Механизмы валидации. Часть 1.

Реализуем добавление записи в таблицу с пользователями. При добавлении, сделаем так, чтобы нельзя было ввести «запрещенные» значения.

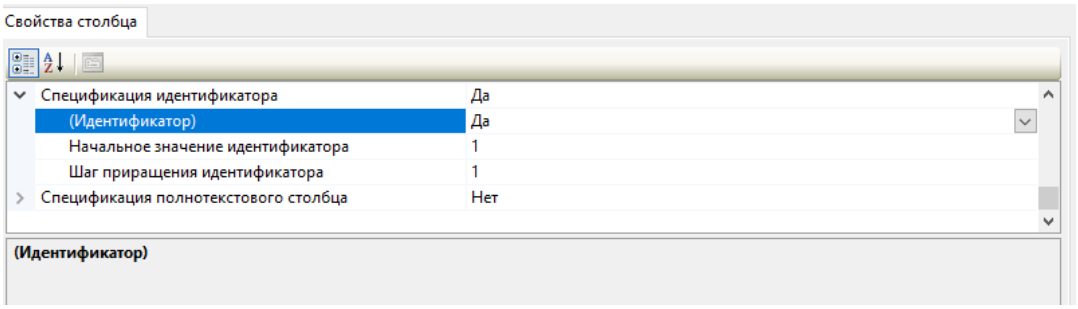
База данных

База данных имеет название ValidationUser. В базу добавим одну таблицу User:

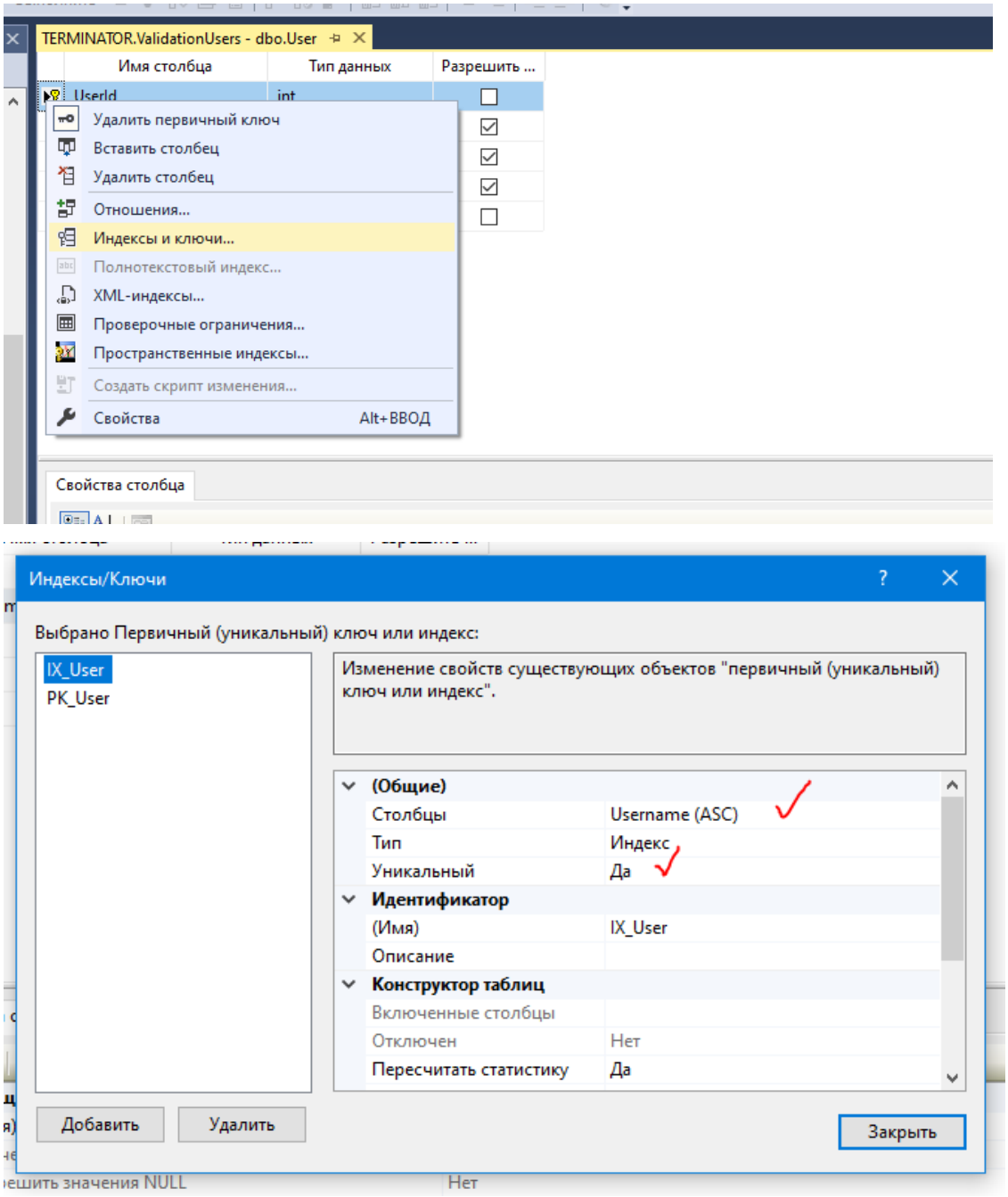
Имя столбца	Тип данных	Разрешить ...
UserId	int	☐
Username	nvarchar(40)	☑
Name	nvarchar(40)	☑
Age	int	☑
		☐

Здесь:

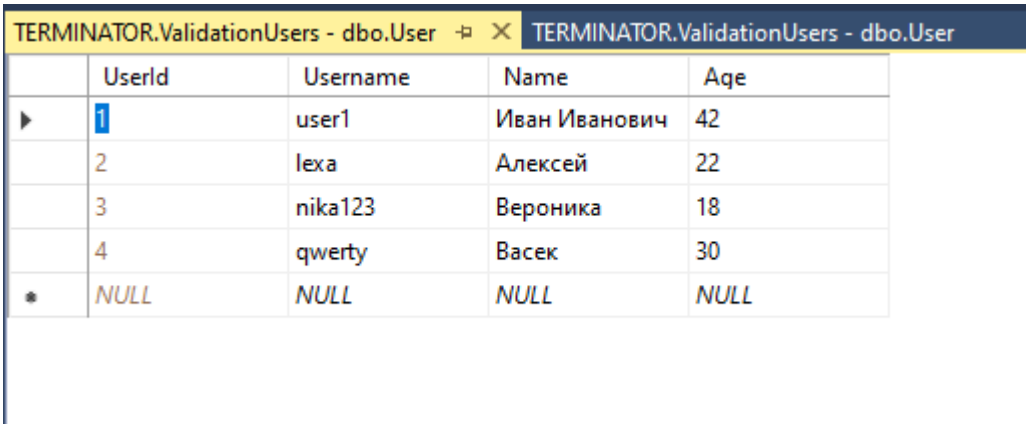
- UserId – первичный ключ. Следует задать автоинкримент:



