|  |  |
| --- | --- |
| Департамент образования Ярославской области  Государственное профессиональное образовательное  автономное учреждение Ярославской области  «ЯРОСЛАВСКИЙ ПРОМЫШЛЕННО-ЭКОНОМИЧЕСКИЙ КОЛЛЕДЖ  им. Н.П. ПАСТУХОВА» | |
|  |  |
| **Отчет по УЧЕБНОЙ практике**  **ПМ.02 Осуществление интеграции программных модулей** | |
| УП.02.09.02.07.19ИП2 | |
|  | |
|  | |
|  | |
|  | |
|  | Студент  \_\_\_\_\_\_\_\_\_\_\_\_\_ Кувыркин И.М.  «6» мая 2022 г |
|  | Преподаватель  \_\_\_\_\_\_\_\_\_\_\_\_\_ Куксов Е.О.  «6» мая 2022 г. |
|  |  |
| 2022 | |

**Практическая работа №1.1**

**Тема**

Создание базы данных.

**Цель работы**

По заданной схеме данных разработать базу данных, создать и сохранить скрипт с БД.

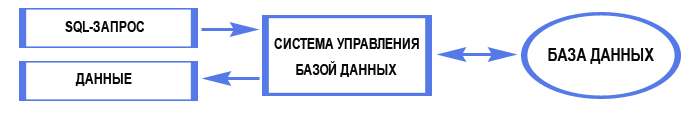
**Теоретический материал**

*База данных* - набор сведений, хранящихся некоторым упорядоченным способом.

*Система управления базами данных* — это совокупность языковых и программных средств, которая осуществляет доступ к данным, позволяет их создавать, менять и удалять, обеспечивает безопасность данных и т.д.

*SQL* - язык структурированных запросов, основной задачей которого является предоставление простого способа считывания и записи информации в базу данных.

Простейшая схема работы с базой данных выглядит примерно так:



В реляционной базе данных все данные можно представить в виде простых таблиц. Таблицы в реляционных базах данных обладают рядом свойств. Основными являются следующие:

* В таблице не может быть двух одинаковых строк.
* Столбцы располагаются в определенном порядке, который создается при создании таблицы. В таблице может не быть ни одной строки, но обязательно должен быть хотя бы один столбец.
* У каждого столбца есть уникальное имя (в пределах таблицы), и все значения в одном столбце имеют один тип (число, текст, дата...).
* На пересечении каждого столбца и строки может находиться только атомарное значение (одно значение, не состоящее из группы значений). Таблицы, удовлетворяющие этому условию, называют нормализованными.

*Первичный ключ* (сокращенно РК - primary key) - столбец, значения которого во всех строках различны.

В реляционных базах данных некоторая информация из одних таблиц присутствует в других, т.е. между ними имеются связи.

Всего существует 3 типа связей:

* [Один к одному](https://office-menu.ru/uroki-sql/41-tipy-svyazej-v-relyatsionnykh-bazakh-dannykh#onetoone);
* [Один ко многим](https://office-menu.ru/uroki-sql/41-tipy-svyazej-v-relyatsionnykh-bazakh-dannykh#onetomany);
* [Многие ко многим](https://office-menu.ru/uroki-sql/41-tipy-svyazej-v-relyatsionnykh-bazakh-dannykh#manytomany).

Ход работы:

В ходе работы была открыта программа SQL Server в котором на панели задач открыли вкладку сервер и убрали галочку с запретить изменения сохранения.

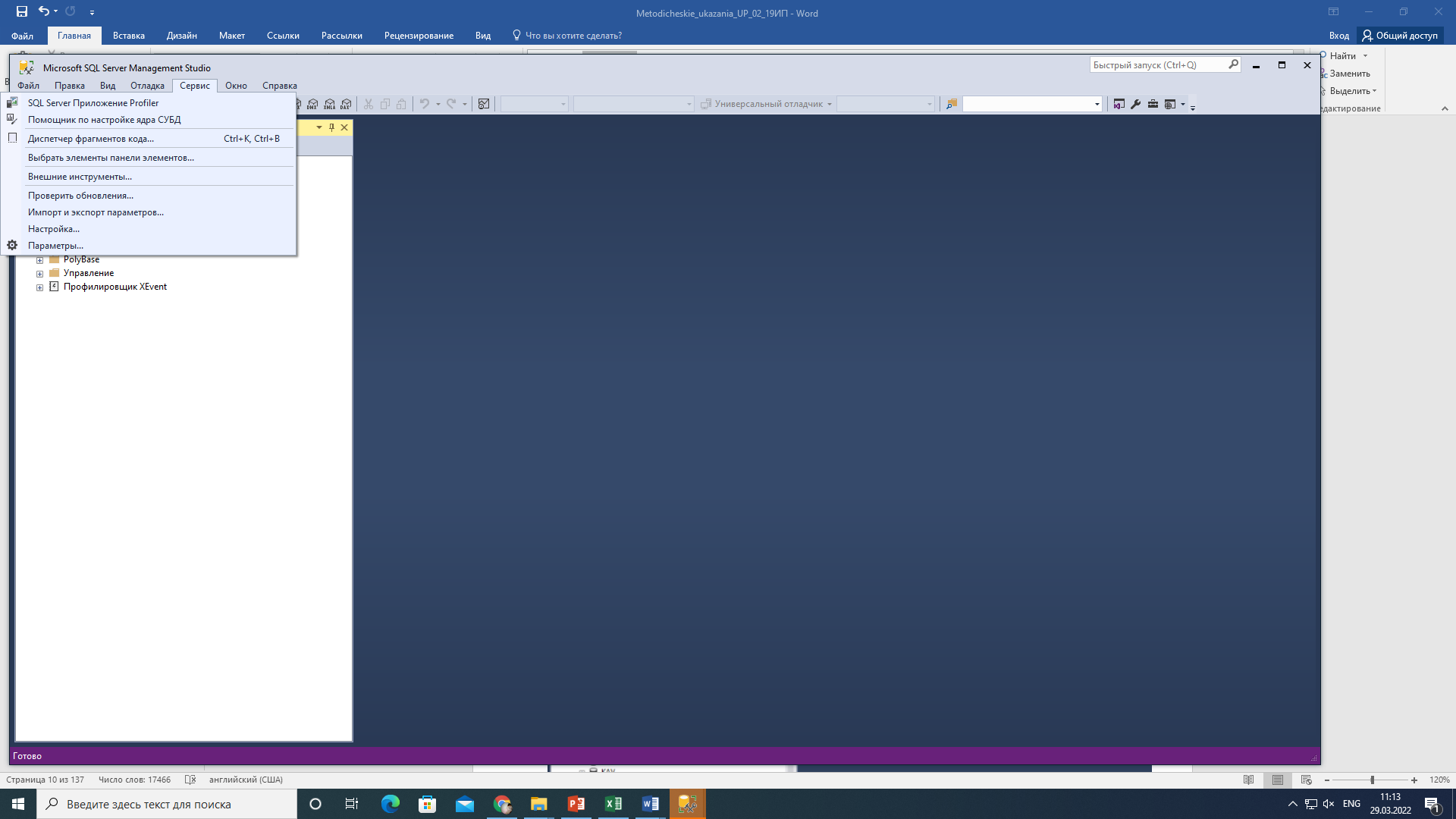


Рисунок 1. Главное меню.

Выберите в главном меню пункт «Параметры» (рисунок 1-2).

