МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО «Уральский государственный экономический университет»

Часовских В.П.

Формализация информации и Big Data

02.03.03 - Математическое обеспечение и администрирование информационных ссистем профиль разработка и администрирование информационных систем

Лабораторный-практическийцикл, работы 7-15(1-8)

Реляционные базы данных

Екатеринбург 2024

Оглавление

Введение

Информационные системы, использующие базы данных, в настоящее время представляют собой одну из важнейших областей современных компьютерных технологий. С этой сферой связана большая часть современного рынка программных продуктов. Одной из общих тенденций в развитии таких систем являются процессы интеграции и стандартизации, затрагивающие структуры данных и способы их обработки и интерпретации, системное и прикладное программное обеспечение, средства разработки взаимодействия компонентов баз, данных и т.п. Современные системы управления базами данных(СУБД) основаны на реляционной модели представления данных—в большой степени благодаря простоте и четкости ее концептуальных понятий и строгому математическому обоснованию.

Реляционная база данных, определения и понятия

Реляционная база данных представляет собой множество **взаимосвязанных двумерных таблиц -** реляционных таблиц, называемых также **отношениями,** в каждой из которых содержатся сведения об одной сущности автоматизируемой предметной области.

Логическую структуру реляционной базы данных образует совокупность реляционных таблиц, между которыми установлены связи.

В таблицах базы должны сохраняться все данные, необходимые для решения задач предметной области. Причем каждый элемент данных должен храниться только в одном экземпляре.

Для создания таблиц, соответствующих реляционной модели данных, используется процесс, называемый нормализацией данных. **Нормализация -** это удаление из таблиц повторяющихся данных путем их переноса в новые таблицы, записи которых не содержат повторяющихся значений.

Структура реляционной таблицы определяется составом полей. Каждое поле отражает определенную характеристику сущности. Для поля указывается

тип и размер элементарного данного, размещаемого в нем, и ряд других свойств. Содержимое поля отображается в столбце таблицы. Столбец таблицы содержит данные одного типа.

Содержание таблицы заключено в ее строках, однотипных по структуре. Каждая строка таблицы содержит данные о конкретном экземпляре сущности и называется **записью**. Структура записи определяется составом входящих в нее полей. Для однозначного определения (идентификации) каждой записи таблица должна иметь уникальный (**первичный**) ключ. По значению ключа таблицы отыскивается единственная запись в таблице. Ключ может состоять из одного или нескольких полей таблицы. Значение уникального ключа не может повторяться в нескольких записях.

Логические связи между таблицами дают возможность объединять данные из разных таблиц. Связь каждой пары таблиц обеспечивается одинаковыми полями в них – ключом связи. Так обеспечивается рациональное хранение недублированных данных и их объединение в соответствии с требованиями решаемых задач.

Для двух таблиц, находящихся в отношении типа 1:М, устанавливается связь по уникальному ключу таблицы, представляющей в отношении сторону «один» - главной таблицы в связи. Во второй таблице, представляющей в отношении сторону «многие» и именуемой подчиненной, этот ключ связи может быть либо частью уникального ключа, либо не входить в состав ключа. В подчиненной таблице ключ связи называется еще **внешним ключом**.

РЕАЛИЗАЦИЯ БАЗЫ ДАННЫХ С ПОМОЩЬЮ СУБД MICROSOFT SQL SERVER

На сегодняшний день известно более двух десятков серверных СУБД, из которых наиболее популярными являются Oracle, Microsoft SQL Server, Informix, DB2, Sybase, InterBase, MySQL.

Microsoft SQL Server — система управления реляционными базами данных (СУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов —SQL, создан совместно Microsoft и Sybase. SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

Сервер СУБД не имеет интерфейса пользователя и для выполнения операций с базой данных ему необходимо посылать команды либо с помощью командной строки или с помощью какой-либо прикладной программы.

В Microsoft SQL Server основным средством администратора БД служит утилита Microsoft SQL Server Management Studio.

SQL Management Studio for SQL Server – это законченное решение для администрирования и разработки баз данных. Для разработчиков приложений или баз данных, администраторов или аналитиков, в SQL Studio найдутся все необходимые средства, чтобы сделать вашу работу продуктивной как никогда ранее. SQL Studio соединяет эти средства в единую мощную и удобную рабочую среду.

SQL Studio предоставляет незаменимые средства для администрирования баз данных и управления их объектами, а также для миграции, сравнения и извлечения баз данных, импорта, экспорта и сравнения данных.

Для выполнения данных лабораторных работ вам понадобится установленная СУБД Microsoft SQL Server версии не ниже 2014.

Взаимодействие с СУБД Microsoft SQL Server осуществляется через графическую оболочку Microsoft SQL Server Management Studio.

Указания к выполнению лабораторных работ

Лабораторные работы выполняются индивидуально. При этом каждая очередная Лабораторная работа является продолжением выполненной ранее и поэтому они должны обязательно выполняться последовательно.

!!!! Перед началом выполнения лабораторных работ необходимо создать рабочую папку БД_Фамилия для хранения файлов, получаемых при выполнении практической работы.

Выполнив лабораторные работы, студент оформляет отчет. По каждой отдельной работе в отчете должен быть отдельный раздел.

Отчет должен содержать: название лабораторной работы; цель работы, условия задания; текст запросов (если требуется); результаты выполнения запроса, скриншоты проделанной работы.

При оформлении отчета соблюдать следующие требования:

1. Поля страницы – обычные.
2. Цвет шрифта – черный.
3. Основной тип шрифта - Times New Roman, размер 14.
4. Межстрочный интервал – полуторный.
5. Выравнивание абзаца – по ширине.
6. Отступ первой строки – 1,25.
7. Нумерация страниц – сквозная на весь отчет, внизу, по центру. На титульной странице номер не отображается (но учитывается).
8. Разделы должны иметь заголовки и быть пронумерованы.
9. На все иллюстрации должна быть ссылка в тексте (…представлен на рис. 1 …..). Подрисуночный текст выравнен по центру.
10. Оглавление в отчете должно быть автоматическим, выполненное средствами MS Word.

Лабораторная работа 1

УСТАНОВКА СОЕДИНЕНИЯ С СЕРВЕРОМ MICROSOFT SQL SERVER И ПРИНЦИПЫ СОЗДАНИЯ БАЗ ДАННЫХ

ЗАДАНИЕ

1. Создать соединение с локальным или удаленным сервером.
2. Изучить пользовательский интерфейс SQL Server Management Studio.
3. Познакомиться с основными принципами создания и удаления базы данных в MS SQL Server.
4. Создать БД с помощью мастера и с помощью запроса (в отчете отобразить создание с помощью обоих методов).
5. Создать резервную копию для восстановления БД.
6. Оформить в отчете раздел по лабораторной работе 1.

Ход работы

1. После запуска среды разработки `SQL Server Management Studio` появится окно подключения к серверу (рис. 1).

В параметрах указываем:

Тип сервера – Компонент `Database Engine`.

Имя сервера. Подключение может быть локальным или удаленным. Представляет собой название компьютера в сети, на котором установлен сервер СУБД. Если сервер установлен на том же компьютере, где сейчас работает пользователь, то в качестве имени используется имя компьютера и идентификатор сервера;

проверка подлинности – `Windows` (по умолчанию),

имя пользователя – имя пользователя по умолчанию, зарегистрированного на сервере MS SQL Server (задается при установке сервера),

пароль – пусто или пароль для пользователя, заданного для сервера MS SQL Server;

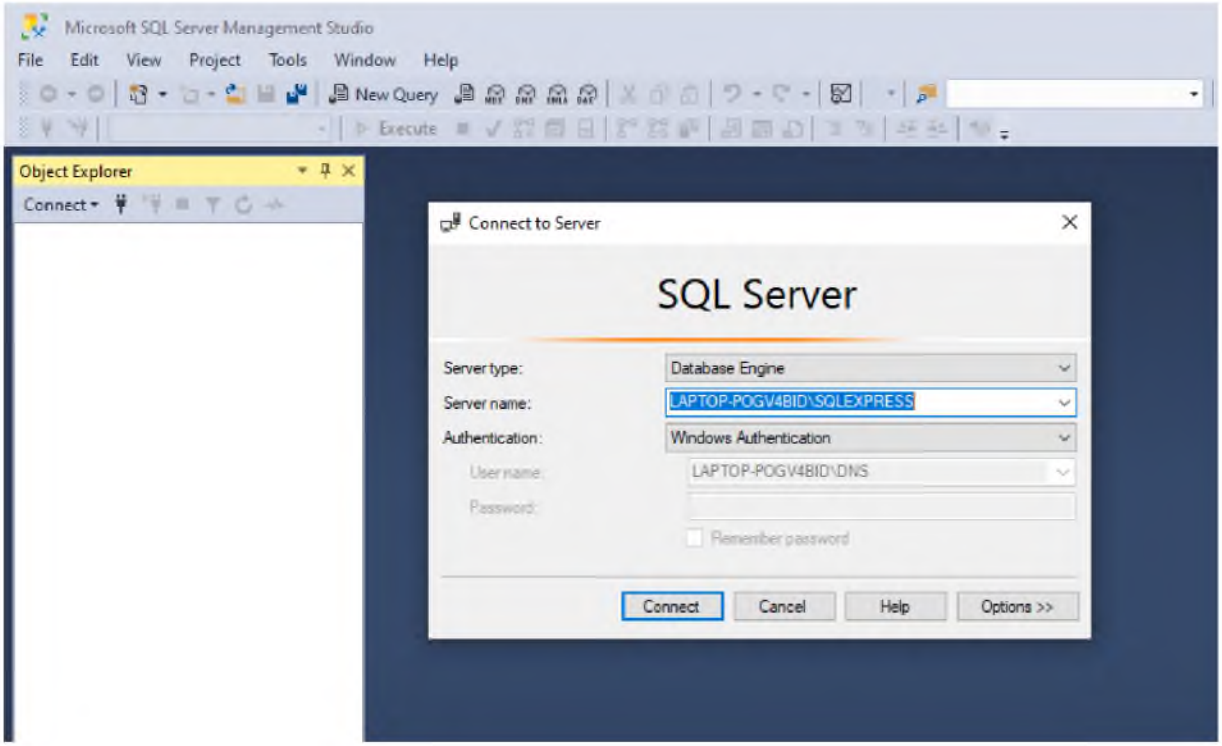


Рис. 1. Соединение с сервером

2. Нажмите `Connect` (Соединить), появится окно среды разработки `SQL Server Management Studio` (рис. 2). Если соединение будет совершенно успешно, то на экране появятся данные сервера.

3. Сведения о базе данных отображаются в обозревателе объектов и окнах документов. Обозреватель объектов является представлением в виде дерева, в котором отображаются все объекты базы данных на сервере.

Общие сведения о базах данных MS SQL Server

Кроме четырех системных баз, `SQL Server` может обрабатывать до 32 734 баз данных, определяемых пользователем.

База данных представляет собой:

— набор взаимосвязанных таблиц;

— связанный набор страниц, выделенных для хранения данных MS SQL Server;

— совокупность данных при архивации;

— два и более файла;

— важную совокупность данных для целей защиты и управления.

Файлы базы данных

База данных состоит из двух и более файлов, каждый из которых может использоваться лишь одной базой.

У файлов существуют два имени: **логическое** и **физическое**. Логическое имя подчиняется стандартным правилам выбора имен объектов SQL Server. Физическое имя представляет собой полное имя любого локального или сетевого файла. Максимальное число файлов в базе данных — 32 768.

Файлы делятся на три типа:

— **первичные файлы**. Используются для хранения данных и информации, определяющих начальные действия с базой. База данных содержит лишь один первичный файл. Стандартное расширение — **.mdf.**

— **вторичные файлы**. Одна или несколько вспомогательных областей для хранения данных. Могут использоваться для распределения операций чтения/записи по нескольким дискам. Стандартное расширение — **.ndf.**

— **файлы журналов.** Содержат журналы транзакций базы данных. База данных содержит по крайней мере один файл журнала. Стандартное расширение — **.ldf.** Перед непосредственной записью транзакций в файл данных все вносимые изменения записываются в журнал.

Группы файлов

Группы файлов предназначены для объединения нескольких файлов. Каждый файл может входить не более чем в одну группу. Файлы журналов не могут принадлежать никаким группам. Группы файлов используются для распределения операций чтения/записи по нескольким дискам. Если группа содержит более одного файла, операции записи распределяются между файлами группы. Базы данных могут содержать до 32 768 групп файлов. У каждой базы данных имеется первичная группа файлов. Она содержит

первичный файл данных и все файлы, которые не были явно назначены в другую группу файлов. Имя первичной группы файлов — **PRIMARY**.

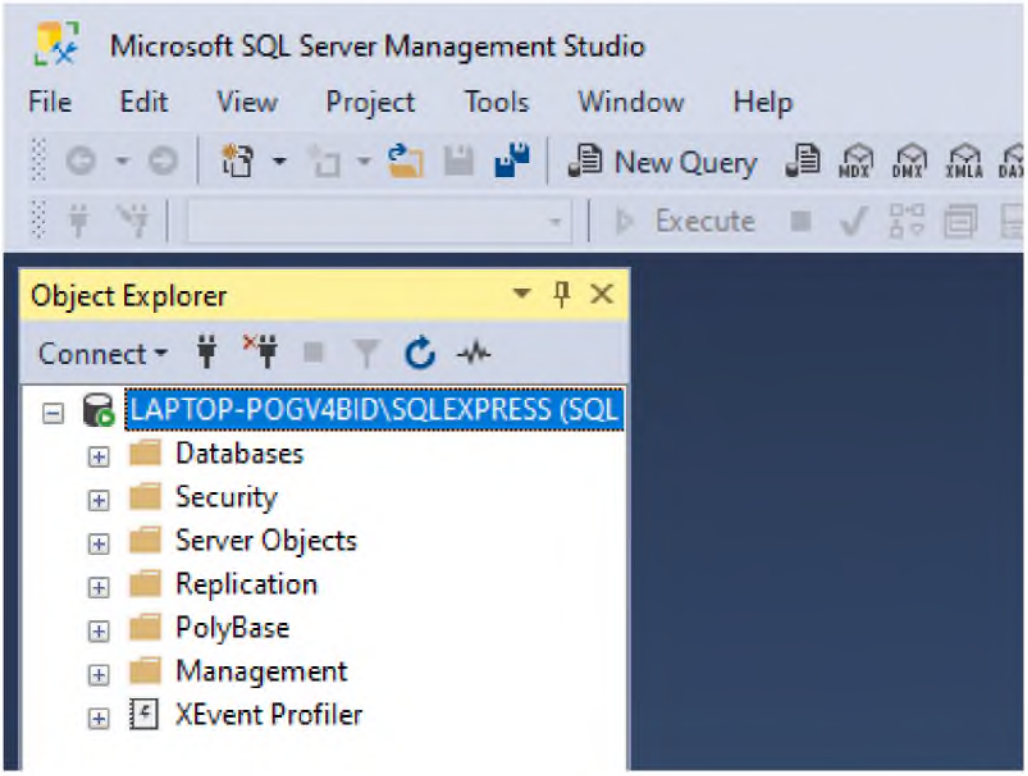


Рис. 2. Рабочее окно Server Management Studio

Создание и удаление базы данных

Для создания базы данных можно использовать один из двух способов:

Первый способ создания БД.

1. Выполнить команду Базы данных/Создать базу данных...(в контекстном меню папки Базы данных) в программе SQL Server Management Studio (рис.3), ввести параметры создаваемой базы данных в диалоговом окне Создание базы данных (рис. 4) и нажать кнопку OK.

2. В поле Имя базы данных введите имя будущей базы данных, например – `University` (!!! При создании новой базы данных используйте следующие правила присвоения имени: Название_БД_группа_Фамилия). Используйте только английский язык. Поле Владелец - задан по умолчанию, в зависимости от настройки сервера. Папка с базой данных будет создана по умолчанию на диске.

C:\Program Files\Microsoft SQL Server\MSSQL10.SQLEXPRESS2\MSSQL\DATA \.

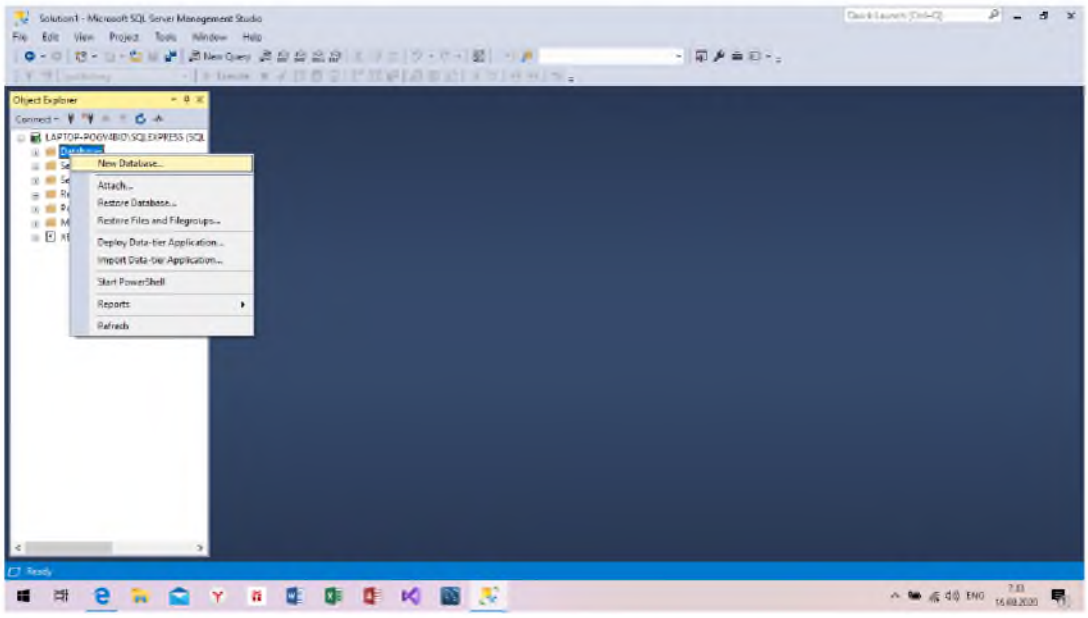


Рис.3. Выполнение команды Создать новую БД

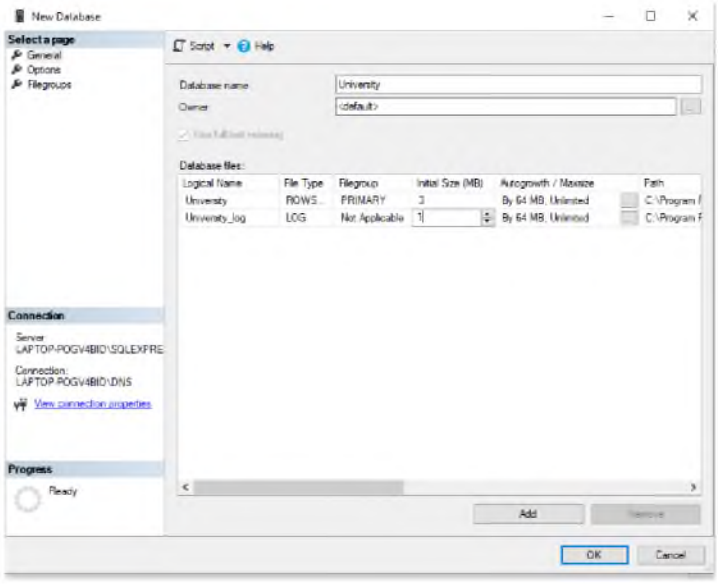
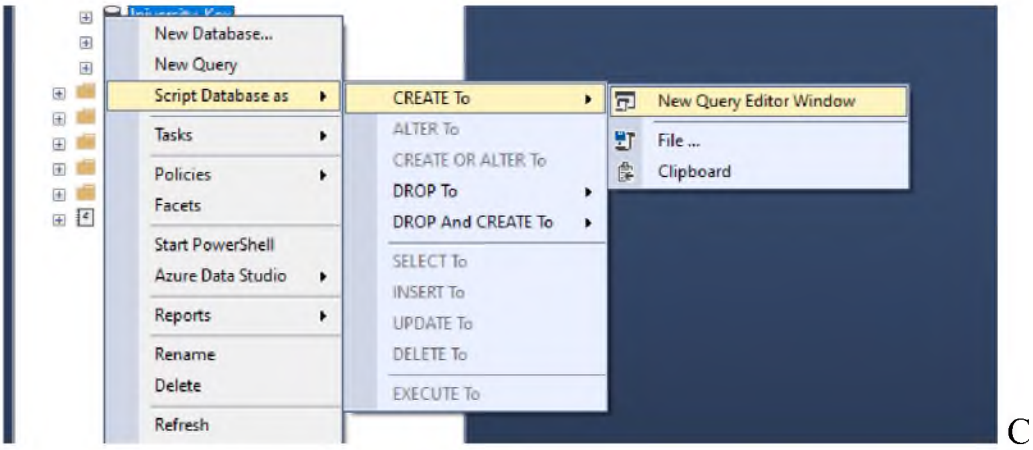


Рис. 4. Диалоговое окно создания базы данных

3. После нажатия на кнопку OK программа SQL Server Management Studio создаст базу данных, имя которой вы увидите в обозревателе объектов.

SQL Server Management Studio создает базу данных, а также генерирует необходимый SQL-код для создания базы данных с теми свойствами, которые указаны в этом диалоговом окне и передаст его серверу СУБД для выполнения.

4. Нажмите на имени базы данных University правой клавишей и из контекстного меню выберите Создать скрипт как.. CREATE To, рис.5.



С

Рис.5. Скрипт создания БД

5. Полученный скрипт (рис. 6.) на создание БД University **сохраните** в рабочей папке.

6. Удалите созданную БД, выбрав в конткстном меню команду Удалить.

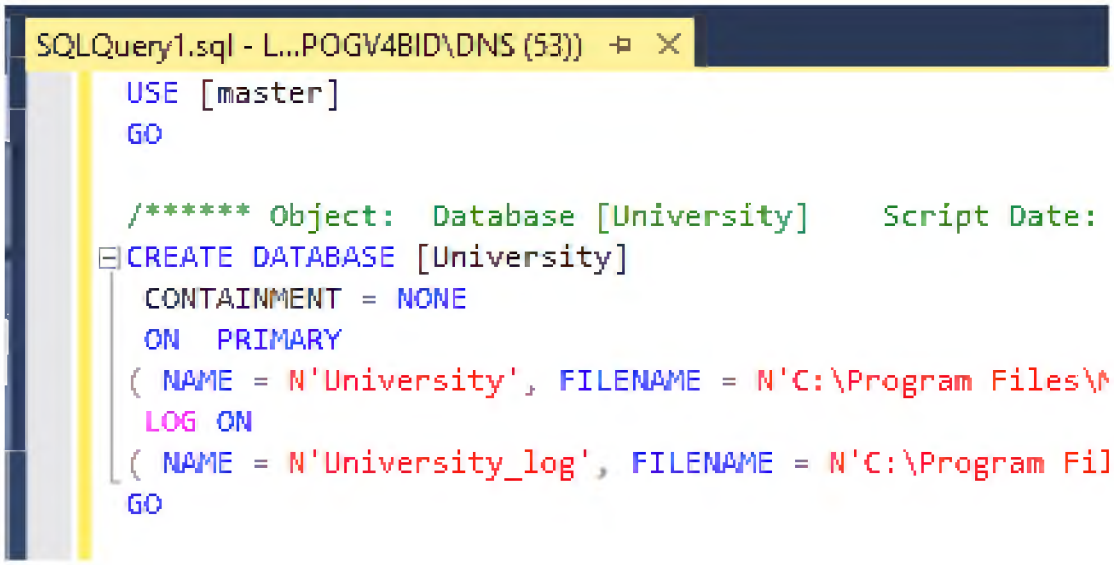


Рис.6. Сгенерированный sql-код созданной базы данных

Второй способ создания БД

Создание новой БД с помощью запроса Новую БД можно создать, используя стандартные команды языка SQL. Все команды языка SQL набираются на вкладке нового запроса (SQLQuery). Для того чтобы создать новый запрос на панели инструментов необходимо нажать кнопку NewQuery/Создать запрос. Для выполнения команд языка SQL на панели инструментов необходимо нажать кнопку Execute/Выполнить или на вкладке нового запроса набрать команду GO.

Создание баз данных разрешается любому пользователю с ролью системного администратора или всем, кому системный администратор предоставил такое право.

Пример: Создать БД «publishing», расположенную в файле D:\publishing.mdf и имеющую начальный размер файла данных 5 Мб., максимальный размер файла данных неограничен. и шаг увеличения файла данных равный 1 Мб. Файл журнала транзакций данной БД имеет имя publishing_log и расположен в файле D:\publishing_log.ldf. Данный файл имеет

начальный размер равный 1 Мб., максимальный размер равный 2 Гб. и шаг увеличения равный 100 Кб.

```
CREATE DATABASE [publishing] ON  PRIMARY
 (
    NAME = 'publishing',
    FILENAME = 'D:\publishing.mdf' ,
    SIZE = 5120KB ,
     MAXSIZE = UNLIMITED,
      FILEGROWTH = 1024KB )
    LOG ON
    (
    NAME = 'publishing_log',
      FILENAME = 'D:\publishing_log.ldf' ,
     SIZE = 1024KB ,
      MAXSIZE = 2048GB ,
      FILEGROWTH = 10%
    )
    GO
```

Если при создании базы не указан первичный файл данных и/или файл журнала, то отсутствующий файл (или файлы) создается с именем по умолчанию. Физические файлы будут находиться в стандартном каталоге. Первичному файлу присваивается имя имя_базы.mdf, а файлу журнала — имя_ базы_log.ldf. Если размер файлов не задан, то при создании размер первичного файла совпадает с размером первичного устройства базы model, а размер файла журнала и вторичных файлов данных равен 1 Мбайт. Он может быть и больше, если размер первичного файла базы данных model превышает 1 Мбайт. Хотя имена и размеры файлов указывать не обязательно, на практике это всегда следует делать. SQL Server создает базу данных за два этапа. На первом этапе база model копируется в новую базу данных, а на втором этапе инициализируется все неиспользуемое пространство.

Команда CREATE DATABASE имеет следующие параметры: • PRIMARY — файл определяется как первичное устройство.

NAME — логическое имя; по умолчанию совпадает с именем файла. • FILENAME — полное имя файла на диске.

SIZE — исходный размер файла. Минимальный размер файла журнала равен 512 Кбайт.

MAXSIZE — максимальный размер файла.

UNLIMITED — размер файла не ограничивается.

FILEGROWTH — приращение размера в мегабайтах (MB), килобайтах (KB) или процентах (%). По умолчанию приращение равно 10%.

FOR LOAD — обеспечивает обратную совместимость со сценариями SQL, написанными для предыдущих версий SQL Server.

FOR ATTACH — указывает, что файлы базы данных уже существуют.

Создание базы данных с помощью запроса

1. Выполнить в программе SQL Server Management Studio команду Создать запрос/ New Query. Наберите предложенный код на рис. 8. (имена и путь создания базы указывать свои).

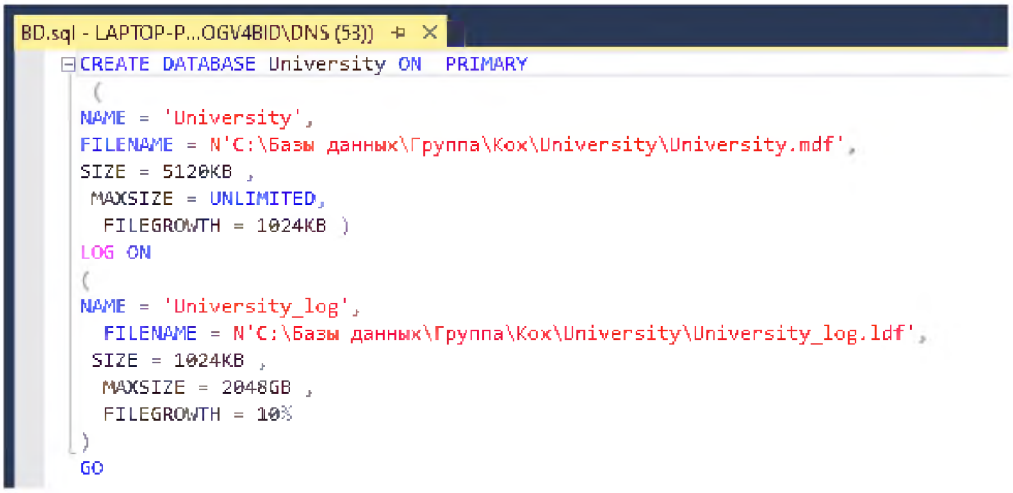


Рис.8.

2. Запустите на выполнение скрипт (Выполнить/Execute).

3. Сохраните созданный скрипт в рабочую папку.

4. После успешного выполнения и обновления проводника у вас должна появится база данных `University`.

5. Для переименования БД необходимо в Обозревателе объектов щёлкнуть по ней правой кнопкой мыши и в появившемся меню выбрать пункт Rename /Переименовать. Для обновления — пункт Refresh/Обновить, а для изменения свойств, описанных выше, — пункт Properties/Свойства.

Для удаления базы данных можно использовать один из трех способов:

Выполнить в программе SQL Server Management Studio команду контекстного меню Удалить, выбрав перед этим в списке базу данных, а затем подтвердить свое желание в диалоговом окне.

Выполнить оператор DROP DATABASE в SQL-редакторе.

Удалить файл с базой данных.

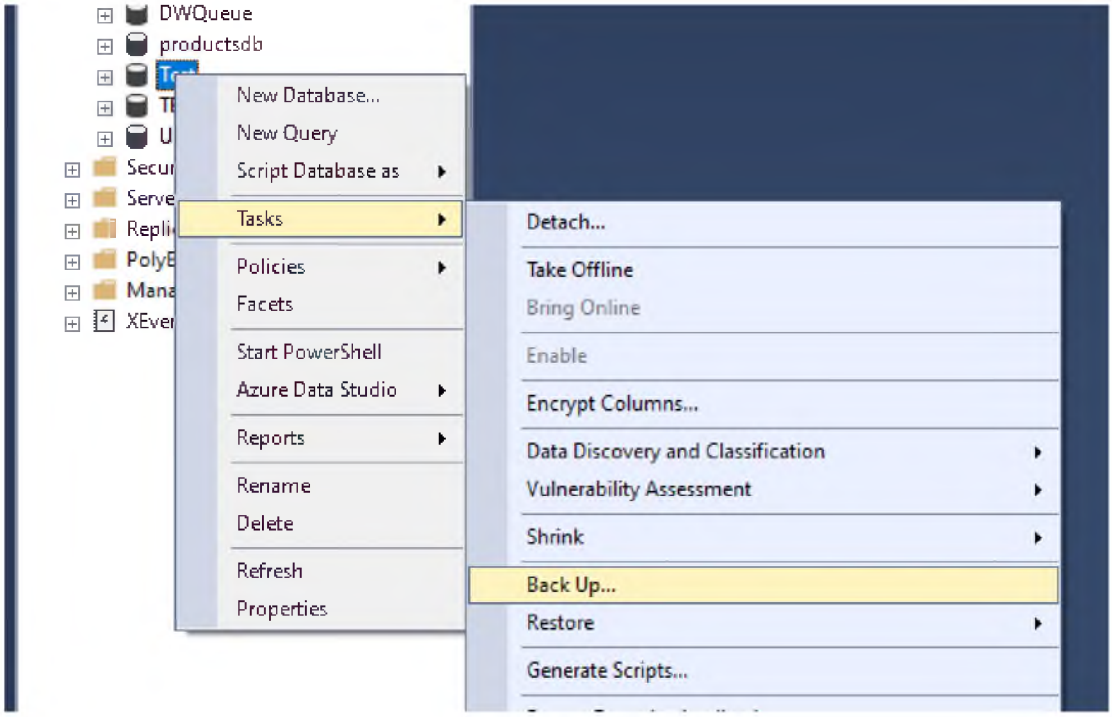
Синтаксис оператора `DROP DATABASE`:

```
DROP DATABASE database_name;
```

Резервное копирование и восстановление

Резервное копирование (backup) базы данных и восстановление из резервной копии (restore) – два важнейших и наиболее частых процесса, осуществляемых администраторами баз данных. Резервное копирование базы данных – единственный надежный способ предохранить данные от потери в результате поломки диска, сбоев электропитания, действий злоумышленников и ошибок в программах. В процессе резервного копирования создается независимый от платформы "снимок" базы данных, с помощью которого можно перенести данные на другую операционную систему или даже другую платформу. Полный цикл: резервное копирование и восстановление из резервной копии приводит к корректировке статистической информации, является средством от излишнего "разбухания" базы данных и необходимой операцией обслуживания базы данных. Кроме того, миграция от одной версии сервера к другой также происходит при помощи процесса backup/restore.

Для создания резервной копии базы данных с помощью программы " SQL Server Management Studio " необходимо подключиться к базе данных, выбрать из контекстного меню базы данных Задачи/ (Tasks) Создать резервную копию (Back Up Database), рис. 8.1. В открывшемся диалоговом окне "Мастер резервного копирования" задать несколько параметров и нажать кнопку [Выполнить], После выбора пути и файла для резервной копии в окне Back Up Database нажатием на OK запускаем процесс создания резервной копии. В случае успешной работы появится сообщение. В результате будет создан файл с резервной копией. Стандартным расширением таких файлов для " SQL Server Management Studio " является "*.bak". Файл с резервной копией базы данных обычно на порядок меньше оригинала.



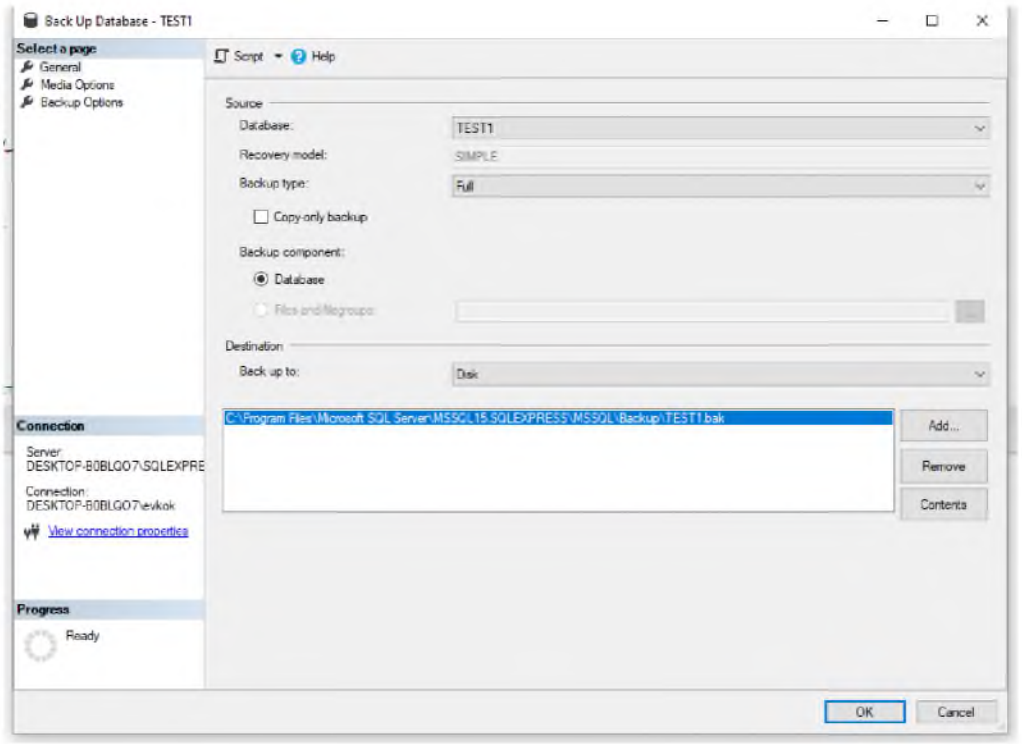
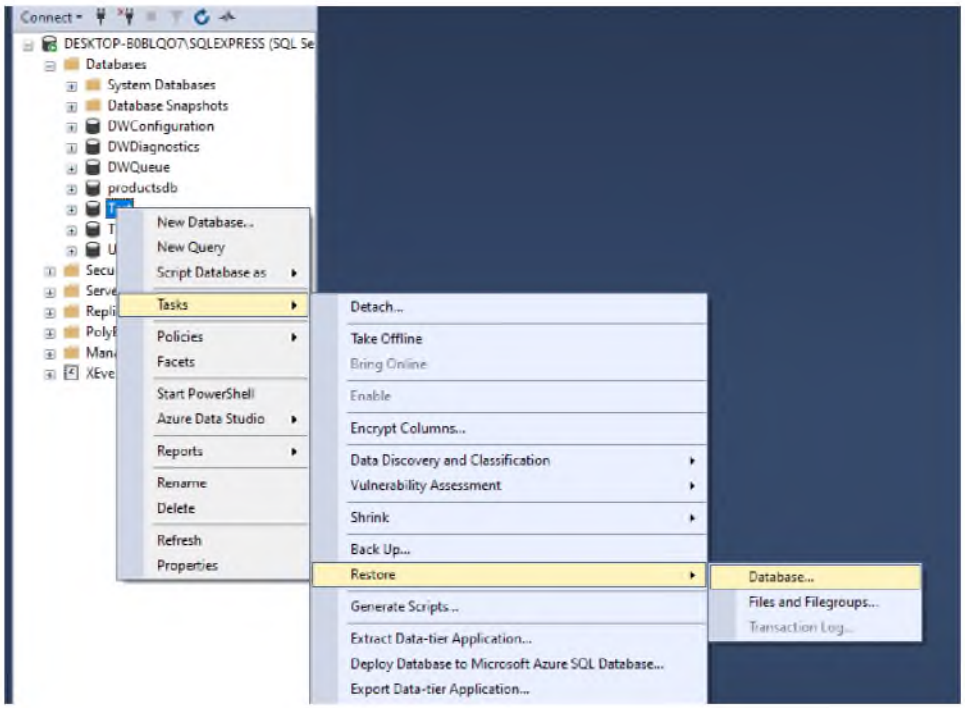
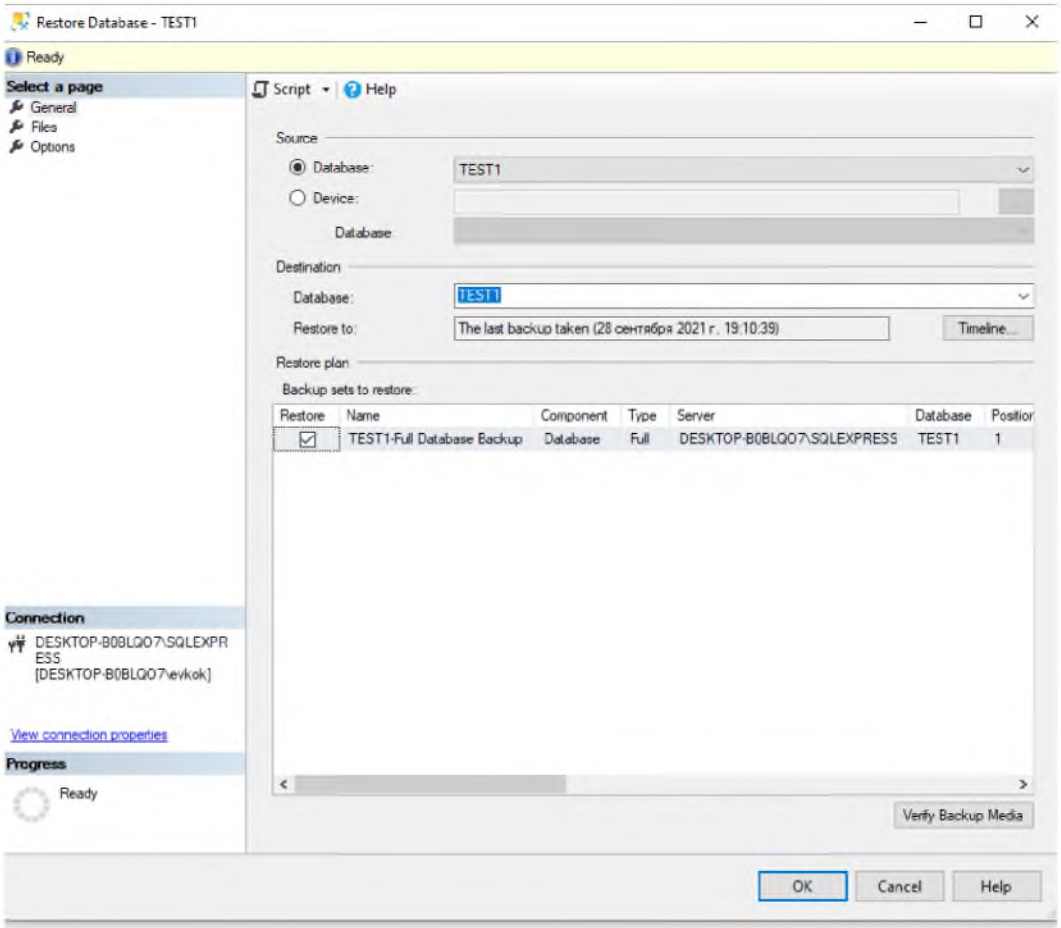
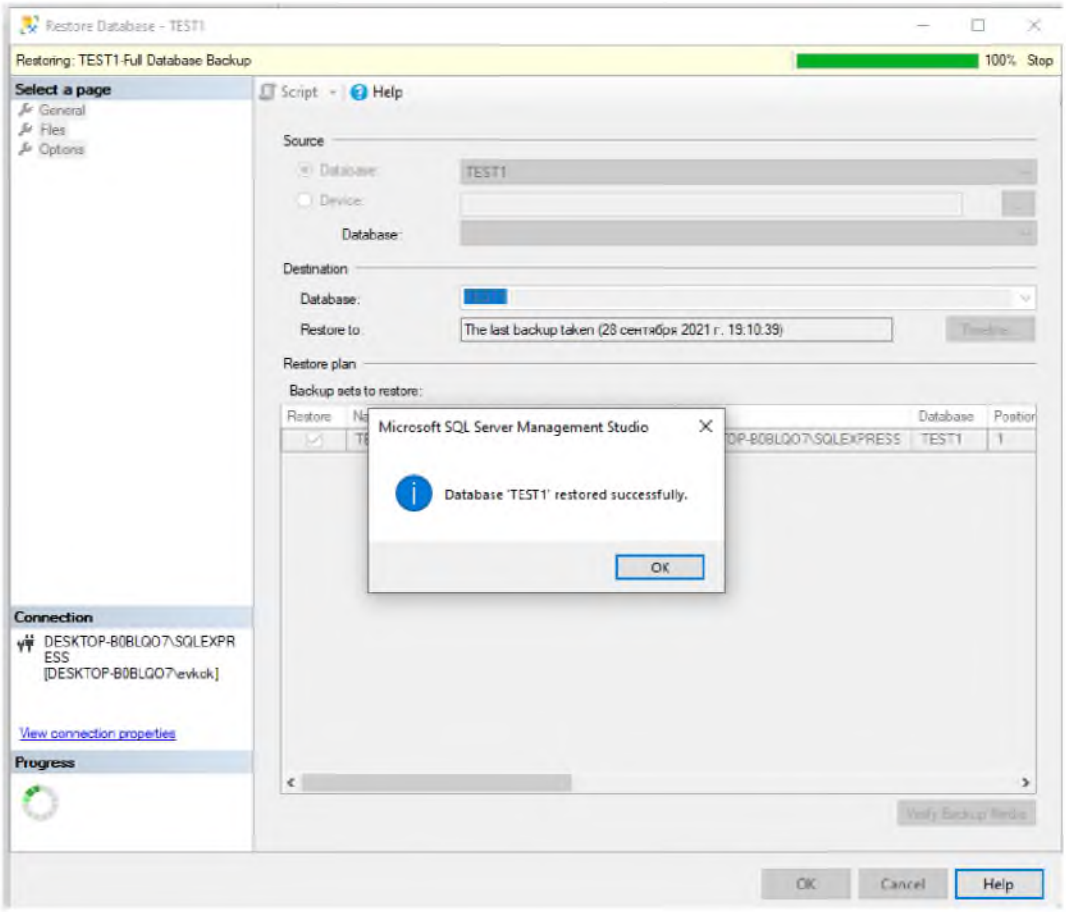


Рис. 8.1. Создать резервную копию

Для восстановления базы данных из резервной копии используется команда "База данных/ Восстановление базы данных (Restore), рис8.2. В результате откроется диалоговое окно "Мастер восстановления баз данных", в котором надо выбрать имя БД куда будет восстанавливаться база данных, в которую будет помещен результат, способ восстановления, файл, из которого будет восстанавливаться база данных, отмечаем выбранную резервную копию, и нажать кнопку [Восстановить]. Запускаем процесс восстановления.


