Федеральное государственное бюджетное образовательное учреждение высшего образования

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**Департамент анализа данных,**

**принятия решений и финансовых технологий**

**Пояснительная записка к междисциплинарной курсовой работе**

на тему:

**Разработка информационно-справочной системы**

**«Климатические показатели регионов по годам»**

Выполнила:

студент группы ПИ18-1

Пыльцына А. C.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

Научный руководитель:

доцент, к.т.н. Гайдамака А. И.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

2020

**Оглавление**

**Введение 3**

**1. Постановка задачи 3**

**2. Описание предметной области 5**

**3. Актуальность автоматизации 5**

[**4. Описание программы**](file:///C:\Users\3C8A~1\AppData\Local\Temp\КР-Пояснительная%20записка.docx#_tyjcwt) **6**

[4.1. Алгоритмические решения](file:///C:\Users\3C8A~1\AppData\Local\Temp\КР-Пояснительная%20записка.docx#_3dy6vkm) 6

[4.2. Описание интерфейса программы 6](file:///C:\Users\3C8A~1\AppData\Local\Temp\КР-Пояснительная%20записка.docx#_1t3h5sf)

[4.3. Состав приложения](file:///C:\Users\3C8A~1\AppData\Local\Temp\КР-Пояснительная%20записка.docx#_4d34og8) 10

[**5. Назначение и состав классов программы**](file:///C:\Users\3C8A~1\AppData\Local\Temp\КР-Пояснительная%20записка.docx#_2s8eyo1) **11**

[5.1. Диаграмма классов](file:///C:\Users\3C8A~1\AppData\Local\Temp\КР-Пояснительная%20записка.docx#_17dp8vu) 11

[5.2. Класс](file:///C:\Users\3C8A~1\AppData\Local\Temp\КР-Пояснительная%20записка.docx#_3rdcrjn) регионы11

[5.3. Класс](file:///C:\Users\3C8A~1\AppData\Local\Temp\КР-Пояснительная%20записка.docx#_3rdcrjn) текущие показатели [11](file:///C:\Users\3C8A~1\AppData\Local\Temp\КР-Пояснительная%20записка.docx#_3rdcrjn)

[5.4. Класс](file:///C:\Users\3C8A~1\AppData\Local\Temp\КР-Пояснительная%20записка.docx#_3rdcrjn) SQL запросов12

[5.4. Формы](file:///C:\Users\3C8A~1\AppData\Local\Temp\КР-Пояснительная%20записка.docx#_lnxbz9) 12

[**Заключение**](file:///C:\Users\3C8A~1\AppData\Local\Temp\КР-Пояснительная%20записка.docx#_35nkun2) **13**

**Список литературы 13**

[**Приложение. Исходный код программы**](file:///C:\Users\3C8A~1\AppData\Local\Temp\КР-Пояснительная%20записка.docx#_44sinio) **20**

**Введение**

Целью курсовой работы является приобретение навыков в анализе предметной области и в разработке сложного Windows-приложения на объектно-ориентированном языке программирования C# в среде Visual Studio. Язык С# является одним из наиболее мощных языков для разработки настольных приложений. Разработка приложения позволит углубить знания этого языка, а также методов визуального проектирования и анализа предметной области.

Анализ предметной области позволяет разработать информационную модель задачи. Важным этапом этого анализа является выбор из предметной области наиболее важных объектов и процессов автоматизации, а также разработка новых IT-подходов и функций для решения проблем предметной области.

Второй целью курсовой работы, и не менее важной, является разработка пояснительной записки, отвечающей таким требованиям, как полнота, лаконичность, техническая корректность и т.д. Техническая документация является составной частью программного продукта и, следовательно, определяет его качество. Поэтому приобретение навыков в разработке технической документации, а в данном случае пояснительной записки, также является важной задачей.

**1. Постановка задачи**

В соответствии с выбранной темой требуется разработать Windows-приложение, выполняющее мониторинг показателей регионов по годам, а также реализующее сортировку показателей, удовлетворяющих критерию.

Приложение должно содержать средства манипулирования объектами, сохранения их в файле и загрузки из файла.

*Требования:*

1. Разрабатываемое приложение должно представлять собой Windows приложение.
2. Должны быть разработаны пользовательские классы (один или несколько), описывающие объекты предметной области, и несколько форм пользовательского интерфейса.
3. Разработчик самостоятельно определяет интерфейс программы и ее функциональность, однако приложение в обязательном порядке независимо от предметной области, указанной в задании, должно выполнять следующие операции:

* Отображать в сетках DataGridView данные предметной области:
* Для информационной модели, основанной на списках, на момент первого запуска программы допускается отсутствие файла. В этом случае списки объектов должны быть созданы в программе.
* Реализовать добавление в источник данных нового объекта, удаление объекта из источника данных, редактирование объекта в источнике данных.
* Реализовать фильтрацию записей источника данных, удовлетворяющих введенному пользователем сложному критерию.
* Реализовать сортировку записей источника данных.
* Для информационной модели, основанной на списках, сохранять источник данных в файле баз данных.
* После сохранения данных при запуске программы загрузить данные из файла или базы данных.
* Используя меню или панель инструментов, вызвать приложение Блокнот для просмотра справки о программе: файл Help.txt текущего каталога программы.
* Создать пункт меню «Об авторе».
* Разработать несколько полезных пользователю функций для отображения статистических данных, например, средних, максимальных или минимальных значений, данных для построения гистограммы или графика и т.п.

1. Список должен быть реализован в виде коллекции, например, динамического массива типа List<T> или BindingList<T>. Программа не должна завершаться аварийно: сообщения о некорректном вводе данных, противоречивых или недопустимых значениях данных, при отсутствии данных по функциональному запросу пользователя и других нештатных ситуациях отображать в окнах сообщений.
2. Программа должна быть читабельной и содержать полезные комментарии.

**2. Описание предметной области**

В разрабатываемом приложении предметной областью автоматизации является статистический анализ данных годовых температурных показателей различных регионов. C точки зрения заказчика, сбор информации среднегодовых показателей температуры для последующего анализа и обработки является источником дохода.

Анализ данных организован следующим образом: собираются данные за текущие и прошлые температурные показатели определенного региона, пользователь заносит их в базу данных, затем проводится анализ полученных значений.

Большая часть работы по анализу температурных показателей может быть обеспечена современными IT-технологиями. Однако непосредственный сбор и обработка данных является наиболее приоритетной и распространённой задачей, поэтому этот вид деятельности необходимо автоматизировать в первую очередь.

Анализ данных определенного региона определяется заполнением среднегодовых температурных показателей этого региона. При заполнении данных о температурном показателе, берется год и среднегодовая температура за этот год.

**3. Актуальность автоматизации**

Погода очень важна не только для урожая, но и для настроения человека. Поэтому люди уделяют столько внимания прогнозам погоды. Статистические методы прогнозирования погоды используют прошлые записи метеорологических данных, исходя из предположения что в будущем погода будет повторяться. Основная цель изучения прошлых метеорологических данных – выяснить те аспекты погоды, которые являются хорошими показателями будущих событий. Таким образом можно делать прогноз погоды с большим шагом по территории. Это особенно полезно при проектировании только одного аспекта погоды за раз. Статистический метод имеют большое значение для долгосрочных прогнозов погоды.

Приложение позволяет обрабатывать информацию о средних климатических показателях по регионам, например, получить различные статистические данные, такие как минимальную, среднюю и максимальную температуру за определенные период времени, влажность воздуха, давление, силу ветра и количество осадков. Так же приложение позволяет построить график для более детального изучения.

