. Для изменения роли необходимо выбрать от 1 до 2, где 1 – это администратор системы, а 2 – это пользователь системы. Кнопки “Сохранить” и “Назад” работают точно также, как на окне добавления пользователя. Окно редактирования пациентов оформлено в том же стиле и имеет те же функции, что и окно пользователей.

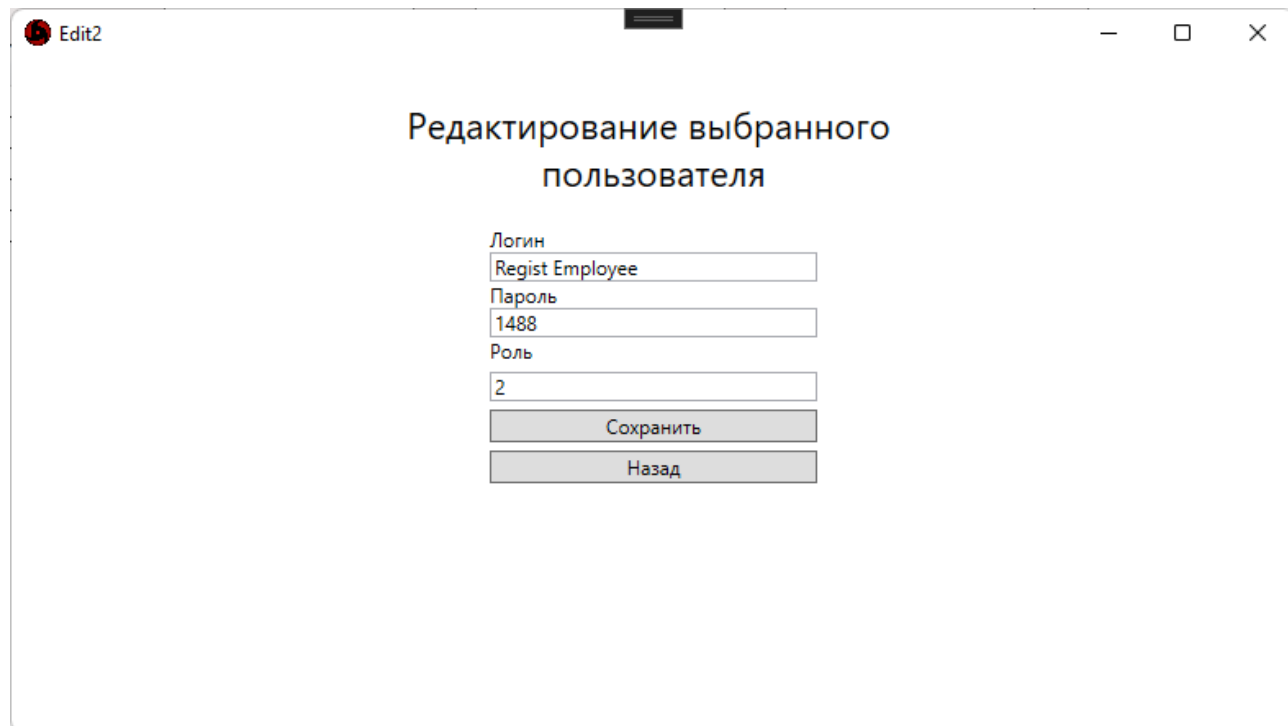


Рис. 3.4 Окно редактирования пользователей в базе данных

Код, который отвечает за редактирование пользователей и сохранение нового пользователя системы в базу данных пользователей представлен ниже в формате “.xaml”:

```
<Grid>
    <StackPanel VerticalAlignment="Top"
        HorizontalAlignment="Center"
        MinWidth="200" Margin="0,102,0,0">

        <TextBlock Text="Логин"/>
        <TextBox x:Name="LoginTxb2" Text="{Binding Login}"/>

        <TextBlock Text="Пароль"/>
        <TextBox x:Name="PassTxb2" Text="{Binding Password}"/>

        <TextBlock Text="Роль"/>
        <TextBox HorizontalAlignment="Left" Margin="0,5,0,0" TextWrapping="Wrap"
            VerticalAlignment="Top" Width="200" x:Name="RoleTXB2" Text="{Binding RoleID}"/>

        <Button Content="Сохранить"
```

```

Click="Update_Btn" Margin="0 5 0 0"/>

<Button Content="Назад"
Click="Cancel_Btn" Margin="0 5 0 0"/>

</StackPanel>
<TextBlock HorizontalAlignment="Center" Margin="0,29,0,0" TextWrapping="Wrap"
VerticalAlignment="Top" FontSize="22"><Run Language="ru-ru" Text="Редактирование
пользователей"/></TextBlock>

</Grid>

