

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

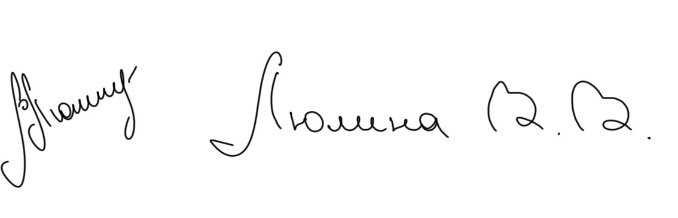
**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**САМАРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

Институт автоматики и информационных технологий  
Межвузовская кафедра информационных и развивающих образовательных систем и технологий

**КУРСОВАЯ РАБОТА**по дисциплине *«Проектирование интерфейсов»   
«Проектирование интерфейса справочной системы фильмотеки»*

Выполнила: студентка 3 курса,   
гр. 3– ИАИТ – ГИП – 118А   
Люлина Валерия Вадимовна  
Подпись обучающегося:



Руководитель работы:   
доцент Козлов В.В.  
Оценка, подпись:

Самара, 2021 г.

**Оглавление**

[**Введение** 3](#_Toc74994063)

[**Инструменты реализации** 4](#_Toc74994064)

[**Язык** 4](#_Toc74994065)

[**Среда разработки** 4](#_Toc74994066)

[**База данных** 5](#_Toc74994067)

[**Реализация** 7](#_Toc74994068)

[**Форма Start** 7](#_Toc74994069)

[**Класс SQLiteHelper** 7](#_Toc74994070)

[**Форма Film** 8](#_Toc74994071)

[**Форма Genre** 15](#_Toc74994072)

[**Описание базы данных** 16](#_Toc74994073)

[**Список рисунков** 18](#_Toc74994074)

[**Заключение** 19](#_Toc74994075)

[**Список источников** 20](#_Toc74994076)

[**Листинг** 21](#_Toc74994077)

# **Введение**

При решении поставленной задачи проектировании интерфейса справочной системы фильмотеки в виде desktop приложения используется язык C# на платформе .NET с использованием пользовательского интерфейса для создания классических приложений Windows Forms

Для реализации проекта, состоящего из нескольких файлов, была использована интегрированная среда разработки Visual Studio Community из линейки продуктов компании Microsoft.

Актуальность: в современном мире существует большое количество фильмов разных жанров, возрастов и интересов. Для того чтобы можно было не потеряться в многообразии существующего выбора необходим справочник с возможностью поиска конкретного фильма по названию или сборника по определенному критерию, например жанр.

Постановка задачи: необходимо спроектировать законченный программный продукт с графическим интерфейсом, включающей в себя работу с базами данных, с графикой (или построением графиков) и формированием отчета в формате PDF.

Цели разработки программного продукта:

1. создать базу данных, в которой хранятся сведения о фильмах, жанрах, актерах, режиссерах и композиторах
2. обеспечить возможность просмотра данных
3. обеспечить возможность поиска и фильтрации данных;
4. обеспечить возможность сохранения документа в PDF
5. обеспечить возможность создания диаграммы для наглядного просмотра соотношения количества фильмов к жанру с сохранением документа в формате .png

# **Инструменты реализации**

## **Язык**

На сегодняшний день язык программирования C# один из самых мощных, быстро развивающихся и востребованных языков в IT-отрасли. В настоящий момент на нем пишут приложения от простых калькуляторов до крупных веб-порталов, обслуживающих ежедневно миллионы пользователей.

C# является объектно-ориентированным и в этом плане много перенял у Java и С++. Так, C# поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. Объектно-ориентированный подход дает возможность решить задачи по построению крупных, но в тоже время гибких и расширяемых приложений.

Когда говорят о C#, нередко имеют в виду технологии платформы .NET. Также, когда говорят о .NET, имеют в виду C#. Данный язык был создан специально для работы с этим фреймворком, однако само его понятие несколько шире. Можно выделить следующие основные черты:

* **Кроссплатформенность**. .NET является переносимой платформой (с некоторыми ограничениями). Использованием различных технологий на платформе .NET, позволяет разрабатывать на языке C# для самых различных платформ от Windows и Linux до Android и iOS.
* **Мощная библиотека классов**. При написании на C# текстового редактора, чата или сложного веб-сайта так или иначе используется библиотека классов .NET.
* **Разнообразие технологий**. Общеязыковая среда исполнения CLR и базовая библиотека классов являются основой для целого стека технологий, которые разработчики могут задействовать при построении тех или иных приложений. Например, для работы с базами данных в этом стеке технологий предназначена технология ADO.NET и Entity Framework Core. Для построения графических приложений с богатым насыщенным интерфейсом - технология WPF и UWP, для создания более простых графических приложений - Windows Forms.

Также еще следует отметить такую особенность языка C# и фреймворка .NET, как автоматическая сборка мусора. А это значит, что в большинстве случаев разработчику не придется, в отличие от С++, заботиться об освобождении памяти. Вышеупомянутая общеязыковая среда CLR сама вызовет сборщик мусора и очистит память.

## **Среда разработки**

C# — популярный язык программирования. Он является основным инструментом для разработки программного обеспечения компании Microsoft. Этот объектно-ориентированный язык применяется для создания веб-приложений и десктопных программ. Существует несколько IDE для разработки программных продуктов, один из которых Visual Studio, которая и была выбрана для работы. Помимо нее существуют следующие среды разработки:

* Eclipse
* Visual Studio Code

Visual Studio признана лучшей IDE для C#. Дело в том, что оба продукта принадлежат корпорации Microsoft. Поэтому они идеально подходят для работы друг с другом.

В число преимуществ Visual Studio входит следующее:

* Среда содержит множество инструментов, которые очень хорошо работают на C#.
* Наличие бесплатной версии — Community Edition.
* Community содержит все что нужно для независимого разработчика.
* Самое эффективное ПО для разработки на любой платформе, включая .Net и C#.
* Возможность хранения данных в облаке.

У продукта есть и некоторые недостатки:

* Требовательность к ресурсам.
* После перехода на платную версии, могут слететь настройки и корпоративный сервер.
* Сложность при самостоятельном освоении.

Eclipse aCute — плагин для Eclipse IDE. Он облегчает разработку на C#. aCute дает возможность применять редактор C#, в состав которого входит Eclipse IDE, поддерживающий языки посредством сервера Omni-sharp. В отличие от VS, плагин разработан сообществом. Следовательно, нет никаких гарантий, что он всегда будет работать качественно.

Visual Studio Code — отличный высокопроизводительный легковесный редактор, предлагающий хорошую поддержку завершения проекта. Работает на Node JS. Содержит плагины для VIM и Emacs. Он отлично работает на Mac, Linux, Windows, однако Обладает небольшим функционалом, поэтому не подходит для реализации больших проектов.

Интегрированная среда разработки Visual Studio — это стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений. Интегрированная среда разработки (IDE) представляет собой многофункциональную программу, которую можно использовать для различных аспектов разработки программного обеспечения. Помимо стандартного редактора и отладчика, которые существуют в большинстве сред IDE, Visual Studio включает в себя компиляторы, средства автозавершения кода, графические конструкторы и многие другие функции для упрощения процесса разработки и именно поэтому она была выбрана как IDE для разработки законченного программного продукта

## **База данных**

База данных — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. Существует некоторое множество различный реляционных БД(их также называю СУБД), например:

* MySQL
* MS SQL
* SQLite
* PostgreSQL и др.

Несмотря на то, что все СУБД выполняют одну и ту же основную задачу (т.е. дают возможность пользователям создавать, изменять и получать доступ к информации, находящейся в базах данных), сам процесс выполнения этой задачи варьируется в широких пределах. Кроме того, функции и возможности каждой СУБД могут существенно отличаться. Различные СУБД документированы по-разному: более или менее тщательно. По-разному предоставляется и техническая поддержка.

MySQL - одна из самых популярных баз данных для веб-приложений. В подавляющем большинстве является стандартом для веб-серверов, которые работают под управлением операционной системы Linux. MySQL - это бесплатный пакет программ, однако новые версии выходят постоянно, расширяя функционал и улучшая безопасность. Существуют специальные платные версии, предназначенные для коммерческого использования. В бесплатной версии наибольший упор делается на скорость и надежность, а не на полноту функционала, который может стать и достоинством, и недостатком - в зависимости от области внедрения.

Ещё одной из популярных СУБД является программный продукт Microsoft SQL-сервер (MS SQL). Это СУБД, движок которой работает и на облачных серверах и на локальных серверах, причем можно комбинировать типы применяемых серверов одновременно. Вскоре после выпуска Microsoft SQL сервер 2016, Microsoft адаптировала продукт для операционной системы Linux, а на Windows-платформе он работал изначально.

PostgreSQL является одним из нескольких бесплатных популярных вариантов СУБД, также как и MySQLчасто используется для ведения баз данных веб-сайтов. Это была одна из первых разработанных систем управления базами данных, поэтому в настоящее время она хорошо развита, и позволяет пользователям управлять как структурированными, так и неструктурированными данными. Может быть использован на большинстве основных платформ, включая Linux.

И, наконец, SQLite — самая распространенная СУБД в мире, включена во все популярные ОС и является компактной и встраиваемой. Работает без сервера. Для разработчиков — встраивается прямо в приложение. Простота и удобство встраивания SQLite привели к тому, что библиотека используется в браузерах, музыкальных плеерах и многих других программах. Для разработки законченного продукта хватит данной простой СУБД, которая по своей сути – это файл, подключаемый к программе, что очень удобно.

Для создания файла базы данных была выбрана программа SQLite Studio

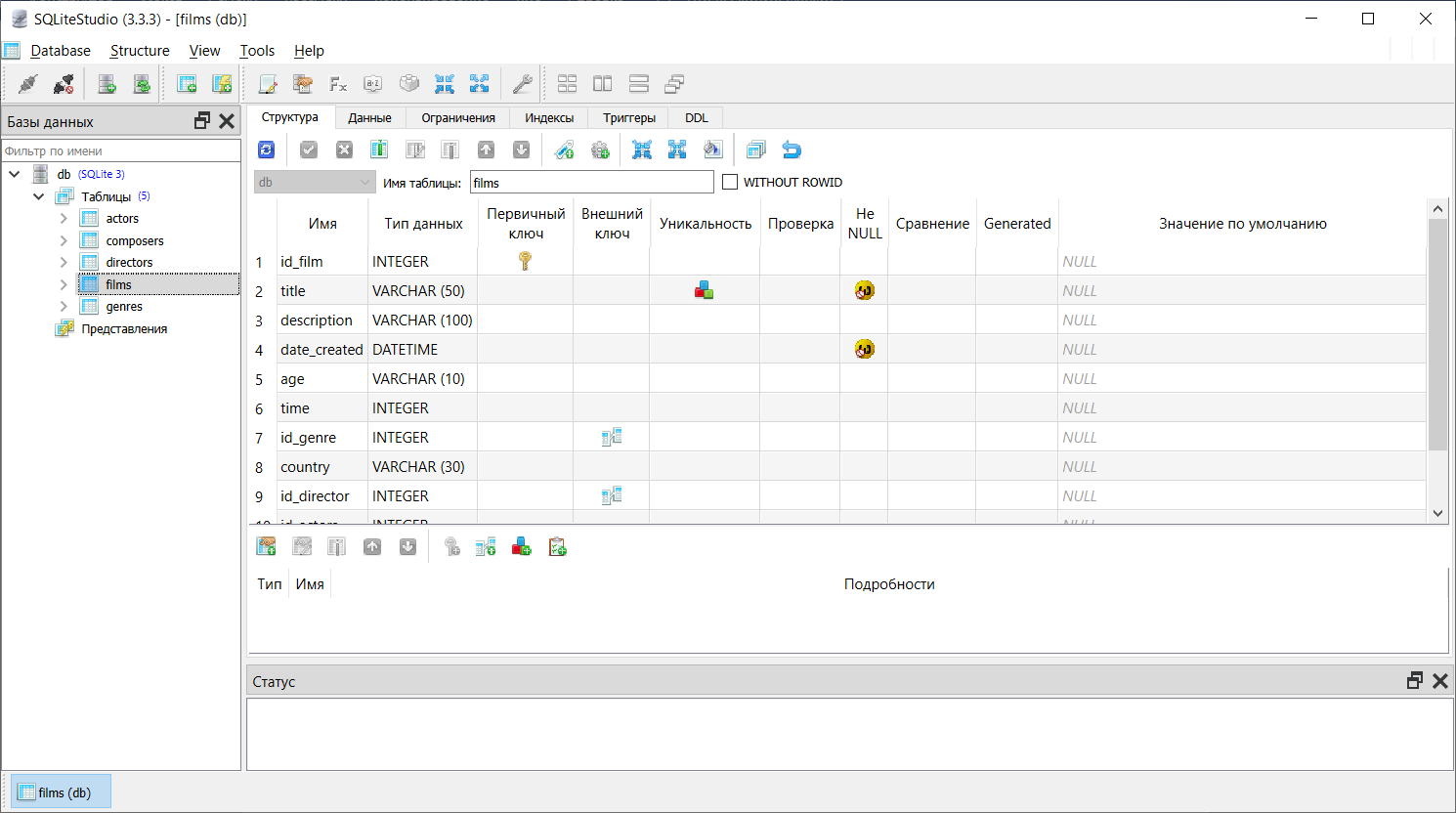


Рисунок 1 SQLite Studio

# **Реализация**

## **Форма Start**

На главной форме было размещено три кнопки: Фильмы, Прочие данные и Выход.