Connect
DESKTOP-B0BLQO7\SQLEXPRESS (SQL Se
Databases
System Databases
Database Snapshots
DWConfiguration
DWDiagnostics
DWQueue
productsdb
New Database...
New Query
Script Database as
Tasks
Policies
Facets
Start PowerShell
Azure Data Studio
Reports
Rename
Delete
Refresh
Properties
Detach...
Take Offline
Bring Online
Enable
Encrypt Columns...
Data Discovery and Classification
Vulnerability Assessment
Shrink
Back Up...
Restore
Generate Scripts...
Extract Data-tier Application...
Deploy Database to Microsoft Azure SQL Database...
Export Data-tier Application...
Database...
Files and Filegroups...
Transaction Log...


Restore Database - TEST1
Ready
Select a page
General
Files
Options
Script
Help
Source
Database:
TEST1
Device:
Database
Destination
Database:
TEST1
Restore to:
The last backup taken (28 сентября 2021 г. 19:10:39)
Timeline...
Restore plan
Backup sets to restore:
Restore
Name
Component
Type
Server
Database
Position
TEST1-Full Database Backup
Database
Full
DESKTOP-B0BLQO7\SQLEXPRESS
TEST1
1
Connection
DESKTOP-B0BLQO7\SQLEXPRESS
[DESKTOP-B0BLQO7\evkok]
View connection properties
Progress
Ready
Verify Backup Media
OK
Cancel
Help



Резервное копирование и восстановление базы данных, наряду с процессом извлечения метаданных и последующего выполнения полученного сценария, можно использовать при переносе разрабатываемой базы данных между различными компьютерами для обеспечения самостоятельной работы студентов над практическими работами или курсовым проектом.

Перенос базы данных между серверами MS SQL Server

Правой кнопкой мыши нажать на БД которую необходимо перенести и в контекстном меню выбрать «Задачи – Сформировать скрипты». Указать путь для сохранения скрипта.

Копирование и перенос на другой сервер БД

Не выполнять! Только ознакомиться! Для просмотра, запуска, остановки служб MS SQL Server необходимо запустить утилиту SQL Server Configuration Manager (рис.9).

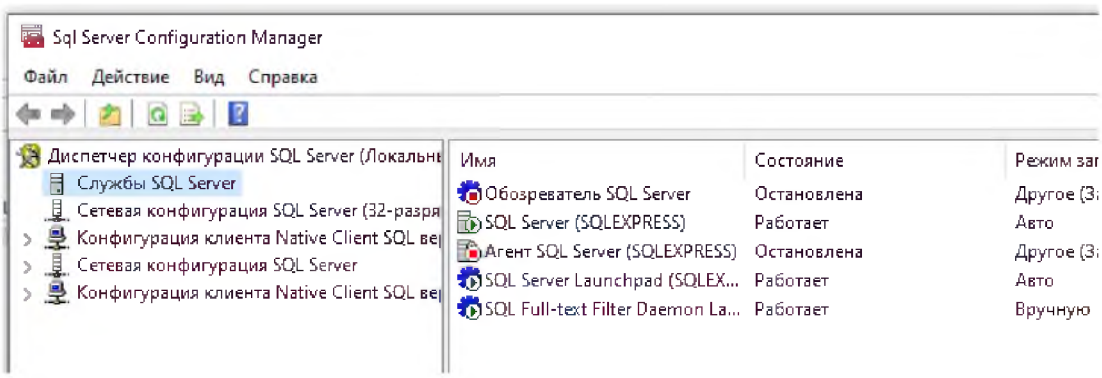


Рис.9. Список служб сервера БД

Для того чтобы скопировать БД необходимо остановить службу SQL Server (в ее контекстном меню выбрать Stop). Далее в подпапке …\MSSQL.1\MSSQL\Data\ скопировать файлы с вашим названием БД (по умолчанию их два). Не забудьте потом снова запустить службу SQL Server (в ее контекстном меню выбрать Start). Для того чтобы подключить скопированную БД на другом сервере, нужно предварительно скопировать ваши файлы в папку …\MSSQL.1\MSSQL\Data\ соответствующего сервера. Далее запустить утилиту SQL Server Management Studio. В появившемся окне с названием Object Explorer Проводник объектов (его можно вызвать по <F8>) выбираем DataBases (Базы данных) и по <правой кнопке мыши> в контекстном меню выбираем Attach… (Присоединить…). В появившемся окне Attach DataBases (Присоединение базы данных) нажать <Add> и выбрать ваш файл БД с расширением .mdf.

После выполнения лабораторной работы в отчете создать раздел Лабораторная работа 1. В нем должна содержаться следующая информация:

— название лабораторной работы;

— цель работы;

— задание;

— текст запросов;

— результат запросов;

— вывод.

Обязательно должны быть скриншоты (к лр 1):

— соединения с сервером;

— создания БД двумя способами (путь создания должен виден полностью);

— результат создания БД (в окне обозревателя объектов);

— полученного скрипта;

— диалогового окна создания резервной копии;

— диалогового окна восстановления бд из резервной копии;

Соблюдать требования предъявляемые к оформлению отчета.

Лабораторная работа 2

Задание

1. Изучить способы создания, изменения и удаления таблиц. Получить навыки использования приложения " SQL Server Management Studio " для создания, удаления и изменения структуры таблиц.
2. Используя инструменты SQL Management Studio создать таблицы.

Язык запросов SQL

Появление и развитие языка SQL связано с созданием теории реляционных БД. Математической основой языка SQL является реляционная алгебра и реляционное исчисление. Прообраз языка возник в 1970 году в лаборатории Санта-Тереза фирмы IBM. В настоящее время популярность SQL настолько велика, что разработчики нереляционных СУБД снабжают свои системы SQL-интерфейсом. SQL сочетает в себе возможности языка определения данных, языка манипулирования данными и языка запросов. При этом он реализует и основные функции реляционных СУБД. SQL не является языком программирования в традиционном представлении. На нем пишутся не программы, а запросы к БД, поэтому этот язык называют языком запросов, языком декларативным, а не процедурным. Это означает, что с его помощью можно сформулировать, что необходимо получить, однако нельзя указать, как это следует сделать. В отличие от процедурных языков программирования (Си, Паскаль), в языке SQL отсутствуют алгоритмические конструкции, операторы цикла, условные переходы и т.д.

Язык SQL выходит в состав 4 семейств языков и выполняет их роли. SQL является языком **DDL** (Data Definition Language – язык описания данных) так как реализует следующие функции:

- **CREATE** – создание объектов в БД;
- **ALTER** – изменение структуры БД;
- **DROP** – удаление объектов из БД;

— **TRUNCATE** – удаление всех записей из БД;

— **COMMENT** – добавление комментариев;

— **RENAME** – изменение имени объекта БД.

SQL является языком **DML** (Data Manipulation Language – язык манипулирования данных) так как реализует следующие функции:

— **SELECT** – извлечение данных из БД;

— **UPDATE** – обновление данных в БД;

— **DELETE** – удаление данных из БД;

— **INSERT** – вставка данных в БД;

— **MERGE** – вставка и обновление;

— **CALL** – вызов **PL/SQL** или Java подпрограммы;

— **EXPLAIN PLAN** – создание планов выполнения запросов;

— **LOCK TABLE** – для блокирования таблиц.

SQL является языком **DCL** (Data Control Language – язык управления данными) так как реализует следующие функции:

— **GRANT** – предоставляет права доступа пользователям к БД;

— **REVOKE** – отмена прав доступа.

SQL является языком **TCL** (Transaction Control Language – язык транзакций) так как реализует следующие функции:

— **COMMIT** – сохранение работы;

— **SAVEPOINT** - определение точки транзакции;

— **ROLLBACK** – восстановление БД на оригинал с последним **COMMIT**;

— **SET TRANSACTION** – изменение параметров транзакций.

Запрос в языке SQL состоит из одного или нескольких операторов, следующих один за другим и разделенных точкой с запятой. Каждая последовательность операторов языка SQL реализует определенное действие над БД. Оно осуществляется за несколько шагов, на каждом из которых над

таблицами выполняются определенные действия. Каждый оператор SQL начинается с ключевого слова, которое определяет, что делает этот оператор (SELECT, INSERT, DELETE). В операторе содержатся предложения, содержащие сведения о том, над какими данными производятся операции. Каждое предложение начинается с ключевого слова, такого как FROM, WHERE и др. Структура предложения зависит от его типа: ряд предложений содержит имена полей или таблиц, некоторые могут включать дополнительные ключевые слова, константы или выражения.

Создание таблиц

После создания базы данных мы пока не может вносить в нее какие-либо данные т.к. пока еще не создана структура куда мы можем их поместить и работать с ними. Так ка Microsoft SQL Server – реляционная СУБД, поэтому все данные в Sql хранятся в виде двумерных таблиц со строками и столбцами. Строки называются кортежами или записями, а столбцы – доменами или полями.

Вся информация обрабатывается по записям. Каждая запись состоит из полей. Каждое поле имеет три характеристики:

— имя поля – используется для обращения к полю;

— значение поля – определяет информацию, хранимую в поле;

— тип данных поля – определяет, какой вид информации можно хранить в поле.

Основные ограничения, которым должны удовлетворять таблицы:

1. Каждый столбец в таблице имеет уникальное имя.
2. Все данные в столбце должны быть одного типа.
3. Порядок строк и столбцов в таблице не имеет значения.
4. В таблице не может быть двух одинаковых строк.

В SQL сервер используется следующие типы данных:

Битовые типы данных, которые содержат последовательности нулей и единиц: Binary(n) и Varbinary(n), где n длина. Содержимое полей типа Binary

всегда равно n, разница заполняется пробелами. Varbinary размер поля равен n или большему;

Целочисленные типы данных – типы данных для хранения целых чисел (в скобках указан диапазон значений типа данных): Tinyint (0-255), Smallint (±32000), Int (±2000000000), Bigint (±263);

Типы данных для хранения дробных чисел:

Типы числовых данных с фиксированными точностью и масштабом. Типы decimal и numeric являются взаимозаменяемыми синонимами.

decimal[(p[, s])] и numeric[(p[, s])] Числа с фиксированной точностью и масштабом. При использовании максимальной точности числа могут принимать значения в диапазоне от -10^38+1 до 10^38-1. Синонимами типа decimal по стандарту ISO являются типы dec и dec(p, s). Тип numeric функционально эквивалентен типу decimal.

p (точность). Максимальное общее число хранимых десятичных разрядов. Это число включает символы слева и справа от десятичной запятой. Точность должна быть значением в диапазоне от 1 до максимум 38. Точность по умолчанию составляет 18.

s (масштаб). Максимальное число хранимых десятичных разрядов справа от десятичной запятой. Это число отнимается от *p* для определения максимального количества цифр слева от десятичной запятой. Масштаб должен иметь значение от 0 до *p* и может быть указан только при заданной точности. По умолчанию масштаб принимает значение 0, поэтому 0 <= *s* <= *p*. Максимальный размер хранилища зависит от точности.

float [(*n*)], где *n* — это количество битов, используемых для хранения мантиссы числа в формате float при экспоненциальном представлении. Определяет точность данных и размер для хранения. Если указан параметр *n*, это должно быть значение в диапазоне от 1 до 53. Значение *n* по умолчанию — 53. В приложении SQL Server параметр n может принимать одно из двух возможных значений. Если 1<=n<=24, n принимает значение 24. Если 25<=n<=53, n принимает значение 53.

Специальные типы данных: Bit – логический тип данных является заменой логическому типу Boolean в Visual Basic, Text - тип для хранения больших объемов текста, одно поле может хранить до 2 Гб текста, Image – тип данных для хранения до 2Гб рисунков, RowGUID – уникальный идентификатор строки таблицы, SQL_Variant - аналогичен типу Variant в Visual Basic;

Типы данных даты и времени: Datetime (от 1.01.1953 до 3.12. 1999). SmallDatetime (от 1.01.19 до 6.07 2079);

Денежные типы данных для хранения финансовой информации: Money (±1015 и 4 знака после нуля), Smallmoney (± 20000,0000);

Автоматически обновляемые типы данных - аналоги счетчиков, но в данной роли они не используются: RowVersion уникальный идентификатор строки. TimeStamp – закодированное дата и время создания строки.

Ход работы

1. Создайте новую БД `Training_base_Фамилия` (способ создания выберите самостоятельно).

2. Создайте таблицу `Salespeople (Продавцы)`. Для этого щёлкните правой кнопкой мыши по папке `Tables` («Таблицы») и в появившемся меню выберите пункт `New Table` (Создать таблицу), рис.10. Появится окно создания новой таблицы (рис. 11).

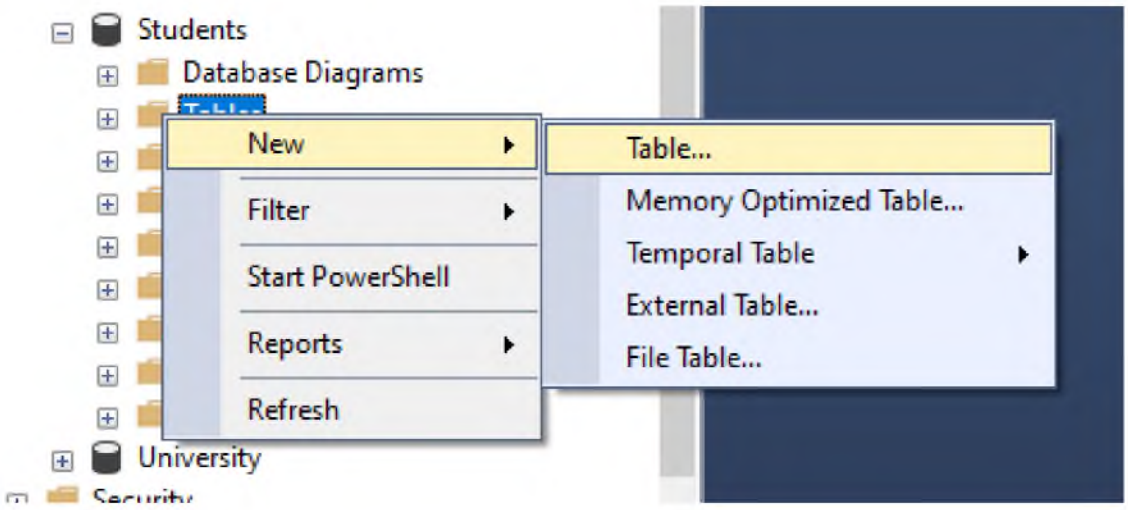


Рис. 10. Выбор команды Создать таблицу

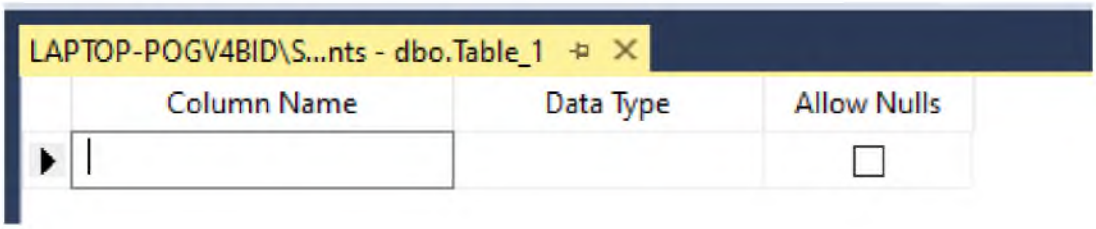


Рис. 11. Окно создания таблицы

Данная таблица имеет следующие столбцы:

— `Column Name` (Имя столбца) — имя столбца должно всегда начинаться с буквы и не должно содержать различных специальных символов и знаков препинания. Если имя поля содержит пробелы, то оно автоматически заключается в квадратные скобки.

— `Data Type` (Тип данных) — тип данных поля.

— `Allow Nulls` (Разрешить значения `Null`) — допуск значения `Null`. Если эта опция поля включена, то в случае незаполнения поля в него будет автоматически подставлено значение `Null`. То есть поле необязательно для заполнения. Замечание: Под таблицей определения полей располагается таблица свойств выделенного поля `Column Properties` (Свойства столбца). В данной таблице настраиваются свойства выделенного поля. Некоторые из них будут рассмотрены ниже. Перейдем к созданию полей и к настройке их свойств. В таблице определения полей задайте значения столбцов `Column Name` («Имя столбца»), `Date Type` (Тип данных) и `Allow Nulls` (Разрешить значения `Null`), как показано на рис. 12.

Из рис. 12 следует, что таблица `Salespeople` имеет четыре поля:

— `SNUM` — уникальный номер (ключевое поле), приписанный каждому продавцу (номер служащего), числовой, целое. Чтобы сделать поле ключевым, выделите поле, а затем на панели инструментов нажмите кнопку с изображением ключа (или в контекстном меню поля). В таблице определения полей, рядом с полем `SNUM` появиться изображение ключа, говорящее о том, что поле ключевое;

— SNAME — имя продавца, текстовое поле, предназначенное для хранения строк, имеющих длину не более 10 символов;

— CITY — Место расположения продавца, текстовое поле, предназначенное для хранения строк, имеющих длину не более 10 символов.

— COMM – вознаграждение (комиссионные) продавца, числовое поле с плавающей точкой (decimal (6,2) означает, 6 - максимальное общее число хранимых десятичных разрядов, включая символы слева и справа от десятичной запятой, 2 - число хранимых десятичных разрядов справа от десятичной запятой).

— Сохраните таблицу под именем Salespeople. Обновите папку Таблицы. Таблица Salespeople отобразится в обозревателе объектов в папке Tables (Таблицы) БД «Training_base» (Рис. 13).

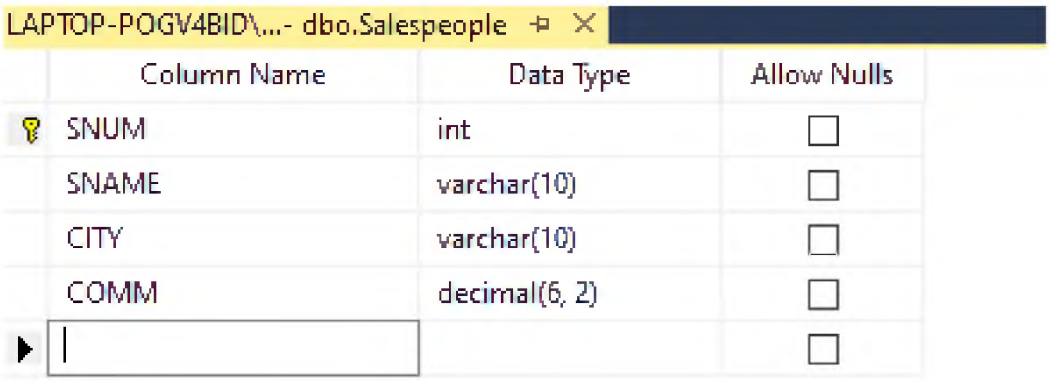


Рис. 12. Создание таблицы Salespeople

В обозревателе объектов таблица Salespeople отображается как dbo.Salespeople. Префикс dbo обозначает, что таблица является объектом БД (Data Base Object). В дальнейшем при работе с объектами БД префикс dbo можно опускать.

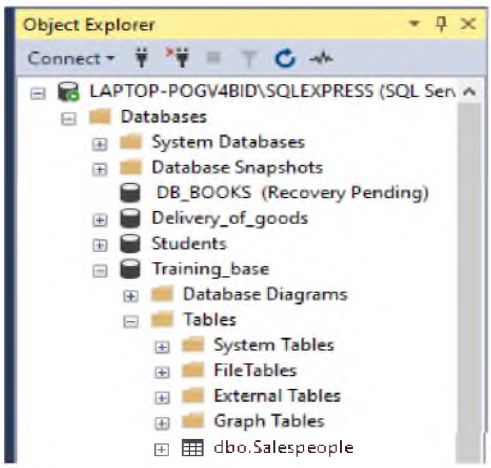


Рис. 13. Созданная таблица `Salespeople` в окне обозревателя

3. Создайте вторую таблицу- `Customers (Покупатели)`.

4. Добавьте в нее поля: (в соответствии с рис. 14).

— `CNUM` – уникальный номер, присвоенный покупателю;

— `CNAME` – имя покупателя;

— `CITY` – место расположения покупателя;

— `RATING` – цифровой код, определяющий уровень предпочтения данного покупателя. Чем больше, тем больше предпочтение.

— `SNUM` – номер продавца, назначенного данному покупателю (из таблицы `Salespeople`).

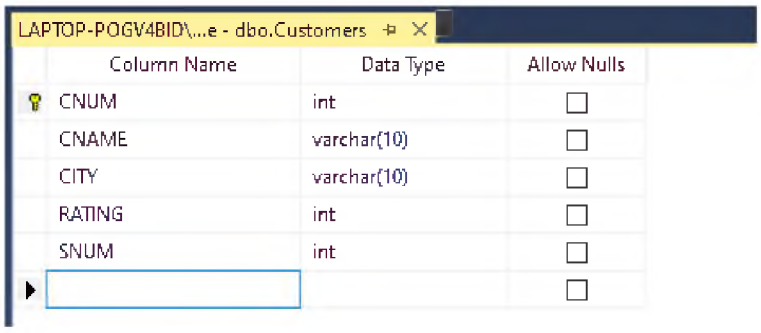


Рис. 14. Создание таблицы `Customers`

5. Создадите таблицу `Orders` (`Заказы`). Включите в нее следующие поля:

— `ONUM` – уникальный номер, присвоенный данной покупке;

— `AMT` – количество (сумма заказа);

— ODATE – дата заказа (покупки);

— CNUM – номер покупателя, сделавшего покупку (из таблицы (Customers);

— SNUM – номер продавца, обслуживающего покупателя (из таблицы Salespeople).

Поле ONUM (уникальный номер, присвоенный данной покупке) является первичным ключом, в данном примере будет числовым **счётчиком**. То есть данное поле должно автоматически заполняться числовыми значениями. Более того, оно должно быть **ключевым**.

Сделайте поле ONUM **счётчиком**. Для этого выделите поле, просто щёлкнув по нему мышкой в таблице определения полей. В таблице свойств поля отобразятся свойства поля ONUM. Разверните группу свойств Identity Specification (Спецификация идентификатора). Свойство (Is Identity) (Идентификатор) установите в значение Yes (Да). Задайте свойства Identity Seed («Начальное значение идентификатора») и Identity Increment («Шаг приращения идентификатора») равными 1 (Рис. 15). Эти настройки показывают, что значение поля ONUM у первой записи в таблице будет равным 1, у второй — 2, у третьей — 3 и т.д. Теперь сделайте поле ONUM ключевым.

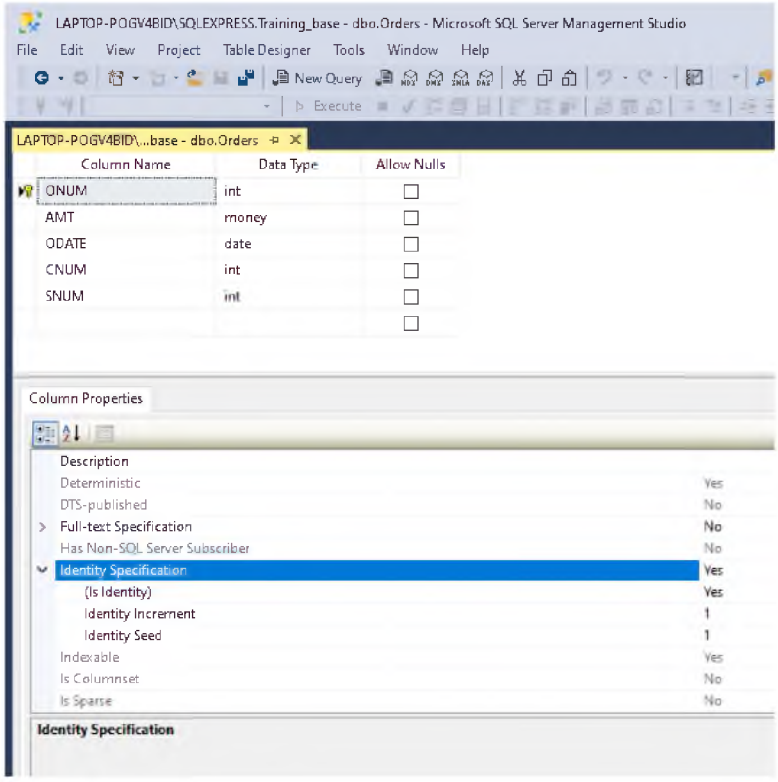


Рис.15. Таблица `Orders`

Создание таблиц с помощью запроса

Для создания таблиц в SQL Server в первую очередь необходимо сделать активной ту БД, в которой создается таблица. Для этого в новом запросе можно набрать команду: **USE** <Имя БД>, либо на панели инструментов необходимо выбрать в *выпадающем списке* рабочую БД. После выбора БД можно создавать таблицы.

Таблицы создаются командой:

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

Здесь:

`table_name` – имя создаваемой таблицы;

`column1`- имя первого поля таблицы;

`datatype` – тип поля;

Если имя поля содержит пробел, то оно заключается в квадратные скобки.

Пример создания таблицы с помощью запроса приведен на рис.15.1.

```
CREATE TABLE Student (
ID_student INTEGER NOT NULL,
Surname VARCHAR(20) NOT NULL,
ID_specialization INTEGER CHECK(ID_specialization<12),
Note VARCHAR(20) DEFAULT 'зачислен'
CONSTRAINT PK_Stud PRIMARY KEY (ID_student)
);
```

Рис. 15.1. Запрос на создание таблицы Student

`NOT NULL` – столбец не может содержать значение `NULL`, т. е. значения этого столбца должны быть определены.

`PRIMARY KEY` – столбец является первичным ключом. В каждой таблице только один столбец (составной) может быть первичным ключом. Это означает, что он не может содержать значение NULL, а вводимое в него значение должно отличаться от всех остальных значений в этом столбце.

`DEFAULT значение` – устанавливает значение по умолчанию.

`CHECK (условие)`- позволяет производить проверку условия при вводе данных. Значение будет сохранено, если условие выполняется, в противном случае – нет.

Если необходимо использовать поле в качестве числового счетчика, то запрос на создание таблицы может быть создан, как показано на рис. 15.2.

Ограничение PRIMARY KEY может также быть применено для нескольких полей, составляющих уникальную комбинацию значений — составной первичный ключ. Пример запроса приведен на рис. 15.3.

```
CREATE TABLE Факультет
(
    Код_факультета int IDENTITY (1,1) NOT NULL,
    Наименование_факультета varchar(50) NOT NULL,
    Фамилия_декана varchar(50) NOT NULL,
    Телефон_декана int NOT NULL
 CONSTRAINT PK_Факультет PRIMARY KEY (Код_факультета)
);
```

Рис. 15.2. Запрос на создание таблицы Факультет

```
CREATE TABLE NEW_EXAM_MARKS
(EXAM_ID INT NOT NULL,
STUDENT_ID INT NOT NULL,
SUBJ_ID INT NOT NULL,
MARK INT,
DATA DATE,
CONSTRAINT EX_PR_KEY PRIMARY KEY (EXAM_ID, STUDENT_ID));
```

Рис. 15.3. Запрос на создание таблицы с составным первичным ключом

6. Создайте запрос на создание таблиц `Salespeople_1`, `Customers_1` и `Orders_1`. Выполните его.

После выполнения лабораторной работы в отчете создать раздел Лабораторная работа 2. В нем должна содержаться следующая информация:

— название лабораторной работы;

— цель работы;

— задание;

— описание таблиц (спецификацию полей);

— вывод по работе.

Обязательно должны быть скриншоты (к лр 2):

— подключения к БД (должно быть видно имя сервера и полное имя БД);

— создания всех таблиц двумя способами (должно быть видно имя сервера);

— установления поля – типа счетчик и ключевого;

Соблюдать требования предъявляемые к оформлению отчета.

Лабораторная работа 3

Связывание таблиц.

Задание

Обеспечить целостность данных.

Целостность базы данных (database integrity) — соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам. Каждое правило, налагающее некоторое ограничение на возможное состояние базы данных, называется **ограничением целостности** (integrity constraint). Примеры правил: вес детали должен быть положительным; количество знаков в телефонном номере не должно превышать 25; возраст родителей не может быть меньше возраста их биологического ребёнка и т.д.

Целостность базы данных - свойство базы данных, означающее, что БД содержит полную и **непротиворечивую** информацию, необходимую для корректного функционирования приложений

Задача аналитика и проектировщика базы данных — возможно более полно выявить все имеющиеся ограничения целостности и задать их в базе данных.

Целостность БД не гарантирует достоверности содержащейся в ней информации, но обеспечивает по крайней мере правдоподобность этой информации, отвергая заведомо невероятные, невозможные значения. Таким образом, не следует путать целостность БД с достоверностью БД. Достоверность (или истинность) есть соответствие фактов, хранящихся в базе данных, реальному миру. Очевидно, что для определения достоверности БД требуется обладание полными знаниями как о содержимом БД, так и о реальном мире. Для определения целостности БД требуется лишь обладание

знаниями о содержимом БД и о заданных для неё правилах. Поэтому СУБД может (и должна) контролировать целостность БД, но принципиально не в состоянии контролировать достоверность БД. Контроль достоверности БД может быть возложен только на человека, да и то в ограниченных масштабах, поскольку в ряде случаев люди тоже не обладают полнотой знаний о реальном мире.

Итак, БД может быть целостной, но не достоверной. Возможно, и обратное: БД может быть достоверной, но не целостной. Последнее имеет место, если правила (ограничения целостности) заданы неверно.

Внешний ключ – это столбец (или комбинация столбцов), совпадающий с первичным ключом некоторой таблицы. Если значение внешнего ключа соответствует значению первичного ключа, становится понятно, что между объектами базы данных представленными совпадающими строками, существует логическое взаимоотношение. Одним из главных ограничений отношения является ссылочная целостность, которая определяет, что каждое значение внешнего ключа, не являющееся NULL-значением, должно ссылаться (REFERENCES) на некоторое существующее значение первичного ключа.

Для обеспечения целостности данных в SQL Server используют диаграммы. Диаграммы — это компоненты БД, которые блокируют удаление записей из первичных таблиц, если существуют связанные с ними записи во вторичных таблицах. Следовательно, диаграммы предотвращают нарушение целостности данных. В SQL Server диаграммы создаются при помощи мастера диаграмм.

Ход работы

1. В обозревателе объектов для базы данных Training_base в контекстном меню папки «Database Diagrams» (Диаграммы баз данных), выберите пункт «New Datebase Diagram» (Создать диаграмму базы данных), рис. 17.

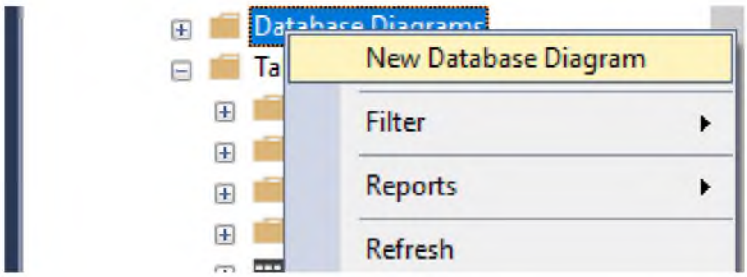


Рис. 17. Выбор команды создать диаграмму

2. Откроется окно «Add table» (Добавление таблицы), рис. 18.

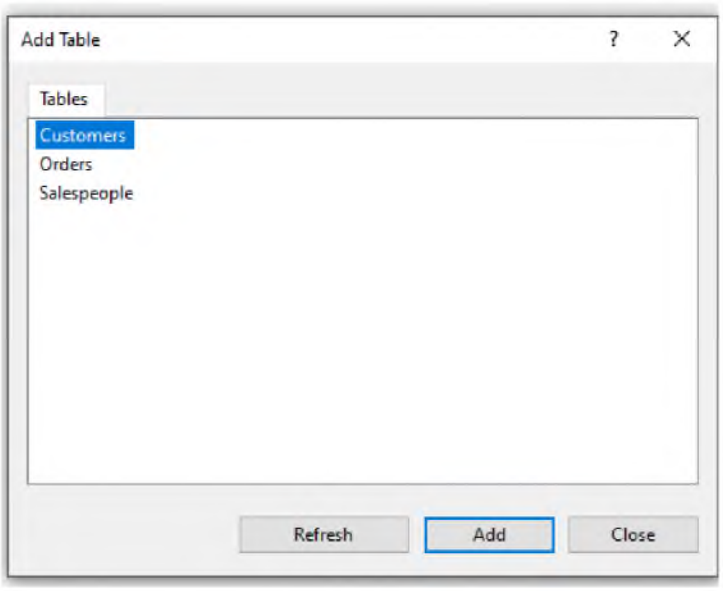


Рис. 18. Добавление таблиц в диаграмму

3. Добавьте все имеющиеся таблицы БД `Training_base` (рис.19).

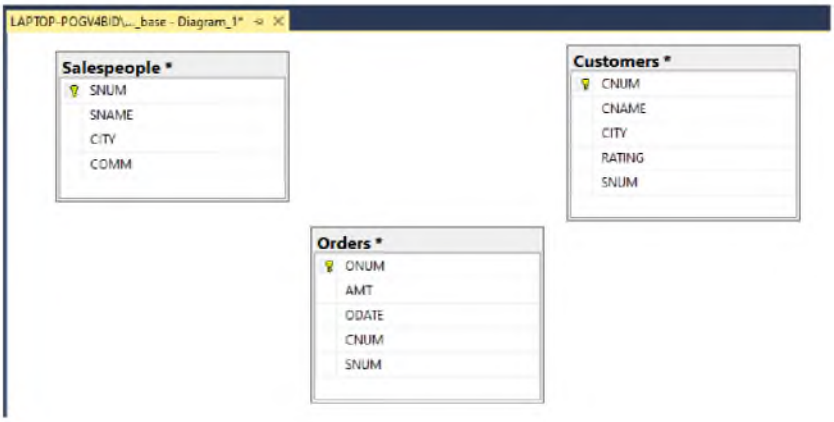


Рис. 19. Добавленные таблицы на диаграмму

4. Установить связи между таблицами, для этого необходимо удерживая мышью поле первичного ключа одной таблицы перетянуть на поле внешнего ключа другой таблицы. Например, после перетягивания **ключевого**

поля SNUM из таблицы Salespeople на поле SNUM из таблицы Orders, где оно является **внешним** ключом откроется окно «Tables and Columns» (Таблицы и столбцы), рис. 20.

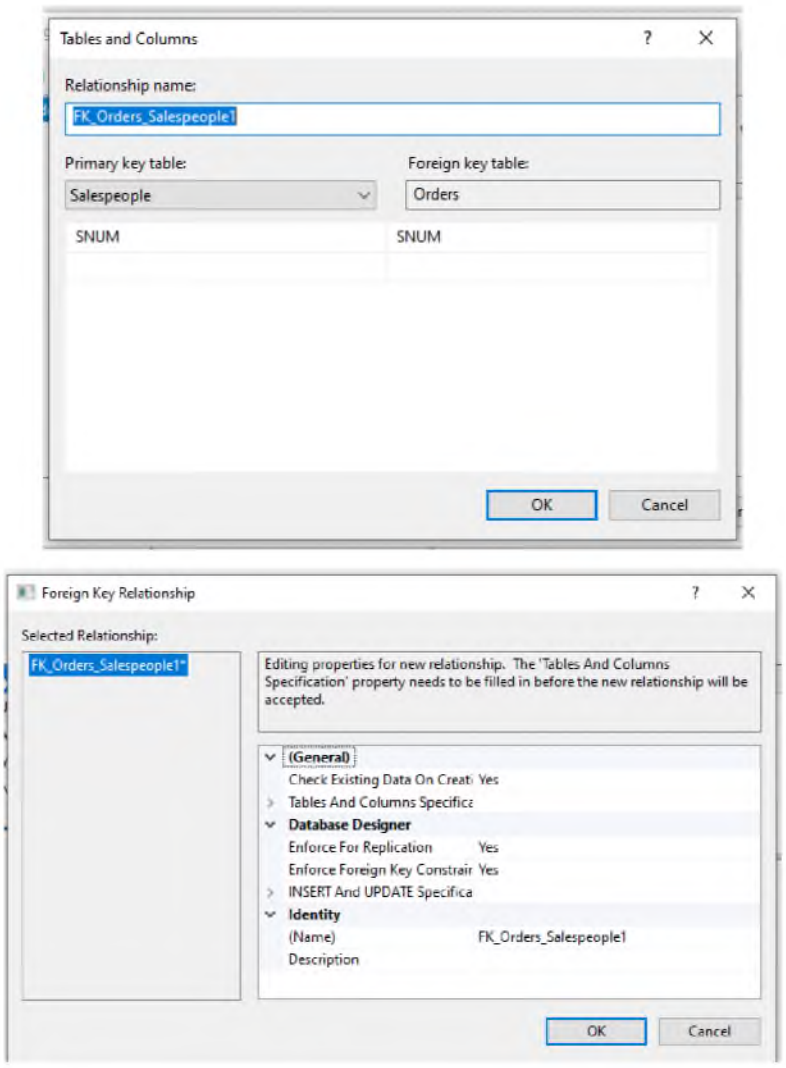


Рис. 20. Установление связи между таблицами

5. Аналогично установите связи между таблицами Customers и Order по полю CNUM. В результате будут установлены связи между таблицами типа один ко многим (символ ключа на линии связи — это сторона связи один, символ бесконечности на линии связи — это сторона связи многие, рис. 21).