Поскольку программа позволит сократить непроизводительные, ручные затраты работников по работе с обработкой данных и поднять уровень анализа среднегодовых климатических показателей, автоматизация этой предметной области является **актуальной** задачей. Для решения этой актуальной задачи разработано приложение, предназначенное для анализа средних климатических показателей разных регионов по различным критериям. Также программа предназначена для выполнения ряда сервисных функций по работе с базой данных. Программа позволяет вводить, изменять, сохранять и удалять годовые показатели из файла.

**4. Описание программы**

**4.1. Алгоритмические решения**

Хранение списка регионов и показателей организовано в БД-файле Microsoft Access. Главная форма программы состоит из форм, на которых располагаются таблица с данными показателей, списка регионов, а так же элементы управления записями (добавление, изменение, удаление). Из окна главной формы так же возможно выводить форму редактирования записей, отчеты по показателям (построение графика, минимальная, средняя и максимальная температура), настройки и сведения о программе. Форма редактирования записей позволяет добавлять и изменять записи в базе данных. При работе осуществляется контроль вносимых данных и защита от некорректного ввода. Таблица предоставляет удобную сортировку в зависимости от требований пользователя. Выход из приложения осуществляется нажатием на кнопку “Закрыть” в верхнем правом углу главной формы программы.

Просмотр записей регионов и показателей осуществляется через главную форму программы. При выборе региона в списке и нажатии на элемент, обновляются данные, взятые из БД таблицы Показатели, для элемента DataGridView, соответствующие текущему выбранному региону.

Чтобы получить дополнительную информацию, которую хочет извлечь пользователь, в программе предусмотрена сортировка по разным критериям. Для этого необходимо щелкнуть на желаемую колонку таблицы. Сортировка позволяет построить показатели от большего к меньшему или наоборот для удобства просмотра.

Пункт меню «Об авторе», содержит описание процесса использования программы. Другой пункт - «О программе» - вызывает MessageBox, где отображена информация о создателе приложения. Пункт «Построить график по текущему региону» - расширяет окно на котором есть элемент Chart, который рисует гистограмму показателей по годам по выбранному текущему региону.

**4.2. Описание интерфейса программы**

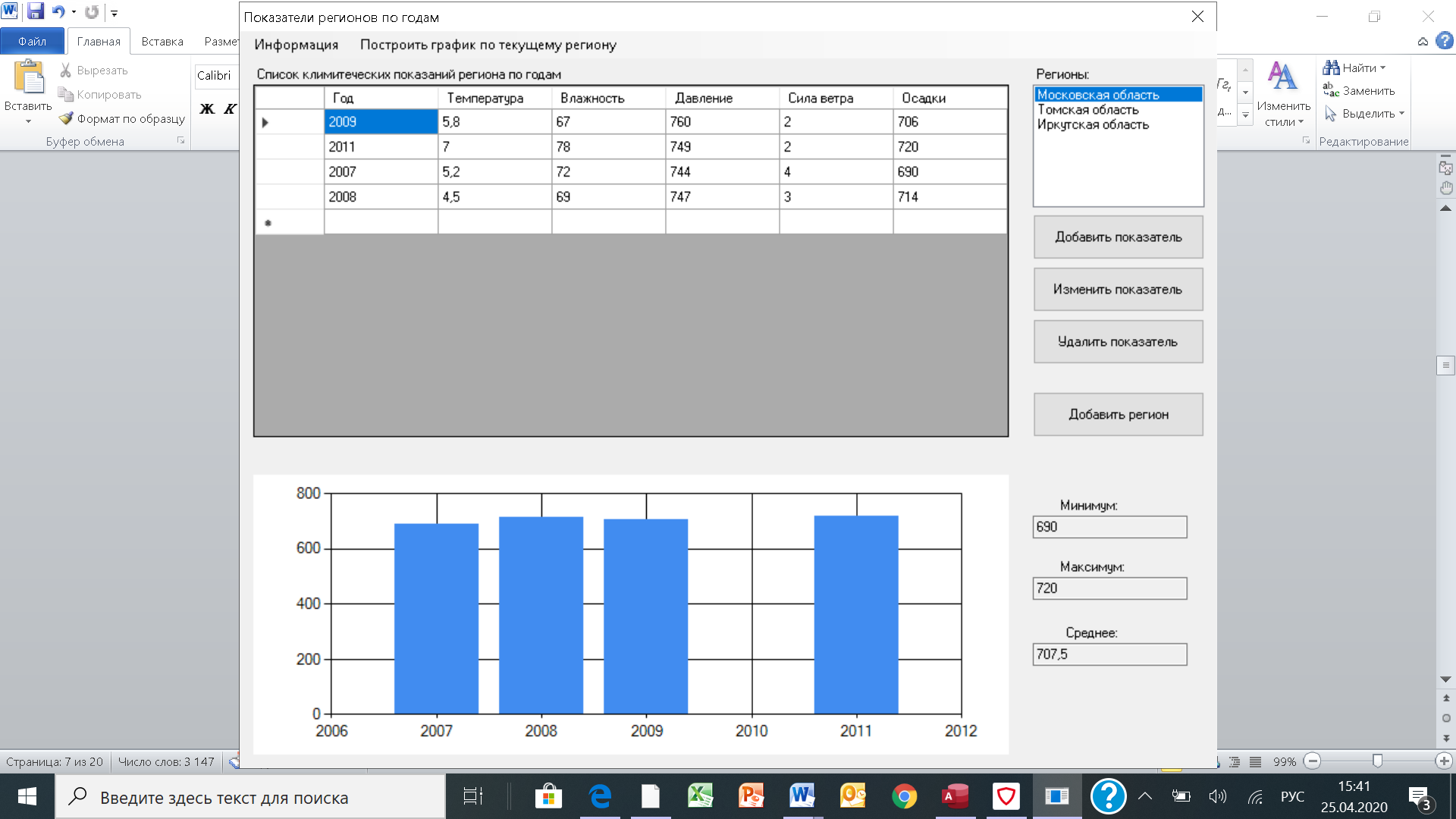
В приложении для реализации интерфейса используется 7 элементов управления и компонентов:

* DataGridView;
* Button;
* TextBox;
* Label;
* Chart;
* ListBox;
* MenuStrip.

При открытии формы элемент ListBox заполняется данными взятыми из БД таблицы Регионы, а в программе заполняется список, состоящий из элементов класса Region List<Region>.