```

3.2 Программирование информационной системы

Для работы интерфейса необходимо запрограммировать элементы окон, с которыми взаимодействует пользователь системы. Для решения этого используется язык программирования “C#” вместе с “WPF” – графической составляющей системы. Ниже будут представлены основные и важные части кода для программирования действий, совершаемых над системой пользователями.

Ниже представлен код окна “авторизация”, который позволяет пользователю входить в систему со своим логином и паролем, а также использовать капчу при входе.

Фрагмент кода, с помощью которого проверяется корректность введения пользователем логина:

```

private void Login_KeyUp(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Enter)
    {
        using (var db = new MedTestEntities())
        {
            var login = db.User.AsNoTracking().FirstOrDefault(l => l.Login == Login.Text.Trim());
            if (login == null)
            {
                MessageBox.Show("Неверный логин");
            }
            else
            {
                Password.IsEnabled = true;
                Login.IsEnabled = false;
                CodeBox.IsEnabled = false;
                Password.Focus();
            }
            Globals.Role = login.RoleID;
            Globals.userinfo = login;
        }
    }
}

```

Фрагмент кода, с помощью которого проверяется корректность введения пользователем пароля:

```

private void Password_KeyUp(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Enter)

```



```

        {
            using (var db = new MedTestEntities())
            {
                var login1 = db.User.AsNoTracking().FirstOrDefault(l => l.Login == Login.Text.Trim() &
l.Password == Password.Password);
                if (login1 == null)
                {
                    MessageBox.Show("Неверный пароль");
                }
                else
                {
                    Password.IsEnabled = false;
                    CodeBox.IsEnabled = true;
                    Login.IsEnabled = false;
                    gencode();
                    CodeBox.Focus();
                }
                Globals.Role = login1.RoleID;
                Globals.userinfo = login1;
            }
        }
    }
}

```

Фрагмент кода, с помощью которого проверяется правильность введения капчи
ПОЛЬЗОВАТЕЛЕМ:

```

private void Sign_In(object sender, RoutedEventArgs e)
{
    using (var db = new MedTestEntities())
    {
        var auth = db.User.AsNoTracking().FirstOrDefault(m => m.Login == Login.Text && m.Password ==
Password.Password);
        if (auth != null & code == CodeBox.Text)
        {
            timer.Stop();
            Globals.Role = auth.RoleID;
            Globals.userinfo = auth;
            Window1 winavto = new Window1();
            winavto.Show();
            Close();
        }
        else
        {
            MessageBox.Show("Неверный код, повторите попытку!", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Warning);
            timer.Stop();
        }
    }
}

```

Фрагмент кода, с помощью которого генерируется код капчи для входа:

```

private void gencode()
{
    code = null;
    Random random = new Random();
    string[] massiveCharacter = new string[] { "1", "2", "3", "4", "5", "6", "7", "8", "9", "a", "B", "c", "d", "E",
"F", "j", "f", "z", "f", "S",
"t", "T", "Y", "y", "D", "d", "l", "L", "H", "h", "A", "m", "M", "n", "N", };
    for (int i = 0; i < 4; i++)
    {
        code += massiveCharacter[random.Next(0, massiveCharacter.Length)];
    }
}

```

```

        if (MessageBox.Show(code.ToString(), "Code", MessageBoxButton.OK, MessageBoxImage.Warning)
== DialogResult.OK)
        {
            timer.Interval = TimeSpan.FromSeconds(10);
            timer.Tick += Timer_Tick;
            timer.Start();

            CodeBox.IsEnabled = true;
            Login.IsEnabled = false;
            RefreshKnopka.IsEnabled = true;

        }
    }
}

