**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КОМИ**

**Государственное профессиональное образовательное учреждение**

**«Воркутинский арктический горно-политехнический колледж»**

**КУРСОВОЙ ПРОЕКТ**

По дисциплине МДК.05.02 Разработка кода информационных систем

**Разработка информационной системы «Страховая фирма»**

Выполнил студент гр. ИСП-21 /\_\_\_\_\_\_\_\_/Ильин Владислав Александрович/

(подпись) (Ф.И.О.)

**ОЦЕНКА:\_\_\_\_\_\_\_\_\_\_\_\_\_**

Дата:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ПРОВЕРИЛ**

Научный руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_/ Егоров Данил Павлович/

(подпись) (Ф.И.О.)

Воркута

2024

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 3](#_Toc166054428)

[**ГЛАВА 1. ВЫБОР ИНСТРУМЕНТАРИЯ** 4](#_Toc166054429)

[**1.1** **Платформа .NET** 4](#_Toc166054430)

[**1.2** **Язык программирования С#** 4](#_Toc166054431)

[**1.3** **Windows Presentation Foundation (WPF)** 5](#_Toc166054432)

[**1.4** **СУБД SQL server** 6](#_Toc166054433)

[**1.5** **Microsoft SQL Server Management Studio** 7](#_Toc166054434)

[**1.6** **Entity Framework** 8](#_Toc166054435)

[**ГЛАВА 2. ПРОЕТИРОВАНИЕ БАЗЫ ДАННЫХ** 10](#_Toc166054436)

[**2.1 Разработка диаграммы ERD** 10](#_Toc166054437)

[**2.2 Разработка базы данных** 12](#_Toc166054438)

[**ГЛАВА 3. РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ** 12](#_Toc166054439)

[**3.1 Разработка прототипа информационной системы** 12](#_Toc166054440)

[**3.2 Программирование информационной системы** 14](#_Toc166054441)

[**3.2.1 Разработка модуля «Авторизация»** 14](#_Toc166054442)

[**3.2.2 Разработка модуля CAPTCHA** 17](#_Toc166054443)

[**3.2.3 Разработка модуля Создание заявки** 18](#_Toc166054444)

[**3.2.4 Разработка модуля Рассмотрение заявки** 21](#_Toc166054445)

[**3.2.5 Разработка модуля «Отчет»** 32](#_Toc166054446)

[**ЗАКЛЮЧЕНИЕ** 34](#_Toc166054447)

[**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ** 35](#_Toc166054448)

# **ВВЕДЕНИЕ**

В настоящее время разработка информационных систем является актуальной задачей для многих компаний, включая страховые фирмы. Информационная система "Страховая фирма" представляет собой комплекс программных и аппаратных средств, позволяющих управлять всеми процессами, связанными со страхованием, эффективно обрабатывать информацию и проводить анализ данных.

Предмет – информационная система «Страховая фирма»

Объект – разработка информационной системы «Страховая фирма»

Цель данного курсового проекта состоит в разработке информационной системы, которая позволит упростить и автоматизировать процессы страхования, сократить временные затраты на обработку информации, улучшить качество обслуживания клиентов и повысить общую эффективность бизнеса страховой компании.

Задачи:

* анализ деятельности страховых фирм;
* проектирование информационной системы;
* разработка информационной системы.

В ходе работы над проектом будут рассмотрены основные требования к информационной системе для страховой фирмы, разработаны алгоритмы работы системы, спроектирована база данных, создан пользовательский интерфейс и проведено тестирование функциональности.

Результаты данного курсового проекта могут быть использованы страховыми компаниями для улучшения своей деятельности, повышения конкурентоспособности на рынке и удовлетворения потребностей клиентов.

# **ГЛАВА 1. ВЫБОР ИНСТРУМЕНТАРИЯ**

# **Платформа .NET**

.NET – это платформа, разработанная компанией Microsoft, которая предоставляет среду для создания и выполнения программного обеспечения. Она включает в себя набор инструментов, библиотек и технологий, которые упрощают разработку приложений для Windows, веб-приложений, мобильных приложений и игр.

Основными компонентами платформы .NET являются следующие:

Common Language Runtime (CLR) - виртуальная машина, которая управляет выполнением программ на различных языках, таких как C#, F# и Visual Basic. CLR обеспечивает управление памятью, отслеживание ошибок, управление потоками и другие сервисы.

Framework Class Library (FCL) - набор библиотек, предоставляющих готовые классы и методы для решения распространенных задач разработки программного обеспечения. FCL включает в себя классы для работы с файлами, сетью, графикой, базами данных и многими другими.

Visual Studio - интегрированная среда разработки, которая предоставляет инструменты для создания, отладки и развертывания приложений на платформе .NET. Visual Studio поддерживает различные языки программирования и предлагает широкий выбор расширений и инструментов для повышения производительности разработчиков.

Платформа .NET позволяет создавать мощные и производительные приложения для различных платформ, обеспечивая удобство разработки и масштабируемость проектов. Она также активно поддерживается сообществом разработчиков и имеет множество ресурсов и документации для обучения и совершенствования навыков в области разработки на .NET.

# **Язык программирования С#**

C# (C-Sharp) - это современный объектно-ориентированный язык программирования, разработанный компанией Microsoft в начале 2000-х годов. Он является одним из ключевых языков для разработки приложений под платформу .NET.

Вот некоторые основные особенности C#:

Синтаксис: Синтаксис C# схож с синтаксисом языка программирования Java. Он имеет четкую структуру, позволяющую удобно организовывать код.

Объектно-ориентированный: C# поддерживает принципы объектно-ориентированного программирования, такие как наследование, инкапсуляция и полиморфизм.

Платформонезависимость: с помощью платформы .NET, приложения, написанные на C#, могут быть запущены на различных операционных системах, таких как Windows, Linux и macOS.

Мощные инструменты: C# предоставляет разработчикам доступ к широкому спектру библиотек и инструментов, упрощающих разработку приложений.

Поддержка многопоточности: C# предоставляет удобные средства для работы с многопоточностью, что позволяет создавать высокопроизводительные и эффективные приложения.

Типобезопасность: C# - язык со строгой типизацией, что помогает выявлять ошибки и повышает надежность кода.

Поддержка LINQ: C# поддерживает Language Integrated Query (LINQ), что упрощает работу с коллекциями данных и реляционными базами данных.

C# широко используется для разработки различных типов приложений, включая веб-приложения, мобильные приложения, игры, облачные приложения и многое другое. Он является популярным выбором среди разработчиков благодаря своей эффективности, удобству и мощным возможностям.