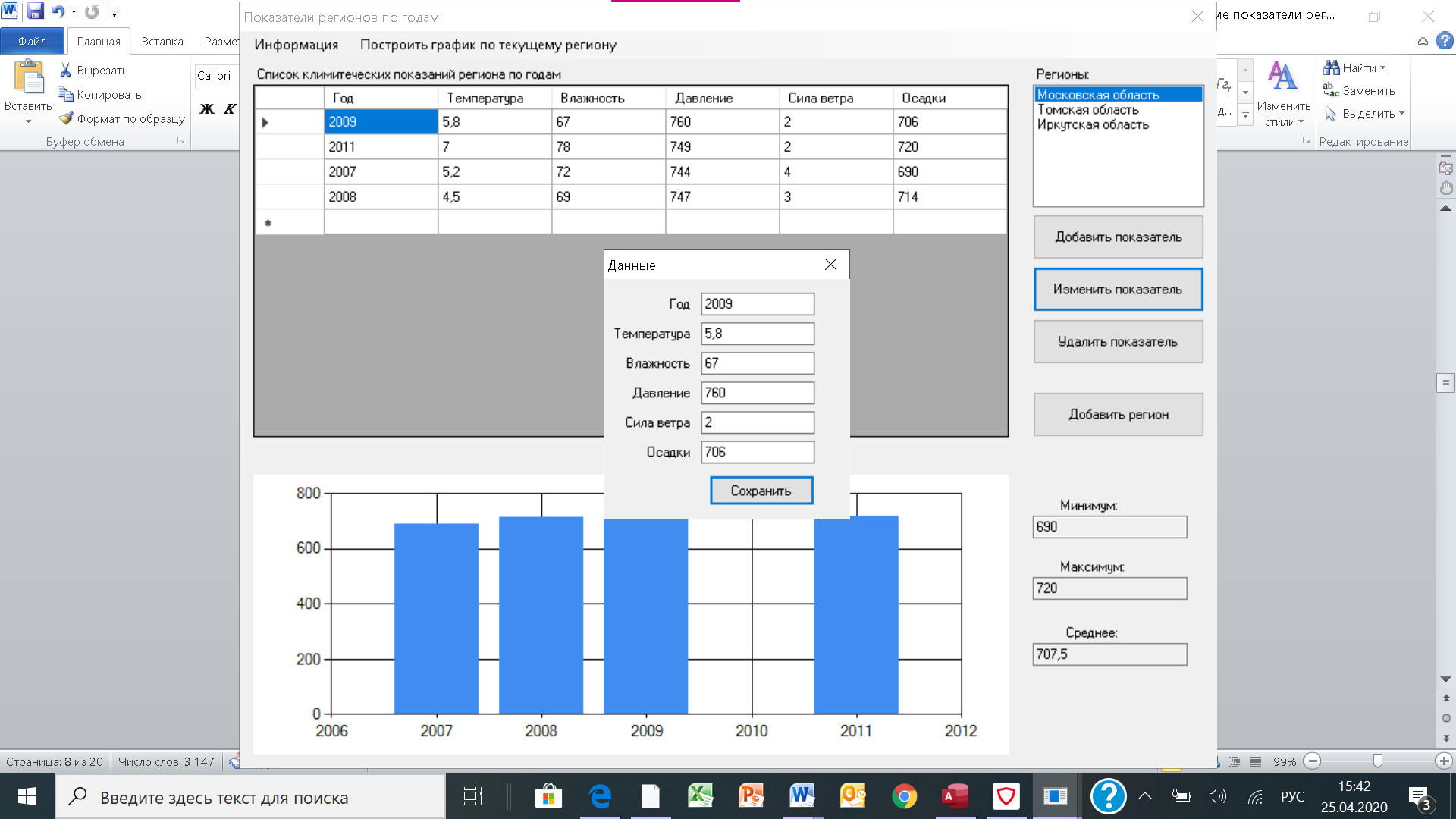
Наиболее сложным элементом управления является таблица (сетка) DataGridView. Загрузка в таблицу записей показателей регионов (DataGridView1) осуществляется благодаря сформированному заранее SQL запросу к БД и получения ответа на запрос в зависимости от выбранного региона.

Запуская Windows приложение, мы попадаем на стартовую страницу, где выведены в списке все регионы и таблица с показателями текущего выбранного региона(рис. 4.1).

