

Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Минцаев Магомед Шахмухамедович
Должность: Ректор
Дата подписания: 04.10.2023 17:17:04
Уникальный программный ключ:
236bcc35c296f119d6aafdc22836b21db52dbc07071e86865e58235f0fa4304cc

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ГРОЗНЕНСКИЙ ГОСУДАРСТВЕННЫЙ НЕФТЯНОЙ**

**ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
имени академика М.Д. Миллионщикова**

Кафедра «Информационные технологии»

И.Р. Усамов, Л.С. Умарова

**Методические указания к выполнению лабораторных работ
по дисциплине «Управление базами данных»**

**Направление подготовки
(ИВТ ?)**

Направленность (профиль)

**Квалификация
бакалавр**

Грозный 2023

Содержание

Введение	3
Лабораторная работа №1	4
Лабораторная работа №2	17
Лабораторная работа №3	23
Лабораторная работа №4	36
Лабораторная работа №5	40
Лабораторная работа №6	45
Лабораторная работа №7	49
Лабораторная работа №8	61
Лабораторная работа №9	68
Использованные источники.....	76
Список литературы.....	77

Введение

Цель дисциплины «Управление данными»: заключается в изучение теоретических основ и приобретение студентами практических навыков по использованию современных технологий сбора, обработки, хранения и передачи информации на основе систем управления базами данных (СУБД); в обучении принципам построения информационных моделей данных и проведения анализа полученных результатов; выработка умения практического использования команд языка SQL для решения задач пользователя и администратора; а также формирование умений использовать нормативно-правовые документы, международные и отечественные стандарты в области баз данных; подготовка к научно-исследовательской и производственной деятельности бакалавров, связанной с проектированием, эксплуатацией и сопровождением баз данных.

Задачи изучения дисциплины:

- ознакомление с основными понятиями и терминологией информационных систем на основе баз данных;
- выработку умения практического использования команд языка SQL для решения задач пользователя и администратора;
- формирование умений проводить описание информационного обеспечения решения прикладных задач;
- ознакомление с проблемами и возможностями администрирования в СУБД;
- изучение принципов построения баз данных различной архитектуры
- изучение способов защиты данных в СУБД.

Лабораторная работа №1. Интерфейс СУБД MSSQL

Для выполнения данной лабораторной работы Вам понадобится установленная СУБД MicrosoftSQLServer версии не ниже 2014. Взаимодействие с СУБД MicrosoftSQLServer осуществляется через графическую оболочку MicrosoftSQLServerManagementStudio.

Задание

Используя инструмент MicrosoftSQLManagementStudio создать базу данных спроектированную в лабораторной работе.

Ход работы

1. Установить SQLServer версии не ниже 2014 если не установлен, процесс установки подробно описан в данном методическом указании (задав пароль для пользователя student);
2. Осуществить подключение к БД, используя логин: student и пароль, указанный при установке сервера;
3. Используя инструмент SQLManagementStudio БД и 1 таблицу с помощью мастера;
4. Написать отчет о проделанной работе, оформленный согласно требованиям, включающий в себя:
 - титульный лист; цель работы;
 - выполненные задачи;
 - ход работы (в ходе работе представить SQL запросы на создание БД и и скриншот создаваемой таблицы);

Пример создания новой базы данных

Создание новой базы данных с помощью мастера

Для начала необходимо запустить среду разработки SQLServerManagement Studio. Для этого в меню «Пуск» выбираем пункт Программы\MicrosoftSQL Server

20XX\SQL Server Management Studio

После запуска среды разработки появится окно подключения к серверу Соединение с сервером (рисунок 1.). В этом окне необходимо нажать кнопку Соединить.

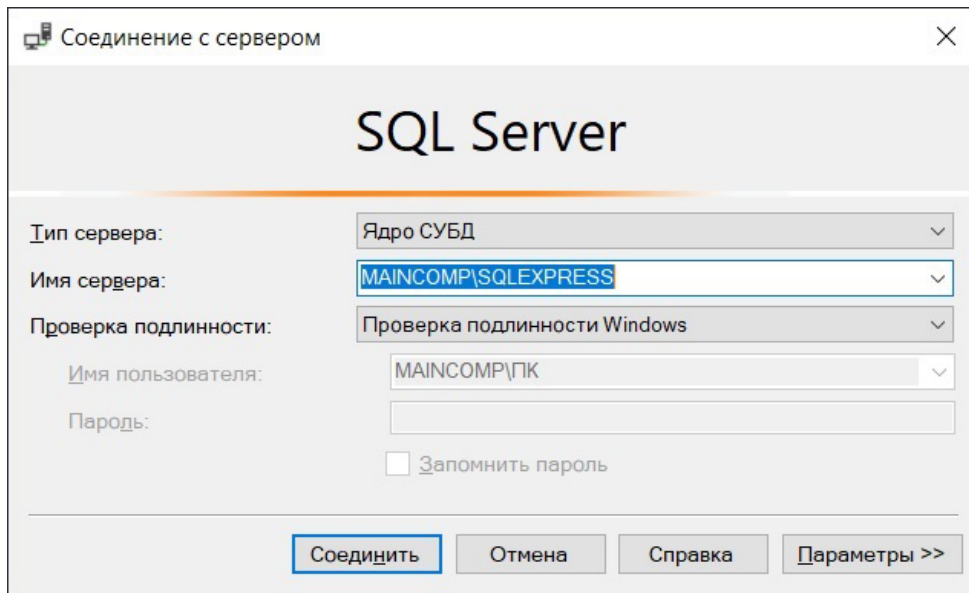


Рисунок 1.1 – Соединение с сервером

Если в процессе установки Microsoft SQL Server 20XX был задан логин и пароль подключения к серверу, то перед нажатием кнопки «Соединить», в выпадающем списке Проверка подлинности нужно выбрать Проверка подлинности SQL Server, а затем необходимо ввести заданные при установке логин и пароль.

После нажатия кнопки «Соединить» появится окно среды разработки SQL Server Management Studio (рисунке 1.2).

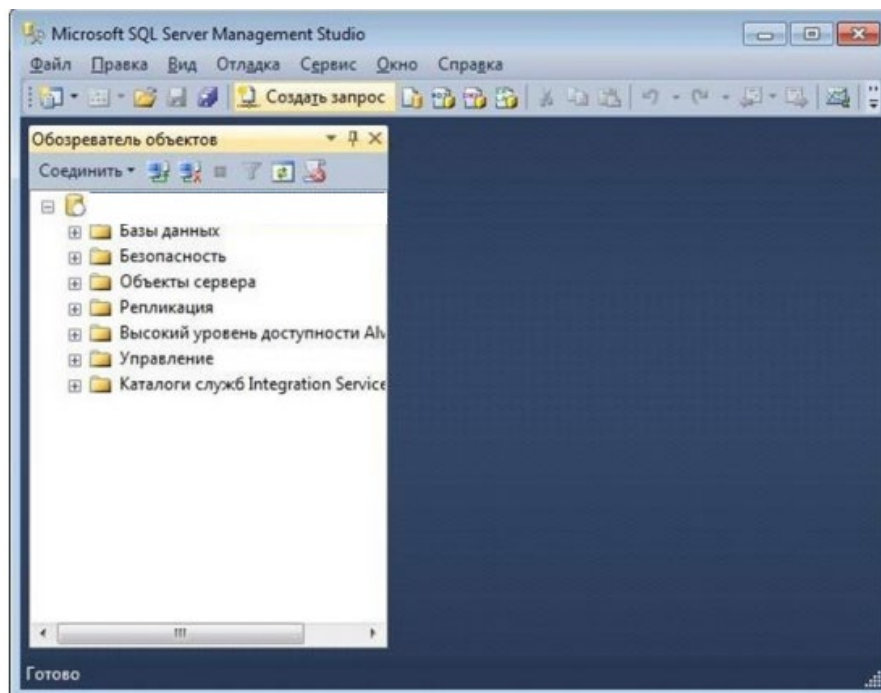


Рис.1.2. – Рабочее окно Server Management Studio

Данное окно имеет следующую структуру (рисунок 1.3):

В самом верху расположено меню, которое содержит полный набор команд для управления сервером и выполнения различных операций.

Ниже располагается панель, которая содержит кнопки для выполнения наиболее часто производимых операций. Внешний вид данной панели зависит от выполняемой операции.

Ниже расположена Панель обозревателя объектов – это панель с древовидной структурой, отображающая все объекты сервера, а также позволяющая производить различные операции, как с самим сервером, так и с БД. Обозреватель объектов является основным инструментом для разработки БД.

В обозревателе объектов сами объекты находятся в папках. Чтобы открыть папку необходимо щёлкнуть по знаку «+» слева от изображения папки.

Теперь перейдём непосредственно к созданию файла данных. Для этого в обозревателе объектов щёлкните правой кнопкой мыши на папке Базы данных (рисунок 3) и в появившемся меню выберите пункт «Создать базу данных...».

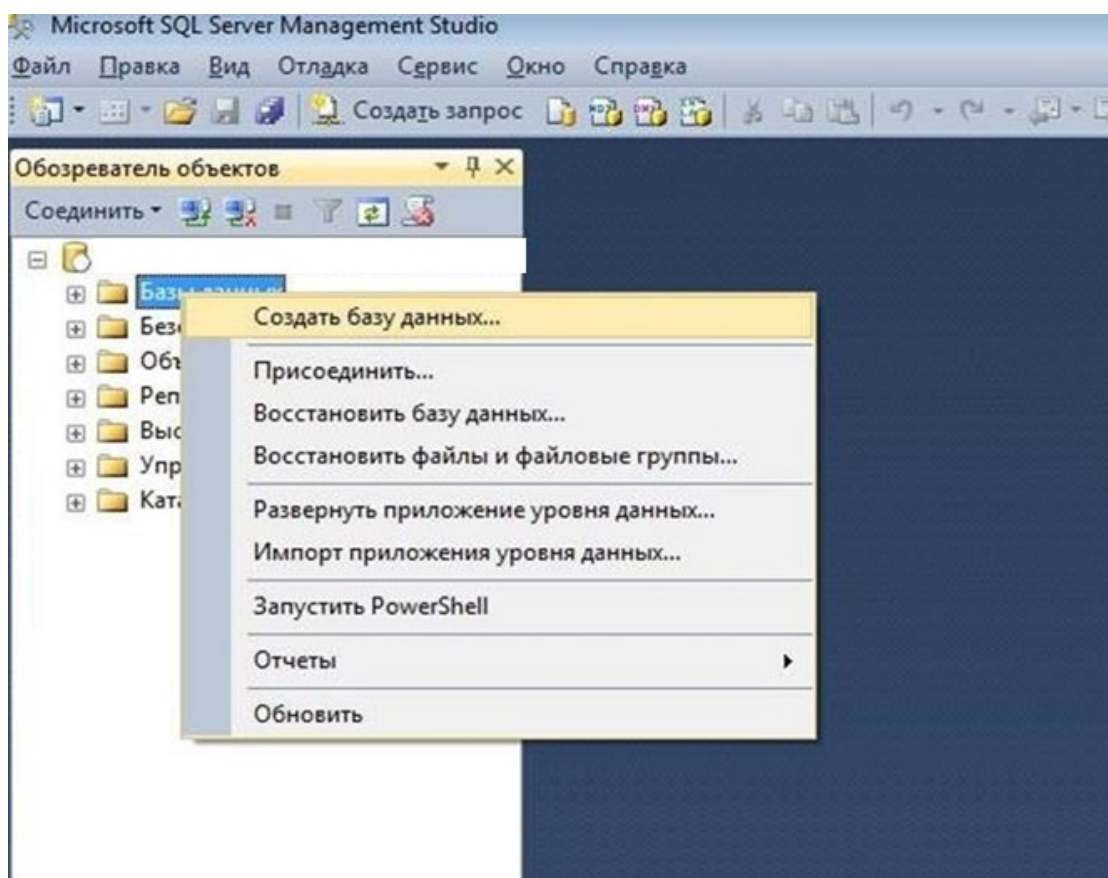


Рис.1.3. – Создание базы данных

Появится окно настроек параметров файла данных новой базы данных Создание базы данных (рисунок 1.4). В левой части окна настроек имеется список «Выбор страницы». Этот список позволяет переключаться между группами настроек.