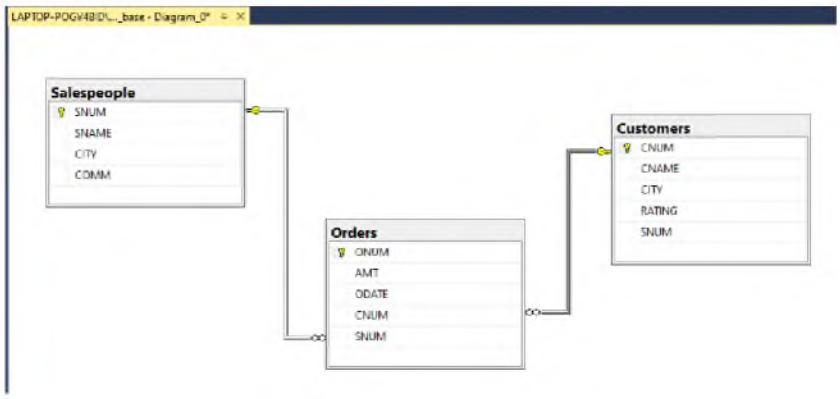


Рис. 21. Диаграмма базы данных `Training_base`

Связывание таблиц с помощью конструкции FOREIGN KEY

Для указания внешнего ключа таблицы нужно добавить конструкцию `FOREIGN KEY` в конструкцию `CREATE TABLE`:

```
Синтаксис ограничения FOREIGN KEY
FOREIGN KEY (<список столбцов>)
REFERENCES <родительская таблица>
[(<родительский ключ>)];
```

`FOREIGN KEY` – команда, указывающая какой из полей у нас будет внешним ключом.

`REFERENCES` – указывает на какое поле в соединяемой таблице у нас будет ссылаться внешний ключ (после данной команды указывается название таблицы, а в скобках название поля)

`ON UPDATE` и `ON DELETE` – указывают какие действия будут выполняться при изменении родительского ключа или удалении строк родительской таблицы соответственно. `NO ACTION` означает, что когда родительский ключ изменяется или удаляется из БД, то никаких специальных действий не производится. `SET NULL` означает, что при удалении родительского ключа (для `ON DELETE SET NULL`) или его изменении (для `ON UPDATE SET NULL`) столбцы дочернего ключа будут устанавливаться в значение `NULL` во всех строках дочерней таблицы, которые ссылаются на удаляемую/изменяемую строку родительской таблицы.

Пример создания связи с помощью конструкции FOREIGN KEY приведен на рис. 21.1.

```
CREATE TABLE UNIVERSITY (
UNIV_ID INT NOT NULL,
UNIV_NAME VARCHAR(50) NOT NULL,
RATING INT NOT NULL,
CITY VARCHAR(50) NOT NULL
CONSTRAINT PK_UNIV PRIMARY KEY (UNIV_ID)
);

GO
CREATE TABLE STUDENT (
STUDENT_ID INT NOT NULL,
SURNAME VARCHAR(25) NOT NULL,
ST_NAME VARCHAR(10) NOT NULL,
STIPEND INT,
KURS INT,
CITY VARCHAR(15),
BIRTHDAY DATE,
UNIV_ID INT
CONSTRAINT PK_STU PRIMARY KEY (STUDENT_ID)
CONSTRAINT UNIV_FOR_KEY FOREIGN KEY (UNIV_ID)
REFERENCES UNIVERSITY (UNIV_ID));
```

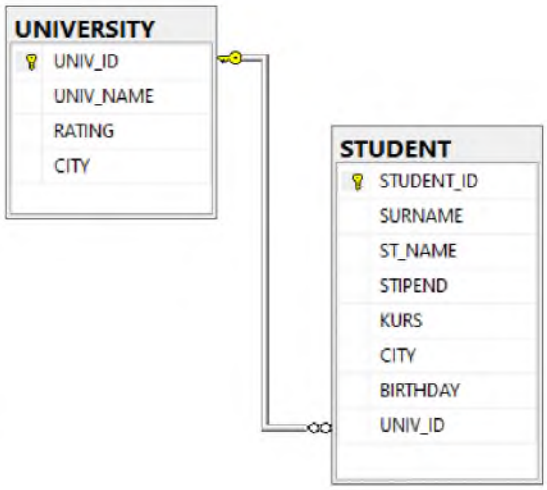


Рис. 21.1. Создание вязи с помощью конструкции FOREIGN KEY

Если таблицы созданы без объявления внешнего ключа, то добавление ограничения FOREIGN KEY добавляется, как показано на рис. 21.2, используя оператор ALTER.

```
CREATE TABLE UNIVERSITY (
UNIV_ID INT NOT NULL,
UNIV_NAME VARCHAR(50) NOT NULL,
RATING INT NOT NULL,
CITY VARCHAR(50) NOT NULL
CONSTRAINT PK_UNIV PRIMARY KEY (UNIV_ID)
);
```

```
CREATE TABLE STUDENT (
STUDENT_ID INT NOT NULL,
SURNAME VARCHAR(25) NOT NULL,
ST_NAME VARCHAR(10) NOT NULL,
STIPEND INT,
KURS INT,
CITY VARCHAR(15),
BIRTHDAY DATE,
UNIV_ID INT
CONSTRAINT PK_STU PRIMARY KEY (STUDENT_ID)
);
```

```
ALTER TABLE STUDENT ADD CONSTRAINT UNIV_FOR_KEY
                    FOREIGN KEY (UNIV_ID)
                    REFERENCES UNIVERSITY(UNIV_ID);
```

Рис. 21.2. Добавление внешнего ключа, используя оператор ALTER

7. Создайте запрос на установление связей между таблицами Salespeople_1, Customers_1 и Orders_1.

Примечание: как говорилось ранее, команды CREATE, DROP, ALTER относятся к DDL.

Заполнения таблиц начальными данными

1. Для начала заполните таблицу Salespeople. Для заполнения этой таблицы в обозревателе объектов в контекстном меню таблицы Salespeople (Рис. 22) и в появившемся меню выберите пункт Edit Top 200 Rows (Изменить первые 200 строк). В рабочей области MS SQL Server Management Studio появится окно заполнения таблиц.

2. Заполните таблицу Salespeople, как показано на рис.23.

3. Заполним таблицу Customers (рис.24).

4. Заполним таблицу Orders (рис.25). Так как поле ONUM является первичным полем связи и **ключевым числовым счётчиком**, то оно заполняется автоматически (заполнять его не нужно).

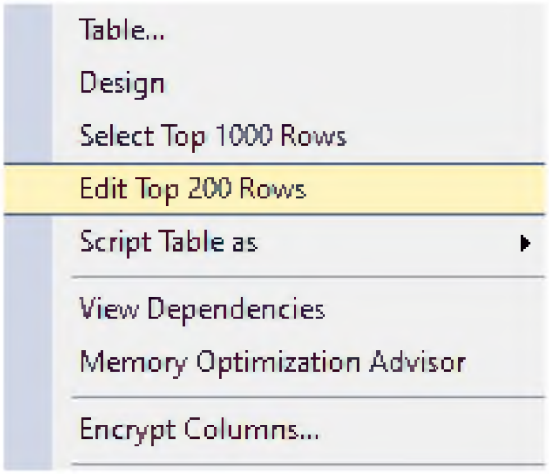


Рис. 22. Изменить данные в таблице

LAPTOP-POGV4BID\...- dbo.Salespeople

SNUM	SNAME	CITY	COMM
1001	Peel	London	0,12
1002	Serres	San Jose	0,13
1003	Axelrod	New York	0,10
1004	Motika	London	0,11
1007	Rifkin	Barselona	0,15
NULL	NULL	NULL	NULL

Рис. 23. Заполнение таблицы `Salespeople`

LAPTOP-POGV4BID\...e - dbo.Customers

CNUM	CNAME	CITY	RATING	SNUM
2001	Hoffman	London	100	1001
2002	Giovanni	Rome	200	1003
2003	Liu	San Jose	200	1002
2004	Grasse	Berlin	300	1002
2006	Clemens	London	100	1001
2007	Pereira	Rome	100	1004
2008	Cisneros	San Jose	300	1007
NULL	NULL	NULL	NULL	NULL

Рис. 24. Заполнение таблицы `Customers`

LAPTOP-POGV4BID\...base - dbo.Orders

ONUM	AMT	ODATE	CNUM	SNUM
1	18,6900	2019-03-10	2008	1007
2	767,1900	2019-03-10	2001	1001
3	1900,1000	2019-03-10	2007	1004
4	5160,4500	2019-03-10	2003	1002
5	1098,1600	2019-03-10	2008	1007
6	1713,2500	2019-04-10	2002	1003
7	78,7800	2019-04-10	2004	1002
8	4756,2300	2019-05-10	2006	1001
9	1309,2500	2019-06-10	2004	1002
10	9897,8800	2019-06-10	2006	1001
NULL	NULL	NULL	NULL	NULL

Рис. 25. Заполнение таблицы `Orders`

Добавление данных в таблицу

В SQL Server 20XX заполнение таблиц производится при помощи следующей команды:

```
INSERT INTO <Имя таблицы> (<Список полей>)
VALUES (<Значения полей>);
```

<Имя таблицы> – таблица, куда вводим данные.

<Список полей> – список полей, куда вводим данные, если не указываем, то подразумевается заполнение всех полей, в списке полей поля указываются через запятую.

<Значения полей> – значение полей через запятую.

Пример

```
INSERT INTO Posts (P_POST, P_SAL)
VALUES ('Менеджер', 15000);
```

Или для добавления множества записей можно использовать конструкция из следующего примера

```
INSERT INTO Posts VALUES
('Менеджер', 15000),
('Редактор', 20000),
('Программист', 50000),
('Главный редактор', 150000)
```

Строковые данные, как и даты в запросах указываются в одинарных кавычках, числовые данные указываются без кавычек. VALUES позволяет использовать один или несколько списков вставляемых значений данных. Список значений должен быть заключен в скобки. Если нет необходимости заполнять все поля, то можно указать только те имена полей, значения которых необходимо изменить. В перечислении можно указывать поля в любом порядке, но в списке VALUES значения должны идти в том же порядке, в котором перечислены поля.

```
INSERT INTO tbPeoples (vcSurname, idPosition, vcName)
VALUES ('Смирнов', 12, 'Сергей')
```

5. Создайте запрос на добавление записей в таблицы Salespeople_1, Customers_1 и Orders_1.

Удаление отдельных столбцов и отдельных строк из таблицы

Из таблицы можно удалить все столбцы, либо отдельные записи. Это осуществляется командой

```
WHERE <Условие>
```

Примечание: WHERE – означает условие, т.е. в данном случае мы удаляем данные, если выполняется условие, которое находится после ключевого слова WHERE.

Минимальная команда для удаления выглядит следующим образом:

```
DELETE <Имя таблицы>
```

Например,

```
DELETE tbPeoples
```

Эта команда удаляет все строки из таблицы tbPeoples.

Либо для удаления таблицы полностью:

```
DROP TABLE <Имя таблицы>
```

Для полной очистки таблицы и сброса счетчика AUTOINCREMENT используется конструкция

```
TRUNCATE TABLE <Имя таблицы>
```

Пример использования этой конструкции представлен ниже:

```
TRUNCATE TABLE Posts;
```

Инструкция TRUNCATE TABLE выполняется быстрее, чем инструкция DELETE, и использует меньше системных ресурсов и ресурсов журнала транзакций. Если условие указано, то удаляются записи поля, которые соответствуют условию. Ключевое слово TOP задает количество или процент удаляемых случайных строк. Если с инструкцией DELETE применяется предложение TOP (n), то операция удаления производится над n случайно выбранных строк.

Пример: Удалить 20 случайных строк из таблицы PurchaseOrderDetail, имеющих дату ранее 1 июля 2006 г

```
DELETE TOP (20)
FROM Purchasing.PurchaseOrderDetail
WHERE DueDate < '2006/07/01';
```

Пример:

Удалить записи из таблицы Posts, у которых значение поля P_SAL > 100000

```
DELETE Posts
WHERE P_SAL > 100000
```

Пример:

Удалить записи из таблицы Posts, у которых значение поля P_Post будет содержать часть слова «джер»

```
DELETE FROM Posts WHERE P_Post LIKE '%джер';
```

Пример:

Удалить записи из таблицы Posts, у которых значение поля P_ID будет между 5 и 8

```
DELETE FROM Posts WHERE P_ID BETWEEN 5 AND 8;
```

Изменение данных в таблице

Для этого используется следующая команда:

```
UPDATE <Имя таблицы>
SET
<Имя поля1> =<Выражение1>,
[<Имя поля2>=<Выражение2>,]
…
[WHERE <Условие>]
```

<Имя поля1>, <Имя поля2> - имена изменяемых полей,

<Выражение1>, <Выражение2> - либо конкретные значения, либо NULL, либо операторы SELECT. Здесь SELECT применяется как функция.

<Условие> – условие, которым должны соответствовать записи, поля которых изменяем.

Пример:

В таблице Posts переименовать название Менеджер в Редактора

```
UPDATE Posts
```

```
SET P_POST = 'Редактор'
WHERE P_Post = 'Менеджер'
```

Здесь командой SET в поле P_POST устанавливаем значение Редактор там, где по условию WHERE в этом поле значение равно Менеджер.

Пример:

Увеличить значение даты у всех строк на 1

```
UPDATE peoples
SET dayBirthday = dayBirthday +1
```

Из примера видно, что оператор UPDATE позволяет использовать математические вычисления.

После выполнения лабораторной работы 3 в отчете создать раздел: Лабораторная работа 3. Раздел должен содержать следующую информацию:

— название лабораторной работы;

— цель работы;

— задание;

— вывод по работе.

Обязательно должны быть скриншоты (к лр 3):

— создание связей между таблицами двумя способами (должно быть видно имя сервера и полное имя БД);

— заполнения всех таблиц данными (должно быть видно имя сервера);

— запроса на добавление записей;

— диаграммы.

Самостоятельная работа 1

1. С помощью **запроса** создайте в БД `Training_base` еще 2 таблицы: `Product` с полями: `PNUM` (код продукта) – ключевое, `PNAME` (название продукта) `PRICE` (цена) и `Order_Details` (состав заказа) с полями: `ONUM`, `PNUM`, `QUANTITY` (количество). Составной ключ (`ONUM`, `PNUM`).

2. С помощью запроса установите необходимые связи между таблицами `Product`, `Order_Details` и `Orders`. При необходимости добавить поля.

3. Создайте запросы на добавление множества записей в таблицы: `Product` и `Order_Details` (не нарушайте целостность и очередность заполнения таблиц). Значения выбрать самостоятельно

4. Создать запрос на изменение записей в таблице (значения выбрать самостоятельно).

5. Создать запрос на удаление записи в таблице (значения выбрать самостоятельно).

После выполнения работы в отчете создать раздел Самостоятельная работа 1. В нем должна содержаться следующая информация:

— название СР;

— цель работы;

— задание;

— вывод по работе.

Обязательно должны быть скриншоты (к ср 1):

— создания в БД `Training_base` таблиц `Product`, и `Order_Details` с помощью запроса;

— результат создания таблиц;

— создание необходимых связей с помощью запроса (должно быть видно имя сервера и полное имя БД);

— диаграммы;

— запроса на добавление записей в таблицы;

— результат добавления записей;

— запроса на изменения записей в таблице;

— результата изменения записей;

— запроса на удаление записи из таблицы.

Соблюдать требования предъявляемые к оформлению отчета.

Лабораторная работа 4

Создание запросов

Оператор SELECT (ВЫБРАТЬ) языка **SQL** является самым важным и наиболее часто используемым оператором. Он предназначен для выборки информации из таблиц базы данных. Упрощенный синтаксис оператора **SELECT** выглядит следующим образом:

SELECT [DISTINCT] <список выражений над атрибутами и константами>

FROM <список таблиц>

[WHERE <условие выборки или соединения>]

[GROUP BY <список атрибутов>]

[HAVING <условие для группы>]

[UNION <выражение с оператором SELECT>]

[ORDER BY <список атрибутов, по которым упорядочить вывод>];

В квадратных скобках указаны элементы, которые могут отсутствовать в запросе.

Ключевое слово SELECT сообщает базе данных, что данное предложение является запросом на выборку информации. После слова SELECT через запятую перечисляются наименования полей (список атрибутов), содержимое которых запрашивается.

Обязательным ключевым словом в предложении-запросе SELECT является слово FROM (ИЗ). За ключевым словом FROM указывается список разделенных запятыми имен таблиц, из которых извлекается информация.

Задание

Создать запросы на извлечение данных.

Ход работы

Выполнение простых sql – запросов.

1. Подключиться к базе данных и выполнить команду `New Query`/Создать запрос.

2. Ввести текст запроса согласно рис.27.

3. Нажать на панели инструментов кнопку `Execute`/Выполнить.

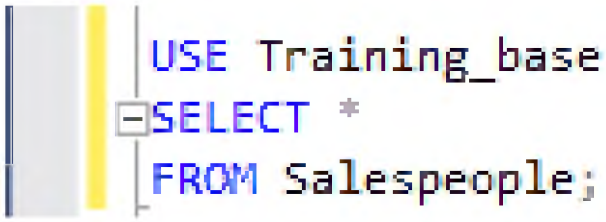
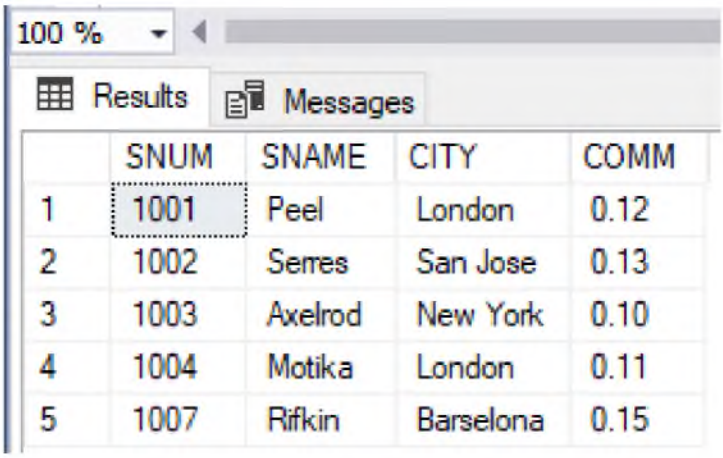


Рис. 27. Окно запроса 1

`USE` <Имя_БД> Обращение к БД

4. Если запрос правильный, то в результате произойдет его выполнение и Results / Результат будет отображен на вкладке Результаты (рис. 28).



	SNUM	SNAME	CITY	COMM
1	1001	Peel	London	0.12
2	1002	Serres	San Jose	0.13
3	1003	Axelrod	New York	0.10
4	1004	Motika	London	0.11
5	1007	Rifkin	Barselona	0.15

Рис. 28. Окно с результатом выполнения запроса

Приведенный запрос осуществляет выборку всех значений из таблицы Salespeople. Сохраните в текущей папке.

Запрос 2. Вывод данных таблицы Customers с присвоением псевдонима, рис. 29.

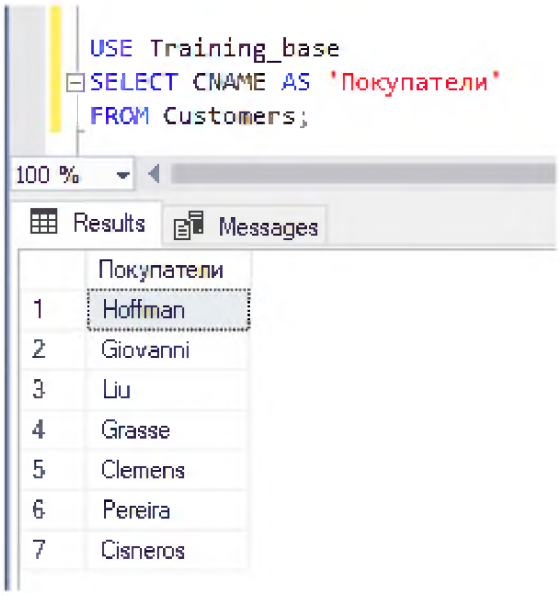


Рис. 29. Окно запроса 2

Команда AS изменяет имя столбца в результирующей таблице на имя, указанное после этой команды в кавычках (CNAME изменяется на Покупатели).

Запрос 3. Вывести информацию о продавцах с вознаграждением больше 0,12 (рис. 30).

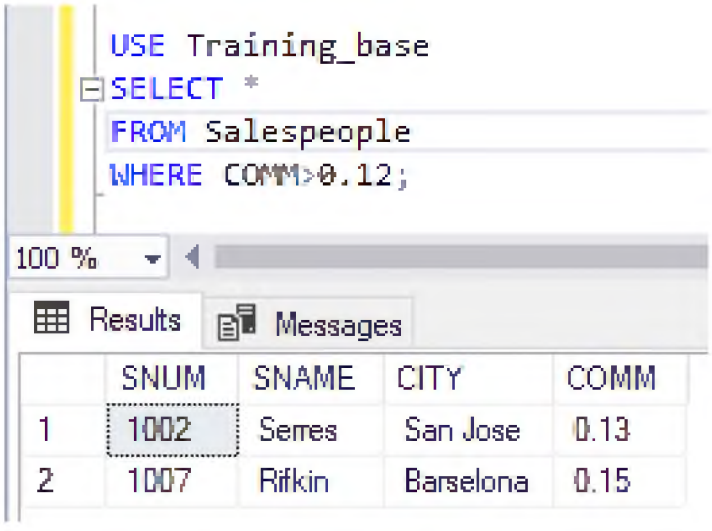


Рис. 30. Окно запроса 3

Предложение `WHERE` команды `SELECT` позволяет определить предикат, условие, которое может быть либо истинным, либо ложным для каждой строки

таблицы. Команда извлекает только те строки из таблицы, для которых предикат имеет значение «истина».

Запрос 4. Составное условие: извлечение из таблицы Salespeople записей, чьи коды больше 1001, но меньше или равен 1004, рис. 31.

```
USE Training_base
SELECT *
FROM Salespeople
WHERE SNUM > 1001 AND SNUM<1004;
```

100 %

Results | Messages

	SNUM	SNAME	CITY	COMM
1	1002	Serres	San Jose	0.13
2	1003	Axelrod	New York	0.10

Рис. 31. Окно запроса 4

AND, OR, NOT – это стандартные булевы операторы. Они связывают одно или несколько значений «истина/ложь» и в результате получают единственное значение «истина/ложь».

AND берет два булевых выражения (в виде A AND B) в качестве аргументов и дает в результате истину, если они оба истинны.

OR берет два булевых выражения (в виде A OR B) в качестве аргументов и оценивает результат как истину, если хотя бы один из них истинен.

NOT берет единственное булево выражение (в виде NOT A) в в качестве аргумента и изменяет его значение с истинного на ложное или с ложного на истинное.

Запрос 5. Составное условие: извлечение из таблицы Customers записей, чьи коды находятся между 2003 и 2007, рис. 32.

Оператор BETWEEN задает границы, в которые должно попадать значение, чтобы предикат был истинным 9граничные значения тоже включаются в этот диапазон).

Оператор IN полностью определяет множество, которому данное значение может принадлежать.

```
USE Training_base
SELECT *
FROM Customers
WHERE CNUM BETWEEN 2003 AND 2007;
```

100 %

Results | Messages

	CNUM	CNAME	CITY	RATING	SNUM
1	2003	Liu	San Jose	200	1002
2	2004	Grasse	Berlin	300	1002
3	2006	Clemens	London	100	1001
4	2007	Pereira	Rome	100	1004

Рис. 32. Окно запроса 5

Запрос 6. Составное условие: требуется что бы выводились строки где CNUM вне указанного далее диапазона т.е. то, что находится между 2003 и 2005 не выводилось, рис. 36.

```
USE Training_base
SELECT *
FROM Customers
WHERE CNUM NOT BETWEEN 2003 AND 2005;
```

100 %

Results | Messages

	CNUM	CNAME	CITY	RATING	SNUM
1	2001	Hoffman	London	100	1001
2	2002	Giovanni	Rome	200	1003
3	2006	Clemens	London	100	1001
4	2007	Pereira	Rome	100	1004
5	2008	Cisneros	San Jose	300	1007

Рис. 36. Окно запроса 6

Запрос 7. Вывод записей, удовлетворяющих не диапазону, а списку `IN` указывает, что необходимо в результирующую таблицу поместить только те строки, значения поля SNUM, которых равны значениям, указанным в скобках после `IN`, рис. 37.

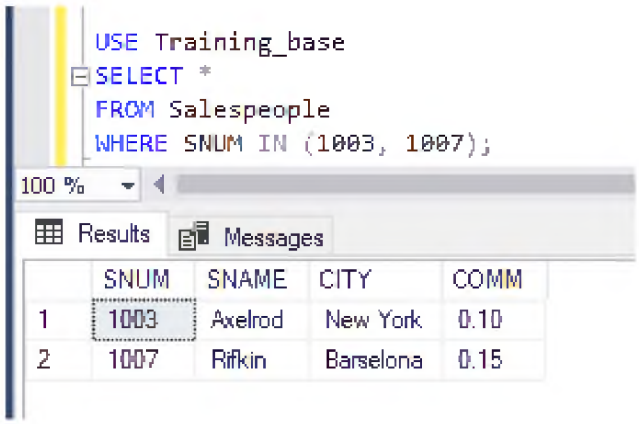


Рис. 37. Окно запроса 7

Запрос 8. Вывод записей, удовлетворяющих условию, заданному текстом: вывести все записи о продавцах, с именемAxelrod, рис. 38.

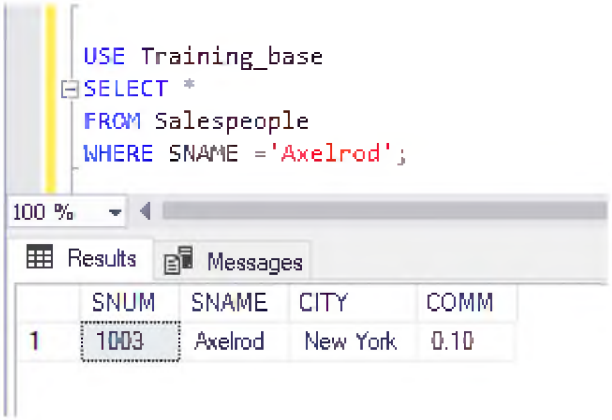


Рис. 38. Окно запроса 8

Запрос 9. Вывод записей, удовлетворяющих условию, заданному частью текста. `LIKE` дает возможность указать какую-либо часть значения. В данном случае мы получаем все строки, значения поля CNAME (имя покупателя), которых будет начинаться с буквы G, рис. 39.

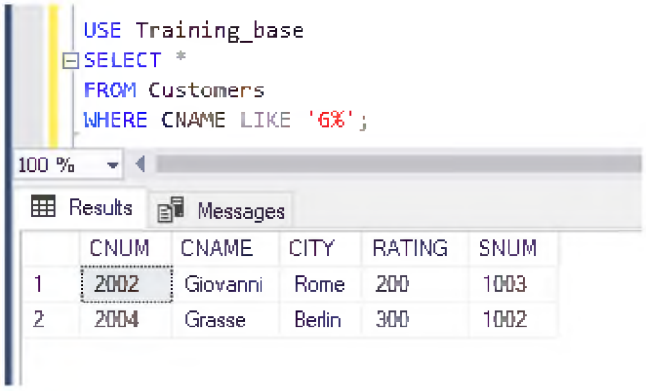


Рис. 39. Окно запроса 9

Оператор LIKE применим только к символьным типам полей, поскольку он используется для поиска подстрок. Он осуществляет просмотр строки для выяснения: входит ли заданная подстрока в указанное поле. С этой же целью используются шаблоны – специальные символы, которые могут обозначать все, что угодно.

Символ «подчеркивание» (_) заменяет один любой символ. Например, образцу ‘b_t’ соответствуют ‘bat’ или ‘bit, но не соответствует ‘brat’.

Символ «процент» (%) заменяет последовательность символов произвольной длины, в том числе и нулевой. Например, образцу ‘%p%t’ соответствуют ‘put’, ‘posit’, ‘opt’, но не ‘spite’

Запрос 10. Сортировка по значению одного из столбцов. В данном случае сортировка по полю SNAME, рис. 40.

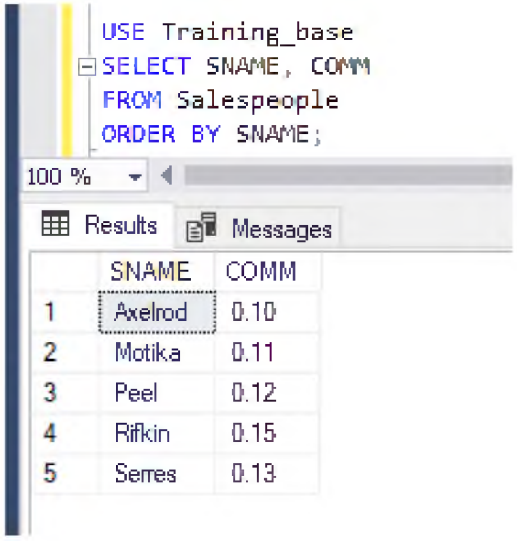


Рис. 40. Окно запроса 10

Таблицы являются неупорядоченными множествами, и исходящие из них данные необязательно представляются в какой-либо определенной последовательности. Команда ORDER BY упорядочивает в соответствии со значениями одного или нескольких выбранных столбцов. Можно задавать возрастающую (ASC) или убывающую (DESC) последовательность сортировки для каждого из столбцов.

Запрос 11. Извлечение из таблицы Salespeople записей, номер которых от 1003 до 1007 с сортировкой по полю SNUM и полю SNAME, рис. 41.

Рис. 41. Окно запроса 11

Запрос 12. Изменение порядка сортировки. `DESC` означает сортировка по убыванию, рис. 42.

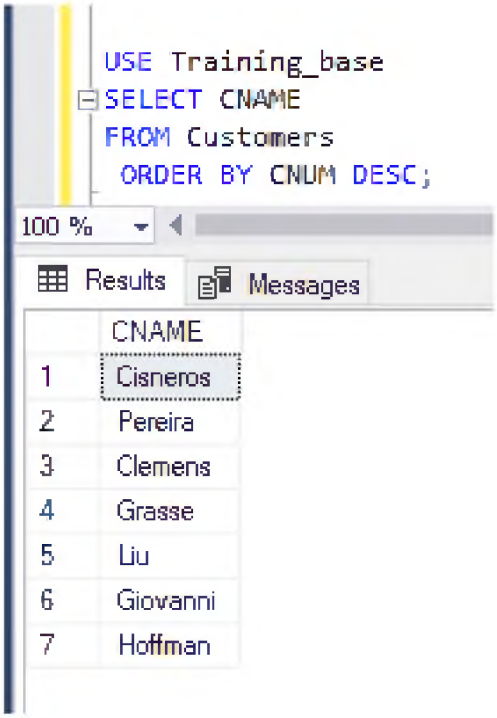


Рис. 42. Окно запроса 12

Запрос 13. Извлечение трех записей с обратной сортировкой по полю CNUM, рис. 43.

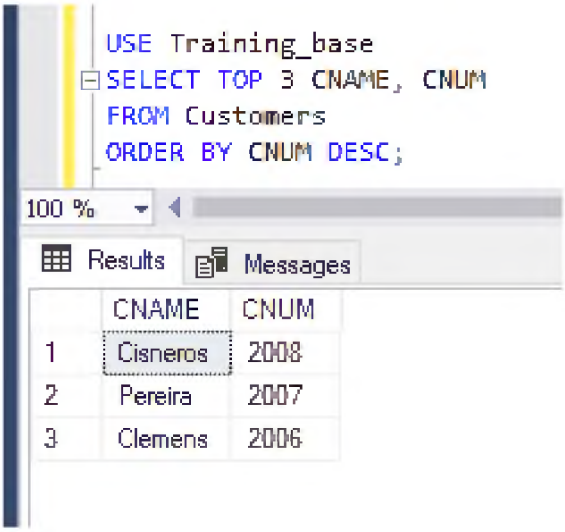


Рис. 43. Окно запроса 13

Функции агрегирования:

— `COUNT` определяет количество строк или значений полей;

— `SUM` вычисляет арифметическую сумму;

— `AVG` вычисляет среднее значение для всех выбранных значений данного поля;

— `MAX` вычисляет наибольшее значение;

— `MIN` вычисляет наименьшее значение.

Запрос 14. Подсчет среднего значения по всем комиссионным, с присвоением псевдонима, рис. 44.

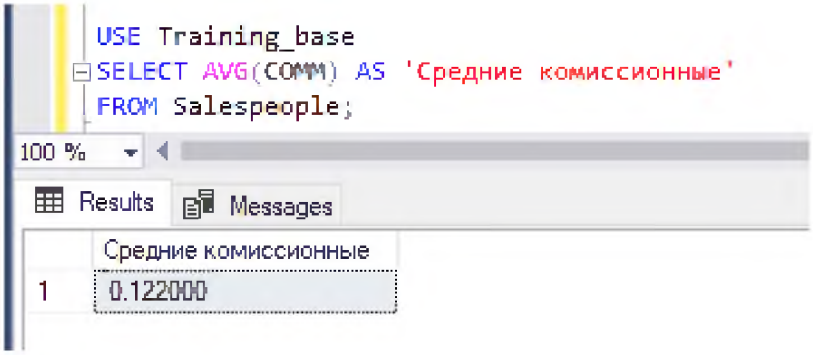


Рис. 44. Окно запроса 14

Запрос 15. Подсчет количества заказов (числа строк) в таблице Orders, значения столбца которых отличны от NULL, с присвоением псевдонима, рис. 45.

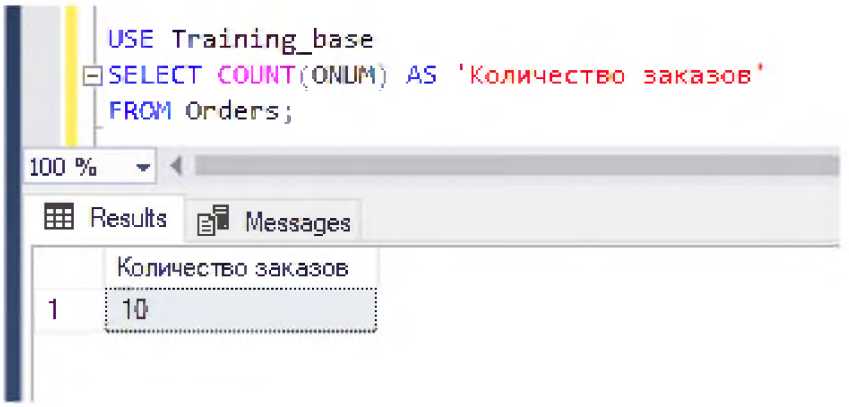


Рис. 45. Окно запроса 15

Запрос 16. Извлечение максимального значения столбца AMT, рис. 46.

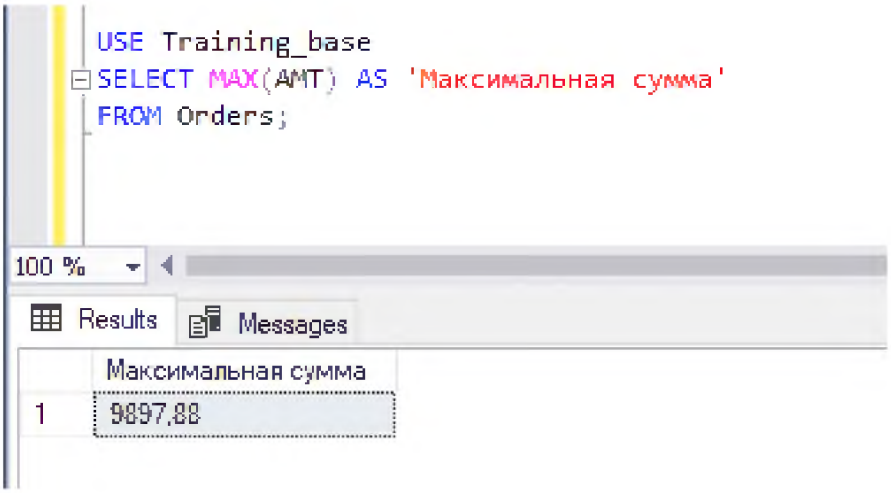
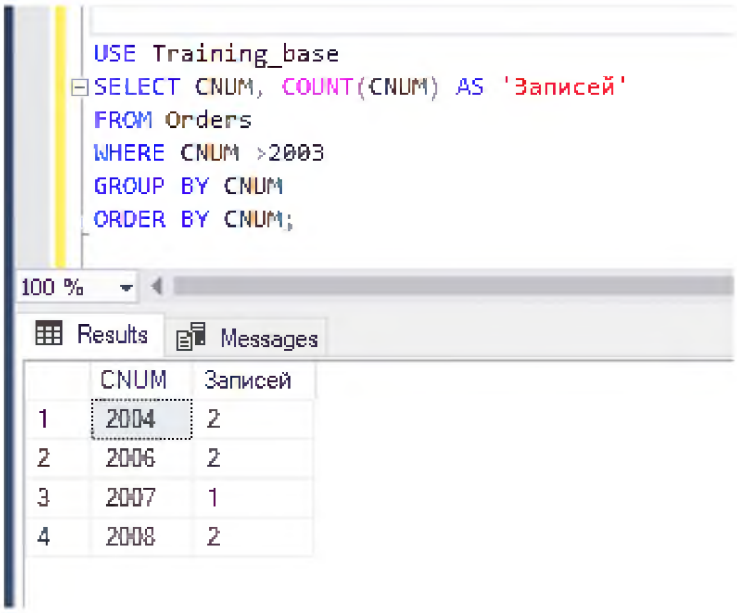


Рис. 46. Окно запроса 16

Предложение `GROUP BY` (группировать по) позволяет определять подмножество значений отдельного поля в терминах другого поля и применять функции агрегирования к полученному подмножеству.

Запрос 17. Вывод числа записей, соответствующих каждому из уникальных значений CNUM больше 2003, рис. 47.



```
USE Training_base
SELECT CNUM, COUNT(CNUM) AS 'Записей'
FROM Orders
WHERE CNUM >2003
GROUP BY CNUM
ORDER BY CNUM;
```

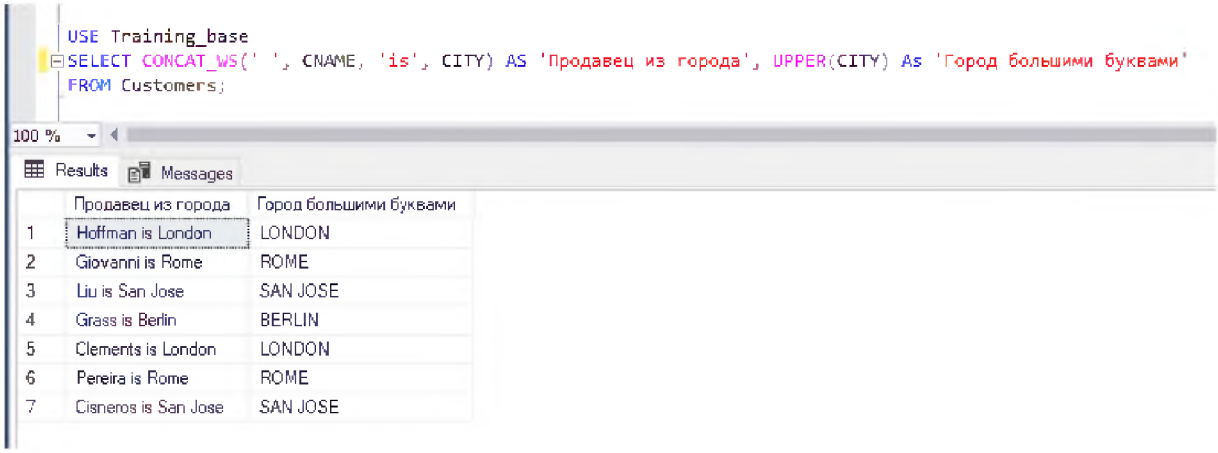
100 %

Results | Messages

	CNUM	Записей
1	2004	2
2	2006	2
3	2007	1
4	2008	2

Рис. 47. Окно запроса 17

Запрос 18. Выборка данных из таблицы Customers, где сцеплены CNAME (имя покупателя) и CITY (город), населенный пункт записан прописными буквами, рис. 48.



```
USE Training_base
SELECT CONCAT_WS(' ', CNAME, 'is', CITY) AS 'Продавец из города', UPPER(CITY) As 'Город большими буквами'
FROM Customers;
```

100 %

Results | Messages

	Продавец из города	Город большими буквами
1	Hoffman is London	LONDON
2	Giovanni is Rome	ROME
3	Liu is San Jose	SAN JOSE
4	Grass is Berlin	BERLIN
5	Clements is London	LONDON
6	Pereira is Rome	ROME
7	Cisneros is San Jose	SAN JOSE

Рис. 48. Окно запроса 18

Подзапросы и вложенные запросы

Одни запросы могут управлять другими. Это можно сделать, размещая один запрос внутри предиката, помещенного в другом, и используя выходные данные вложенного запроса для sопределения истинности или ложности предиката.