# **Windows Presentation Foundation (WPF)**

Windows Presentation Foundation (WPF) - это технология, используемая для создания графических пользовательских интерфейсов (GUI) в приложениях на платформе Windows. WPF была выпущена компанией Microsoft в 2006 году и является частью платформы .NET Framework.

WPF предоставляет разработчикам широкий набор инструментов для создания современных и интерактивных пользовательских интерфейсов, включая возможность использования векторной графики, анимации, стилей и шаблонов. WPF также обладает расширенной системой привязки данных и поддерживает различные способы компоновки элементов интерфейса.

Одним из ключевых преимуществ WPF является его разделение логики приложения и внешнего вида, что позволяет разработчикам создавать более гибкие и масштабируемые приложения. WPF также обеспечивает возможность создания кросс-платформенных приложений, которые могут работать как на ПК под управлением Windows, так и на мобильных устройствах.

В целом, WPF предоставляет разработчикам мощный набор инструментов для создания современных и инновационных пользовательских интерфейсов, что делает его популярным выбором для создания приложений на платформе Windows.

# **СУБД SQL server**

Microsoft SQL Server - это реляционная система управления базами данных, разработанная компанией Microsoft. Она предназначена для хранения и управления данными в корпоративных и различных приложениях.

SQL Server поддерживает SQL (Structured Query Language), который используется для работы с базами данных. Она имеет множество функций, включая хранение данных, обработку транзакций, анализ данных, репликацию и многое другое.

SQL Server имеет несколько версий, включая Express, Developer, Standard и Enterprise. Каждая из них имеет различные возможности и ограничения, а также разные цены.

SQL Server также имеет ряд дополнительных инструментов и служб, таких как SQL Server Management Studio (SSMS), который используется для управления базами данных, и SQL Server Reporting Services (SSRS), который используется для создания и публикации отчетов.

SQL Server также интегрируется с другими продуктами Microsoft, такими как Azure Cloud, SharePoint и Excel, что облегчает работу с данными в различных средах.

В общем, Microsoft SQL Server является одной из самых популярных и мощных систем управления базами данных на сегодняшний день, и она широко используется компаниями по всему миру для хранения и управления своими данными.

# **Microsoft SQL Server Management Studio**

Microsoft SQL Server Management Studio (SSMS) - это интегрированная среда для управления и администрирования баз данных Microsoft SQL Server. SSMS предоставляет широкий набор инструментов и возможностей для разработчиков и администраторов баз данных.

SSMS позволяет выполнить множество операций с базами данных, такие как создание и удаление баз данных, создание и изменение таблиц, хранимых процедур, триггеров и многое другое. Также SSMS предоставляет возможность выполнения SQL-запросов, выполнения резервного копирования и восстановления баз данных, настройки безопасности и мониторинга производительности.

SSMS имеет удобный и интуитивно понятный пользовательский интерфейс, который значительно упрощает работу с базами данных. Он поддерживает различные версии Microsoft SQL Server, начиная с SQL Server 2005.

Кроме того, SSMS предоставляет возможность настройки и расширения функциональности с помощью дополнений и плагинов. Также с помощью SSMS можно управлять несколькими серверами баз данных одновременно, что способствует повышению производительности и эффективности работы.

В целом, Microsoft SQL Server Management Studio является мощным инструментом для работы с базами данных Microsoft SQL Server и предоставляет широкий спектр возможностей для разработчиков и администраторов.

# **Entity Framework**

Entity Framework (EF) - это набор инструментов от Microsoft для работы с данными в приложениях на платформе .NET. EF позволяет разработчикам работать с данными в виде объектов и классов, а не с прямыми запросами к базе данных. Он позволяет создавать модели данных, которые отображаются на структуру базы данных, и автоматически генерировать SQL-запросы для выполнения операций с данными.

Основные компоненты Entity Framework:

DbContext: это основной класс, который представляет контекст данных и позволяет взаимодействовать с базой данных. Он содержит методы для выполнения операций, таких как добавление, удаление и обновление данных.

Entity Data Model: это модель данных, которая описывает структуру базы данных и ее отображение на объекты в коде. Entity Data Model обычно создается с помощью Entity Framework Designer или Code First подхода.

LINQ to Entities: это часть Entity Framework, которая позволяет писать запросы к данным с использованием LINQ (Language Integrated Query). LINQ to Entities позволяет выразить запросы на языке C# или VB.NET и преобразовать их в SQL-запросы для выполнения на сервере базы данных.

Code First: это подход к созданию моделей данных в Entity Framework, при котором классы и свойства объектов представляют собой модель данных, а база данных создается автоматически на основе этих классов.

Entity Framework поддерживает различные версии базы данных, такие как SQL Server, MySQL и Oracle. Он предоставляет механизм отслеживания изменений, автоматическую генерацию команд SQL и возможность использования хранимых процедур.

Entity Framework удобен для работы с данными в приложениях на платформе .NET и позволяет упростить процесс работы с базой данных, уменьшить объем кода и повысить производительность приложения.