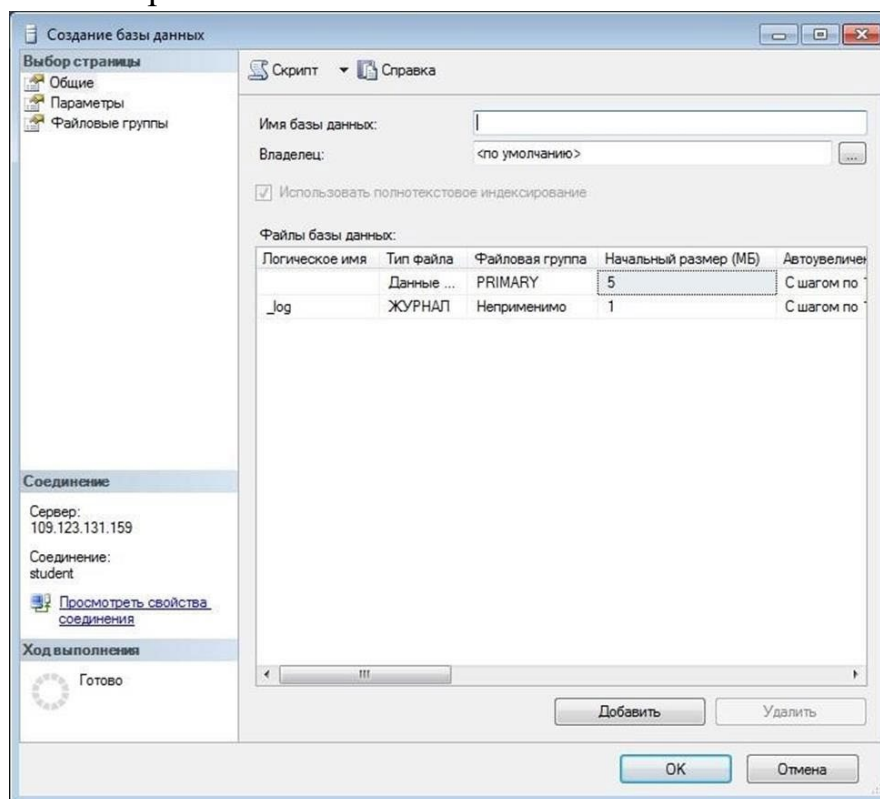


Рис.1.4. – Окно «Создание базы данных», страница «Общие»

Для начала настроим основные параметры. Для выбора основных настроек нужно щёлкнуть мышью по пункту «Общие» в списке «Выбор страницы». В правой части окна «Создание базы данных» появятся основные настройки (рисунок 4):

Верхней части окна расположено два параметра: «Имя базы данных» и «Владелец».

Необходимо задать параметр «Имя базы данных», в котором должно содержаться название группы, фамилия и название базы данных в таком формате 8VM64_Фамилия_ИмяБД.

Параметр Владелец оставьте без изменений. Под вышеприведёнными параметрами в виде таблицы располагаются настройки файла данных и журнала транзакций. Таблица имеет следующие столбцы:

- **Логическое имя** – логическое имя файла данных и журнала транзакций. По этим именам будет происходить обращение к вышеприведённым файлам в БД. Можно заметить, что файл данных имеет то же имя что и БД, а имя файла журнала транзакций составлено из имени БД и суффикса «_log».

Тип файла – этот параметр показывает, является ли файл файлом данных или журналом транзакций.

Файловая группа – группа файлов, показывает к какой группе файлов относится файл. Группы файлов настраиваются в группе настроек Файловая группа.

Начальный размер (МБ) – начальный размер файла данных и журнала транзакций в мегабайтах.

Авторасширение – как только файл заполняется информацией его размер автоматически увеличивается на величину, указанную в параметре Авторасширение. Увеличение можно задавать как в мегабайтах так и в процентах. Здесь же можно задать максимальный размер файлов. Для изменения этого параметра надо нажать кнопку «...». В нашем случае размер файлов не ограничен. Файл данных увеличивается на 1 мегабайт, а файл журнала транзакций на 10%.

Путь – путь к папке, где хранятся файлы. Для изменения этого параметра также надо нажать кнопку «...».

Имена файлов – по умолчанию имена файлов аналогичны логическим именам. Однако файл данных имеет расширение .mdf, а файл журнала транзакций – расширение .ldf. В рассматриваемом случае все основные настройки без изменений были оставлены без изменений. Теперь перейдём к другим второстепенным настройкам файла данных. Для доступа к этим настройкам необходимо щёлкнуть мышью по пункту Параметры в списке Выбор страницы. Появится следующее окно (рисунок 4). В правой части окна мы видим следующие настройки:

Параметры сортировки – этот параметр отвечает за обработку текстовых строк, их сравнение, текстовый поиск и т.д. Рекомендуется оставить его как есть. При этом данный параметр будет равен значению, заданному на вкладке Параметры сортировки, при установке сервера.

Модель восстановления – данный параметр отвечает за информацию, предназначенную для восстановления БД, хранящуюся в файле транзакций. Чем полнее модель восстановления, тем больше вероятность восстановления данных при сбое системы или ошибках пользователей, но и больше размер файла журнала транзакций. При наличии места на диске, рекомендуется оставить этот параметр в значении Простая.

Уровень совместимости – определяет совместимость файла данных с более ранними версиями сервера. Если планируется перенос данных на другую, более раннюю версию сервера, то её необходимо указать в этом параметре.

Другие параметры – данные параметры являются необязательными для изменения. В нашем случае все параметры в разделе Другие параметры, рекомендуется оставить как на рисунке 1.5.

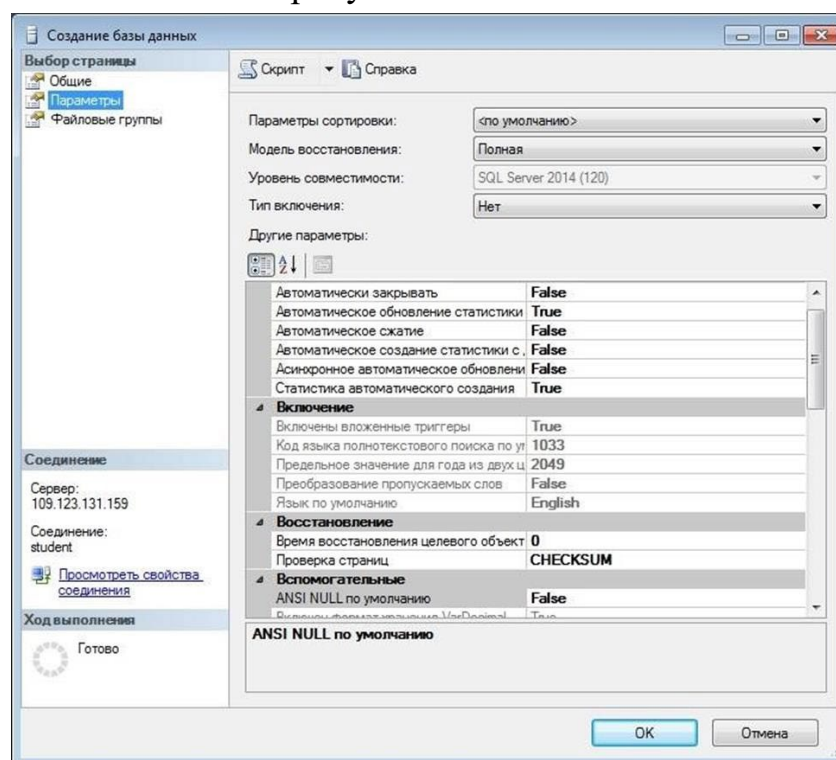


Рис. 1.5 – Окно «Создание базы данных», страница «Параметры»

Наконец рассмотрим последнюю группу настроек Файловые группы. Данная группа настроек отвечает за группы файлов (рисунок 1.5). Группы файлов представлены в таблице Строки в правой части окна (Рис.1.6). Данная таблица имеет следующие столбцы:

Имя – имя группы файлов.

Файлы – количество файлов, входящих в группу.

Только для чтения – файлы в группе будут только для чтения. То есть, их можно только просматривать, но нельзя изменять.

По умолчанию – группа по умолчанию. Все новые файлы данных будут входить в эту группу. В рассматриваемой БД нет необходимости добавлять новые группы файлов. Поэтому оставим группу настроек Файловые группы без изменений.

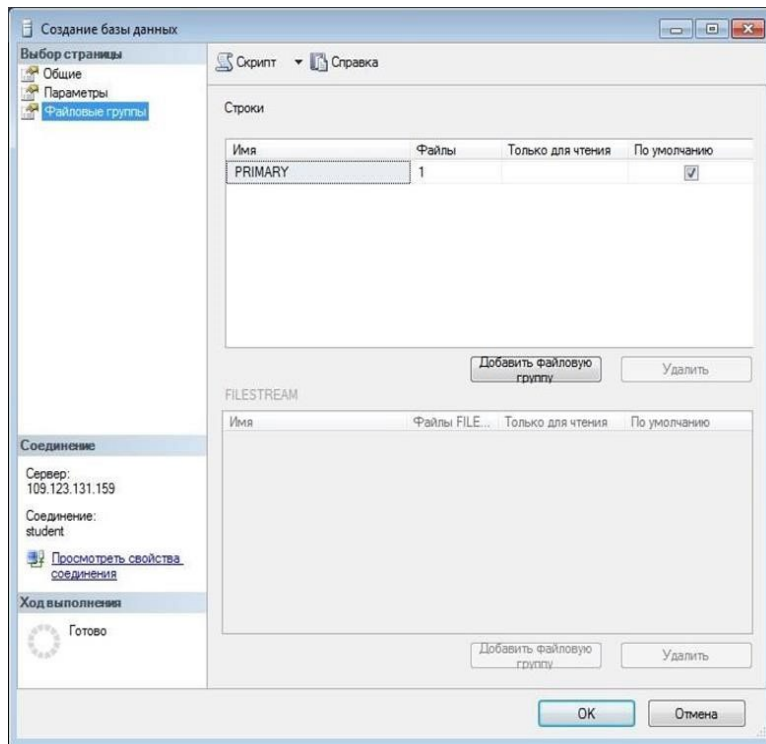


Рис. 1.6– Окно Создание базы данных, страница Файловые группы

На этом мы заканчиваем настройку свойств наших файлов. Для принятия всех настроек и создание фала данных и журнала транзакций нашей БД в окне «Создание базы данных» нажмём кнопку ОК.