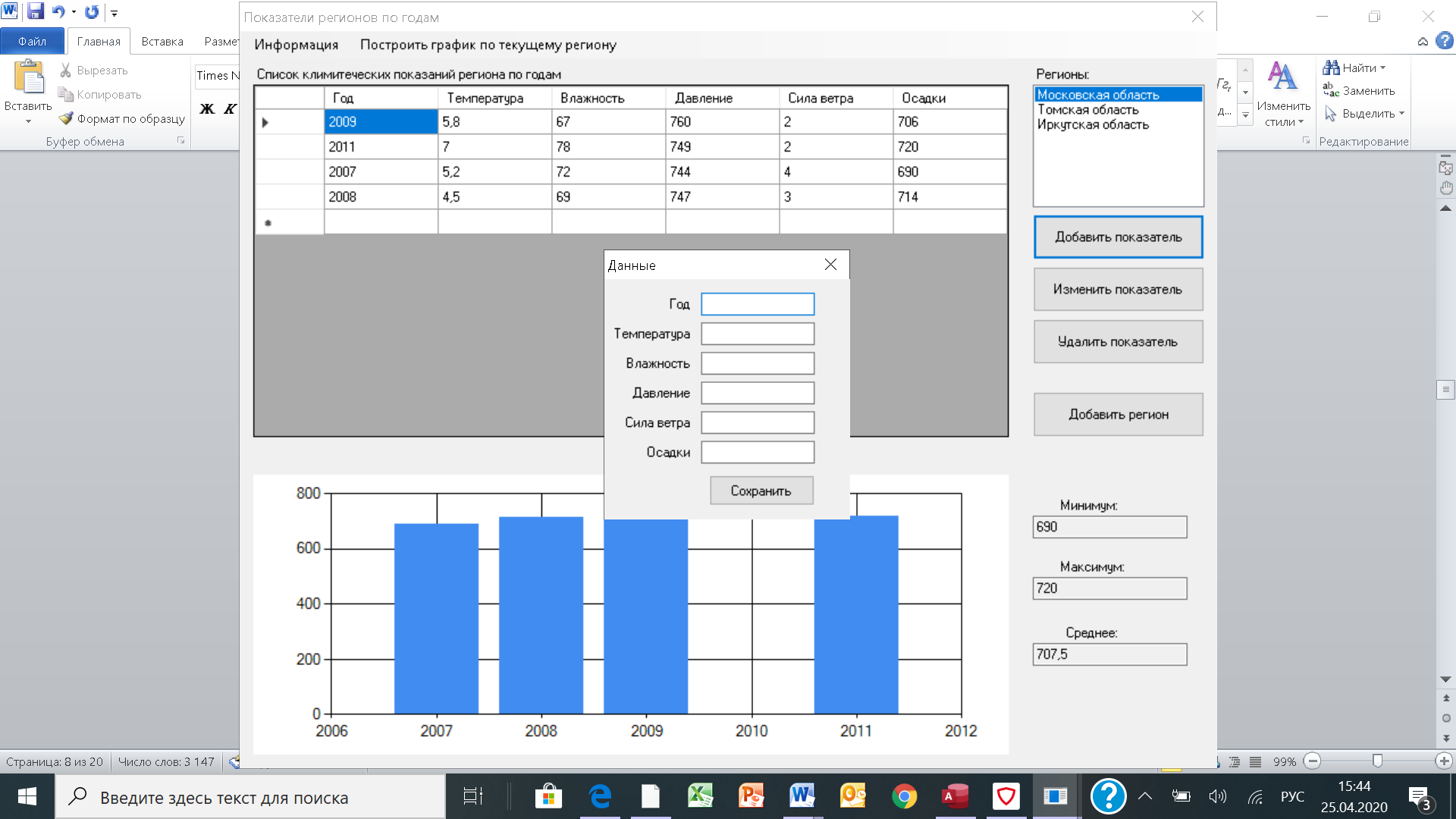


***Рис. 4.1.*** *Список Регионов и таблица Показателей*

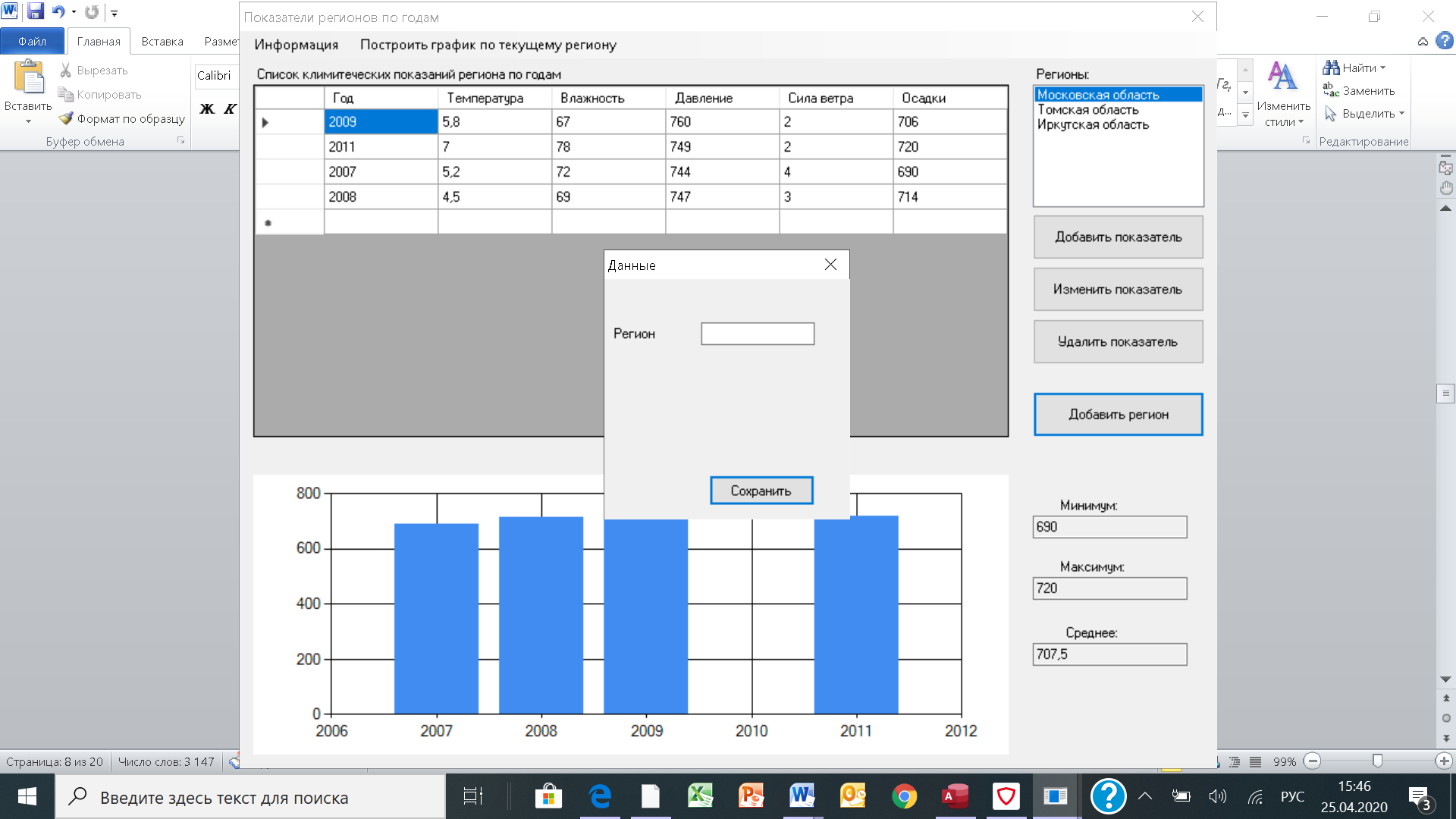
Данные в БД-файле сохраняются сразу после нажатия кнопки «Сохранить» в окне «Данные», таким образом можно сразу же добавлять, изменять и удалять данные напрямую из БД.



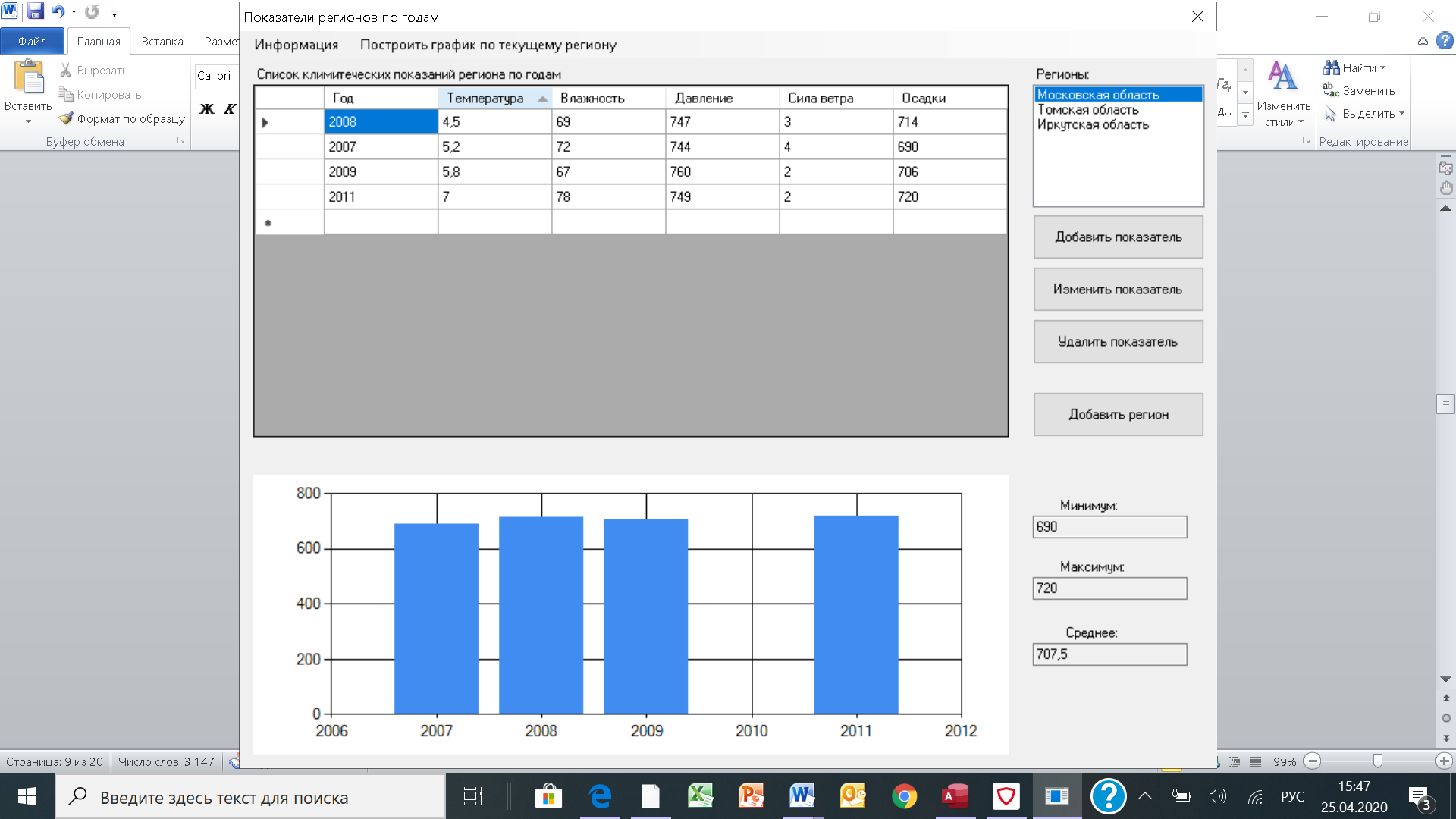
***Рис. 4.1.1*** *Изменение данных Показателей*



