|  |  |  |
| --- | --- | --- |
| Департамент образования Ярославской области  Государственное профессиональное образовательное автономное учреждение  Ярославской области  «ЯРОСЛАВСКИЙ ПРОМЫШЛЕННО-ЭКОНОМИЧЕСКИЙ КОЛЛЕДЖ  им. Н.П. ПАСТУХОВА» | | |
| **Отчет По учебной практике**  **по профессиональному модулю ПМ.05**  **Соадминистрирование и автоматизация баз данных и серверов** | | |
| ОУП ПМ.05.09.02.07.19П1.22 | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | Студент  \_\_\_\_\_\_\_\_\_\_\_\_\_ И.О. Фамилия  «\_\_\_\_» декабрь 2019 г. |
|  | Преподаватель  \_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.В. Маянцева  «\_\_\_\_» декабрь 2019 г. |
| 2020 | | |

**Содержание**

[**Введение** 3](#_Toc119329170)

[**Практическая работа №1.1** 5](#_Toc119329171)

[**Практическая работа №1.2** 10](#_Toc119329172)

[**Практическая работа №1.3** 12](#_Toc119329173)

[**Практическая работа №1.4** 14](#_Toc119329174)

[**Практическая работа №1.5** 16](#_Toc119329175)

[**Практическая работа №2** 17](#_Toc119329176)

[**Практическая работа №3** 20](#_Toc119329177)

**Введение**

Данные методические указания предназначены для студентов 4 курса специальности 09.02.07 «Информационные системы и программирование».

Методические указания содержат:

* требования к знаниям и умениям студентов;
* правила выполнения лабораторных работ (практических заданий);
* содержание лабораторных работ (практических заданий);
* контрольные вопросы;
* список использованной литературы.

В результате выполнения данного курса работ студент должен:

*Профессиональные компетенции*

ПК 7.1. Выявлять технические проблемы, возникающие в процессе эксплуатации баз данных и серверов

ПК 7.2. Осуществлять администрирование отдельных компонент серверов.

ПК 7.3. Формировать требования к конфигурации локальных компьютерных сетей и серверного оборудования, необходимые для работы баз данных и серверов.

ПК 7.4. Осуществлять администрирование баз данных в рамках своей компетенции.

ПК 7.5. Проводить аудит систем безопасности баз данных и серверов, с использованием регламентов по защите информации.

*Общие компетенции*

ОК 1. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам.

ОК 2. Использовать современные средства поиска, анализа и интерпретации информации, и информационные технологии для выполнения задач профессиональной деятельности.

ОК 3. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях.

ОК 4. Эффективно взаимодействовать и работать в коллективе и команде

ОК 5. Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 6. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей, в том числе с учетом гармонизации межнациональных и межрелигиозных отношений, применять стандарты антикоррупционного поведения.

ОК 7. Содействовать сохранению окружающей среды, ресурсосбережению, применять знания об изменении климата, принципы бережливого производства, эффективно действовать в чрезвычайных ситуациях.

ОК 8. Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности.

ОК 9. Пользоваться профессиональной документацией на государственном и иностранном языках.

|  |  |  |
| --- | --- | --- |
| *Код*  *ПК, ОК* | *Умения* | *Знания* |
| ОК 1, 2, 3, 4, 5, 6, 7,8,9  ПК 7.1., 7.2., 7.3., 7.4., 7.5. | У1. проектировать и создавать базы данных;  выполнять запросы по обработке данных на языке SQL;  У2. осуществлять основные функции по администрированию баз данных;  У3. разрабатывать политику безопасности SQL сервера, базы данных и отдельных объектов базы данных;  У4. владеть технологиями проведения сертификации программного средства. | З1. модели данных, основные операции и ограничения;  З2. технологию установки и настройки сервера баз данных;  З3. требования к безопасности сервера базы данных;  З4. государственные стандарты и требования к обслуживанию баз данных |

**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:

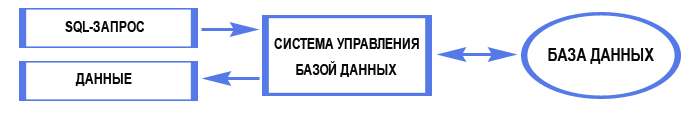


Рисунок 1. Схема работы с базой данных

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).

**Практическая часть**

В ходе выполнения данной практической работы при помощи программного продукта в MS SQl Server была создана БД и представлены скрипты данной БД(Рисунки 3 - 21), на основе ERD диаграммы (Рисунок 2).