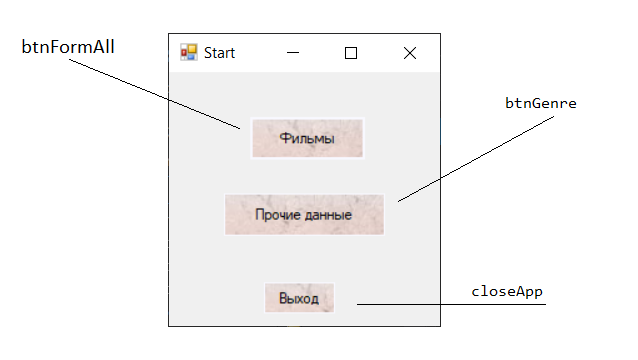


Рисунок 2 Форма №1 Start

С помощью вкладки «Свойства» были заданы рамки, фон и стиль кнопок.

Первая кнопка btnFormAll\_Click при нажатии переводит на другую форму Film, сворачивая(скрывая) родительскую форму c помощью метода Hide().

Вторая кнопка по такому же принципу открывает форму Genre, скрывая Start.

Третья кнопка закрывает приложение методом Application.Exit();

## **Класс SQLiteHelper**

Прежде чем приступить к описанию формы с базой данной необходимо описать как именно были получены данные.

Для того, чтобы данные были выведены на экран необходимо добавить элемент управления DataGridView. При создании запрашивается источник данных. Из списка элементов выносится компонент BindingSource предназначенный для упрощения процесса привязки элементов управления к источнику данных.

В классе по одному типу находятся пять методов, отвечающих за чтение для каждой DataGridView. Отличаются они лишь типом и наполнением запроса SELECT.

Метод начинается с оператора try-catch, с помощью которого можно легко выловить ошибки. В блоке try находятся три конструкции using. Первая создает подключение SQLiteConnection с параметром «Data Source = db.sqlite;Vesion=3» и открывается соединение методом Open(). Для того чтобы пользоваться возможностями SQLite необходимо в NuGet установить дополнительную библиотеку System.Data.SQLite. Автоматически установятся следующие файлы:

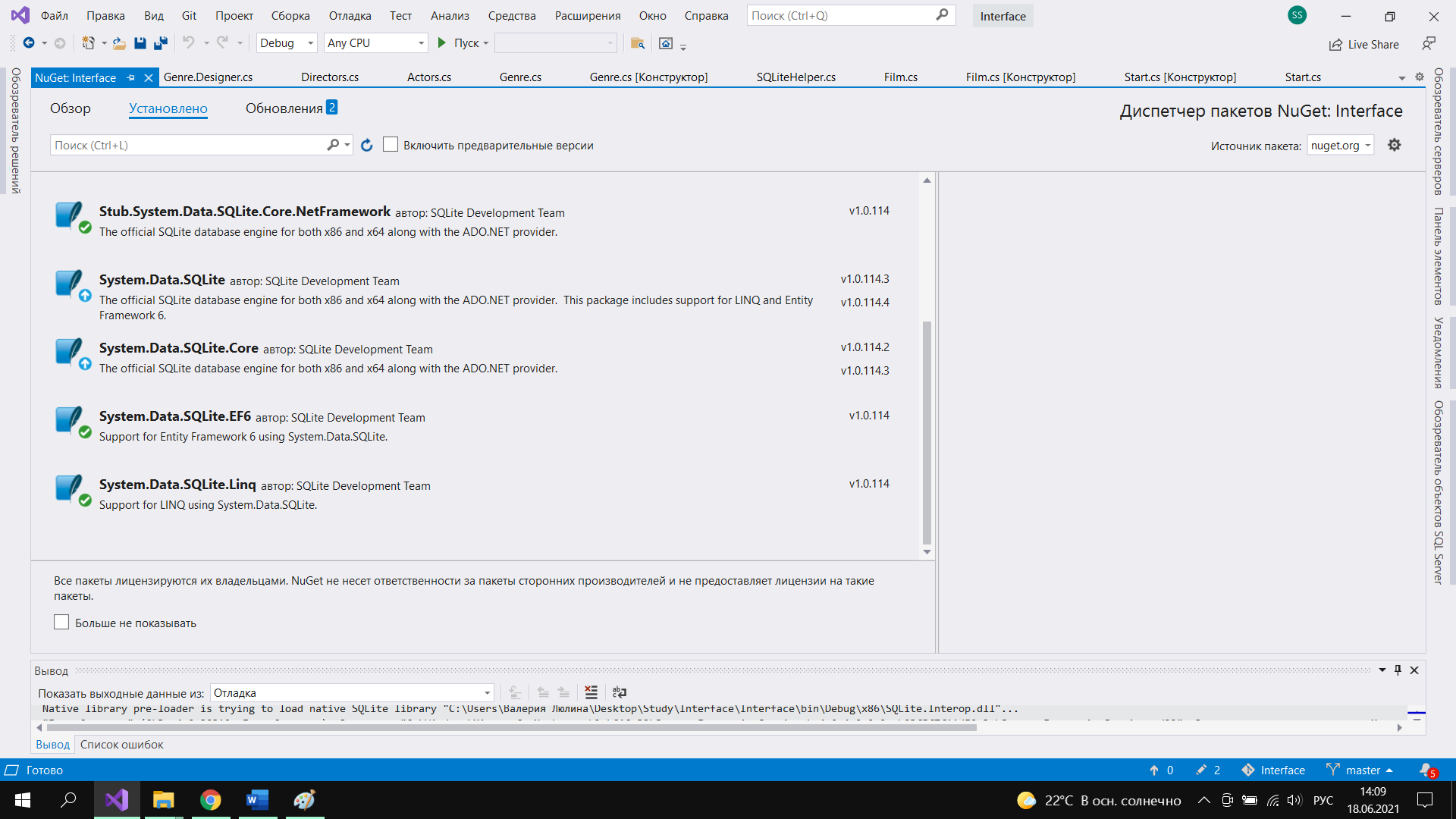


Рисунок 3 Библиотеки для работы с SQLite

Во-второй конструкции прописывается команда SELECT с помощью SQLiteCommand, показывающая какие именно данные и откуда брать, например:

using (var cmd = new SQLiteCommand(@"SELECT Name\_G FROM genres ", connection))

В третьей конструкции используется метод ExecuteReader:

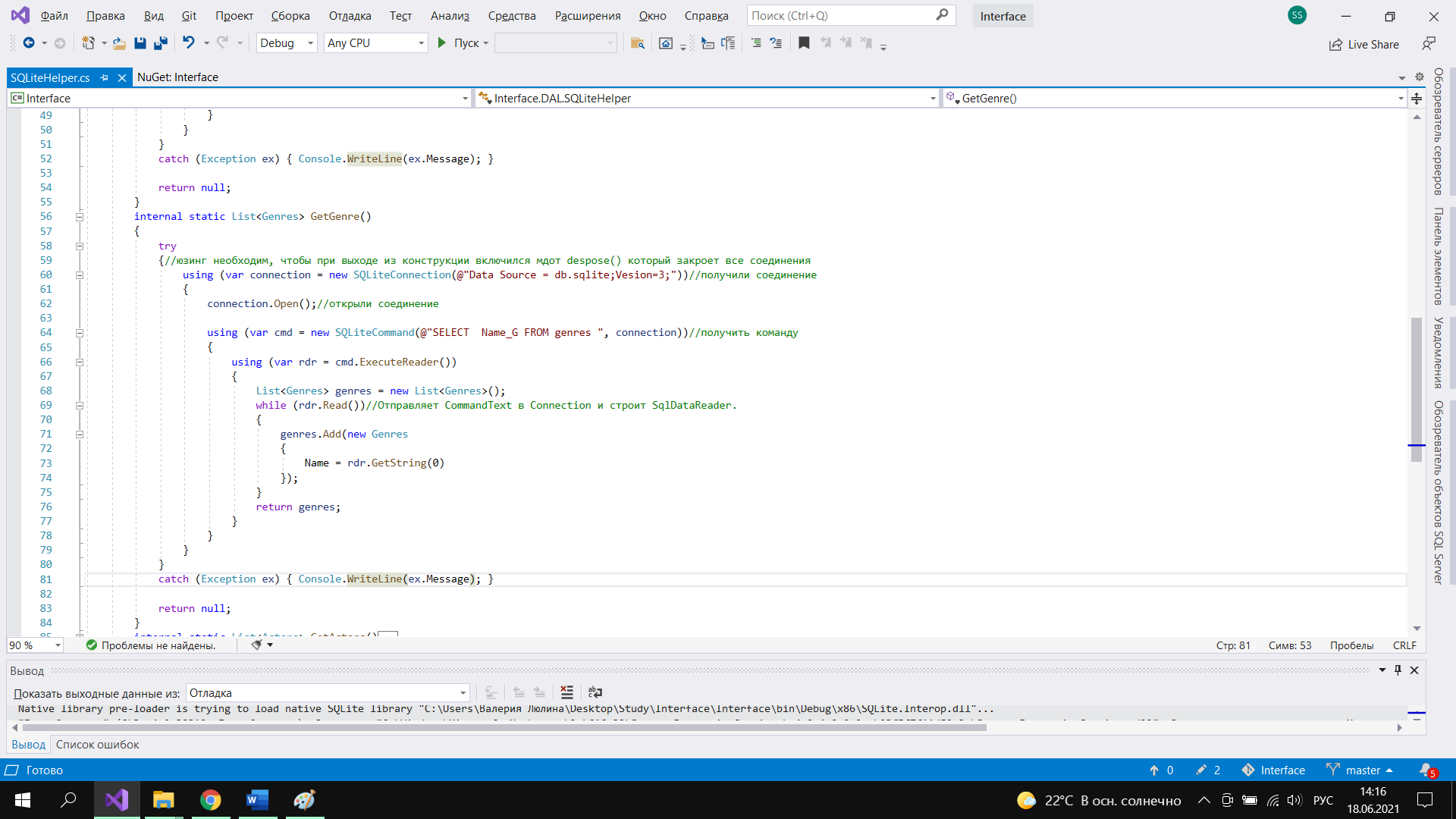


Рисунок 4 Код для возвращения столбцов из БД

В представленном примере создается List типа Genres, где первое это динамический список, а второе класс, хранящий в себе public string столбцов, которые читаются с помощью метода Read(), то есть считывания с переносом строки. В самом цикле метод Add() добавляет данные в список и возвращает это все в локальной переменной genres.

Если что-то пойдет не так весь метод вернет Null.

## **Форма Film**

На данной форме имеется две вкладки. На первой четыре кнопки, одна строка поиска и один впадающий список.