***Рис. 4.1.2*** *Добавление данных Показателей*

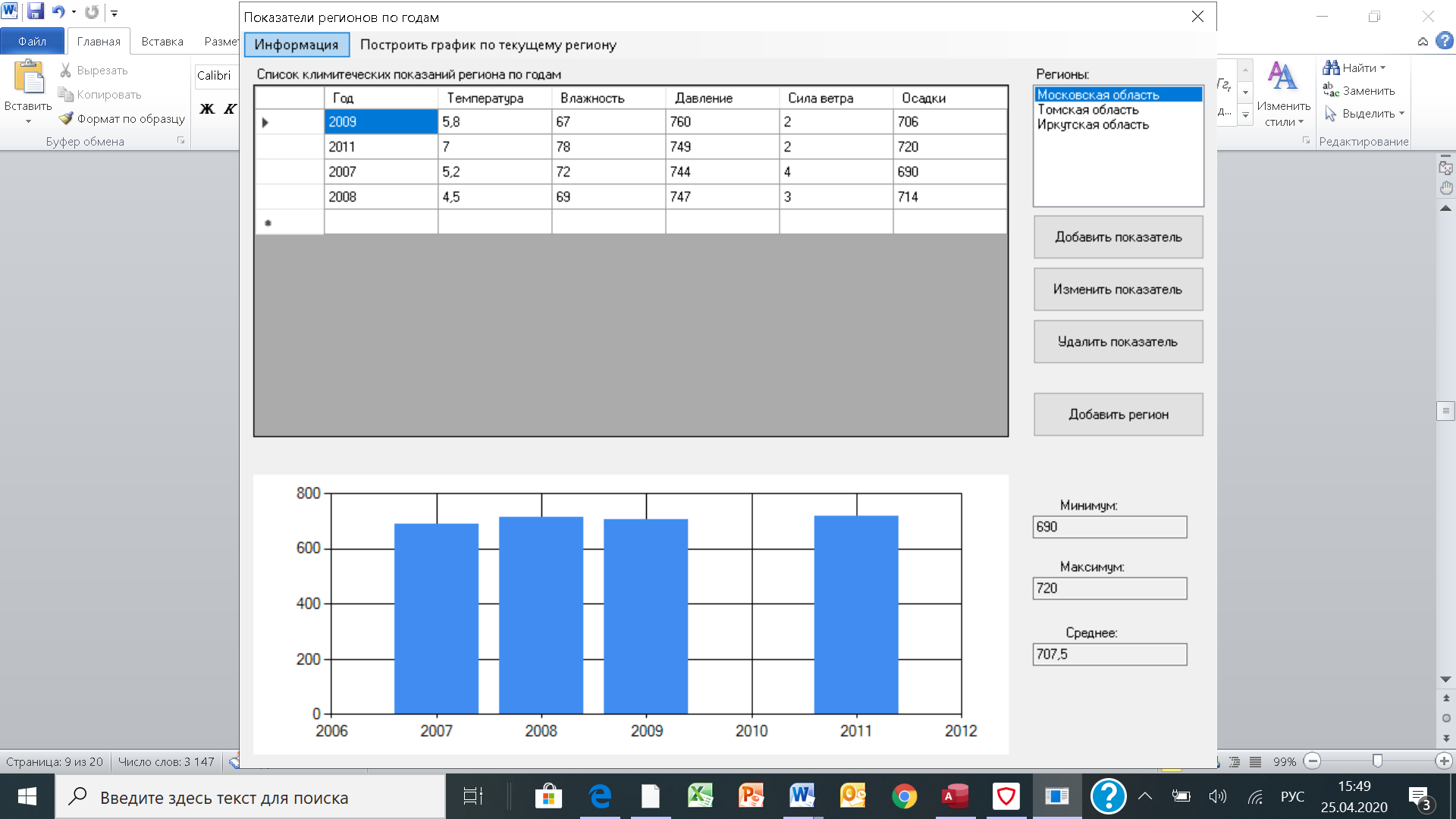


***Рис. 4.1.3*** *Добавление данных Региона*

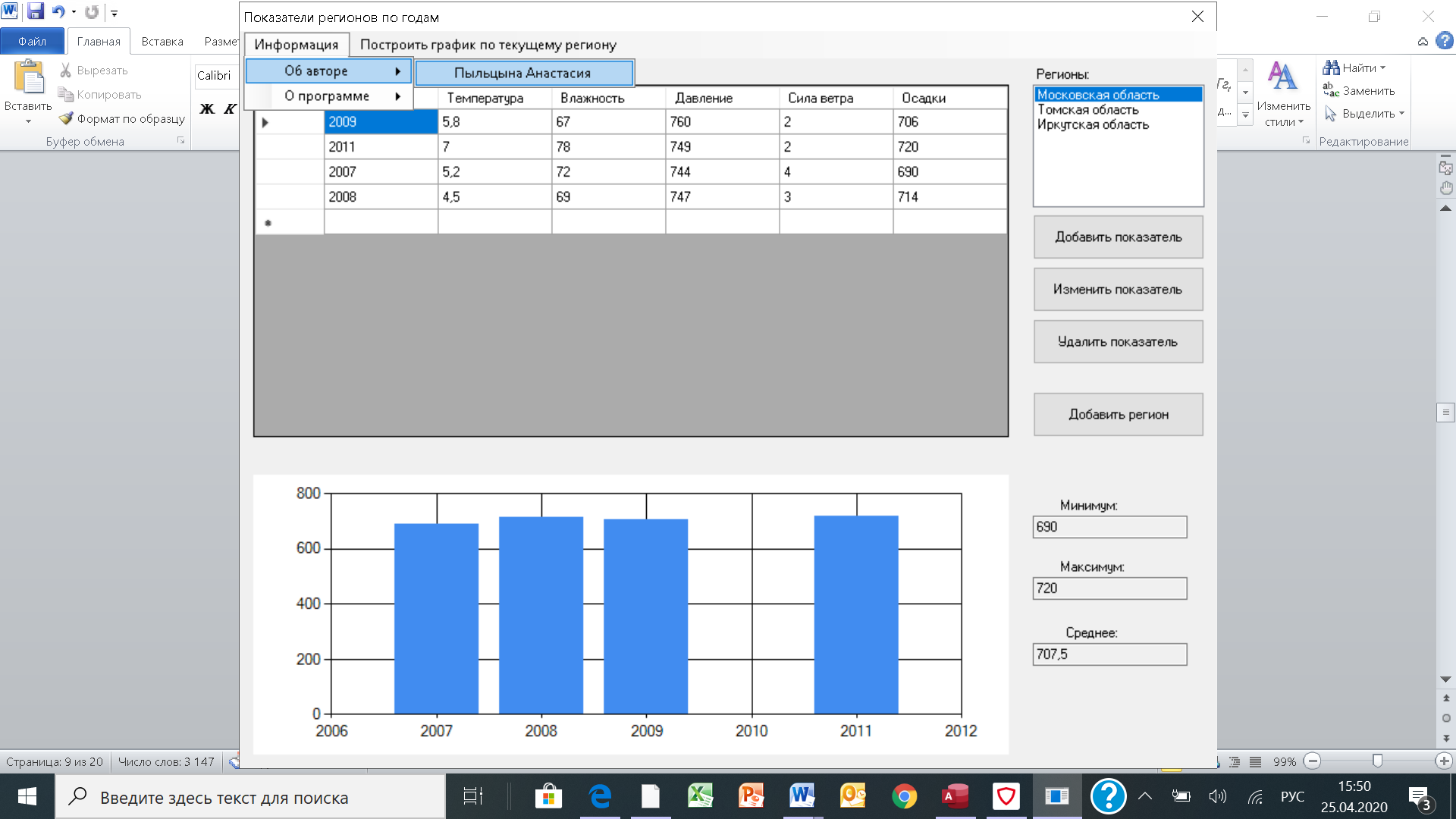


***Рис. 4.2*** *Сортировка показателей*

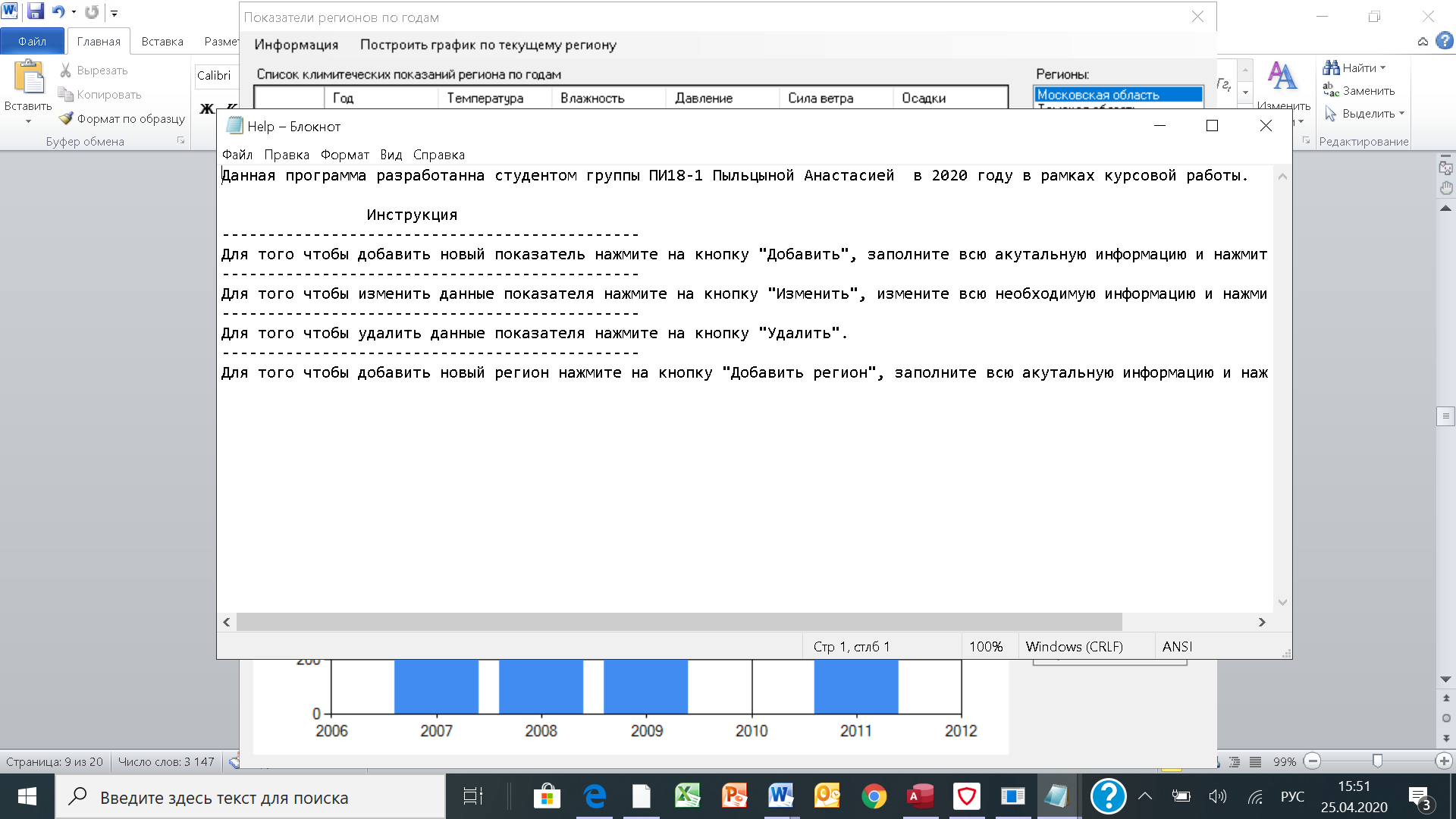
На рис. 4.2 представлена сортировка, в которой средние показатели, отсортированы по возрастанию.



***Рис. 4.3*** *Вкладка «Информация»*



***Рис. 4.3.1*** *Вкладка «Об авторе»*



***Рис. 4.3.2*** *Вкладка «О программе»*

**4.3. Состав приложения**

В состав приложения входят следующие файлы и папки:

* IBR.exe – программа;
* Help.txt – файл с инструкцией;
* БД.mdb – БД-файл содержащий всю информацию

**5. Назначение и состав классов программы**

**5.1. Диаграмма классов**

Разрабатываемая программа состоит из трех классов, содержащих множество полей, методов и свойств. Взаимосвязь основных классов и форм приложения можно увидеть на диаграмме, изображенной на рис. 5.1. Стрелками изображены ассоциации.

Форма

***Рис. 5.1*** *Структура приложения*

**5.2. Класс Регионы**

Определение класса **Rigion**регионов представлено в Приложении. В классе описаны следующие открытые свойства и конструкторы:

* getRig – свойство, позволяющее возвращать Имя региона;
* getId – свойство, позволяющее возвращать id региона;
* Rigion (*параметры*) – параметрический конструктор, осуществляющий инициализацию свойств класса.

**5.3. Класс Текущий показатель**

Определение класса **CurrentIndicator**текущего показателя представлено в Приложении. Класс представляет собой статический класс, т.е. экземпляр этого класса не может быть создан. В классе описаны следующие открытые свойства:

* RigId – свойство, позволяющее возвращать и устанавливать id региона к которому относится показатель;
* Year – свойство, позволяющее возвращать и устанавливать год;
* Avr – свойство, позволяющее возвращать и устанавливать показатель.
* Vlajnost – свойство, позволяющее возвращать и устанавливать показатель.
* Davlenie – свойство, позволяющее возвращать и устанавливать показатель.
* Veter – свойство, позволяющее возвращать и устанавливать показатель.
* Osadki – свойство, позволяющее возвращать и устанавливать показатель.