```

Фрагмент кода, с помощью которого обновляется код капчи:

```

private void Refresh(object sender, RoutedEventArgs e)
{
    timer.Stop();
    gencode();
    CodeBox.Focus();
}

```

Ниже представлен код окна “главное окно”, который позволяет пользователю или администратору (в зависимости от прав пользования) просматривать таблицу пациентов, переходить на другие вкладки системы, формировать отчёты, редактировать таблицу пациентов, удалять и добавлять новых пациентов в базу данных системы. По аналогии с кодом “главного окна” выполнено окно “сотрудники”, в котором совершаются те же взаимодействия что и в “главном окне”.

Фрагмент кода, который разграничивает права доступа в зависимости от роли пользователя:

```

public Window1()
{
    InitializeComponent();

    if (MainWindow.Globals.Role == 1)
    {
        AddBtn.Visibility = Visibility.Visible;
        EditBtn.Visibility = Visibility.Visible;
        RemoveBtn.Visibility = Visibility.Visible;
    }
    else
    {
        AddBtn.Visibility = Visibility.Collapsed;
        EditBtn.Visibility = Visibility.Collapsed;
        RemoveBtn.Visibility = Visibility.Collapsed;
    }
}

```

Фрагмент кода, который по нажатию на кнопки переносит пользователя на другие окна системы:

```

private void sotrudnichki_Click(object sender, RoutedEventArgs e)
{
    Window2 win2 = new Window2();
    win2.Show();
    this.Close();
}

```

```

private void prikazy_Click(object sender, RoutedEventArgs e)
{
    Window3 win3 = new Window3();
    win3.Show();
    this.Close();
}

private void otchety_Click(object sender, RoutedEventArgs e)
{
    Window4 win4 = new Window4();
    win4.Show();
    this.Close();
}

private void raspisanye_Click(object sender, RoutedEventArgs e)
{
    Window5 win5 = new Window5();
    win5.Show();
    this.Close();
}

private void settings_Click(object sender, RoutedEventArgs e)
{
    Window6 win6 = new Window6();
    win6.Show();
    this.Close();
}

```

Фрагмент кода, который добавляет нового пациента, изменяет выбранного и удаляет выбранного, то есть редактирует записи из таблицы пациентов:

```

private void Add_Btn_Click(object sender, RoutedEventArgs e)
{
    AddBtn win7 = new AddBtn();
    win7.Show();
    this.Close();
}

private void Add_Btn_Click11(object sender, RoutedEventArgs e)
{
    if (UsersGrid.SelectedItem != null)
    {
        Edit1 edit = new Edit1(UsersGrid.SelectedItem as Information);
        edit.ShowDialog();
        AppData.db.SaveChanges();
    }
    else
    {
        MessageBox.Show("Выберите пользователя");
    }
    MedTestEntities.GetContext().ChangeTracker.Entries().ToList().ForEach(p => p.Reload());
}

private void RemoveBtn_Click(object sender, RoutedEventArgs e)
{
    if (MessageBox.Show("Вы действительно хотите удалить пациента?", "Уведомление",
        MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)
    {
        var CurrentPac = UsersGrid.SelectedItem as Information;
        AppData.db.Information.Remove(CurrentPac);
        AppData.db.SaveChanges();

        UsersGrid.ItemsSource = AppData.db.Information.ToList();
    }
}