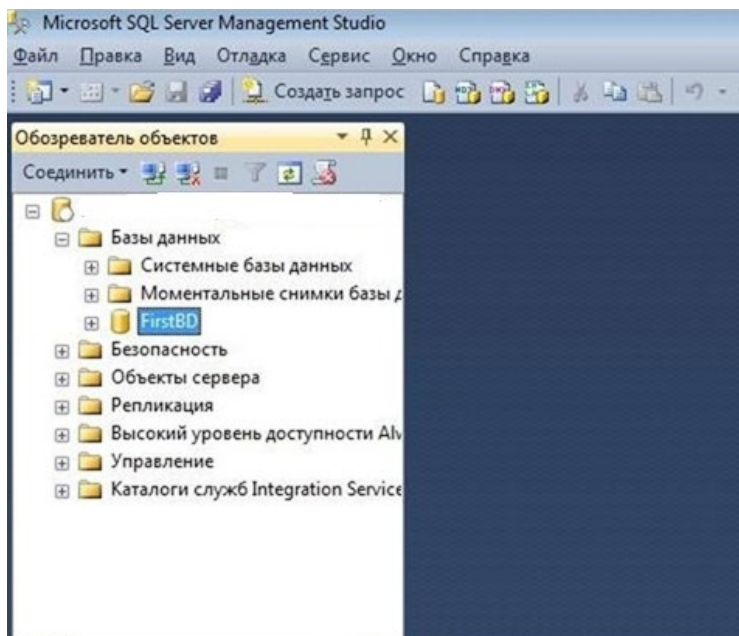
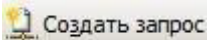
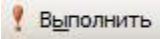


Рис. 1.7 – Результат создания БД

Создание новой БД с помощью запроса (Дополнительно)

Новую БД можно создать, используя стандартные команды языка T-SQL. Все команды языка T-SQL набираются на вкладке нового запроса (SQLQuery). Для того чтобы создать новый запрос на панели инструментов необходимо нажать кнопку . Для выполнения команд языка T-SQL на панели инструментов необходимо нажать кнопку  или на вкладке нового запроса набрать команду GO.

Для создания нового файла данных используется команда `CREATEDATABASE`, которая имеет следующий синтаксис:

```
CREATEDATABASE [Имя БД] ONPRIMARY  
(  
    NAME=<Логическоеимя>,  
    FILENAME=<Имяфайла>,  
    SIZE=<Нач.размер>,  
    MAXSIZE=<Макс.размер>,  
    FILEGROWTH=<Шаг>) LOGON (  
    NAME=<Логическоеимя>,  
    FILENAME=<Имяфайла>,  
    SIZE=<Нач.размер>,  
    MAXSIZE=<Макс.размер>,  
    FILEGROWTH=<Шаг>)  
)
```

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.

```
CREATEDATABASE [publishing] ONPRIMARY (  
    NAME='publishing',  
    FILENAME='D:\publishing.mdf',  
    SIZE= 5120KB,  
    MAXSIZE=UNLIMITED,  
    FILEGROWTH= 1024KB)  
LOGON (  
    NAME='publishing_log',
```

```
FILENAME='D:\publishing_log.ldf',  
SIZE= 1024KB,  
MAXSIZE= 2048GB,  
FILEGROWTH= 10% )  
GO
```

Комментарий: **если у пользователя нет прав доступа к диску данный пункт пропускаем**

Пример создания таблиц

Вся информация в базе данных хранится в таблицах. Таблицы состоят из записей. Запись – это строка в таблице. Вся информация обрабатывается по записям. Каждая запись состоит из полей. После это столбец таблицы. 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. 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**: *Real* семь знаков после запятой, *Float(m)* может хранить числа из *m* знаков, максимальное *m=38*, *Decimal(mn)* дробные числа с *m* знаков до запятой и *n* после;
 - **Специальные типы данных**: *Bit*– логический тип данных является заменой логическому типу *Boolean* в VisualBasic, *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($\pm 20000,0000$);

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

Создание таблиц с помощью мастера

Перейдем к созданию таблиц. Все таблицы нашей БД находятся в подпапке «Таблицы» папки «publishing» в окне обозревателя объектов (рисунок 1.8).

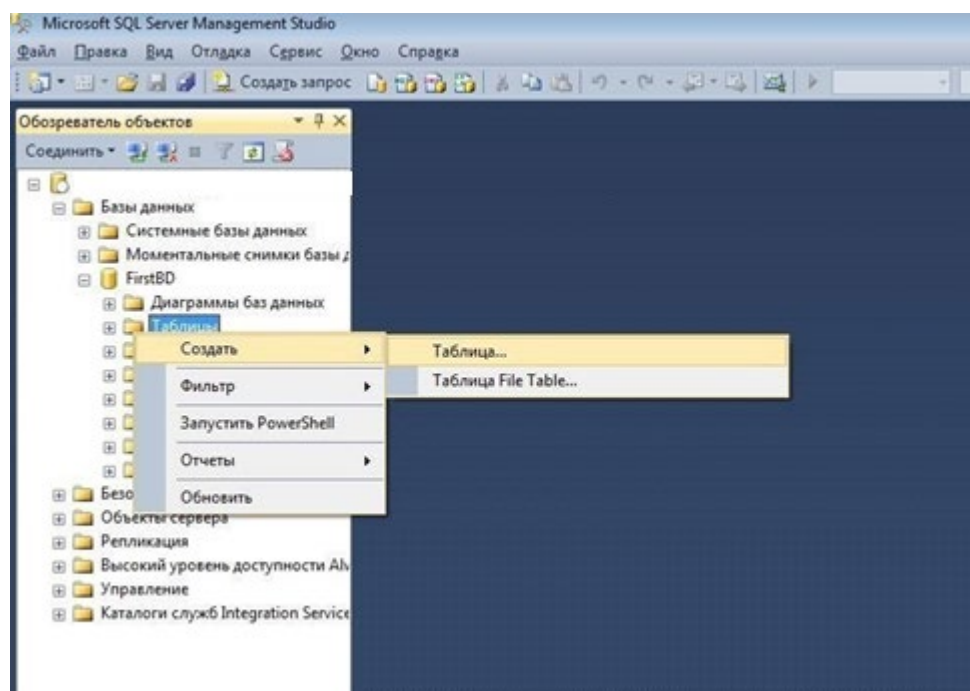


Рис. 1.8 – Обозреватель объектов

Создадим таблицу «Сотрудники» (из примера, рассматриваемого в лабораторной работе 1). Для этого необходимо кликнуть правой кнопкой мыши по папке «Таблицы» и в появившемся меню выбрать пункт «Создать таблицу». Появится окно создания новой таблицы (рисунок 1.9).

Имя столбца	Тип данных	Разрешит...
		<input type="checkbox"/>

Рис.1.9 – Создание новой таблицы

В правой части окна расположена таблица определения полей новой таблицы.

Данная таблица имеет следующие столбцы:

- Имя столбца должно всегда начинаться с буквы и не должно

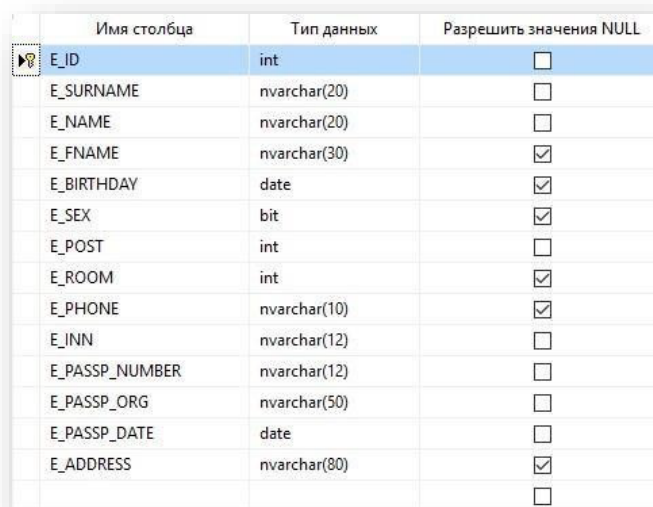
содержать различных специальных символов и знаков препинания. Если имя поля содержит пробелы, то оно автоматически заключается в квадратные скобки.

- Тип данных столбца.

- Разрешить значения Null. Если эта опция поля включена, то в случае не заполнения поля в него будет автоматически подставлено значение Null. То есть, поле необязательно для заполнения.

Под таблицей определения полей располагается таблица свойств выделенного поля «Свойства столбца». В данной таблице настраиваются свойства выделенного поля. Некоторые из них будут рассмотрены ниже.

Перейдём к созданию полей и настройке их свойств. В таблице определения полей задайте значения столбцов «Имя столбца», «Тип данных» и «Разрешить значения Null», как показано на рисунке 1.10.



Имя столбца	Тип данных	Разрешить значения NULL
E_ID	int	<input type="checkbox"/>
E_SURNAME	nvarchar(20)	<input type="checkbox"/>
E_NAME	nvarchar(20)	<input type="checkbox"/>
E_FNAME	nvarchar(30)	<input checked="" type="checkbox"/>
E_BIRTHDAY	date	<input checked="" type="checkbox"/>
E_SEX	bit	<input checked="" type="checkbox"/>
E_POST	int	<input type="checkbox"/>
E_ROOM	int	<input checked="" type="checkbox"/>
E_PHONE	nvarchar(10)	<input checked="" type="checkbox"/>
E_INN	nvarchar(12)	<input type="checkbox"/>
E_PASSP_NUMBER	nvarchar(12)	<input type="checkbox"/>
E_PASSP_ORG	nvarchar(50)	<input type="checkbox"/>
E_PASSP_DATE	date	<input type="checkbox"/>
E_ADDRESS	nvarchar(80)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Рис.1.10 – Пример создания таблицы «Сотрудники»

Из рисунка 1.10 следует, что таблица Сотрудники (Employees) имеет следующий набор столбцов:

-E_ID–идентификатор сотрудника(первичный ключ, уникальный);

- E_SURNAME– фамилия сотрудника (обязательное, строка 20символов);

- E_NAME – имя сотрудника (обязательное, строка 20 символов);

- E_FNAME – отчество (не обязательно, строка 30 символов);
- E_BIRTHDAY – дата рождения (дата);
- E_SEX – пол (логический тип данных);
- E_POST– идентификатор должности, служит для связи с таблицей «должности» (posts), в которой хранятся все должности (внешний ключ – связи между таблицами будут рассмотрены ниже);
- E_ROOM– номер комнаты (составной внешний ключ, служит для связи с таблицей «Комнаты»;
- E_TEL– номер телефона (составной внешний ключ, служит для связи с таблицей «Комнаты»;
- E_INN– идентификационный номер налогоплательщика;
- E_PASSP_NUMBER– номер паспорта (строка 12 символов, обязательное);
- E_PASSP_ORG – кем выдан паспорт(строка, 50 символов, обязательной);
- E_PASSP_DATE – дата выдачи паспорта (дата, обязательное);
- E_ADDRESS– адрес проживания (строка 80 символов).

Так как, поле E_ID будет являться первичным полем связи, то мы должны сделать его числовым счётчиком. То есть данное поле должно автоматически заполняться числовыми значениями. Более того, оно должно быть ключевым. Для этого выделите поле, просто щёлкнув по нему мышкой в таблице определения полей. В таблице свойств столбца отобразятся свойства поля E_ID. Разверните группу свойств *Спецификация идентификатора* Свойство (*Идентификатор*) установите в значение *Да*. Задайте свойства *Начальное значение идентификатора* и *Шаг приращения идентификатора* равными 1 (рисунок 1.11).

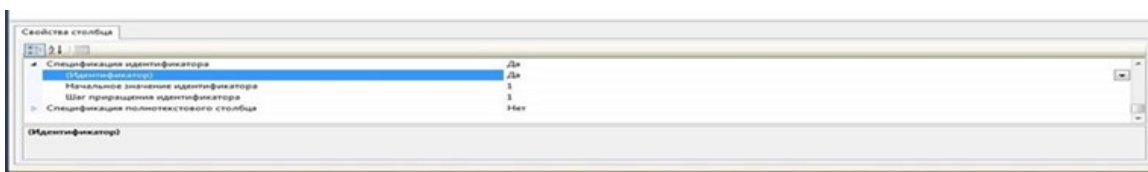





Рис. 1.11 – Свойства столбца E_ID

Эти настройки показывают, что значение поля E_ID у первой записи в таблице будет равным 1, у второй – 2, у третьей 3 и т.д.