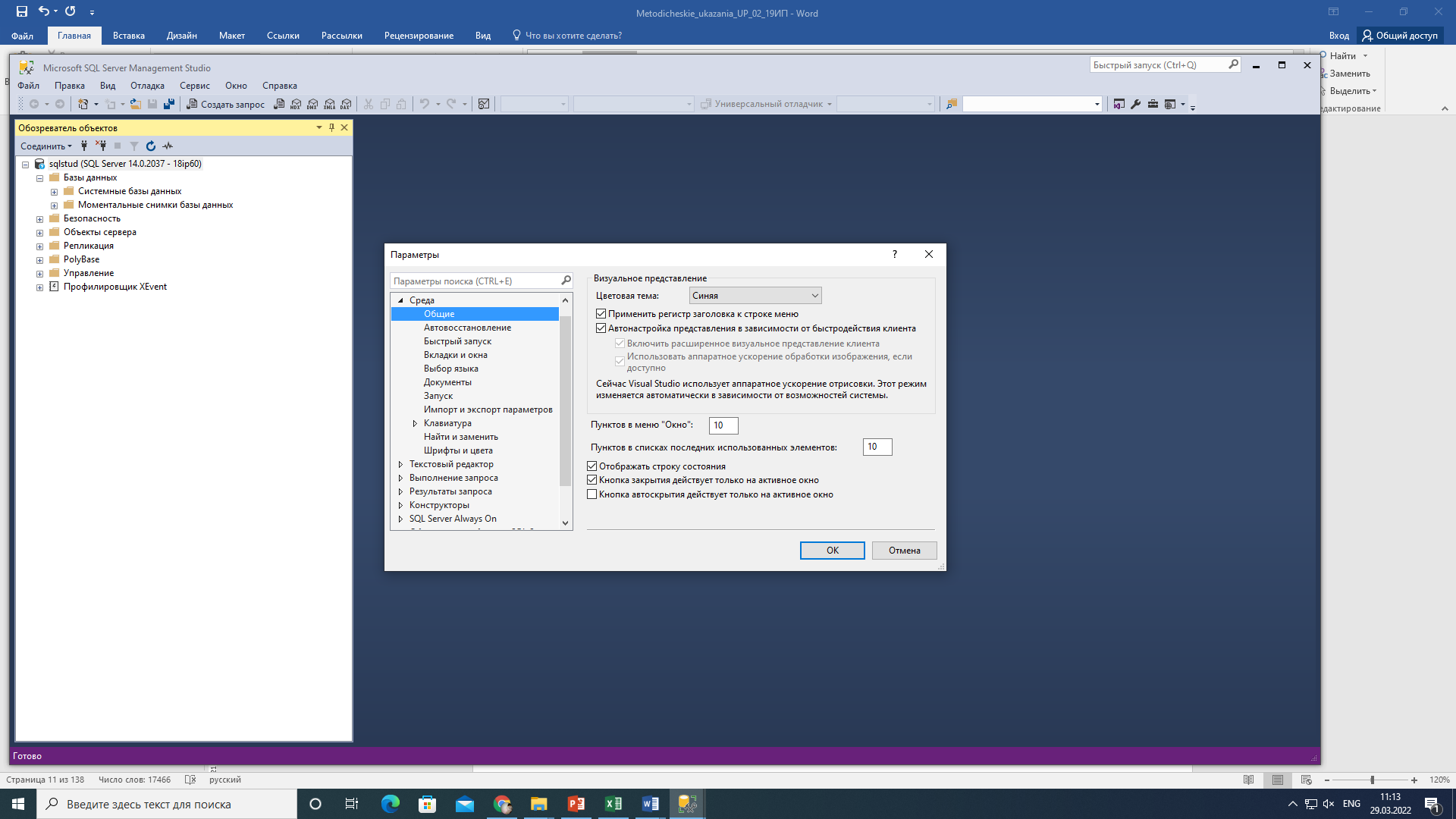


Рисунок 2. Окно «Параметры»

Уберите галочку у пункта «Запретить сохранение изменений, требующих повторного создания таблиц» (рисунок 3-4).

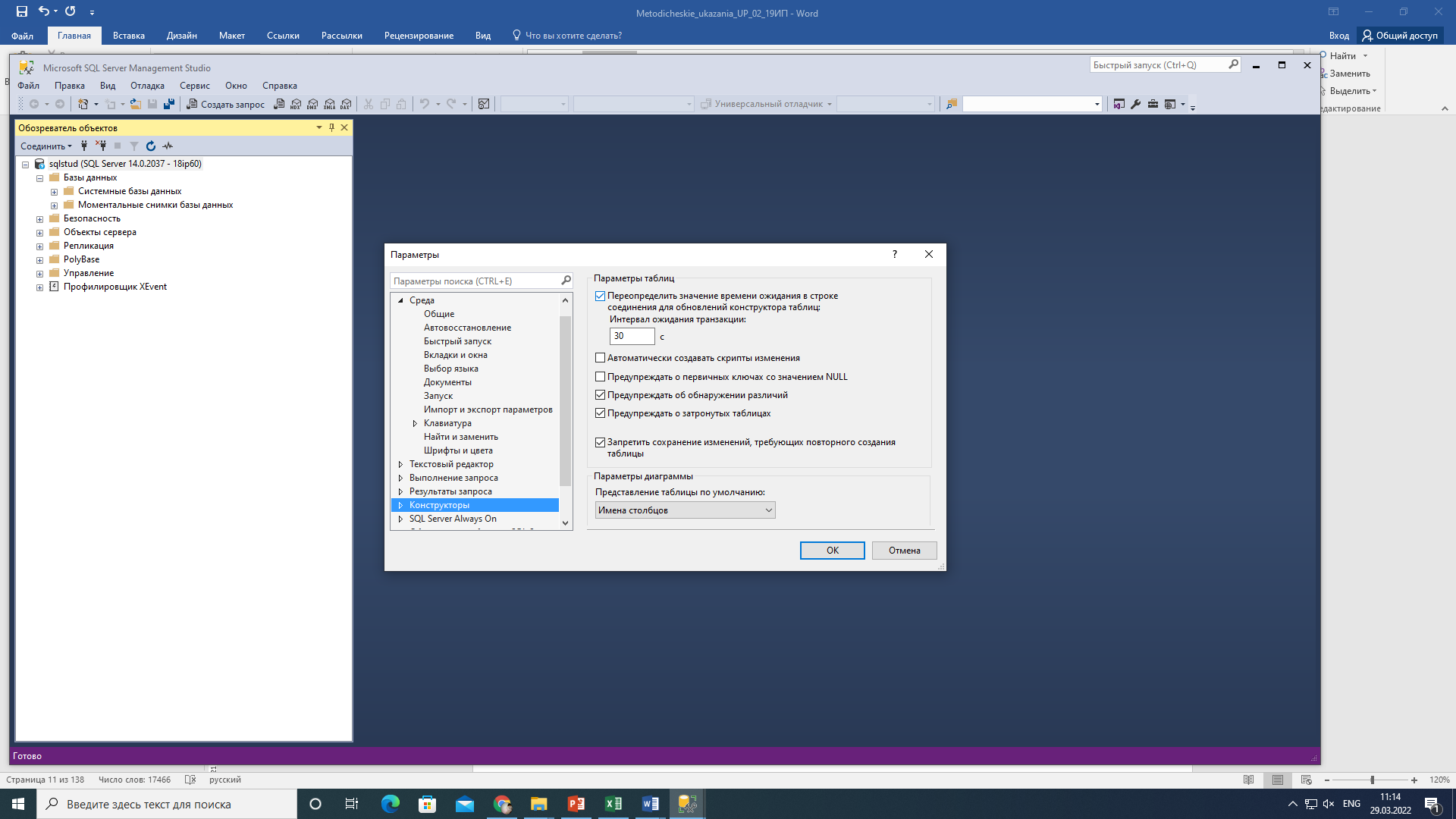


Рисунок 3 Окно «Параметры»

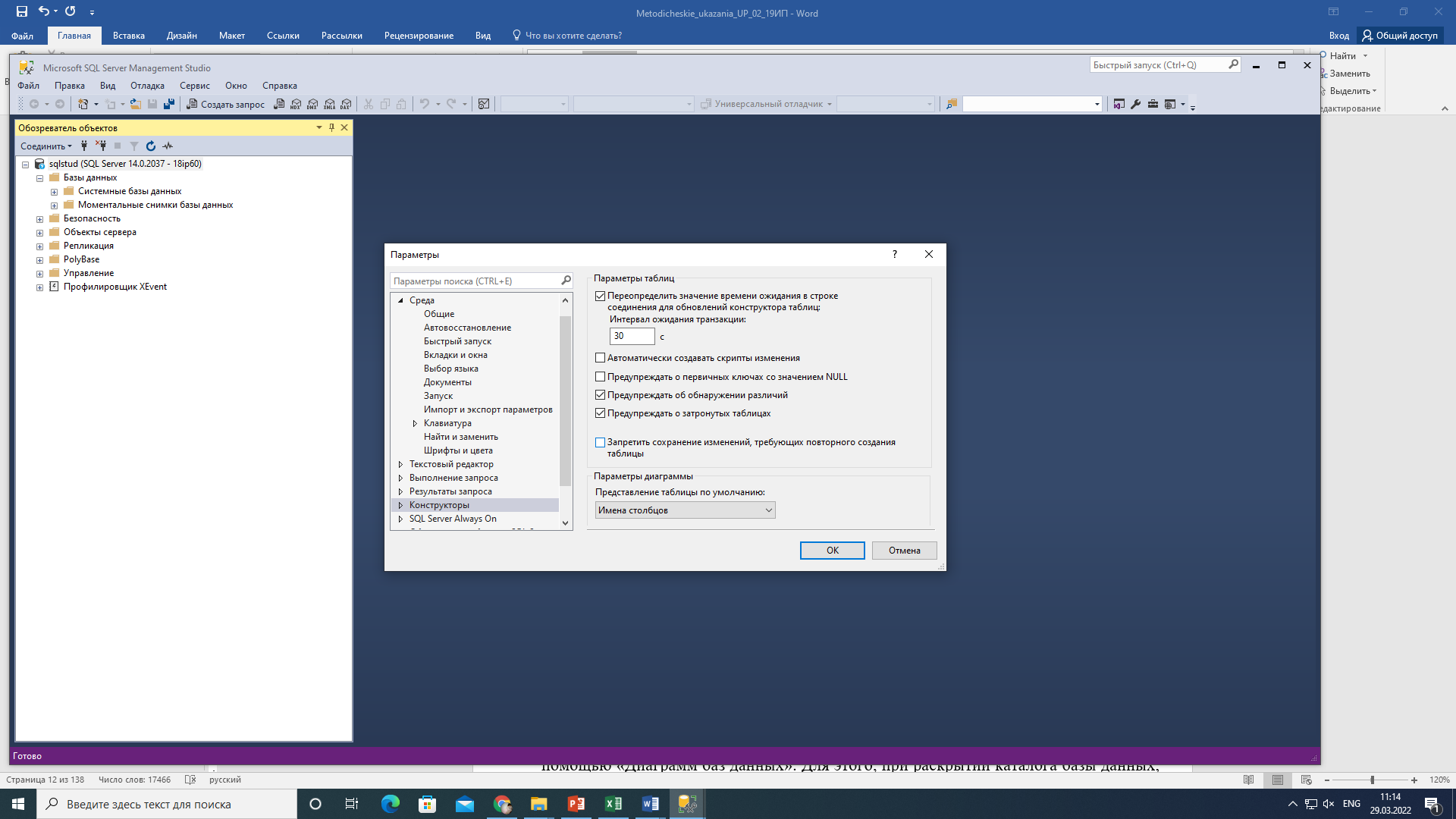


Рисунок 4. Окно «Параметры»

**Создание схемы данных.**

Схема базы данных используется для описания архитектуры базы данных, и она помогает обеспечить следующее:

* Единообразное форматирование записей данных.
* Все записи имеют уникальный первичный ключ.
* Никакие важные данные не пропущены.