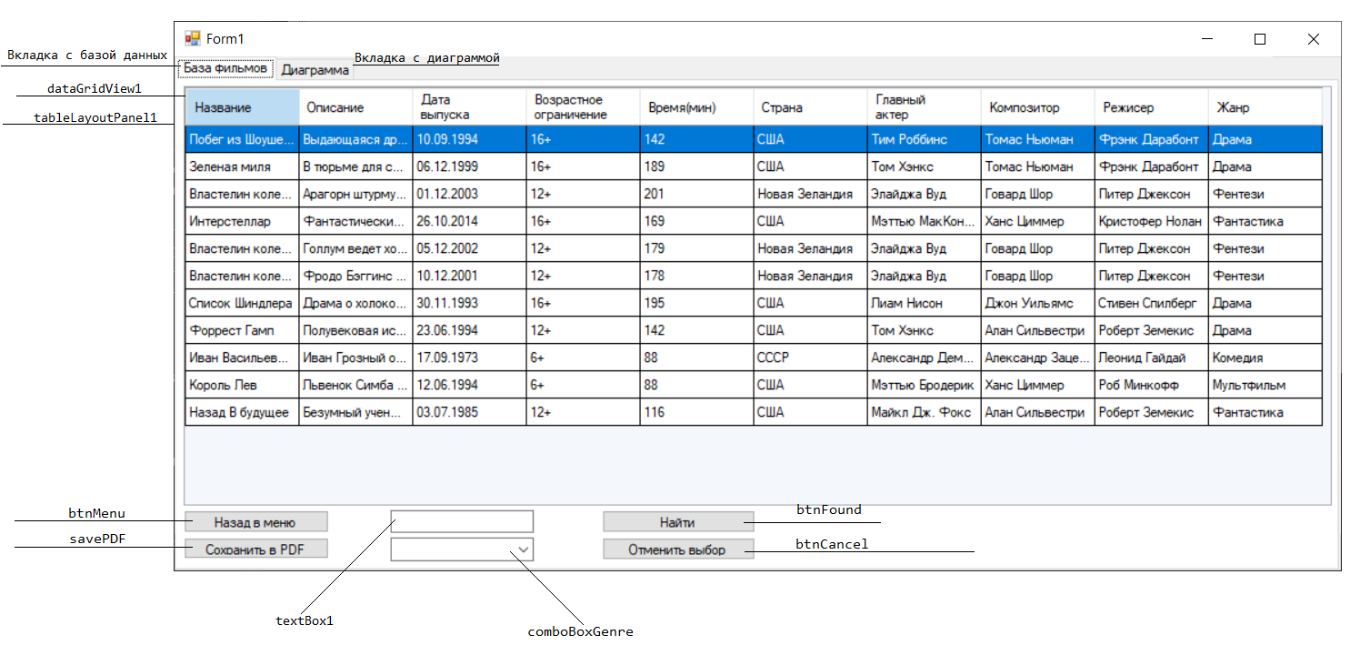


Рисунок 5 Форма Film первая вкладка

Код формы начинается с ее инициализации. Создается поле \_list и ему присваивается BindingSourse c методом DataSource, который возвращает источник данных. Создается переменная list, которую вызывается метод GetFilms() и записывается список, полученный в классе SQLiteHelper. В условии if прописывается, что если list не null и количество строк в нем больше нуля, то в конец методом AddRange() добавятся эти строки. Так как BindingSourse связан в конструкторе с DataGridView компилятор понимает куда именно записывать считанные строки.

Строка dataGridView1.AutoGenerateColumns = true; необходима для того, чтобы столбцы, прописанные в классе Films сами появились.

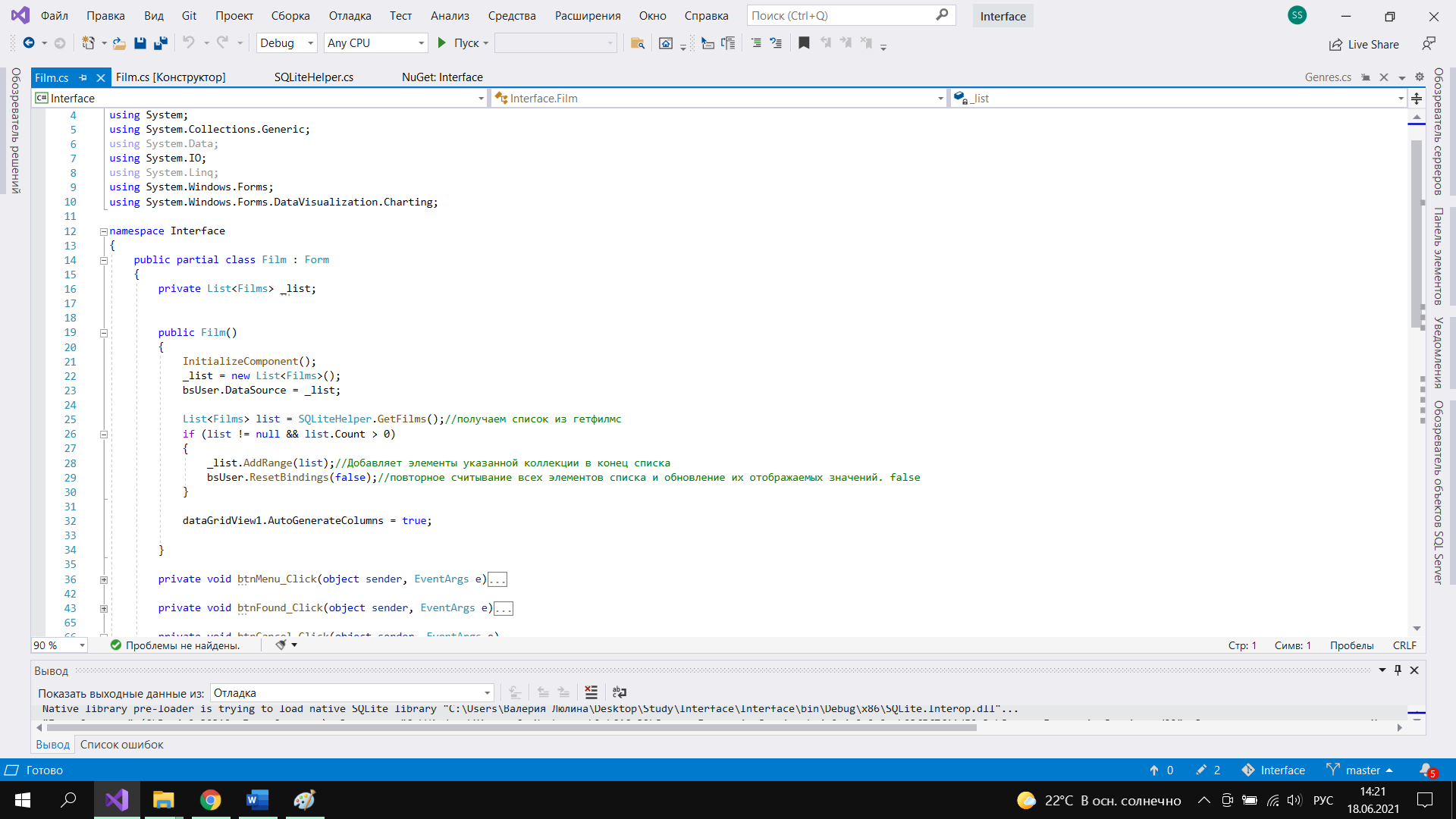


Рисунок 6 Добавление строк на DataGridView

Теперь, когда получены данные с ними можно работать.

Сама простая кнопка на данной форме – btnMenu. Она методом Close() закрывает форму Film и с помощью Visible() делает родительскую форму, ранее скрытую при переходе, видимой.

Кнопка savePDF сохраняет данные из DataGridView в PDF файл. Для того, чтобы все работало в первую очередь необходимо вновь обратиться к NuGet и установить библиотеку классов для генерации, анализа и изменения документов в форматах Portable Document Format, а также XML, HTML и PDF - ITextSharp.

По двойному клику открывается метод с котором в первую очередь необходимо прописать какой брать шрифт. Так как база данных состоит из русских символов подойдет Arial, а константа IDENTITY\_H сообщит компилятору о том, что необходимо видеть русские) и прочие) символы кириллицы.

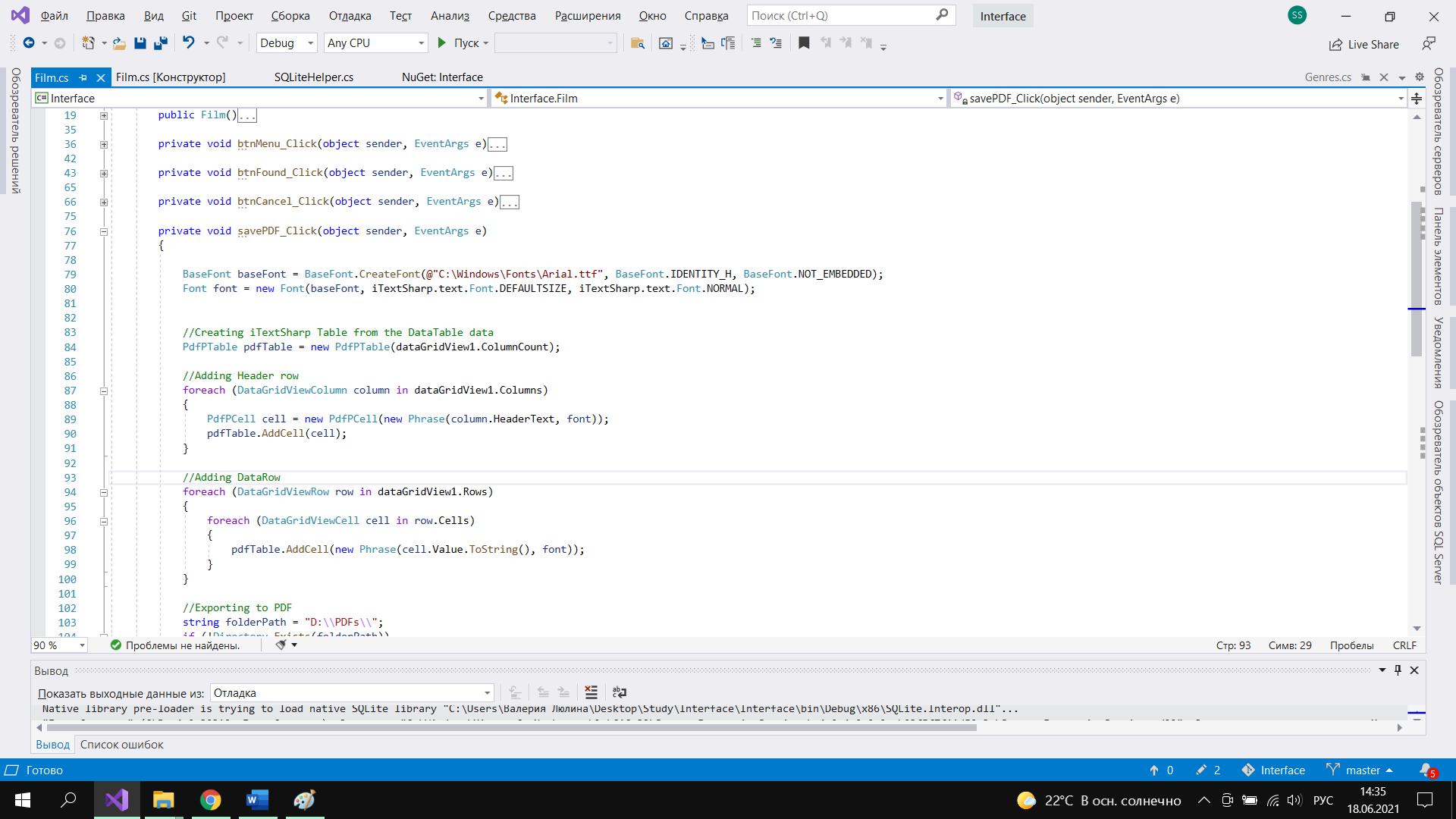


Рисунок 7 Выбор шрифта

Необходимо создать переменную типа PdfPTable в которую будет записываться происходящее. Двумя циклами foreach заполняются сначала названия столбцов, а затем непосредственно строки. Необходимо обязательно указать ранее созданную переменную font, чтобы читались не только цифры.