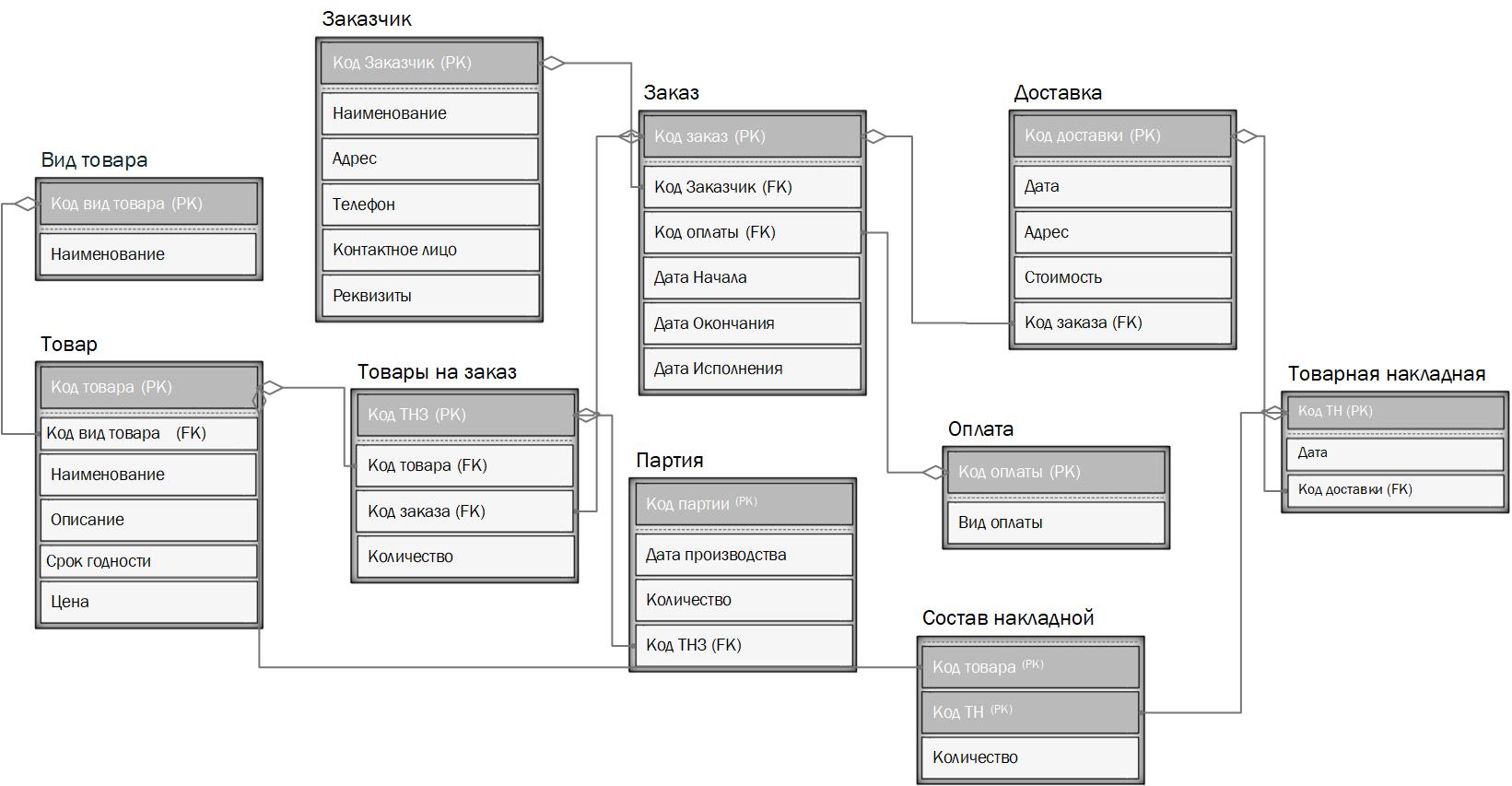


Рисунок 2. ERD диаграмма

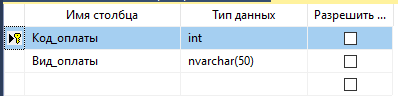


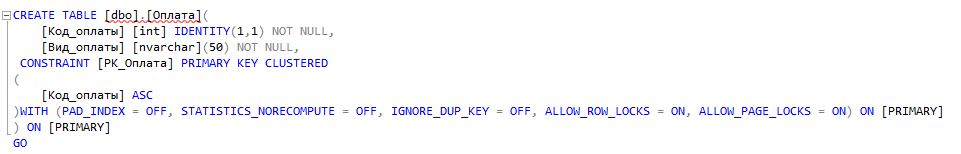
Рисунок 3. Таблица «Оплата» 

Рисунок 4. Скрипт к таблице Оплата

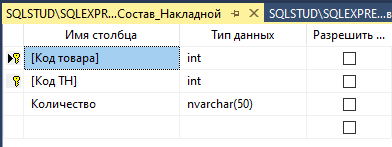


Рисунок 5. Таблица «Состав накладной»

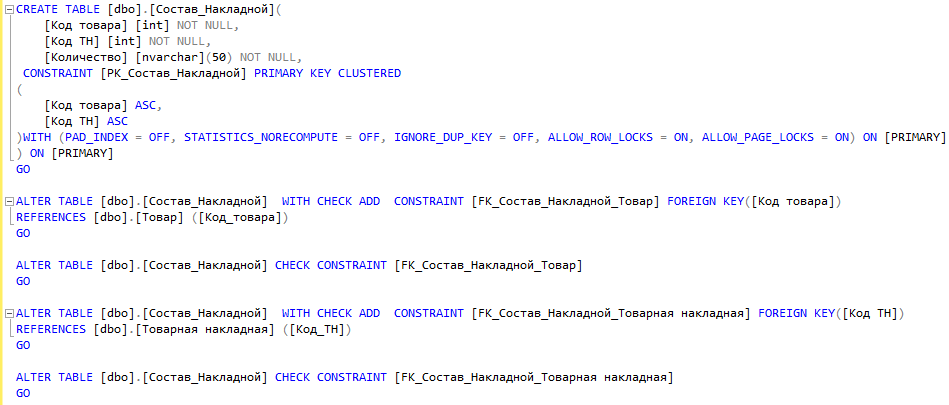


Рисунок 6. Скрипт к таблице Состав накладной

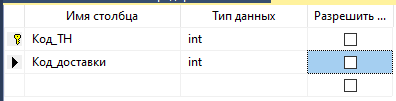


Рисунок 7. Таблица "Товарная накладная"

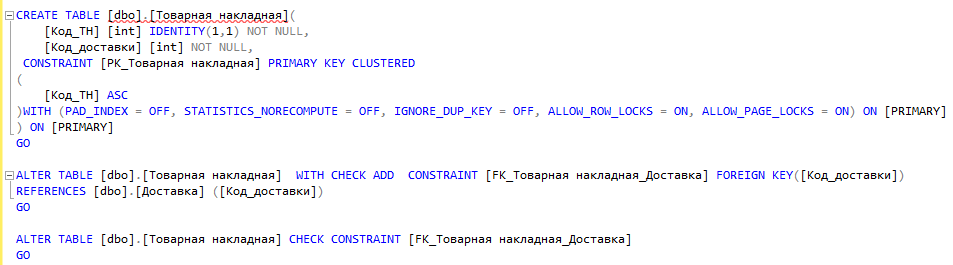


Рисунок 8. Скрипт к таблице Товарная накладная

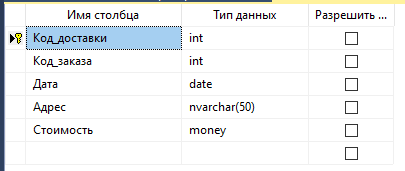


Рисунок 9.Таблица "Доставка"

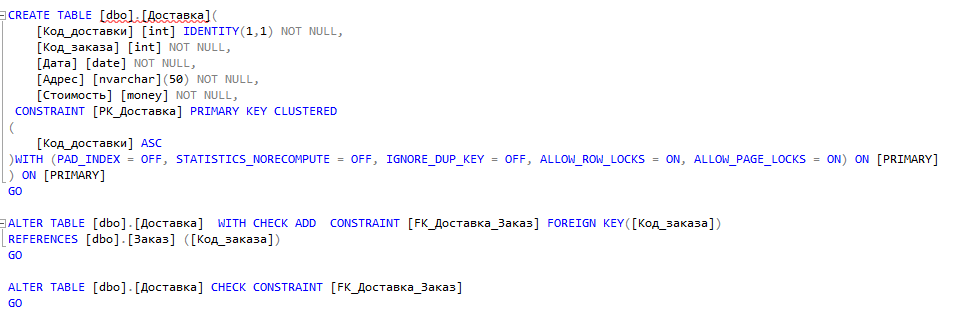


Рисунок 10. Скрипт к таблице Доставка

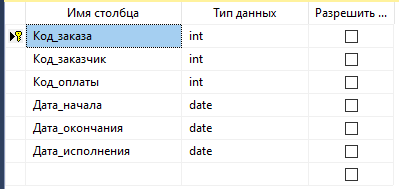


Рисунок 11. Таблица "Заказ"