Наиболее быстрым способ для создания базы данных является создание ее с помощью «Диаграмм баз данных». Для этого, при раскрытии каталога базы данных, необходимо кликнуть правой кнопкой мыши по папке «Диаграммы базы данных» и в отрывшемся меню выбрать «Создать диаграмму базы данных»

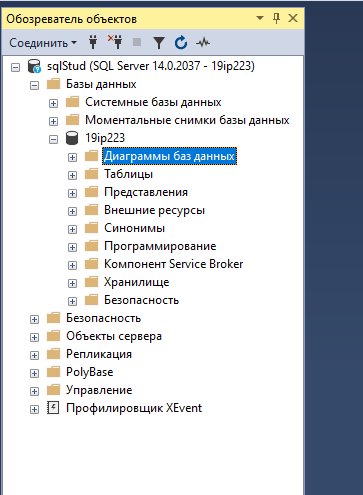


Рисунок 5. Выбор пункта «Создание диаграммы базы данных»

После того как была создана диаграмма, в неё были включены таблицы по области библиотека, так же были заданы ключи и соединения между таблицами.

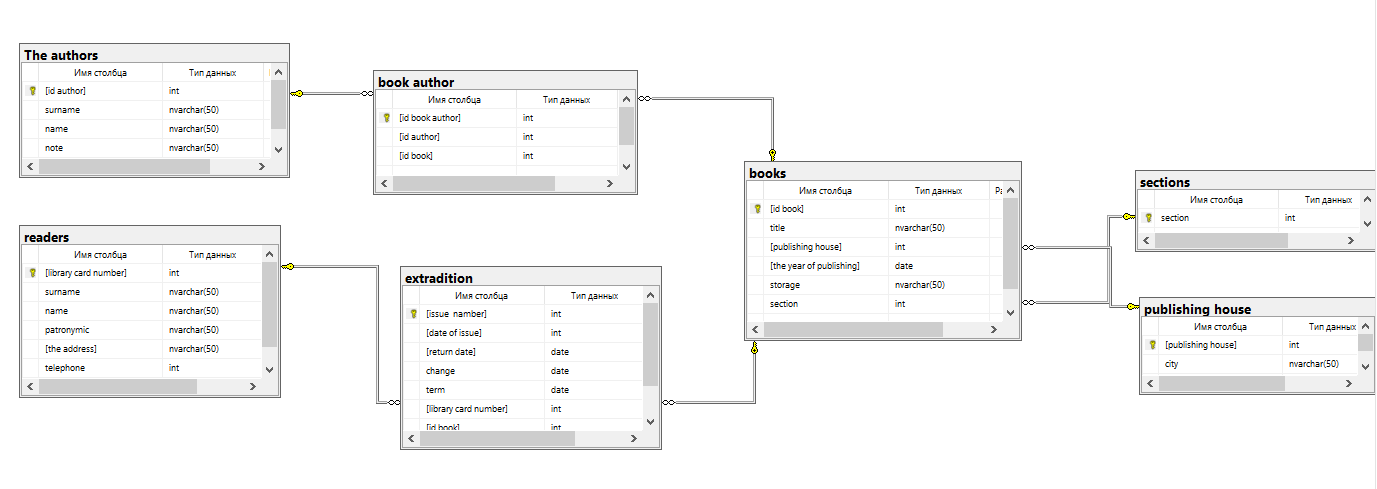


Рисунок 6 таблицы базы данных

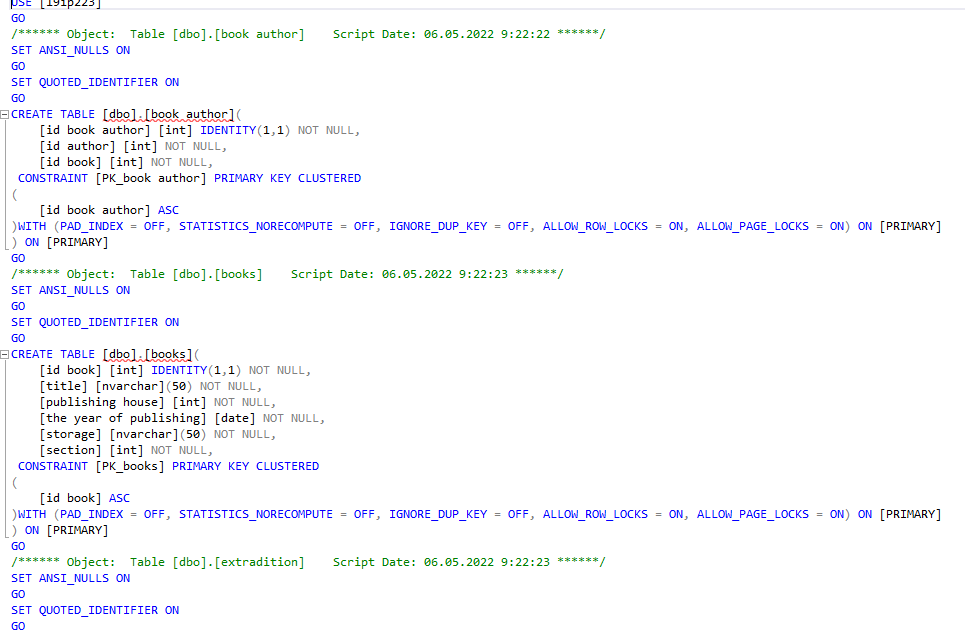


Рисунок 7 скрипт

**Контрольные вопросы**

1. Что такое «база данных»?

Набор сведений, хранящихся некоторым упорядоченным способом.

1. Что такое первичный ключ (Primary key)?

Столбец, значения которого во всех строках различны.

1. Что такое внешний ключ (Foreign key)?

Это столбец (или группа столбцов), используемый в реляционной базе данных для связи данных между таблицами.

1. Какие связи существуют в реляционных базах данных?

Связь «Один к одному» - одно записи из одно таблицы соответствует другая запись из другой таблицы. Такая связь может использоваться, когда у вас имеется очень большая таблица, но часть полей будет использовать реже, чем остальная часть.

Связь «Один ко многим» - когда запись в одной таблице соответствует множеству записей в другой таблице.

Связь «Многие ко многим» - когда множество записей в одной таблице соответствуют множеству записей в другой таблице.

1. Что подразумевается под целостностью данных?

Соответствие имеющейся в [базе данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) информации её внутренней логике, структуре и всем явно заданным правилам. Каждое правило, налагающее некоторое ограничение на возможное состояние базы данных, называется [ограничением целостности](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%D1%86%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%B8&action=edit&redlink=1).

1. Что такое нормализация и каковы ее преимущества?

На пересечении каждого столбца и строки может находиться только атомарное значение (одно значение, не состоящее из группы значений). Таблицы, удовлетворяющие этому условию, называют нормализованными.

Преимущества нормализации

Нормализация имеет целый ряд преимуществ:

* лучшая общая организация базы данных;
* сокращение числа ненужных повторений данных;
* согласованность данных внутри базы данных;
* более гибкая структура базы данных;
* эффективные возможности обеспечения безопасности и надежности базы данных.

**Практическая работа №1.2**

**Тема**

Подключение базы данных.

**Цель работы**

По заданной схеме данных разработать базу данных, создать и сохранить скрипт с БД, реализовать программное приложение с возможностью авторизации в системе.

**Теоретический материал**

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

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

Преимущества WPF:

* Использование традиционных языков .NET-платформы - C# и VB.NET для создания логики приложения;
* Возможность декларативного определения графического интерфейса с помощью специального языка разметки XAML, основанном на xml и представляющем альтернативу программному созданию графики и элементов управления, а также возможность комбинировать XAML и C#/VB.NET;
* Независимость от разрешения экрана: поскольку в WPF все элементы измеряются в независимых от устройства единицах, приложения на WPF легко масштабируются под разные экраны с разным разрешением;
* Новые возможности, которых сложно было достичь в WinForms, например, создание трехмерных моделей, привязка данных, использование таких элементов, как стили, шаблоны, темы и др.;
* Хорошее взаимодействие с WinForms, благодаря чему, например, в приложениях WPF можно использовать традиционные элементы управления из WinForms;
* Богатые возможности по созданию различных приложений: это и мультимедиа, и двухмерная и трехмерная графика, и богатый набор встроенных элементов управления, а также возможность самим создавать новые элементы, создание анимаций, привязка данных, стили, шаблоны, темы и многое другое;
* Аппаратное ускорение графики - вне зависимости от того, работаете ли вы с 2D или 3D, графикой или текстом, все компоненты приложения транслируются в объекты, понятные Direct3D, и затем визуализируются с помощью процессора на видеокарте, что повышает производительность, делает графику более плавной;
* Создание приложений под множество ОС семейства Windows - от Windows XP до Windows 10.

Также стоит учитывать, что по сравнению с приложениями на Windows Forms объем программ на WPF и потребление ими памяти в процессе работы в среднем несколько выше. Но это с лихвой компенсируется более широкими графическими возможностями и повышенной производительностью при отрисовке графики.

**Ход работы:**

В ходе работы была запущена программа Visual Studio в которой создали проект WPF

**Подключение базы данных.**

Откроем среду разработки Visual Studio и создадим новый проект «WPF». В качестве названия проекта укажите наименование вашей предметной области + «IS» (рисунок 7). Нажмите кнопку «Ок».