**5.4. Класс SQL запросов**

Определение класса **SQL\_query**SQL запросы представлено в Приложении. Класс представляет собой статический класс, т.е. экземпляр этого класса не может быть создан. В классе описаны следующие методы и открытые поля для работы с базой данных:

* update – публичная переменная, флаг изменения, необходимый для корректного управления программы;
* add – публичная переменная, флаг добавления, необходимый для корректного управления программы;
* ConnectionStirng – частная переменная, хранящая в себе строку для подключения к локальной БД;
* SendQuery (*параметры*) – метод, посылающий запрос к базе данных и возвращающий таблицу соответствующую выполненному запросу;
* QueryOfCng (*параметры*) – метод, служить для управления элементами базы данных, таких как добавление, изменение, удаление.

**5.5. Формы**

Ниже представлены пользовательские классы, наследующие базовую функциональность от класса Form.

* Change - класс, содержащий обработчики событий для редактирования и добавления информации таблицы показателей по годам и таблицы регионы;
* Main - класс, содержащий все основные обработчики событий и методы.

**Заключение**

Разработав Windows приложение, был выполнен ряд поставленных задач. Справочно-информационная система написана на языке программирования С# с использованием классов, методов и свойств. Для реализации некоторых задач потребовалось обратиться к объектам операционной системы: системный реестр, файловая система, процессы.

Программа может модернизироваться и обновляться.

Программа может быть расширена статистическими функциями.

**Список литературы**

1. Горелов С.В., Волков А.Г. Разработка Windows-приложений. Часть 1. Учебное пособие. Образовательный портал Финансового университета. 2018.
2. Горелов С.В. Разработка Windows-приложений. Часть 2. Учебное пособие. Образовательный портал Финансового университета. 2018.
3. Г. Шилдт. Полный справочник по C#. - М.: «Вильямс», 2004.
4. Кариев, Ч. А. Разработка Windows-приложений на основе Visual C# (+ CD-ROM) / Ч.А. Кариев. - М.: Интернет-университет информационных технологий, Бином. Лаборатория знаний, 2015.
5. Абрамян, Михаил Visual C# на примерах / Михаил Абрамян. - М.: БХВ-Петербург, 2016.
6. Албахари, Джозеф C# 3.0. Справочник / Джозеф Албахари , Бен Албахари. - М.: БХВ-Петербург, 2015.
7. Официальный сайт Microsoft: [Интернет-ресурс]. URL: https://msdn.microsoft.com org.