SQL позволяет вкладывать запросы друг в друга. Обычно внутренний запрос генерирует значения, которые тестируются на предмет истинности предиката.

Запрос 19. Предположим, известно имя, но неизвестно значение поля SNUM для продавца Motika. Необходимо извлечь все ее заказы из таблицы Orders, рис. 49.

```
USE Training_base
SELECT *
FROM Orders
WHERE SNUM =
    (SELECT SNUM
        FROM Salespeople
        WHERE SNAME='Motika');
```

Рис. 49. Окно запроса 19

Вложенные запросы следует рассматривать снизу-вверх, т.е. здесь мы сначала получаем строку с SNUM (номер продавца) с именем Motika из таблицы Salespeople (Продавцы. В результате выбранной оказывается единственная строка с SNUM=1004 (это можно проверить есть выполнить только подзапрос, рис.50).

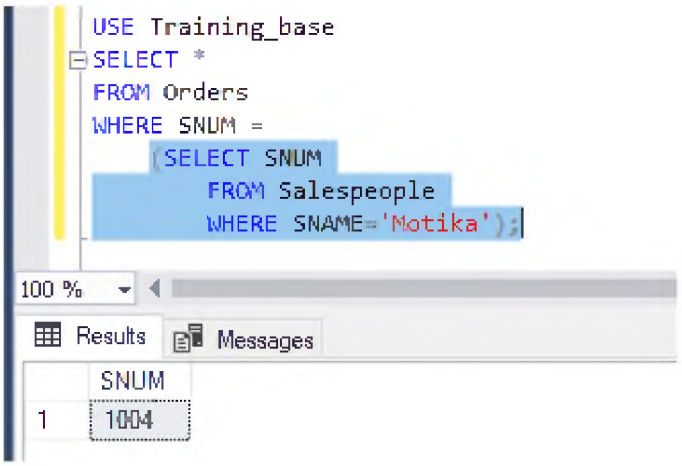


Рис. 50. Окно выполнения подзапроса

SQL подставляет вывод этого значения в предикат основного запроса вместо самого подзапроса, теперь предикат читается следующим образом:

`WHERE SNUM=1004`

Затем основной запрос выполняется, как обычный, рис. 51.

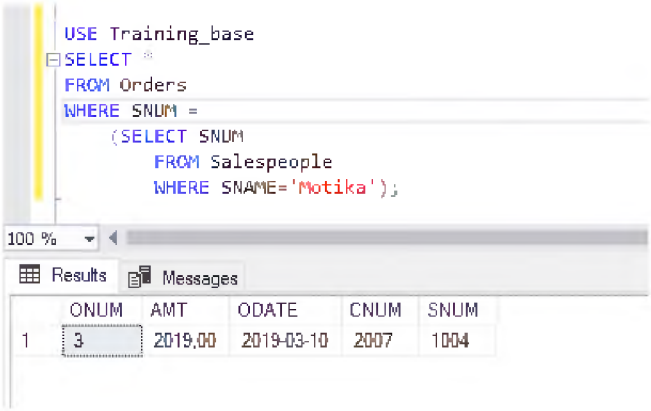
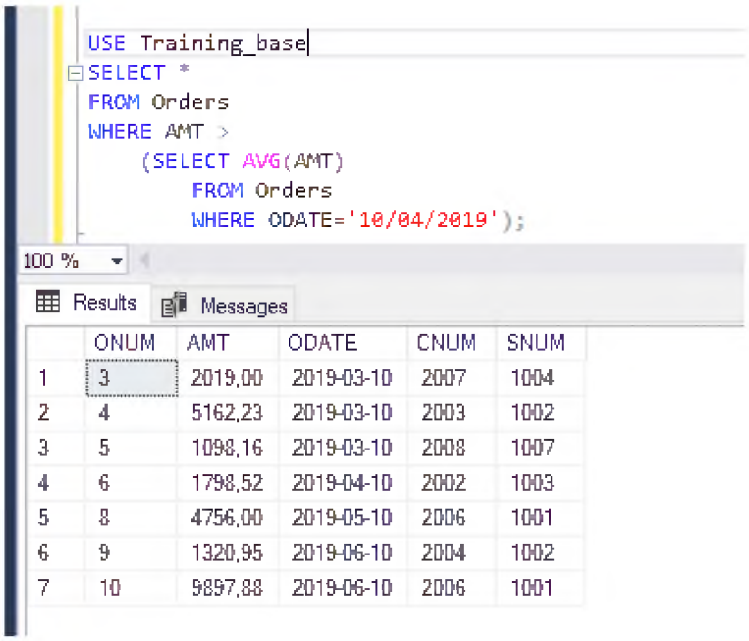


Рис. 51. Окно выполнения запроса 19

Подзапрос должен выбирать один и только один столбец, а тип данных этого столбца должен соответствовать типу значения, указанному в предикате. Часто выбранное поле и это значение имеют одно и то же имя (в данном случае `SNUM`) .

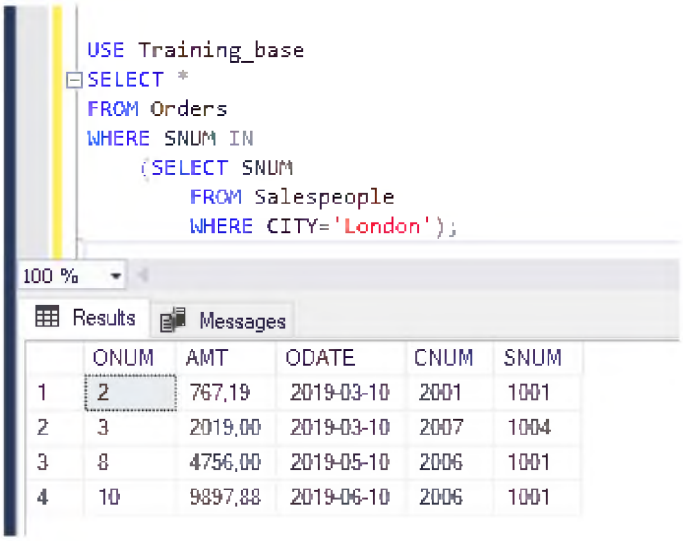
Запрос 20. Узнать все заказы, стоимость которых превышает среднюю стоимость заказов за 4 октября 2019 г, рис. 52.



	ONUM	AMT	ODATE	CNUM	SNUM
1	3	2019,00	2019-03-10	2007	1004
2	4	5162,23	2019-03-10	2003	1002
3	5	1098,16	2019-03-10	2008	1007
4	6	1798,52	2019-04-10	2002	1003
5	8	4756,00	2019-05-10	2006	1001
6	9	1320,95	2019-06-10	2004	1002
7	10	9897,88	2019-06-10	2006	1001

Рис. 52. Окно запроса 20

Запрос 21. Найти все заказы (Orders) для продавцов (Salespeople) из London, рис. 53.



	ONUM	AMT	ODATE	CNUM	SNUM
1	2	767,19	2019-03-10	2001	1001
2	3	2019,00	2019-03-10	2007	1004
3	8	4756,00	2019-05-10	2006	1001
4	10	9897,88	2019-06-10	2006	1001

Рис. 53. Окно запроса 21

Использование оператора EXISTS

Используемое в SQL ключевое слово `EXISTS` (СУЩЕСТВУЕТ) представляет собой предикат, принимающий значение истина или ложь.

Используя подзапросы в качестве аргумента, этот оператор оценив генерирует выходные данные, то есть в случае существования (возврата) хотя

бы одного найденного значения. В противном случае результат подзапроса — ложный. Оператор EXISTS не может принимать значение неизвестно).

Запрос 22. Извлечь данные из таблицы Customers в том случае, если один (или более) покупатель из нее находится в San Jose, рис. 54.

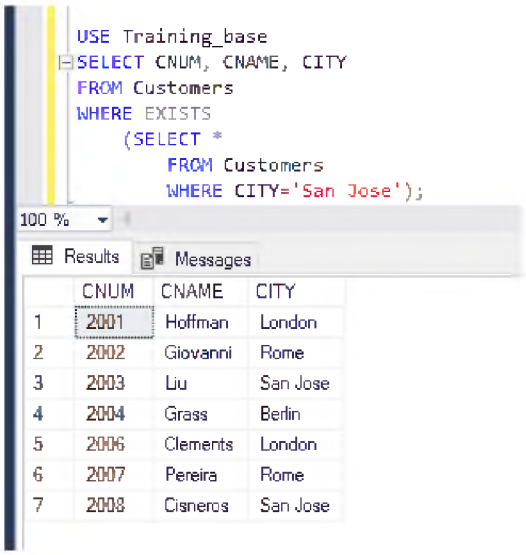


Рис. 54. Окно запроса 22

Оператор объединения UNION

Оператор языка SQL UNION предназначен для объединения результирующих таблиц базы данных, полученных с применением слова SELECT. Условие объединения результирующих таблиц: совпадение числа, порядка следования и типа данных столбцов. Оператор **UNION** по умолчанию выбирает только отдельные значения. Чтобы разрешить повторяющиеся значения, использовать **UNION ALL.**

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

Использование оператора **UNION** возможно только при объединении запросов, соответствующие столбцы которых совместимы по объединению. Совместимость по объединению означает, что столбцы, как минимум, должны относиться к одному типу.

Запрос 23. Вывести имена всех продавцов и покупателей с указанием городов и статуса Продавец/Покупатель.

```
SELECT SNUM, CITY, 'Продавец' as 'Статус'
FROM Salespeople
UNION ALL
SELECT CNUM, CITY, 'Покупатель'
FROM Customers;
```

100 %

Results | Messages

	SNUM	CITY	Статус
1	1001	London	Продавец
2	1002	San Jose	Продавец
3	1003	New York	Продавец
4	1004	London	Продавец
5	1007	Barcelona	Продавец
6	2001	London	Покупатель
7	2002	Rome	Покупатель
8	2003	San Jose	Покупатель
9	2004	Berlin	Покупатель
10	2006	London	Покупатель
11	2007	Rome	Покупатель
12	2008	San Jose	Покупатель

Рис. 54.1. Окно запроса 23

Соединение таблиц. Оператор JOIN

При явном объявлении соединения используются следующие ключевые слова:

`CROSS JOIN` – определяет декартово произведение двух таблиц т.е. при выводе будут присутствовать все возможные комбинации строк выбранных таблиц;

`[INNER] JOIN` - определяет естественное соединение двух таблиц;

`LEFT [OUTER] JOIN` – определяет операцию левого соединения;

`RIGHT [OUTER] JOIN` – определяет операцию правого соединения;

`FULL [OUTER] JOIN` - определяет соединение правого и левого внешнего соединений.

Пример 24. Вывести номера заказов для каждого продавца.

```
/*Вывести номера заказов для каждого продавца*/
SELECT SNAME, ONUM
        FROM Salespeople INNER JOIN Orders
    ON Salespeople.SNUM = Orders.SNUM;
```

100 %

Results | Messages

	SNAME	ONUM
1	Rifkin	1
2	Pell	2
3	Motika	3
4	Serres	4
5	Rifkin	5
6	Axelrod	6
7	Serres	7
8	Pell	8
9	Serres	9
10	Pell	10

Рис. 54.2. Окно запроса 24

Примеры использования выражения CASE

CASE – это инструкция, которая проверяет список условий и возвращает соответствующий результат. Если говорить в целом о программировании, то CASE – это что-то вроде многократного использования конструкции IF-ELSE, во многих языках есть похожая конструкция SWITCH, так вот CASE, как я уже отметил, делает примерно то же самое.

Выражение CASE можно использовать практически в любой инструкции SQL, где есть возможность использовать допустимые выражения, например: SELECT, UPDATE, WHERE, SET и даже в ORDER BY.

CASE имеет два так называемых формата:

Простое выражение CASE – это простое сравнение значения (выражения) с набором других значений (выражений);

Поисковое выражение CASE – в данном случае CASE содержит набор логических выражений, которые вычисляются, чтобы вернуть результат.

Простое выражение CASE

```
CASE input_expression
```

```
        WHEN when_expression THEN result_expression [
...n ]
        [ ELSE else_result_expression ]
    END
```

Поисковое выражение CASE

```
CASE
        WHEN          Boolean_expression          THEN
result_expression [ ...n ]
        [ ELSE else_result_expression ]
    END
```

Описание параметров:

input_expression — выражение, которое необходимо проверить в простом формате CASE;

WHEN when_expression — выражение, с которым сравнивается input_expression, в случае с простым форматом. Тип данных when_expression должен быть такой же, как и у input_expression, или хотя бы неявно преобразовываться;

THEN result_expression — выражение, которое будет возвращено, если текущее условие выполняется;

ELSE else_result_expression – дополнительный параметр ELSE, который предназначен для случаев, когда ни одно из перечисленных в CASE условий не выполнилось. Это необязательный параметр. Если ELSE не указано, а условия не выполнились, вернётся NULL;

WHEN Boolean_expression — логическое выражение, используемое в поисковом формате CASE, которое служит для вычисления результата. Это своего рода проверочное условие и таких условий может быть несколько.

CASE возвращает результат первого выражения (THEN result_expression), условие которого выполнилось, т.е. WHEN возвращает TRUE. Таким образом, если CASE содержит несколько эквивалентных условий WHEN, которые будут возвращать TRUE, вернется результат (указанный в THEN) первого выражения.

Тип данных возвращаемого результата выражением CASE, будет соответствовать наиболее приоритетному типу данных из набора типов в выражениях result_expressions и else_result_expression.

Пример 25

Допустим, что у нас есть таблица с товарами, она имеет следующую структуру и данные, рис. 55.

```
--Создание таблицы
   CREATE TABLE TestTable(
        [ProductId]    [INT] IDENTITY(1,1) NOT NULL,
        [ProductName]   [VARCHAR](100) NOT NULL,
        [Price]                [Money] NULL
   )
   GO
   --Добавление строк в таблицу
   INSERT INTO TestTable(ProductName, Price)
          VALUES ('Системный блок', 300),
                    ('Монитор', 200),
                        ('Клавиатура', 100),
                        ('Мышь', 50),
                        ('Принтер', 200)
   GO
   --Выборка данных
   SELECT * FROM TestTable
```

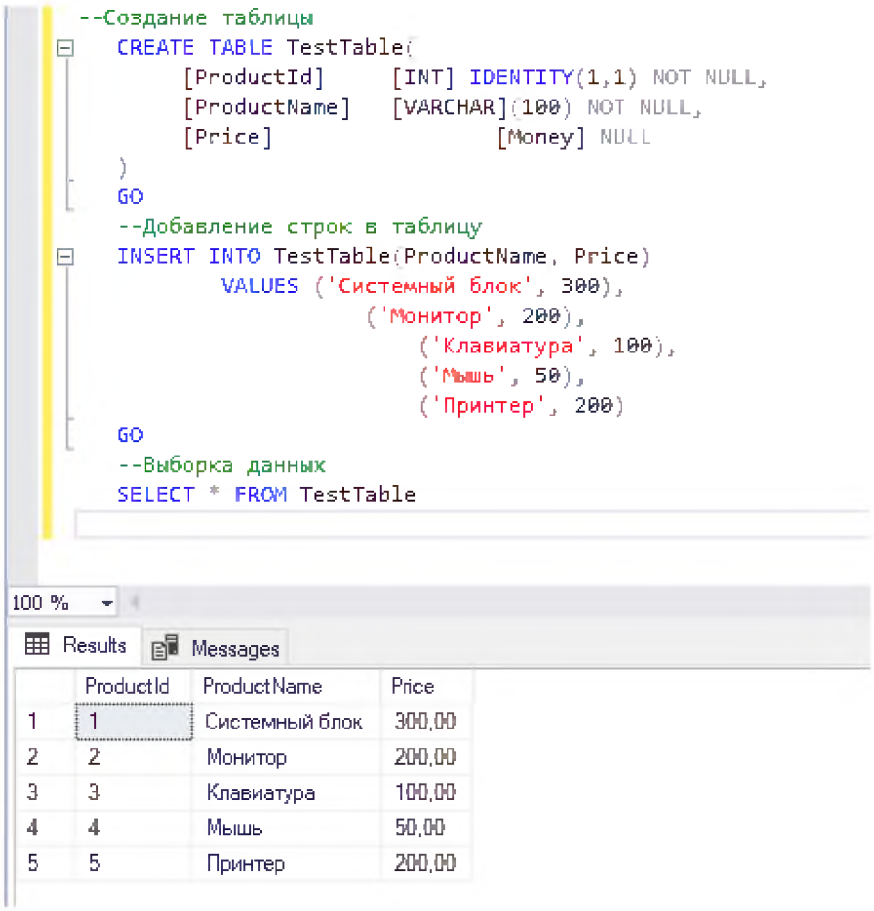


Рис. 55. Создание таблицы с данными

Пример простого выражения CASE в инструкции SELECT

Пример 26

В нижеприведенном примере проверяется значение столбца ProductId, если оно равняется одному из перечисленных значений в выражении WHEN, то будет выводиться соответствующее значение из выражения THEN. Если нам встретится значение, которого мы не указали, CASE вернет пусто, т.е. значение из ELSE, рис. 56.

```
--Простое выражение CASE
SELECT CASE ProductId WHEN 1 THEN 'Один'
                      WHEN 2 THEN 'Два'
                      WHEN 3 THEN 'Три'
                      WHEN 4 THEN 'Четыре'
                      WHEN 5 THEN 'Пять'
                      ELSE ''
            END AS IdText,

            ProductId, ProductName, Price

FROM TestTable
```

100 %

Results | Messages

	IdText	ProductId	ProductName	Price
1	Один	1	Системный блок	300,00
2	Два	2	Монитор	200,00
3	Три	3	Клавиатура	100,00
4	Четыре	4	Мышь	50,00
5	Пять	5	Принтер	200,00

Рис. 56. Пример простого выражения CASE в инструкции SELECT

Пример поискового выражения CASE в инструкции SELECT

Пример 27

Давайте немного усложним предыдущий запрос, добавим в него некий анализ цены (Price). В случае, если по факту у нас может выполниться несколько условий, результат будет возвращен первого выражения с TRUE (*т.е. условие выполнилось*), выражения, идущие после, не обрабатываются, рис. 57.

```
--------------------------------
--Пример поискового выражения CASE
SELECT CASE WHEN Price > 100 THEN 'Больше 100'
                        WHEN Price = 100 THEN 'Равно 100'
                        WHEN Price < 100 THEN 'Меньше 100'
                        WHEN Price = 300 AND ProductId = 1 THEN 'Цена равна 300 и Id равен 1'
                        ELSE 'Нет подходящего условия'
            END AS IdText,

            ProductId, ProductName, Price

FROM TestTable
```

100 %

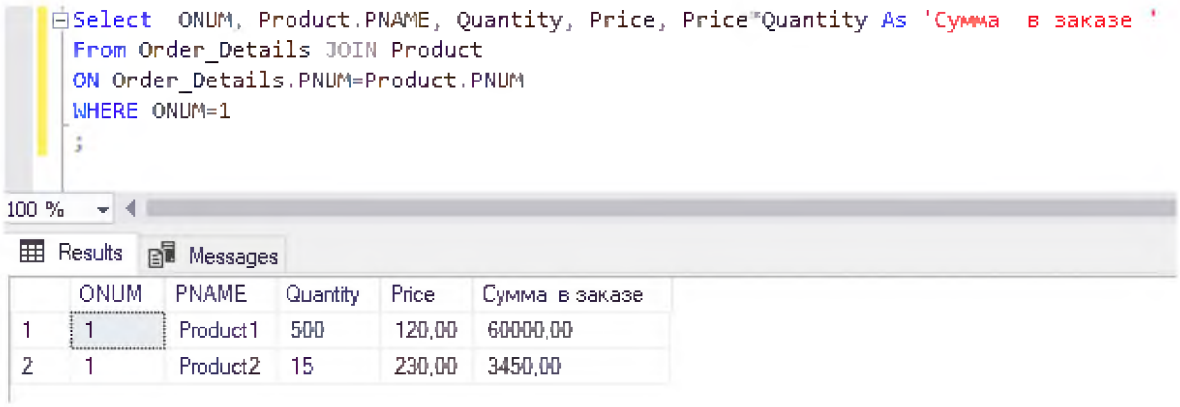
Results | Messages

	IdText	ProductId	ProductName	Price
1	Больше 100	1	Системный блок	300,00
2	Больше 100	2	Монитор	200,00
3	Равно 100	3	Клавиатура	100,00
4	Меньше 100	4	Мышь	50,00
5	Больше 100	5	Принтер	200,00

Рис. 57. Пример поискового выражения CASE в инструкции SELECT

Как видите, условие WHEN Price = 300 AND ProductId = 1 с виду выполняется, но CASE вернул результат первого выражения, после которого анализ был прекращен, и дело до следующих условий не дошло.

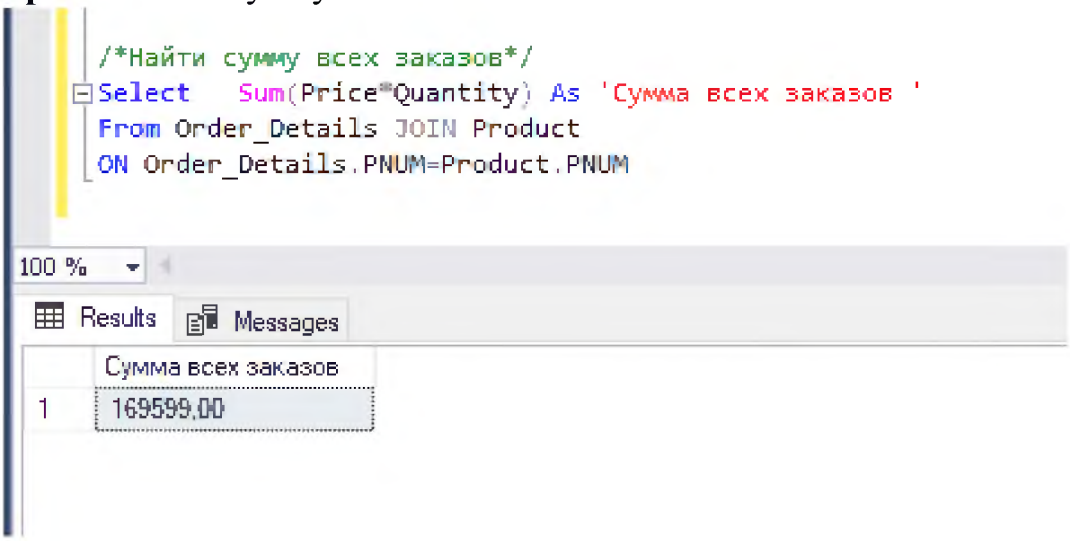
Пример 28. Вывести информацию о заказе 1, указать наименование товара и сумму по каждому товару.



	ONUM	PNAME	Quantity	Price	Сумма в заказе
1	1	Product1	500	120,00	60000,00
2	1	Product2	15	230,00	3450,00

Рис. 57.1. Окно запроса 28

Пример 29. Найти сумму всех заказов.



	Сумма всех заказов
1	169599,00

Рис. 57.2. Окно запроса 29

После выполнения лабораторной работы в отчете создать раздел Лабораторная работа 4. В нем должна содержаться следующая информация:

— название лабораторной работы;

— цель работы;

— задание;

— вывод по работе.

Обязательно должны быть скриншоты (к лр 4): создание и выполнение запросов 1-29 (должно быть видно имя сервера и полное имя БД).

Соблюдать требования, предъявляемые к оформлению отчета.

Самостоятельная работа 2

Все запросы можно писать в одном файле (для запуска необходимого, выделяя нужный фрагмент код запроса)

Создайте самостоятельно запросы по приведённым ниже условиям (+ см. лекции):

1. Необходимо узнать, какие продавцы имеют заказы, нужен только список номеров продавцов без повторений.
2. Необходимо узнать, имена всех продавцов в Лондоне .
3. Создайте запрос, который выдает всю информацию о покупателях, где продавец имеет номер 1001.
4. Создайте запрос, который выдает рейтинг и имя каждого покупателя. проживающего в `San Jose`.
5. Создайте запрос, который покажет все заявки, превышающие 4000$.
6. Выбрать информацию о покупателях из San Jose чей рейтинг превышает 200.
7. Выбрать информацию о покупателях, которые либо проживают в San Jose, либо не имеют рейтинг, превышающий 200.
8. Создайте запрос для таблицы Customers, включающий в выходные данные всех покупателей, для которых RATING <= 100, в том случае, если они расположены не в Rome.
9. Создать запрос на удаление поля `AMT` из таблицы Orders. **Поясните, для чего удаляется поле `AMT` .**
10. Найти покупателей, имена которых начинаются на G.
11. Вывести в одну таблицу сведения обо всех продавцах и покупателях Лондона.
12. Вывести в одну таблицу имена, рейтинги и названия городов для всех покупателей. Те, у кого рейтинг 200 и выше, должны иметь комментарий «High Rating», все остальные – «Low Rating».

13. Напишите запрос, который бы использовал оператор INNER JOIN для получения всех заказов для покупателя с фамилией Cisneros.

14. Вывести дату и номера заказов, которые проводил продавец с именем Rifkin.

15. Вывести имена продавцов и количество заказов у каждого продавца.

16. Найти сумму каждого заказа.

После выполнения самостоятельной работы в отчете создать раздел Самостоятельная работа 2. В нем должна содержаться следующая информация:

— название самостоятельной работы;

— цель работы;

— задание;

— вывод по работе.

Обязательно должны быть скриншоты (к ср 2): создание и выполнение запросов 1-16 самостоятельной работы 2.

Соблюдать требования, предъявляемые к оформлению отчета.

Лабораторная работа 5

Создание представлений

Представление (view) – объект, содержимое которого берется или выводится из других таблиц, то есть не содержит собственных данных. Поскольку значения в таблицах меняются, это автоматически приводит к изменениям в представлениях.

Представления подобны окнам, через которые просматривается информация, реально содержащаяся в базовых таблицах.

Представление определяется с помощью команды CREATE VIEW, состоящей из ключевых слов CREATE VIEW (создать представление), имени создаваемого представления и ключевого слова AS, после которого следует запрос.

Как и любую другую таблицу представление можно использовать: формулировать к нему запросы, выполнять обновление, вставку, удаление и соединение с другими таблицами и представлениями.

Преимущество использования представления вместо базовой таблицы состоит в том, что оно обновляется автоматически при изменении формирующих его таблиц. Содержимое представления не фиксируется, а повторно вычисляется всякий раз, когда пользователь ссылается на представление в команде. Представления значительно расширяют возможности управления данными, например предоставляют пользователям доступ не ко всей информации, хранящейся в таблице, а только к ее части.

В обозревателе объектов все представления БД находятся в папке `Views` (Представления).

1. Создайте представление (статический запрос) «`Salespeople+Customers`». Для этого необходимо в обозревателе объектов в БД `Training_base` щёлкнуть правой кнопкой мыши по папке

Views (Представления), затем в появившемся меню выбрать пункт New View (Создать представление). Появится окно Add Table (Добавление таблицы), предназначенное для выбора таблиц и запросов, участвующих в новом запросе (рис. 58).

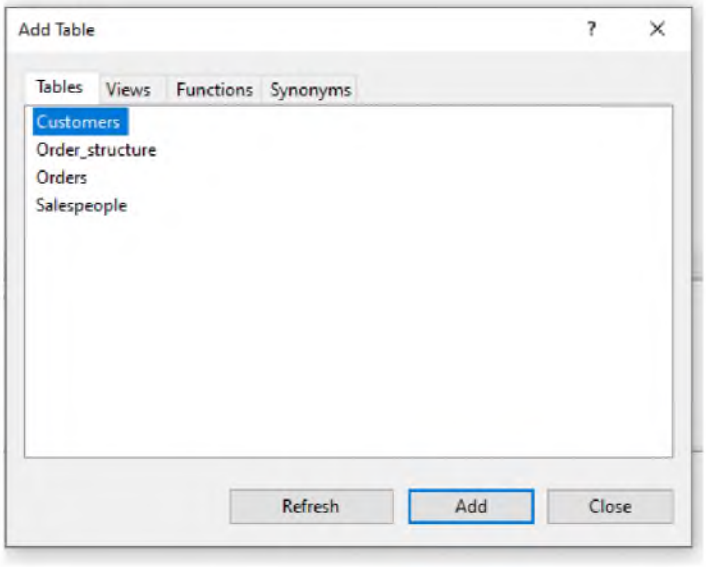


Рис.58. Окно диалога для добавления таблиц и запросов в конструктор запросов

2. Добавить в новый запрос таблицы Salespeople и Customers. Для этого в окне Add Table (Добавление таблицы) выделите таблицу Salespeople и нажмите кнопку Add (Добавить). Аналогично добавьте таблицу Customers. После добавления таблиц, участвующих в запросе закройте окно Add Table («Добавление таблицы»), нажав кнопку Close (Закрыть). Появится окно конструктора запросов (рис. 59).

Если необходимо снова отобразить окно (Добавление таблицы) для добавления новых таблиц или запросов, то для этого на панели инструментов нужно нажать кнопку Add Table.

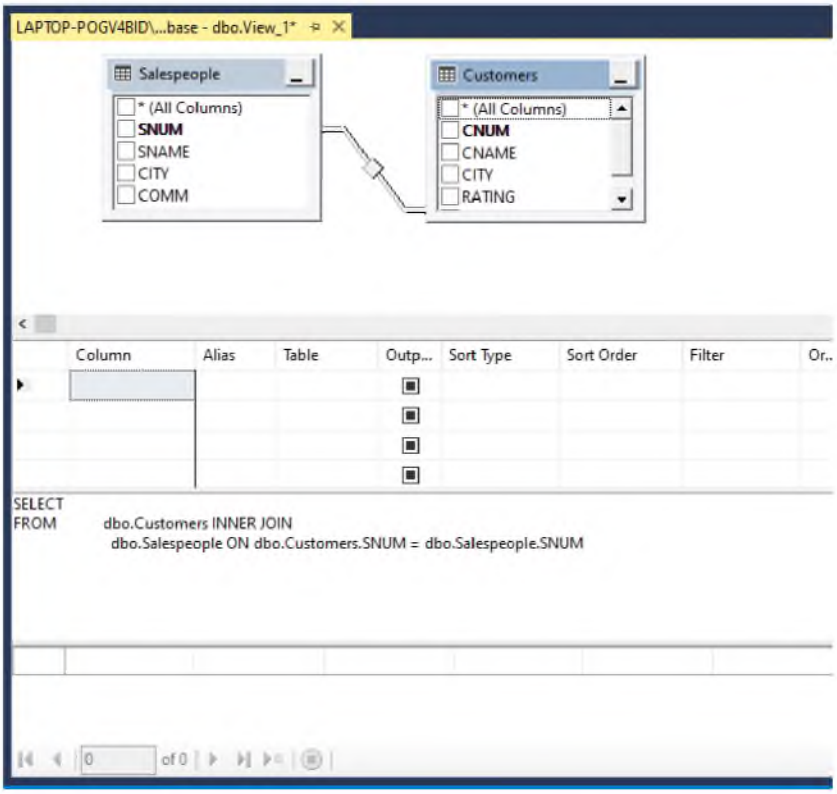


Рис. 59. Окно конструктора запросов

Окно конструктора запросов состоит из следующих панелей:

Схема данных — отображает поля таблиц и запросов, участвующих в запросе, позволяет выбирать отображаемые поля, позволяет устанавливать связи между участниками запроса по специальным полям связи. Эта панель включается и выключается следующей кнопкой на панели инструментов.

Таблица отображаемых полей — показывает отображаемые поля (столбец Column (Столбец)), позволяет задавать им псевдонимы (столбец "Alias" (Псевдоним)), позволяет устанавливать тип сортировки записей по одному или нескольким полям (столбец Sort Type (Тип сортировки)), позволяет задавать порядок сортировки (столбец Sort Orden (Порядок сортировки)), позволяет задавать условия отбора записей в фильтрах (столбцы Filter (Фильтр) и Or... (Или...)). Также эта таблица позволяет менять порядок отображения полей в запросе. Эта панель включается и выключается кнопкой на панели инструментов.

Код SQL — код создаваемого запроса на языке SQL. Эта панель включается и выключается кнопкой на панели инструментов.

Результат — показывает результат запроса после его выполнения.

Отображаемые поля обозначаются галочкой (слева от имени поля) на схеме данных, а также отображаются в таблице отображаемых полей. Чтобы сделать поле отображаемым при выполнении запроса необходимо щёлкнуть мышью по пустому квадрату (слева от имени поля) на схеме данных, в квадрате появится галочка.

Если необходимо сделать поле невидимым при выполнении запроса, то нужно убрать галочку, расположенную слева от имени поля на схеме данных. Для этого просто щёлкните мышью по галочке.

Если необходимо отобразить все поля таблицы, то необходимо установить галочку слева от пункта “* (All Columns)” (Все поля), принадлежащего соответствующей таблице на схеме данных.

Определите отображаемые поля запроса (Отображаются все поля кроме полей с кодами, то есть полей связи). Перед сохранением запроса проверим его работоспособность, выполнив его. Для запуска запроса на панели инструментов нажмите кнопку выполнения. Либо щёлкните правой кнопкой мыши в любом месте окна конструктора запросов и в появившемся меню выберите пункт Execute SQL (Выполнить код SQL). Результат выполнения запроса появиться в виде таблицы в области результата (рис. 60)

Если после выполнения запроса результат не появился, а появилось сообщение об ошибке, то в этом случае проверьте, правильно ли создана связь. Если запрос выполняется правильно, то его необходимо сохранить под именем `Salespeople+Customers`.

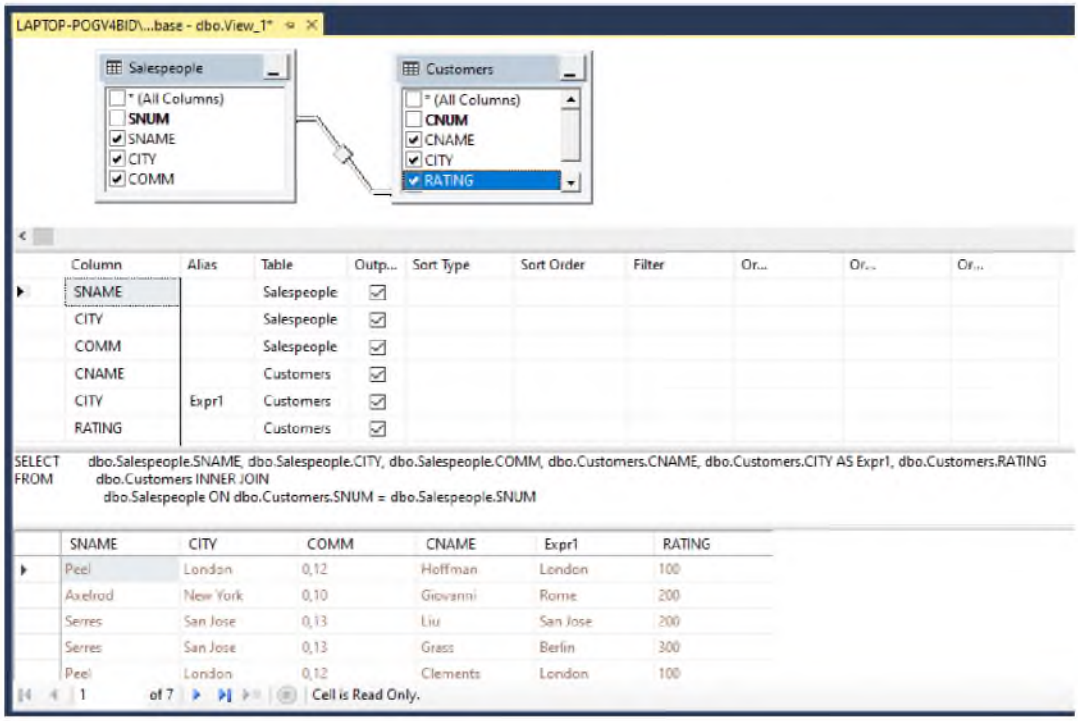


Рис. 60. Результат выполнения

Запрос появится в папке `Views` (Представления) БД `Training_base` в обозревателе объектов, рис. 61.

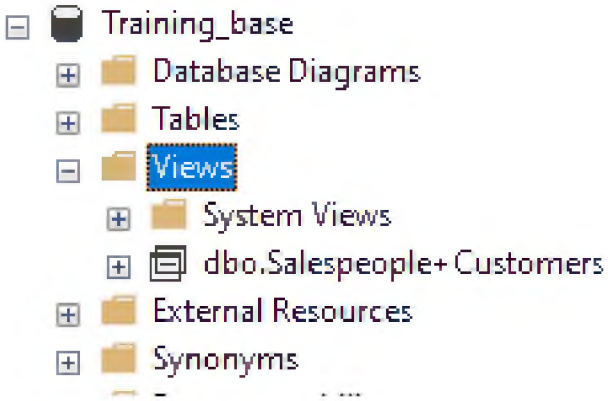


Рис.61. Отображение нового представления в обозревателе объектов

Проверим работоспособность созданного запроса вне конструктора запросов. Запустим вновь созданный запрос «`Salespeople+Customers`» без использования конструктора запросов. Для выполнения уже сохранённого запроса необходимо щёлкнуть правой кнопкой мыши по запросу и в появившемся меню выбрать пункт Select top 1000 rows (Выбрать первые 1000 строк). Выполните эту операцию для запроса «`Salespeople+Customers`» (рис. 62).

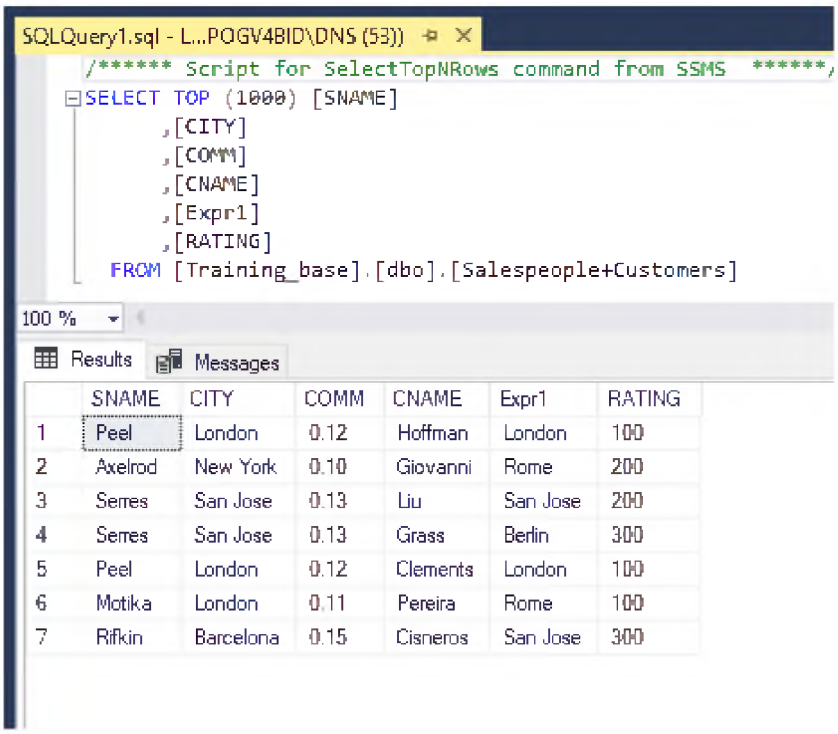


Рис. 62. Выполнение запроса `Salespeople+Customers`

3. Самостоятельно создайте представление «Customers + Orders». Поменять порядок отображаемых полей в запросе (на свое усмотрение). Для этого в таблице отображаемых полей необходимо перетащить поля мышью вверх или вниз за заголовок строки таблицы (столбец перед столбцом “Column” («Столбец»)).