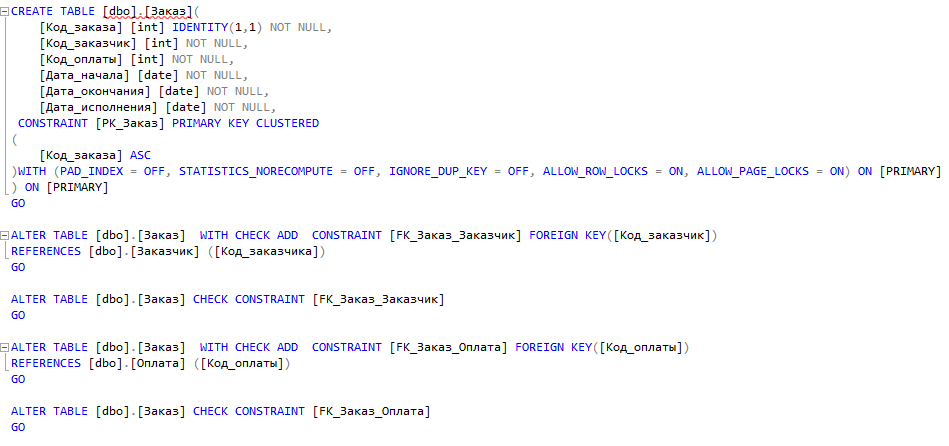


Рисунок 12. Скрипт к таблице Заказ

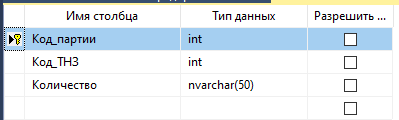


Рисунок 13. Таблица "Партии"

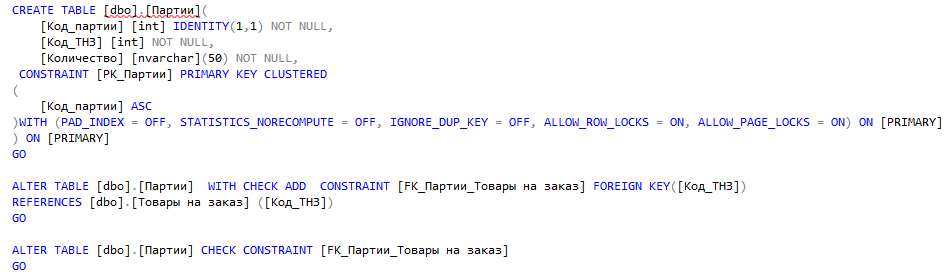


Рисунок 14. Скрипт к таблице Партии

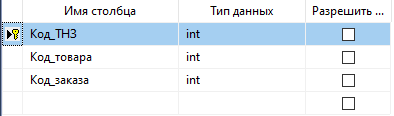


Рисунок 15. Таблица "Товары на заказ"

****

Рисунок 16. Скрипт Товары на заказ

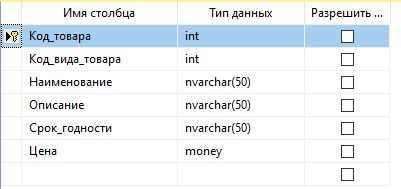


Рисунок 17. Таблица "Товар"

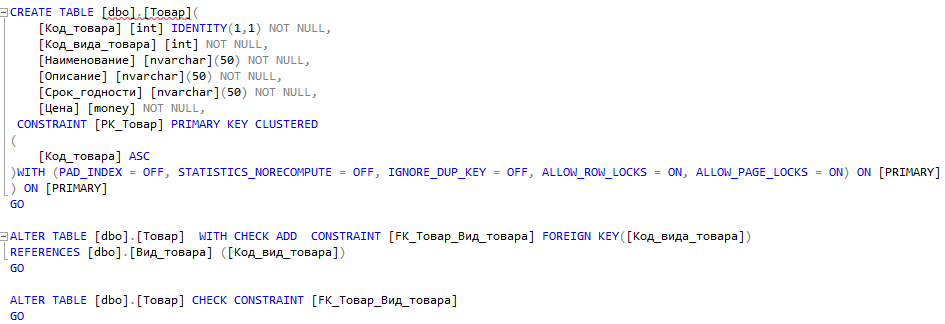


Рисунок 18. Скрипт к таблице Товар

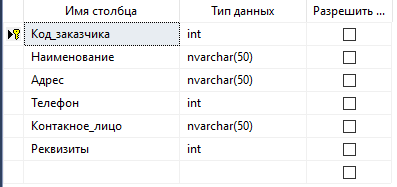


Рисунок 19. Таблица "Заказчик"

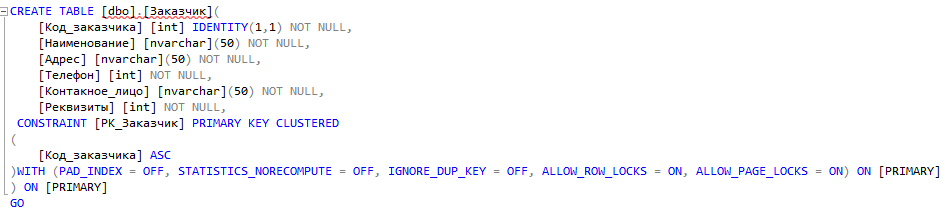


Рисунок 20. Скрипт к таблице Заказчик

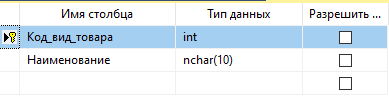


Рисунок 21. Таблица "Вид товара"

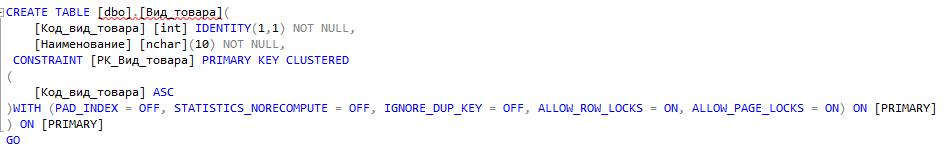


Рисунок 22. Скрипт к таблице Вид товара

Следующим шагом был построена диаграммы базы данных, которая представлена на рисунке 23.