**Приложение. Исходный код программы**

**Файл Program.cs:**

using System;

using System.Windows.Forms;

namespace IBR

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Main());

}

}

}

**Файл Rigion.cs:**

namespace IBR

{

public class Rigion//класс регион

{

private int id;//поле id региона

private string rigion;//поле имя региона

public string Rig//геттер получения имени региона

{

get { return rigion; }

}

public int Id//геттер получения id региона

{

get { return id; }

}

public Rigion(int id, string rig)//конструктор класса

{

this.id = id;//присваиваем данные текущему экземпляру

this.rigion = rig;

}

}

}

**Файл CurrentIndicator.cs:**

namespace IBR

{

public static class CurrentIndicator//класс текущего показателя

{

private static int regionId;//id региона

private static string year;//год

private static string avr;//значение

private static string vlajnost;

private static string davlenie;

private static string veter;

private static string osadki;

public static int RigId//геттер сеттер получения id региона

{

set { regionId = value; }

get { return regionId; }

}

public static string Year//геттер сеттер получения года

{

set { year = value; }

get { return year; }

}

public static string Avr//геттер сеттер получения показателя

{

set { avr = value; }

get { return avr; }

}

public static string Vlajnost//геттер сеттер получения показателя

{

set { vlajnost = value; }

get { return vlajnost; }

}

public static string Davlenie//геттер сеттер получения показателя

{

set { davlenie = value; }

get { return davlenie; }

}

public static string Veter//геттер сеттер получения показателя

{

set { veter = value; }

get { return veter; }

}

public static string Osadki//геттер сеттер получения показателя

{

set { osadki = value; }

get { return osadki; }

}

}

}

**Файл SQL\_query.cs:**

using System;

using System.Data;

using System.Data.OleDb;

namespace IBR

{

public static class SQL\_query//класс запросов к базе данных

{

private static string ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" +

@"Data Source=" + Environment.CurrentDirectory + "\\БД.mdb";//строка подключения

public static bool update;//флаг изменения

public static bool add;//флаг добавления

public static DataTable SendQuery(string qr, string table)//функция обрабатывающая запрос к базе данных и возвращает таблицу соответственно выполненому запросу

{

OleDbDataAdapter dataAdapter = new OleDbDataAdapter(qr, ConnectionString);

DataSet ds = new DataSet();

dataAdapter.Fill(ds, table);

return ds.Tables[table];

}

public static void QueryOfCng(string qr)//функция для управления элементами в базе данных (добавление, изменение, удаление)

{

OleDbConnection conn = new OleDbConnection(ConnectionString);

conn.Open();

OleDbCommand myCommand = conn.CreateCommand();

myCommand.CommandText = qr;

myCommand.ExecuteNonQuery();

conn.Close();

}

}

}

**Файл Main.cs:**

using System;

using System.Collections.Generic;

using System.Data;

using System.Windows.Forms;

using System.Diagnostics;

namespace IBR

{

public partial class Main : Form

{

public Main()

{

InitializeComponent();

}

static List<Rigion> listRig = new List<Rigion>();//список регионов

private void Main\_Load(object sender, EventArgs e)//событие загрузки формы

{

listRig.Clear();//очистка списка регионов

listBox1.Items.Clear();//очистка списка в форме

DataTable dt = SQL\_query.SendQuery("SELECT \* FROM [Регионы]", "[Регионы]");//выполняем запрос и сохраняем данные в виде таблицы

for (int i = 0; i < dt.Rows.Count; i++)//бегаем циклом по количеству строк в таблице

{

Rigion rig = new Rigion((int)dt.Rows[i][0], (string)dt.Rows[i][1]);//создаем объект класса регион в кноструктор передаем параметры id и имя

listRig.Add(rig);//добавляем регион к списку регионов

listBox1.Items.Insert(i, dt.Rows[i][1]);//добавляем регион в список в форме

}

listBox1.SelectedIndex = 0;//выбираем первый элемент из списка в форме

}

private void listBox1\_SelectedIndexChanged(object sender, EventArgs e)//событие выбора элемента из списка в форме

{

int index = listBox1.SelectedIndex;//получаем текущий отпуск

if (index < 0) return;//если не выбрано индекса то не выполняем дальше код

for (int i = 0; i < listRig.Count; i++)//бегаем по списку регионов

{

if ((string)listBox1.Items[index] == listRig[i].Rig)//если выбранный регион равен региону из списка

{

dataGridView1.DataSource = SQL\_query.SendQuery("SELECT \* FROM [Показатели по годам] WHERE [Показатели по годам].[Регион] = '" + listRig[i].Id + "'", "[Показатели по годам]");//выполнчем запрос и получаем данные для дгв из БД

}

}

dataGridView1.Columns[0].Visible = false;//скрываем колонку с id

dataGridView1.Columns[7].Visible = false;//скрываем колонку с принадлежащем показателю региону

}

private void button3\_Click(object sender, EventArgs e)//кнопка удалить

{

try

{

string id = dataGridView1[0, dataGridView1.CurrentRow.Index].Value.ToString();//получаем id выбранного показателя

SQL\_query.QueryOfCng("DELETE FROM [Показатели по годам] WHERE [Показатели по годам].[Код] = " + id);//удаляем выбранный показатель из БД

listBox1\_SelectedIndexChanged(null, null);//вызываем срабатывания события выбора элемента из списка

}

catch (Exception)

{

}

}

public int GetId()//функция получения текущего id показателя

{

int index = -1;

for (int i = 0; i < listRig.Count; i++)//бегаем по списку регионов

{

if ((string)listBox1.Items[listBox1.SelectedIndex] == listRig[i].Rig)//если выбранный регион из списка совпал с требуемым регионом

{

index = listRig[i].Id;//запоминаем индекс региона

}

}

return index;//возвращаем значение из функции

}

private void GetData()//функция получения данных

{

CurrentIndicator.RigId = GetId();//записываем в переменную класса текущего показателся корректный id

try//блок ловит ошибку

{

CurrentIndicator.Year = dataGridView1[1, dataGridView1.CurrentRow.Index].Value.ToString();//записываем в переменную класса корректный год

CurrentIndicator.Avr = dataGridView1[2, dataGridView1.CurrentRow.Index].Value.ToString();//записываем в переменную класса корректную среднюю температуру года

CurrentIndicator.Vlajnost = dataGridView1[3, dataGridView1.CurrentRow.Index].Value.ToString();//записываем в переменную класса корректную среднюю влажность года

CurrentIndicator.Davlenie = dataGridView1[4, dataGridView1.CurrentRow.Index].Value.ToString();//записываем в переменную класса корректную среднее давление года

CurrentIndicator.Veter = dataGridView1[5, dataGridView1.CurrentRow.Index].Value.ToString();//записываем в переменную класса корректную среднюю силу ветра года

CurrentIndicator.Osadki = dataGridView1[6, dataGridView1.CurrentRow.Index].Value.ToString();//записываем в переменную класса корректную среднюю количества осадков года

}

catch (Exception)//если ошибка поймана то она обрабатываеся в этом блоке

{

}

}

private void button2\_Click(object sender, EventArgs e)//кнопка изменения региона

{

try//блок ловит ошибку

{

Change.changeInd = true;//флаг изменения показателя

SQL\_query.update = true;//флаг для запроса на обновление показателя

GetData();//получаем данные выбранного показателя

Change cng = new Change();//создаем форму для добавления показателя

cng.ShowDialog();//открываем форму

}

catch (Exception)//если ошибка поймана то она обрабатываеся в этом блоке

{

}

}

private void button1\_Click(object sender, EventArgs e)//кнопка добавить регион

{

try

{

Change.changeInd = true;//флаг изменения показателя

SQL\_query.add = true;//флаг для запроса на добавление показателя

GetData();//получаем данные выбранного показателя

Change cng = new Change();//создаем форму для добавления показателя

cng.ShowDialog();//открываем форму

}

catch (Exception)

{

}

}

private void Main\_Activated(object sender, EventArgs e)//событие активации формы