```

```

        MessageBox.Show("Успешно");
    }

```

Фрагмент кода, который формирует отчёты по таблице пациентов:

```

private void Export_Click(object sender, RoutedEventArgs e)
{
    var allZep = MedTestEntities.GetContext().Information.ToList();
    var allpac = MedTestEntities.GetContext().Information.ToList();

    var appl = new Word.Application();

    Word.Document document = appl.Documents.Add();

    Word.Paragraph userParagraph = document.Paragraphs.Add();
    Word.Range userRange = userParagraph.Range;
    userRange.Text = "Отчёт о пациентах";

    userRange.InsertParagraphAfter();

    Word.Paragraph tableParagraph = document.Paragraphs.Add();
    Word.Range tableRange = tableParagraph.Range;
    Word.Table paymentsTable = document.Tables.Add(tableRange, allZep.Count() + 1, 7);
    paymentsTable.Borders.InsideLineStyle = paymentsTable.Borders.OutsideLineStyle
        = Word.WdLineStyle.wdLineStyleSingle;
    paymentsTable.Range.Cells.VerticalAlignment
Word.WdCellVerticalAlignment.wdCellAlignVerticalCenter;

    Word.Range cellRange;

    cellRange = paymentsTable.Cell(1, 1).Range;
    cellRange.Text = "ID";
    cellRange = paymentsTable.Cell(1, 2).Range;
    cellRange.Text = "ФИО пациента";
    cellRange = paymentsTable.Cell(1, 3).Range;
    cellRange.Text = "Номер в базе";
    cellRange = paymentsTable.Cell(1, 4).Range;
    cellRange.Text = "Дата поступления";
    cellRange = paymentsTable.Cell(1, 5).Range;
    cellRange.Text = "История лечения";
    cellRange = paymentsTable.Cell(1, 6).Range;
    cellRange.Text = "Статус";
    cellRange = paymentsTable.Cell(1, 7).Range;
    cellRange.Text = "День рождения";

    paymentsTable.Rows[1].Range.Bold = 1;
    paymentsTable.Rows[1].Range.ParagraphFormat.Alignment
Word.WdParagraphAlignment.wdAlignParagraphCenter;

    for (int i = 0; i < allZep.Count(); i++)
    {
        var currentCategory = allZep[i];
        cellRange = paymentsTable.Cell(i + 2, 1).Range;
        cellRange.Text = Convert.ToString(currentCategory.ID);
        cellRange.ParagraphFormat.Alignment = Word.WdParagraphAlignment.wdAlignParagraphCenter;

        cellRange = paymentsTable.Cell(i + 2, 2).Range;
        cellRange.Text = Convert.ToString(currentCategory.FIO);

        cellRange = paymentsTable.Cell(i + 2, 3).Range;
        cellRange.Text = Convert.ToString(currentCategory.NumberBase);

        cellRange = paymentsTable.Cell(i + 2, 4).Range;

```

```

        cellRange.Text = Convert.ToString(currentCategory.DateRecive.ToString("dd.MM.yyyy"));

        cellRange = paymentsTable.Cell(i + 2, 5).Range;
        cellRange.Text = Convert.ToString(currentCategory.History);

        cellRange = paymentsTable.Cell(i + 2, 6).Range;
        cellRange.Text = Convert.ToString(currentCategory.Status);

        cellRange = paymentsTable.Cell(i + 2, 7).Range;
        cellRange.Text = currentCategory.Birthday.ToString("dd.MM.yyyy");
    }

    appl.Visible = true;
}
}
}

```

Ниже представлен код окна “добавление пользователя”, который позволяет добавлять в базу данных пользователей с разграничением прав доступа. По аналогии с кодом окна “добавление пользователя” выполнено окно “добавление пациента”, в котором совершаются те же взаимодействия, что и в окне “добавление пациента”.

Фрагмент кода, который подключает приложение к базе данных пользователей:

```

public AddUser()
{
    InitializeComponent();
    RoleTxb1.ItemsSource = AppData.db.Role.ToList();
}

```

Фрагмент кода, который выполняет по нажатию кнопки сохранение нового пользователя в базу данных:

```

private void SaveClick(object sender, RoutedEventArgs e)
{
    User user = new User();

    user.Login = LoginTxb1.Text;
    user.Password = PassTxb1.Text;

    var CurrentRole = RoleTxb1.SelectedItem as Role;
    user.RoleID = CurrentRole.ID;

    AppData.db.User.Add(user);
    AppData.db.SaveChanges();
    MessageBox.Show("Пользователь был добавлен в базу");
}

```

Фрагмент кода, который по нажатию кнопки возвращает пользователя на прошлое окно:

```

private void Goba_Click(object sender, RoutedEventArgs e)
{
    Window2 winUs = new Window2();
    winUs.Show();
    this.Close();
}
}
}

```

Ниже представлен код окна “редактирование текущего пользователя”, который позволяет редактировать выбранную запись для внесения изменений в неё с последующим

сохранением изменений в базе данных. По аналогии с окном “редактирование текущего пользователя” выполнено окно “редактирование текущего пациента”, в котором совершаются те же взаимодействия, что и в окне “редактирование текущего пациента”.