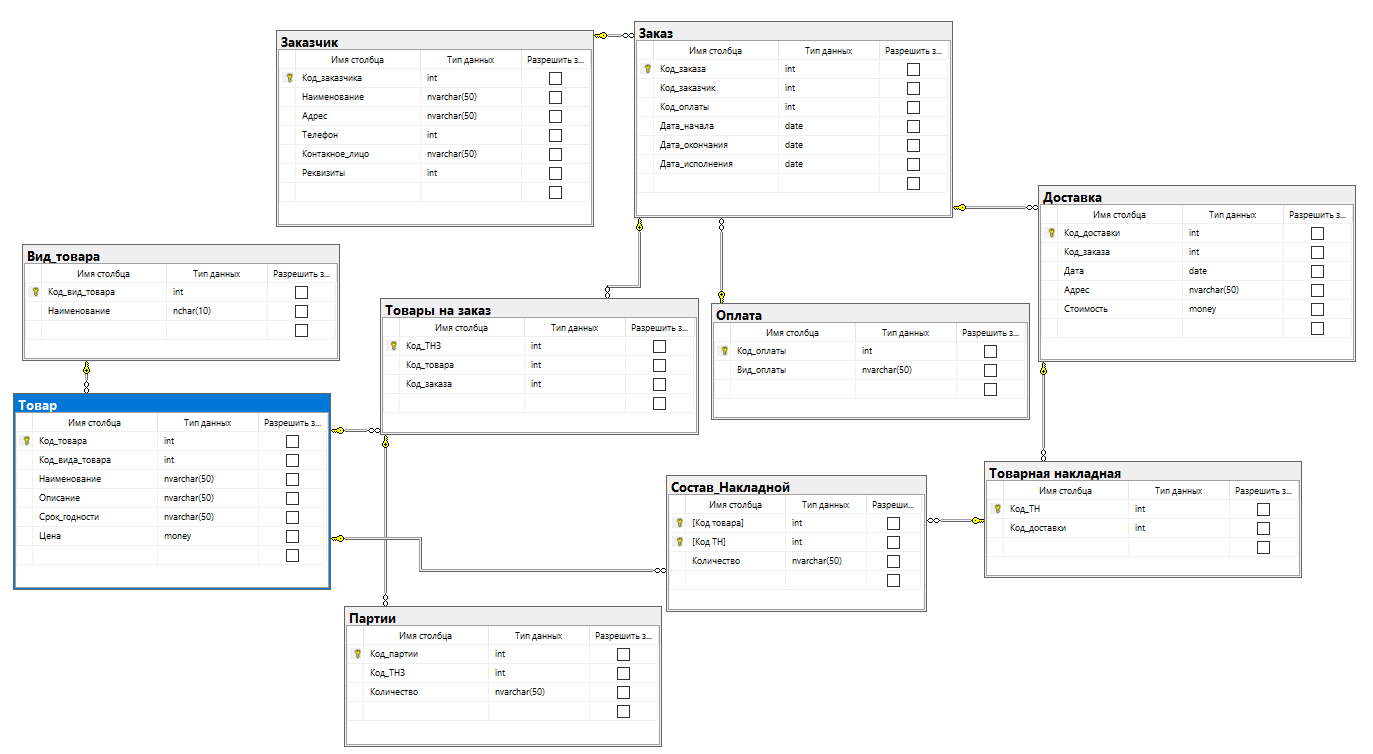


Рисунок 23. Диаграмма Базы Данных

После создания базы данных и диаграммы необходимо было заполнить таблицы первичными данными. Результат выполнения работ представлен на рисунках 24 – 33.

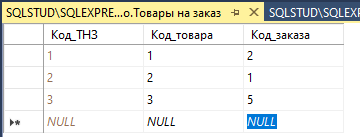


Рисунок 24. Таблица Товары на заказ

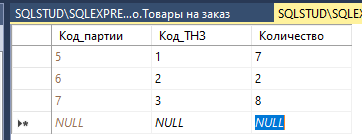


Рисунок 25. Таблица Партии

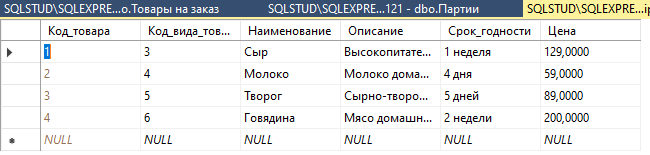


Рисунок 26. Таблица Товар

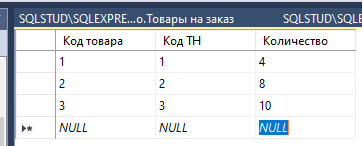


Рисунок 27. Таблица Состав накладной

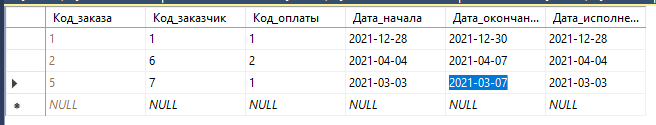


Рисунок 28. Таблица Заказ

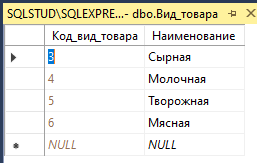


Рисунок 29. Таблица Вид товара

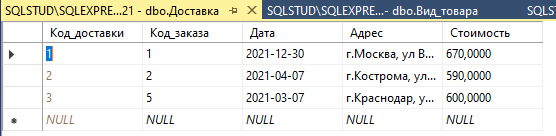


Рисунок 30. Таблица Доставка

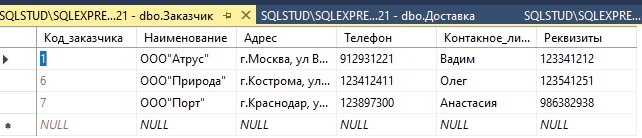


Рисунок 31. Таблица Заказчик

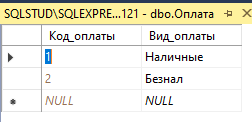


Рисунок 32. Таблица Оплата

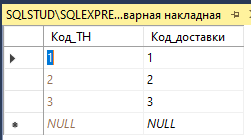


Рисунок 33. Таблица Товарная накладная

**Вывод:** Входе выполнения практической работы по заданной схеме данных была разработана база данных, создан и сохранен скрипт с БД.

**Контрольные вопросы**

1. Что такое «база данных»?

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

1. Что такое первичный ключ (Primary key)?

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

1. Что такое внешний ключ (Foreign key)?

Внешний ключ – это столбец, или несколько столбцов, ссылающихся на первичный ключ другой таблицы.

1. Какие связи существуют в реляционных базах данных?

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).

1. Что подразумевается под целостностью данных?

Целостность базы данных (англ. database integrity) — соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам.

1. Что такое нормализация и каковы ее преимущества?

Нормализация – это метод проектирования базы данных, который позволяет привести базу данных к минимальной избыточности.

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.

**Практическая часть:**

Открыли среду разработки Visual Studio и создали новый проект «WPF». В качестве названия проекта указали наименование своей предметной области + «IS» (UZPIS).

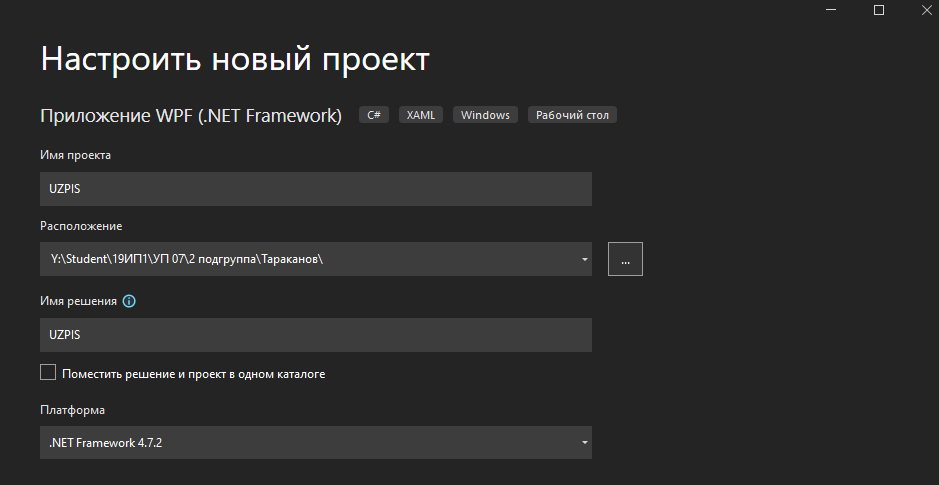
****

Рисунок 34. Создание проекта WPF

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» и из списка элементов выберем элемент «Модель ADO.NET EDM». Задали данному элементу название.