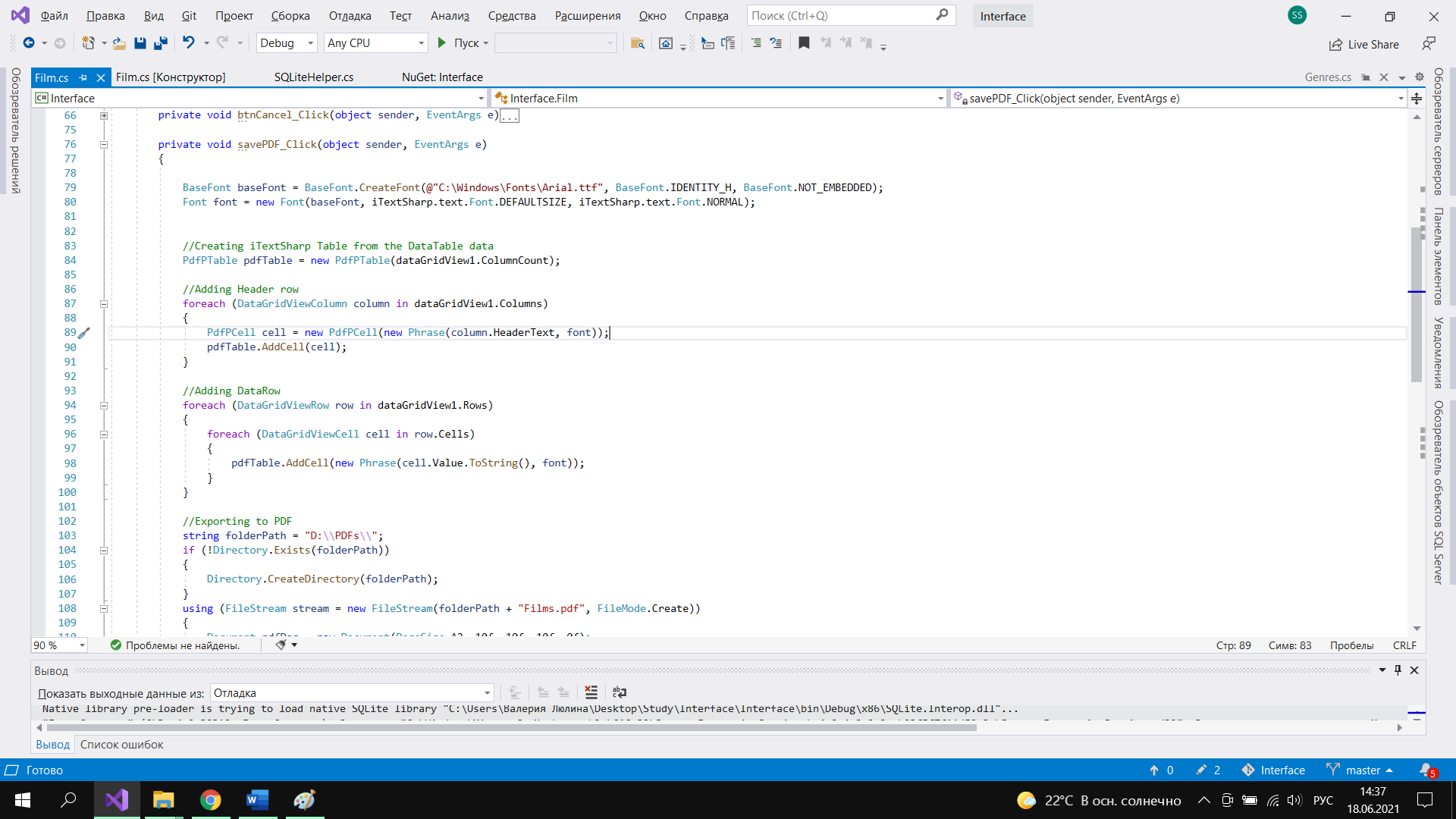


Рисунок 8 Заполнение таблицы в PDF

Далее определяется путь куда сохранится файл. При условии, что папка PDF не существовала ранее она будет создана.

Затем снова используется конструкция using, удобно закрывающее соединение и уже в ней открывается поток stream на заполнение. Если все пройдет правильно, то появится сообщение: «PDF-документ сохранен».

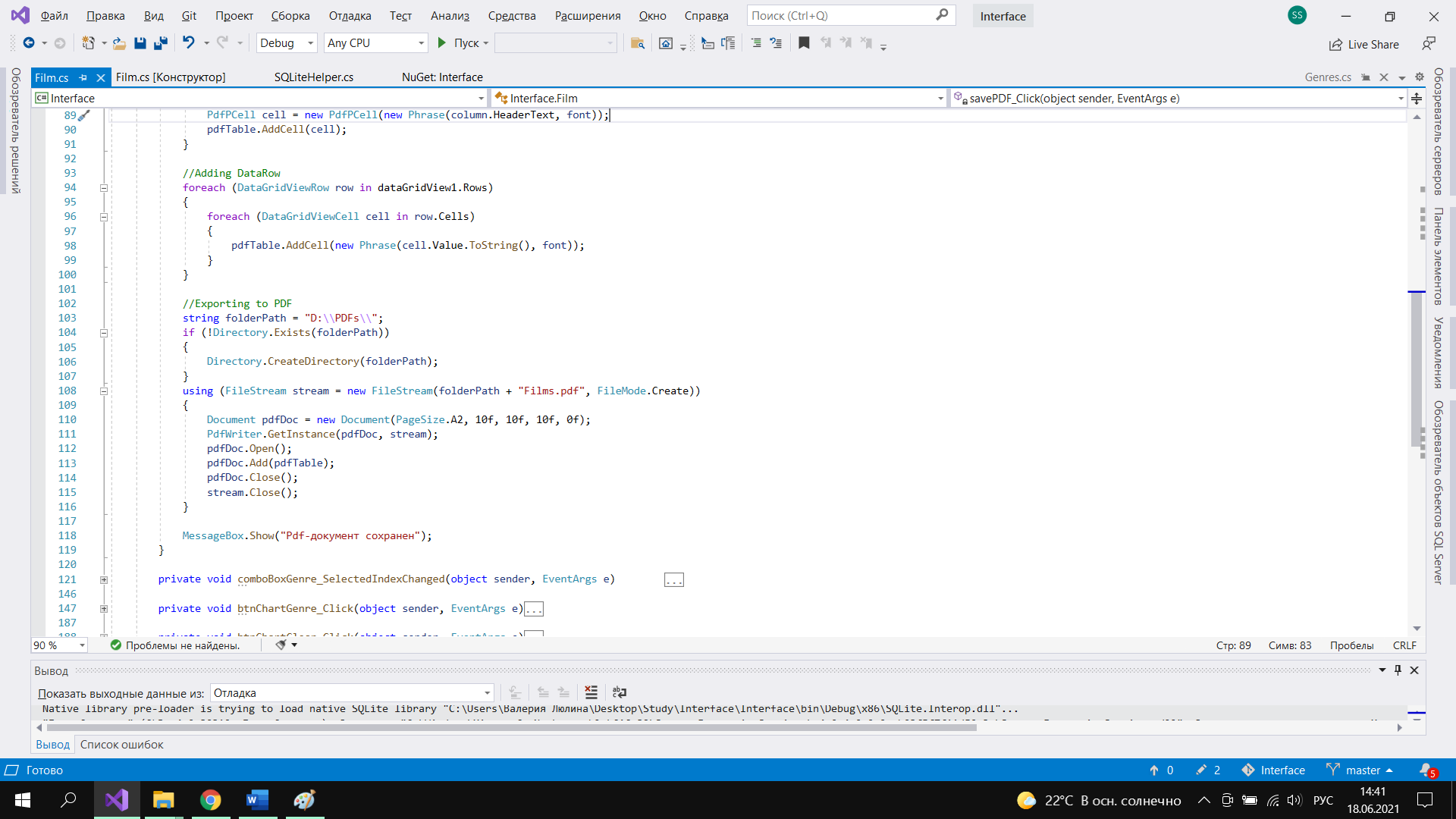


Рисунок 9 Поток и путь к папке PDF

Результат:

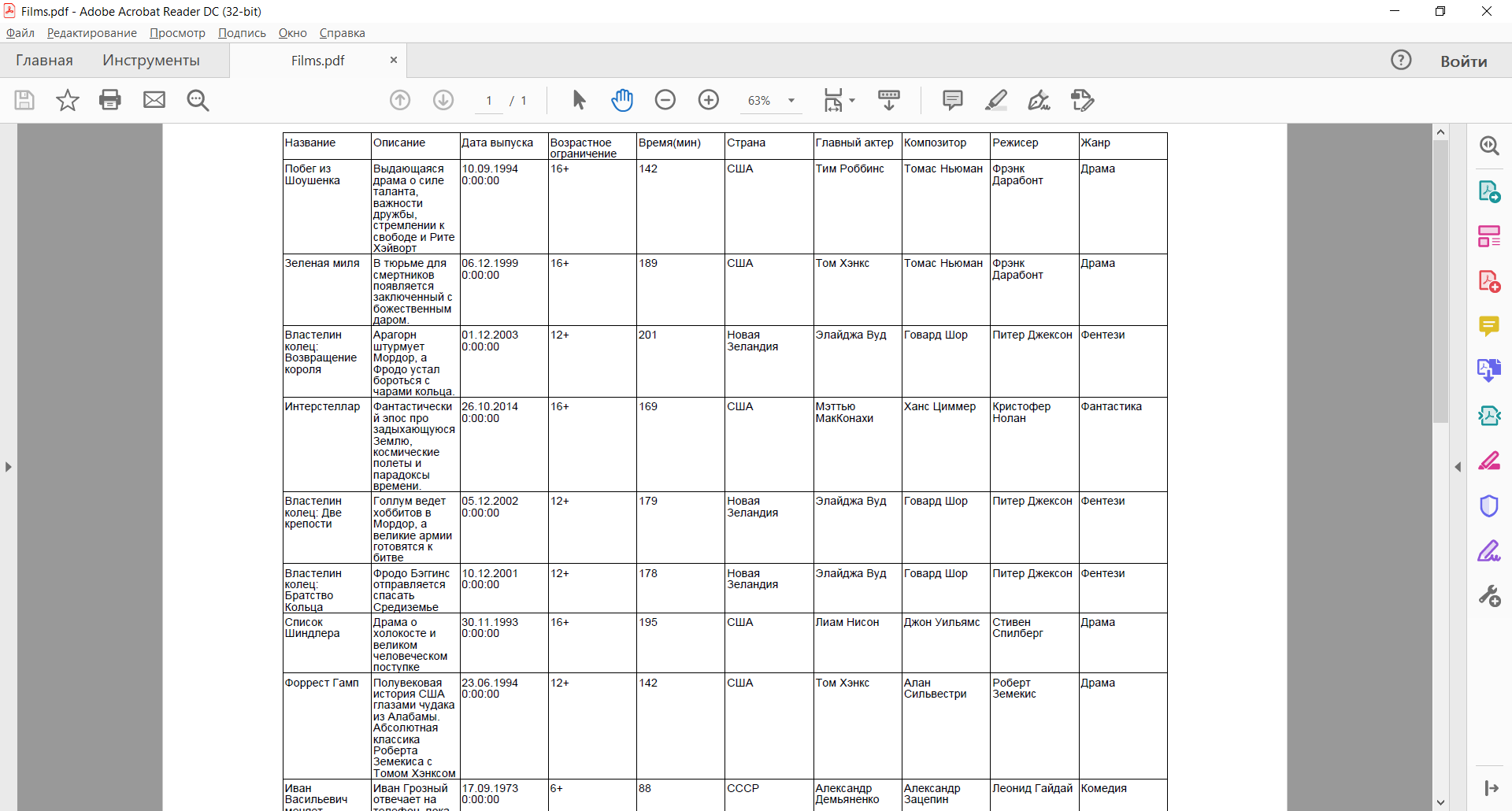


Рисунок 10 Результат PDF

Следующая для рассмотрения кнопка: btnFound. Она непосредственно связана с textBox1 и суть ее работы заключается в поиске совпадений введенных в эту строку и скрытие тех, где совпадения не найдены. Вот что будет, если ввести «США»:

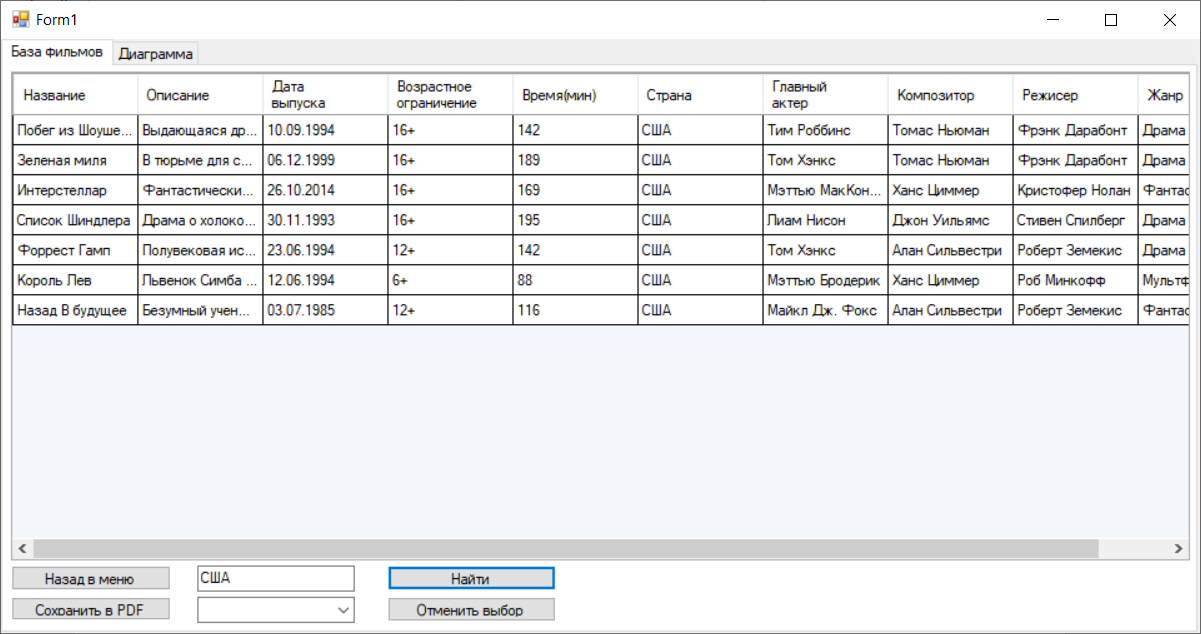


Рисунок 11 Работа кнопки btnFound

В коде это прописывается так: сначала компилятор идет по i-й строке и заходит в j-й столбец. Если строки не пустые, то открывается второе условия, где ищутся совпадения по textBox1, причем такие, чтобы регистр не был важен. Если такие строки найдены, метод Visible() их покажет, если нет, то компилятор зайдет в else, где этот же метод с другим булевым выражением скроет ненужные строки.