- Добавим индекс (сделаем значения Username уникальными):

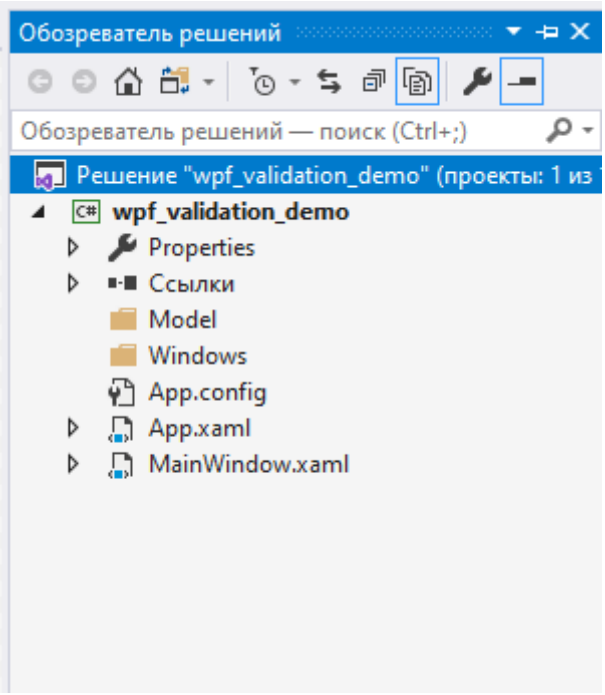


Заполним таблицу произвольными значениями:

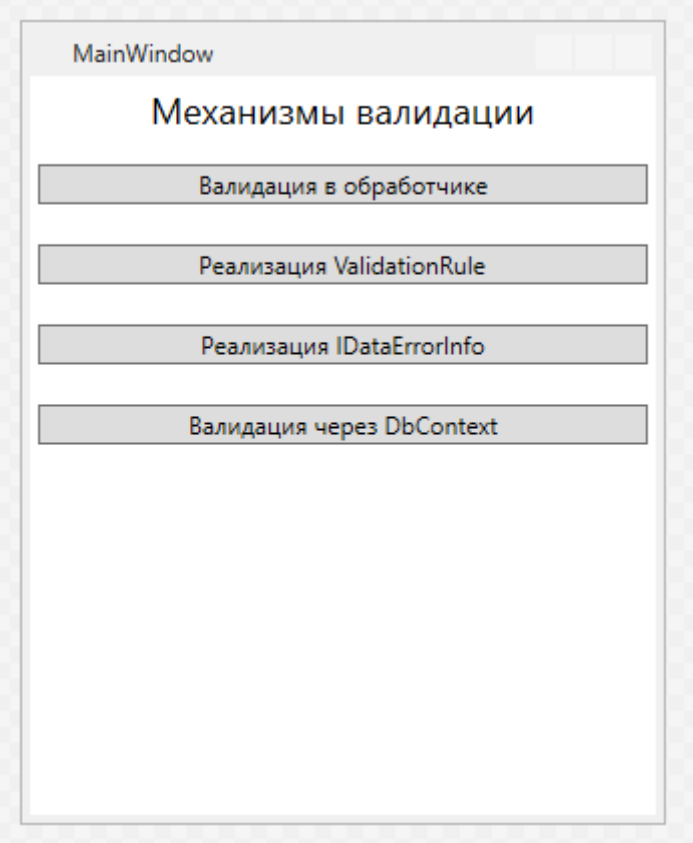


Проект WPF

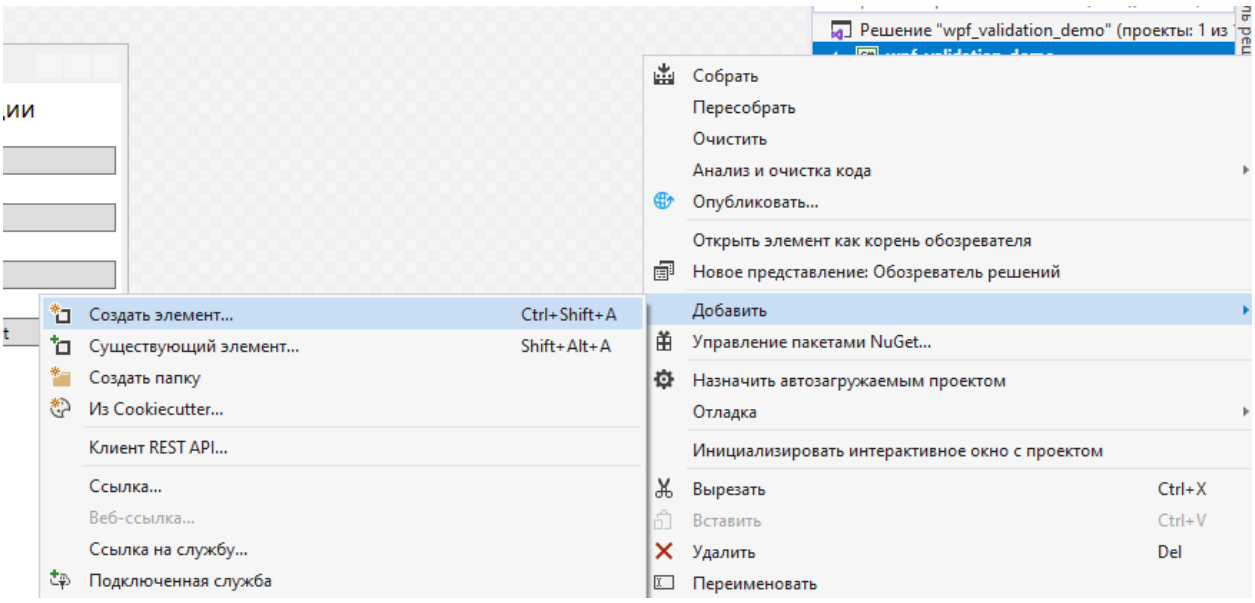
Создадим проект WPF. Добавим папки Windows и Model.