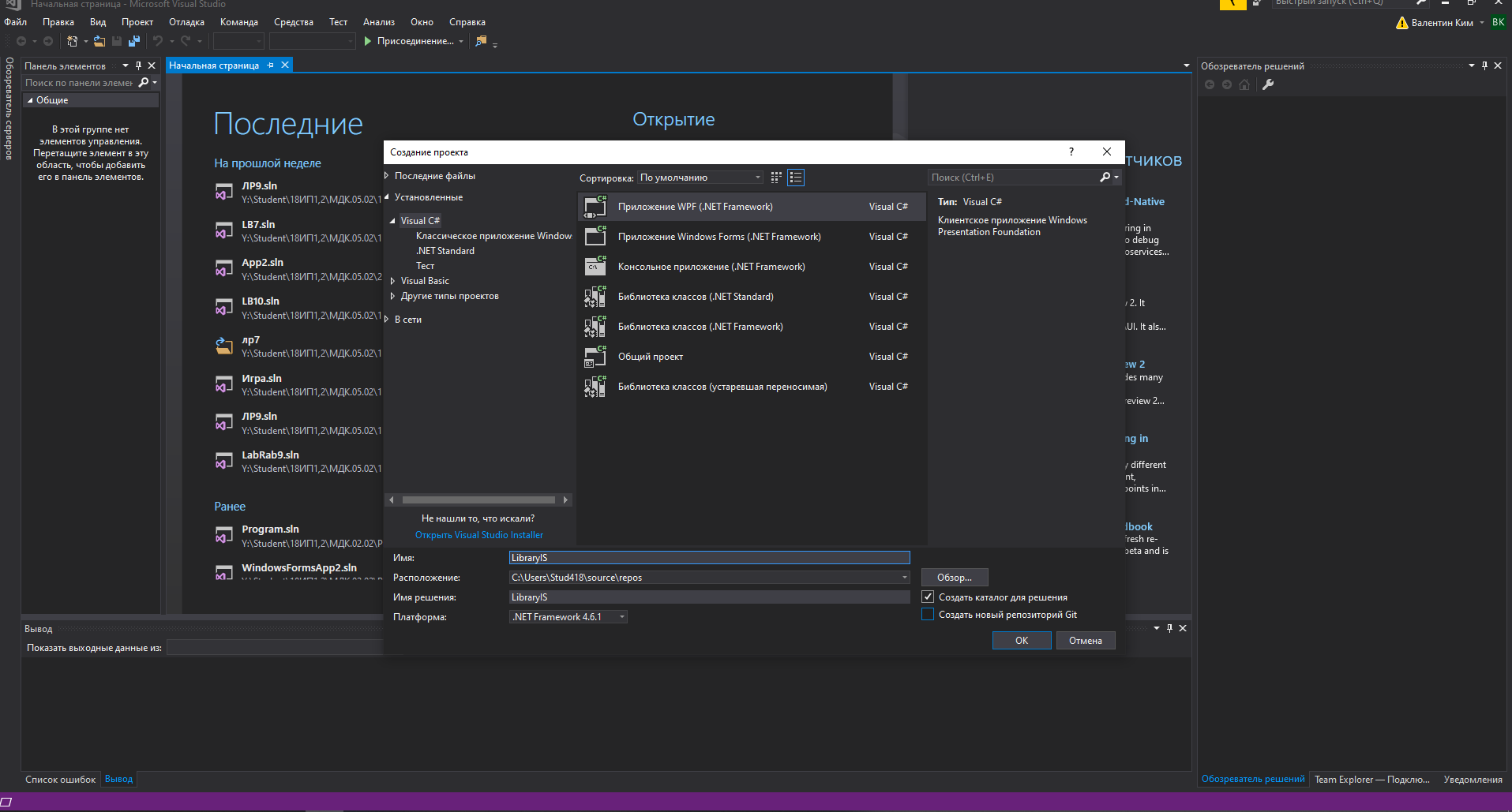


Рисунок 7. Окно «Создание проекта»

Следующим шагом будет подключение базы данных к проекту. Для этого на панели проекта нажмем правой кнопкой мыши по нашему проекту и выберем пункт «Добавить» -> «Создать новый элемент» (рисунок 8).

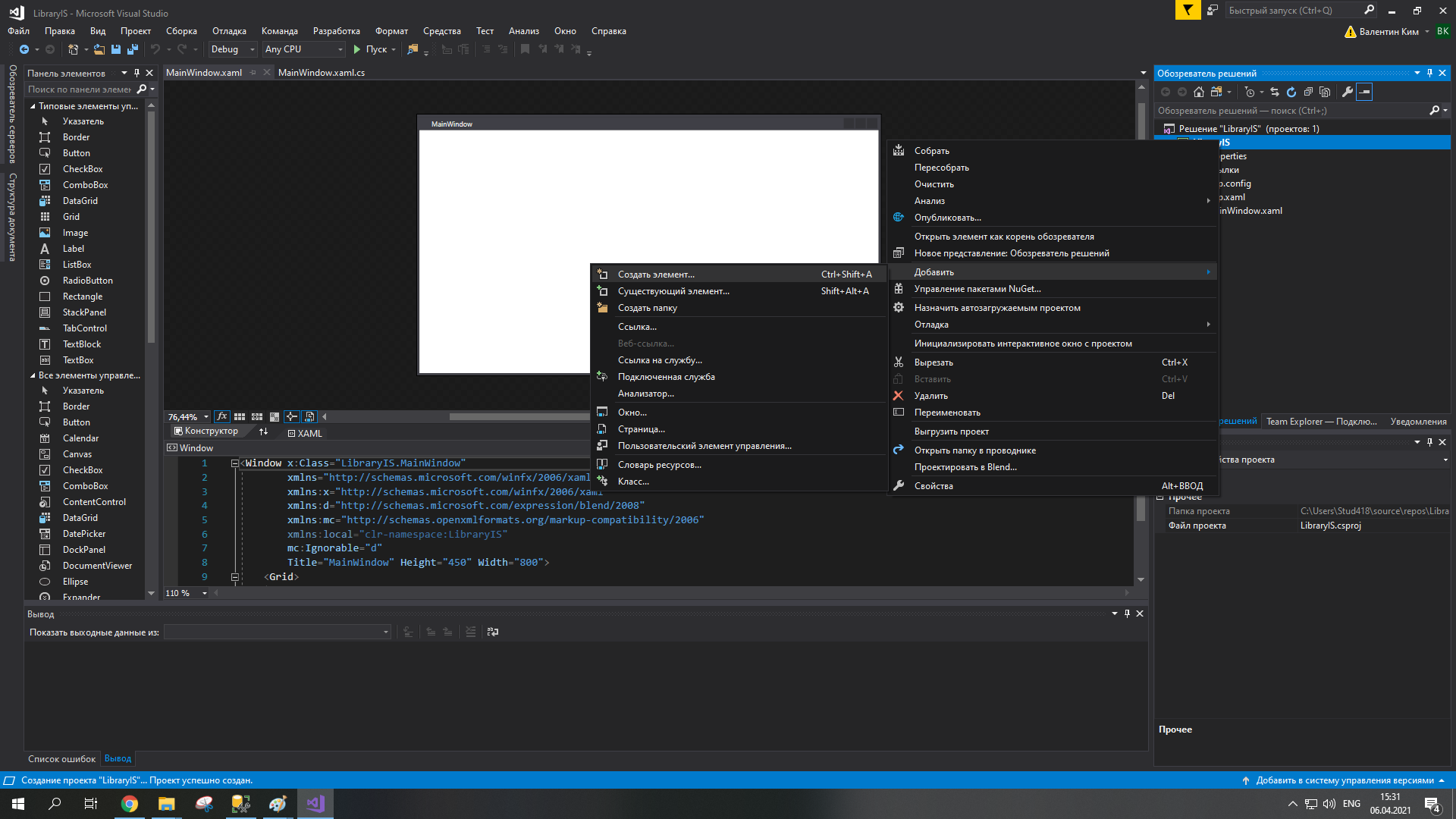


Рисунок 8. Создание нового элемента

Из списка элементов выберем элемент «Модель ADO.NET EDM». Зададим данному элементу название. (рисунок 9).