{

if (!Change.changeInd)//проверка флага

{

Main\_Load(null, null);//вызов события загрузки формы

}

listBox1\_SelectedIndexChanged(null, null);//вызов события выбора элемента из списка

}

private void button4\_Click(object sender, EventArgs e)//кнопка добавить регион

{

Change.changeInd = false;//флаг изменения региона

Change cng = new Change();//создаем форму для добавления показателя

cng.ShowDialog();//открываем форму

}

private void PrintGraph(int index)

{

chart1.Series[0].Points.Clear();//очищаем график

double max = -1000;//переменная хранения максимального значения

double min = 1000;//переменная хранения минимального значения

double avr = 0;//переменная хранения среднего значения

for (int i = 0; i < dataGridView1.RowCount - 1; i++)//бегаем по элементам строк в таблице

{

double ind = Convert.ToDouble(dataGridView1.Rows[i].Cells[index].Value);//получем текущую температуру

avr += ind;//прибавляем к общей для вычисления среднего значений между годами

if (ind > max) { max = ind; }//сохраняем максимальное значение

if (ind < min) { min = ind; }//сохраняем минимальное значение

double x = Convert.ToDouble(dataGridView1.Rows[i].Cells[1].Value);//год

double y = ind;//температура

chart1.Series[0].Points.AddXY(x, y);//строим график

}

avr /= dataGridView1.RowCount - 1;//вычисляем среднее значение

textBox1.Text = min.ToString();//вывод в текст бокс минимума

textBox2.Text = max.ToString();//вывод в текст бокс максимума

textBox3.Text = Math.Round(avr, 2).ToString(); //вывод в текст бокс среднего

}

private void button5\_Click(object sender, EventArgs e)//кнопка <<

{

this.Width = 480;//устанавливаем ширину экрана

}

private void информацияToolStripMenuItem\_Click(object sender, EventArgs e)//кнопка об авторе

{

MessageBox.Show("Приложение разработано студентом факультета ПМиИТ Группы ПИ18-1 Пыльцыной Анастасией в 2020 году в рамках курсовой работы.");//вызываем меседж бокс с сообщением

}

private void оПрограммеToolStripMenuItem\_Click(object sender, EventArgs e)//кнопка справка о программе

{

Process.Start("Help.txt");//запуск сторонней программы

}

private void графикToolStripMenuItem\_Click(object sender, EventArgs e)

{

PrintGraph(2);//строим график по показателю температура

}

private void влажностьToolStripMenuItem\_Click(object sender, EventArgs e)

{

PrintGraph(3);//строим график по показателю влажность

}

private void давлениеToolStripMenuItem\_Click(object sender, EventArgs e)

{

PrintGraph(4);//строим график по показателю давление

}

private void силаВетраToolStripMenuItem\_Click(object sender, EventArgs e)

{

PrintGraph(5);//строим график по показателю сила ветра

}

private void осадкиToolStripMenuItem\_Click(object sender, EventArgs e)

{

PrintGraph(6);//строим график по показателю осадки

}

private void menuStrip1\_ItemClicked(object sender, ToolStripItemClickedEventArgs e)

{

}

}

}

**Файл Change.cs:**

using System;

using System.Data;

using System.Windows.Forms;

namespace IBR

{

public partial class Change : Form

{

public Change()

{

InitializeComponent();

}

public static bool changeInd = true;

void ChangeParam()//функция меняющая форму в зависимости от выставленного флага

{

if (changeInd)//проверка если установлен флаг "изменение показателя" то настраиваемформы под показатель

{

label1.Visible = true;

textBox1.Visible = true;

label2.Text = "Температура";

label3.Visible = true;

textBox3.Visible = true;

label4.Visible = true;

textBox4.Visible = true;

label5.Visible = true;

textBox5.Visible = true;

label6.Visible = true;

textBox6.Visible = true;

}

else//иначе настрыиваем форму под регион

{

label1.Visible = false;

textBox1.Visible = false;

label3.Visible = false;

textBox3.Visible = false;

label4.Visible = false;

textBox4.Visible = false;

label5.Visible = false;

textBox5.Visible = false;

label6.Visible = false;

textBox6.Visible = false;

label2.Text = "Регион";

}

}

private void Change\_Load(object sender, EventArgs e)//загрузка формы

{

try

{

ChangeParam();

if (SQL\_query.update)//если запрос на изменение то заполним текстбоксы текущими значениями

{

DataTable dt = SQL\_query.SendQuery("SELECT \* FROM [Показатели по годам] WHERE [Показатели по годам].[Год] = " + CurrentIndicator.Year + " AND [Показатели по годам].[Температура] = '" + CurrentIndicator.Avr + "' AND [Показатели по годам].[Регион] = '" + CurrentIndicator.RigId + "'", "[Показатели по годам]");

//DataTable dt = SQL\_query.SendQuery("SELECT \* FROM [Показатели по годам] WHERE [Температура] = '" + CurrentIndicator.Avr + "'", "[Показатели по годам]");

textBox1.Text = Convert.ToString(dt.Rows[0][1]);

textBox2.Text = Convert.ToString(dt.Rows[0][2]);

textBox3.Text = Convert.ToString(dt.Rows[0][3]);

textBox4.Text = Convert.ToString(dt.Rows[0][4]);

textBox5.Text = Convert.ToString(dt.Rows[0][5]);

textBox6.Text = Convert.ToString(dt.Rows[0][6]);

}

else if (SQL\_query.add)

{

}

}

catch (Exception)

{

}

}

private void button1\_Click(object sender, EventArgs e)//кнопка сохранить

{

try

{

if (changeInd)//если меняем показатель

{

if (SQL\_query.update)//если запрос на изменение

{

SQL\_query.QueryOfCng("UPDATE [Показатели по годам] SET " +

"[Показатели по годам].[Год] = " + textBox1.Text +

", [Показатели по годам].[Температура] = " + textBox2.Text +

", [Показатели по годам].[Влажность] = " + textBox3.Text +

", [Показатели по годам].[Давление] = " + textBox4.Text +

", [Показатели по годам].[Сила ветра] = " + textBox5.Text +

", [Показатели по годам].[Осадки] = " + textBox6.Text +

" WHERE [Показатели по годам].[Год] = " + CurrentIndicator.Year +

" AND [Показатели по годам].[Температура] = '" + CurrentIndicator.Avr +

"' AND [Показатели по годам].[Регион] = '" + CurrentIndicator.RigId + "'");

SQL\_query.update = false;

}

else if (SQL\_query.add)//если запрос на добавление нового показателя

{

SQL\_query.QueryOfCng("INSERT INTO [Показатели по годам] ([Год], [Температура], [Влажность], [Давление], [Сила ветра], [Осадки], [Регион]) VALUES ('" +

textBox1.Text + "', '" +

textBox2.Text + "', '" +

textBox3.Text + "', '" +

textBox4.Text + "', '" +

textBox5.Text + "', '" +

textBox6.Text + "', '" +

CurrentIndicator.RigId + "')");

SQL\_query.add = false;

}

}

else//иначе добавляем регион

{

SQL\_query.QueryOfCng("INSERT INTO Регионы (Регион) VALUES ('" + textBox2.Text + "')");

}

}

catch (Exception)

{

}

Close();//закрываем форму

}

private void textBox1\_KeyPress(object sender, KeyPressEventArgs e)//событие ввода символа

{

if (!Char.IsDigit(e.KeyChar) && e.KeyChar != Convert.ToChar(8))//ввод только цифр

{

e.Handled = true;

}

}

private void Change\_FormClosing(object sender, FormClosingEventArgs e)

{

SQL\_query.update = false;

SQL\_query.add = false;

}

private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)//событие ввода символа

{

//if (changeInd)

//{

// if (!Char.IsDigit(e.KeyChar) && e.KeyChar != Convert.ToChar(8))//ввод только цифр

// {

// e.Handled = true;

// }

//}

}

}

}