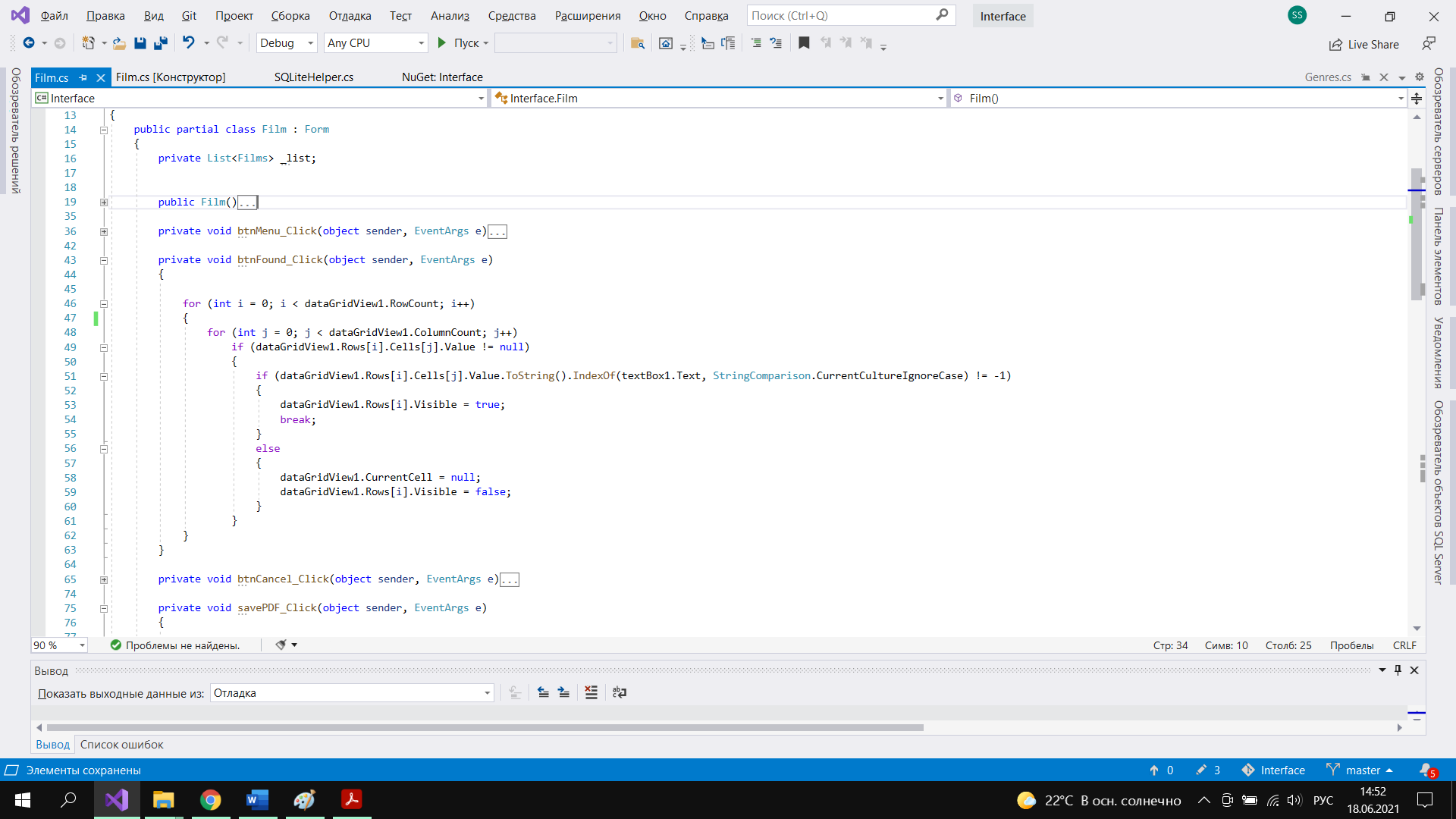


Рисунок 12 Код поиска

Для того, чтобы вернуть все как было необходимо нажать кнопку btnCancel. Внутри обработчика этой кнопки будет лишь один цикл проходящий по строкам, который методом Visible () просто вернет все, что было скрыто.

Последний управляющий элемент на данной вкладке это comboBox Genre. В него еще на уровне конструктора записываются элементы, которые можно выбрать, а потом он будет работать также, как и btnFound, только вместо textBox1.Text будет comboBoxGenre, а вместо IndexOf будет метод Contains(потому что нет необходимости в заранее вписанных словах сравнивать регистр). Пример:

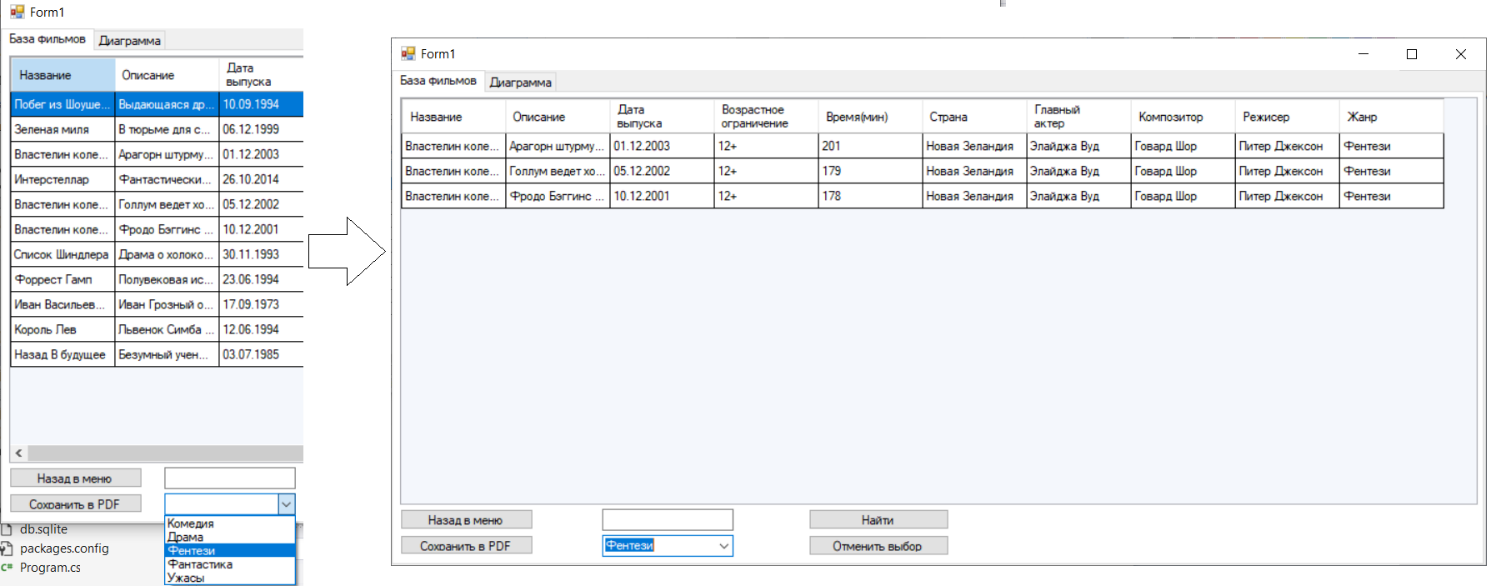


Рисунок 13 Работа comboBox

Вернуть все как было можно той же кнопкой «Отменить выбор».

Во второй вкладке находится элемент chart отвечающий за формирование графиков, а также 3 кнопки: Жанры, сохранить в PNG и очистить.



Рисунок 14 Вторая вкладка формы Films

При нажатии на кнопку «Жанры» в первую очередь происходит обращение к методу GetGenre. Это необходимо для того, чтобы посчитать количество жанров и записать это число в переменную countRows. Затем создается три массива, один числовой и два строчный. В первом цикле строки из list(то есть GetGenre) записываются в массив строк arr[]. Списком list нельзя пользоваться потому как нельзя явно конвертировать его тип в string, зато с помощью строки, находящейся в классе Genres:

public static implicit operator string(Genres v) => v.Name;

можно завернуть массив.

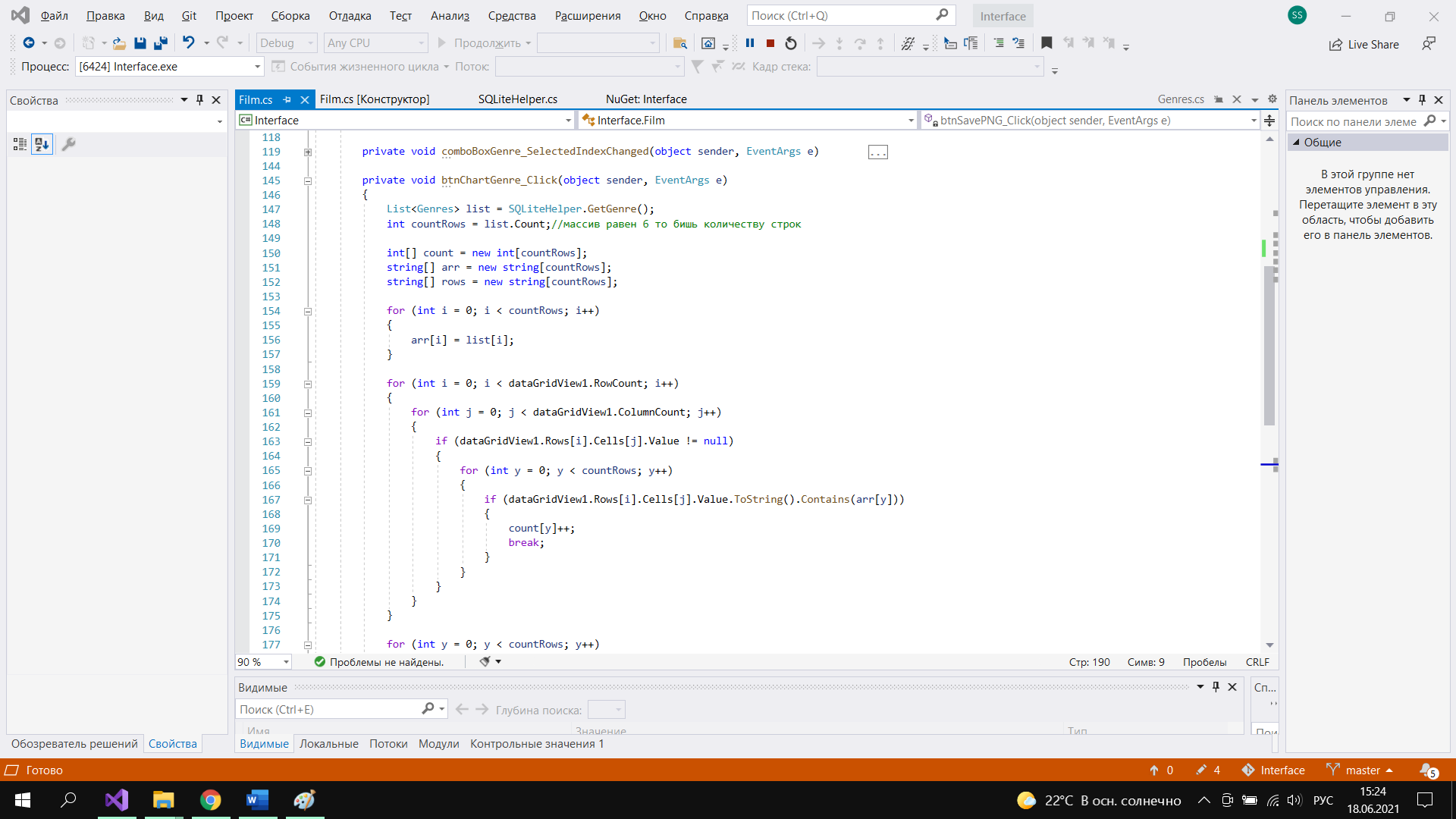


Рисунок 15 Перезапись жанров в массив

После того как массив arr получил список жанров необходимо обратиться к конструкции, которая ранее использовалась в btnFound и comboBoxGenre. Разница в том, что в конструкцию добавляется третий цикл, чтобы пройтись по каждому названию жанра arr[y]. При этом в последнем условии включен «счетчик» count[y] который записывает в массив число, соответствующее количеству совпадений.