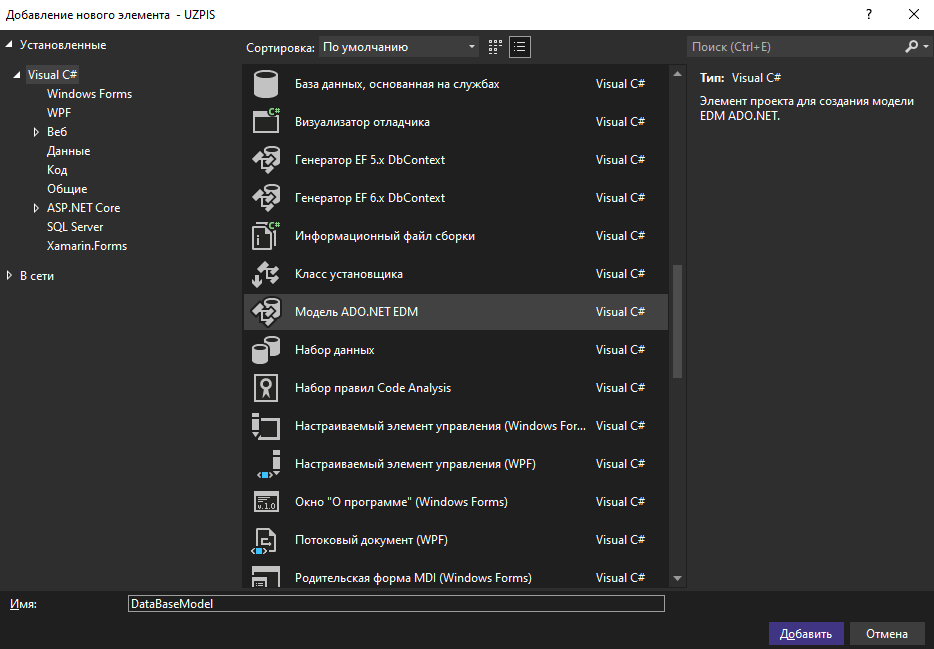


Рисунок 35. Создание нового элемента «Модель ADO.NET EDM».

Откроется мастер создания моделей EDM. Выбираем «Конструктор EF из базы данных» и жмем кнопку «Далее». В следующем окне нажимаем на кнопку «Создать соединение» (Рисунок 26). В окне «Свойства подключения» укажите актуальные для вас данные подключения. Поменяли «Проверка подлинности Windows» на «Проверка подлинности SQL»

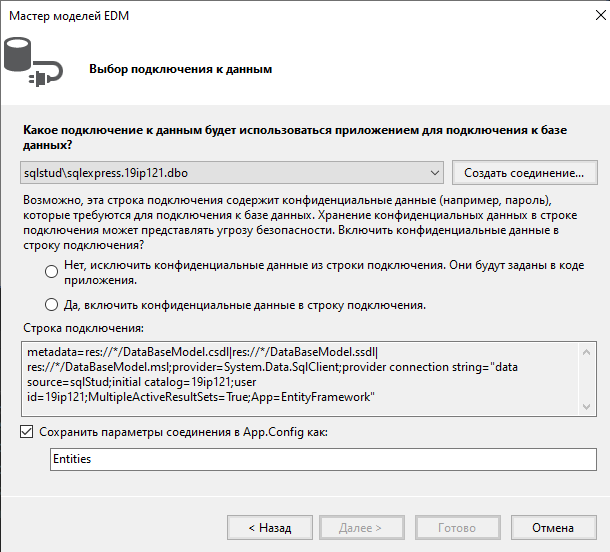


Рисунок 36. Мастер моделей EDM, создание соединения с БД

Следующим шагом указали необходимые таблицы. Пространство имен может указать свое название, либо оставить без изменений. (рисунок 37).

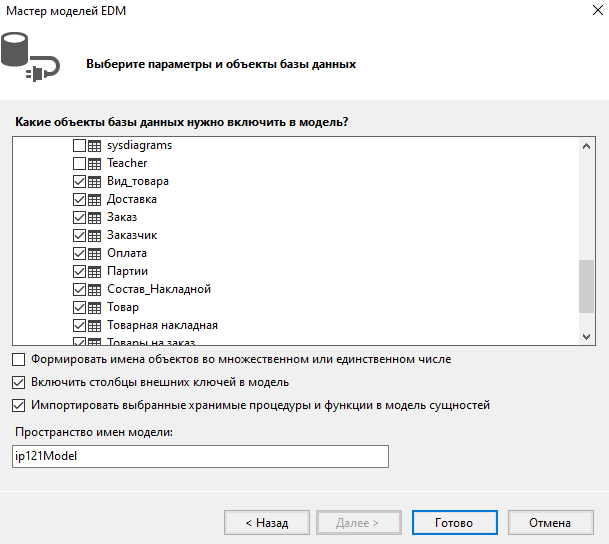


Рисунок 37Мастер моделей EDM, выбор необходимых таблиц

После завершения работы с мастером, система показала нам схему данных базы данных (Рисунок 38).