Фрагмент кода, который подключается к базе данных и заполняет ячейки окна из уже готовых данных базы пользователей:

```
public Edit2(User selectedUser)
{
    currentUser = selectedUser;
    InitializeComponent();
    DataContext = currentUser;
    LoginTxb2.Text = currentUser.Login;
    PassTxb2.Text = currentUser.Password;
    RoleTXB2.Text = Convert.ToString(currentUser.RoleID);
}
```

Фрагмент кода, который по нажатию кнопки обновляет изменённые данные в таблице пользователей базы данных и в самом приложении:

```
private void Update_Btn(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Данные успешно изменены");
    Close();
}
```

Фрагмент кода, который по нажатию кнопки возвращает пользователя на прошлое окно:

```
private void Cancel_Btn(object sender, RoutedEventArgs e)
{
    Close();
}
```

ЗАКЛЮЧЕНИЕ

По итогу проделанной работы можно сказать, что получилось хорошее приложение для информационной системы «Поликлиника», которое обеспечивает пользователю удобство при работе с приложением.

Поставленные цели и задачи при проектировании информационной системы были выполнены в полной мере, а именно выполнены следующие задачи:

- создана система регистрации клиентов/больных в базу данных поликлиники;
- создан учёт медицинских карт в системе учёта;
- построена концептуальная информационная модель;
- сформирована физическая структура базы данных;
- реализовано простое пользовательское приложение.

Работая над курсовым проектом, были сделаны следующие основные выводы, что каждая автоматизированная система должна иметь:

- идейную ценность, в виде решения какой-либо проблемы;
- представление о том, как будет выглядеть конечный продукт;
- технические требования к реализации;
- план разработки, в котором описаны все ступени разработки проекта;
- основные инструменты разработки;

В первой части были рассмотрены:

- инструменты разработки приложения;
- языки программирования, используемые для создания приложения;
- инструменты разработки базы данных для автоматизированной системы;

Во второй части были рассмотрены:

- диаграмма ER – сущность связь;
- процесс разработки базы данных по модели диаграммы ER;

В третьей части были рассмотрены:

- процесс создания графического интерфейса приложения;
- процесс программирования приложения для корректной работы;

Ссылка на репозиторий на GitHub: https://github.com/Kot-Patriot/Kursovoy_project-Raykov

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Трей Нэш. Ускоренный курс по языку программирования C# : ускоренный курс/ Трей Нэш. - США: Apress, 2008. – 576 с.
2. Джон Скит. «C# для профессионалов: тонкости программирования» (C# in Depth): пособие/ Джон скит. - США.: MANNING, 2014. –608 с.

Интернет - ресурсы

1. Помощь по ошибкам и инструкции по написанию кода от “Майкрософт”: [Режим доступа]. - <https://learn.microsoft.com/en-us/dotnet/tutorials/c#> (дата обращения 12.04.2023)
2. SQL Server Management Studio (SSMS), работа с базами данных от “Майкрософт”: [Режим доступа]. - <https://learn.microsoft.com/en-us/sql/ssms> (дата обращения 11.10.2022)
3. Visual Studio IDE, среда разработки от “Майкрософт”: [Режим доступа]. - <https://visualstudio.microsoft.com/ru/visualstudio> (дата обращения 23.10.2022)
4. Wikipedia C#, объяснение принципов разработки на языке программирования C#: [Режим доступа]. - <https://en.wikipedia.org/wiki/C#/help> (дата обращения 15.10.2022)
5. Wikipedia SQL Server, помощь и объяснение с примерами работы с базами данных с помощью Microsoft SQL Server: [Режим доступа]. <https://en.wikipedia.org/wiki/SQLServer> (дата обращения 28.10.2022)