# **ГЛАВА 2. ПРОЕТИРОВАНИЕ БАЗЫ ДАННЫХ**

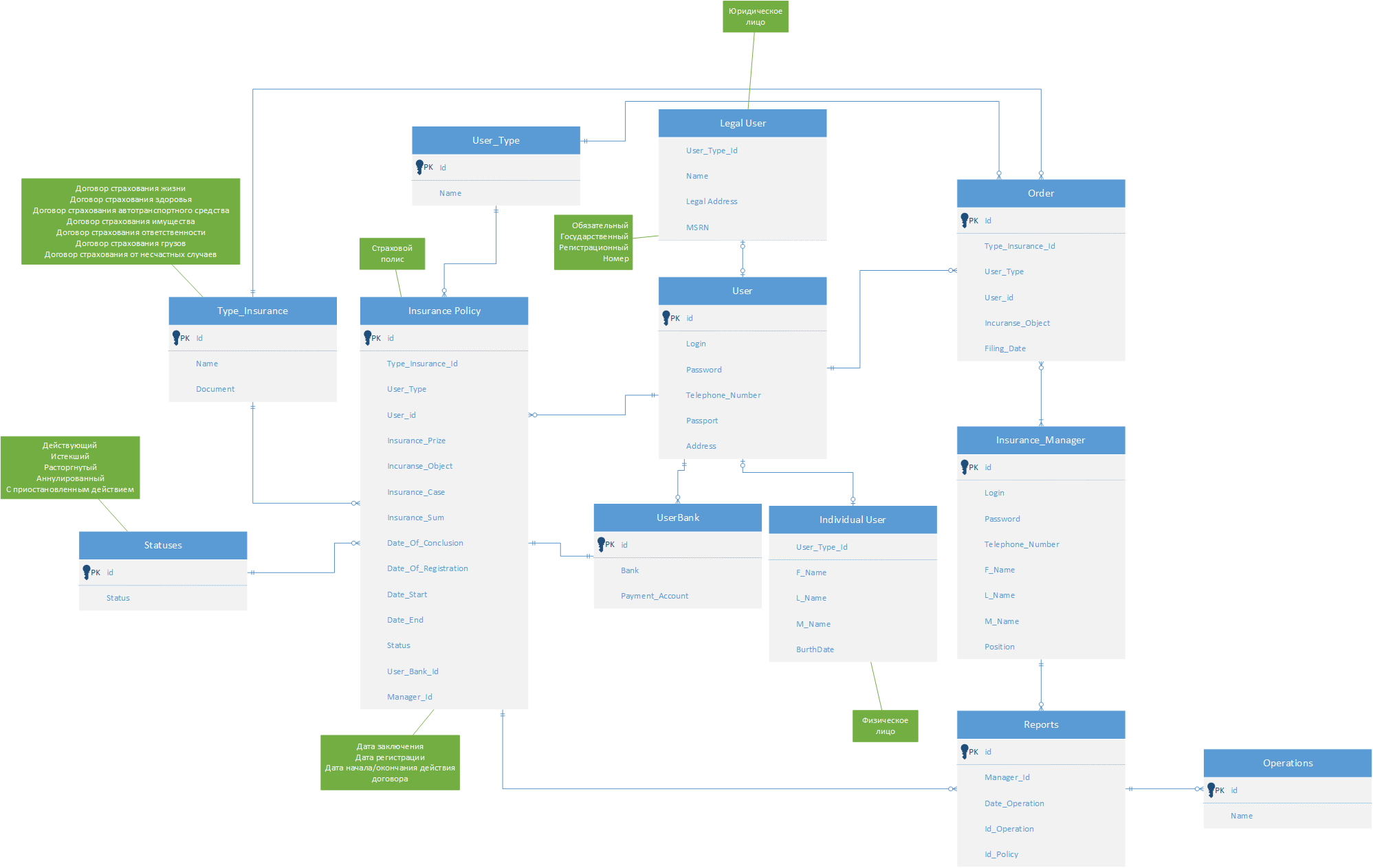
# **2.1 Разработка диаграммы ERD**

ER-диаграмма (сущность-связь диаграмма) - это схема сущностей и их взаимосвязей, которая используется для моделирования структуры данных в базах данных.

ER-диаграмма состоит из сущностей (Entity), атрибутов (Attribute) и связей (Relationship). Сущность представляет отдельный объект или сущность, который мы хотим хранить в базе данных. Атрибуты — это характеристики, которые описывают сущность. Связи определяют отношения между сущностями.

ER-диаграмма помогает понять структуру данных, определить ключевые сущности, их атрибуты и взаимосвязи между ними. Она помогает разработчикам и заказчикам лучше понять требования к базе данных и определить наиболее эффективную структуру для хранения данных.

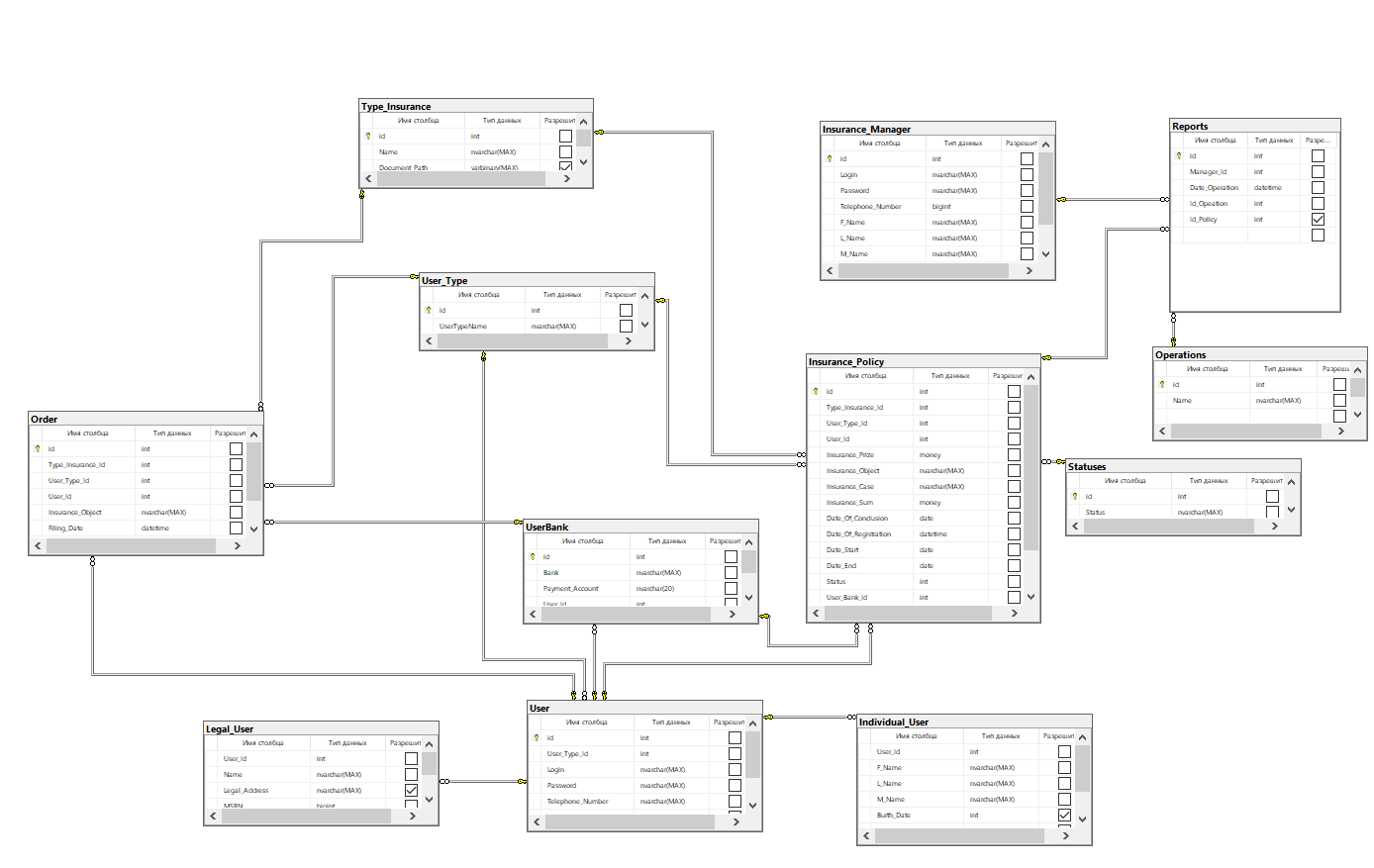
ER-диаграммы применяются на этапе проектирования базы данных, перед созданием самой базы данных. Они позволяют визуализировать структуру данных и облегчают коммуникацию между разработчиками и заказчиками.