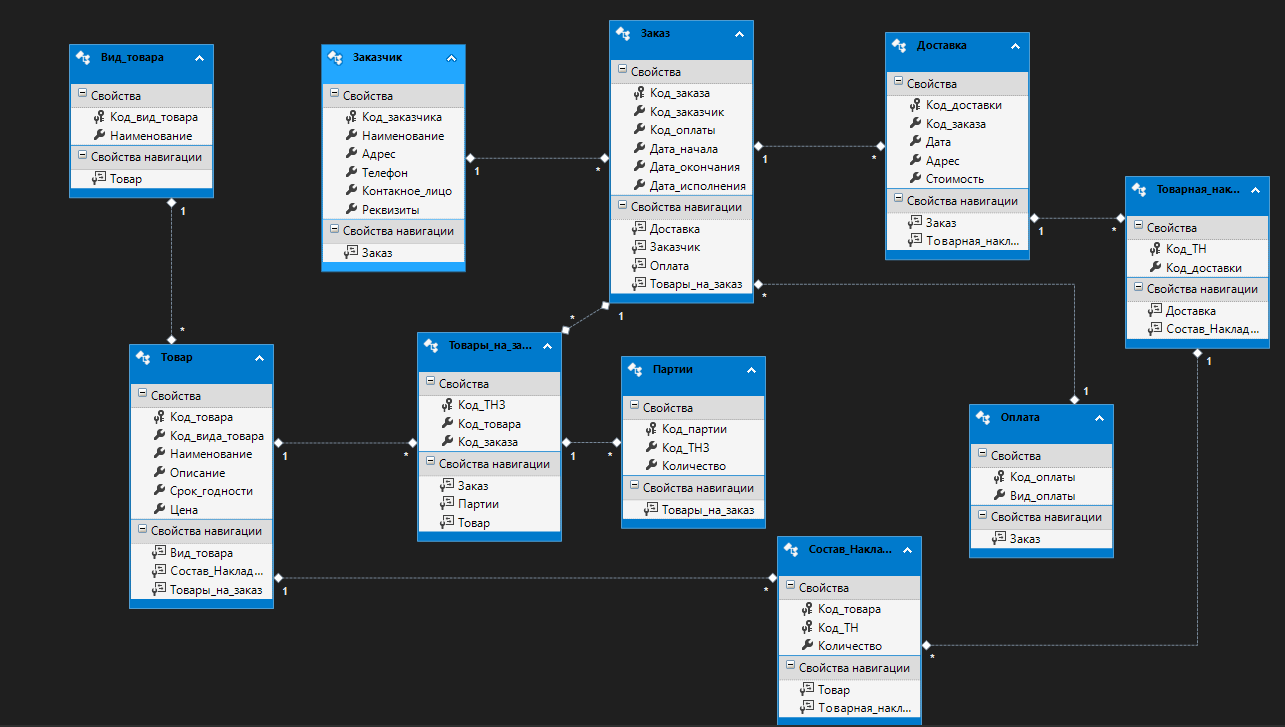


Рисунок 38. Отражение связей между сущностями.

Для возможности работы с базой данных необходимо создать класс и в этом классе дописать код. Результат выполненной работы представлен на рисунках 39 – 40.

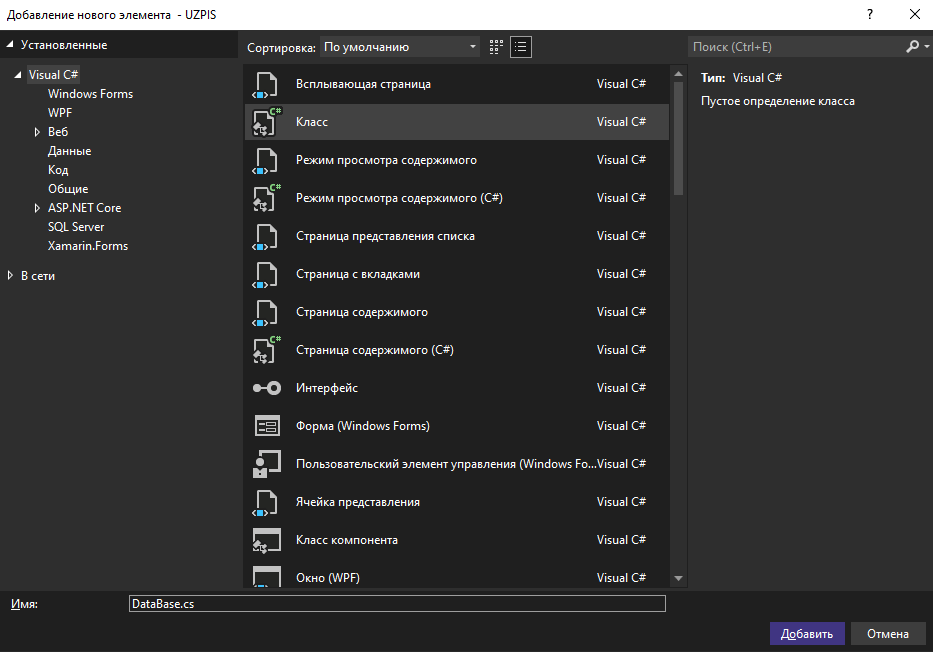


Рисунок 39. Добавление нового элемента. Класс

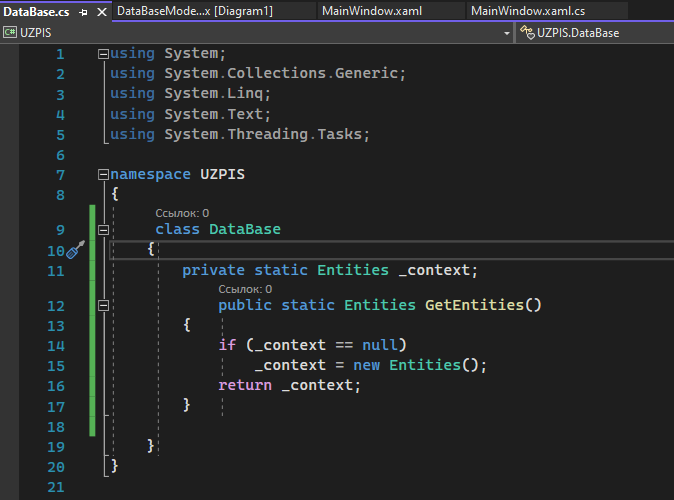


Рисунок 40. Публичный статичный метод

**Вывод:** Входе выполнения практической работы по заданной схеме данных была разработана база данных, создан и сохранен скрипт с БД, а также реализовано программное приложение с возможностью авторизации в системе.

**Контрольные вопросы**

1. Как создать подключение к базе данных?

Откроем среду разработки Visual Studio и создадим новый проект «WPF». В качестве названия проекта укажите наименование вашей предметной области + «IS». Нажмите кнопку «Ок». 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». Из списка элементов выберем элемент «Модель ADO.NET EDM». Зададим данному элементу название. Откроется мастер создания моделей EDM. Выбираем «Конструктор EF из базы данных» и жмем кнопку «Далее». В следующем окне нажимаем на кнопку «Создать соединение». В окне «Свойства подключения» укажите актуальные для вас данные подключения. Поменяйте «Проверка подлинности Windows» на «Проверка подлинности SQL». Выбор версии оставьте без изменений. В следующем окне укажите необходимые таблицы. Пространство имен может указать свое название, либо оставить без изменений. После завершения работы с мастером, система покажет вам схему данных вашей базы данных. Обратите внимание, как здесь отражены связи между сущностями (например, в связях «многие ко многим» отсутствует промежуточная таблица, которая связывала две сущности). Теперь нужно реализовать возможность работать с базой данных непосредственно из кода. Для этого создадим новый класс, например, с названием DataBase. В классе нам необходимо немного дописать код, чтобы с помощью данного класса мы могли иметь доступ к базе данных. Для этого: необходимо объявить статичную приватную переменную, которая будет являться объектом класса «Entities». Написать публичный статичный метод, который будет возвращать нам данный объект. В случае, если этот объект не был создан, необходимо создать его.