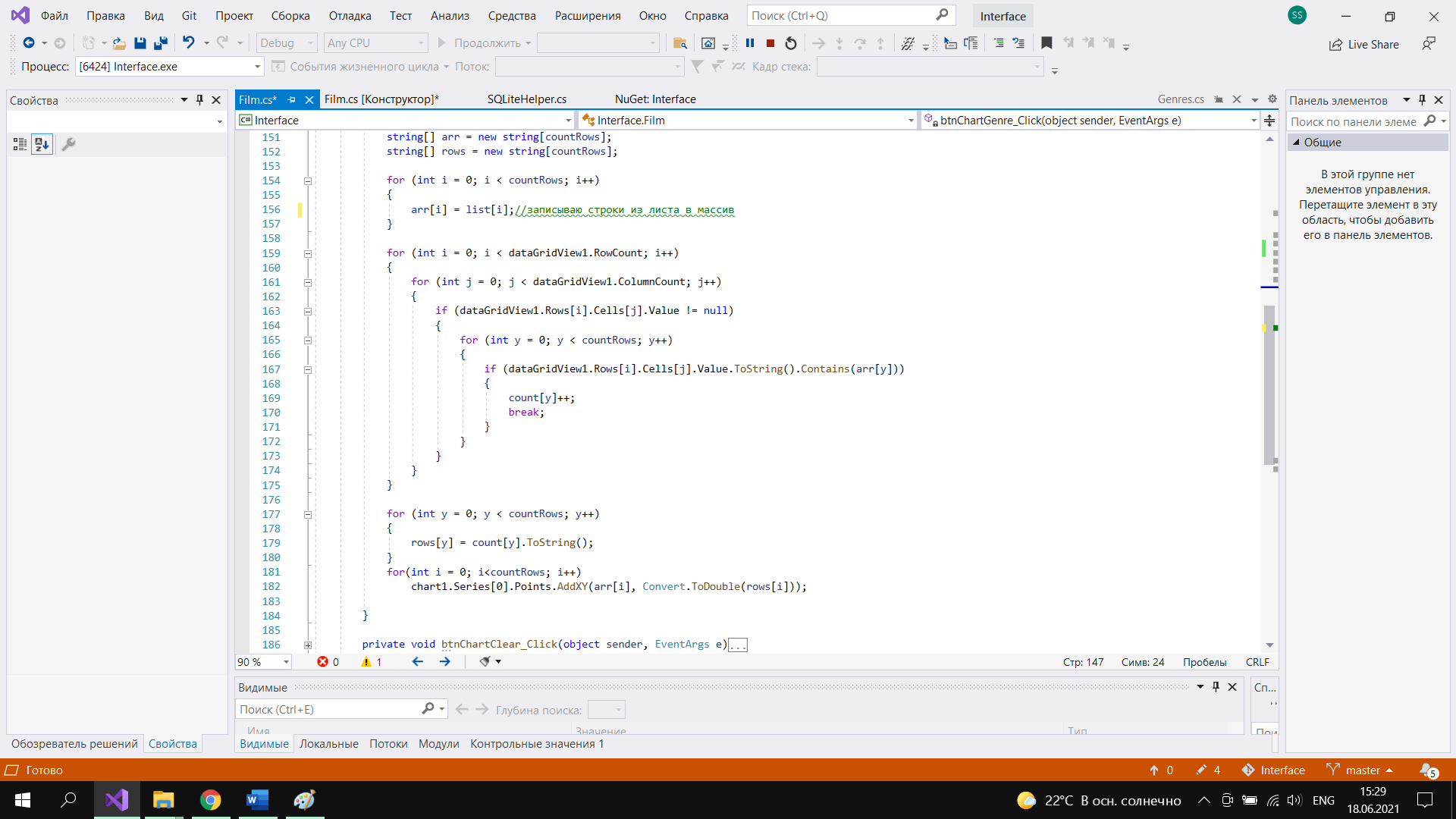


Рисунок 16 Поиск совпадение по массиву

В предпоследнем цикле данного обработчика событий происходит конвертация массива чисел в строки. Это необходимо для отображения диаграммы. В последнем цикле количество жанров передается виде точки.

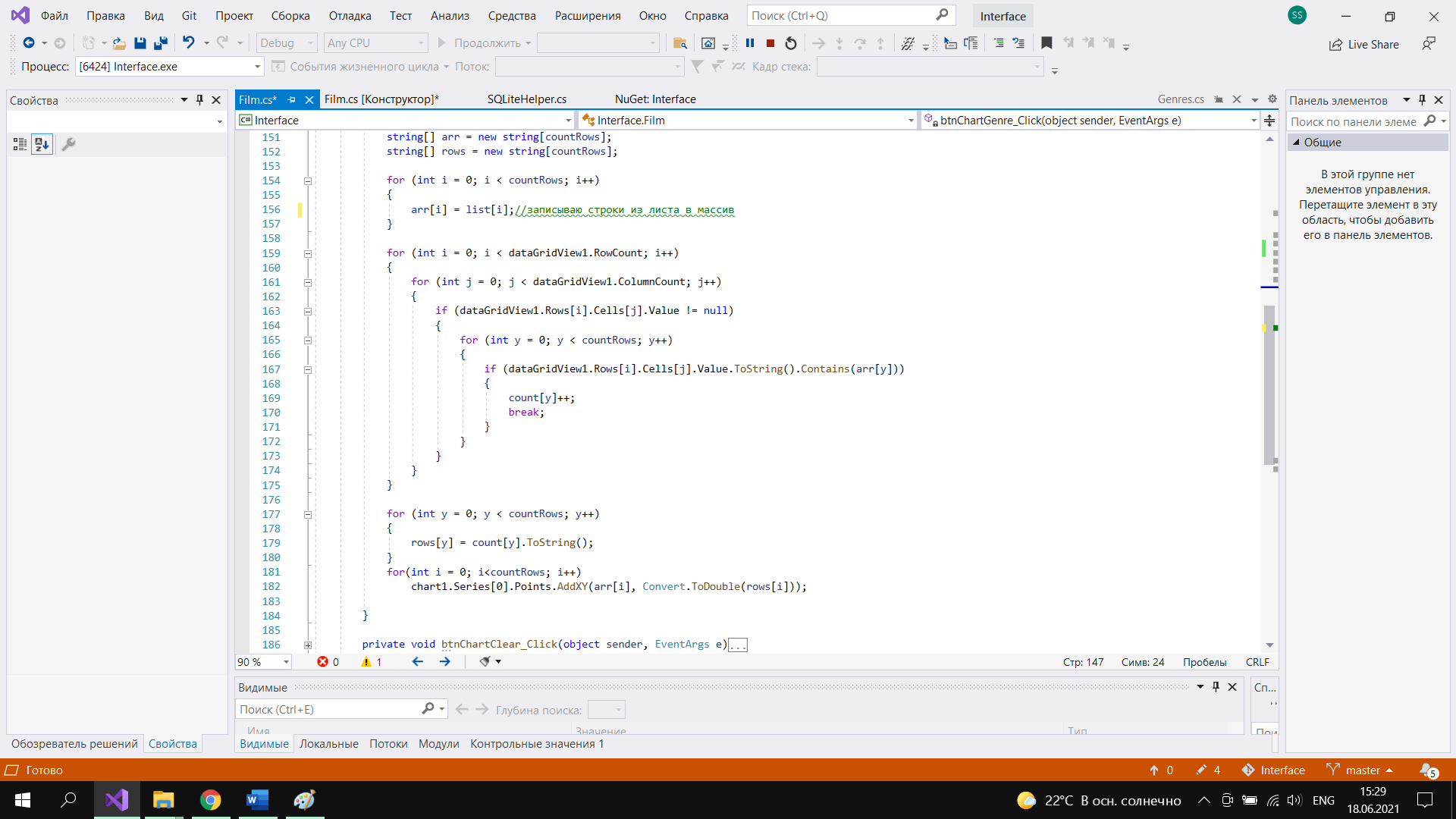


Рисунок 17 Добавление значений в диаграмму

Результат:

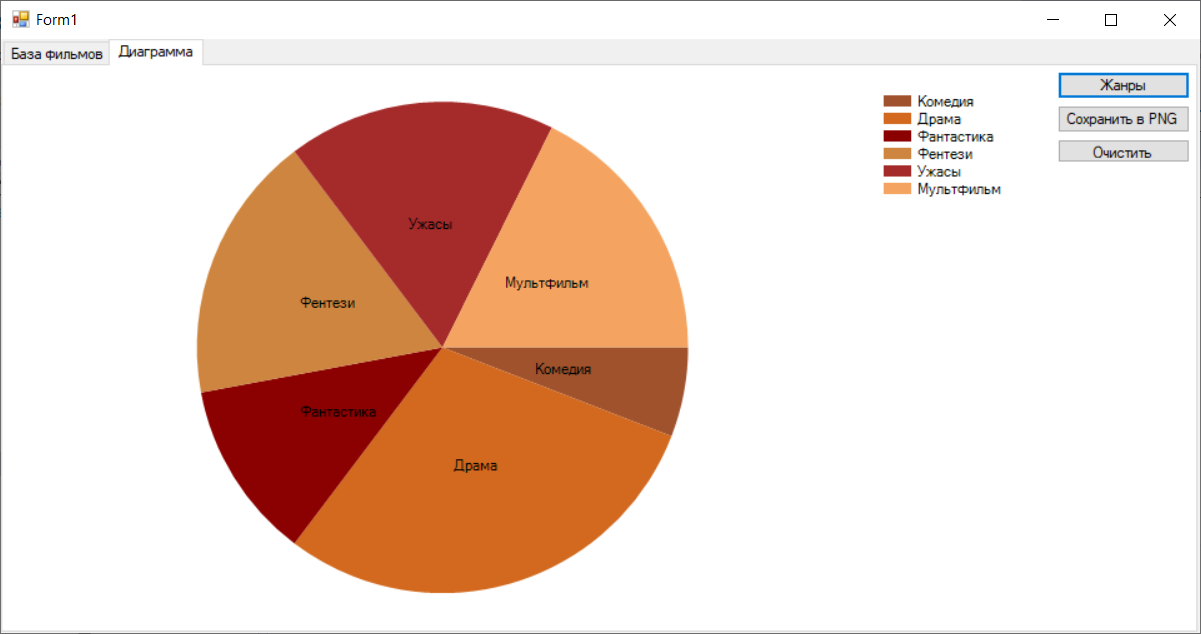


Рисунок 18 Результат динамической диаграммы

Кнопка btnSavePNG сохраняет диаграмму в .png файл с помощью метода SaveImage:

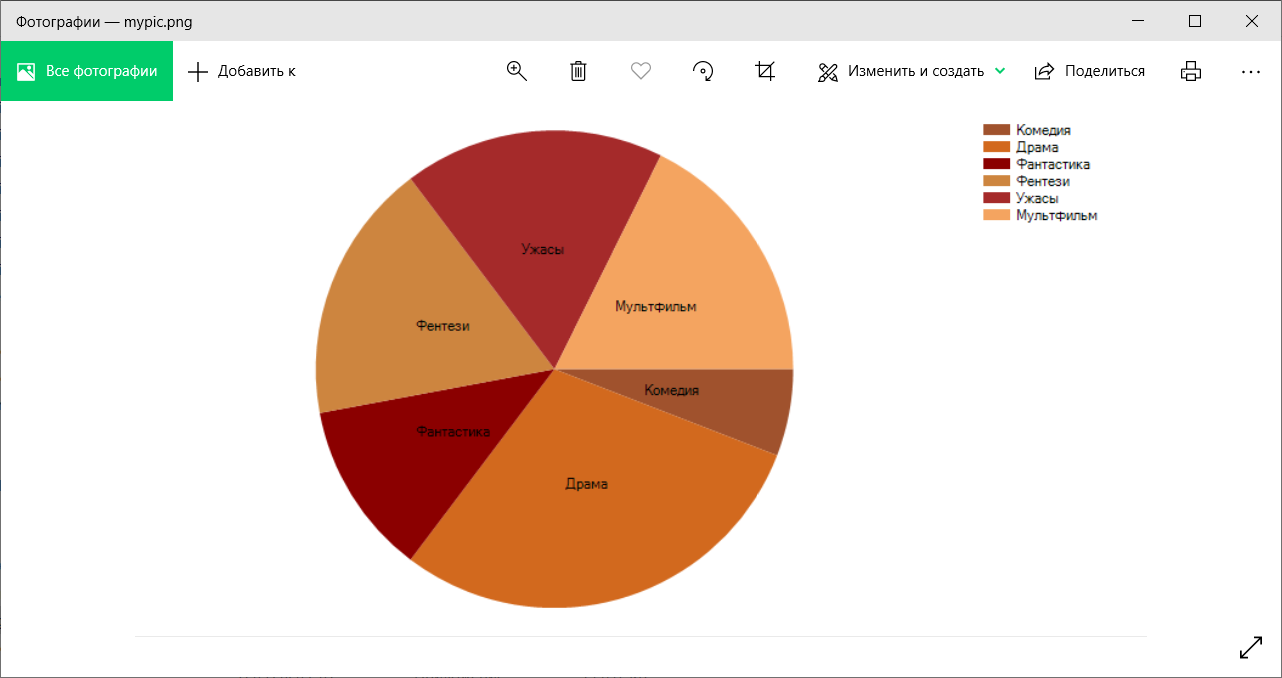


Рисунок 19 диаграмма в png

Кнопка btnChartClear удаляет все точки и соответственно диаграмму. При желании можно создать других разных кнопок для создания диаграмм и не закрывая программу с помощью кнопки «Очистить» вывести и другие.