*Рисунок 2.1*

На рисунке 2.1 представлена диаграмма сущности и связи в ИС «Страховая фирма» главными сущностями здесь являются User которая наследует Legal\_User и Individual\_User, Insurance\_Policy которая представляет собой страховой полис Order которая представляет собой заявку, которую формирует пользователь и которую затем обрабатывает менеджер.

# **2.2 Разработка базы данных**

База данных разработана на базе SQL Server Management Studio   


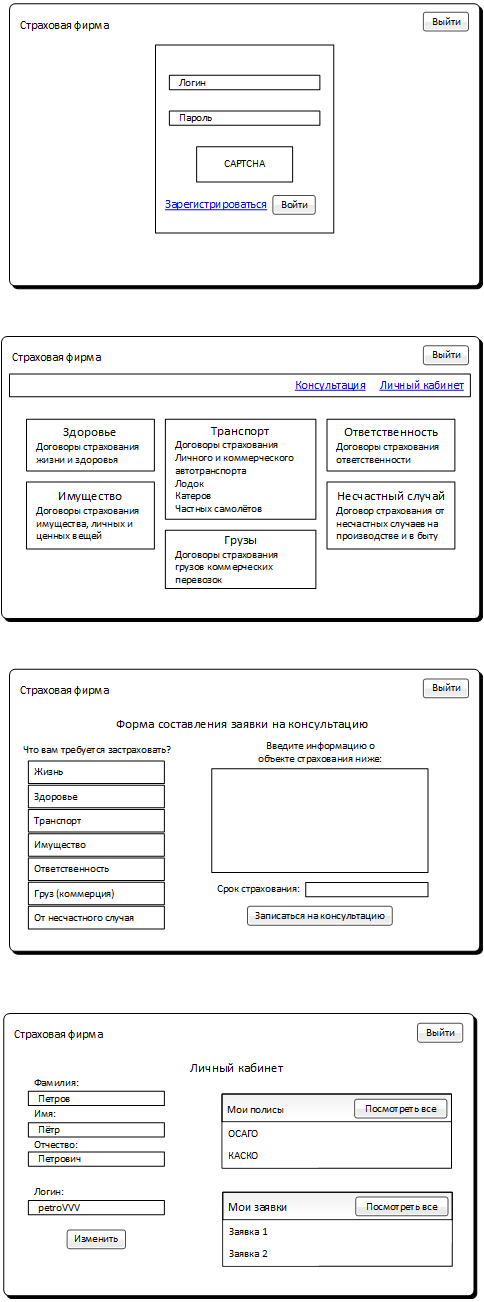
*Рисунок 2.2*

На рисунке 2.2 представлена база данные ИС «Страховая фирма»

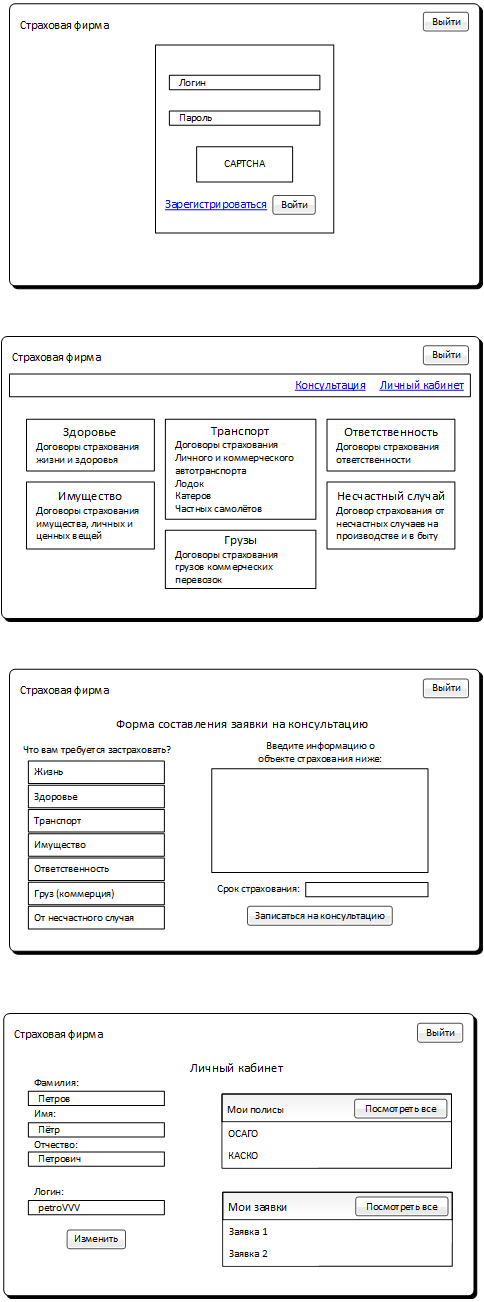
В базе данных 12 таблиц, а также применена методика формализации базы данных.