**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 часть:**

Для выполнения практической работы необходимо было создать руководство по стилю, данная работы была выполнения в MS Word, результат проделанной работы представлен на Рисунке 41.

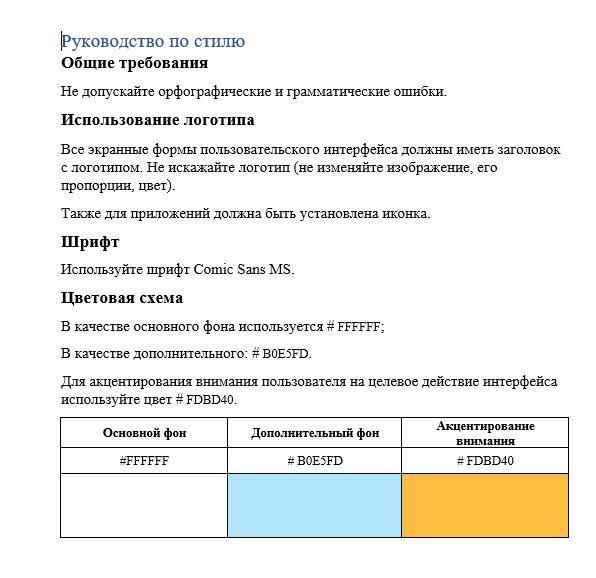


Рисунок 41. Руководство по стилю

Далее был найден логотип и иконка для будущего приложения (Рисунок 42)



Рисунок 42. Логотип и иконка для приложения

Следующим шагом было размещение файлов в репозитории. Для этого необходимо создать новый репозиторий с названием PR1\_3 (Рисунок 43)

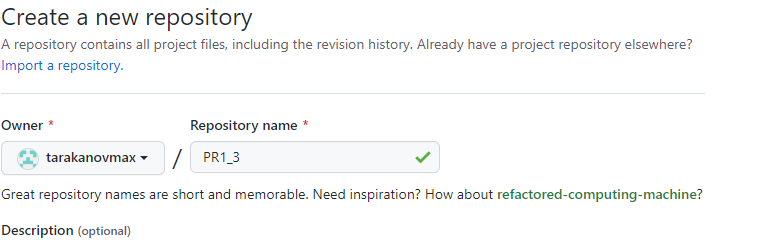
****

Рисунок 43. Репозиторий PR1\_3

На рисунках 44 – 45 представлена код, написанный в Git Bush для инициализации новой переменной, данных о пользователе, создания файла RADME.md, создание коммитов и подключении к удаленному доступу к созданному репозиторию в GitHub.

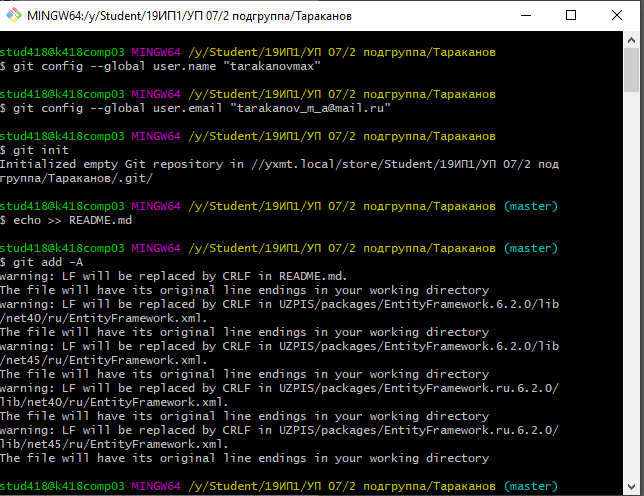
****

Рисунок 44. Код в Git Bush. Фрагмент 1

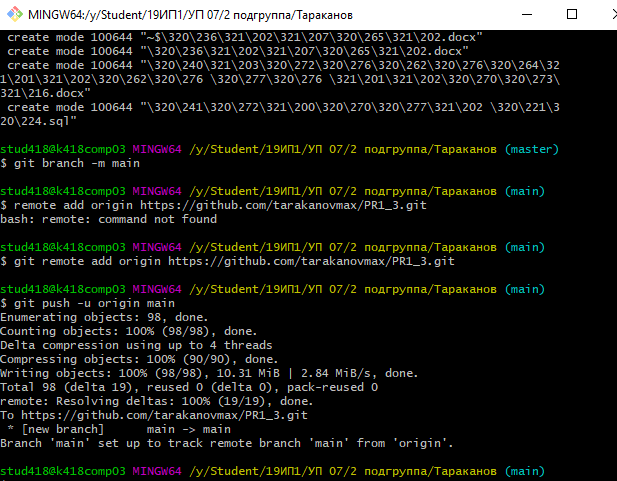
****

Рисунок 45. Код в Git Bush. Фрагмент 2

Результат выполнения проделанной работы представлен на рисунке 46.

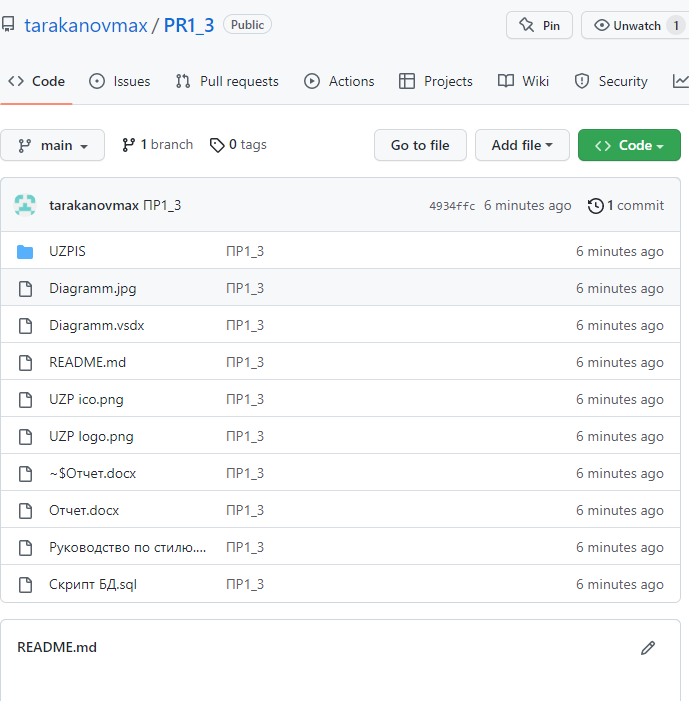
****

Рисунок 46. Файлы в репозитории на GitHub

**Вывод:** Входе выполнения практической работы был размещен скрипт с базой данных, руководство по стилю, логотип и иконку в удаленный репозиторий проекта.

**Контрольные вопросы**

1. В чем заключается экономия времени при использовании системы контроля версий?

Если умрет любая копия репозитория, то это не приведет к потере кодовой базы, поскольку она может быть восстановлена с компьютера любого разработчика. Каждая копия является полным бэкапом данных.