## **Форма Genre**

Форма Genre содержит в себе tableLayoutPanel1 с двумя ячейками, в одной из них кнопка «Назад» с функционалом как у кнопки «Назад в меню» в форме Film. В другой находится tabControl1. Этот компонент позволяет не создавать разные формы, а сделать несколько вкладок в одной. В каждой форме свой datagridview, соединенный со своим bindingsourse, и выводит каждый свой SELECT. Прицип работы уже был описан выше

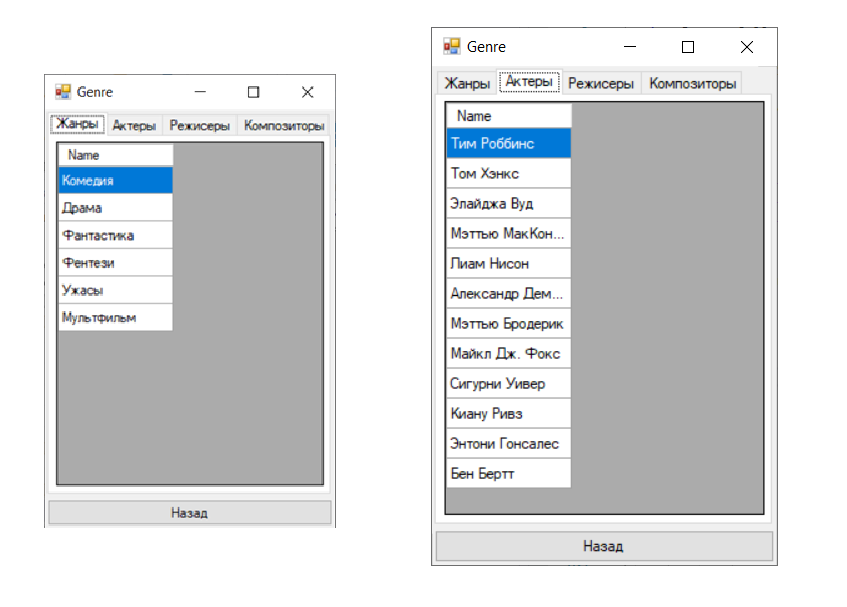


Рисунок 20 Работа формы Genre

# **Описание базы данных**

База данных содержит в себе пять таблиц: films, genres, directors, composers, actors.

Таблица films имеет в себе одиннадцать полей, четыре из которых внешний ключ подсоединенный к другим таблицам.

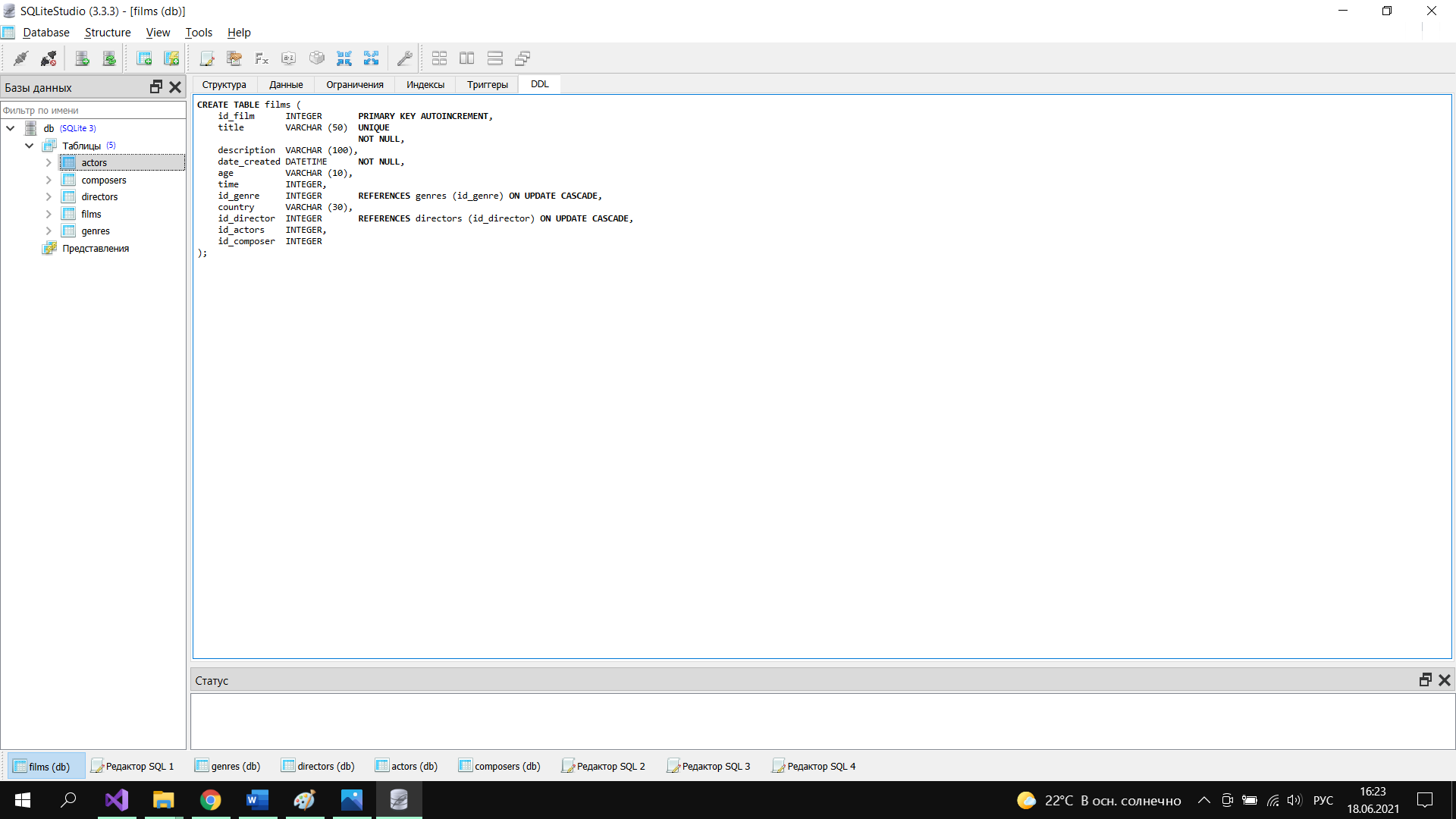


Рисунок 21 SQL код таблицы films

В таблице находится один первичный ключ с автозаполнением числами, при создании новой записи. Три поля типа varchar – это текстовые поля, где в скобках указывается количество символов, которые могут быть записаны. Одно поле типа datetime содержит, соответственно, дату (в данном случаю дату мировой премьеры). И шесть полей integer, то есть целочисленных. Название является уникальным и, как и дата создания, не может быть null.

Таблица actors содержит в себе три поля, два из них являются связывающими и одно varchar это имя актера.

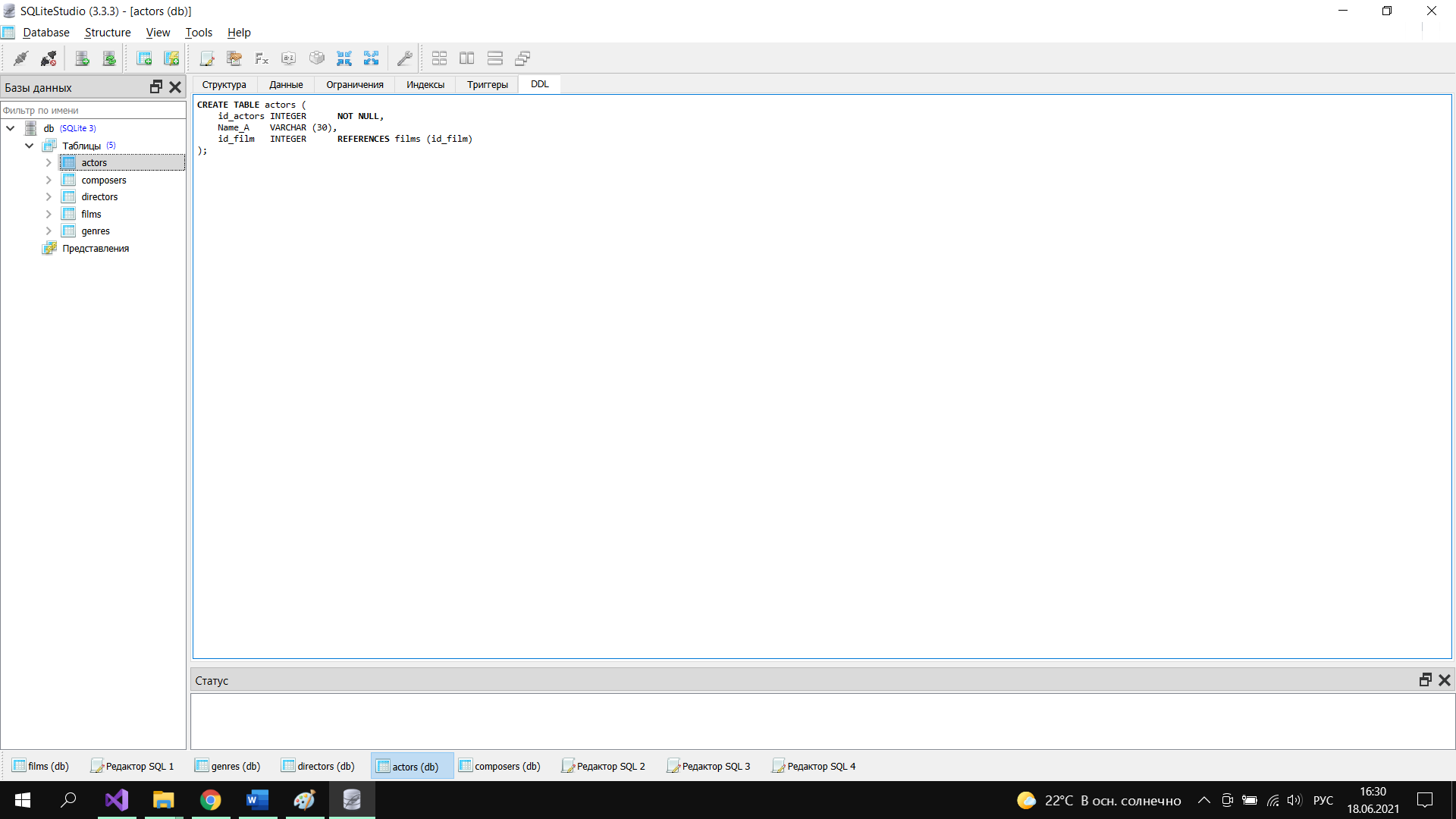


Рисунок 22 SQL код таблицы actors

По такому же принципу существуют оставшиеся три таблицы, в них меняется только название поля id\_\* и Name\_\*.