Задайте псевдонимы для каждого из полей, просто записав псевдонимы в столбце Alias (Псевдоним) таблицы отображаемых полей.

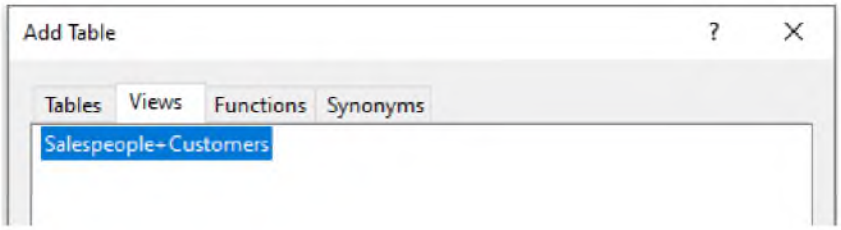
Проверьте работоспособность нового запроса, выполнив его. Обратите внимание на то, что реальные названия полей были заменены их псевдонимами.

Создание фильтров

На основе `Views «Salespeople+Customers»` создайте фильтры, отображающие продавцов и покупателей (с рейтингом 200).

4. Создайте новое представление. Источником выберите Views «Salespeople+Customers» рис. 63.

5. В появившемся окне конструктора запросов определите в качестве отображаемых полей все поля запроса «Salespeople+Customers».



6. Рис. 63. Добавление представления в конструктор запросов

7. Установите критерий отбора записей в фильтре. Пусть фильтр будет отображает только тех покупателей, имеющих рейтинг равный 200. Для определения условия отбора записей в таблице отображаемых полей в строке, соответствующей полю, на которое накладывается условие, в столбце “Filter” («Фильтр»), необходимо задать условие. В нашем случае условие накладывается на поле «RATING». Следовательно, в строке «RATING» в столбце “Filter” («Фильтр») нужно задать следующее условие отбора =200 (рис. 64).

8. Сохраните фильтр под именем RATING200.

9. Запрос (представление) «RATING200» появится в обозревателе объектов.

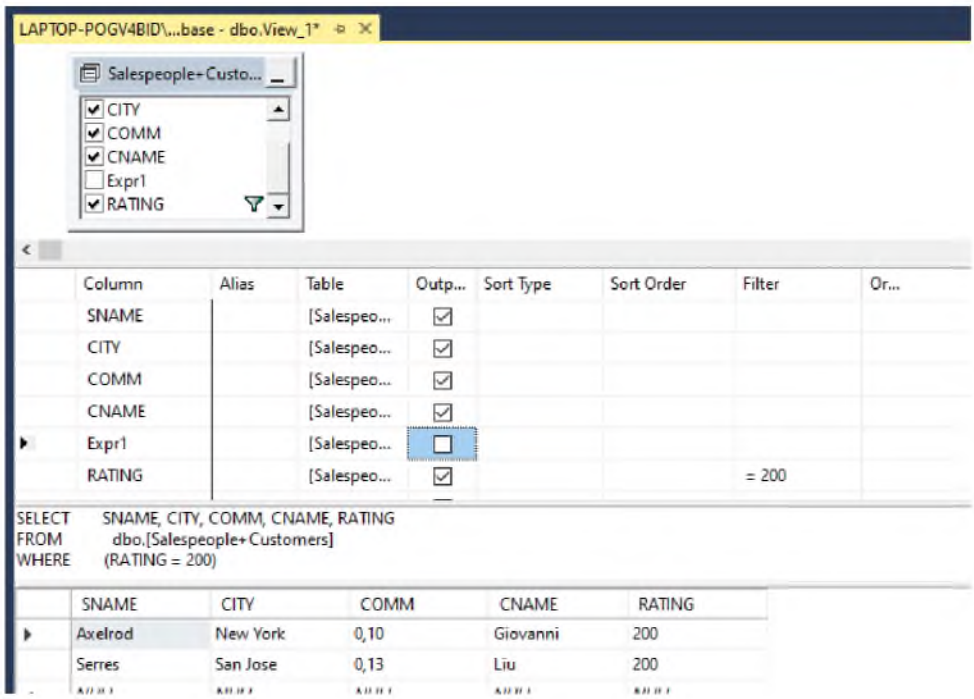


Рис. 64. Установка фильтра

10. Создайте представление Londonstaff (создать новый запрос) в котором будет выведена информация о продавцах из Лондона (рис.65).

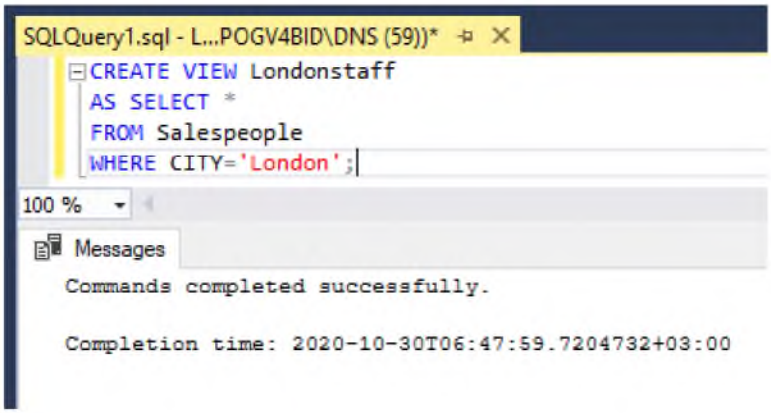


Рис.65. Создание представления `Londonstaff`

11. Выполните запрос к представлению `Londonstaff`, рис. 65.1.

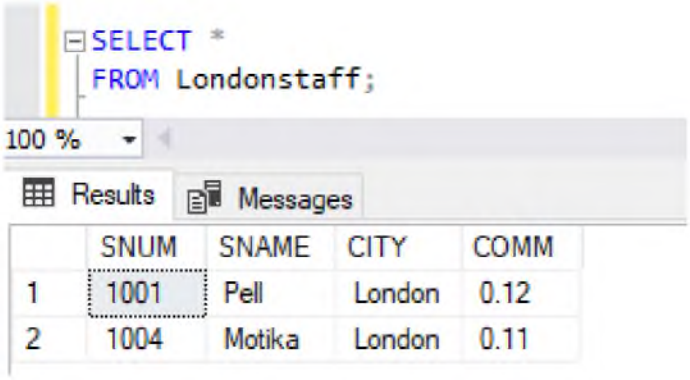


Рис.65.1. Представление `Londonstaff`

Что нельзя сделать с помощью представлений?

Существует много типов представлений, которые выполняют только чтение. Это значит, что для них могут быть сформулированы запросы, но их нельзя использовать в качестве объекта для команд обновления.

Существует несколько аспектов в области запросов, которые не укладываются в рамки определений представления: единственное представление должно базироваться на единственном запросе; UNION и UNION ALL недопустимы. ORDER BY тоже нельзя использовать в определении представления. Выходные данные для запроса, формирующего представление, должны быть неупорядоченными по определению.

12. Создайте представление `Dateorders` в котором будет выведена информация о количестве заказов в день. (рис.65.2).

```
CREATE VIEW Dateorders
AS SELECT ODATE, COUNT(*) As 'OCOUNT'
FROM Orders
GROUP BY ODATE;
```

Рис.65. 2. Создание представления `Dateorders`

13. Выведите информацию представления `Dateorders`.

После выполнения лабораторной работы в отчете создать раздел Лабораторная работа 5. В нем должна содержаться следующая информация:

— название лабораторной работы;

— цель работы;

— задание;

— вывод по работе.

Обязательно должны быть скриншоты (к лр 5):

— создания представлений `Salespeople+Customers`, `Customers + Orders` и их выполнение (должно быть видно имя сервера и полное имя БД);

— создание и выполнение фильтра: `RATING200`,

— результат создания объектов в окне обозревателя;

— представления `Londonstaff`;

— представления `Dateorders;`

Соблюдать требования предъявляемые к оформлению отчета.

Самостоятельная работа 3

1. Создайте фильтр «COMM_15». Создается аналогично фильтру «RATING200».

2. С помощью запроса создайте представление Highrating, показывающее всех покупателей с наивысшими рейтингами.

3. С помощью запроса создайте представление Multcustomers, которое показывает каждого продавца вместе со всеми его покупателями.

4. С помощью запроса создайте представление Citynumber, показывающее количество продавцов в каждом городе.

5. С помощью запроса создайте представление Commissions, включить поля: SNUM, COMM (только значения из интервала 0.10 и 0.13).

После выполнения самостоятельной работы в отчете создать разделы Самостоятельная работа 3. В нем должна содержаться следующая информация:

— название работы;

— цель работы;

— задание;

— вывод по работе.

— фильтра COMM_15;

— представлений Highrating, Multcustomers, Citynumber и Commissions.

Соблюдать требования предъявляемые к оформлению отчета.

Лабораторная работа 6

Хранимые процедуры

Хранимая процедура — SQL запрос, который имеет параметры, то есть он выполняется как обычная процедура (мы задаем ее имя и передаем в хранимую процедуру значение параметров.) В зависимости от значения параметров хранимой процедуры мы получаем тот или иной результат запроса. В SQL-сервере хранимые процедуры реализуют динамические запросы, выполняемые на стороне сервера.

Хранимые процедуры позволяют объединить последовательность запросов и сохранить их на сервере. Это очень удобный инструмент.

По сути, хранимые процедуры представляет набор инструкций, которые выполняются как единое целое. Тем самым хранимые процедуры позволяют упростить комплексные операции и вынести их в единый объект.

Также хранимые процедуры позволяют ограничить доступ к данным в таблицах и тем самым уменьшить вероятность преднамеренных или неосознанных нежелательных действий в отношении этих данных.

И еще один важный аспект - производительность. Хранимые процедуры обычно выполняются быстрее, чем обычные SQL-инструкции. Все, потому что код процедур компилируется один раз при первом ее запуске, а затем сохраняется в скомпилированной форме.

Таким образом, хранимая процедура имеет три ключевых особенности: упрощение кода, безопасность и производительность.

Programmability / Stored Procedures (Программирование / Хранимые процедуры) базы данных Training_base (рис. 66).

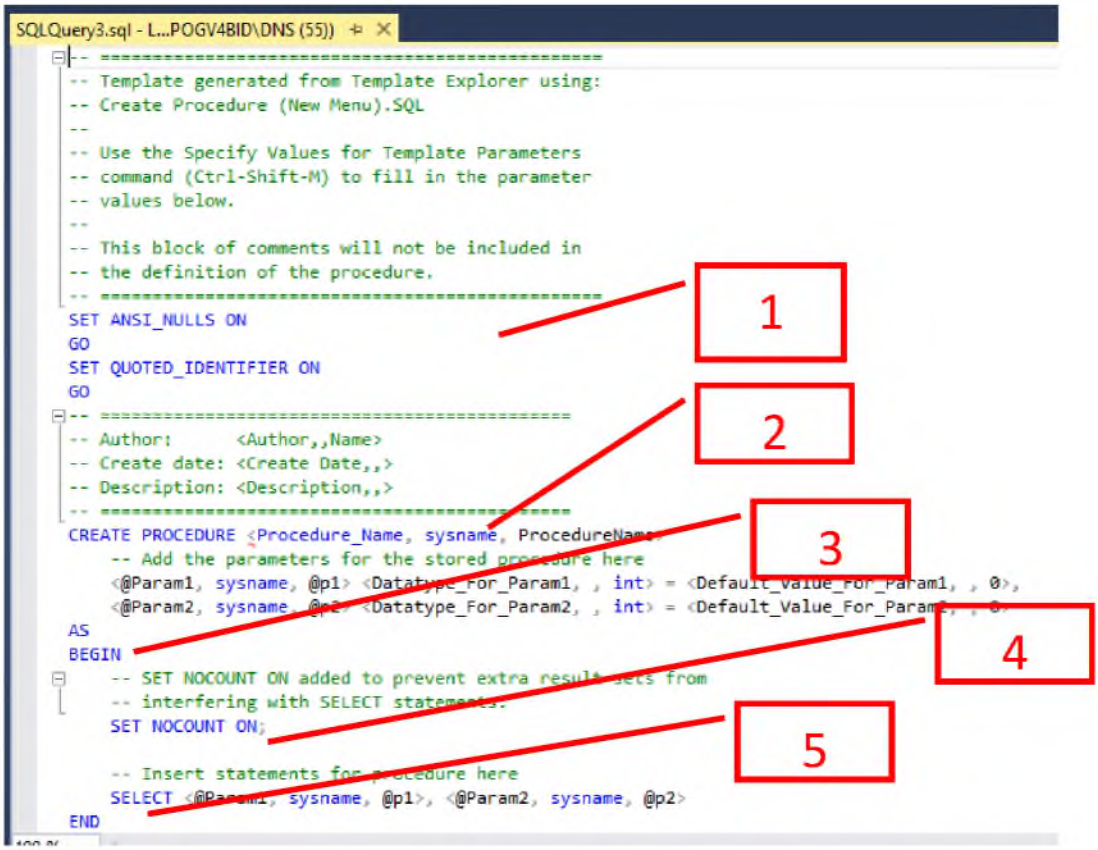


Рис. 66. Окно новой хранимой процедуры

Хранимая процедура имеет следующую структуру:

1. Область настройки параметров синтаксиса процедуры. Позволяет настраивать некоторые синтаксические правила, используемые при наборе кода процедуры.

В нашем случае это:

— SET ANSI_NULLS ON — включает использование значений NULL (Пусто) в кодировке ANSI;

— SET QUOTED_IDENTIFIER ON — включает возможность использования двойных кавычек для определения идентификаторов;

2. Область определения имени процедуры (Procedure_Name) и параметров, передаваемых в процедуру (@Paraml, @Param2). Определение параметров имеет следующий синтаксис:

@<Имя параметра> <Тип данных> = Значение по умолчанию>

Параметры разделяются между собой запятыми;

3. Начало тела процедуры, обозначается служебным словом BEGIN;

4. Тело процедуры, содержит команды языка программирования запросов SQL;

5. Конец тела процедуры, обозначается служебным словом END.

В коде зелёным цветом выделяются комментарии. Они не обрабатываются сервером и выполняют функцию пояснений к коду. Строки комментариев начинаются с подстроки «--».

6. Создайте код процедуры, вычисляющей среднее трёх чисел, как это показано на рис.67.

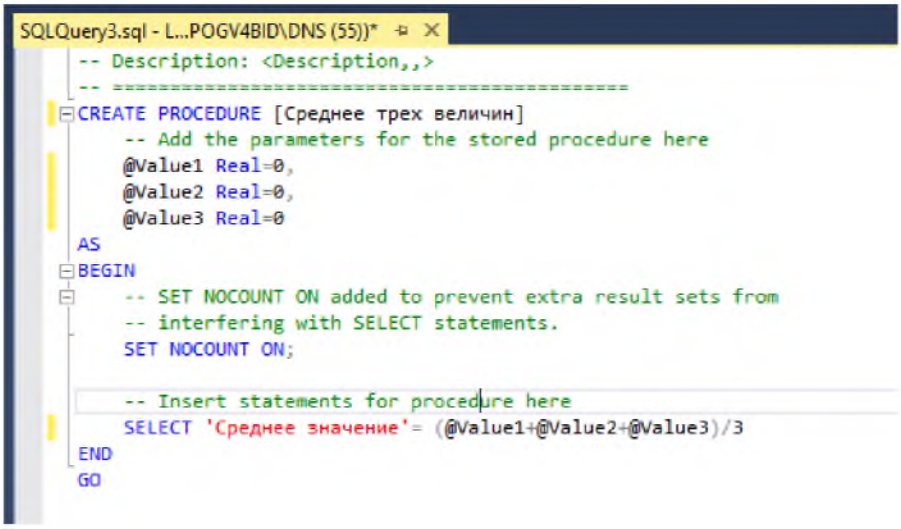


Рис. 67. Процедура вычисления среднего трех чисел

CREATE PROCRDURE [Среднее трёх величин] — определяет имя создаваемой процедуры как «Среднее трёх величин»;

@Value1 Real = 0, @Value2 Real = 0, @Value3 Real = 0 — определяют три параметра процедуры Value1, Value2 и Value3. Данным параметрам можно присвоить дробные числа (Тип данных Real), значения по умолчанию равны 0;

SELECT 'Среднее значение' = (@Valuel + @Value2 + @Value3)/3 — вычисляет среднее и выводит результат с подписью «Среднее значение».

7. Для создания процедуры выполнить запрос.

8. Сохраните под именем Среднее трех величин.

9. Запустите процедуру. Для этого создайте новый запрос и наберите

```
EXEC [Среднее трёх величин] 1, 7, 9.
```

10. Создайте хранимую процедуру для отбора покупателей из таблицы Customers по их имени (CNAME). Для этого создайте новую хранимую процедуру, рис. 68.

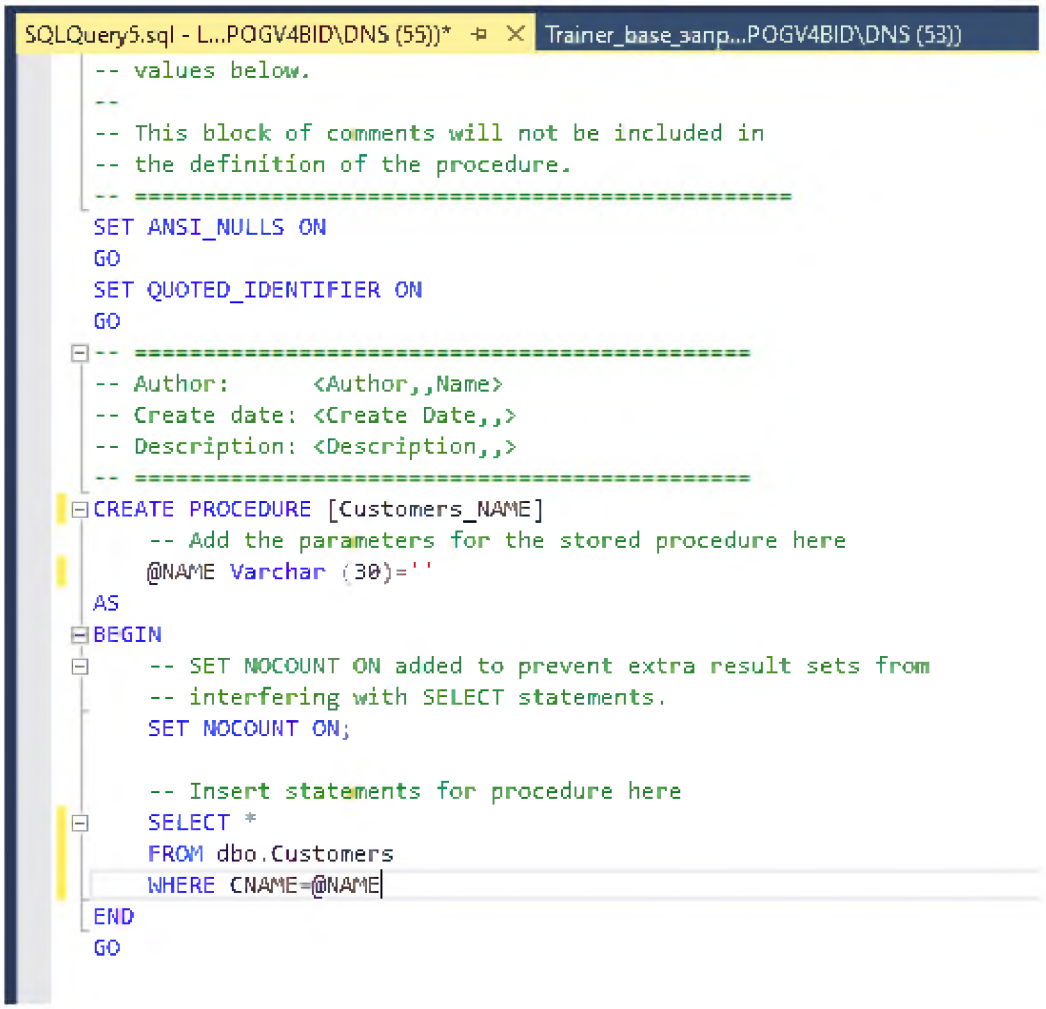


Рис. 68. Текст процедуры для отбора покупателей по Фамилии из таблицы Покупатели

Где,

CREATE PROCRDURE [Customers_NAME] — определяет имя создаваемой процедуры как;

@NAME Varchar (30) =' ' — определяют единственный параметр процедуры NAME. Параметру можно присвоить текстовые сроки переменной длины длиной до 30 символов (Тип данных Varchar(30)), значения по умолчанию равны пустой строке;

FROM dbo.Customers WHERE CNAME=@NAME — отображает все поля (*) из таблицы Customers, где значение поля CNAME равно значению параметра NAME (CNAME = @NAME).

11. Выполните запрос.

12. Сохраните под именем Customers_NAME.

13. Запустите процедуру. Для этого создайте новый запрос и наберите

14. EXEC [Customers_NAME] ‘Clements’.

15. Создайте новую хранимую процедуру отображающая продавцов, у которых вознаграждение выше заданного.

Где,

CREATE PROCRDURE [Salespeople_COMM] — определяет имя создаваемой процедуры;

@Grade Real = 0 — определяет параметр процедуры. Параметру можно присваивать дробные числа (тип данных Real), значение по умолчанию равно 0;

SELECT * FROM dbo.[Salespeople] WHERE [COMM] > @Grade — отображает все поля (*) из таблицы dbo.[Salespeople], где комиссионные больше, чем значение параметра Grade.

16. Выполните и сохраните процедуру Salespeople_COMM.

17. Проверьте работоспособность, создав новый запрос (EXEC [Salespeople_COMM] 0.12).

Создание хранимой процедуры с помощью запроса

Для создания хранимой процедуры применяется команда CREATE PROCEDURE или CREATE PROC.

18. Создайте процедуру с помощью запроса. Для этого создайте новый запрос (рис. 69).

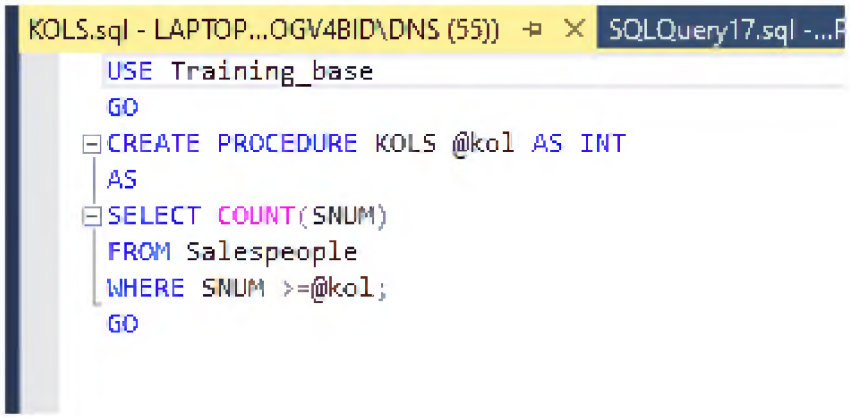


Рис. 69. Создание процедуры.

Поскольку команда CREATE PROCEDURE должна вызываться в отдельном пакете, то после команды USE, которая устанавливает текущую базу данных, используется команда GO для определения нового пакета. Для отделения тела процедуры от остальной части скрипта код процедуры нередко помещается в блок BEGIN...END.

19. Выполните запрос.

20. Сохраните процедуру.

21. Проверьте ее работоспособность. EXEC KOLS 1002;

22. Объясните ее действия.

После добавления процедуры ее можно увидеть в узле базы данных в SQL Server Management Studio в подузле Программирование -> Хранимые процедуры.

Для удаления процедуры применяется команда DROP PROCEDURE.

Для модификации структуры хранимых процедур используется инструкция ALTER PROCEDURE (или ALTER PROC). Инструкция ALTER PROCEDURE обычно применяется для изменения инструкций Transact-SQL внутри процедуры. Все параметры инструкции ALTER PROCEDURE имеют

такое же значение, как и одноименные параметры инструкции CREATE PROCEDURE. Процедуры могут принимать параметры. Параметры бывают входными - с их помощью в процедуру можно передать некоторые значения. И также параметры бывают выходными - они позволяют возвратить из процедуры некоторое значение.

Триггеры. Определение триггеров

Триггеры представляют специальный тип хранимой процедуры, которая вызывается автоматически при выполнении определенного действия над таблицей или представлением, в частности, при добавлении, изменении или удалении данных, то есть при выполнении команд INSERT, UPDATE, DELETE.

Формальное определение триггера:

```
CREATE TRIGGER имя_триггера
ON {имя_таблицы | имя_представления}
{AFTER | INSTEAD OF} [INSERT | UPDATE | DELETE]
AS выражения_sql
```

Для создания триггера применяется выражение CREATE TRIGGER, после которого идет имя триггера. Как правило, имя триггера отражает тип операций и имя таблицы, над которой производится операция.

Каждый триггер ассоциируется с определенной таблицей или представлением, имя которых указывается после слова ON.

Затем устанавливается тип триггера. Мы можем использовать один из двух типов:

AFTER: выполняется после выполнения действия. Определяется только для таблиц.

INSTEAD OF: выполняется вместо действия (то есть, по сути, действие - добавление, изменение или удаление - вообще не выполняется). Определяется для таблиц и представлений

После типа триггера идет указание операции, для которой определяется триггер: INSERT, UPDATE или DELETE.

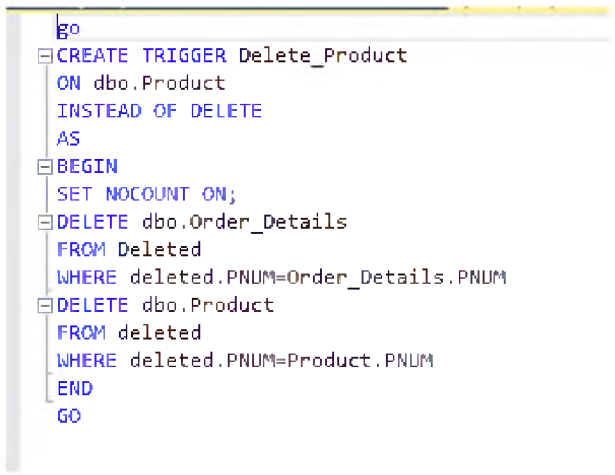
Для триггера AFTER можно применять сразу для нескольких действий, например, UPDATE и INSERT. В этом случае операции указываются через запятую. Для триггера INSTEAD OF можно определить только одно действие.

И затем после слова AS идет набор выражений SQL, которые, собственно, и составляют тело триггера.

Триггеры DML в ходе работы имеют доступ к специальным таблицам inserted и deleted, хранящим результаты изменений, произведенных оператором, вызвавшим триггер.

23. Создайте запрос на удаление записи из таблицы Product (будет ошибка, есть связанные записи в таблице Order_Details).

24. Создайте триггер Delete_Product на удаление записи из таблицы Product, рис 69.1.



```
go
CREATE TRIGGER Delete_Product
ON dbo.Product
INSTEAD OF DELETE
AS
BEGIN
SET NOCOUNT ON;
DELETE dbo.Order_Details
FROM Deleted
WHERE deleted.PNUM=Order_Details.PNUM
DELETE dbo.Product
FROM deleted
WHERE deleted.PNUM=Product.PNUM
END
GO
```

Рис. 69.1. Создание триггера `Delete_Product`

25. Убедитесь, что триггер Delete_Product создан.

26. Повторите выполнение запроса на удаление записи из таблицы Product.

Функции

Пользовательские функции очень похожи на хранимые процедуры. Так же в них можно передавать параметры, и они выполняют некоторые действия, однако их главным отличием от хранимых процедур является то, что они

выводят (возвращают) какой-то результат. Более того, они вызываются только при помощи оператора SELECT, аналогично встроенным функциям. Все пользовательские функции делятся на 2 вида:

1) Скалярные функции — функции, которые возвращают число или текст, то есть одно или несколько значений;

2) Табличные функции — функции, которые выводят результат виде таблицы.

Команда CREATE FUNCTION создаёт определяемую пользователем функцию в SQL Server.

27. Создайте скалярную пользовательскую функцию. Для этого в обозревателе объектов в БД “Training_base” в папке “Programmability” («Программирование»), щёлкните правой кнопкой мыши по папке “Functions” («Функции») и в появившемся меню выберите пункт “New / Scalar-valued Function” («Создать / Скалярная функция»). Появится окно новой скалярной пользовательской функции.

28. Внесите изменения, как на рис. 70.

29. Выполните функцию.

30. Сохраните.

31. Проверьте работоспособность, для этого создайте новый запрос:

32. SELECT dbo.Calculator(4,5, '+'), dbo.Calculator(3,7, '*'), dbo.Calculator(64,4,'/')*2;

33. Выполните запрос.

34. Измените его, чтобы в результате отображались наименование столбцов: сумма; произведение; пример.

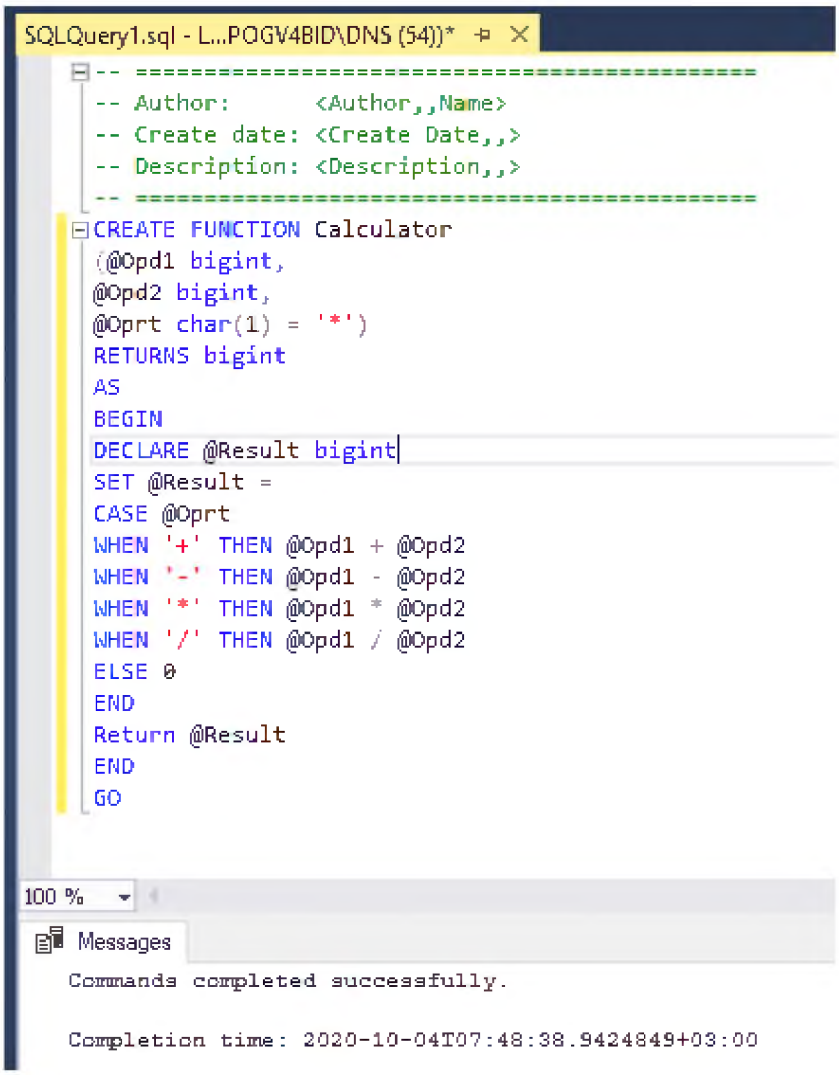


Рис. 70. Текст пользовательской функции «Calculator»

После выполнения лабораторной работы 6 в отчете создать раздел Лабораторная работа 6. В нем должна содержаться следующая информация:

— название лабораторной работы;

— цель работы;

— задание;

— вывод по работе.

Обязательно должны быть скриншоты (к лр 6):

— создание и выполнение хранимой процедуры Среднее трех чисел (должно быть видно имя сервера и полное имя БД);

— создание и выполнение хранимой процедуры Customers_NAME;

— создание и выполнение хранимой процедуры Salespeople_COMM;

— создание и выполнение хранимой процедуры KOLS (включая объяснения);

— создание и выполнение триггера Delete_Product;

— создание и выполнение скалярнй функции Calculator (должно быть видно имя сервера и полное имя БД);

— результат создания объектов в окне обозревателя

Соблюдать требования предъявляемые к оформлению отчета.

Лабораторная работа 7

ИСПОЛЬЗОВАНИЕ КОМАНДНОЙ СТРОКИ SQL SERVER

Microsoft SQL Server предоставляет пользователям разнообразные графические пользовательские интерфейсы для извлечения и обработки данных, а также для настройки баз данных SQL Server. Однако иногда проще работать из командной строки.

Для доступа к MSSQL из командной строки у Microsoft есть утилита sqlcmd

1. Запустите командную строку Windows – cmd, рис. 70.1.

(Чтобы запустить SQLCMD нажмите комбинацию клавиш Windows + R, чтобы открыть окно выполнения, введите cmd, чтобы запустить приложение Command).

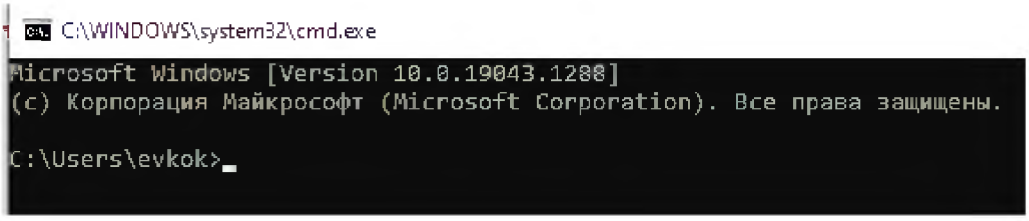


Рис. 70.1. Командная строка Windows

2. Подключитесь к именованному экземпляру SQL Server. Для этого укажите имя сервера и имя экземпляра SQL Server с которым необходимо соединиться.

sqlcmd -S DESKTOP\SQLEXPRESS -E, рис. 70.2

где:

–S указывает на server\instance_name;

–E trusted connection (доверительное соединение).

Чтобы определить имя компьютера с помощью командной строки необходимо в открывшемся окне командной строки выполнить команду «hostname», после чего операционная система выведет текущее название системы, рис.7.3.

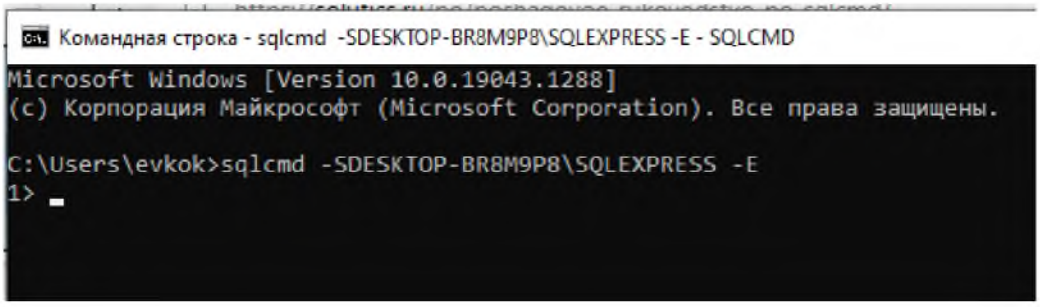


Рис. 70.2. Подключение к серверу

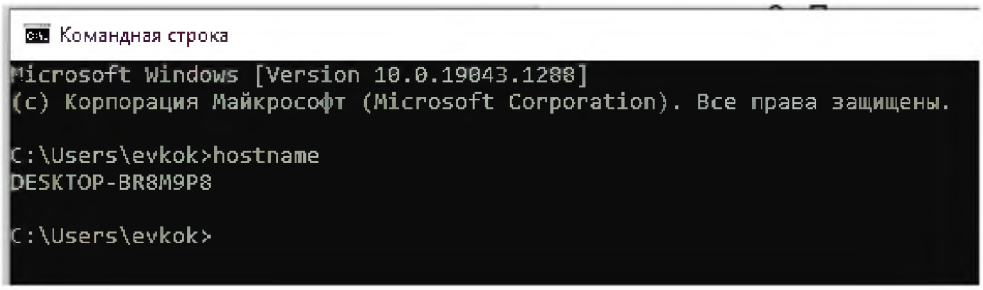


Рис. 70.3. Вывод имени компьютера

Отображаемая в командной строке цифра 1> означает, что подключение состоялось и есть готовность принимать запросы для исполнения.

Если ранее при создании пользователя SQL Server для пользователя была включена аутентификацию SQL Server, то при подключении требуется указать имя пользователя и ввести его пароль.

Чтобы это выполнить, предварительно необходимо выйти из sqlcmd и заново войти с указанием учетных данных:

```
sqlcmd -S DESKTOP\SQLEXPRESS -U DB_User
```

Командная строка запросит пароль для пользователя DB_User, который необходимо ввести с клавиатуры.

При желании можно указать пароль в командной строке, хотя это не рекомендуется по соображениям безопасности:

```
sqlcmd -S DESKTOP\SQLEXPRESS -U DB_User -P 12345678
```

3. Выведите список баз данных в экземпляре SQL, для этого наберите

```
select name from sys.databases go,
```

рис. 70.4.

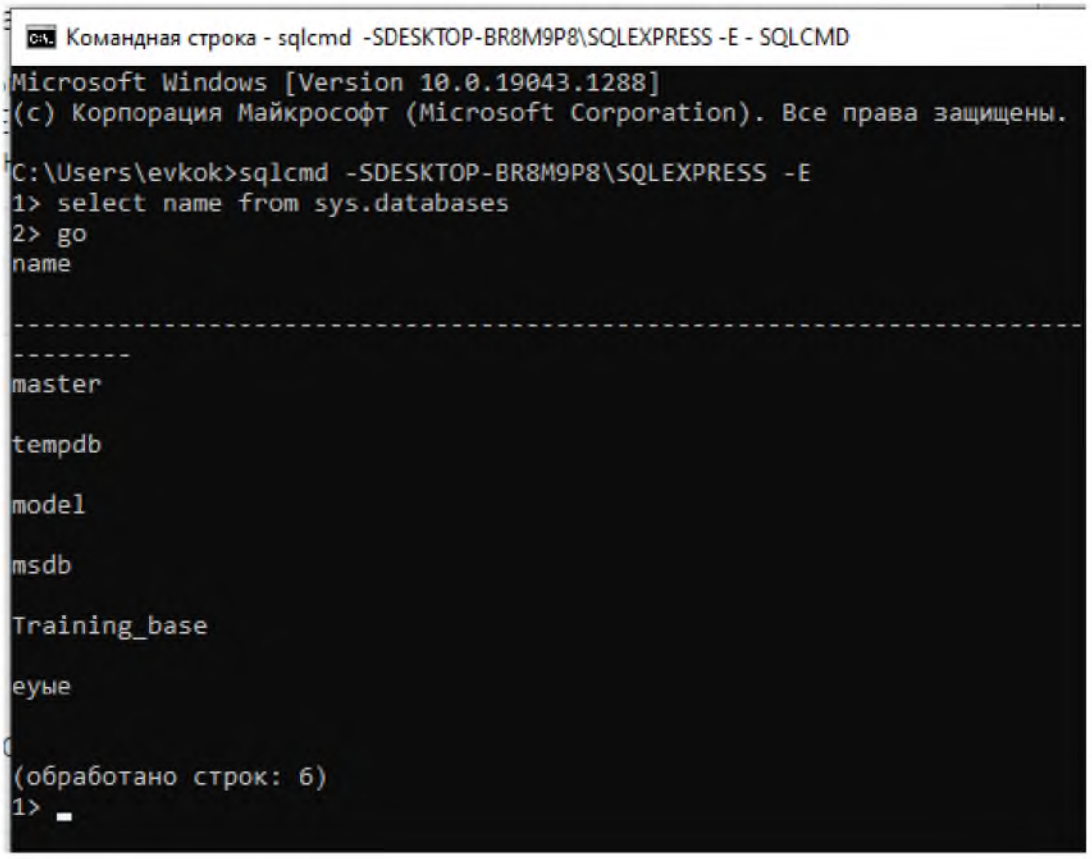


Рис. 70.4. Вывод списка бд на сервере

4. Подключитесь к базе данных Training_base и выведи все записи из таблицы Product, рис. 70.5.