 SID, говорящее о том, что поле ключевое.

Теперь сделаем поле E_ID ключевым полем. Выделите поле, а затем на панели инструментов нажмите кнопку с изображением ключа . В таблице определения полей, рядом с полем E_ID появится изображение ключа. На этом настройку таблицы «Сотрудники» можно считать на данном этапе завершённой.

Закройте окно создания новой таблицы, нажав кнопку закрытия  в верхнем правом углу окна, над таблицей определения полей.

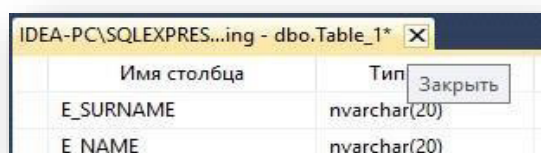


Рис. 1.12 – Закрытие настройки таблицы

Появится окно с запросом о сохранении таблицы. В этом окне необходимо нажать Да. Появится окно Выбор имени, предназначенное для определения имени новой таблицы.

В этом окне задайте имя новой таблицы как «employees» и нажмите кнопку ОК. Таблица «employees» отобразится в обозревателе объектов в папке Таблицы базы данных «publishing».

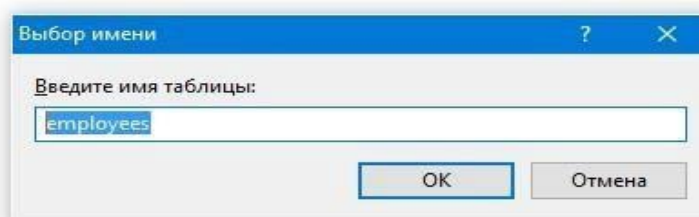


Рис. 1.13 – Ввод имени таблицы

В обозревателе объектов таблица «employees» отображается как dbo.employees. Префикс dbo обозначает, что таблица является объектом базы данных (DataBaseObject). В дальнейшем при работе с объектами базы данных префикс dbo можно опускать.

Лабораторная работа №2. Создание баз данных в MSSQLServer

Цель работы – с помощью операторов языка TransactSQL научиться создавать базы данных и совокупность связанных таблиц, принадлежащих указанной базе данных.

Задание:

Создать базу данных. В качестве примера базы данных, которая будет создана программно с помощью операторов языка TransactSQL, выберем БД «Книжное дело». Схема базы данных и структура таблиц данной БД представлена в конце лабораторной работы.

Краткие теоретические сведения

База данных представляет собой группу файлов, хранящихся на жестком диске. Эти файлы могут относиться к трем типам: файлы с первичными данными, файлы с вторичными данными и файлы журнала транзакций. Любая база данных SQLServer содержит, по крайней мере, два файла: первичный файл данных (с расширением .mdf) и файл журнала транзакций (с расширением .ldf). Существует два способа их создания:

- графически с помощью SQL Server Management Studio
- посредством кода Transact-SQL

Создание баз данных в SQL Server Management Studio

Использование данной утилиты является самым простым способом создания базы данных.

Создание баз данных с помощью Transact-SQL

Для программного создания базы данных (например, в программе установки приложения) используется инструкция `CREATEDATABASE` языка T-SQL (сокращенная форма от Transact-SQL). Данная инструкция может включать в себя множество опций, определяющих различные параметры новой базы данных.

Таблицы представляют собой объекты базы данных, используемые непосредственно для хранения всех данных. Одним из самых главных правил организации баз данных является то, что в одной таблице должны храниться данные лишь об одном конкретном типе сущности (например, клиенты, товары, заказы и т. п.).

Данные в таблицах организованы по полям и записям. Поля (или столбцы таблицы) содержат определенный тип информации, например, фамилию, адрес, телефонный номер. Запись (или строка таблицы) - группа

связанных полей, содержащих информацию об отдельном экземпляре сущности.

Любое поле таблицы характеризуется как минимум тремя обязательными свойствами:

- *Имя столбца*. Реализует способ обращения к конкретному полю в таблице. Рекомендуется всегда присваивать полям смысловые имена.
- *Тип данных*. Определяет, информация какого типа может храниться в данном поле.
- *Разрешить значения null*. Определяет, допустимо ли для данного поля отсутствие фактических данных, для обозначения которого используется так называемый маркер пустого значения null.

Порядок выполнения работы

1. Создать структуру новой базы данных.

Запустить **SQL Server Management Studio**.

Для написания программного кода в **SQL Server Management Studio** нужно нажать кнопку «Создать запрос» («New query») на панели инструментов «Стандартная» («Standart»).

Создать новую базу данных (на тему своего задания по примеру **DB_Books**) с названием **DB_Books** помощью команды:

```
CREATEDATABASEDB_BOOKS
```

Для выполнения команды нажать F5.

Открыть программу **SQL Server Management Studio**. Проверить наличие БД **DB_Books**, если ее не видите в разделе **DataBases**, то нажмите F5 для обновления.

2. Создать в базе данных перечисленные таблицы с помощью следующих команд (для создания новой страницы для кода в **SQL Server Management Studio** нажать кнопку «Создать запрос»):

```
use DB_BOOKS;
```

```
CREATE TABLE Authors(Code_author INT PRIMARY KEY,name_authorCHAR(30), Birthday DATETIME);
```

```
CREATE TABLE Publishing_house(Code_publish INT PRIMARY KEY, Publish CHAR(30), City CHAR(20));
```

```
CREATE TABLE Books(Code_book INT PRIMARY KEY, Title_bookCHAR(40), Code_author INT FOREIGN KEY REFERENCES
```

```
Authors(Code_author), Pages INT, Code_publish INT FOREIGN  
KEYREFERENCES Publishing_house(Code_publish));
```

```
CREATE TABLE Deliveries(Code_delivery INT PRIMARY  
KEY,Name_delivery CHAR(30), Name_company CHAR(20),  
AddressVARCHAR(100), Phone BIGINT, INN CHAR(13));
```

```
CREATE TABLE Purchases(Code_purchase INT PRIMARY  
KEY,Code_book INT FOREIGN KEY REFERENCES  
Books(Code_book),Date_order SMALLDATETIME, Code_delivery INT,  
Type_purchase BIT, Cost FLOAT, Amount INT FOREIGN KEYREFERENCES  
Deliveries(Code_delivery));
```

Запустите команду клавишей F5.

3.В утилите SQLServerManagementStudio проверить наличие БД DB_Books и таблиц в ней.

В разделе диаграмм создать новую диаграмму, в которую добавить из списка пять наших таблиц, проверить связи между таблицами (рис. 2.1).

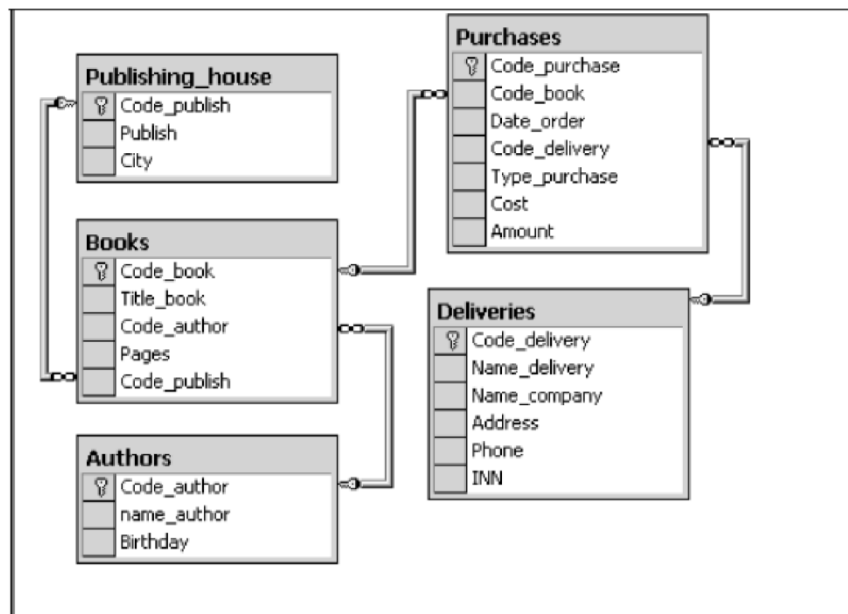


Рис. 2.1. Результат создания диаграммы

Удаление базы данных

Для удаления базы данных можно использовать один из трех способов:

1. Выполнить в программе " SQLServerManagementStudio " команду контекстного меню "Удалить", выбрав перед этим в списке базу данных, а затем подтвердить свое желание в диалоговом окне.
2. Выполнить оператор DROPDATABASE в SQL-редакторе.

3. Удалить файл с базой данных. Синтаксис оператора
DROPDATABASE:
DROPDATABASEdatabase_name;

Использованные операторы:

CREATETABLE – команда создания таблицы в текущей БД.

USE – сделать активной конкретную БД.

CREATEDATABASE – команда создания новой БД.

PRIMARYKEY – признак создания ключевого поля.

FOREIGNKEY...REFERENCES... – признак создания поля связи с другой таблицей.

3. Заполнение таблиц выполняется с помощью команды INSERT

Синтаксис команды выглядит следующим образом:

USE books;

INSERT INTO Authors (Code_author ,name_author, Birthday)

VALUES (001, ' Лев Николаевич Толстой', '28-08-1828'),

(002, ' Джордж Оруэлл', '25-07-1903'),

(003, ' Чарльз Диккенс', '07-02-1812');

Также заполнить таблицы в СУБД MSSQL следующим образом:
выбрать необходимую таблицу, правая кнопка/изменить первые 200 строк.

Варианты заданий к лабораторной работе №2

В утилите SQLServerManagementStudio создать новую базу данных с помощью оператора **CreateDatabase**, название БД определить, исходя из предметной области. Закомментировать оператор (-- – однострочный комментарий, /* */ – многострочный комментарий). Программно сделать активной созданную БД с помощью оператора **Use**. Создать перечисленные таблицы с помощью операторов **Createtable**, причем самостоятельно определить типы таблиц (родительская или подчиненная), типы полей и их размеры, найти поля типа Primarykey и Foreignkey. В SQLServerManagementStudio в разделе диаграмм созданной БД сгенерировать новую диаграмму, проверить связи между таблицами.

Требования к содержанию отчета:

Итоги лабораторной работы должны содержать БД по выбранному варианту, и как минимум по 10 записей в каждой из таблиц.