# **Список рисунков**

[Рисунок 1 SQLite Studio 5](#_Toc74926486)

[Рисунок 2 Форма №1 Start 7](#_Toc74926487)

[Рисунок 3 Библиотеки для работы с SQLite 7](#_Toc74926488)

[Рисунок 4 Код для возвращения столбцов из БД 7](#_Toc74926489)

[Рисунок 5 Форма Film первая вкладка 7](#_Toc74926490)

[Рисунок 6 Добавление строк на DataGridView 7](#_Toc74926491)

[Рисунок 7 Выбор шрифта 7](#_Toc74926492)

[Рисунок 8 Заполнение таблицы в PDF 7](#_Toc74926493)

[Рисунок 9 Поток и путь к папке PDF 7](#_Toc74926494)

[Рисунок 10 Результат PDF 7](#_Toc74926495)

[Рисунок 11 Работа кнопки btnFound 7](#_Toc74926496)

[Рисунок 12 Код поиска 7](#_Toc74926497)

[Рисунок 13 Работа comboBox 7](#_Toc74926498)

[Рисунок 14 Вторая вкладка формы Films 7](#_Toc74926499)

[Рисунок 15 Перезапись жанров в массив 7](#_Toc74926500)

[Рисунок 16 Поиск совпадение по массиву 7](#_Toc74926501)

[Рисунок 17 Добавление значений в диаграмму 7](#_Toc74926502)

[Рисунок 18 Результат динамической диаграммы 7](#_Toc74926503)

[Рисунок 19 диаграмма в png 7](#_Toc74926504)

[Рисунок 20 Работа формы Genre 6](#_Toc74926505)

[Рисунок 21 SQL код таблицы films 6](#_Toc74926506)

[Рисунок 22 SQL код таблицы actors 7](#_Toc74926507)

# **Заключение**

В ходе работы был спроектирован законченный программный продукт с графическим интерфейсом, включающей в себя работу с базами данных, с построением графиков и формированием отчета в формате PDF.

Была создана база данных с пятью таблицами, содержащими в себе информацию о фильмах, актерах, режиссерах, композиторах и жанрах. Обеспечена возможность просмотра данных в пяти разных формах, в одной из которых реализован поиск и фильтрация по жанру. Обеспечена возможность сохранения таблицы в PDF и создания диаграммы с последующим сохранением в расширении png.

# **Список источников**

1. Чарльз Петцольд «Программирование для Microsoft Windows на С#. В 2-х томах» 2002 г.
2. Windows Forms для .NET  [Электронный ресурс]  — Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/desktop/winforms/?view=netdesktop-5.0>
3. Албахари Дж. C# 6.0. Справочник. Полное описание языка [Текст] / Дж . Албахари, Б. Албахари — 6-е изд. — Москва: Вильямc, 2016. — 1040 c.
4. Грин Дж. Изучаем C# / Дж. Грин, Э. Стиллмен. – СПб.: «Питер», 2012.
5. Windows Forms для .NET  [Электронный ресурс]  — Режим доступа: <https://www.sqlite.org/src/help>
6. Биллиг, В. А. Основы объектного программирования на C# (C# 3.0, Visual Studio 2008) / В.А. Биллиг. - М.: Интернет-университет информационных технологий, Бином. Лаборатория знаний, 2016. - 584 c.
7. Задачи по программированию. - М.: Бином. Лаборатория знаний, 2014. - 824 c.
8. Черпаков, И. В. Основы программирования. Учебник и практикум / И.В. Черпаков. - М.: Юрайт, 2016. - 220 c.

# **Листинг**

## **Start.cs**

using System;

using System.Windows.Forms;

namespace Interface

{

public partial class Start : Form

{

public Start()

{

InitializeComponent();

}

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

{

Form frm = new Film();

frm.Show();

this.Hide();

}

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

{

Form frm = new Genre();

frm.Show();

this.Hide();

}

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

{

Application.Exit();

}

}

}

## **SQLiteHelper.cs**

using System;

using System.Collections.Generic;

using System.Data.SQLite;

namespace Interface.DAL

{

class SQLiteHelper

{

internal static List<Films> GetFilms()

{

try

{

using (var connection = new SQLiteConnection(@"Data Source = db.sqlite;Vesion=3;"))

{

connection.Open();

using (var cmd = new SQLiteCommand(@"SELECT DISTINCT title, description, date\_created, age, time, country, Name\_A, Name\_C, Name\_D, Name\_G FROM films, actors, composers, directors, genres WHERE films.id\_actors = actors.id\_actors AND films.id\_composer = composers.id\_composer AND films.id\_director = directors.id\_director AND films.id\_genre = genres.id\_genre", connection))

{

using (var rdr = cmd.ExecuteReader())

{

List<Films> films = new List<Films>();

while (rdr.Read())

{

films.Add(new Films

{

title = rdr.GetString(0),

Description = rdr.GetString(1),

Date = rdr.GetDateTime(2),

Age = rdr.GetString(3),

Time = rdr.GetInt32(4),

Country = rdr.GetString(5),

Actor = rdr.GetString(6),

Composer = rdr.GetString(7),

Director = rdr.GetString(8),

Genre = rdr.GetString(9)

});

}

return films;

}

}

}

}

catch (Exception ex) { Console.WriteLine(ex.Message); }

return null;

}

internal static List<Genres> GetGenre()

{

try

{

using (var connection = new SQLiteConnection(@"Data Source = db.sqlite;Vesion=3;"))

{

connection.Open();

using (var cmd = new SQLiteCommand(@"SELECT Name\_G FROM genres ", connection))

{

using (var rdr = cmd.ExecuteReader())

{

List<Genres> genres = new List<Genres>();

while (rdr.Read())

{

genres.Add(new Genres

{

Name = rdr.GetString(0)

});

}

return genres;

}

}

}

}

catch (Exception ex) { Console.WriteLine(ex.Message); }

return null;

}

internal static List<Actors> GetActors()

{

try

{

using (var connection = new SQLiteConnection(@"Data Source = db.sqlite;Vesion=3;"))

{

connection.Open();

using (var cmd = new SQLiteCommand(@"SELECT DISTINCT Name\_A FROM actors ", connection))

{

using (var rdr = cmd.ExecuteReader())

{

List<Actors> actors = new List<Actors>();

while (rdr.Read())

{

actors.Add(new Actors

{

Name = rdr.GetString(0)

});

}

return actors;

}

}

}

}

catch (Exception ex) { Console.WriteLine(ex.Message); }

return null;

}

## **Genre.cs**

using System;

using Interface.DAL;

using System.Collections.Generic;

using System.Windows.Forms;

namespace Interface

{

public partial class Genre : Form

{

private List<Genres> \_list;

private List<Actors> \_listA;

public Genre()

{

InitializeComponent();

\_list = new List<Genres>();

bsGenre.DataSource = \_list;

dataGridView1.AutoGenerateColumns = true;

dataGridView2.AutoGenerateColumns = true;

List<Genres> list = SQLiteHelper.GetGenre();

if (list != null && list.Count > 0)

{

\_list.AddRange(list);

bsGenre.ResetBindings(false);

}

ActorsOpen();

}

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

{

Close();

Form f = new Start();

f.Visible = true;

}

private void ActorsOpen()

{

dataGridView2.AutoGenerateColumns = true;

\_listA = new List<Actors>();

bsActor.DataSource = \_listA;

List<Actors> listA = SQLiteHelper.GetActors();

if (listA != null && listA.Count > 0)

{

\_listA.AddRange(listA);

bsActor.ResetBindings(false);

}

}

}

}

## **Film.cs**

using Interface.DAL;

using iTextSharp.text;

using iTextSharp.text.pdf;

using System;

using System.Collections.Generic;

using System.IO;

using System.Windows.Forms;

using System.Windows.Forms.DataVisualization.Charting;

namespace Interface

{

public partial class Film : Form

{

private List<Films> \_list;

public Film()

{

InitializeComponent();

\_list = new List<Films>();

bsUser.DataSource = \_list;

List<Films> list = SQLiteHelper.GetFilms();

if (list != null && list.Count > 0)

{

\_list.AddRange(list);

bsUser.ResetBindings(false);

dataGridView1.AutoGenerateColumns = true;

}

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

{

Close();

Form f = new Start();

f.Visible = true;

}

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

{

for (int i = 0; i < dataGridView1.RowCount; i++)

{

for (int j = 0; j < dataGridView1.ColumnCount; j++)

if (dataGridView1.Rows[i].Cells[j].Value != null)

{

if (dataGridView1.Rows[i].Cells[j].Value.ToString().IndexOf(textBox1.Text, StringComparison.CurrentCultureIgnoreCase) != -1)

{

dataGridView1.Rows[i].Visible = true;

break;

}

else

{

dataGridView1.CurrentCell = null;

dataGridView1.Rows[i].Visible = false;

}

}

}

}

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

{

for (int i = 0; i < dataGridView1.RowCount; i++)

{

dataGridView1.Rows[i].Visible = true;

}

}

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

{

BaseFont baseFont = BaseFont.CreateFont(@"C:\Windows\Fonts\Arial.ttf", BaseFont.IDENTITY\_H, BaseFont.NOT\_EMBEDDED);

Font font = new Font(baseFont, iTextSharp.text.Font.DEFAULTSIZE, iTextSharp.text.Font.NORMAL);

PdfPTable pdfTable = new PdfPTable(dataGridView1.ColumnCount);

foreach (DataGridViewColumn column in dataGridView1.Columns)

{

PdfPCell cell = new PdfPCell(new Phrase(column.HeaderText, font));

pdfTable.AddCell(cell);

}

foreach (DataGridViewRow row in dataGridView1.Rows)

{

foreach (DataGridViewCell cell in row.Cells)

{

pdfTable.AddCell(new Phrase(cell.Value.ToString(), font));

}

}

string folderPath = "D:\\PDFs\\";

if (!Directory.Exists(folderPath))

{

Directory.CreateDirectory(folderPath);

}

using (FileStream stream = new FileStream(folderPath + "Films.pdf", FileMode.Create))

{

Document pdfDoc = new Document(PageSize.A2, 10f, 10f, 10f, 0f);

PdfWriter.GetInstance(pdfDoc, stream);

pdfDoc.Open();

pdfDoc.Add(pdfTable);

pdfDoc.Close();

stream.Close();

}

MessageBox.Show("Pdf-документ сохранен");

}

private void comboBoxGenre\_SelectedIndexChanged(object sender, EventArgs e) {

for (int i = 0; i < dataGridView1.RowCount; i++)

{

dataGridView1.Rows[i].Selected = false;

for (int j = 0; j < dataGridView1.ColumnCount; j++)

if (dataGridView1.Rows[i].Cells[j].Value != null)

{

if (!dataGridView1.Rows[i].Cells[j].Value.ToString().Contains(comboBoxGenre.Text))

{

dataGridView1.CurrentCell = null;

dataGridView1.Rows[i].Visible = false;

}

else if (dataGridView1.Rows[i].Cells[j].Value.ToString().IndexOf(comboBoxGenre.Text, StringComparison.CurrentCultureIgnoreCase) != -1)

{

dataGridView1.Rows[i].Visible = true;

break;

}

}

}

}

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

{

List<Genres> list = SQLiteHelper.GetGenre();

int countRows = list.Count;

int[] count = new int[countRows];

string[] arr = new string[countRows];

string[] rows = new string[countRows];

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

{

arr[i] = list[i];

}

for (int i = 0; i < dataGridView1.RowCount; i++)

{

for (int j = 0; j < dataGridView1.ColumnCount; j++)

{

if (dataGridView1.Rows[i].Cells[j].Value != null)

{

for (int y = 0; y < countRows; y++)

{

if (dataGridView1.Rows[i].Cells[j].Value.ToString().Contains(arr[y]))

{

count[y]++;

break;

}

}

}

}

}

for (int y = 0; y < countRows; y++)

{

rows[y] = count[y].ToString();

}

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

chart1.Series[0].Points.AddXY(arr[i], Convert.ToDouble(rows[i]));

}

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

{

foreach (var series in chart1.Series)

{

series.Points.Clear();

}

}

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

{

string folderPath = @"D:\\mychart.png";

chart1.SaveImage(folderPath, ChartImageFormat.Png);

}

}

}

## **Films.cs**

using System;

using System.ComponentModel;

namespace Interface.DAL

{

internal class Films

{

[DisplayName("Название")]

public string title { get; internal set; }

[DisplayName("Описание")]

public string Description { get; internal set; }

[DisplayName("Дата выпуска")]

public DateTime Date { get; internal set; }

[DisplayName("Возрастное ограничение")]

public string Age { get; internal set; }

[DisplayName("Время(мин)")]

public int Time { get; internal set; }

[DisplayName("Страна")]

public string Country { get; internal set; }

[DisplayName("Главный актер")]

public string Actor { get; internal set; }

[DisplayName("Композитор")]

public string Composer { get; internal set; }

[DisplayName("Режисер")]

public string Director { get; internal set; }

[DisplayName("Жанр")]

public string Genre { get; internal set; }

}

}

## **Program.cs**

using System;

using System.Windows.Forms;

namespace Interface.DAL

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Start());

}

}

}

## **Genres.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Interface.DAL

{

class Genres

{

public string Name { get; internal set; }

public static implicit operator string(Genres v) => v.Name;

}

}