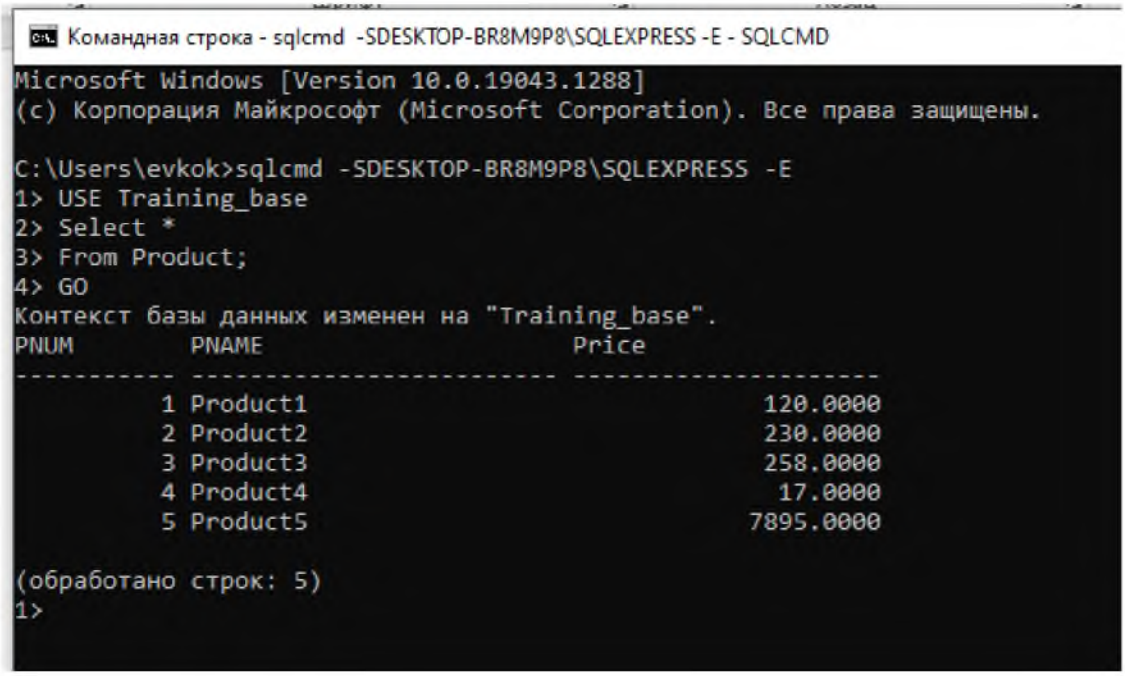


Рис. 70.5. Запрос на выборку всех записей из таблицы Produc

В тех случаях, когда необходимо выполнить преобразования от типов с высшим приоритетом к типам с низшим приоритетом, то надо выполнять явное приведение типов. Для этого в SQL определены две функции: `CONVERT` и `CAST`.

Функция `CAST` преобразует выражение одного типа к другому. Она имеет следующую форму:

```
CAST(выражение AS тип_данных)
```

Большую часть преобразований охватывает функция `CAST`. Если же необходимо какое-то дополнительное форматирование, то можно использовать функцию `CONVERT`. Она имеет следующую форму:

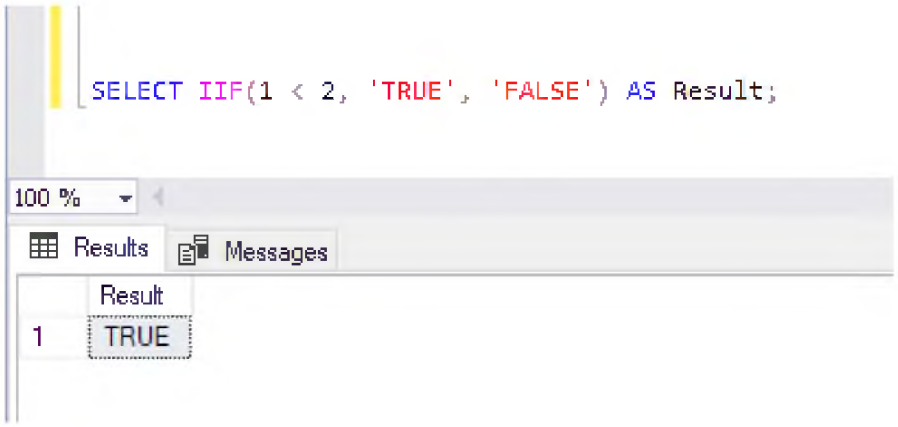
```
CONVERT(тип_данных, выражение [, стиль])
```

Третий необязательный параметр задает стиль форматирования данных. Этот параметр представляет числовое значение, которое для разных типов данных имеет разную интерпретацию.

В SQL реализована возможность заносить в поле значение в зависимости от условия. Для этого используется функция IIF:

```
IIF(логическое_выражение, выражение_1, выражение_2)
```

Функция вычисляет логическое_выражение, если оно истина – в поле заносится значение выражения_1, в противном случае – значение выражения_2. Все три параметра IIF() являются обязательными.



Основные принципы управления учетными записями и ролями в MS SQL Server

1. Создайте новую учетную запись для базы данных `University (!!!)`. Для этого выберите в Обозревателе объектов раздел Безопасность (Security)/Имена входа (Logins). Добавьте новое имя входа (рис. 71) – Proba, установите опцию Проверка подлинности SQL Server/SQL Server authentication, присвойте свой **пароль**, примените к **выбранной базе данных**, установите язык по умолчанию – **русский** (рис. 72).

2. В этом же диалоговом окне, перейдите на вкладку Роли сервера/Server Roles. Установите для пользователя Proba роль sysadmin (рис. 73, 74).

3. Далее в разделе Сопоставление пользователя/User mapping, установите для базы данных University у пользователя Proba права доступа Db_owner, означающие, что пользователь может выполнять любые действия с БД. Ниже перечислены все возможные варианты прав доступа.

4. Перейдите на раздел Состояние/Status. Установите опции Разрешение к подключению к ядру СУБД – предоставить и имя входа включить (рис.75).

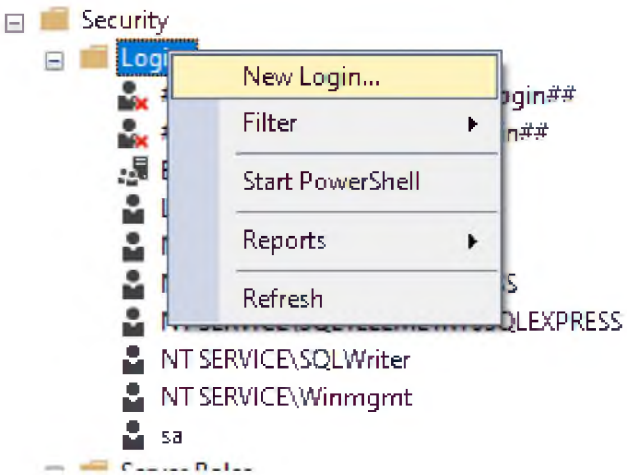


Рис. 71. Добавить новое имя

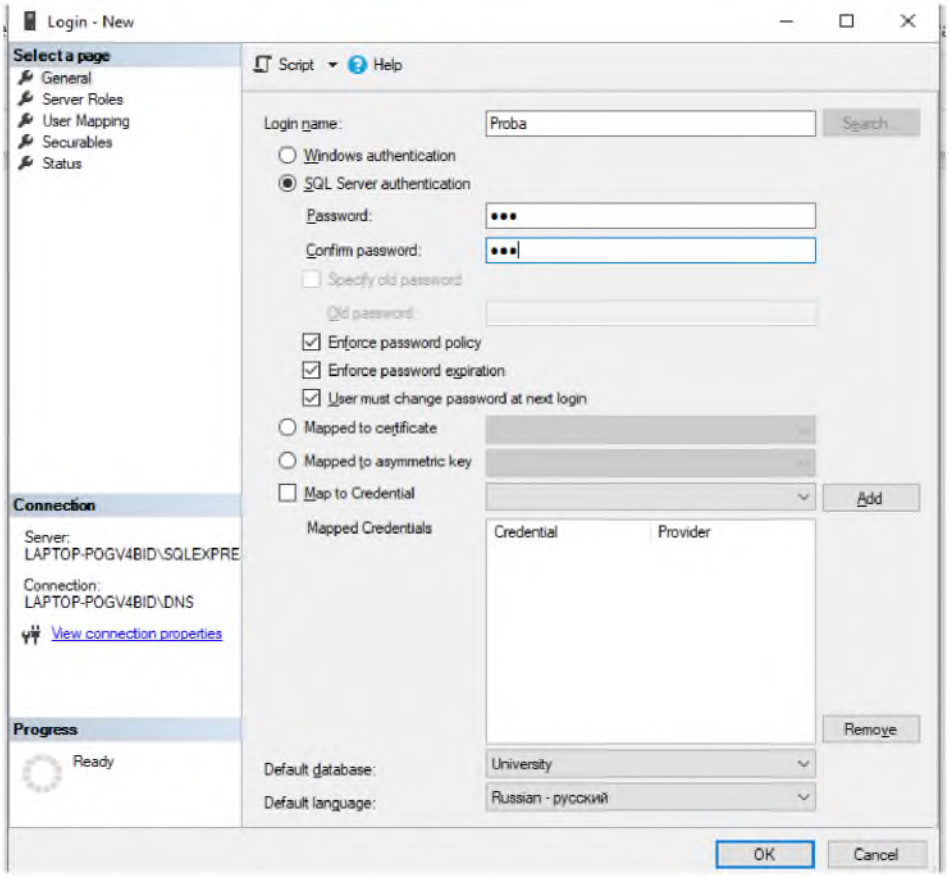


Рис. 72. Установка параметров

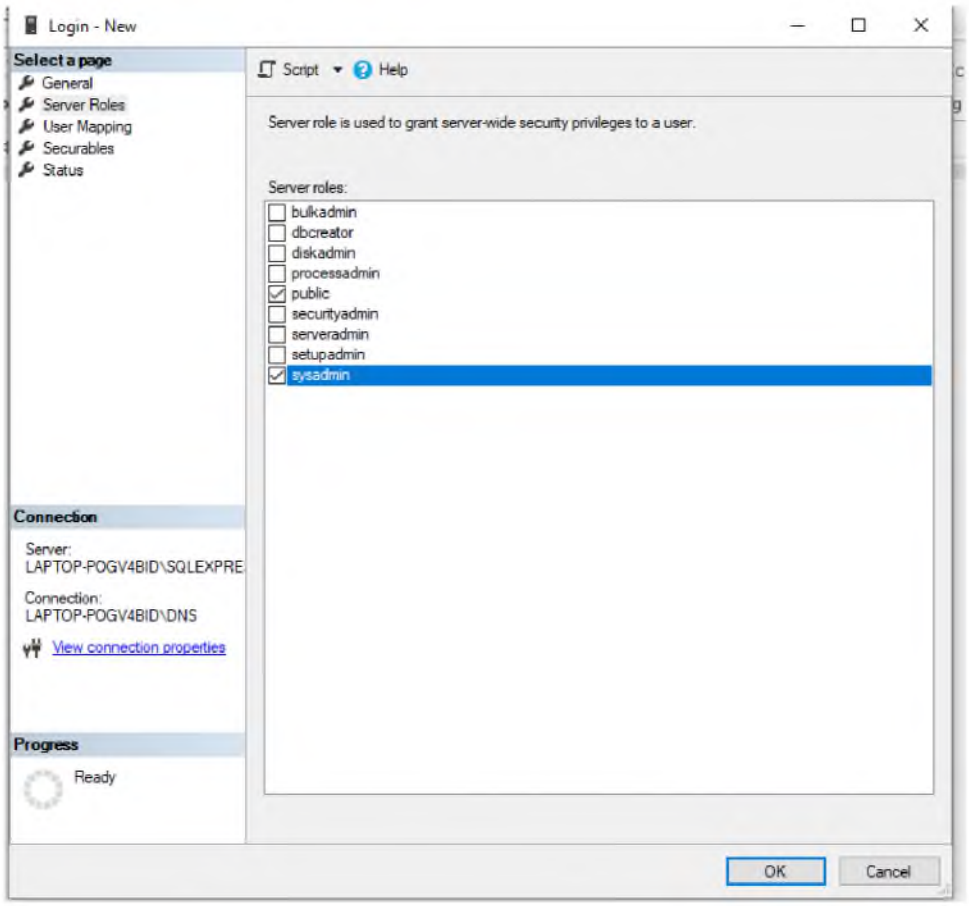


Рис. 73. Настройка серверной роли для нового пользователя

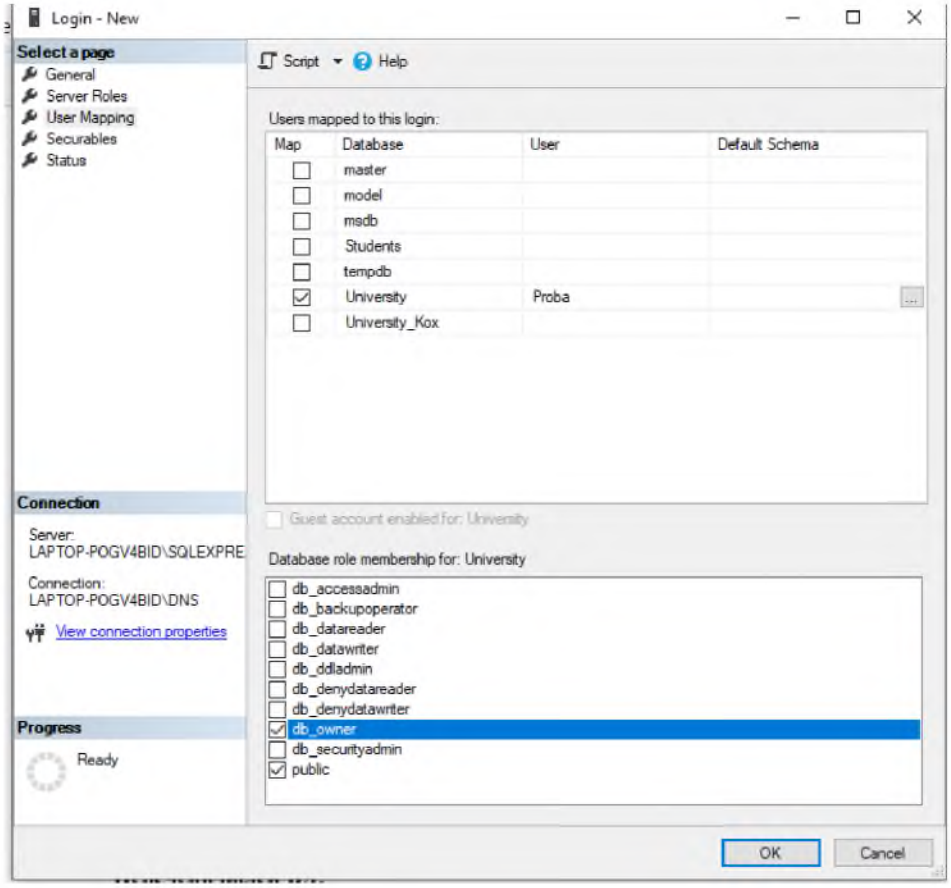


Рис. 74. Настройка роли базы данных для нового пользователя

Перечень ролей БД:

Public – минимальные права доступа к БД (на просмотр);

Db_owner – может выполнять любые действия с БД;

Db_accessadmin – добавляет и удаляет пользователей БД;

Db_sequrityadmin – управляет ролями в БД и разрешениями на запуск команд и работу с объектами БД;

Db_ddladmin – добавляет, изменяет и удаляет объекты БД;

Db_backupoperator – осуществляет резервное копирования БД;

Db_dataSTUDENT – может просматривать все данные в каждой таблице в БД;

Db_datawriter - может добавлять, удалять и изменять данные в каждой таблице в БД;

Db_denydataSTUDENT – запрет на просмотр всех данных в каждой таблице в БД;

Db_denydatawriter - запрет на добавление, удаление и изменение всех данных в каждой таблице в БД;

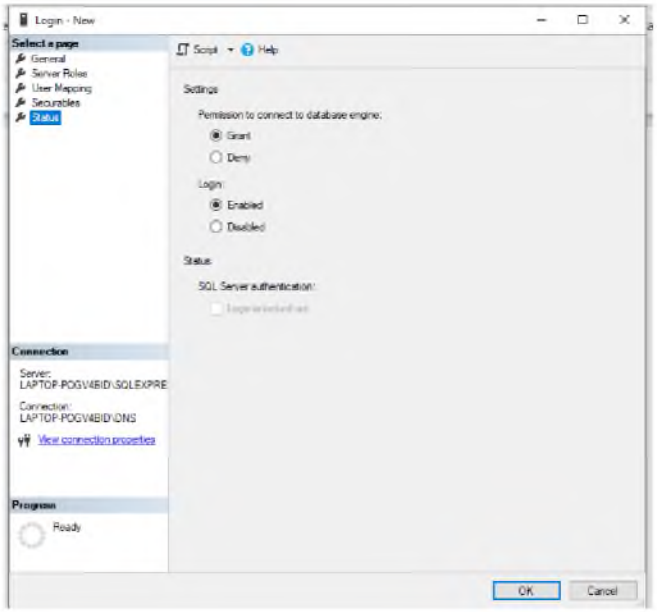


Рис. 75. Разрешение к подключению к ядру СУБД

5. После нажатия на <OK> в БД появится пользователь Proba с правами собственника БД, который может выполнять все манипуляции с БД University.

6. Откройте в окне обозревателя объектов БД University и перейдите на вкладку Безопасность, там вы найдите только что созданного пользователя.

Создание ролей программно

Для упрощения управления правами доступа в системе создаются роли, которые затем можно назначать группе пользователей.

7. Создайте для данного примера (БД University) роли декана (DEKAN) и студента (STUDENT).

Пример создания роли декана, рис. 76:

```
USE University --сделать текущей БД University
EXEC sp_addrole 'DEKAN';
```

Рис. 76. Пример создания роли декана

8. Обновите. Откройте вкладку Безопасность/Роли/Роли базы данных и убедитесь, что роль DEKAN создана.

Пример создания роли студента, рис. 77.

```
USE University --сделать текущей БД university
EXEC sp_addrole 'STUDENT'
```

Рис. 77. Пример создания роли декана

Декан должен обладать правами на чтение, удаление, изменение, добавление во все таблицы БД University, а также должен иметь возможность запускать на исполнение процедуры и функции БД University. Поэтому роли декана из системных привилегий назначаем EXECUTE, а из привилегий доступа к объектам назначаем DELETE, INSERT, UPDATE, SELECT. Студент должен обладать правами на чтение из таблиц. Поэтому роли читателя из привилегий доступа к объектам назначаем SELECT.

Оператор представления привилегий

Синтаксис:

```
GRANT <привилегия>, ...
ON < объект >, …
```

```
TO <имя>
[WITH grant option];
```

Атрибут WITH GRANT OPTION дает право пользователю самому раздавать права, которые он получил. С помощью оператора GRANT для каждого пользователя формируется список привилегий, привилегии управляют работой сервера данных с точки зрения защиты данных. Выполнению каждой транзакции предшествует проверка привилегий пользователя, сеанс которого породил транзакцию.

Например, (не выполнять) !!!!!:

GRANT select, update (Sales, num) ON Sales_data TO user1 WITH GRANT OPTION

Пользователь, предоставивший привилегию другому, называется грантор (grantor — предоставитель). Привилегия является предоставляемой, если право на нее можно предоставить другим пользователям. PUBLIC — имя роли, которую получает пользователь при добавлении в список пользователей конкретной БД, включает в себя минимальный набор прав на чтение данных из таблиц и представлений в БД.

9. Для примера создайте таблицу Discuplinu (с помощью запроса) в БД University. выполнив следующий sql-запрос, рис. 78:

```
USE University --сделать текущей БД University
CREATE TABLE Discuplnu (
Kod_Discuplnu INT NOT NULL,
name_Discuplnu NCHAR(30) NULL,
kol_Discuplnu INT NULL
CONSTRAINT PK_Dis PRIMARY KEY (Kod_Discuplnu));
```

Рис. 78. Создание таблицы Discuplinu

10. Выполните код и обновите вкладку Таблицы. Вы должны увидеть созданную таблицу для сохранения данных о всех дисциплинах. Эта таблица пока пустая с тремя столбцами `Kod_Discuplinu`, `name_Discuplinu`, `kol_chasov`.

Роль декана названа DEKAN. Операторы назначения прав доступа для этой роли представлены ниже:

```
GRANT DELETE, INSERT, UPDATE, SELECT ON Discuplinu TO DEKAN
GRANT EXECUTE TO DEKAN
```

Роль студента названа STUDENT. Операторы назначения прав доступа для этой роли представлены ниже:

```
GRANT SELECT ON Discuplinu TO STUDENT
```

Примените роли декана и студента к созданной таблице.

Создание пользователей с определенной ролью

Пример создания декана Ivanov_Dek и присвоения ему роли:

```
EXEC sp_addlogin 'Ivanov_Dek','Ivanov', 'University'
use University
EXEC sp_adduser 'Ivanov_Dek','Ivanov_Dek'
EXEC sp_addrolemember 'DEKAN', 'Ivanov_Dek'
```

Пример создания студент Petrov_Stud и присвоения роли:

```
EXEC sp_addlogin 'Petrov_Stud','Petrov', 'University'
use University
EXEC sp_adduser 'Petrov_Stud','Petrov_Stud'
EXEC sp_addrolemember 'STUDENT', 'Petrov_Stud'
```

Выполните команды. Перейдите в окне Обозреватель объектов на Роли/Роли базы данных/Dekan и просмотрите его свойства. Просмотрите назначенные общие свойства, защищаемые объекты и расширенные свойства.

После выполнения лабораторной работы 7 в отчете создать разделы Лабораторная работа 7. В нем должна содержаться следующая информация:

— название лабораторной работы;

— цель работы;

— задание;

— вывод по работе.

Обязательно должны быть скриншоты (к лр 7):

— подключения к серверу, используя командную строку;

— вывода списка всех бд на сервере используя командную строку;

— запроса на вывод всех записей таблицы Product используя командную строку;

— создание новой учетной запись для базы данных `University;`

— настройки серверной роли для нового пользователя;

— создание ролей декана и студента (программно);

— назначения прав доступа для каждой роли.

Самостоятельная работа 4

1. Создайте хранимую процедуру KOLS_N с помощью запроса. Которая отображает количество продавцов с определенным кодом и именем, начинающимся на определенную букву. Ниже приведена подсказка.

```
@kol AS INT,
@NAME AS VARCHAR (10)

WHERE SNUM >=@kol
AND SNAME LIKE @NAME;
```

2. Создайте триггер, который будет срабатывать при добавлении и обновлении данных (увеличение цены на 5%) в таблицу Product, см. пример лекции.

3. Создайте в базе данных `Training_base` еще две таблицы `Category` (Категории) с полями: `CategoryID, Categoryname, Description` и `Supplier` (Поставщики) с полями: `SupplierID, CompanyName, City,` **использовать командную строку**.

4. Осуществите необходимые связи между таблицами (**использовать командную строку**). При необходимости добавить дополнительно поля, рис. 78.1.

5. Заполните таблицы данными (**использовать командную строку**). Значения задать самостоятельно.

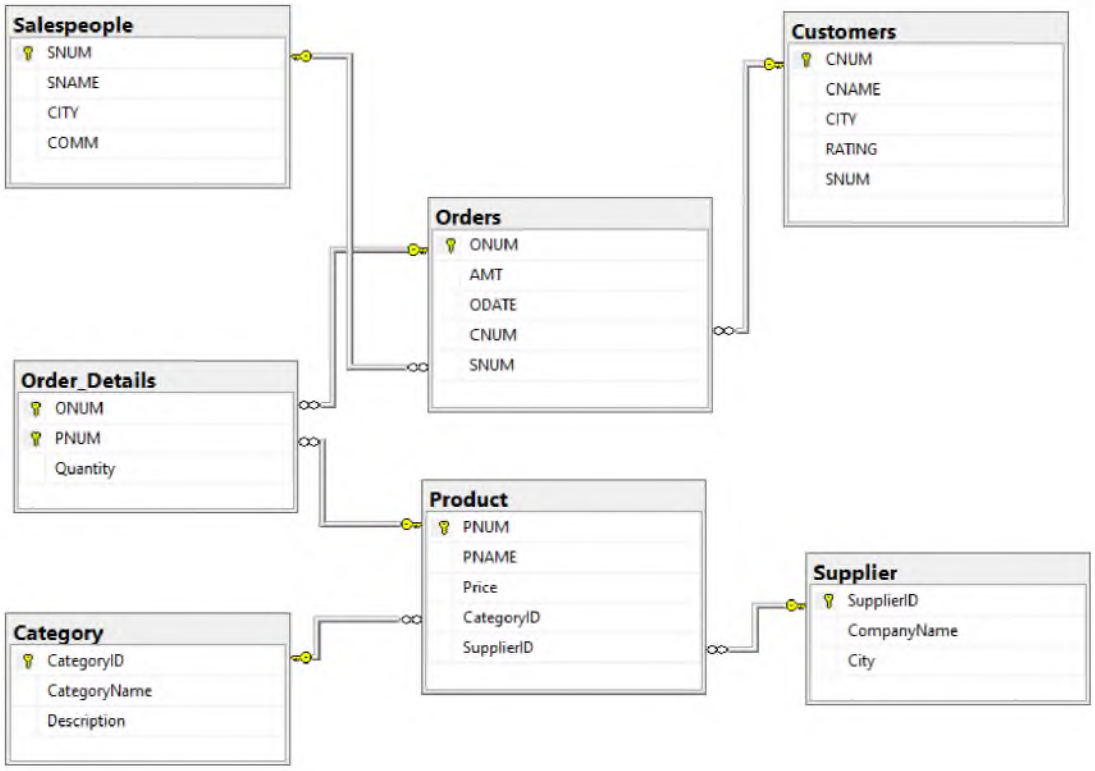


Рис. 78.1. Схема данных

1. Для каждого продукта вычислить налог на добавленную стоимость для полученных сумм (имя столбца НДС), который включен в стоимость и составляет k = 18%, а также стоимость продукта (Стоимость_без_НДС) без него. Все значения вывести с точностью два знака после запятой, отсортировав по Стоимости.

2. Вывести информацию (название и цену) о тех продуктах, цены которых превышают минимальную цену продукта (товара) не более чем на 150 рублей в отсортированном по возрастанию цены виде.

3. Для каждого продукта/товара из таблицы Product установить скидку следующим образом: если цена меньше 200 рублей, то скидка будет составлять 5% от цены, в противном случае 7%. Все значения вывести с точностью два знака после запятой.

После выполнения самостоятельной работы в отчете создать раздел самостоятельная работа 4. В нем должна содержаться следующая информация:

— название лабораторной работы;

— цель работы;

— задание;

— вывод по работе.

Обязательно должны быть скриншоты (к ср 4):

— создание и выполнение хранимой процедуры `KOLS_N`.

— результат создания объектов в окне обозревателя

— создание и выполнение триггера;

— результат создания объектов в окне обозревателя ;

— запроса создания таблиц Category и Supplier;

— запроса осуществления связей и их наполнения;

— сервера, что объекты созданы;

— схемы данных;

— запроса расчета стоимости без НДС;

— запроса на вывод информации, стоимость которых превышает минимум на 150 р;

— запроса расчета скидки.

Соблюдать требования предъявляемые к оформлению отчета.

Лабораторная работа 8

Создание проекта

Visual Studio – это интегрированная среда разработки (IDE) от компании Microsoft., которая позволяет просматривать и изменять практически любой код, а также отлаживать, выполнять сборку и публиковать приложения для устройств с Android, iOS, Windows, а также в Интернете и облаке. Доступны версии как для компьютеров Mac, так и для компьютеров с Windows.

Редакции Visual Studio

Community – бесплатная версия среды разработки Visual Studio. Чтобы ее использовать, необходимо создать учетную запись Visual Studio, в противном случае она будет действовать 30 дней. Редакция имеет меньший функционал по сравнению с платными редакциями, однако она включает все необходимое для создания полноценных приложений. Походит для индивидуальных разработчиков и обучения;

Professional – это мощная полнофункциональная интегрированная среда разработки для программистов, создающих и развертывающих инновационные приложения для любой платформы. Редакция содержит профессиональные инструменты для разработки приложений. Подходит для небольших групп разработчиков;

Enterprise – это мощная, полнофункциональная IDE для разработчиков которые разрабатывают, тестируют и развертывают сложные приложения для любой платформы, включая платформы Майкрософт. Комплексное решение для разработки приложений. Подходит для групп любого размера с высокими требованиями к качеству и масштабу.

Visual Studio Professional и Visual Studio Enterprise являются платными версиями и предназначены для команд профессиональных разработчиков.

Visual Studio Community 2022 (ноябрь 2021) – это бесплатная полнофункциональная интегрированная среда разработки для учащихся, участников проектов с открытым кодом и индивидуальных разработчиков.

Последняя версия Community предлагает большое количество улучшений, которые сильно расширили возможности программного продукта. Особенности программы:

Упрощенная установка.

Функциональные инструменты программирования, разрешающие обнаружить и успешно ликвидировать неточности в коде, проводить рефакторинг.

Модернизированная отладка, во время которой проводится выявление проблем производительности.

Веб-инструменты ASP.NET, Node.js, Python и JavaScript, необходимые для создания веб-приложений.

Несколько поддерживаемых языков программирования, среди которых C#, Visual Basic, F#, JavaScript, C++, TypeScript, Python, возможность включить поддержку новых языков.

C# – объектно-ориентированный язык со строгой типизацией. Разработан в 1998-2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft.NET Framework.

Основным понятием в языке C# является объект. Под объектом понимаются все сущности языка – от констант и переменных базовых типов до агрегированных типов данных любой сложности.

Объектом называется совокупность данных (полей), определяющих состояние объекта, и набор функций (методов), обеспечивающих изменение указанных данных (состояния объекта) и доступ к ним.

Тип любого объекта определяет совокупность его возможных состояний и набор допустимых действий. Наиболее часто понятие тип в языках программирования непосредственно связан с понятием переменной.

Большинство форм создается путем добавления элементов управления на поверхность формы для определения пользовательского интерфейса. Элемент управления — это компонент в форме, используемый для вывода

информации или ввода данных пользователем. Элементы управления на форме называется объектами. Каждый объект обладает своим набором свойств, событий и методов.

Свойства объекта – это его характеристики (высота, ширина и т.д.).

События объекта – это события операционных систем или события, инициируемые пользователем, на которые может реагировать объект (нажатие кнопки).

Методы объекта – действия, которые можно производить с объектом в ходе выполнения программ.

Все формы в клиентском приложении делятся на три группы:

1. Формы для работы с данными – формы, содержащие как объекты управления, так и объекты просмотра данных. Такие формы предназначены для отображения, изменения, удаления и анализа данных.

2. Кнопочные формы – формы, содержащие только объекты управления, предназначаются для открытия всех других форм. Кнопочная форма, которая появляется первой после запуска программы, называется, главной кнопочной формой.

3. Информационные и служебные формы – формы, содержащие только элементы управления, предназначены для отображения служебной информации (справки), несвязанной с таблицами, запросами и фильтрами, либо для выполнения служебных операций, не связанных с данными (Например, форма с калькулятором).

Методы формы

Методы применяются для исполнения тех или иных действий. Методы, являющиеся членами класса, выполняют действия, составляющие функциональность данного класса. Любая форма инкапсулирует базовый набор функций, унаследованный от класса `System.Windows.Forms.Form`, куда входят методы, управляющие отображением формы и доступом к ней в пользовательском окружении. Вот некоторые методы:

Show() – загружает экземпляр класса формы в память, отображает его на экране и передает ему фокус ввода, при этом свойство Visible автоматически устанавливается в true. Если экземпляр формы уже загружен, но пока не виден (например, если его свойство Visible установлено в false), вызов метода Show() даст тот же результат, что и установка свойства Visible в true).

ShowDialog() – выполняет те же действия, что и Show(), но делает окно формы модальным. Это означает, что другим формам приложения не удастся получить фокус, пока не закрыта форма, показанная при помощи метода ShowDialog(). Сделав окно формы модальным, вы заставите пользователя выполнить некоторые действия на этой форме, и только после этого он сможет продолжить работу с приложением.

Activate() – активирует форму, и она получает фокус ввода. Если вызвать этот метод в окне приложения, у которого в текущий момент нет активных форм, окно этого приложения на панели задач начнет мигать. В любом случае активировать разрешено только видимую форму.

Hide() – делает форму невидимой. Форма остается в памяти, но остается невидимой, пока не будет вызван метод Show() или свойство Visible этой формы не будет установлено в true. Метод Hide() устанавливает свойство Visible в false.

Close() – закрывает форму. Этот метод закрывает все удерживаемые формой ресурсы. После вызова метода Close() сделать форму видимой, вызвав метод Show(), не удастся, поскольку ресурсы формы уже освобождены. Вызов Close() на стартовой форме приложения завершает приложение.

Для вызова любого из этих методов необходима ссылка на форму, т. е в памяти должен быть заранее созданный экземпляр формы. При запуске приложения автоматически создает экземпляр стартовой формы в дополнение к тем экземплярам форм, которые создаются в код

Существует два вида дизайна форм:

1. Ленточные формы - формы, выводящие информацию по одной записи.

2. Табличные формы – формы, выводящие информацию в виде таблицы.

Объекты связи используются только в клиентском интерфейсе. В web-интерфейса функции объекта связи выполняет сервер.

Создание пользовательского интерфейса в среде Visual Studio

1. Запустите программу. Выберите - Создать проект. В окне Выбор проекта выберите **Приложение Windows Forms (NET Framework)**, язык программирования C#, присвойте проекту имя – Project_Training_base (рис. 79).

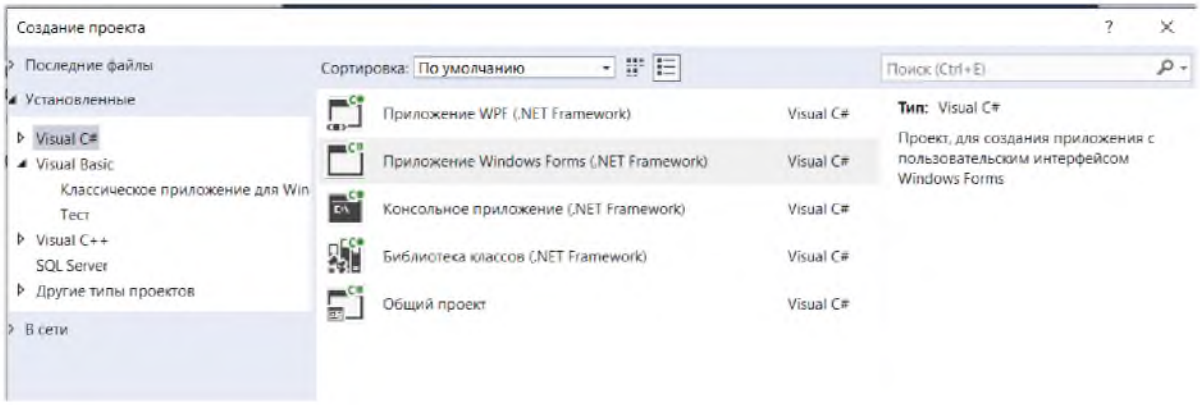


Рис. 79. Создание проекта

1. Подключите к проекту созданную ранее в Microsoft SQL Server БД Training_base. Для этого выберите в меню Проект - Добавить новый источник данных (рис. 80).

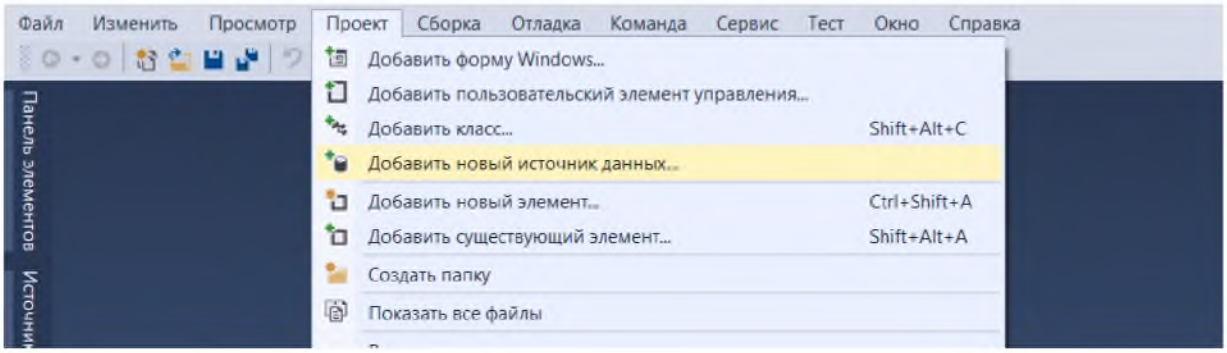


Рис. 80. Добавление нового источника данных

2. В окне Выбор типа источника данных (Choose a Data Source Type) можно выбрать один из трёх видов источников данных: – База данных (Database); – Служба (Service); – Объект (Object). Так как мы подключаем наш проект к БД, то выбираем вариант База данных (Database) (рис. 81) и нажимаем кнопку Далее. Появится окно Выбора подключения БД (Choose Your Data Connection), выберите Набор данных (рис.82).

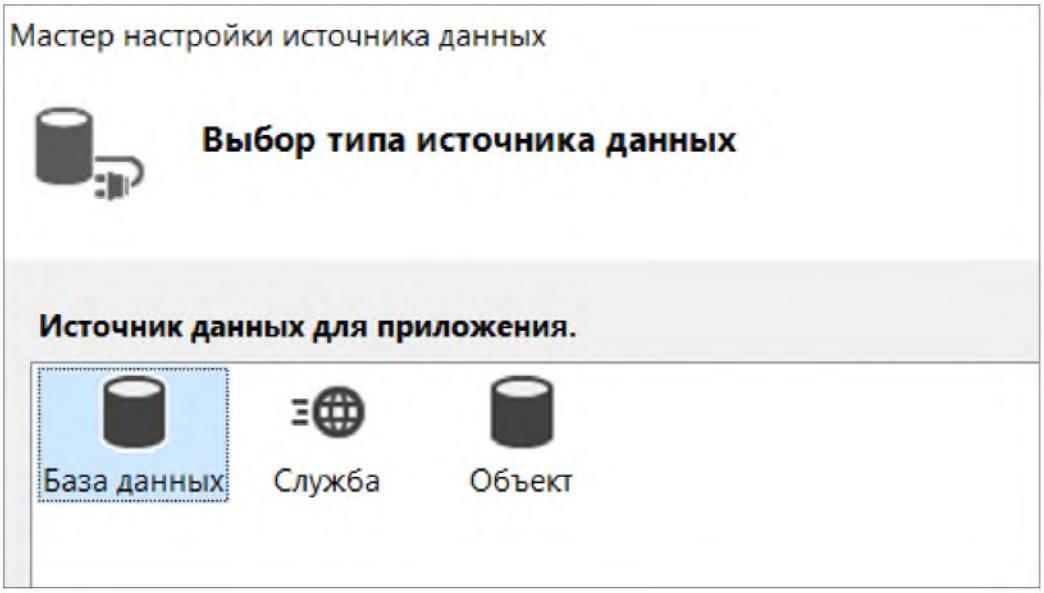


Рис. 81. Выбор типа источника данных

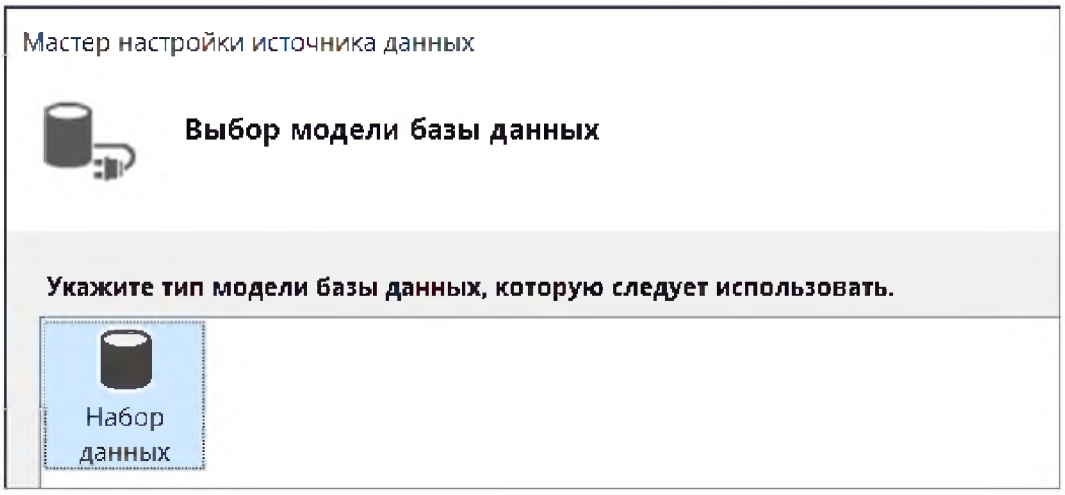


Рис. 82. Выбор типа модели БД

3. В окне выбора подключения к БД для создания нового подключения нажмите кнопку Создать подключение (New Connection). Появится окно добавления нового источника данных (рис. 83).