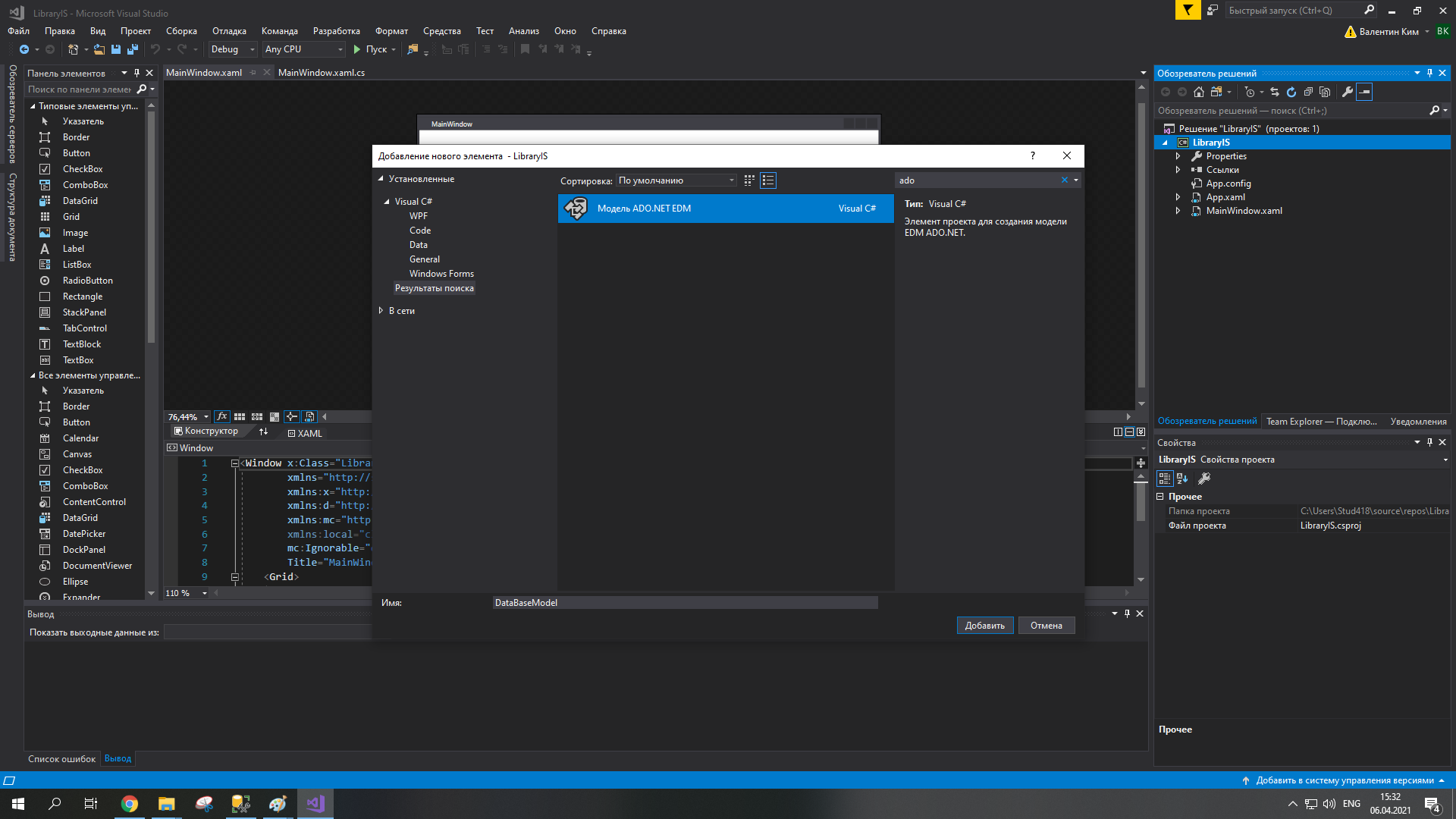


Рисунок 9. Создание нового элемента «Модель ADO.NET EDM».

Откроется мастер создания моделей EDM. Выбираем «Конструктор EF из базы данных» и жмем кнопку «Далее» (рисунок 10).

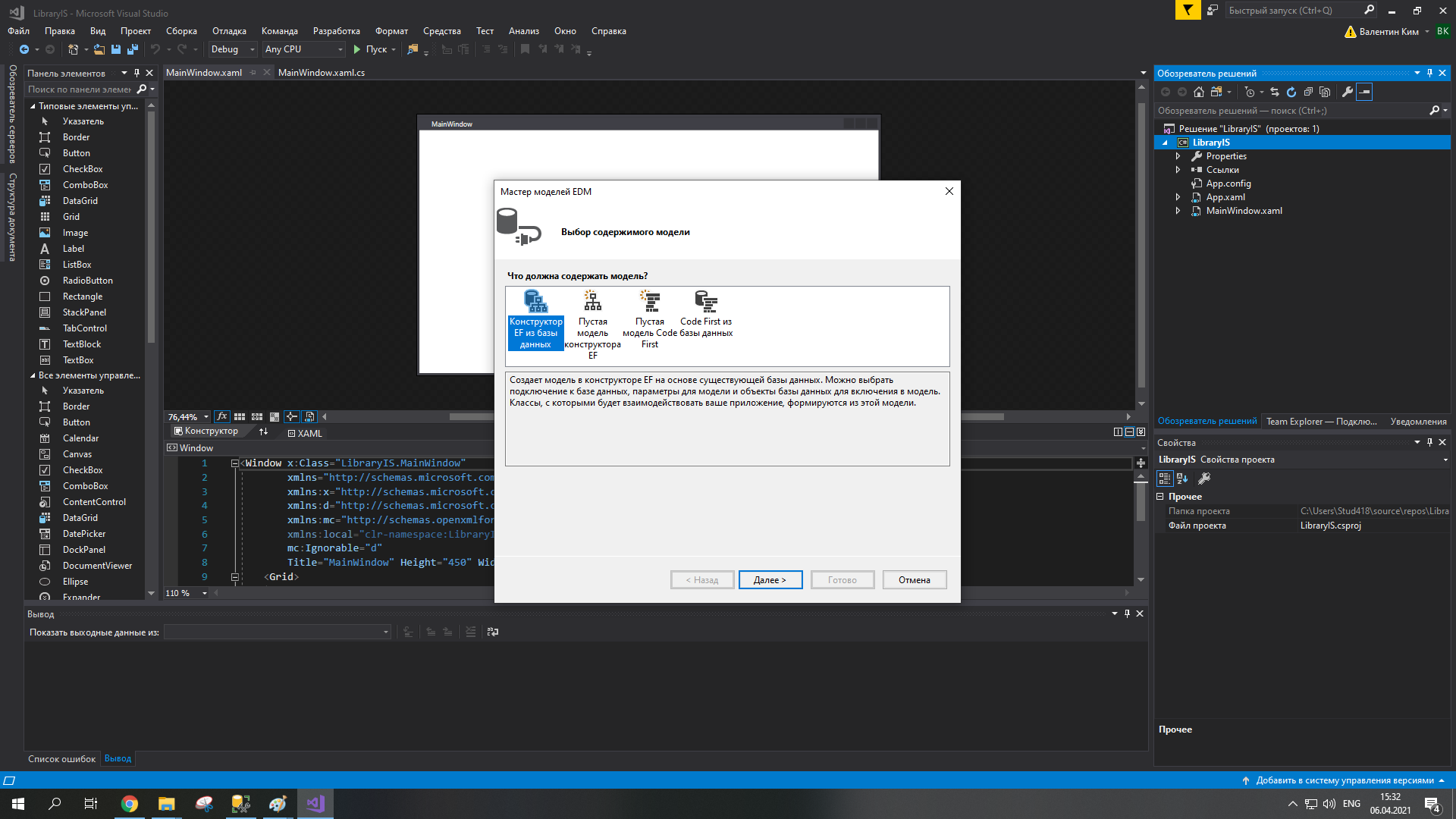


Рисунок 10. Мастер моделей EDM, шаг 1

В следующем окне нажимаем на кнопку «Создать соединение» (рисунок 11).

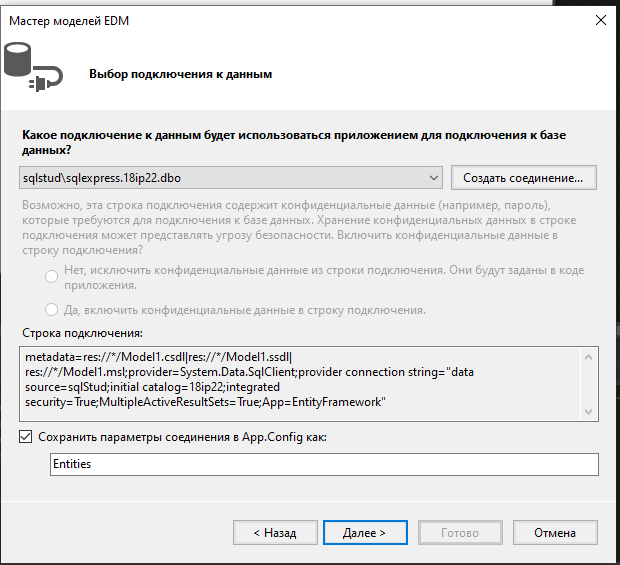


Рисунок 11. Мастер моделей EDM, шаг 2.

Задали соединение где были прописаны данные для подключения. После чего было произведено соединение где произошла выгрузка таблиц в среду Visual Studio (рисунок 12)

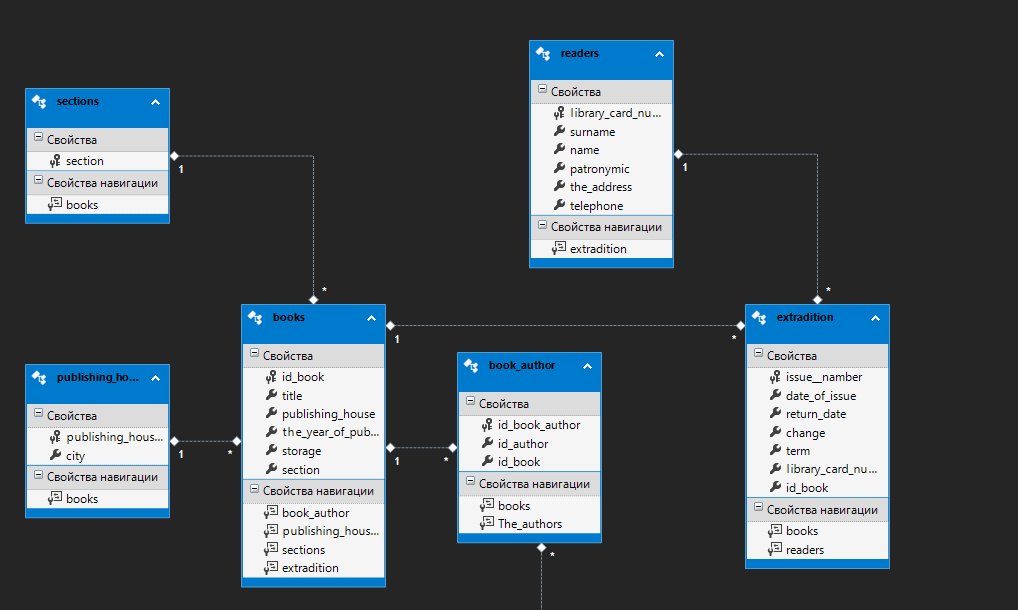


Рисунок 12 база данных

Теперь нужно реализовать возможность работать с базой данных непосредственно из кода.