# **ГЛАВА 3. РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ**

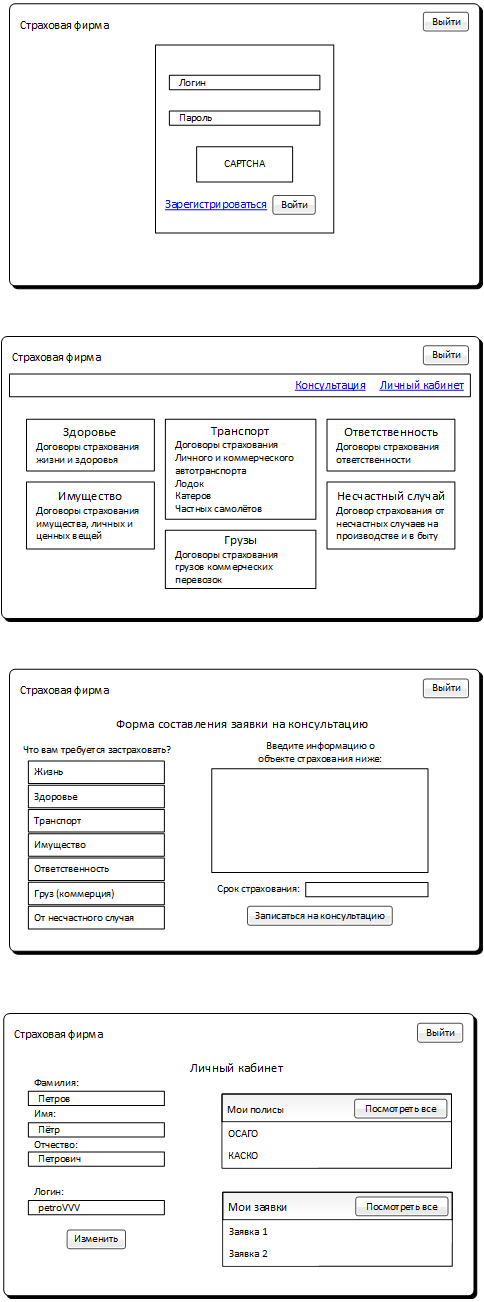
# **3.1 Разработка прототипа информационной системы**

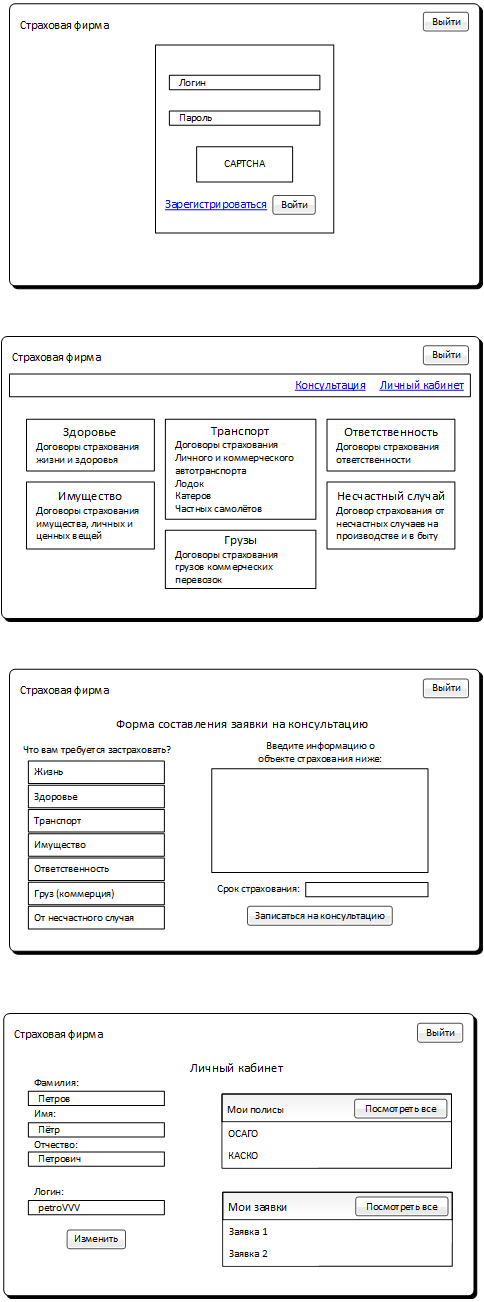
*Рисунок 3.1.1*

На рисунке 3.1.1 показано хменичное изображение формы авторизации. В ней есть два поля для ввода логина и пароля, экземпляр формы CAPTCHA кнопка Войти и гипперсылка для регистрации.

*Рисунок 3.1.**2*

На рисунке 3.1.2 сизображена главная страница прользователя. На ней расположено описание предоставляемых услуг, гипперссылка консультации и гипперссылка личного кабинета.

*Рисунок 3.1.3*

На рисунке 3.1.3 изображена страница личного кабинета пользователя. На ней расположены таблицы с полисами пользователя и его заявками.

*Рисунок 3.1.4*

На рисунке 3.1.4 изображена страница консультации где пользователь может выбрать тип страховки, ввести данные об объекте страхования, ввести данные своего банка и расчётного счёта

Прототип информационной системы был разработан при помощи проволочной диаграммы на платформе Visio. На диаграмме показано окно авторизации, главное окно пользователя, страница формирования заявки на консультацию к менеджеру, а также личный кабинет пользователя с показанными там заявками и активными полисами.

# **3.2 Программирование информационной системы**

# **3.2.1 Разработка модуля «Авторизация»**

Модуль авторизации, как и прочие был разработан при помощи модели обмена информацией с базой данных LINQ, что продемонстрировано ниже:

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace Insurance\_сompany

{

/// <summary>

/// Логика взаимодействия для LoginPage.xaml

/// </summary>

public partial class LoginPage : Page

{

private CAPTCHA captcha = new CAPTCHA(); // Инициализируем экземпляр класса капча

public static int user\_id = 0; // Создаём Публичную переменную с id пользователя

public static int manager\_id = 0;

private bool user\_is\_log\_in;

public LoginPage()

{

InitializeComponent();

CapOut.IsEnabled = false; // Делаем текстбокс не активным

captcha.CaptchaIsGenerate = false; // Мы не проходили капчу

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

/// Приравниваем публичные поля КапИн и КапАут и

/// нашим КапИн и КУапАут (текстбоксы вход и выход капчи)

using (SqlConnection connection = new SqlConnection(Manager.connectionString))

{

connection.Open(); // Открывает наш коннект

StringBuilder errors = new StringBuilder(); // Инициализируем экземпляр класса стрингБилдер

var converter = new System.Windows.Media.BrushConverter();

var lightGray = (Brush)converter.ConvertFromString("#FFABADB3");

string query = "SELECT id FROM [User] WHERE Login = @Login AND Password = @Password";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@Login", Login.Text);

command.Parameters.AddWithValue("@Password", Password.Password);

using (SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

user\_id = reader.GetInt32(0); // Получаем значение столбца id

user\_is\_log\_in = true;

Login.BorderBrush = lightGray;

Password.BorderBrush = lightGray;

}

}

if (user\_is\_log\_in == false)