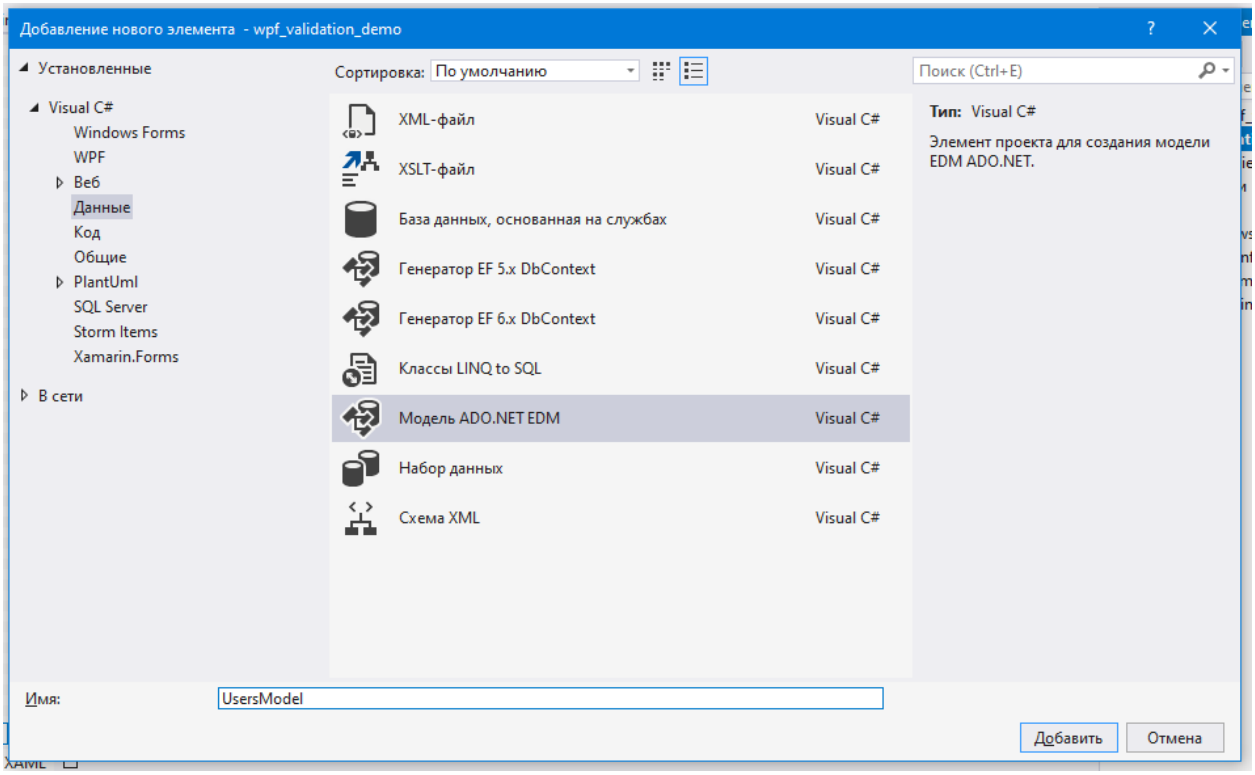


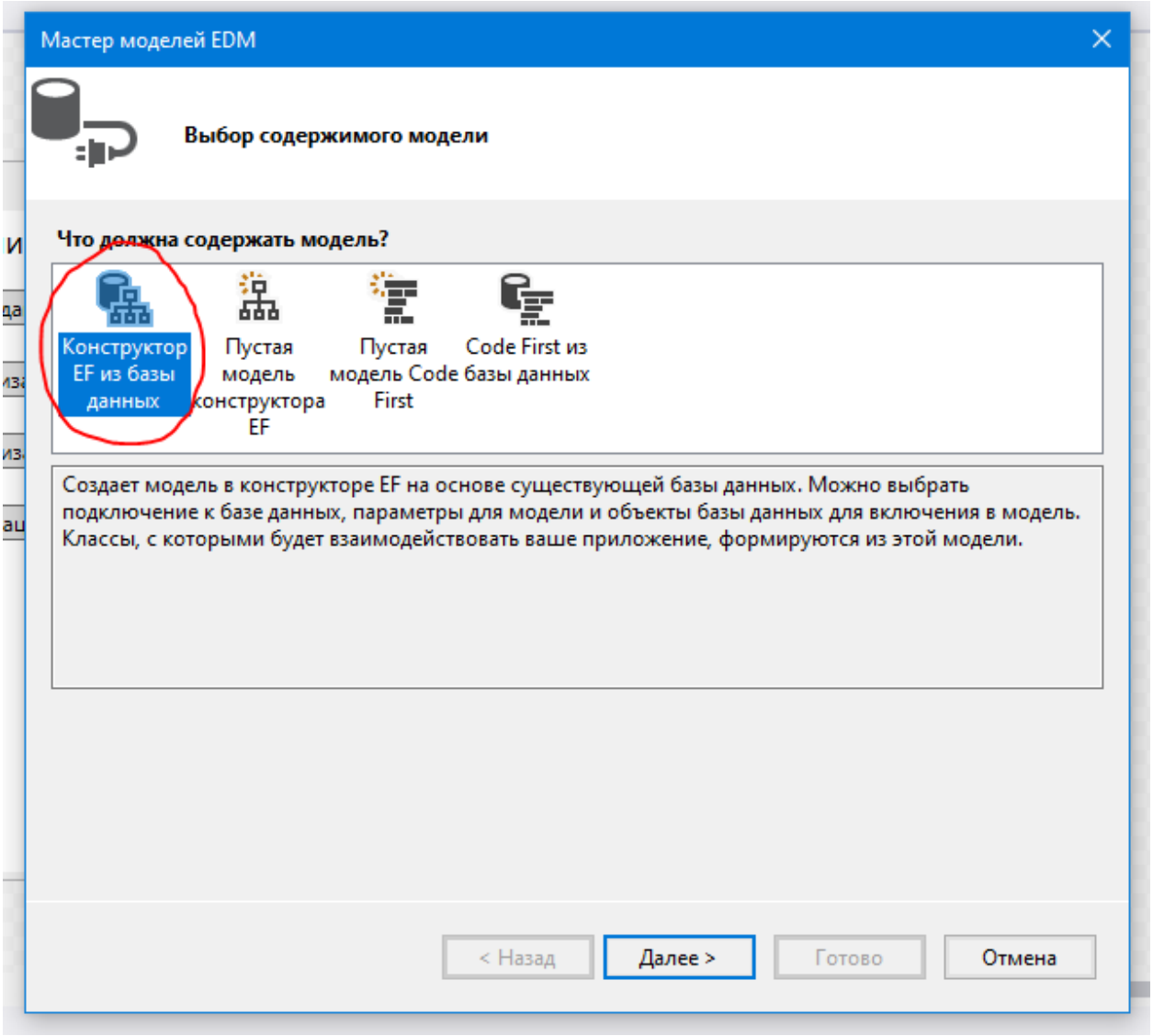
Главное окно:



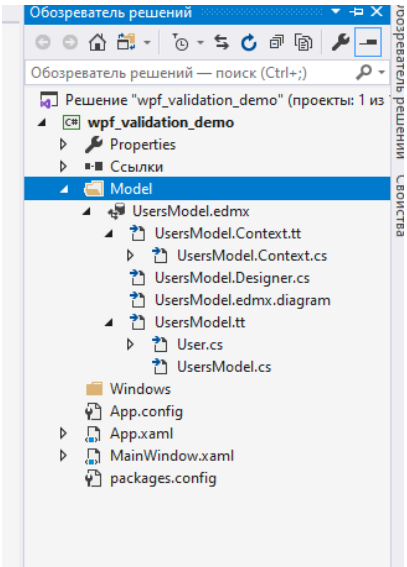
Используем Database First:


Решение "wpf_validation_demo" (проекты: 1 из
Собрать
Пересобрать
Очистить
Анализ и очистка кода
Опубликовать...
Открыть элемент как корень обозревателя
Новое представление: Обозреватель решений
Создать элемент... Ctrl+Shift+A
Существующий элемент... Shift+Alt+A
Создать папку
Из Cookiecutter...
Клиент REST API...
Ссылка...
Веб-ссылка...
Ссылка на службу...
Подключенная служба
Добавить
Управление пакетами NuGet...
Назначить автозагружаемым проектом
Отладка
Инициализировать интерактивное окно с проектом
Вырезать Ctrl+X
Вставить Ctrl+V
Удалить Del
Переименовать


Добавление нового элемента - wpf_validation_demo
Установленные
Visual C#
Windows Forms
WPF
Веб
Данные
Код
Общие
PlantUml
SQL Server
Storm Items
Xamarin.Forms
В сети
Сортировка: По умолчанию
Поиск (Ctrl+E)
XML-файл Visual C#
XSLT-файл Visual C#
База данных, основанная на службах Visual C#
Генератор EF 5.x DbContext Visual C#
Генератор EF 6.x DbContext Visual C#
Классы LINQ to SQL Visual C#
Модель ADO.NET EDM Visual C#
Набор данных Visual C#
Схема XML Visual C#
Тип: Visual C#
Элемент проекта для создания модели EDM ADO.NET.
Имя: UsersModel
Добавить
Отмена



Далее, следуйте инструкциям мастера (в принципе, там будут знакомые вам окна). Создание модели студией может занять какое-то время (вплоть до нескольких минут).



Теперь создадим статический класс (можно создать файл в корне проекта). Будем его использовать для обращения к базе из всех точек программы.

```
namespace wpf_validation_demo
{
    using Model;

    public static class Connection
    {
        public static ValidationUsersEntities Database { get; set; }
    }
}
```

В конструкторе главного окна создадим необходимый экземпляр:

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        Connection.Database = new Model.ValidationUsersEntities();
    }
}
```

Валидация в обработчике

Здесь, реализуем валидацию «по старинке» без использования привязки и каких-либо специальных возможностей WPF. Используйте данный вариант только при дефиците времени, отсутствия возможности поиска информации или для очень маленьких проектов.

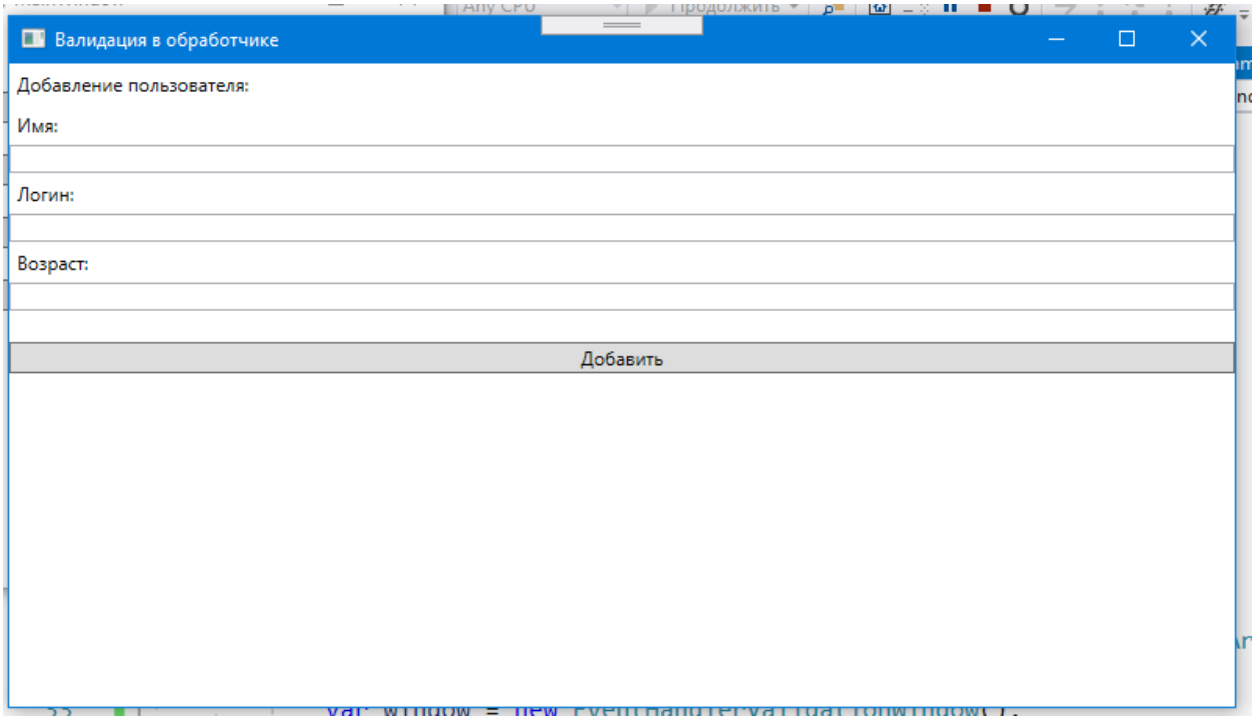
Создайте окно (у меня оно называется EventHandlerValidationWindow) и реализуйте переход на него:

```
private void GoToSimpleValidationWindow(object sender, RoutedEventArgs e) =>
    new EventHandlerValidationWindow().ShowDialog();
```

«Расписанный» вариант (используйте вариант выше, если данный пункт не вызывает вопросов):

```
private void GoToSimpleValidationWindow(object sender, RoutedEventArgs e)
{
    var window = new EventHandlerValidationWindow();
    window.ShowDialog();
}
```

Примерный вид окна (некоторый минимум, вы можете его улучшить):



Для кнопки «добавить» опишем обработчик:

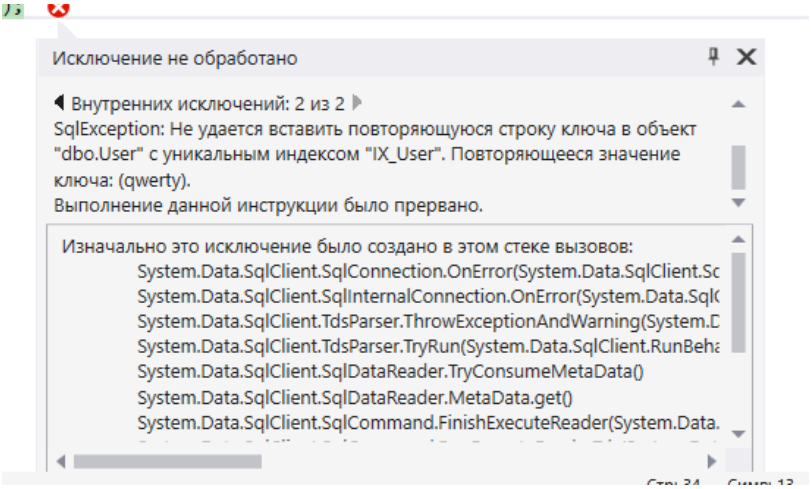
```
<Button Content="Добавить" Name="AddUserButton" Click="AddUserButton_Click" />
```

```
private void AddUserButton_Click(object sender, RoutedEventArgs e)
{
    // stackPanel - имя основной StackPanel
    if (stackPanel.Children.OfType<TextBox>().Any(tb => tb.Text.Trim() == ""))
    {
        MessageBox.Show("Все поля должны быть заполнены!");
        return;
    }

    int age = -1;

    if (!int.TryParse(AgeTextBox.Text, out age))
    {
        MessageBox.Show("Возраст должен быть целым числом!");
        return;
    }
    else if (age < 0 || age > 100)
    {
        MessageBox.Show("Возраст должен быть от 0 до 100!");
        return;
    }

    var newUser = new User { Age = age, Name = NameTextBox.Text, Username = UsernameTextBox.Text };
    Connection.Database.User.Add(newUser);
    Connection.Database.SaveChanges();
```

Если непонятно первое условие, где используется LINQ, то можете заменить его на три отдельных проверки каждого из полей ввода (три if или использовать && / ||).

Проверьте работоспособность.

Все работает, но имеется следующая проблема: мы нигде не учитываем тот факт, что значения в Username должны быть уникальными. Если попробовать добавить запись с уже используемым Username, то мы получим:



Эту проблему можно решить двумя способами.

Первый способ – выполнять проверку на наличие значения в БД:

```
if (Connection.Database.User.FirstOrDefault(u => u.Username == UsernameTextBox.Text.Trim()) != null)
{
    MessageBox.Show("Пользователь с таким именем уже существует!");
    return;
}
```

FirstOrDefault возвращает первый найденный элемент коллекции, удовлетворяющий условию в скобках, или значение по умолчанию (null), если элемент не найден. Условие в скобках – совпадение Username с введенным в TextBox значением.

Или через цикл (не рекомендуется):

```
bool hasUser = false;

foreach (User user in Connection.Database.User.AsEnumerable())
{
    if (user.Username == UsernameTextBox.Text.Trim())
    {
        hasUser = true;
        break;
    }
}

if (hasUser)
{
    MessageBox.Show("Пользователь с таким именем уже существует!");
    return;
}
```

Второй способ – отлавливать исключение:

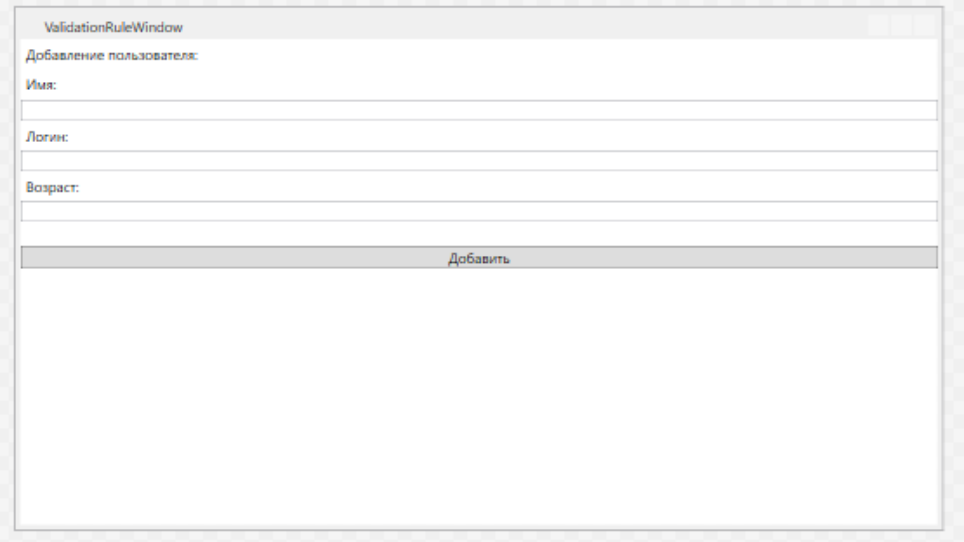
```
Connection.Database.User.Add(newUser);