Для этого был создан новый класс, например, с названием DataBase (рисунок 13).

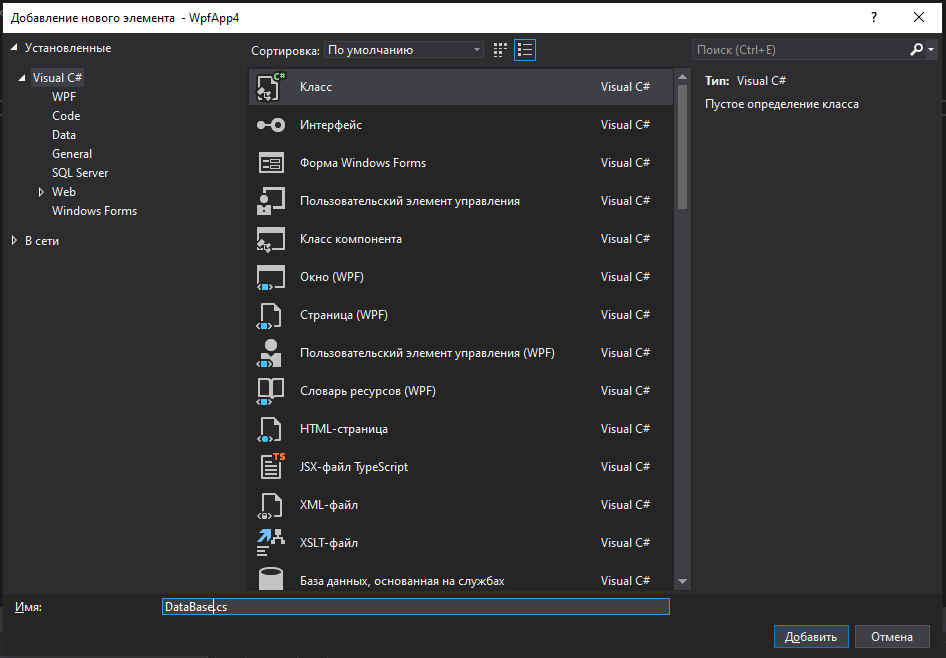
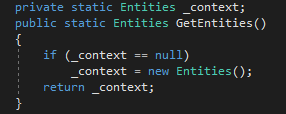


Рисунок 13. Добавление нового элемента.

После того как был создан класс, был прописан публичный статистический метод, который будет возвращать данный объект.



Получим следующий код (рисунок 14).

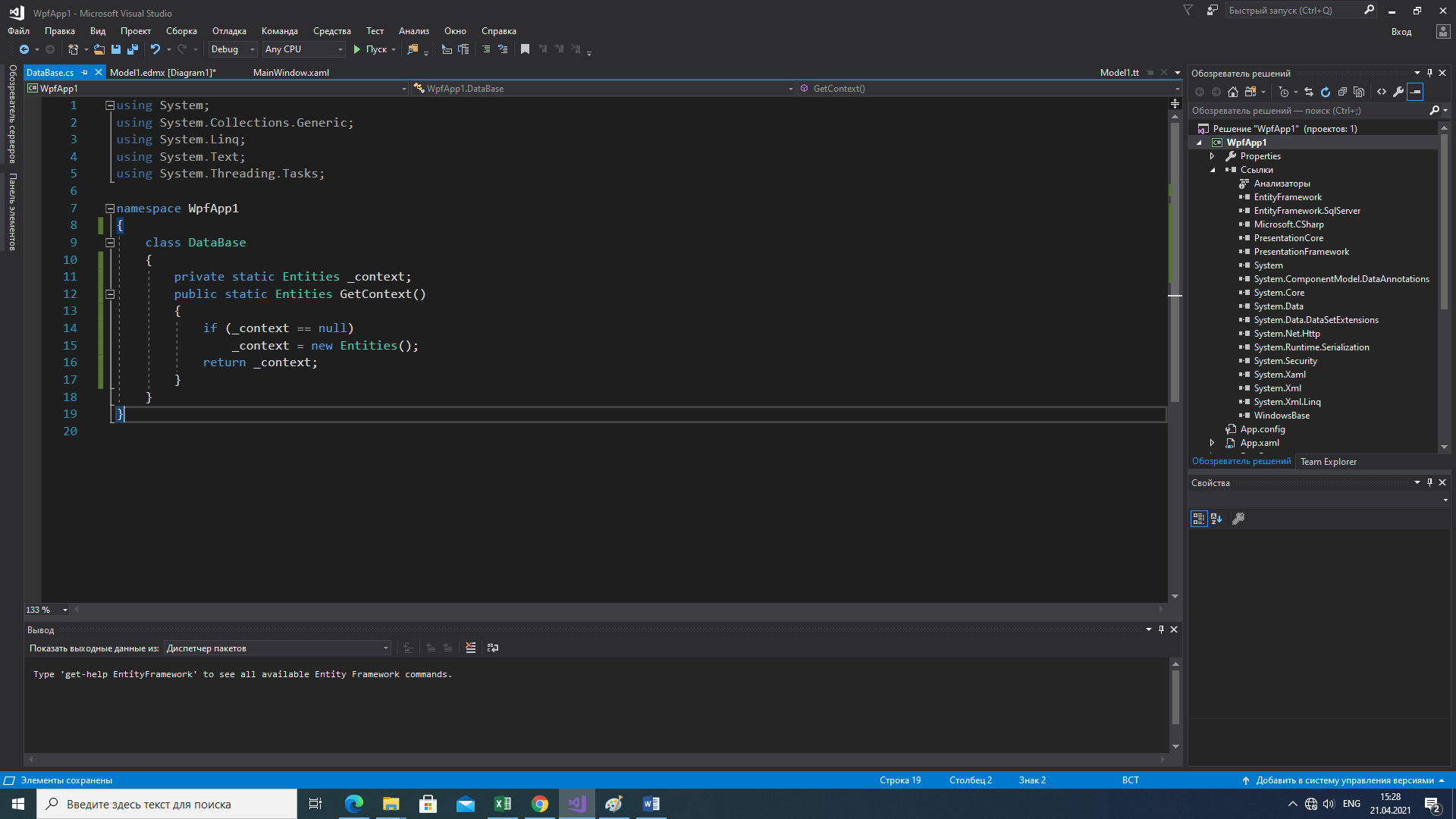


Рисунок 15. Публичный статичный метод

**Контрольные вопросы**

1. Как создать подключение к базе данных?

Для того чтобы создать подключение базы данных нужно в обозревателе решений создать новый элемент «Модель ADO.NET EDM». Задать название, после чего настроить соединение где мы указываем сервер, пароль. После чего выбираем таблицы для выгрузки и нажимаем готово .

**Практическая работа №1.3**

**Тема**

Создание руководства по стилю, логотипа, иконки. Размещение файлов в репозитории.

**Цель работы**

Разместить скрипт с базой данных, руководство по стилю, логотип и иконку в удаленный репозиторий проекта.

**Теоретический материал**

Система контроля версий — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Git — одна из распределенных систем контроля версий.

GitHub — один из сервисов для использования системы контроля версий Git.

Git — одна из систем контроля версий.

Git предназначена, в основном, для работы распределенной команды разработчиков.

То есть разработчики могут находиться в разных концах света и работать над одним проектом.

Система Git очень экономична и не требует рассылки большого количества файлов. Отслеживаются и пересылаются изменения в файлах и ссылки на эти изменения. То есть основная рассылка — это рассылка разницы в ваших редактированиях.

Отсылаются только различия в папках и файлах. В любой момент времени вы можете возвратиться к тому или иному состоянию системы. Многие компании уделяют внимание хорошей и быстрой коммуникации между сотрудниками. В этом отношении, система контроля версий предоставляет большие возможности. Всю мощь и гибкость системы управления версиями вы сможете ощутить после изучения некоторого теоретического материала и применения на практике.

Основные обозначения:

repository — некоторое хранилище файлов, ссылок на изменения в файлах

commit — отслеживание изменений, сохраняет разницу в изменениях

working directory — рабочий каталог на вашем компьютере

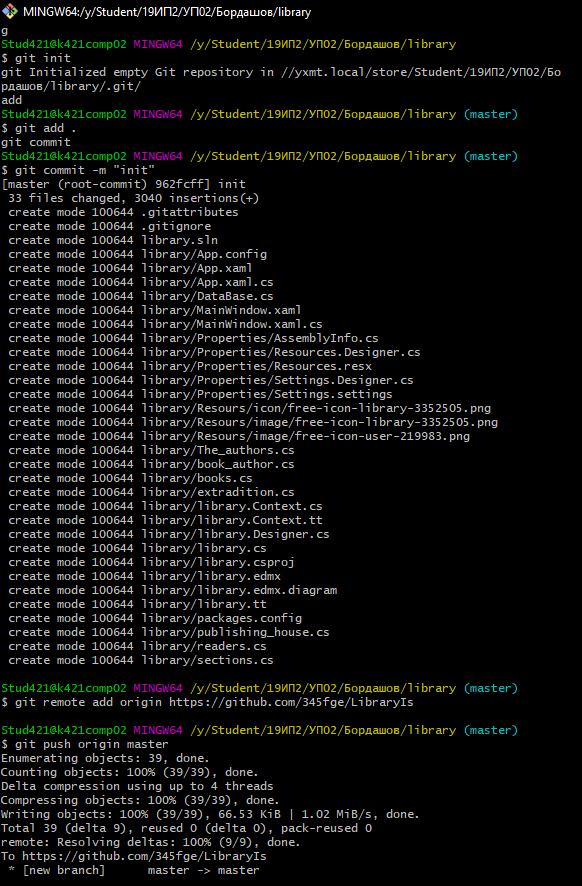
staging area — область подготовленных файлов или рабочая область

branch — ветка, состоит из набора коммитов, обычно ссылается на последний коммит

merge — слияние, слияние веток в одну

pull — втянуть, взять проект с сервера, получить изменения из удаленного репозитория

push — вытолкнуть, отправить изменения на сервер



**Практическая работа №1.4**

**Тема**

Создание словаря стилей. Создание Git репозитория с помощью Visual Studio.