{

query = "SELECT id FROM Insurance\_Manager WHERE Login = @Login AND Password = @Password";

using (SqlCommand command2 = new SqlCommand(query, connection))

{

command2.Parameters.AddWithValue("@Login", Login.Text);

command2.Parameters.AddWithValue("@Password", Password.Password);

using (SqlDataReader reader2 = command2.ExecuteReader())

{

if (reader2.Read())

{

manager\_id = reader2.GetInt32(0); // Получаем значение столбца id

//if (captcha.CaptchaIsGenerate == false) // Капча не была сгенерирована

//{

// CapIn.BorderBrush = System.Windows.Media.Brushes.Red;

// errors.AppendLine("Пройдите тест CAPTCHA");

//}

//else if (captcha.CheckSequence() != true) // Или капча была пройдена не верно

//{

// CapIn.BorderBrush = System.Windows.Media.Brushes.Red;

// errors.AppendLine("Повторите тест CAPTCHA");

//}

//else { CapIn.BorderBrush = lightGray; }

Login.Text = "";

Password.Password = "";

CapOut.Text = "";

CapIn.Text = "";

captcha.CaptchaIsGenerate = false;

Manager.MainFrame.Navigate(new ManagerPage());

return;

}

else if (!reader2.Read())

{

Login.BorderBrush = System.Windows.Media.Brushes.Red;

Password.BorderBrush = System.Windows.Media.Brushes.Red;

errors.AppendLine("Пользователь не найден");

}

else

{

Login.BorderBrush = lightGray;

Password.BorderBrush = lightGray;

}

}

}

}

//if (captcha.CaptchaIsGenerate == false) // Капча не была сгенерирована

//{

// CapIn.BorderBrush = System.Windows.Media.Brushes.Red;

// errors.AppendLine("Пройдите тест CAPTCHA");

//}

//else if (captcha.CheckSequence() != true) // Или капча была пройдена не верно

//{

// CapIn.BorderBrush = System.Windows.Media.Brushes.Red;

// errors.AppendLine("Повторите тест CAPTCHA");

//}

//else { CapIn.BorderBrush = lightGray; }

if (errors.Length > 0) //Выводи ошибки если есть

{

MessageBox.Show(errors.ToString(), "Ошибка входа" );

CapOut.Text = "";

CapIn.Text = "";

return; // Завершаем исполнение метода и дальше по коду не идём

}

Login.Text = "";

Password.Password = "";

CapOut.Text = "";

CapIn.Text = "";

captcha.CaptchaIsGenerate = false;

user\_is\_log\_in = false;

Manager.MainFrame.Navigate(new UserPage());

}

}

}

private void GenerateRandomSequence(object sender, RoutedEventArgs e)

{

CapOut.Text = captcha.GenerateRandomSequence(); //Записываем в наш текстбокс то, что скажет капча из экземпляра класса

}

}

}

# **3.2.2 Разработка модуля CAPTCHA**

Модуль CAPTCHA используется для не допущения в систему ботов, которые могут сильно замедлить работу системы нагружая её множественными запросами, поэтому очень важно защитить информационную систему от такого вида угроз. Реализацию модуля CAPTCHA представлена ниже:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace Insurance\_сompany

{

internal class CAPTCHA

{

/// Создаём поля для взаимодействия с капчей

public string CapIn { get; set; }

public string CapOut { get; set; }

public bool CaptchaIsGenerate { get; set; } // Создаём приватное поле с типом данных true/false

private string GenerateRandomString(int length)

{

const string chars = "abcdefghijklmnopqrstuvwxyz0123456789"; // Создаём константу с буковами и цыферами

StringBuilder sb = new StringBuilder(); // Инициализируем экземпляр класса СтрингБилдер

Random random = new Random(); // Инициализируем экземпляр класса Рандом

for (int i = 0; i < length; i++)

{

sb.Append(chars[random.Next(chars.Length)]); // Добавляем в экземпляр класса стирнгБилдер заданное количество случайных цыфарок и буковок

}

CaptchaIsGenerate = true;

return sb.ToString(); // Возвращаем строковую случайную последовательность символов

}

public string GenerateRandomSequence()

{

string randomSequence = GenerateRandomString(6); // Инициализируем экемпляр функции генерации случайной последовательности символов

return randomSequence; // Записываем экземпляр функции в текстбокс

}

public bool CheckSequence()

{

/// Сравниваем один тестбокс с другим

if (CapOut == CapIn) { return true; } // Если да, то возвращаем тру

else { return false; } // Если нет, то нет

}

}

}

Данный модуль может вызываться из любого другого модуля, что делает его универсальным и исключает необходимость копировать его в разных модулях.

# **3.2.3 Разработка модуля Создание заявки**

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Windows.Controls.Primitives;

using System.Xml.Linq;

namespace Insurance\_сompany

{

/// <summary>

/// Логика взаимодействия для CreateOrderPage.xaml

/// </summary>

public partial class CreateOrderPage : Page

{

public CreateOrderPage()

{

InitializeComponent();

using (SqlConnection connection = new SqlConnection(Manager.connectionString))

{

connection.Open();

string query = "SELECT Name FROM Type\_Insurance";

using (SqlCommand command = new SqlCommand(query, connection))

{

using (DataTable dataTable = new DataTable())

{

using (SqlDataAdapter adapter = new SqlDataAdapter(command))

{

try

{

adapter.Fill(dataTable);

TypeInsurance.ItemsSource = dataTable.DefaultView;

}

catch (Exception ex)

{

MessageBox.Show("Error: " + ex.Message);

}

}

}

}

query = "SELECT Bank, Payment\_Account FROM UserBank WHERE User\_Id = @user\_id";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@user\_id", LoginPage.user\_id);

using (SqlDataAdapter adapter = new SqlDataAdapter(command))

{

DataTable dataTable = new DataTable(); adapter.Fill(dataTable);

Bank.ItemsSource = dataTable.AsEnumerable().Select(row => row.Field<string>("Bank")).ToList();

Account.ItemsSource = dataTable.AsEnumerable().Select(row => row.Field<string>("Payment\_Account")).ToList();

}

}

}

}

private void BtnSave\_Click(object sender, RoutedEventArgs e)

{

using (SqlConnection connection = new SqlConnection(Manager.connectionString))

{

connection.Open();

int user\_type\_id = 0;

int selectedIndex = TypeInsurance.SelectedIndex;

string query = "SELECT User\_Type\_Id FROM [User] WHERE id = @user\_id";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@user\_id", LoginPage.user\_id);

using (SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

user\_type\_id = reader.GetInt32(0);