try
{
    Connection.Database.SaveChanges();
}
catch (Exception error)
{
    MessageBox.Show("Ошибка при добавлении: " + error.Message);
    Connection.Database.User.Remove(newUser);
}
```

Попробуйте оба варианта и выберите какой-то один. Что нехорошо в коде, который отлавливает исключение? Можно ли улучшить решение?

Закончите разработку данного окна и еще раз проверьте работоспособность.

Валидация с помощью ValidationRule

Добавим еще одно окно с аналогичным внешним видом:



Добавим публичное свойство и используем его как DataContext:

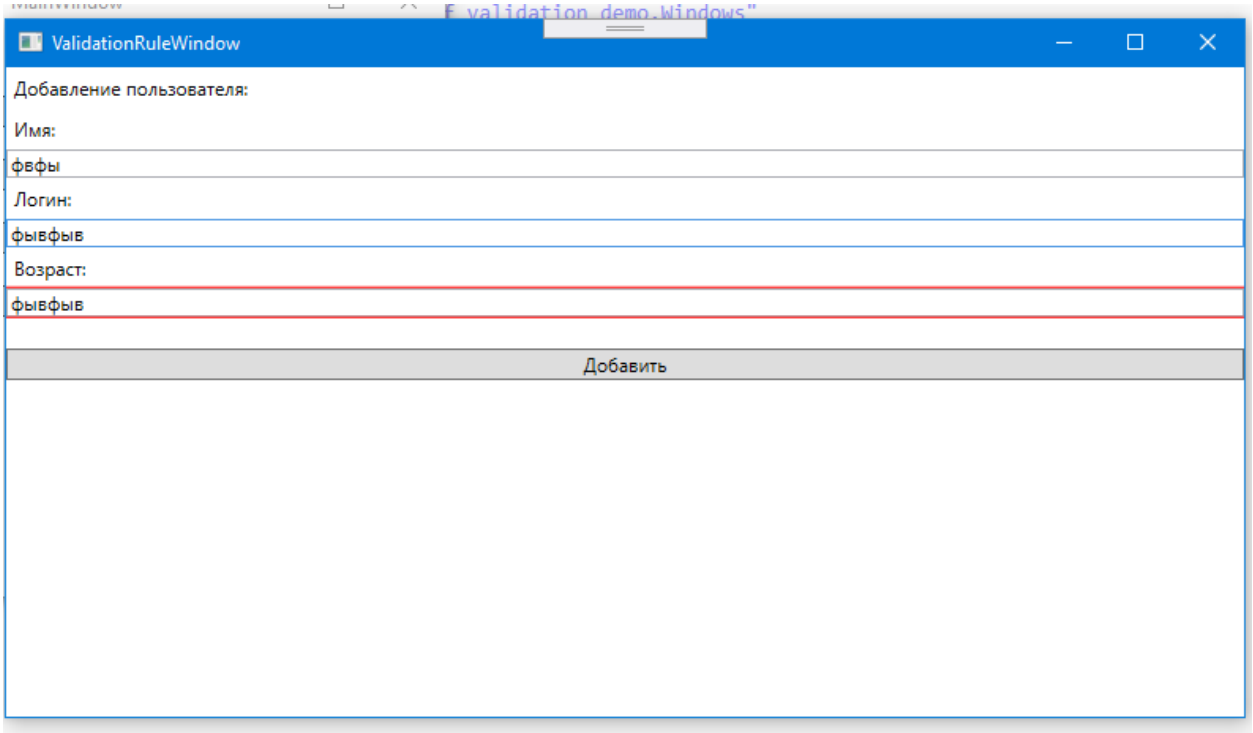
```
public partial class ValidationRuleWindow : Window
{
    public User newUser { get; set; }

    public ValidationRuleWindow()
    {
        InitializeComponent();
        DataContext = newUser = new User();
    }
}
```

Далее, выполним привязку:

```
<StackPanel Name="stackPanel">
        <Label Content="Добавление пользователя: " />
```

```
        <Label Content="Имя: " />
        <TextBox Name="NameTextBox" Text="{Binding Name}" />
        <Label Content="Логин: " />
        <TextBox Name="UsernameTextBox" Text="{Binding Username}" />
        <Label Content="Возраст: " />
        <TextBox Name="AgeTextBox" Text="{Binding Age}" />
        <Label Height="20" />
        <Button Content="Добавить" Name="AddUserButton" />
    </StackPanel>
```

После чего попробуйте запустить программу и вызвать данное окно. Если ввести в поле возраст нечисловое значение, то мы увидим, что поле ввода будет выделено красным цветом (отработает после потери фокуса элементом).



Почему это работает? Дело в том, что по умолчанию привязка предполагает правило, согласно которому значения, выбрасывающие исключения, являются невалидными. В данном случае невозможно преобразовать текстовую строку в int, что и приводит к возникновению исключения.

См. начало статьи https://metanit.com/sharp/wpf/14.php

Сделаем кнопку «Добавить» недоступной, если есть какие-либо ошибки. Для этого настроим привязку:

```
<TextBox Name="AgeTextBox"
         Text="{Binding Age, NotifyOnValidationError=True}"
         Validation.Error="TextBox_Error" />
```

NotifyOnValidationError говорит о необходимости генерации события Error при ошибочном значении.

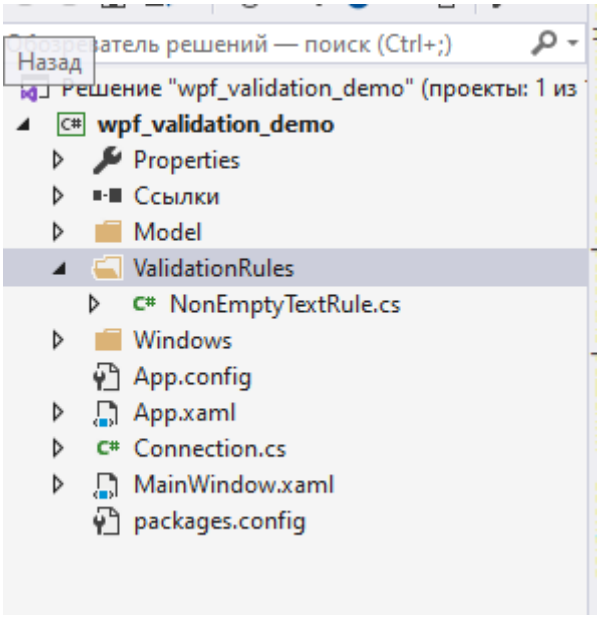
Validation.Error – обработчик события, возникающего при ошибке. Код данного обработчика:

```
private void TextBox_Error(object sender, ValidationErrorEventArgs e)
{
    if (e.Action == ValidationErrorEventAction.Added)
    {
        AddUserButton.IsEnabled = false;
    }

    if (e.Action == ValidationErrorEventAction.Removed)
    {
        AddUserButton.IsEnabled = true;
    }
}
```

Проверьте, что после возникновения ошибки кнопка становится недоступной. Поскольку мы будем использовать несколько правил, и у нас может быть несколько ошибок, изменим код таким образом, чтобы делать кнопку доступной исходя из количества ошибок:

```
private int errorCount = 0;


private void TextBox_Error(object sender, ValidationErrorEventArgs e)
{
    if (e.Action == ValidationErrorEventAction.Added)
    {
        errorCount++;
    }

    if (e.Action == ValidationErrorEventAction.Removed)
    {
        errorCount--;
    }

    AddUserButton.IsEnabled = errorCount == 0;
}
```

Теперь, добавим свои правила. Для этого нужно унаследовать класс от ValidationRule и реализовать метод Validate.