**Цель работы**

Создать и настроить проект в Visual Studio, создать репозиторий, используя средства Visual Studio.

**Теоретический материал**

Чтобы разделить ресурсы между множеством проектов можно создать словарь ресурсов.

**Словарь ресурсов** представляет собой просто XAML-документ, который всего лишь хранит необходимые ресурсы.

Чтобы использовать словарь ресурсов, где-нибудь в приложении его необходимо объединить с коллекцией ресурсов. Это можно делать в каком-то конкретном окне, однако чаще объединение осуществляется на уровне коллекции ресурсов приложения, как показано ниже:

<Application x:Class="WpfApplication1.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

StartupUri="MainWindow.xaml">

<Application.Resources>

<ResourceDictionary>

<ResourceDictionary.MergedDictionaries>

<ResourceDictionary Source="Dictionary1.xaml"></ResourceDictionary>

</ResourceDictionary.MergedDictionaries>

</ResourceDictionary>

</Application.Resources>

</Application>

В приведенном коде разметки объект ResourceDictionary создается явно. Коллекция ресурсов всегда представляет собой объект ResourceDictionary, но данный случай является одним из тех, в которых эта деталь должна быть обязательно указана явно, чтобы иметь возможность также устанавливать свойство **ResourceDictionary.MergedDictionaries**. Если не предпринять этот шаг, значением свойства **MergedDictionaries** будет null.

**MergedDictionaries** — это коллекция объектов **ResourceDictionary**, которые будут использоваться для пополнения коллекции ресурсов.

Чтобы добавить собственные ресурсы и включить их в словари ресурсов, необходимо просто разместить их перед или после раздела **MergedProperties**.

Как упоминалось ранее, хранить ресурсы с одинаковыми именами в разных, но перекрывающихся коллекциях ресурсов вполне допустимо, однако объединять словари ресурсов, которые содержат ресурсы с одинаковыми именами — нет. При обнаружении дубликата во время компиляции приложения генерируется исключение **XamlParseException**.

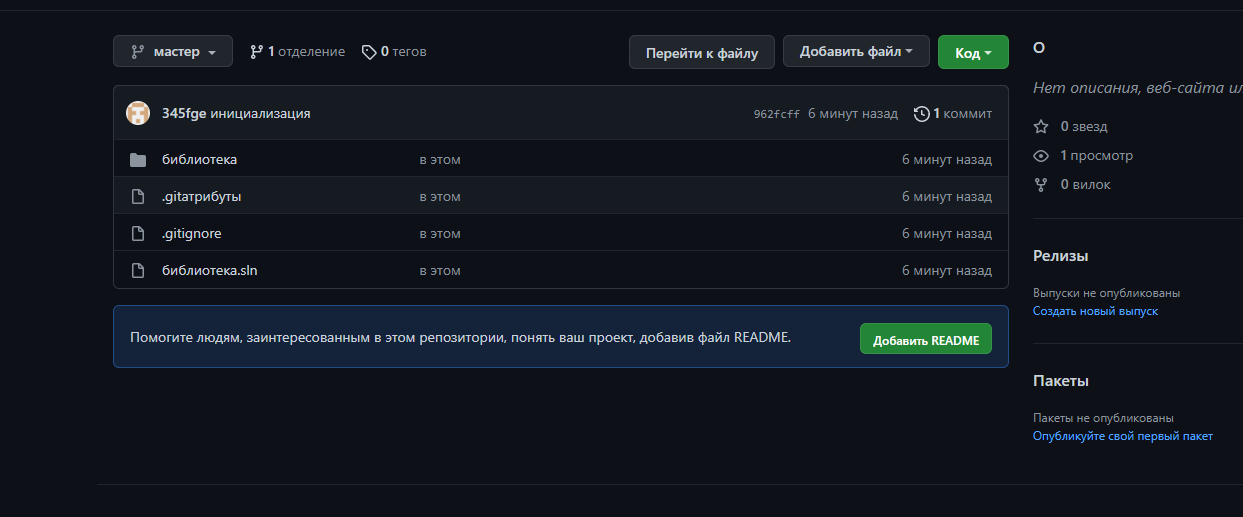
Одна из причин применения словарей ресурсов — определение одной или нескольких многократно используемых "обложек" приложения, которые можно применять к элементам управления. Еще одна причина связана с необходимостью сохранения содержимого, которое должно быть локализовано (такого как строки сообщений об ошибках).

**Стили** позволяют определить набор некоторых свойств и их значений, которые потом могут применяться к элементам в xaml. Стили хранятся в ресурсах и отделяют значения свойств элементов от пользовательского интерфейса. Также стили могут задавать некоторые аспекты поведения элементов с помощью триггеров. Аналогом стилей могут служить каскадные таблицы стилей (CSS), которые применяются в коде html на веб-страницах.

Стиль создается как ресурс с помощью объекта Style, который представляет класс **System.Windows.Style**. И как любой другой ресурс, он обязательно должен иметь ключ. С помощью коллекции **Setters** определяется группа свойств, входящих в стиль. В нее входят объекты **Setter**, которые имеют следующие свойства:

* **Property**: указывает на свойство, к которому будет применяться данный сеттер. Имеет следующий синтаксис: Property=" Свойство\_элемента".
* **Value:** устанавливает значение.

**Относительный путь к файлу от документа** — это путь к файлу относительно текущего документа. Такой адрес зависит от расположения файла, в котором он записан.



**Практическая работа №1.5**

**Тема**

Реализация авторизации пользователя.

**Цель работы**

Реализовать авторизацию в системе.

**Теоретический материал**

*Авторизация* — процесс предоставления пользователю или группе пользователей определенных разрешений, прав доступа и привилегий в компьютерной системе.

**Практическая работа №2**

**Тема**

Разработка тестового набора и тестового сценария для программного обеспечения.

**Цель работы**

Реализовать тестовый набор для авторизации пользователя в системе. Разработать тестовый сценарий и провести тестирование.

**Задание**

1. Изучите Пояснения к работе.
2. Реализуйте тестовый набор для авторизации в системе.
3. Разработайте тестовый сценарий.
4. Проведите тестирование.
5. Создайте подключение к базе данных и реализуйте отображение данных на форме.
6. Оформите отчет о проделанной работе.

**Теоретический материал**

**Правила написания тест-кейсов**

1. Заголовок:
   * должен быть чётким, кратким, понятным и однозначно характеризующим суть тест-кейса;
   * не может содержать выполняемые шаги и ожидаемый результат.
2. Предусловие:
   * может содержать полную информацию о состоянии системы или объекта, необходимом для начала выполнения шагов тест-кейса;
   * может содержать ссылки на информационные источники, которые необходимо изучить перед прохождением тест-кейса (инструкции, описание систем…);
   * не может содержать ссылки на тестируемый ресурс, если у информационной системы более одной среды (прод, тест, препрод…), данная информация должна быть вынесена в инструкцию, и ссылка приложена в предусловии;
   * не может содержать данные для авторизации, данная информация должна быть вынесена в инструкцию, и ссылка приложена в предусловии;
   * не может содержать выполняемые шаги и ожидаемый результат, если нам нужно, чтобы до выполнения шагов проверки у нас была открыта главная страница, то мы в предусловии указываем «открыта главная страница сайта»;
   * не может содержать ожидаемый результат.
3. Шаги проверки:
   * должны быть чёткими, понятными и последовательными;
   * следует избегать излишней детализации шагов. Правильно: «ввести в поле число 12».  
     Неправильно: «нажать на клавиатуре на цифру ‘1’, следующим шагом нажать на клавиатуре на цифру ‘2’»;
   * должны использоваться безличные глаголы.  
     Правильно: нажать, ввести, перейти.  
     Неправильно: нажмите, введите, идите;
   * не должно быть комментариев и пояснений, если есть необходимость привести мини-инструкцию, то оформляем инструкции в базе-знаний и в предусловии ссылаемся на неё;
   * не должно быть жёстко прописанных статических данных (логины, пароли, имена файлов) и примеров, для исключения эффекта пестицида.
4. Ожидаемый результат:
   * должен быть у каждого шага проверки;
   * должно быть кратко и понятно описано состояние системы или объекта, наступающее после выполнения соответствующего шага;
   * не должно быть избыточного описания.
5. Общие требования к тест-кейсам:
   * язык описания тест-кейсов должен быть понятен широкому кругу пользователей, а не узкой группе лиц;
   * тест-кейс должен быть максимально независим от других тест-кейсов и не ссылаться на другие тест-кейсы (лучшая практика, когда зависимостей нет вообще);
   * тест-кейсы группируются в функциональные блоки по их назначению;
   * в тест-кейсах проверяющих работу функционала скриншотов быть не должно, иначе вы будете посвящать сотни часов на изменение всех скриншотов в тысячах тест-кейсах при изменении интерфейса тестируемой программы. Скриншоты могут быть добавлены только в тест-кейсы проверяющие отображение страниц и форм.

На самом деле правила простые, однако их не так-то просто соблюдать. Если же придерживаться данных правил, то тест-кейсы будут легко поддерживаемыми, легко читаемыми, не будут вызывать отторжения и могут быть использованы всеми участниками команды в процессе разработки программного обеспечения.