Варианты заданий к лабораторным работам №2

Вариант 1. БД «Учет выданных подарков несовершеннолетним детям сотрудников предприятия»

Код сотрудника	Код сотрудника	Код ребенка
Фамилия	Имя ребенка	Стоимость подарка
Имя	Дата рождения	Дата выдачи подарка
Отчество	Код ребенка	Код выдачи
Должность		
Подразделение		
Дата приема на работу		

Вариант 2. БД «Учет выполненных ремонтных работ»

Код прибора в ремонте	Код прибора	Код мастера
Название прибора	Код мастера	Фамилия мастера
Тип прибора	ФИО владельца прибора	Имя мастера
Дата производства	Дата приема в ремонт	Отчество мастера
	Вид поломки	Разряд мастера
	Стоимость ремонта	Дата приема на работу
	Код ремонта	

Вариант 3. БД «Продажа цветов»

Код цветка	Код цветка	Код продавца
Название цветка	Дата продажи	Фамилия
Сорт цветка	Цена продажи	Имя
Средняя высота	Код продавца	Отчество
Тип листа	Код продажи	Разряд
Цветущий		Оклад
Дополнительные сведения		Дата приема на работу

Вариант 4. БД «Поступление лекарственных средств»

Код лекарства	Код лекарства	Код поставщика
Название лекарства	Код поставщика	Сокращенное название
Показания к применению	Дата поставки	Полное название
Единица измерения	Цена за единицу	Юридический адрес
Количество в упаковке	Количество	Телефон
Название производителя	Код поступления	ФИО руководителя

Вариант 5. БД «Списание оборудования»

Код оборудования	Код оборудования	Код сотрудника
Название оборудования	Причина списания	Фамилия
Тип оборудования	Дата списания	Имя
Дата поступления	Код сотрудника	Отчество
ФИО ответственного	Код списания	Должность
Место установки		Подразделение
		Дата приема на работу

Вариант 6. БД «Поваренная книга»

Код блюда	Код блюда	Код продукта
Тип блюда	Код продукта	Название продукта
Вес блюда	Объем продукта	Ед измерения
Порядок приготовления		
Количество калорий		
Количество углеводов		

Вариант 7. БД «Регистрация входящей документации»

Код регистратора	Код документа	Код организации-отправителя
Фамилия	Номер документа	Сокращенное название
Имя	Дата регистрации	Полное название
Отчество	Краткое содержание документа	Юридический адрес
Должность	Тип документа	Телефон
Дата приема на работу	Код организации-отправителя	ФИО руководителя
	Код регистратора	

Вариант 8. БД «Увольнение сотрудника»

Код сотрудника	Код документа	Код статьи увольнения
Фамилия	Номер документа	Название статьи увольнения
Имя	Дата регистрации	Причина увольнения
Отчество	Дата увольнения	Номер статьи увольнения
Должность	Код статьи увольнения	Номер пункта/ подпункта увольнения
Подразделение	Код сотрудника	
Дата приема на работу	Денежная компенсация	

Вариант 9. БД «Приказ на отпуск»

Код сотрудника	Код документа	Код отпуска
Фамилия	Номер документа	Тип отпуска
Имя	Дата регистрации	Оплата отпуска
Отчество	Дата начала отпуска	Льготы по опуску
Должность	Дата окончания отпуска	
Подразделение	Код сотрудника	
Дата приема на работу	Код отпуска	

Вариант 10. БД «Регистрация выходящей документации»

Код отправителя	Код документа	Код организации-получателя
Фамилия	Номер документа	Сокращенное название
Имя	Дата регистрации	Полное название
Отчество	Краткое содержание документа	Юридический адрес
Должность	Тип документа	Телефон
Дата приема на работу	Код организации-получателя	ФИО руководителя
	Код отправителя	

ЛАБОРАТОРНАЯ РАБОТ 3.

МАНИПУЛИРОВАНИЕ ДАННЫМИ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА SQL. ОПЕРАТОР SELECT.

Цель работы: получить навыки формирования SQL запросов на добавление, изменение, извлечение и удаление данных на примере созданной согласно варианту базы данных.

ЗАДАНИЕ

1. Заполнить БД, созданную в Лабораторной работе №2 используя запросы
2. Создать запросы на извлечение данных

(Требования: запросы должны отражать потребности реальных пользователей, например, найти самую дорогую книгу, самую покупаемую вещь, определить наиболее частых клиентов и т.д.)

3. Создать подзапросы и вложенные запросы

(Требования: запросы должны отражать потребности реальных пользователей, например, найти самую дорогую книгу, самую покупаемую вещь, определить наиболее частых клиентов и т.д.)

ХОД РАБОТЫ

1. Создать базу данных используя мастер создания БД в SQL Server Management Studio согласно схеме, представленной на рисунке 3.1.

2. Написать SQL запросы на добавление данных в таблицы. Данные представлены на рисунках 3.2 – 3.5.

3. Изучить 28 примеров простых и вложенных запросов на извлечение данных из раздела 3.4 данного методического пособия.

Протоколирую результат выполнения запросов в отчет о проделанной работе.

4. Для своей схемы БД (созданной во второй лабораторной работы) написать различные виды запросов. Результаты выполнения представить в отчете.

5. Реализовать подзапросы и вложенные запросы.

6. Составить отчет о проделанной работе. Структура отчета:

- титульный лист: ФИО, группа;
- задание;
- описание хода выполнения работы (написать запросы и прикрепить скрины результатов работы по каждому запросу);
- заключение;

МАНИПУЛИРОВАНИЕ ДАННЫМИ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА SQL

Подготовка к выполнению лабораторной работы (Пример)

Перед тем как приступить к выполнению лабораторной работы номер 3 Вам необходимо создать базу данных используя мастер создания БД в MSSQLServer 20XX согласно схемам предыдущей лабораторной работы (для успешного выполнения запросов все таблицы БД которая выполнялась в предыдущей ЛР должны быть заполнены, желательно по 10 и более записей). Таблицы и значения, указанные в примерах заменить данными из своей БД.

Знакомство с оператором SELECT

Оператор SELECT извлекает строки из базы данных и позволяет делать выборку одной или нескольких строк, или столбцов из одной или нескольких таблиц в SQLServer. Полный синтаксис инструкции SELECT сложен, однако основные компоненты можно описать следующим образом:


```
SELECT column_list
FROM table_name
[WHERE условие]
[GROUP BY условие]
[HAVING условие]
[ORDER BY условие]
```

SELECT Ключевое слово, которое сообщает базе данных о том, что оператор является запросом. Все запросы начинаются с этого слова, за ним следует пробел.

Column_list Список столбцов таблицы, которые выбираются запросом. Столбцы, не указанные в операторе, не будут включены в результат. Если необходимо вывести данные всех столбцов, можно использовать сокращенную запись. Звездочка (*) означает полный список столбцов.

FROM table_name Ключевое слово, которое должно присутствовать в каждом запросе. После него через пробел указывается имя таблицы, являющейся источником данных.

Примеры запросов на извлечение данных

Ниже представлен перечень (24) простых запросов к БД (схема которой описана Выше. Данные примеры покрывают большой спектр конструкций языка SQL, начиная от простых запросов, кончая запросов с использованием функций и сортировок. Для выполнения лабораторной работы, Вам необходимо проделать все 24 запроса и привести результат выполнения в виде скриншота.

Пример 1. Вывод данных таблицы CATALOGS

```
SELECT ID_CATALOG, NAME FROM CATALOGS --
```

Здесь выводятся два поля *ID_CATALOG* и *NAME* из таблицы *CATALOGS*

```
SELECT * FROM CATALOGS --
```

Здесь выводятся все поля из таблицы *CATALOGS*

Пример 2. Вывод данных таблицы *CATALOGS* с присвоением псевдонима

```
SELECT
ID_CATALOG AS 'Идентификатор категории', --
-- команда AS изменяет имя столбца в
результатирующей таблице на имя, указанное после этой команды в
кавычках (ID_CATALOG изменяется на
'Идентификатор категории') NAME AS 'Имя категории'
FROM CATALOGS
```

Пример 3. Извлечение из таблицы *CATALOGS* записи, чей первичный ключ *ID_CATALOG* больше 2

```
SELECT * FROM CATALOGS
WHERE ID_CATALOG > 2 -- здесь WHERE
это условие. Т.е. мы указываем,
что нам нужно выбрать все данные из всех столбцов (*) таблицы
CATALOGS и вывести строки, где значение поля
ID_CATALOG будет больше 2
```

Пример 4. Составное условие: извлечение из таблицы *CATALOGS* записи, чей первичный ключ *ID_CATALOG* больше 2, но меньше или равен 4

```
SELECT * FROM CATALOGS
WHERE ID_CATALOG > 2 AND ID_CATALOG <= 4 --
AND позволяет создавать составные условия, т.е. после
```

WHERE мы можем указать несколько условий для выборки, разделяя их *AND*

```
SELECT * FROM CATALOGS  
WHERE ID_CATALOG BETWEEN 3 AND 7 --
```

Здесь записана упрощенная версия запроса выше. *BETWEEN* буквально означает «между», т.е. весь запрос можно воспринять так:

Выбрать все столбцы из таблицы *CATALOGS*, и вывести те строки, где значение поля *ID_CATALOG* находится между 3 и 7

Пример 5. Противоположная конструкция, которая выводит из таблицы *CATALOGS* записи, чей первичный ключ *ID_CATALOG* меньше 3, но больше 7.

```
SELECT * FROM CATALOGS  
WHERE ID_CATALOG NOT BETWEEN 3 AND 7 --
```

NOT перед *BETWEEN* означает, что по условию требуется чтобы выводились строки где *ID_CATALOG* вне указанного далее диапазона т.е. то, что находится между 3 и 7 не выводилось

Пример 6. Вывод записей, удовлетворяющих не диапазону, а списку

```
SELECT * FROM CATALOGS  
WHERE ID_CATALOG IN (1, 2, 5) --
```

IN указывает, что необходимо в результирующую таблицу поместить только те строки, значения поля *ID_CATALOG* которых равны значениям указанным в скобках после *IN*

Пример 7. Вывод записей, удовлетворяющих условию, заданному текстом:

вывести все записи, содержащие слово процессор

```
SELECT * FROM CATALOGS
```

```
WHERE NAME = 'процессоры' --
```

выводятся все строки, значения поля NAME в которых равно «процессоры».

Пример 8. Вывод записей, удовлетворяющих условию, заданному текстом:

вывести все записи, не содержащие слово процессор

```
SELECT * FROM CATALOGS
```

```
WHERE NOT NAME = 'процессоры' -- тоже, что и в
```

предыдущем примере, только выводится все, кроме «процессоры»

Пример 9. Вывод записей, удовлетворяющих условию, заданному частью текста

```
SELECT * FROM USERS
```

```
WHERE SURNAME LIKE 'И%' -- LIKE
```

дает возможность указать какую-либо часть значения. В

данном случае мы получаем все строки, значения поля SURNAME

которых будет начинаться с буквы И

Пример 10. Работа с датой: извлечение из таблицы ORDERS записи, соответствующие сделкам, осуществленным за февраль 2005 г.

```
SELECT * FROM orders
```

```
WHERE ORDERTIME >= '2005-02-01' AND ORDERTIME <  
'2005-03-01';
```

Пример 11. Сортировка по значению одного из столбцов

```
SELECT * FROM CATALOGS
ORDER BY ID_CATALOG --
```

сортируем все строки по полю ID_CATALOG (по возрасту)

```
SELECT * FROM CATALOGS
ORDER BY NAME --
```

сортируем все строки по полю NAME

Пример 12. Извлечение из таблицы PRODUCTS записи товаров, количество которых COUNT на складе от 4 до 8 с сортировкой по полю COUNT и полю MARK

(для краткости выведем только столбцы COUNT и MARK)

```
SELECT COUNT, MARK FROM PRODUCTS
WHERE COUNT BETWEEN 4 AND 8 ORDER BY COUNT, MARK --
```

выбираем строки, значение поля COUNT которого находятся между 4 и 8, затем сортируем полученные строки сначала по COUNT затем по MARK

Пример 13. Изменение порядка сортировки (по умолчанию, сортировка производится в прямом порядке (ASC))

```
SELECT ORDERTIME FROM ORDERS
ORDER BY ORDERTIME DESC --
```

DESC означает сортировка по убыванию

Пример 14. Извлечение первых пяти записей с обратной сортировкой по полю COUNT

```
SELECT TOP 5 ID_PRODUCT, COUNT FROM PRODUCTS
ORDER BY COUNT DESC
```

Пример 15. Подсчет количества проданных ТОВАРОВ

```
SELECT SUM (NUMBER) AS 'Всего продано' -- SUM означает
сумму значений поля NUMBER
```

```
FROMORDERS
```

Пример 16. Подсчет среднего количества товаров в одном заказе

```
SELECTAVG (NUMBER) AS 'Среднее количество' -- AVG  
означает среднее значение поля NUMBER  
FROMORDERS
```