1. В чем преимущества использования системы контроля версий?

Полная история изменений каждого файла за длительный период. Это касается всех изменений, внесенных огромным количеством людей за долгие годы. Изменением считается создание и удаление файлов, а также редактирование их содержимого. Различные инструменты VCS отличаются тем, насколько хорошо они обрабатывают операции переименования и перемещения файлов. В историю также должны входить сведения об авторе, дата и комментарий с описанием цели каждого изменения. Наличие полной истории позволяет возвращаться к предыдущим версиям, чтобы проводить анализ основных причин возникновения ошибок и устранять проблемы в старых версиях программного обеспечения. Если над программным обеспечением ведется активная работа, то «старой версией» можно считать почти весь код этого ПО.

Ветвление и слияние. Эта возможность полезна не только при одновременной работе участников команды: отдельные люди также могут извлечь из нее пользу и работать над несколькими независимыми направлениями. Создание «веток» в инструментах VCS позволяет иметь несколько независимых друг от друга направлений разработки, а также выполнять их слияние, чтобы разработчики могли проверить, что изменения, внесенные в каждую из веток, не конфликтуют друг с другом. Многие команды разработчиков программного обеспечения создают отдельные ветки для каждой функциональной возможности, для каждого релиза либо и для того, и для другого. Наличие множества различных рабочих процессов позволяет командам выбирать подходящий для них способ использования ветвления и слияния в VCS.

Отслеживаемость. Возможность отслеживать каждое изменение, внесенное в программное обеспечение, и связывать его с ПО для управления проектами и отслеживания ошибок, например Jira, а также оставлять к каждому изменению комментарий с описанием цели и назначения изменения может помочь не только при анализе основных причин возникновения ошибок, но и при проведении другого анализа. История с комментариями во время чтения кода помогает понять, что этот код делает и почему действие реализовано именно таким образом. Благодаря этому разработчики могут вносить корректные и совместимые изменения в соответствии с долгосрочным планом разработки системы. Это особенно важно для эффективной работы с унаследованным кодом, поскольку дает разработчикам возможность точнее оценить объем дальнейшей работы.

1. Что такое Git?

Git (произносится «гит») — распределённая система управления версиями.

1. Как начать использовать Git?

* Установи GIT;
* Настрой Git, так Git связывает Ваш аккаунт и Ваши изменения;
* Настрой Git для Windows;
* Создай локальный GIT репозиторий для своего продукта;
* Подключи стандартные каталоги;
* Создай файл в локальном репозитории;
* Свяжи новый файл с удаленным репозиторием;
* Отправь код на удаленный репозиторий.

1. Основные (наиболее часто используемые) команды Git.

git status - можно посмотреть текущую ветку, список измененных файлов, а также файлы которые нужно добавить под версионный контроль

git add имя\_файла - добавить файл под версионный контроль

git add -A - добавить все файлы

git commit -m "Text message" - зафиксировать изменения

git push - вытолкнуть зафиксированные изменения

git pull - притянуть все изменения из репозитория

1. Какие сервисы существуют для Git?

GitHub; Bitbucket; Directual; GitLab; Beanstalk.

1. Как работать с локальным репозиторием?

* Откройте свой Git Bash и перейдите к хранилищу, которое необходимо связать. First\_Project\_Repo
* Проверьте, чист ли репозиторий, используя команду git status.
* Проверьте, чист ли репозиторий, используя команду git status.
* Выполните команду git remote
* Поскольку связанного репозитория не существует, из Git не было получено никаких выходных данных.
* Теперь с помощью приведенного выше URL-адреса мы свяжем репозиторий. Чтобы связать репозиторий, выполните следующую команду и нажмите клавишу enter:
* git remote add origin <https://github.com/harishrajora805/myFirstRepo.git>. git\_remote\_add\_origin
* Как только это будет сделано, локальный репозиторий будет связан с репозиторием GitHub. Примечание: пожалуйста, используйте свой собственный URL-адрес репозитория для ссылки.
* Как проверить, подключен ли локальный репозиторий к удаленному репозиторию?
* Чтобы проверить, связали ли мы наш репозиторий или нет, снова выполните команду git remote: git\_remote\_origin
* Как видно, исходный репозиторий доступен. Продолжайте и используйте команду git remote-v для просмотра того же результата вместе с URL-адресом, как показано на рисунке.

1. Как работать с распределенным репозиторием?

**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 часть:**

После того, как создали проект, необходимо разработать его структуру. Для этого создали в проекте директорию *«Resources»* (Рисунок 47).

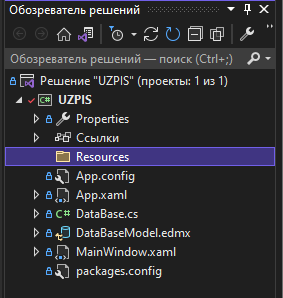
**

Рисунок 47. Созданная директория «Resources». в проекте

Внутри данного каталога были созданы еще три каталога: *«Icons», «Images», «Themes»*. Также в эти каталоги добавили иконку и логотип в соответствующие каталоги и задали для форм проекта добавленную иконку. (Рисунок 48 - 49)

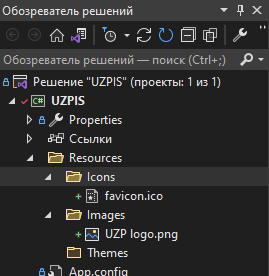


Рисунок 48. Каталоги

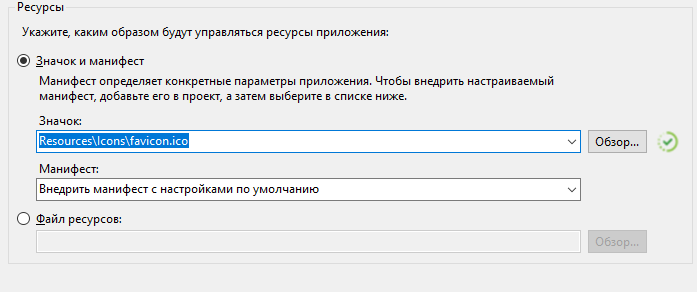


Рисунок 49. Добавление иконки для всех остальных форм

**Вывод:** Входе выполнения практической работы был создан и настроен проект в Visual Studio, а также создан репозиторий, используя средства Visual Studio.

**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 часть:**

**Вывод:** Входе выполнения практической работы была реализована авторизация в системе.

**Контрольные вопросы**

1. Что такое «авторизация», для чего она нужна?
2. Как реализовать авторизацию в wpf-приложении.

**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.

**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.

1. 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.
2. 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.

**Практическая часть:**

**Вывод:** Входе выполнения практической работы был реализован тестовый набор для авторизации пользователя в системе, а также разработан тестовый сценарий и проведено тестирование.

**Контрольные вопросы**

1. Что такое тест-кейс?

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.

**Практическая часть:**

**Вывод:** Входе выполнения практической работы было создано главное окно с возможностью навигации по страницам.