```
namespace wpf_validation_demo.ValidationRules
{
    class NonEmptyTextRule : ValidationRule
    {
        public override ValidationResult Validate(object value, CultureInfo cultureInfo)
        {
            throw new NotImplementedException();
        }
    }
}
```

Значение для проверки – value. Реализуем логику:

```
class NonEmptyTextRule : ValidationRule
{
    public override ValidationResult Validate(object value, CultureInfo cultureInfo)
    {
        string text = value as string;

        if (text != null)
        {
            if (text.Trim() != "")
            {
                return ValidationResult.ValidResult;
            }
        }

        return new ValidationResult(false, "Значение не может быть пустым или состоять из пробелов!");

    }
}
```

Добавим данное правило для имени

Namespace:

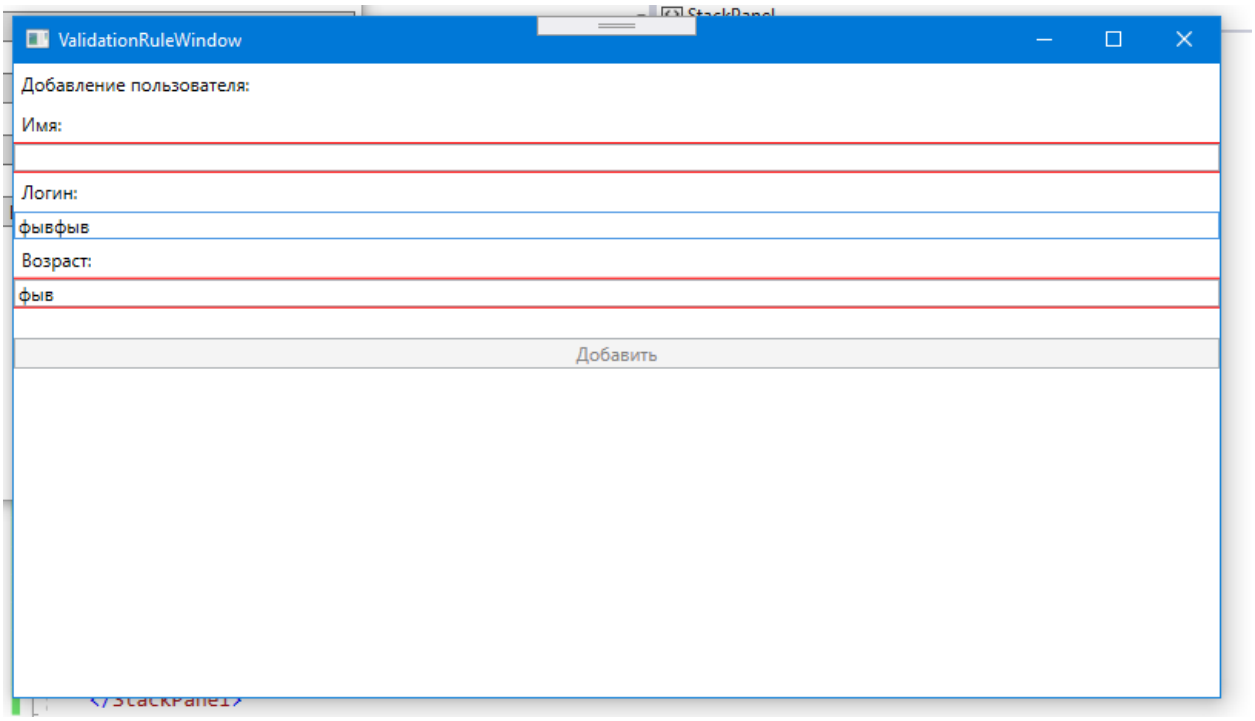
```
xmlns:rules="clr-namespace:wpf_validation_demo.ValidationRules"
```

Binding (распишем свойство Text, чтобы не делать код слишком сложным и нечитаемым из-за вложенных скобок):

```
<TextBox Name="NameTextBox">
            <TextBox.Text>
                <Binding Path="Name" NotifyOnValidationError="True">
```

```
                <Binding.ValidationRules>
                    <rules:NonEmptyTextRule />
                </Binding.ValidationRules>
            </Binding>
        </TextBox.Text>
    </TextBox>
```

Проверим работоспособность:



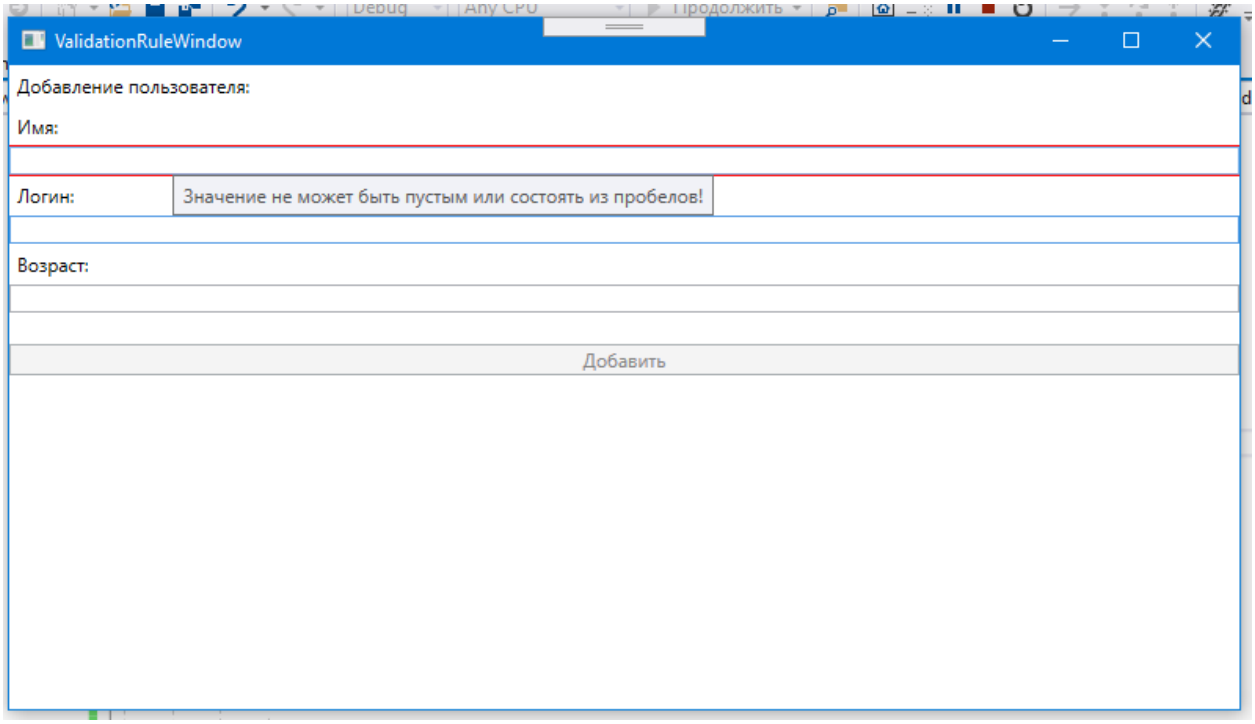
Добавим обработчик (используем ранее написанный метод):