Пример 17. Подсчет числа строк в таблице, значения столбца которых отличны от NULL

```
SELECTCOUNT (ID_ORDER) -- COUNT выводит количество  
строк поля ID_ORDER значения которого не равны NULL  
FROMORDERS
```

Пример 18. Подсчет числа строк в таблице, значения столбца которых отличны от NULL с присвоением псевдонима

```
SELECT COUNT (ID_ORDER) AS TOTAL  
FROMORDERS
```

Пример 19. Извлечение максимального значения столбца ID_CATALOG

```
SELECTMAX (ID_CATALOG) -- MAX позволяет  
извлечь максимальное значение поля из столбца  
ID_CATALOG
```

```
FROMCATALOGS
```

```
SELECTTOP1 * FROMCATALOGS-- TOP 1  
означает вывод первой строки из полученной  
выборки. После TOP может стоять любое число,  
соответственно число строк будет другое
```

```
ORDER BY ID_CATALOG
```

Пример 20. Извлечение минимального значения столбца ID_CATALOG

```
SELECTMIN (ID_CATALOG)
```

```
FROM CATALOGS
SELECT TOP 1 * FROM CATALOGS
ORDER BY ID_CATALOG DESC
```

Пример 21. Вывод числа уникальных значений ID_CATALOG

(сравните результат с SELECT COUNT (ID_CATALOG) FROM PRODUCTS)

```
SELECT COUNT (DISTINCT ID_CATALOG) --
DISTINCT
означает вывод только уникальных значений этого пол
я
FROM PRODUCTS
```

Пример 22. Вывод числа записей, соответствующих каждому из уникальных значений ID_CATALOG

```
SELECT ID_CATALOG, COUNT (ID_CATALOG)
FROM PRODUCTS
GROUP BY ID_CATALOG ORDER BY ID_CATALOG
```

Пример 23. Вывод числа записей, соответствующих каждому из уникальных значений ID_CATALOG больше двух

```
SELECT ID_CATALOG, COUNT (ID_CATALOG)
FROM PRODUCTS
WHERE ID_CATALOG > 2
GROUP BY ID_CATALOG -- группировка записи по
какомулибо полю, оставляя при этом только одну запись с
каждым значением
ORDER BY ID_CATALOG
```

Пример 24. Выбрать категории товаров, для которых добавлено более пяти товаров (ограничение выборки по результатам функции)

```

SELECT ID_CATALOG, COUNT(ID_CATALOG) AS TOTAL
FROM PRODUCTS
GROUP BY ID_CATALOG
HAVING ID_CATALOG > 5
ORDER BY
ID_CATALOG

```

Примечание: HAVING аналогичен WHERE за исключением того, что строки отбираются не по значениям столбцов, а строятся из значений столбцов, указанных в GROUP BY, и значений агрегатных функций, вычисленных для каждой группы, образованной GROUP BY.

Подзапросы и вложенные запросы

Вложенные запросы — это запросы, которые расположены внутри других запросов, таких как SELECT, INSERT, UPDATE или DELETE. К тому же, подзапросы MSSQL могут быть расположены внутри других подзапросов.

Подзапрос MSSQL в условии WHERE

Вы можете использовать операторы сравнения =, >, < и т.д., чтобы сравнить одно значение возвращенное подзапросом с выражением в условии WHERE.

Пример 25. Добавим вложенный запрос для получения самого дорогого товара из таблицы PRODUCTS.

```

SELECT NAME,
PRICE,
COUNT
FROM PRODUCTS
WHERE PRICE = (
SELECT MAX(PRICE)

```



```
FROM PRODUCTS
```

```
);
```

Пояснение: Вложенные запросы следует рассматривать снизу вверх, т.е. здесь мы сначала получаем строку с максимальным значением поля PRICE из таблицы PRODUCTS, затем делаем выборку, где выводим строки с полями NAME, PRICE, COUNT где во вложенной выборке мы нашли максимальный PRICE

Вложенный запрос с операторами IN и NOTIN

Если подзапрос возвращает более одного значения, вы можете использовать операторы IN и NOTIN в условии WHERE.

Пример 26. Используя оператор NOTIN, найдем пользователей, которые не заказали ни одного товара.

```
SELECT NAME, SURNAME
FROM USERS
WHERE ID_USER NOT IN (
SELECT DISTINCT ID_USER
FROM ORDERS
);
```

Вложенный запрос с EXISTS И NOT EXISTS

Когда подзапрос используется с операторами EXISTS или NOTEXISTS, то такой подзапрос возвращает булево значение: TRUE или FALSE. В данном случае вложенный запрос действует как проверка на существование.

Пример 27. Найдем всех пользователей, которые купили товары с ценой больше 1000.

```
SELECT NAME, SURNAME
FROM USERS
WHERE EXISTS
```

```

        (SELECT NAME
        FROM PRODUCTS, ORDERS, USERS
        WHERE ID_PRODUCT =
        ORDERS.ID_PRODUCT          AND
        ORDERS.ID_USER = USERS.ID_USER
        AND PRICE > 1000)

```

Примечание: Конструкция WHERE<имя_столбца> = <имя_столбца> для соединения таблиц является старым стилем и в настоящее время заменяется конструкцией с оператором JOIN, речь о котором пойдет в следующей лабораторной работе.

Подзапрос MSSQL в условии FROM

Когда вы используете вложенный запрос в условии FROM, то результат возвращенный этим подзапросом будет таблицей, которая называется производной.

Пример 28. Найдем максимальное, минимальное и среднее количество элементов в заказе.

```

SELECT
MAX(items),
MIN(items),
FLOOR(AVG(items))
FROM
(
SELECT
        NUMBER,
COUNT(NUMBER) AS items
FROM
ORDERS
GROUP BY

```

NUMBER

) ASLINEITEMS;

ЛАБОРАТОРНАЯ РАБОТА № 4. СОЗДАНИЕ И УПРАВЛЕНИЕ ПРЕДСТАВЛЕНИЯМИ

Цель работы

Изучение назначения представлений баз данных, синтаксиса и семантики команд языка Transact-SQL для их создания, изменения и удаления, системных хранимых процедур для получения информации о представлениях, а также приобретение навыков их создания с помощью графических средств утилиты EnterpriseManager и мастера CreateViewWizard.

Методические рекомендации для выполнения практической работы

Представление (View) для пользователей баз данных выглядит как таблица, но при этом оно не содержит данных, а лишь представляет данные, расположенные в одной или нескольких таблицах. Таким образом, представления – это виртуальные таблицы, определяемые запросом на языке Transact-SQL. Подобно реальным таблицам представления содержат именованные столбцы и строки с данными, которые они динамически выбирают из таблиц и предлагают эти данные пользователю для просмотра. Представления часто применяются для ограничения доступа к конфиденциальным данным в таблицах баз данных. Когда в представление не включается столбец исходной таблицы, то считают, что на таблицу наложен вертикальный фильтр. Если в SQL – запросе установлено одно или несколько условий для выборки строк, то считают, что на таблицу наложен горизонтальный фильтр.

Представление может выбирать данные из других представлений, которые, в свою очередь, могут также основываться на представлениях или таблицах. Вложенность представлений не должна превышать 32. Представления можно создавать, используя базы данных одного сервера (текущего). Максимальное количество столбцов в представлении равно 1024.

Представление не может ссылаться на временные таблицы. Кроме того, нельзя создавать временное представление.

Для представления нельзя определить ограничения целостности, триггеры, правила, или умолчания, а также создать обычный или полнотекстовый индекс.

В основном представления используются для выборки данных. Однако с помощью представлений можно выполнять и изменение данных в таблицах, на основе которых построено представление, при этом требуется соблюдение ряда правил: представление должно содержать, как минимум, одну таблицу в параметре FROM команды SELECT, не разрешается использование функций агрегирования и др. Как и для таблиц, для представлений можно определить следующие права доступа:

SELECT – просмотр данных;

INSERT – добавление данных через представления;

UPDATE – изменение данных в исходных таблицах; **DELETE** –удаление данных в исходных таблицах.

Чтобы иметь возможность создавать представления, надо обладать правами владельца баз данных и иметь соответствующие разрешения для любых таблиц или представлений, упомянутых "в запросе на создание этого представления.

Для создания представления используется следующая команда Transact-SQL:

CREATEVIEW [Имя базы данных.] [имя владельца.]

Имя представления [(Имя колонки [...n])]

[WITH{ENCRYPTION\SCHEMABINDING

VIEW_METADATA}

AS Команда SELECT

[WITH CHECK OPTION]

Если в команде не заданы имена колонок представления, то они определяются по именам выбираемых колонок в команде SELECT. Параметр ENCRYPTION скрывает код создания этого представления, а параметр SHEMABINDING обеспечивает контроль структуры исходных объектов, к которым обращается оператор SELECT. Опция WITHCHECKOPTION не позволяет изменять строки таким образом, чтобы они исчезли при отборе командой SELECT.

Задания для выполнения практической работы

Задание 1. Создать представление auth, ссылающегося на таблицу authors базы данных Pubs и содержащего идентификационный номер автора au_lname и телефон phone, при этом отобразить только авторов из Калифорнии 'CA' или авторов, не подписавших контракт с издательством, выполнив следующую команду:

```
CREATE VIEW auth
WITH SHEMABINDING
AS SELECT au_id, au_lname, au_fname, phone
FROM dbo. Authors
WHERE state = 'CA' OR contract = 0 WITH CHECK OPTION.
```

Задание 2. Создать представление report, которое ссылается на представление auth и таблицы titleauthor и titles и в котором выводятся имя автора au_fname, фамилия автора au_lname и сокращенные названия написанных им книг, выполнив команду:

```
CREATE VIEW report
AS SELECT [Фамилия] = CAST (au_lnameaschar(10)),
[Имя] = CAST(au_fnameaschar(10)),
[Названиекниги] =
CAST (title as char(30)) +
```

```
CASE WHEN LEN (title) >30 THEN '...' END  
FROM auth a, titleauthor ta, titles t WHERE ta.au_id = a.au_id AND  
t.title_id = ta .title_id.
```

Задание 3. Создать представление auth, рассмотренное в первом задании, с помощью графических средств утилиты EnterpriseManager.

Задание 4. Создать представление report, рассмотренное во втором задании, с помощью мастера CreateViewWizard.

Задание 5. Сопоставить запросы, полученные автоматически в заданиях 3и 4, с запросами соответственно в первом и втором заданиях. Модифицировать запросы с помощью команды ALTERVIEW и получить справочную информацию об этих представлениях с помощью процедур sp_help, sp_helptext и sp_depends.

Лабораторная работа №5 Освоение программирования с помощью встроенного языка TransactSQL в MSSQLServer

Цель работы – знакомство с основными принципами программирования в MSSQLServer средствами встроенного языка TransactSQL.

Задачи работы:

1. Знакомство с правилами обозначения синтаксиса команд в справочной системе MSSQLServer (утилита BooksOnline).
2. Изучение правил написания программ на TransactSQL.
3. Изучение правил построения идентификаторов, правил объявления переменных и их типов.
4. Изучение работы с циклами и ветвлениями.
5. Изучение работы с переменными типа Table и Cursor.
6. Проработка всех примеров, анализ результатов их выполнения.
7. Выполнение индивидуальных заданий по вариантам.

Обеспечивающие средства: персональный компьютер с установленным на нем программным обеспечением- MSSQLServer 2012

Задание: освоить основные принципы программирования в MSSQLServer. Для этого используем пример базы данных с названием **DB_Books**, которая была создана в лабораторной работе №1. При выполнении примеров и заданий обращайте внимание на соответствие названий БД, таблиц и других объектов проекта.

Краткие теоретические сведения

Для овладения начальными навыками программирования познакомимся с правилами обозначения синтаксиса команд в справочной системе MSSQLServer. Ниже представлен список специальных знаков и их назначение. В табл. 2.1 представлен список специальных знаков и их назначение.