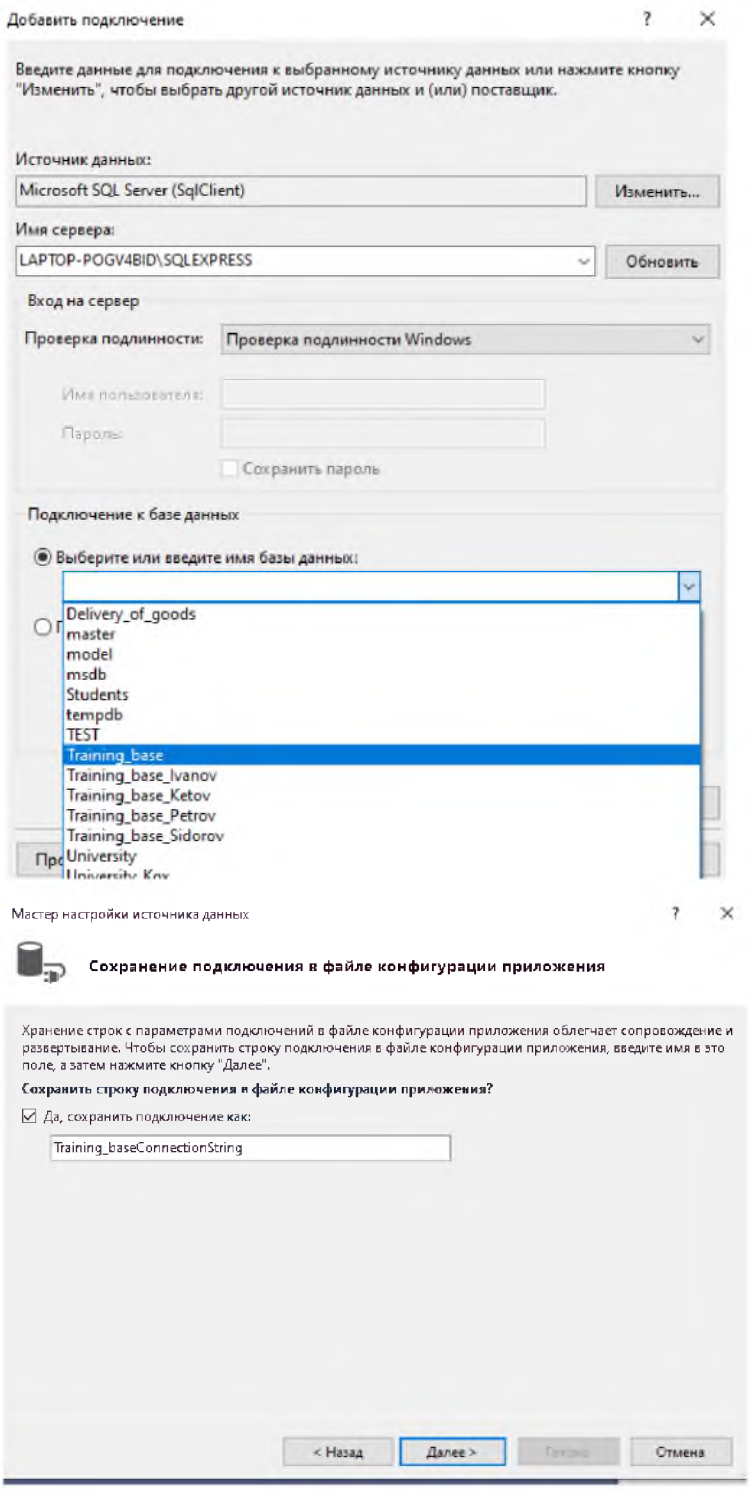


Рис. 83.Добавления нового источника данных

4. Выберите базу данных Microsoft SQL Server и нажмите «Продолжить».

5. Для проверки работоспособности создаваемого соединения нажмите кнопку Проверить подключение. Появится сообщение Проверка подключения выполнена, говорящее о том, что соединение работоспособно.

6. Появится окно с запросом о сохранении строки подключения в файле конфигурации приложения. Для сохранения строки подключения включите опцию Да, сохранить подключение как и нажмите кнопку Далее.

7. Появится окно выбора объектов подключаемой БД (рис. 84). Выберите все объекты к и нажмите кнопку Готово. Подключение завершено.

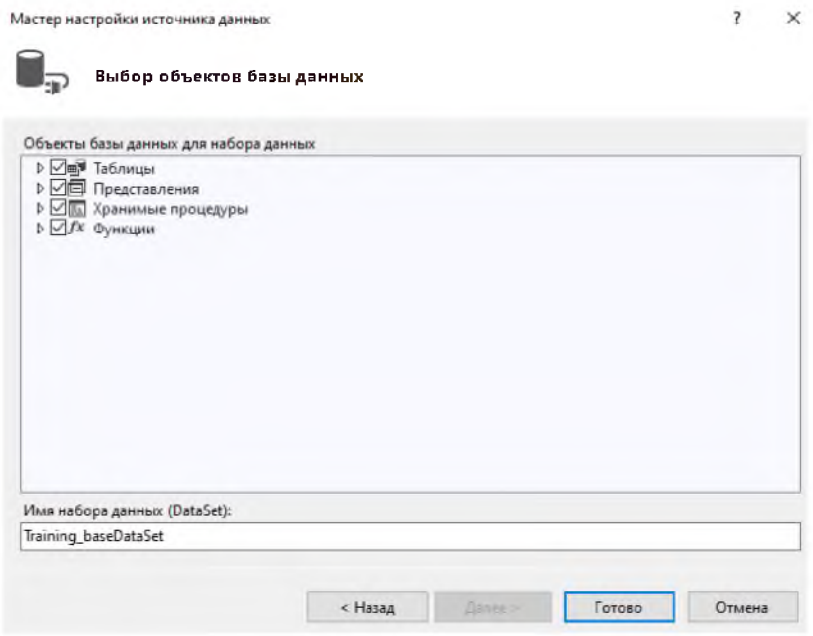


Рис. 84. Выбор объектов базы данных

8. После того, как был добавлен новый источник данных, в обозревателе решений должны появиться файлы (рис. 85).

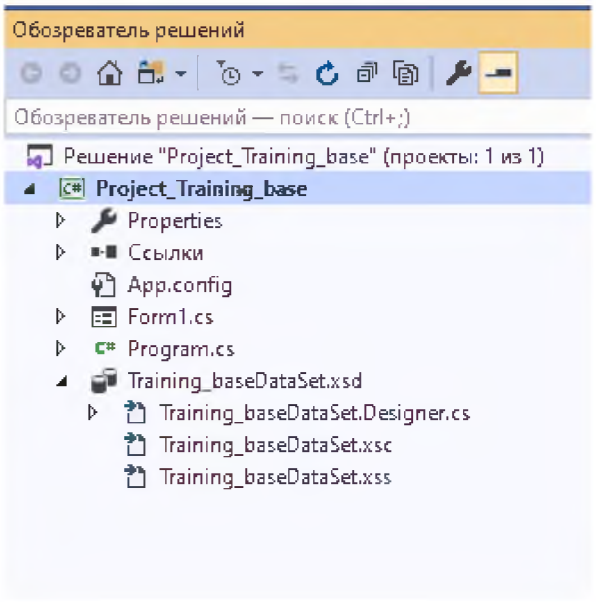


Рис. 85. Окно обозревателя решений после добавления источника данных

9. Откройте Обозреватель серверов (рис. 86). В нем отображены все объекты данной базы данных.

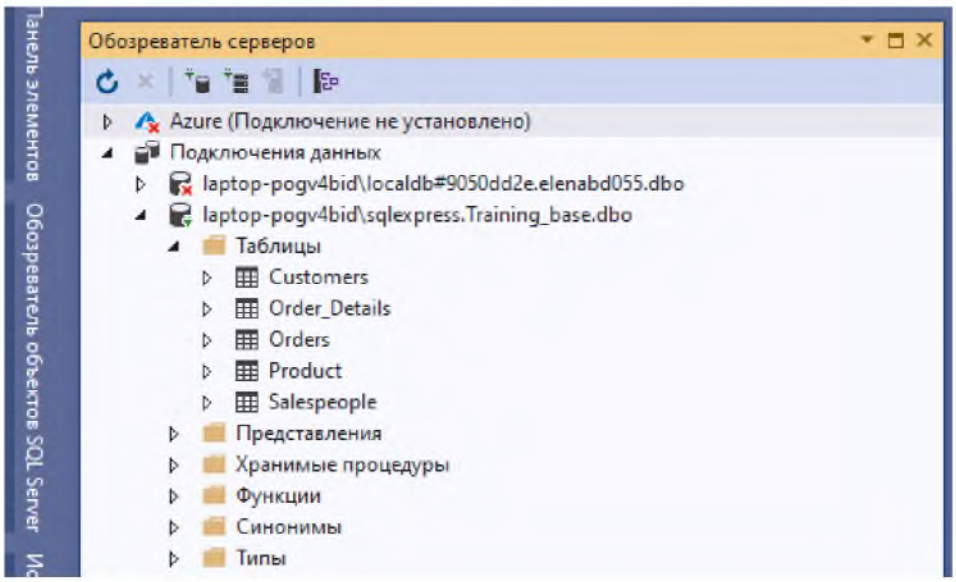


Рис. 86. Обозреватель серверов

Главная кнопочная форма

1. Откройте форму Form1.

2. Используя Панель элементов, поместите на форму надпись (Label) и пять кнопок (Button) как показано на рис. 87.

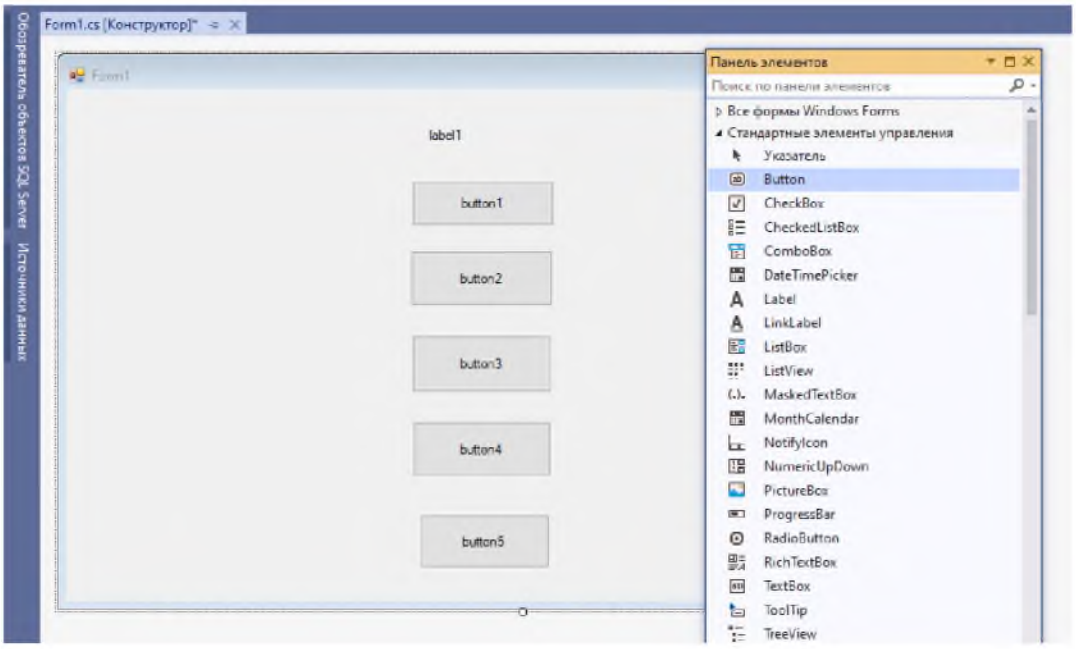


Рис. 87. Добавление элементов на главную форму

3. Для каждого элемента установите следующие свойства:

Для формы (выделите ее)

– FormBorderStyle (Стиль границы формы): Fixed3D (рис. 88);
– MaximizeBox (Кнопка развёртывания формы во весь экран): False;
– MinimizeBox (Кнопка свёртывания формы на панель задач): False;
– Text (Текст надписи в заголовке формы): Учебная база.

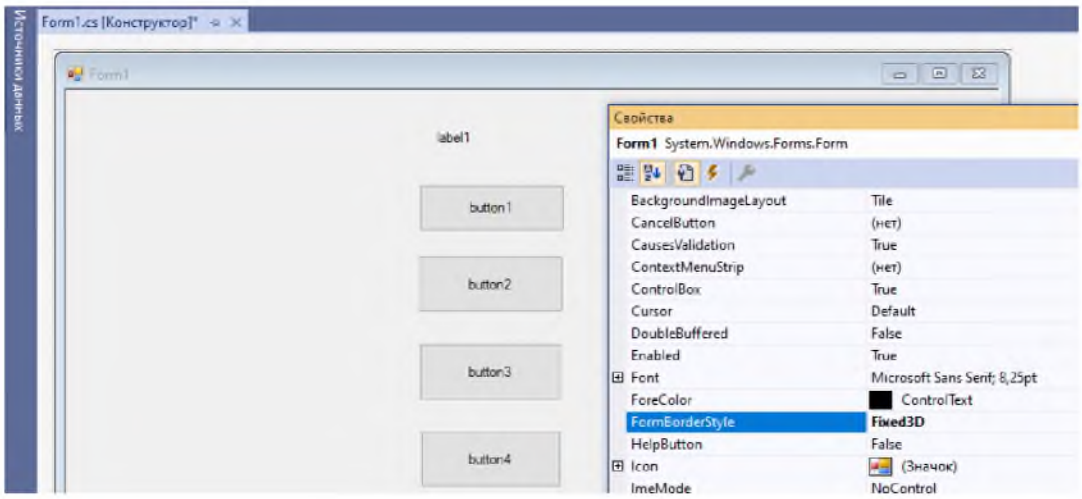


Рис. 88. Параметры свойств формы

Для надписей label, задайте свойства надписи следующим образом:

– AutoSize (Авторазмер): False;
– Font (Шрифт): Microsoft Sans Serif, размер 14;
– ForeColor (Цвет текста): Тёмно синий (0; 0; 192);
– Text (Текст надписи): Учебная база данных
– TextAlign (Выравнивание текста): MiddleCenter.

У кнопок задайте надписи (свойство «Text»), как показано на рис. 89.

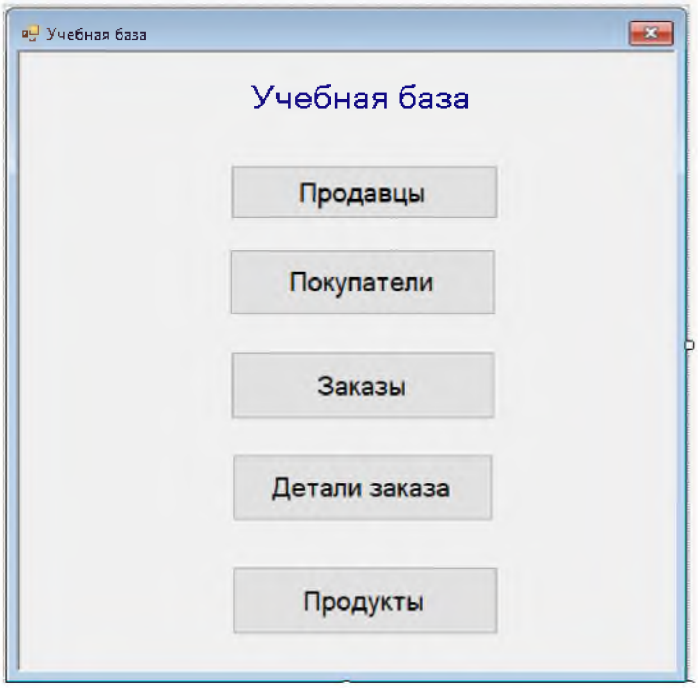


Рис. 89. Результат изменения свойств элементов

4. Добавьте в проект, новую пустую форму. Для этого в оконном меню Проект выберите пункт Добавить новый элемент. Выберите Форма Windows Form (рис. 90). Присвойте имя Продавцы.

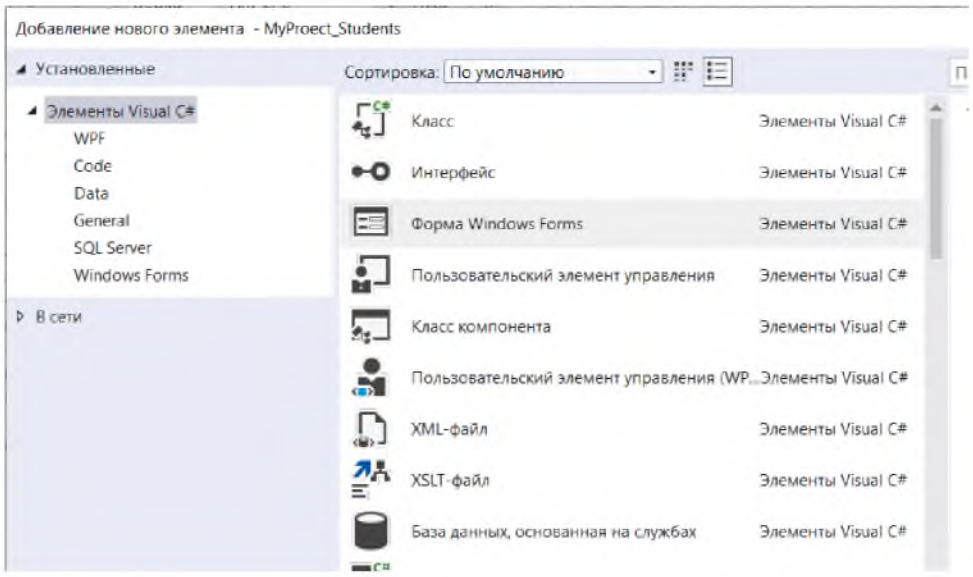


Рис. 90. Добавление в проект новой формы

5. Для формы Продавцы задайте следующие свойства:

– FormBorderStyle (Стиль границы формы): Fixed3D;

– MaximizeBox (Кнопка развёртывания формы во весь экран): False;

– MinimizeBox (Кнопка свёртывания формы на панель задач): False;

– Text (Текст надписи в заголовке формы): Продавцы;

6. На форму добавьте надпись label1, задайте следующие свойства:

– AutoSize (Авторазмер): False;

– Font (Шрифт): Microsoft Sans Serif, размер 16;

– ForeColor (Цвет текста): Тёмно синий (0; 0; 192);

– Text (Текст надписи): Продавцы;

– TextAlign (Выравнивание текста): MiddleCenter.

7. На форму поместите поля таблицы Salespeople (Продавцы). Для этого откройте панель Источники данных/Data Sources (меню Вид - другие окна - Источники данных). На панели «Источники данных» отобразите поля таблицы Salespeople, щёлкнув по значку «+», расположенному слева от имени таблицы. Под полями таблицы в виде подтаблицы располагается таблица Orders Подтаблица показывает, что таблица Orders является вторичной по отношению к таблице Salespeople (рис. 91).

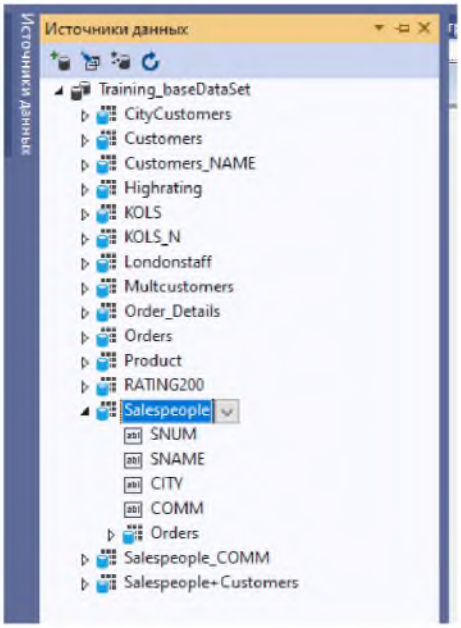


Рис. 91. Панель Источники данных

8. Для того чтобы поместить на новую форму поля таблицы их необходимо перетащить из панели «Источники данных» на форму. Из таблицы Salespeople перетащите мышью на форму все поля, кроме SNUM (Код продавца), рис. 92.

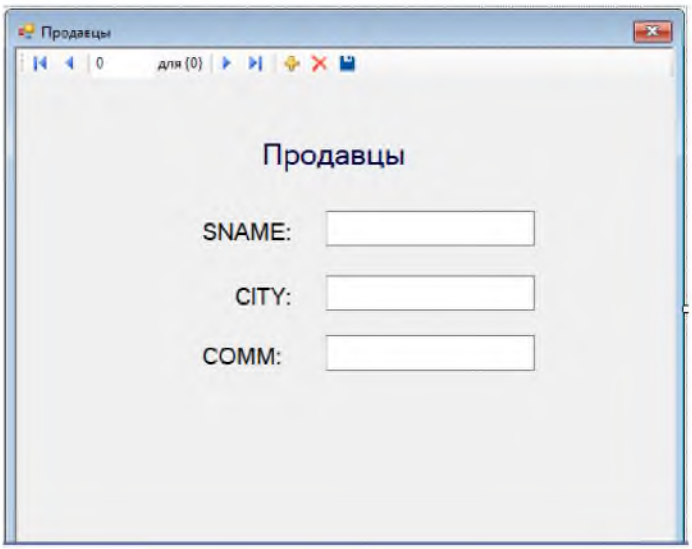


Рис. 92. Форма Продавцы

9. После перетаскивания полей с панели «Источники данных» на форму в верхней части формы появилась навигационная панель, а в нижней части рабочей области среды разработки появились пять невидимых объектов. Эти объекты предназначены для связи нашей формы с таблицей Salespeople расположенной на сервере (рис. 93).

Рис. 93. Объекты связи формы с БД

Рассмотрим функции этих объектов:

training_baseDataSet (Набор данных training_base) – обеспечивает подключение формы к конкретной БД на сервере (в нашем случае это БД Training_base);

salespeopleBindingSource (Источник связи для таблицы «Salespeople») – обеспечивает подключение к конкретной таблице, а также позволяет управлять таблицей;

salespeopleTableAdapter (Адаптер таблиц для таблицы «Salespeople») – обеспечивает передачу данных с формы в таблицу и наоборот.

TableAdapterManager (Менеджер адаптера таблиц) – управляет работой объекта salespeopleTableAdapter;

salespeopleBindingNavigator (Панель управления таблицей) –голубая панель с кнопками управления таблицей, расположенная в верхней части формы.

10. Что бы проверить работоспособность новой формы, её необходимо подключить к главной кнопочной форме, а затем запустить проект и открыть форму Продавцы при помощи кнопки на главной кнопочной форме.

11. Отобразите главную кнопочную форму в рабочей области среды разработки, щёлкнув по вкладке «Form1.cs [Конструктор]» в верхней части рабочей области. Для подключения новой формы Продавцы к главной кнопочной форме дважды щёлкните ЛКМ по кнопке «Продавцы», расположенной на главной кнопочной форме. В появившемся окне кода формы в процедуре «button1_Click» наберите код, предназначенный для открытия формы Продавцы, как это показано на рис. 94.

12. Запустите проект и проверьте подключение формы Продавцы к Главной форме (рис 95). Для запуска проекта можно выбрать команду Отладка – Начать отладку/Запуск без отладки.

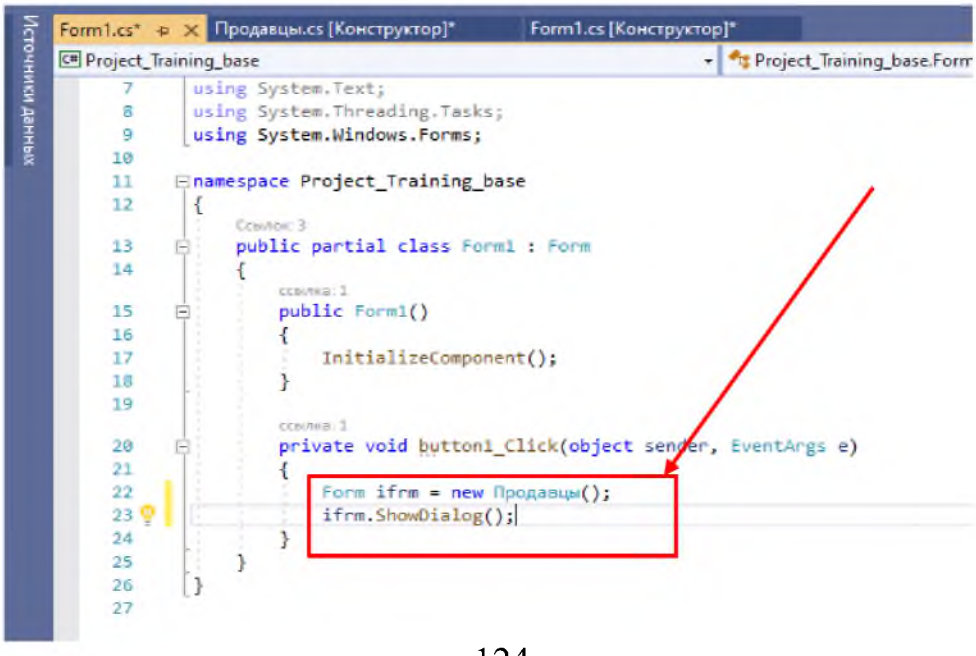


Рис. 94. Код обработки нажатия на кнопку открывающую форму Продавцы

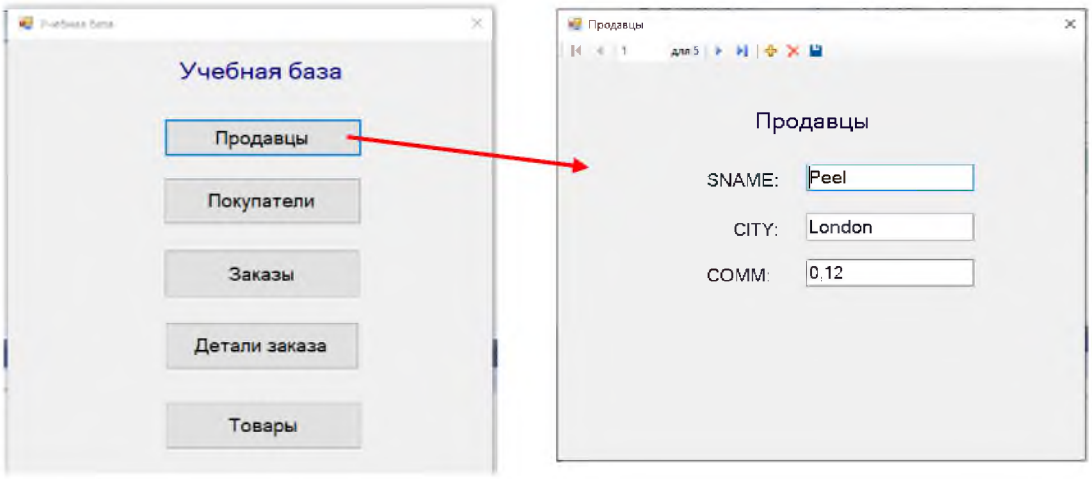


Рис. 95. Открытие формы Продавцы по кнопке на главной форме

13. Создайте форму Покупатели (табличный вид). Для этого добавьте в проект еще одну форму. Присвойте имя Покупатели.

14. Добавьте на форму таблицу для отображения данных (DataGridView) из таблицы «Customers». Для этого на панели «Источники данных» (“Data Sources”), нажмите кнопку раскрывающегося списка, расположенную справа от таблицы «Customers». В появившемся списке объектов для отображения всей таблицы выберите “DataGridView” (рис. 96).

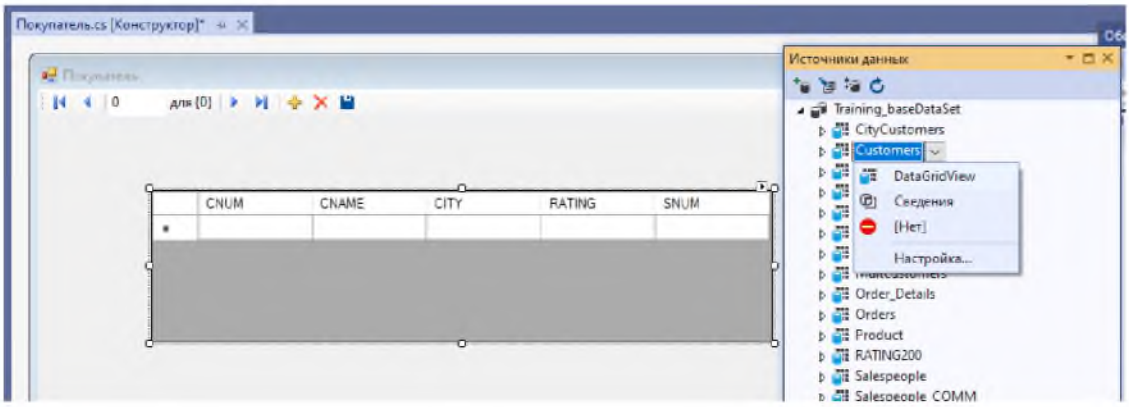


Рис. 96. Внешний вид формы с таблицей, опирающейся на информацию из источника данных

15. Поменяйте ширину всех столбцов на «150» чтобы столбцы вместились во все окно (рис. 97).

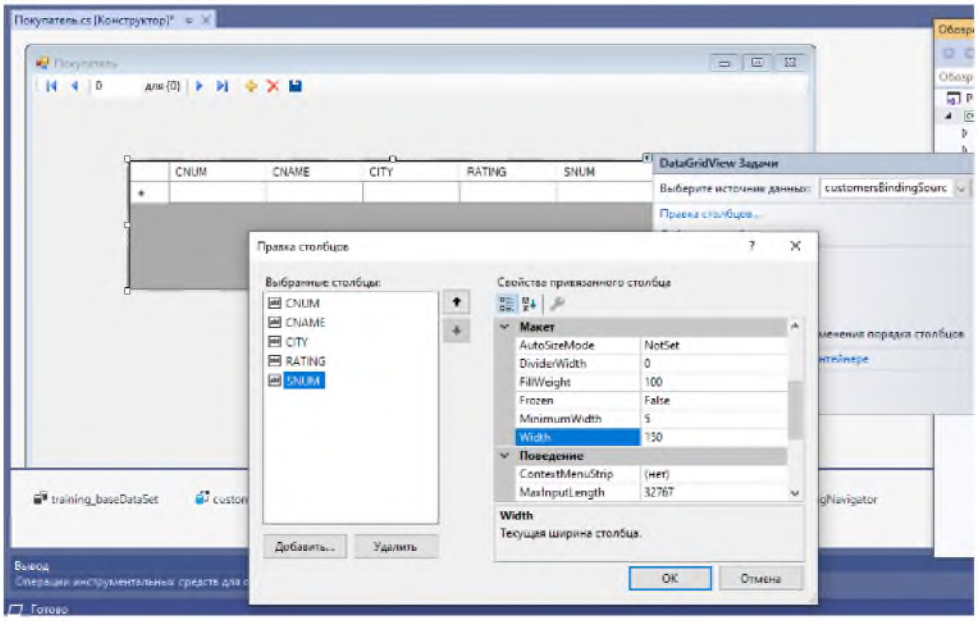


Рис. 97. Свойства привязанных столбцов

16. Подключите форму Покупатели к главной кнопочной форме, а затем запустить проект и открыть форму Покупатели при помощи кнопки на главной кнопочной форме (рис.98).

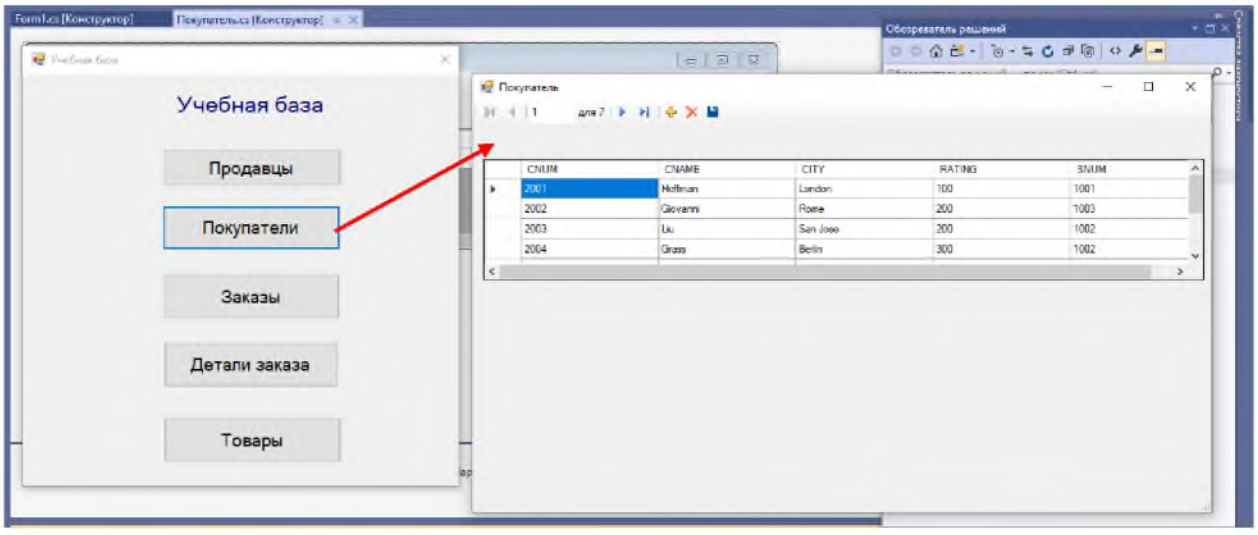


Рис. 98. Вид формы Покупатели

17. Данные мы видим, но редактировать невозможно и именно поэтому нужна кнопка «Сохранить».

18. Добавить на форму кнопку и в свойствах меняем имя на «SaveButton», рис. 99. После добавления кнопки нажимаем на нее два раза и пишем код (рис. 100).

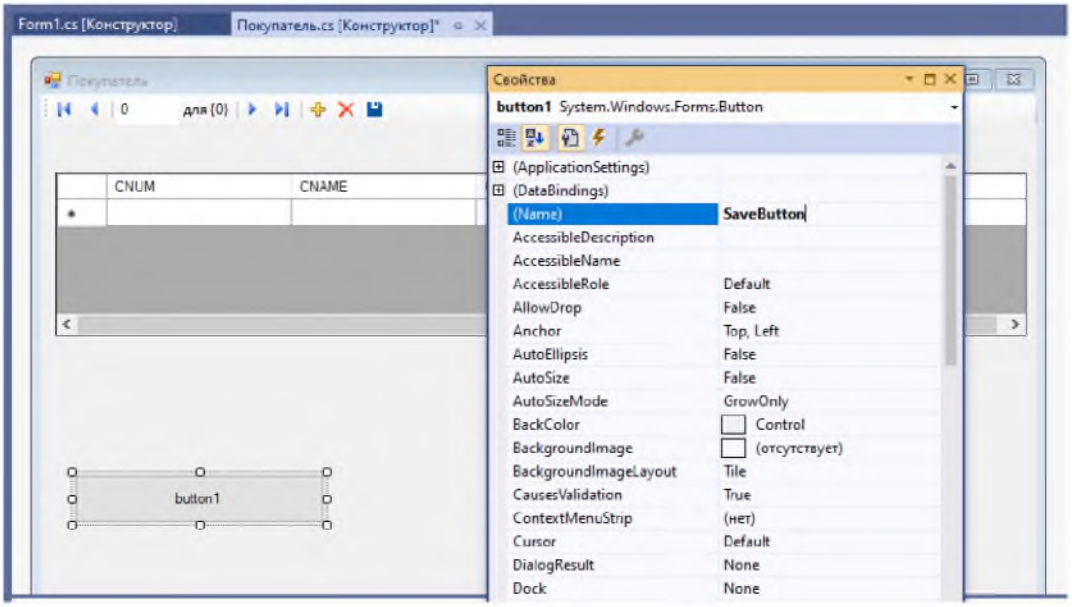


Рис. 99. Добавление кнопки

```
34
        ссылка: 1
35      private void SaveButton_Click(object sender, EventArgs e)
36      {
37          customersTableAdapter.Update(training_baseDataSet);
38      }
```

Рис. 100. Код обработки кнопки Сохранить

Вышеуказанный код обновляет запись в нашей таблице “Customers”. TableAdapter использует команды данных для чтения и записи в базу данных. В “Training_baseDataSet” хранятся настройки подключения базы данных.

После добавления кода запустите свое приложение и попробуйте изменить любую запись и сохранить (рис. 101).

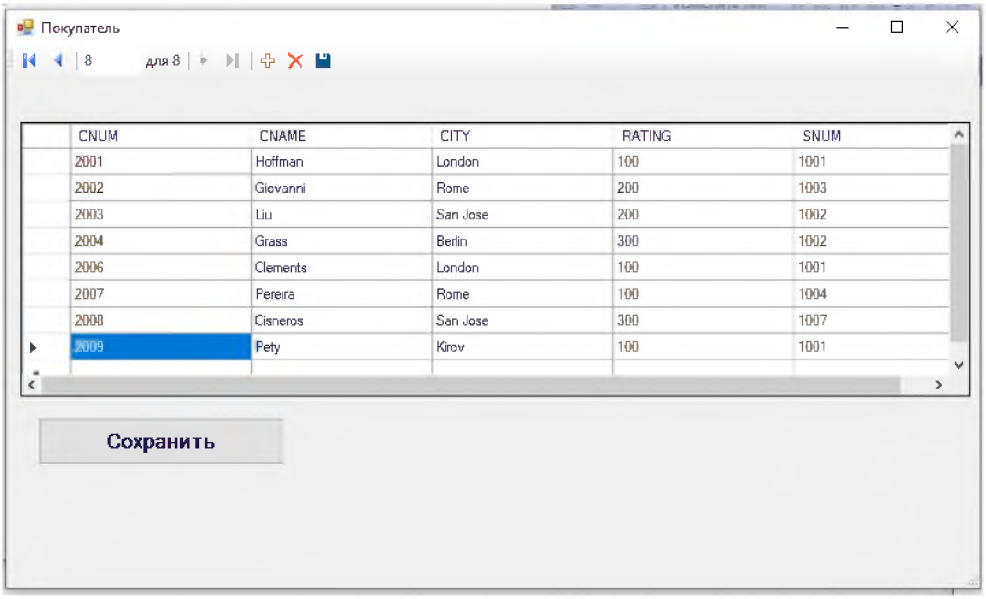


Рис.101. Добавление новой записи

Защита от случайного удаления

Чтобы пользователь случайно не удалил запись в нашей базе данных нужно сделать так чтобы перед удалением приложение спрашивало об удалении записи.

Для этого перейдем к компоненту DataGridView, откроем список событий (рис.101) и установим обработчик для события UserDeletingRow. Нажмите два раза на пустую строчку возле события и перейдете к коду (рис. 103).