}

}

}

if (user\_type\_id == 1)

{

query = "SELECT Passport\_Num, Burth\_Date FROM Individual\_User WHERE User\_Id = @user\_id";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@user\_id", LoginPage.user\_id);

using (SqlDataReader reader = command.ExecuteReader())

{

try

{

reader.Read();

if ((reader["Passport\_Num"].ToString() == "" && reader["Burth\_Date"].ToString() == "") || reader["Passport\_Num"].ToString() == "" || reader["Burth\_Date"].ToString() == "")

{

MessageBox.Show("Введите Номер паспорта и дату рождения в личном кабиненте", "Ошибка!");

return;

}

}

catch { }

}

}

} // Паспорт

if ((Bank.ToString() == null && Account.ToString() == null) || Bank.ToString() == null || Account.ToString() == null)

{

MessageBox.Show("Введите название банка и номер рассчётного счёта", "Ошибка!");

return;

}

int User\_Bank\_Id = 0;

query = "SELECT id FROM UserBank WHERE Bank = @bank AND Payment\_Account = @paymentAccount AND User\_Id = @userId";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@bank", Bank.Text);

command.Parameters.AddWithValue("@paymentAccount", Account.Text);

command.Parameters.AddWithValue("@userId", LoginPage.user\_id);

using (SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

User\_Bank\_Id = reader.GetInt32(0);

Console.WriteLine($"Банк найден, Банку присвоено значение {User\_Bank\_Id}");

}

}

}

if (User\_Bank\_Id == 0)

{

query = "INSERT INTO UserBank (Bank, Payment\_Account, User\_Id) VALUES (@bank, @paymentAccount, @user\_id)";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@bank", Bank.Text);

command.Parameters.AddWithValue("@paymentAccount", Account.Text);

command.Parameters.AddWithValue("@user\_id", LoginPage.user\_id);

command.ExecuteNonQuery();

Console.WriteLine($"Банк не найден, новый банк добавлен");

}

query = "SELECT id FROM UserBank WHERE Bank = @bank AND Payment\_Account = @paymentAccount AND User\_Id = @userId";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@bank", Bank.Text);

command.Parameters.AddWithValue("@paymentAccount", Account.Text);

command.Parameters.AddWithValue("@userId", LoginPage.user\_id);

using (SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

User\_Bank\_Id = reader.GetInt32(0);

Console.WriteLine($"Банк не найден, Банку присвоено значение {User\_Bank\_Id}");

}

}

}

}

query = "INSERT INTO [Order] (Type\_Insurance\_Id, User\_Type\_Id, User\_Id, Insurance\_Object, Filing\_Date, User\_Bank\_Id) VALUES( @Type\_Insurance\_Id, @User\_Type\_Id, @User\_Id, @Insurance\_Object, @Filing\_Date, @User\_Bank\_Id);";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@Type\_Insurance\_Id", selectedIndex + 1); // Вместо Логин пишем то, что в Текстбоксе, и так далее

command.Parameters.AddWithValue("@User\_Id", LoginPage.user\_id); // Вместо ЮсерТайпИд пишем 1

command.Parameters.AddWithValue("@User\_Type\_Id", user\_type\_id);

command.Parameters.AddWithValue("@Insurance\_Object", Insurance\_Object.Text);

command.Parameters.AddWithValue("@Filing\_Date", DateTime.Now);

command.Parameters.AddWithValue("@User\_Bank\_Id", User\_Bank\_Id);

command.ExecuteNonQuery();

MessageBox.Show("Ваша заявка была принята в работу, скоро с вами свяжется менеджер", "Успех!");

Manager.MainFrame.GoBack();

}

}

}

}

}

# **3.2.4 Разработка модуля Рассмотрение заявки**

Модуль рассмотрения заявки спроектирован для менеджера, который должен связаться с клиентом перед заполнением данных, для чего ему предоставляется контактная информация клиента.

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Net;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Xml.Linq;

namespace Insurance\_сompany

{

/// <summary>

/// Логика взаимодействия для OrderPage.xaml

/// </summary>

public partial class OrderPage : Page

{

private string userId = "";

private string typeInsuranceId = "";

private string userBankId = "";

private string typeUserId = "";

private DateTime Date\_Of\_Registration;

public OrderPage()

{

InitializeComponent();

using (SqlConnection connection = new SqlConnection(Manager.connectionString))

{

connection.Open();

string query = "" +

"SELECT " +

"[Order].Type\_Insurance\_Id, " +

"[Order].Insurance\_Object, " +

"[Order].User\_Bank\_Id, " +

"[Order].Filing\_Date, " +

"[User].id " +

"FROM [Order] " +

"JOIN [User] " +

"ON [Order].User\_Id = [User].id " +

"WHERE [Order].id = @OrderId;";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@OrderId", ManagerPage.selectedOrder);

using (SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

typeInsuranceId = reader["Type\_Insurance\_Id"].ToString();

Insurance\_Object.Text = reader["Insurance\_Object"].ToString();

userBankId = reader["User\_Bank\_Id"].ToString();

Date\_Of\_Registration = Convert.ToDateTime(reader["Filing\_Date"]);

userId = reader["id"].ToString();

}

}

}

query = "" +

"SELECT " +

"Type\_Insurance.Name " +

"FROM Type\_Insurance " +

"WHERE Type\_Insurance.id = @typeInsuranceId;";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@typeInsuranceId", typeInsuranceId);

using (SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

Type\_Insurance\_Id.Text = reader["Name"].ToString();

}

}

}

query = "" +

"SELECT " +

"User\_Type\_Id " +

"FROM [User] " +

"WHERE id = @userId";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@userId", userId);

using (SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

typeUserId = reader["User\_Type\_Id"].ToString();

}

}

}

if (typeUserId == "1")

{

query = "" +

"SELECT " +

"F\_Name, " +

"L\_Name, " +

"M\_Name " +

"FROM Individual\_User " +

"WHERE User\_Id = @userId;";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@userId", userId);

using (SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

FLMNameUser.Text = reader["L\_Name"].ToString() + " " + reader["F\_Name"].ToString() + " " + reader["M\_Name"].ToString();

}

}

}

}

else if (typeUserId == "2")

{

query = "" +

"SELECT " +

"Name " +

"FROM Legal\_User " +

"WHERE User\_Id = @userId;";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@userId", userId);

using (SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

FLMNameUser.Text = reader["Name"].ToString();