Идентификаторы – это имена объектов, на которые можно ссылаться в программе, написанной на языке TransactSQL. Первый символ может состоять из букв английского алфавита или “_”, “@”, “#”. Остальные дополнительно из цифр и «\$».

Имя идентификатора не должно совпадать с зарезервированным словом.

Для ограничителей идентификаторов при установленном параметре SETQUOTED_IDENTIFIER можно использовать как квадратные скобки, так и одинарные кавычки, а строковые значения только в одинарных кавычках (режим по умолчанию).

Если использовать установленный параметр в режиме SETQUOTED_IDENTIFIER OFF, то в качестве ограничителей идентификаторов можно использовать только квадратные скобки, а строковые значения указываются в одинарных или двойных кавычках.

Таблица

5.1 Список специальных знаков и их назначение в TransactSQL

Специальные знаки и простейшие операторы в Transact SQL

Знак	Назначение	Знак	Назначение
*	Знак умножения	" "	В них заключают строковые значения, если SET QUOTED_IDENTIFIER OFF
-	Знак вычитания	' '	В них заключают строковые значения
%	Остаток от деления двух чисел	<>	Не равно
+	Знак сложения или конкатенации (объединение двух строк в одну)	[]	Аналог кавычек, в них можно заключать названия идентификаторов, если в их названиях встречаются пробелы
=	Знак равенства или сравнения	!<	Не менее чем
<=	Меньше или равно	!>	Не более чем
>=	Больше или равно	>	Больше
!=	Не равно	<	Меньше
@	Ставится перед именем переменной	.	Разделяет родительские и подчиненные объекты
@@	Указывает на системные функции	/	Знак деления
--	Однострочный комментарий или комментарий с текущей позиции и до конца строки	/* */	Многострочный комментарий

Переменные используются для сохранения промежуточных данных в хранимых процедурах и функциях. Все переменные считаются локальными.

Имя переменной должно начинаться с @.

Объявление переменных Синтаксис в обозначениях MSSQLServer:

```
DECLARE @имя_переменной1 тип_переменной, ...,
@имя_переменнойNтип_переменной
```

Если тип переменной предполагает указание размера, то используется следующий синтаксис для объявления переменных:

```
DECLARE @имя_переменной1 тип_переменной (размер), ...,
@имя_переменнойNтип_переменной(размер)
```

Пример:

```
DECLARE @a INT, @b numeric(10,2)
DECLARE @strCHAR(20)
```

Присвоение значений переменным и вывод значений на экран

Присвоение с помощью **SET**– обычное присвоение, синтаксис:

```
SET @имя_переменной = значение.
```

Пример:

```
DECLARE @aINT, @bnumeric(10,2)
SET @a = 20
SET @b = (@a+@a)/15
SELECT @b --вывод на экран результата
```

Присвоение с помощью **SELECT**– помещение результата запроса в переменную. Если в результате выполнения запроса не будет возвращено ни одной строки, то значение переменной не меняется, т.е. остается старым.

Пример:

```
DECLARE @a INT
SELECT @a = COUNT(*) FROM Authors
```

Пример:

```
DECLARE @strCHAR(30)
SELECT @str = name FROM Authors
```

В данном примере в переменную поместится последнее значение из результата запроса.

Сочетание ключевых слов SET и SELECT

Пример:

```
DECLARE @a INT
SET @a = (SELECT COUNT(*) FROM Authors)
```

Работа с датой и временем

Оператор SETDATEFORMATdmy | ymd | mdy задает порядок следования компонентов даты.

Пример:

```
SET DATEFORMAT dmy
DECLARE @d DateTime
```

```
SET @d = '31.01.2005 13:23:15'  
SET @d = @d+1  
SELECT @d
```

Создание временной таблицы через переменную типа TABLE

Объявляется через DECLARE с указанием в скобках столбцов таблицы, их типов, размеров, значений по умолчанию, а также индексов типа PRIMARY KEY или UNIQUE.

Пример:

```
DECLARE @mytable TABLE(id INT, myname CHAR(20) DEFAULT  
'Введите имя')  
INSERT INTO @mytable(id) VALUES (1) SELECT * FROM @mytable
```

Пример:

```
DECLARE @mytable TABLE(id INT, myname CHAR(20) DEFAULT  
'Введите имя')  
INSERT @mytable SELECT Code_publish, City FROM  
Publishing_house  
SELECT * FROM @mytable
```

Преобразование типов переменных

Функция CAST возвращает значение, преобразованное к указанному типу:
CAST(@переменная или значение AS требуемый_тип_данных)

Пример:

```
DECLARE @d DateTime, @strchar(20)  
SET @d = '31.01.2005 13:23:15'  
SET @str = CAST(@d AS Char(20))  
SELECT 2str
```

Функция CONVERT возвращает значение, преобразованное к указанному типу по заданному формату. Изучить дополнительно, по желанию.

Операторские скобки BEGIN

/* в них нельзя помещать команды, изменяющие структуры объектов БД. Операторские скобки должны содержать хотя бы один оператор. Требуются для конструкций поливариантных ветвлений, условных и циклических конструкций */

END

Условная конструкция IF Синтаксис:

IF условие

Набор операторов1 **ELSE**

Набор операторов2

Пример:

```
DECLARE @a INT
DECLARE @strCHAR(30)
SET @a = (SELECT COUNT(*) FROM Authors) IF @a >10
BEGIN
    SET @str = 'Количество авторов больше 10' SELECT @str
END
ELSE
BEGIN
    SET @str = 'Количествоавторов = ' + str(@a) SELECT @strEND
```

Цикл WHILE

Синтаксис:

```
WHILE Условие Набор операторов1
```

```
BREAK
```

Набор операторов2 CONTINUE Конструкции BREAK и CONTINUE являются необязательными.

Цикл можно принудительно остановить, если в его теле выполнить команду BREAK. Если же нужно начать цикл заново, не дожидаясь выполнения всех команд в теле, необходимо выполнить команду CONTINUE.

Пример:

```
DECLARE @a INT
SET @a = 1
WHILE @a < 100
BEGIN
    PRINT @a -- вывод на экран значения переменной
    IF (@a > 40) AND (@a < 50)
        BREAK --выход и выполнение 1-й команды за циклом
    ELSE
        SET @a = @a + rand()*10
    CONTINUE
END
PRINT @a
```

Лабораторная работа № 6 Создание хранимых процедур в MSSQLServer

Цель работы – научиться создавать и использовать хранимые процедуры на сервере БД. *Задачи работы*:

1. Проработка всех примеров, анализ результатов их выполнения в утилите ManagementStudio. Проверка наличия созданных процедур в текущей БД.
2. Выполнение всех примеров и заданий по ходу лабораторной работы.
3. Выполнение индивидуальных заданий по вариантам.

Обеспечивающие средства: персональный компьютер с установленным на нем программным обеспечением - MSSQLServer 2012

Задание: освоить процедуру создания хранимых процедур

Краткие теоретические сведения

Хранимые процедуры представляют собой набор команд, состоящий из одного или нескольких операторов SQL или функций и сохраняемый в базе данных в откомпилированном виде.

Процедуры и функции - это объекты базы данных и, следовательно, они создаются командой CREATE и уничтожаются командой DROP. При создании процедуры и функции должны быть определены: имя объекта, перечень и типы параметров и логика работы процедуры или функции, описанные на языке PL/SQL

Чтобы создать процедуру или функцию, необходимо иметь системные привилегии CREATEPROCEDURE. Для создания процедуры, функции или пакета в схеме, отличной от текущей схемы пользователя, требуются системные привилегии на CREATEANYPROCEDURE. После определения имени новой процедуры или функции необходимо задать ее имена, типы и виды параметров. Для каждого параметра должен быть указан один из видов параметра - IN, OUT или INOUT. Вид параметра IN предполагает, что значение параметра должно быть определено при обращении к процедуре и не изменяется процедурой.

Порядок выполнения работы

1. Создадим процедуру без параметров:

```
CREATE PROCEDURE Count_Books AS Select count(Code_book) from
Books Go
```

2. Создадим процедуру с входным параметром:

```
CREATE PROCEDURE Count_Books_Pages @Count_pages as Int
AS Select count(Code_book) from Books WHERE
Pages>=@Count_pages Go
```

3. Создадим процедуру с входными параметрами:

```
CREATE PROCEDURE Count_Books_Title @Count_pages as Int, @Title
AS Char(10) AS Select count(Code_book) from Books WHERE
Pages>=@Count_pages
ANDTitle_book LIKE @Title
Go
```

4. Создадим процедуру с входными параметрами и выходным параметром:

```
CREATE PROCEDURE Count_Books_Itogo @Count_pagesInt, @Title
Char(10) , @ItogoInt OUTPUT AS Select @Itogo = count(Code_book) from
Books WHERE Pages>=@Count_pages AND Title_book LIKE @Title
Go
```

5. Создадим процедуру с входными параметрами и RETURN:

```
CREATEPROCEDUREcheckname @paramint
AS IF (SELECT Name_author FROM authors WHERE Code_author =
@param) = 'ПушкинА.С.' RETURN 1
ELSE
RETURN 2
```

6. Создадим процедуру без параметров для увеличения значения ключевого поля в таблице Purchases в 2 раза:

```
CREATE PROC update_proc
AS UPDATE Purchases SET Code_purchase = Code_purchase*2
```

Процедура не возвращает никаких данных.

7. Создадим процедуру с входным параметром для получения всей информации о конкретном авторе:

```
CREATE PROC select_author @k CHAR(30)
AS SELECT * FROM Authors WHERE name_author=@k
```

8. Создадим процедуру с входным параметром и значением по умолчанию для увеличения значения ключевого поля в таблице Purchases в заданное количество раз (по умолчанию в 2 раза):

```
CREATE PROC update_proc @p INT = 2
```

```
AS UPDATE Purchases SET Code_purchase = Code_purchase * @p
```

Процедура не возвращает никаких данных.

9. Создадим процедуру для определения количества заказов, совершенных за указанный период. Это процедура с входным и выходным параметрами

```
CREATE PROC count_purchases
```

```
@d1 SMALLDATETIME, @d2 SMALLDATETIME,
```

```
@c INT OUTPUT
```

```
AS SELECT @c=count(Code_purchase) from Purchases WHERE  
Date_order BETWEEN @d1 AND @d2 SET @c = ISNULL(@c,0)
```

Варианты заданий к лабораторной работе №3

Общие положения

В утилите SQLServerManagementStudio создать новую страницу для кода (кнопка «Создать запрос»). Программно сделать активной созданную БД DB_Books с помощью оператора **Use**. Создать хранимые процедуры с помощью операторов **Createprocedure**, причем самостоятельно определить имена процедур. Каждая процедура будет выполнять по одному SQL запросу, которые были выполнены во второй лабораторной работе. Причем код SQL запросов нужно изменить таким образом, чтобы в них можно было передавать значения полей, по которым осуществляется поиск.

Например, исходное задание и запрос в лабораторной работе №3:

```
/*Выбрать из справочника поставщиков (таблица Deliveries) названия  
компаний, телефоны и ИНН (поля Name_company, Phone и INN), у которых  
название компании (поле Name_company) 'ОАО МИР'. SELECT  
Name_company, Phone, INN FROM Deliveries
```

```
WHERE Name_company = 'ОАОМИР'*/
```

--В данной работе будет создана процедура:

```
CREATE PROC select_name_company @comp CHAR(30)
```

```
AS
```

```
SELECT Name_company, Phone, INN FROM Deliveries
```

```
WHERE Name_company = @comp
```

--Для запуска процедуры используется команда:

EXEC

```
select_name_company 'ОАОМИР'
```

Сохранить файл программы с названием *ФамилияСтудента_ЛАб_3*.