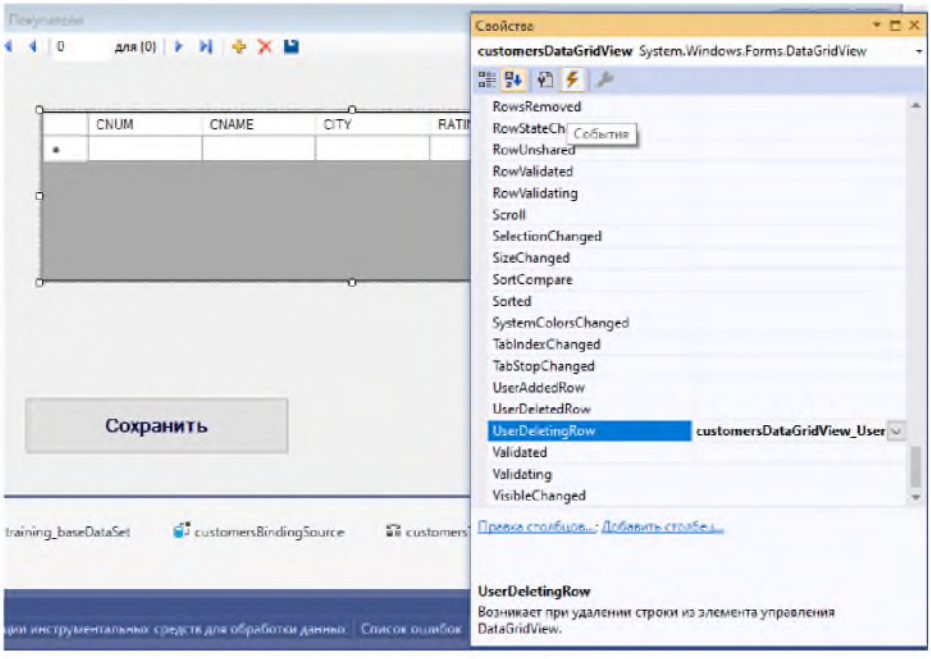


Рис. 102. Вкладка События на панели Свойства

Добавьте код представленный ниже и на рис.25:

```
DialogResult dr = MessageBox.Show("Удалить запись?", "Удаление",
MessageBoxButtons.OKCancel, MessageBoxIcon.Warning,
MessageBoxDefaultButton.Button2);
            if (dr == DialogResult.Cancel)
            {
                e.Cancel = true;
             }
```

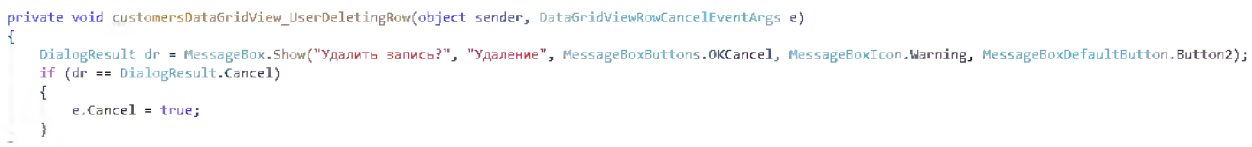


Рис. 103. Код на проверку удаления

После этого проверьте работоспособность данного кода. Запустите приложение и попробуйте удалить запись. Как только пользователь выделит строчку и нажмет кнопку “Delete” должно сработать событие “dataGridView1_UserDeletingRow” и появиться диалоговое окно с вопросом об удалении (рис. 104).

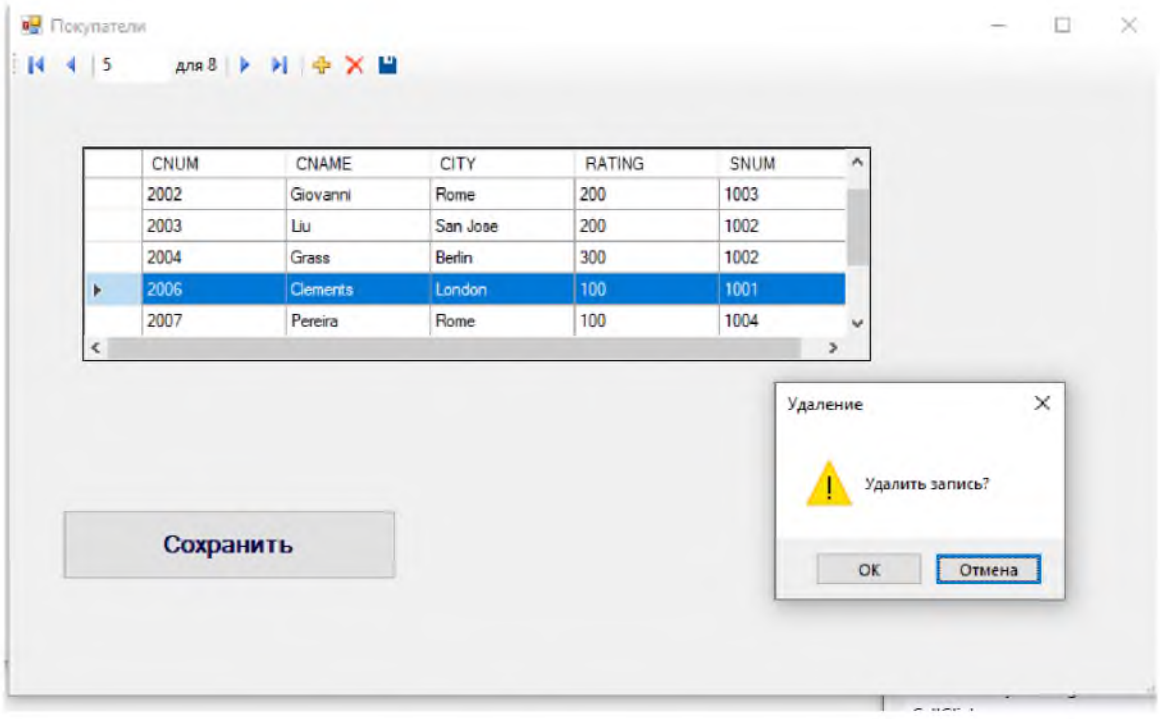


Рис. 104. Попытка удалить запись

19. Добавьте в проект новую форму Продукты.

20. Поместите на нее следующие объекты:

— четыре надписи (Label);

— пять кнопок (Button);

— выпадающий список (ComboBox);

— текстовое поле ввода (TextBox);

— группирующую рамку (GroupBox);

— список (ListBox);

— два переключателя (RadioButton).

21. Расположите объекты как показано на рис. 105.

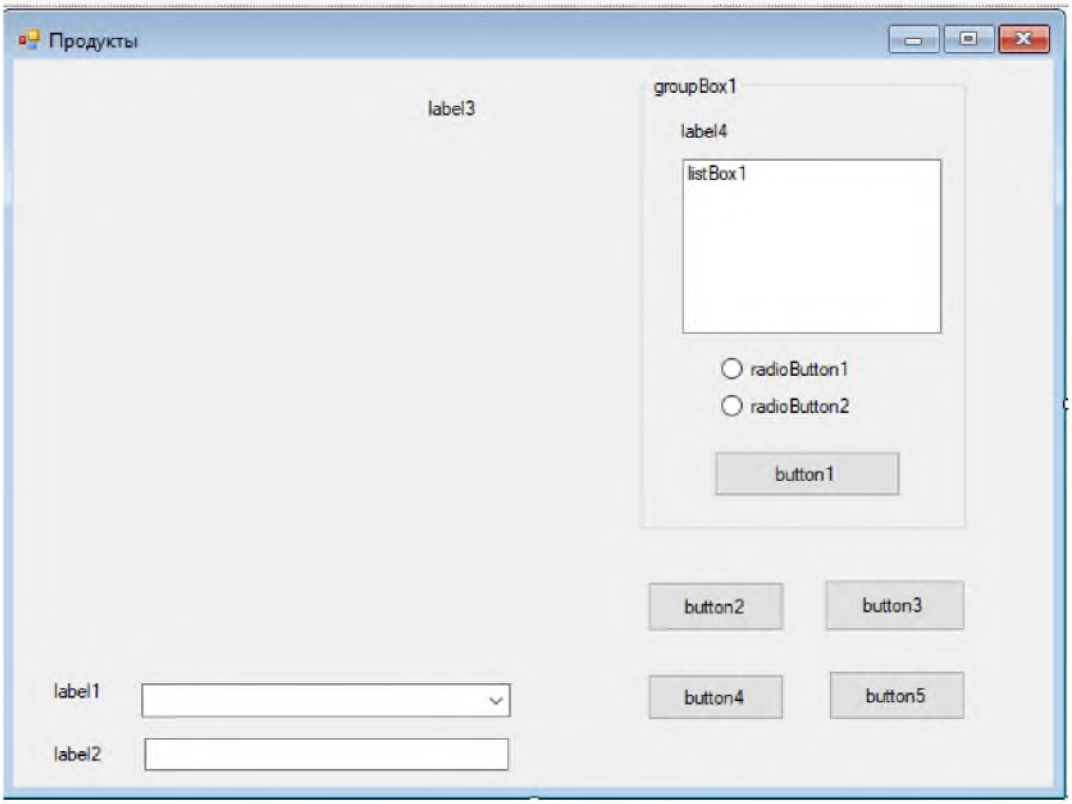


Рис.105. Элементы формы Продукты

Добавьте на форму таблицу для отображения данных (DataGridView) из таблицы Product. Для этого на панели «Источники данных» (Data Sources), нажмите кнопку, расположенную справа от таблицы «Product». В появившемся списке объектов для отображения всей таблицы выберите «DataGridView». Перетащите таблицу «Products» из панели «Источники данных» на форму. Форма примет следующий вид (рис. 106).

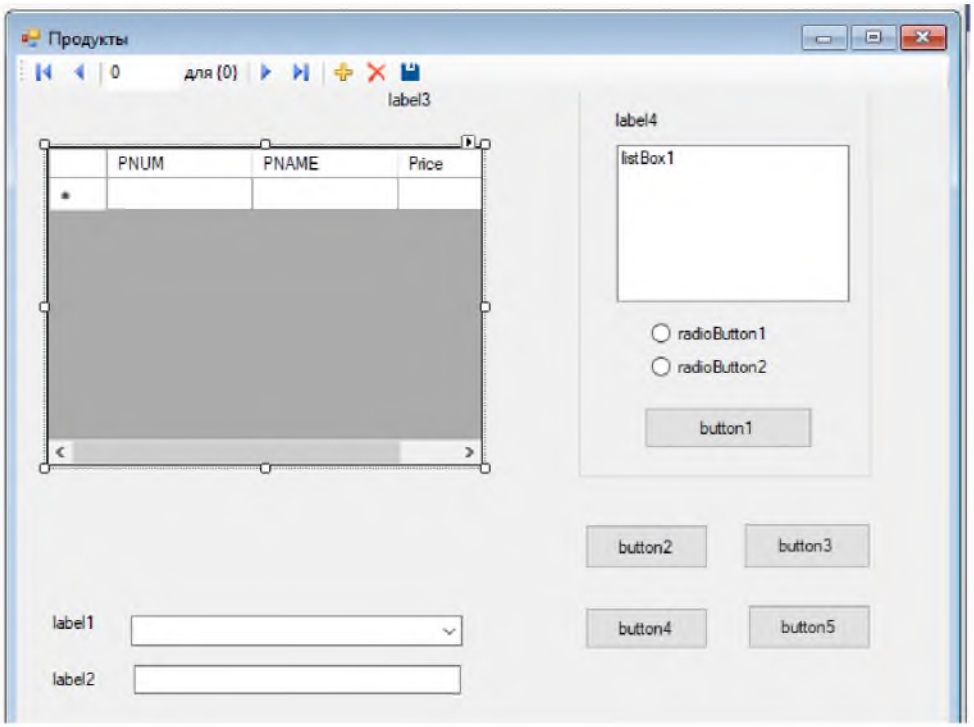


Рис. 106. Добавление таблицы Product на форму

Теперь необходимо настроить свойства объектов. Задайте свойства формы следующим образом:

FormBorderStyle (Стиль границы формы): Fixed3D;

MaximizeBox (Кнопка развёртывания формы во весь экран): False;

MinimizeBox (Кнопка свёртывания формы на панель задач): False;

Text (Текст надписи в заголовке формы): Продукты (Табличный вид).

Задайте свойства надписей (Label1, Label2, Label3 и Label4)

AutoSize (Авторазмер): False;

Text (Текст надписи): «Продукты (Табличный вид)», «Поле для сортировки», «Наименование» и «Критерий».

Задайте надписи на кнопках как: «Сортировать», «Фильтровать», «Показать все», «Найти» и «Закрыть». Для того чтобы нельзя было произвести сортировку не выбрав поля изначально заблокируем кнопку «Сортировать» (свойство Enabled = false).

У группирующей рамки задайте заголовок «Сортировка». У переключателей

(Объекты RadioButton1 и RadioButton2) задайте надписи как «Сортировка по возрастанию» и «Сортировка по убыванию», а у переключателя «Сортировка по возрастанию» (RadioButton1) задайте свойство Checked (Включён) равное True (Истина).

Заполните список (ListBox1) значениями (Коллекция):

— Наименование;

— Цена.

Настроим таблицу для отображения данных, удалив из неё поля с кодами. Выделите таблицу на форме и отобразите её меню задач, кликнув ЛКМ по кнопке, расположенной в верхнем правом углу таблицы. В меню действий выберите пункт «Правка столбцов…» (рис. 107).

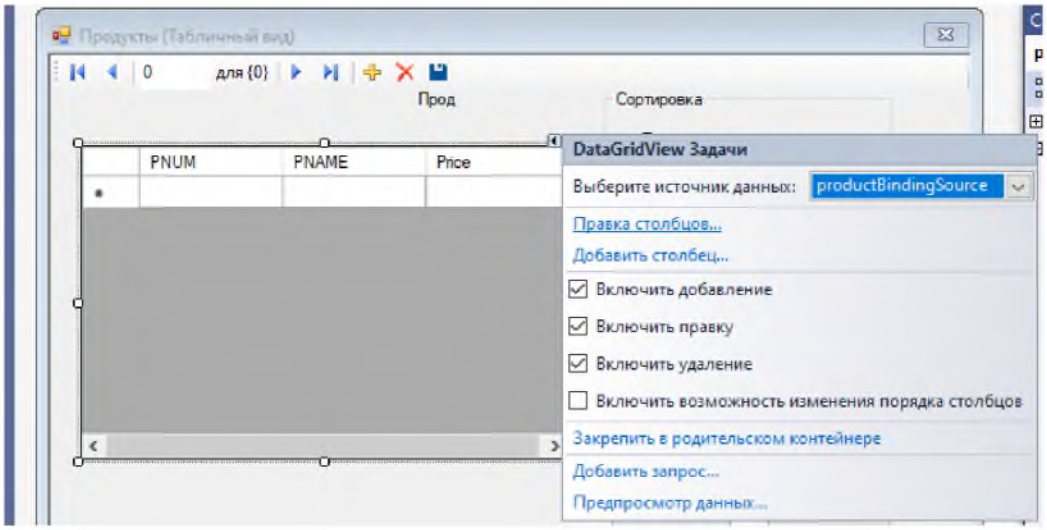


Рис. 107. Меню задачи объекта DataGridView

В окне «Edit Columns» из списка полей удалите пол PNUM выделив его и нажав кнопку Удалить. Также измените свойство HeaderText, чтобы наименование столбцов было на русском языке. Список полей примет вид, показанный на рис. 108.

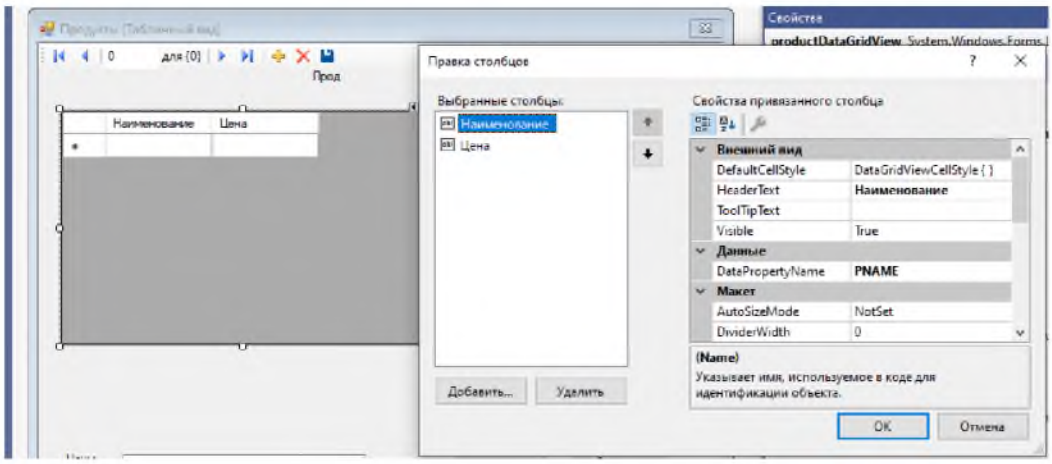


Рис. 108. Окно Правка столбцов

Настройте заполнение выпадающего списка наименованием продуктов из таблицы Product. Отобразите меню действий выпадающего списка. Включите опцию «Использовать элементы, привязанные к данным». В строке Источник данных выберите Другие источники данных, Источники данных проекта, Computer_ShopDataSet, ProductBindingSource. В строке Отобразить члена выберите PNAME (рис. 109).

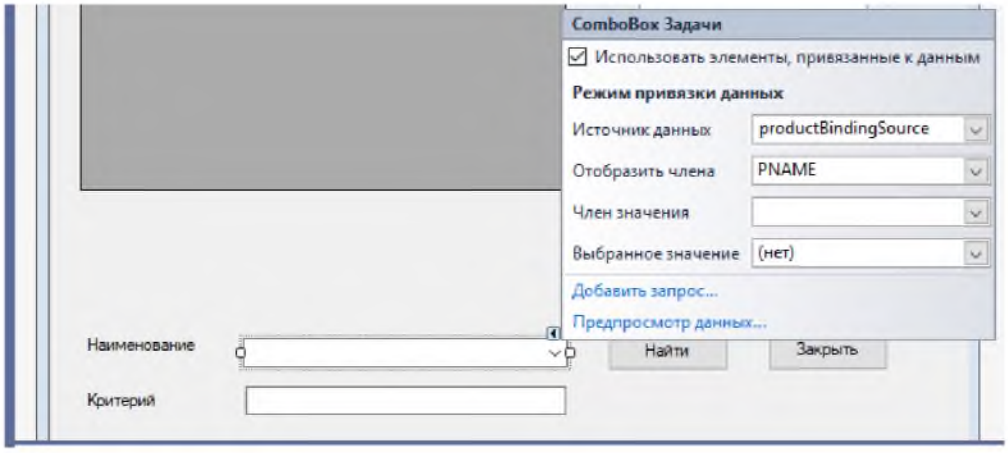


Рис. 109. Настройка ComboBox

На панели невидимых объектов появится дополнительный объект связи productBindingSource, предназначенный для заполнения выпадающего списка.

Работу с кодом начнём написания кода для разблокирования кнопки «Сортировать», при выборе пункта списка (ListBox1). Для создания процедуры события дважды щёлкните ЛКМ по списку. Появится процедура обработки события, происходящего при выборе пункта списка (ListBox1_SelectedIndexChanged), рис. 110. В процедуре наберите команду разблокировки кнопки «Сортировать» (Button1): Button1.Enabled = True.

```
private void listBox1_SelectedIndexChanged(object sender,
EventArgs e)
        {
            button1.Enabled = true;
        }
```

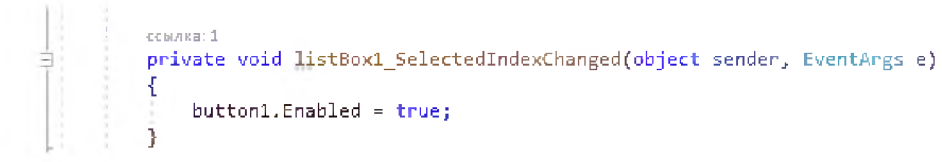


Рис. 110. Процедура обработки события

Теперь перейдём к созданию кода, сортирующего нашу таблицу в зависимости от выбранного поля и порядка сортировки при нажатии кнопки Сортировать, рис. 111. Дважды щёлкните ЛКМ по кнопке «Сортировать». Появится процедура «Button1_Click», выполняемая при щелчке ЛКМ по кнопке. В процедуре наберите код, представленный ниже:

```
private System.Windows.Forms.DataGridViewColumn COL;
        private void button1_Click(object sender, EventArgs e)
        {
            //создает переменную COL для хранения имени
выбранного столбца таблицы
            COL = new System.Windows.Forms.DataGridViewColumn();

            //блок switch, присваивающий в
            //переменную Col имя выбранного столбца таблицы в
зависимости от номера
            //выбранного пункта списка (ListBox1.SelectedIndex).
Если выбран первый пункт
            //списка, то в переменную Col записывается стлбец
            //DataGridViewTextBoxColumn2, если второй, то -
DataGridViewTextBoxColumn3
            //и так далее. Хотелось бы отметить тот факт, что
нумерация пунктов списка
```

```
            //начинается с нуля, а нумерация столбцов с единицы.
Первый столбец Наименование

            //носит имя DataGridViewTextBoxColumn2, так как имя
            //DataGridViewTextBoxColumn1 имеет столбец
заголовков строк;

            switch (listBox1.SelectedIndex)
            {
                case 0:
                    COL = dataGridViewTextBoxColumn2;
                    break;
                case 1:
                    COL = dataGridViewTextBoxColumn3;
                    break;


                }
//Блок If выполняет следующую операцию: если включён
            //переключатель «Сортировка по возрастанию»
(RadioButton1), то отсортировать
            //таблицу по полю заданному в переменной Col по
возрастанию
            //(pRODUCTSDataGridView.Sort (Col,
System.ComponentModel.ListSortDirection.
            //Ascending)), иначе по убыванию
(pRODUCTSDataGridView.Sort (Col, System.
            //ComponentModel.ListSortDirection. Descending)).
            if (radioButton1.Checked)
                productDataGridView.Sort(COL,

System.ComponentModel.ListSortDirection.Ascending);
            else
                productDataGridView.Sort(COL,

System.ComponentModel.ListSortDirection.Descending);
             }
```

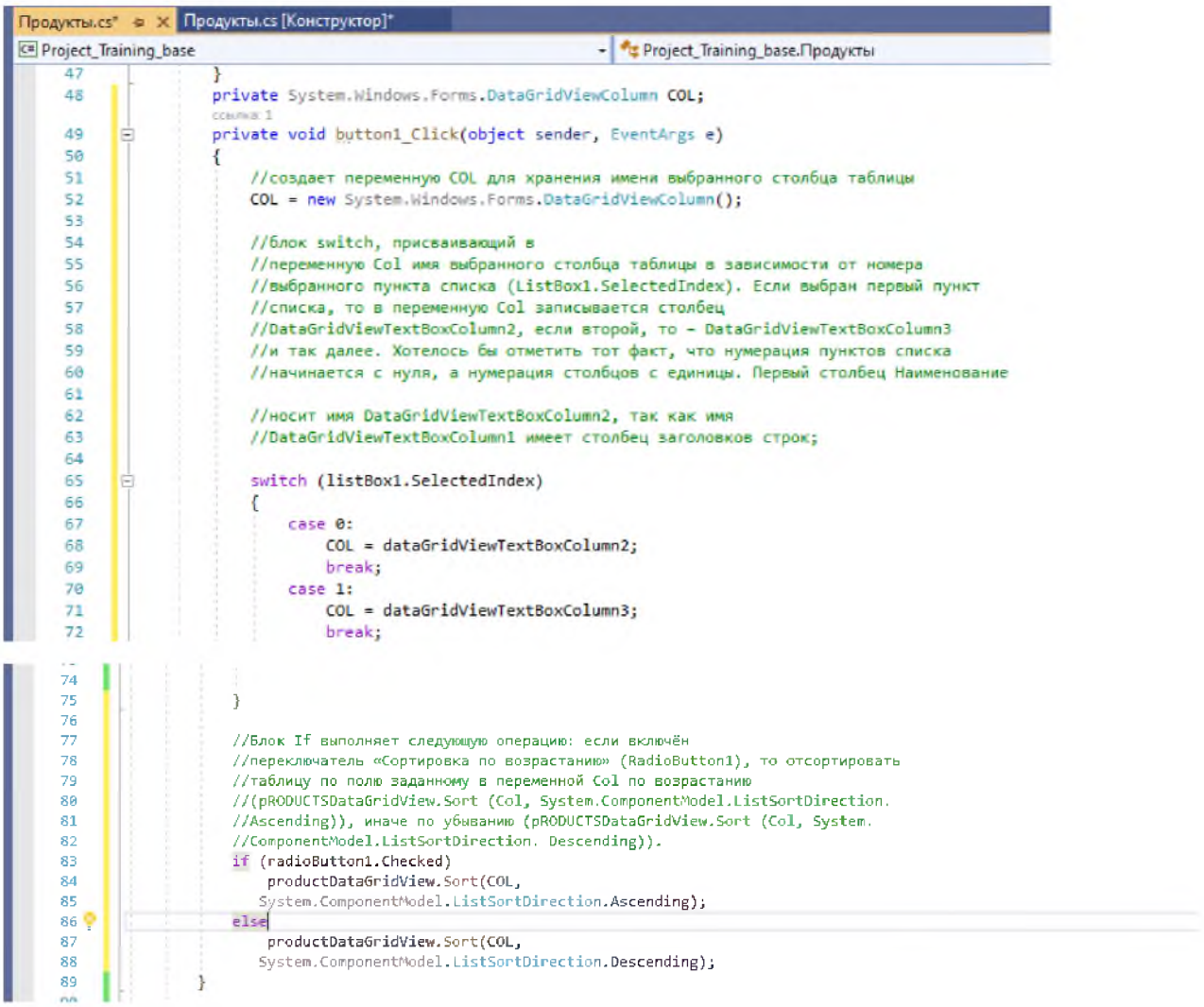


Рис. 111. Процедура обработки события нажатия кнопки Сортировать

Рассмотрим код обработчика события нажатия кнопки «Фильтровать». Дважды щёлкните по кнопке «Фильтровать» и в процедуре обработки события «Button2_Click» наберите код, рис. 112:

```
productBindingSource.Filter = "PNAME='" + comboBox1.Text +
"'";
```

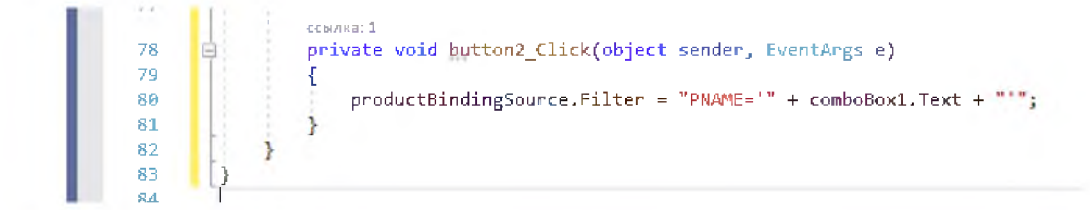


Рис. 112. Код обработки события нажатия кнопки Фильтровать

У объекта productBindingSource имеется текстовое свойство Filter, которое определяет условие фильтрации. Условие фильтрации имеет синтаксис: “<Имя поля><Оператор>’<Значение>’”.В нашем случае значение поля «NAME» приравнивается к значению, выбранному в выпадающем списке (ComboBox1.Text).

Теперь перейдём к кнопке «Показать всё», отменяющей фильтрацию записей. Дважды щёлкните по вышеперечисленной кнопке. Появится процедура Button3_Click, рис. 113. В появившейся процедуре наберите команду, как показано ниже:

```
productBindingSource.Filter = "";
```

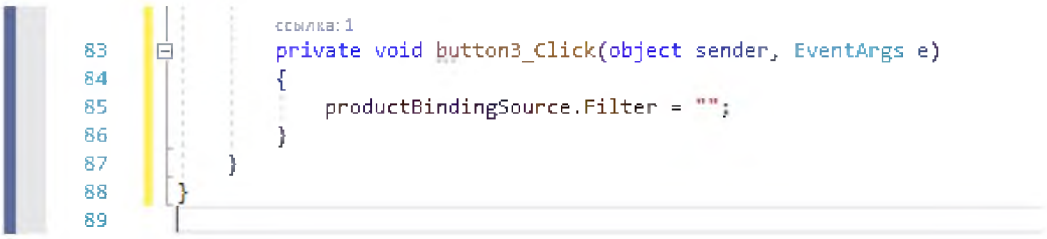


Рис. 113. Код обработки события нажатия кнопки Показать все

Заметим, что если присвоить свойству «Filter» значение пустой строки (“”), то его действие будет отменено.

Далее рассмотрим реализацию поиска информации в таблице. Дважды щёлкните по кнопке «Найти», рис. 114. В появившейся процедуре обработки нажатия кнопки «Button4_Click» наберите код, как показано ниже:

```
private void button4_Click(object sender, EventArgs e)
        {
            //перебирает все ячейки таблицы и
            //устанавливает в них белый цвет фона и чёрный цвет
текста, то есть,
            //отменяет результаты предыдущего поиска
            for (int i = 0; i < productDataGridView.ColumnCount
- 1; i++)
            {
                for (int j = 0; j < productDataGridView.RowCount
- 1; j++)
                {
                    productDataGridView[i, j].Style.BackColor =
Color.White;
                    productDataGridView[i, j].Style.ForeColor =
Color.Black;
                }
            }
            //перебирает все ячейки таблицы и если они
            //содержат текст, введённый в поле ввода (TextBox1),
то устанавливает в них
            //голубой цвет фона и синий цвет текста, чем
выделяет искомые ячейки.
            for (int i = 0; i < productDataGridView.ColumnCount
- 1; i++)
            {
                for (int j = 0; j < productDataGridView.RowCount
- 1; j++)
                {
                    if (productDataGridView[i,
                   j].Value.ToString().IndexOf(textBox1.Text) !=
-1)
                    {
                        productDataGridView[i,
j].Style.BackColor = Color.AliceBlue;
                        productDataGridView[i,
j].Style.ForeColor = Color.Blue;

                    }
```

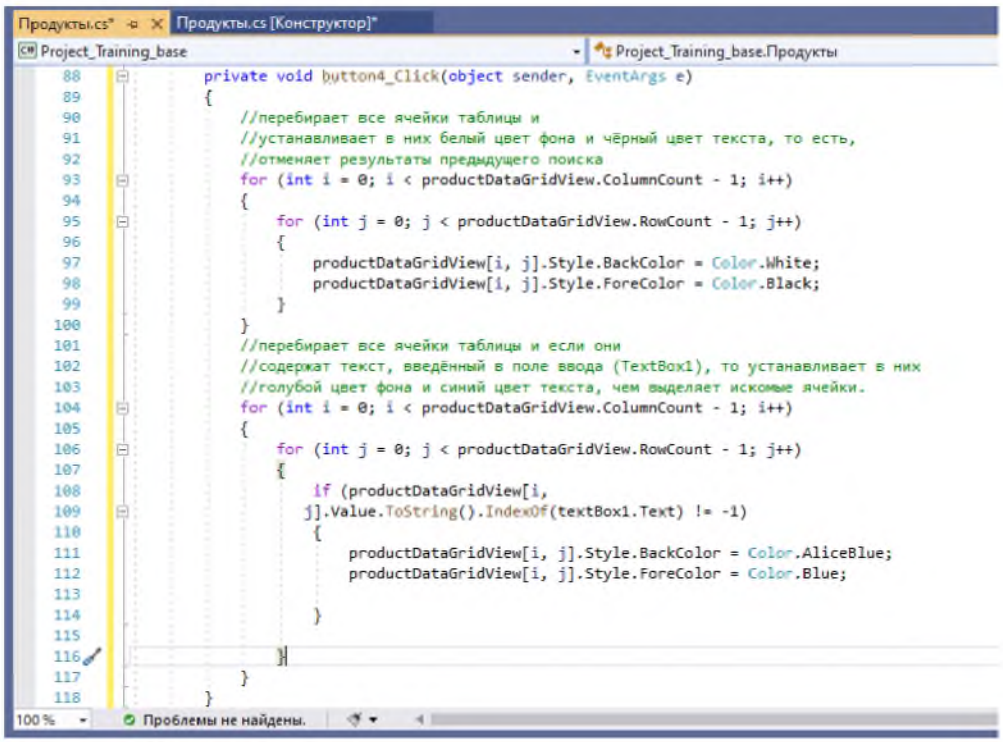


Рис. 114. Код обработки события нажатия кнопки Найти

Напишите код для кнопки Закрыть. Для этого дважды щёлкните ЛКМ по этой кнопке и в появившейся процедуре «Button5_Click» наберите код, рис. 115, как показано ниже:

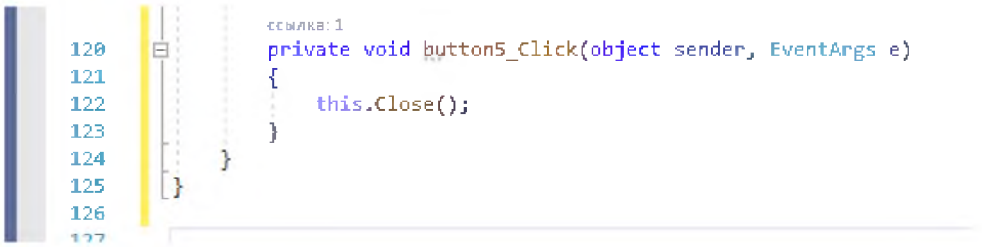


Рис. 115. Код обработки события нажатия кнопки Закрыть

Для отображения формы Продукты, её необходимо подключить к главной кнопочной форме, а затем запустить проект и открыть форму Продукты при помощи кнопки на главной кнопочной форме.

Проверьте работоспособность созданной табличной формы. Запустите проект и на главной кнопочной форме нажмите кнопку Продукты, должна появиться новая табличная форма (рис.116). Проверьте, как работает поиск, фильтрация и сортировка записей в таблице, нажимая на соответствующие

кнопки. После проверки работы формы для возвращения в среду разработки просто закройте все формы.

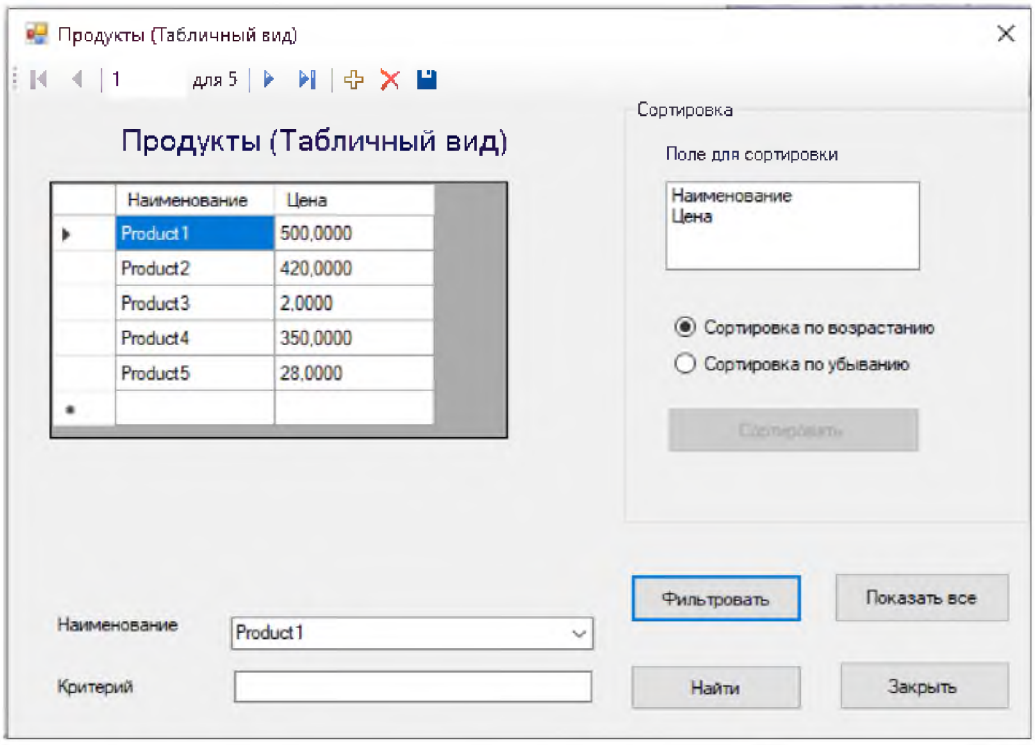


Рис. 116. Форма Продукты

Самостоятельная работа 5

1. В среде Visual Studio создайте новое представление ITOGI (меню Средства- SQL Server – Создать запрос), рис. 117. Выполните и сохраните запрос на создание представления. В примере (рис. 117) рассматривается вариант, когда поле Price находится в таблице Order_Details. Если поле Price находится в таблице Product, то необходимо использовать оператор соединения таблиц.

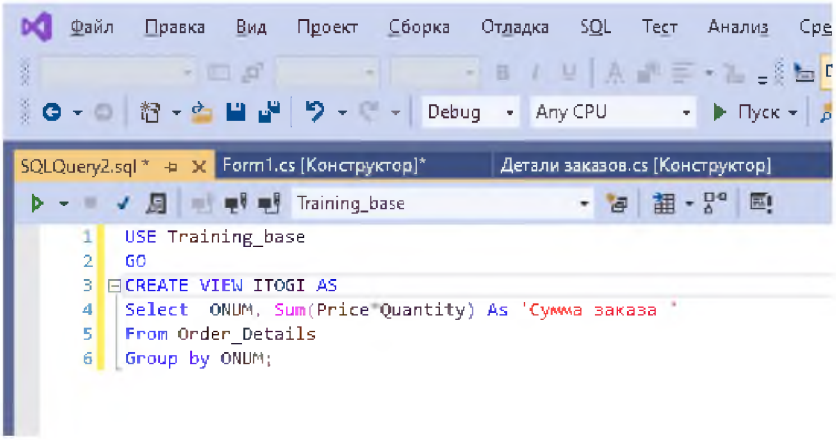


Рис. 117. Создание представления ITOGI

2. Создайте новые формы Итоги и Итоги_тв, оформите в соответствии с рис. 40, форма Итоги открывается по кнопке Заказы на главной форме, форма Итоги_тв по кнопке Тбличный вид формы Итоги, рис. 118.

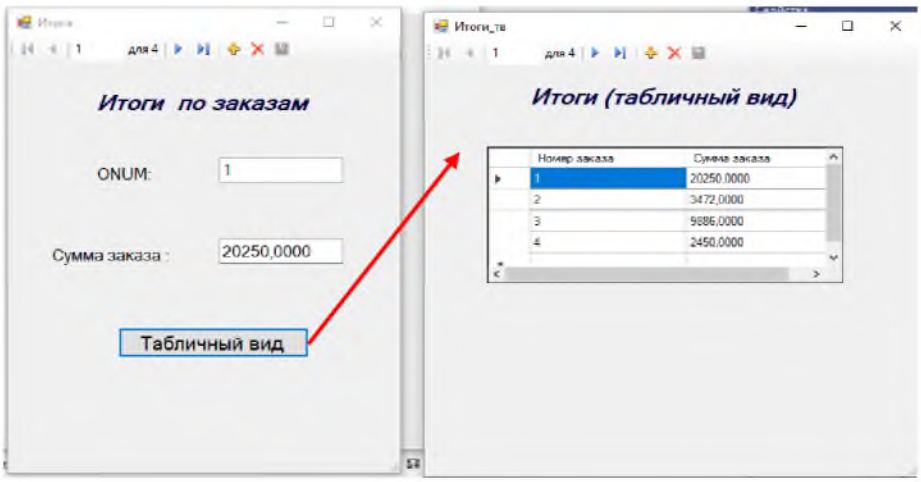


Рис. 118. Формы

3. Создайте форму Детали заказа (открываться должна по кнопке на главной форме), она должна содержать элементы управления (поле поиска, фильтр, сортировка, вычисляемое поле). Оформление выберите самостоятельно.

После выполнения лабораторной работы 8 в отчете создать разделы Лабораторная работа 8. В нем должна содержаться следующая информация:

— название лабораторной работы;

— цель работы;

— задание;

— вывод по работе.

Обязательно должны быть скриншоты (к лр 8):

— создания проекта;

— подключение БД к проекту;

— обозревателя серверов (после подключения БД);

— главной кнопочной формы;

— формы Продавцы (+ фрагмент кода формы);

— формы Покупатели (+ фрагмент кода формы);

— фрагмент кода кнопки Сохранить (формы Покупатели);

— фрагмент случайного удаления (+ фрагмент кода формы);

— формы Продукты (работоспособность сортировки, выбора критерия …);

— создания представления ITOGI в VS;

— формы Итоги (+ фрагмент кода формы);

— формы Итоги_тв (+ фрагмент кода формы);

— формы Детали_заказа (+ фрагмент кода формы).

СПИСОК ЛИТЕРАТУРЫ

1. Мартин Грабер. SQL для простых смертных.
2. Астахова И. Ф., Мельников В. М., Толстобров А. П., Фертиков В. В. СУБД: язык SQL в примерах и задачах. — М.: ФИЗМАТЛИТ, 2009. — 168 с. — ISBN 978-5-9221-0816-4.
3. Т. Карпова. Базы данных. Модели, разработка, реализация. – СПб.
4. Медведкова И.Е., Бугаев Ю.В., Чикунов С.В. Базы данных.
5. Винокурова И.В. Методические указания К Практическим занятиям. Институт менеджмента, маркетинга и финансов, 2015.
6. Прикладное программирование и Базы данных. Учебно-методическое пособие для практических работ. О.В. Игнатьева. РГУПС. Ростов-на-Дону 2017.