**Тестовый документ**

**Аннотация теста**

|  |  |
| --- | --- |
| **Название проекта** |  |
| **Рабочая версия** |  |
| **Имя тестирующего** |  |
| **Дата(ы) теста** |  |

**Расшифровка тестовых информационных полей:**

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| **Название проекта** | Название тестируемого проекта |
| **Рабочая версия** | Версия проекта/программного обеспечения (первый тест считается 1.0). |
| **Имя тестирующего** | Имя того, кто проводил тесты |
| **Дата(ы) теста** | Дата(ы) проведения тестов – это один или несколько дней. Если тесты проводились в более протяженный период времени, нужно отметить отдельную дату для каждого теста. |
| **Тестовый пример #** | Уникальный ID для каждого тестового примера. Следуйте некоторым конвенциям, чтобы указать типы тестов. Например,‘TC\_UI\_1′ означает‘user interface test case #1′ ( ТС\_ПИ\_1: тестовый случай пользовательского интерфейса#1) |
| **Приоритет тестирования**  *(Низкий/Средний/Высокий)* | Насколько важен каждый тест. Приоритет тестирования для бизнес-правил и функциональных тестовых случаев может быть средним или высоким, в то время как незначительные случаи пользовательского интерфейса могут иметь низкий приоритет. |
| **Заголовок/название теста** | Название тестового случая. Например, Подтвердите страницу авторизации с действительным именем пользователя и паролем. |
| **Краткое изложение теста** | Описание того, что должен достичь тест. |
| **Этапы теста** | Перечислите все этапы теста подробно. Запишите этапы теста в том порядке, в котором они должны быть реализованы. Предоставьте как можно больше подробностей и разъяснений. Пронумерованный список – хорошая идея. |
| **Тестовые данные** | Перечислите/опишите все тестовые данные, используемые для данного тестового случая. Так, фактические используемые входные данные можно отслеживать по результатам тестирования. Например, Имя пользователя и пароль для подтверждения входа. |
| **Ожидаемый результат** | Каким должен быть вывод системы после выполнения теста? Подробно опишите ожидаемый результат, включая все сообщения/ошибки, которые должны отображаться на экране. |
| **Фактический результат** | Каким должен быть фактический результат после выполнения теста? Опишите любое релевантное поведение системы после выполнения теста. |
| **Предварительное условие** | Любые предварительные условия, которые должны быть выполнены до выполнения теста. Перечислите все предварительные условия для выполнения этого тестового случая. |
| **Постусловие** | Каким должно быть состояние системы после выполнения теста? |
| **Статус**  *(Зачет/Незачет)* | Если фактический результат не соответствует ожидаемому результату, отметьте тест как неудачный. В ином случае обновление пройдено. |
| **Примечания/комментарии** | Используйте эту область для любых дополнительных заметок/комментариев/вопросов. Эта область предназначена для поддержки вышеуказанных полей (например, если есть некоторые особые условия, которые не могут быть описаны в любом из вышеуказанных полей, или если есть вопросы, связанные с ожидаемыми или фактическими результатами). |

**Тестовый пример #1:**

|  |  |
| --- | --- |
| **Тестовый пример #** |  |
| **Приоритет тестирования** |  |
| **Заголовок/название теста** |  |
| **Краткое изложение теста** |  |
| **Этапы теста** |  |
| **Тестовые данные** |  |
| **Ожидаемый результат** |  |
| **Фактический результат** |  |
| **Статус** |  |
| **Предварительное условие** |  |
| **Постусловие** |  |
| **Примечания/комментарии** |  |

**Выполнение работы**

1. Изучите раздел «Пояснение к работе».
2. Выполните задание.
3. Ответьте на контрольные вопросы.

**Содержание отчета**

Отчет по практической работе должен содержать:

1. Цель работы.
2. Раздел с заданием, в котором расписывается о том, каким образом студент выполнил его, с использованием каких средств, инструментов и технологий. Представленная информация подтверждается скриншотами экрана, файлами, полученными в ходе выполнения практической работы.
3. Вывод.

**Контрольные вопросы**

1. Что такое тест-кейс?
2. Перечислите общие требования к тест-кейсам.

**Практическая работа №3**

**Тема**

Реализация главного окна навигации с использованием страниц.

**Цель работы**

Создать главное окно с возможностью навигации по страницам.

**Теоретический материал**

Чтобы создать страничное приложение в WPF, нужно использовать класс System.Windows.Controls.Page.

Модель для создания страниц в WPF во многом похожа на модель для создания окон. Хотя создавать объекты страниц можно и с помощью одного лишь кода, обычно для каждой страницы создается файл XAML и файл отделенного кода. При компиляции этого приложения компилятор создает производный класс страницы, который объединяет написанный разработчиком код с генерируемыми автоматически связующими элементами (такими как поля, которые ссылаются на каждый именованный элемент на странице).

Хотя страницы и являются самым высокоуровневым компонентом пользовательского интерфейса при проектировании приложения, во время его выполнения контейнером наивысшего уровня они уже не будут. Вместо этого они обслуживаются в другом контейнере. Именно в этом и состоит секрет гибкости, обеспечиваемой WPF в случае страничных приложений, ведь в качестве такого контейнера WPF позволяет использовать любой из нескольких следующих объектов:

* объект NavigationWindow, который представляет собой немного видоизмененную версию класса Window;
* объект Frame, находящийся внутри другого окна;
* объект Frame, находящийся внутри другой страницы;
* объект Frame, обслуживаемый непосредственно в Internet Explorer или Firefox.

**Класс Page**

Класс Page допускает наличие только единственного вложенного элемента. Однако класс Page не является элементом управления содержимым: он на самом деле унаследован непосредственно от класса FrameworkElement. Вдобавок класс Page является более простым и отлаженным, чем класс Window. Он имеет небольшой набор дополнительных свойств, которые позволяют настраивать его внешний вид, взаимодействовать с контейнером только определенным, ограниченным образом и применять навигацию. Все эти свойства перечислены ниже:

Background - принимает кисть, которая позволяет устанавливать заливку для фона

Content - принимает один элемент, который отображается на странице. Обычно в роли такого элемента выступает контейнер компоновки, такой как Grid или StackPanel

Foreground, FontFamily и FontSize - определяют внешний вид по умолчанию для текста внутри страницы. Значения этих свойств наследуются элементами внутри страницы. Например, если устанавливается заливка переднего плана и размер шрифта, по умолчанию содержимое внутри страницы получает эти же настройки

WindowWidth, WindowHeight и WindowTitle - определяют внешний вид окна, в которое упаковывается страница. Эти свойства позволяют управлять хостом путем установки его ширины, высоты и заголовка. Однако они действуют только в том случае, если страница обслуживается в окне (а не во фрейме)

NavigationService - возвращает ссылку на объект NavigationService, которую можно использовать для отправки пользователя на другую страницу программным путем

KeepAlive - определяет, должен ли объект страницы оставаться действующим после перехода пользователя на другую страницу.

ShowsNavigationUI - определяет, должен ли хост для данной страницы отображать навигационные элементы управления (кнопки "назад" и "вперед"). По умолчанию имеет значение true.

Title - устанавливает имя, которое должно применяться для страницы в хронологии навигации. Хост не использует свойство Title для установки заголовка в строке заголовка: для этой цели у него есть свойство WindowTitle.

Также важно обратить внимание на отсутствующие компоненты — в классе Page нет эквивалентов для методов Hide() и Show(), доступных в классе Window. Если потребуется показать другую страницу, придется воспользоваться навигацией.

**MVVM (Model-View-ViewModel)** — способ организации кода. Он помогает отделить пользовательский интерфейс от логики.

**Паттерн MVVM**

**View** содержит только код пользовательского интерфейса. Он отображает экран (текстовые поля, ярлыки, кнопки) и пользовательский ввод. Например, в [Windows Forms](https://ru.wikipedia.org/wiki/Windows_Forms) это обычно Form или Control. В [Windows Presentation Foundation](https://ru.wikipedia.org/wiki/Windows_Presentation_Foundation) (WPF) — это класс Window: файлы XAML и xaml.cs.

**Model** — это классы. Это «вещи» в вашей программе. Например, в программе ввода заказов вашими классами моделей могут быть: Customer, Order, InventoryItem. Эти классы будут содержать логику, необходимую для выполнения функций.

**ViewModel** используются для связи между View и Model. Они обычно не содержат много логики. Они содержат Model, который должен быть использован во View. Когда человек взаимодействует с пользовательским интерфейсом, ViewModel узнает, какое действие необходимо выполнить, и просит Model сделать это.

**Как работает паттерн MVVM?**

Как работает паттерн MVVM можно разобрать на примере из реального мира. Представьте, что у нас есть владелец интернет-магазина, веб-мастер и контент-менеджер:

* владелец интернет-магазина (Model) — занимается своей работой, ни на что не отвлекаясь. Если начинается важная акция, он передает веб-мастеру, что нужно добавить новость о ней в блог;
* веб-мастер (ViewModel) – получает информацию от владельца интернет-магазина и передает ее контент-менеджеру;
* контент-менеджер (View) — пишет новость в блог, основываясь на данных, которые предоставил веб-мастер, и публикует ее.

**Примеры использования MVVM**

Основным примером использования MVVM является **программирование графического интерфейса пользователя (GUI)**. Он используется для простого событийно-управляемого программирования пользовательских интерфейсов путем отделения View от логики бэкенда, управляющей данными.

В WPF View проектируется с помощью языка разметки XAML. Файлы XAML привязываются к ViewModel. Таким образом View отвечает только за представление, а ViewModel — только за управление состоянием приложения.

MVVM очень широко используется в JavaScript-библиотеке Knockout.js.

**Практическая работа №4**

**Тема**

Отображение данных на форме.

**Цель работы**

Реализовать отображение данных базы данных на форме.