```
<TextBox Name="NameTextBox" Validation.Error="TextBox_Error">
```

Немного изменим код, добавив создание ToolTip:

```
private void TextBox_Error(object sender, ValidationErrorEventArgs e)
{
    if (e.Action == ValidationErrorEventAction.Added)
    {
        errorCount++;

        var errorToolTip = new ToolTip();
        errorToolTip.Content = e.Error.ErrorContent;
        (sender as TextBox).ToolTip = errorToolTip;

    }

    if (e.Action == ValidationErrorEventAction.Removed)
    {
        errorCount--;
    }

    AddUserButton.IsEnabled = errorCount == 0;
}
```

Результат:



Чтобы проверка изменялась не при потере фокуса, а при других условиях, можно использовать UpdateSourceTrigger:

```
<TextBox Name="NameTextBox" Validation.Error="TextBox_Error">
            <TextBox.Text>
                <Binding Path="Name" UpdateSourceTrigger="PropertyChanged"
                         NotifyOnValidationError="True">
                    <Binding.ValidationRules>
                        <rules:NonEmptyTextRule />
                    </Binding.ValidationRules>
                </Binding>
            </TextBox.Text>
        </TextBox>
```

Самостоятельная работа.

Добавьте данное правило (NonEmptyTextRule) для остальных TextBox.

Создайте еще одно правило AgeValidationRule, где будет проверено, что:

- строка является числом (используйте TryParse);
- число находится в пределах от 0 до 100.

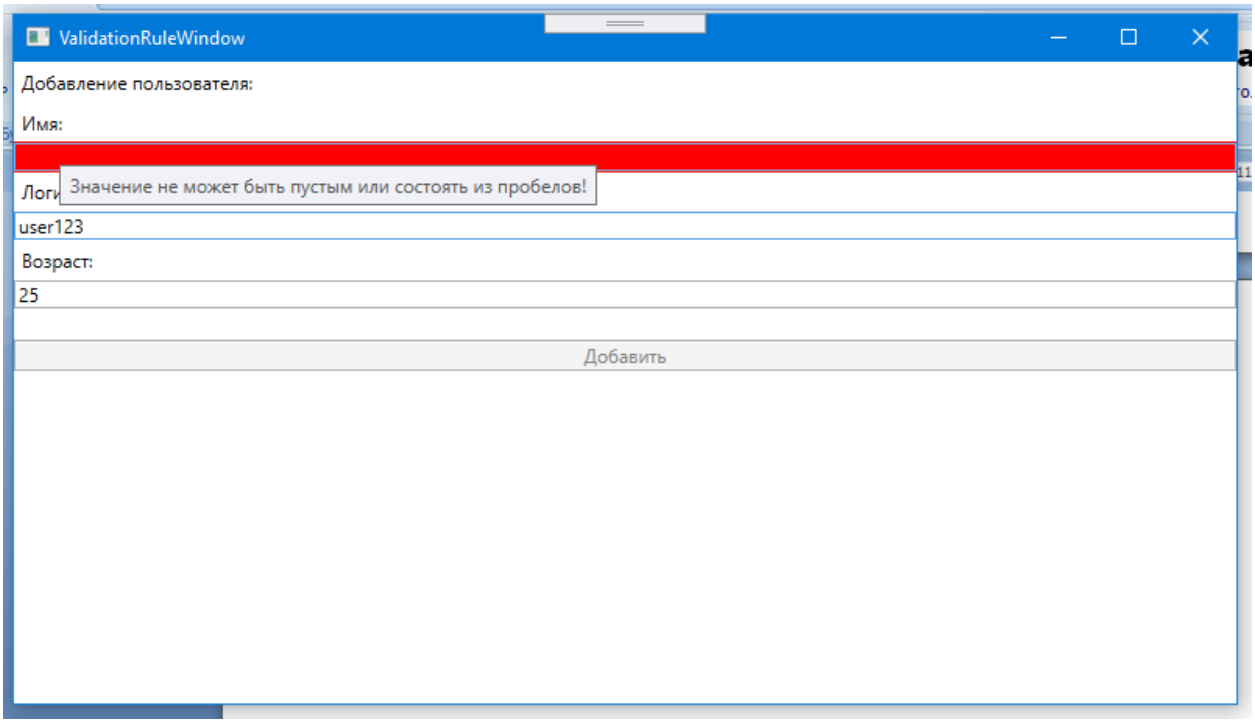
Добавьте данное правило для AgeTextBox (TextBox, где вводится возраст).

Дополнительно

Используйте настройку внешнего вида при возникновении ошибок:

```
    <Window.Resources>
        <Style TargetType="TextBox">
            <Style.Triggers>
                <Trigger Property="Validation.HasError" Value="true">
                    <Setter Property="Background" Value="Red" />
                </Trigger>
            </Style.Triggers>
        </Style>
```

```
    </Window.Resources>
```



После всех необходимых действии, наконец, опишем добавление пользователя:

```
private void AddUserButton_Click(object sender, RoutedEventArgs e)
{
    // следует самостоятельно добавить проверку на наличие username
    // и блок try на тот случай, если возникнет какая-нибудь ошибка
    Connection.Database.User.Add(newUser);
    Connection.Database.SaveChanges();
}
```