}

}

}

}

query = "" +

"SELECT " +

"Bank, " +

"Payment\_Account " +

"FROM UserBank " +

"WHERE id = @userBankId;";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@userBankId", userBankId);

using (SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

UserBankId.Text = reader["Bank"].ToString();

UserPaymentAccount.Text = reader["Payment\_Account"].ToString();

}

}

}

}

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

DateTime dateStart = Date\_Start.SelectedDate.GetValueOrDefault();

int insurance\_Id = 0;

using (SqlConnection connection = new SqlConnection(Manager.connectionString))

{

connection.Open();

string query = "INSERT INTO " +

"Insurance\_Policy " +

"(" +

"Type\_Insurance\_Id, " +

"User\_Type\_Id, " +

"User\_Id, " +

"Insurance\_Prize, " +

"Insurance\_Object, " +

"Insurance\_Case, " +

"Insurance\_Sum, " +

"Date\_Of\_Conclusion, " +

"Date\_Of\_Registration, " +

"Date\_Start, " +

"Date\_End, " +

"Status, " +

"User\_Bank\_Id, " +

"Manager\_Id" +

") " +

"VALUES " +

"(" +

"@Type\_Insurance\_Id, " +

"@User\_Type\_Id, " +

"@User\_Id, " +

"@Insurance\_Prize, " +

"@Insurance\_Object, " +

"@Insurance\_Case, " +

"@Insurance\_Sum, " +

"@Date\_Of\_Conclusion, " +

"@Date\_Of\_Registration, " +

"@Date\_Start, " +

"@Date\_End, " +

"@Status, " +

"@User\_Bank\_Id, " +

"@Manager\_Id" +

"); SELECT SCOPE\_IDENTITY();";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@Type\_Insurance\_Id", typeInsuranceId);

command.Parameters.AddWithValue("@User\_Type\_Id", typeUserId);

command.Parameters.AddWithValue("@User\_Id", userId);

command.Parameters.AddWithValue("@Insurance\_Prize", Insurance\_Prize.Text);

command.Parameters.AddWithValue("@Insurance\_Object", Insurance\_Object.Text);

command.Parameters.AddWithValue("@Insurance\_Case", Insurance\_Case.Text);

command.Parameters.AddWithValue("@Insurance\_Sum", Insurance\_Sum.Text);

command.Parameters.AddWithValue("@Date\_Of\_Conclusion", DateTime.Now);

command.Parameters.AddWithValue("@Date\_Of\_Registration", Date\_Of\_Registration);

command.Parameters.AddWithValue("@Date\_Start", dateStart);

command.Parameters.AddWithValue("@Date\_End", dateStart.AddMonths(int.Parse(Date\_Of\_Conclusion.Text)));

command.Parameters.AddWithValue("@Status", 1);

command.Parameters.AddWithValue("@User\_Bank\_Id", userBankId);

command.Parameters.AddWithValue("@Manager\_Id", LoginPage.manager\_id);

insurance\_Id = Convert.ToInt32(command.ExecuteScalar());

command.ExecuteNonQuery();

}

query = "DELETE FROM [Order] WHERE id = @orderId;";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@orderId", ManagerPage.selectedOrder);

command.ExecuteNonQuery();

}

query = "INSERT INTO Reports (Manager\_Id, Date\_Operation, Id\_Opeation, Id\_Policy) VALUES (@Manager\_Id, @Date\_Operation, 1, @Id\_Policy)";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@Manager\_Id", LoginPage.manager\_id);

command.Parameters.AddWithValue("@Date\_Operation", DateTime.Now);

command.Parameters.AddWithValue("@Id\_Policy", insurance\_Id);

command.ExecuteNonQuery();

}

MessageBox.Show("Страховой полис офрмлен", "Успех!");

Manager.MainFrame.GoBack();

}

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

using (SqlConnection connection = new SqlConnection(Manager.connectionString))

{

connection.Open();

string query = query = "DELETE FROM [Order] WHERE id = @orderId;";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@orderId", ManagerPage.selectedOrder);

command.ExecuteNonQuery();

}

query = "INSERT INTO Reports (Manager\_Id, Date\_Operation, Id\_Opeation) VALUES (@Manager\_Id, @Date\_Operation, 2)";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@Manager\_Id", LoginPage.manager\_id);

command.Parameters.AddWithValue("@Date\_Operation", DateTime.Now);

command.ExecuteNonQuery();

}

MessageBox.Show("Заявка отклонена", "Успех!");

Manager.MainFrame.GoBack();

}

}

}

}

# **3.2.5 Разработка модуля «Отчет»**

private void BtnShow\_Click(object sender, RoutedEventArgs e)

{

var application = new Word.Application();

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

if (InsurancePolicies.SelectedItem != null)

{

DataRowView row = (DataRowView)InsurancePolicies.SelectedItem;

selectedPolicy = row["id"].ToString();

Console.WriteLine("id выбранного договора: " + selectedPolicy);

}

using (SqlConnection connection = new SqlConnection(Manager.connectionString))

{

connection.Open();

string query = "SELECT Type\_Insurance\_Id FROM Insurance\_Policy WHERE id = @id";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@id", selectedPolicy);

using (SqlDataReader reader = command.ExecuteReader())

{

insuranceTypeID = reader.GetInt32(0).ToString();

Console.WriteLine("id выбранного типа договора: " + insuranceTypeID);

}

}

query = "SELECT Name FROM Type\_Insurance WHERE id = @id";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@id", insuranceTypeID);

using (SqlDataReader reader = command.ExecuteReader())

{

Word.Paragraph paragraph = document.Paragraphs.Add();

Word.Range range = paragraph.Range;

range.Text = $"Страховой полис. Тип страхового договора: {reader["Name"]}";

}

}

}

application.Visible = true;

}

# **ЗАКЛЮЧЕНИЕ**

Цель данного курсового проекта по разработке информационной системы, которая позволит упростить и автоматизировать процессы страхования, сократить временные затраты на обработку информации, улучшить качество обслуживания клиентов и повысить общую эффективность бизнеса страховой компании были выполнены.

Был проведён анализ деятельности страховых фирм, проектирование информационной системы, а также разработка информационной системы

В ходе работы над проектом были рассмотрены основные требования к информационной системе для страховой фирмы, разработаны алгоритмы работы системы, спроектирована база данных, создан пользовательский интерфейс и проведено тестирование функциональности.

Результаты данного курсового проекта могут быть использованы страховыми компаниями для улучшения своей деятельности, повышения конкурентоспособности на рынке и удовлетворения потребностей клиентов.

# **СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**