В SQLServerManagementStudio в разделе хранимых процедур БД DB_Books проверить наличие процедур.

Список заданий

В утилите SQLServerManagementStudio создать новую программу.

Программно сделать активной индивидуальную БД, созданную в лабораторной работе №1, с помощью оператора Use. Создать хранимые процедуры с помощью операторов Createprocedure, причем самостоятельно определить имена процедур. В SQLServerManagementStudio в разделе хранимых процедур индивидуальной БД проверить наличие процедур. Варианты заданий для данной работы приведены в Приложении 3.

Требования к содержанию отчета: итоги лабораторной работы представить в виде пояснений и копий экранов выполнения основных заданий, описанных в инструкции по выполнению лабораторной работы, предоставить файл программы индивидуального задания, с названием Фамилия Студента_Лаб_3_№варианта..

Контрольные вопросы для самоподготовки

1. Что такое хранимая процедура?
2. Для чего используется хранимая процедура?
3. Сколько хранимых процедур можно создать у одной таблицы?
4. Какие существуют виды хранимых процедур?
5. Какие уровни языка transact-SQL используются при создании и использовании хранимых процедур?

ЛАБОРАТОРНАЯ РАБОТА №7. СОЗДАНИЕ И УПРАВЛЕНИЕ ФУНКЦИЯМИ

Функции

Функции (хранимые процедуры) используются в SQL Server для реализации на языке Transact-SQL сложных часто используемых алгоритмов обработки данных или различных административных действий создания учетных записей, получения информации об объектах базы данных, управления свойствами сервера и баз данных, управления подсистемой репликации автоматизации и т.д.

Они хранятся в виде исходного текста и являются программными модулями, существующими независимо от таблиц или каких-либо других объектов баз данных.

Исключением являются **расширенные хранимые процедуры**, которые хранятся в двоичном формате в виде динамически подключаемых библиотек типа ***.dll** и создаются с помощью других языков программирования с использованием интерфейса **SQL Server Open Data Services API**. Такие процедуры подключаются, отключаются и выгружаются соответственно командами **sp_addextendedproc**, **sp_dropextendedproc** и **DBCC dlname (FREE)**, где **dllname_имя dll_библиотеки**.

Хранение функций и хранимых процедур в виде исходных модулей языка Transact – SQL на сервере и в соответствующих базах данных **позволяет уменьшить размер запроса**, посылаемого посетителем от клиента на сервер, а, следовательно, **инагрузку на сеть**, что повышает общую производительность системы. Это также позволяет **упростить сопровождение программных комплексов** и внесение изменений в исходный текст модулей, причем большинство изменений не отразится на работоспособности клиентских приложений.

Значительная часть функций и хранимых процедур поставляется в составе SQL Server. Они называются **системными**, или **встроенными (built-in)**.

Кроме того, пользователю предоставляется возможность разрабатывать и включать в свою базу данных **собственные**, или **пользовательские (user-defined) функции и хранимые процедуры**, реализующие специальные алгоритмы обработки данных.

Таким образом, пользовательские функции и хранимые процедуры становятся объектами той базы данных, в которой они создавались. Поэтому при их соз

дании, если необходимо, требуемую базу данных следует сделать текущей с помощью команды **USE** имя базы данных. Системные функции хранятся на экземпляре сервера, а системные хранимые процедуры – в базе **MASTER** этого же экземпляра сервера.

В SQL Server можно создавать и так называемые **временные хранимые процедуры** в базе данных **tempdb** экземпляра сервера, которые существуют лишь некоторое время, после чего автоматически уничтожаются сервером. Они бывают **локальными** и **глобальными**.

Функции и хранимые процедуры могут быть вызваны клиентскими программами, другими функциями или хранимыми процедурами, а также триггерами.

В любом случае необходимо указать имя функции или хранимой процедуры и список аргументов, которые сопоставляются параметрам соответствующей функции или хранимой процедуры при этом тип аргументов и параметров должны совпадать или допускать автоматические преобразования типов. Если для некоторого параметра задано значение по умолчанию и это значение подходит для данного вызова, то соответствующий аргумент может быть опущен.

Поскольку функция возвращает значение, она используется в качестве операнда некоторого выражения в виде вызова функций, состоящего из имени этой функции и списка аргументов, заключенного в круглые скобки, при этом в качестве аргументов могут быть любые выражения языка Transact-SQL, дающие в результате значения требуемых типов.

Аргументы в вызове функции отделяются запятыми.

Если список аргументов пуст, то круглые скобки после имени функции, как правило, задаются.

Исключения составляют некоторые системные функции, для которых круглые скобки не задаются, когда нет аргументов.

Хранимые процедуры могут вызываться только командой **EXECUTE**, или сокращенно **EXEC**. За этой командой должны быть указаны имя процедуры и через пробел список аргументов, если вызывается процедура с параметрами. Аргументы разделяются запятой. Если для параметра задано значение по умолчанию, то аргумент либо совсем не задается (в конце списка), либо используется слово **DEFAULT** (в середине списка).

Процедура может возвращать результаты только через параметры с ключевым словом **OUTPUT**, при этом и аргумент должен быть задан с таким же ключевым словом.

Создание, изменение и удаление функций хранимых процедур производится соответственно командами:

- **CREATEFUNCTION**,
- **CREATEPROCEDURE** ,
- **ALTERFUNCTION**,
- **ALTERPROCEDURE**,
- **DROPFUNCTION**,
- **DROPPROCEDURE**.

При создании функции указывается тип возвращаемого значения и теле функции обязательно задается команда

RETURN, за которой следует выражения для вычисления возвращаемого значения.

В теле процедуры использование команды **RETURN** (конечно, без последующего выражения) обязательно. Когда этой команды нет, выход из процедуры будет происходить после исполнения последней команды процедуры.

Тело, как функции, так и хранимой процедуры начинается ключевым словом **AS**.

Поскольку каждая из них храниться как отдельный объект, то для указания концателане требуется записывать какое-либо специальное ключевое слово или знак.

За командами создания функции или хранимой процедуры перечисляются имена параметров, начинающиеся с символа **@**, и их типы, а также важно значение по умолчанию.

Для функции этот список заключается в круглые скобки, после которых записывается ключевое слово **RETURNS** (возвращает) и тип возвращаемого значения.

Для хранимой процедуры круглые скобки не используются, и задавать тип возвращаемого значения не требуется.

Для тела функции часто используют ключевое слово **begin** после ключевого слова **as** и ключевое слово **end** в конце тела.

Дополнительные опции функции или хранимой процедуры задаются ключевым словом **with** до началателя.

В SQL Server 2003 можно создавать **функции трех классов**:

- **Scalar** – возвращают обычное скалярное значение;
- **Inline** – возвращают таблицу динамической структуры, создаваемую единственной командой тела функции SELECT;
- **Multi – statement** – возвращает обычную таблицу заданной структуры, при этом количество команд в теле функции не ограничивается.

Создание функций

Команда:

CREATE FUNCTION (Transact-SQL)

создает **определяемую пользователем функцию** в SQL Server. Определяемая пользователем функция представляет собой подпрограмму Transact-SQL или среды CLR, которая принимает параметры, выполняет действия, такие как сложные вычисления, а затем возвращает результат этих действий в виде значения. Возвращаемое значение может быть скалярным значением или таблицей. При помощи этой инструкции можно создать подпрограмму, которую можно повторно использовать следующими способами:

- В инструкциях Transact-SQL, например, **SELECT**.
 - В приложениях, вызывающих функцию.
 - В определении другой пользовательской функции.
 - Для параметризации представления или улучшения функциональности индексированного представления.
 - Для определения столбца таблицы.
 - Для определения ограничения **CHECK** на столбец. □
- Для замены хранимой процедуры.

Синтаксис создания скалярной функции.

```

--Transact-SQL Scalar Function Syntax
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type
  [ = default ] [ READONLY ] }
  [ ,...n ]
]
)
RETURNS return_data_type
  [ WITH <function_option> [ ,...n ] ]
  [ AS ]
BEGIN
    function_body
    RETURN scalar_expression
END
[ ; ]

```

Аргументы `schema_name` - Имя схемы, к которой принадлежит определяемая пользователем функция.

`function_name` - Имя определяемой пользователем функции. Имена функций должны удовлетворять правилам построения идентификаторов и должны быть уникальными в пределах базы данных и схемы.

`@parameter_name` - Аргумент пользовательской функции. Может быть объявлен один или несколько аргументов.

Для функций допускается не более 2100 параметров. При выполнении функции значение каждого из объявленных параметров должно быть указано пользователем, если для них не определены значения по умолчанию.

[`type_schema_name.`] `parameter_data_type` - Тип данных параметра
[`=default`] - Значение по умолчанию для аргумента. Если определено значение `default`, то функция выполняется даже в том случае, если для данного аргумента значение не указано.

`READONLY` - Указывает, что параметр не может быть обновлен или изменен при определении функции. Если тип параметра является определяемым пользователем табличным типом, то должно быть указано ключевое слово `READONLY`.

`return_data_type` - Возвращаемое значение скалярной функции, определяемой пользователем.

`function_body` - Указывает серию инструкций Transact-SQL

`scalar_expression` - Указывает скалярное значение, возвращаемое скалярной функцией.

Предложение EXECUTEAS - Указывает контекст безопасности, в котором выполняется определяемая пользователем функция. Иными словами, есть возможность управлять тем, какую учетную запись пользователя SQLServer использует при определении разрешений на объекты базы данных, на которые ссылается функция.

Синтаксис функции, которая возвращает табличное значение:

```
--Transact-SQL Inline Table-Valued Function Syntax
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type
  [ = default ] [ READONLY ] }
  [ ,...n ]
]
)
RETURNS TABLE
  [ WITH <function_option> [ ,...n ] ]
  [ AS ]
RETURN [ ( ) select_stmt [ ) ]
[ ; ]
```

Аргументы:

TABLE - Указывает, что возвращаемым значением функции с табличным значением, является таблица. Функциям с табличным значением, могут передаваться только константы и @local_variables.

Во встроенных функциях с табличным значением возвращаемое значение TABLE определяется при использовании единственной инструкции **SELECT**. Встроенные функции не имеют соответствующих возвращаемых переменных.

select_stmt - Одиночная инструкция **SELECT**, определяющая возвращаемое значение встроенной функции с табличным значением.

В функциях допустимы следующие инструкции.

- Инструкции присваивания.
- Инструкции управления потоком, за исключением инструкций **TRY...CATCH**.
- Инструкции **DECLARE**, объявляющие локальные переменные и локальные курсоры.
- Инструкции **SELECT**, которые содержат списки выбора с выражениями, присваивающими значения локальным переменным.
- Операции над локальными курсорами, которые объявляются, открываются, закрываются и освобождаются в теле функции. Допустимы только те инструкции **FETCH**, которые предложением **INTO** присваивают

значения локальным переменным. Инструкции **FETCH**, возвращающие данные клиенту, недопустимы.

- Инструкции **INSERT**, **UPDATE** и **DELETE**, которые изменяют локальные табличные переменные.
- Инструкции **EXECUTE**, вызывающие расширенные хранимые процедуры. **Ограничения**

Определяемые пользователем функция не может выполнять действия, изменяющие состояние базы данных.

Определяемые пользователем функции не могут содержать предложение **OUTPUT INTO**, целью которого является таблица.

Определяемые пользователем функции могут быть **вложенными**, то есть из одной функции может быть вызвана другая. Вложенность определяемых пользователем функций не может превышать 32 уровней.

Пример создание функции

В среде SQL Server Management Studio все пользовательские функции находятся в папке «Функции», расположенной в папке «Программирование» в обозревателе объектов (рис.7.1).

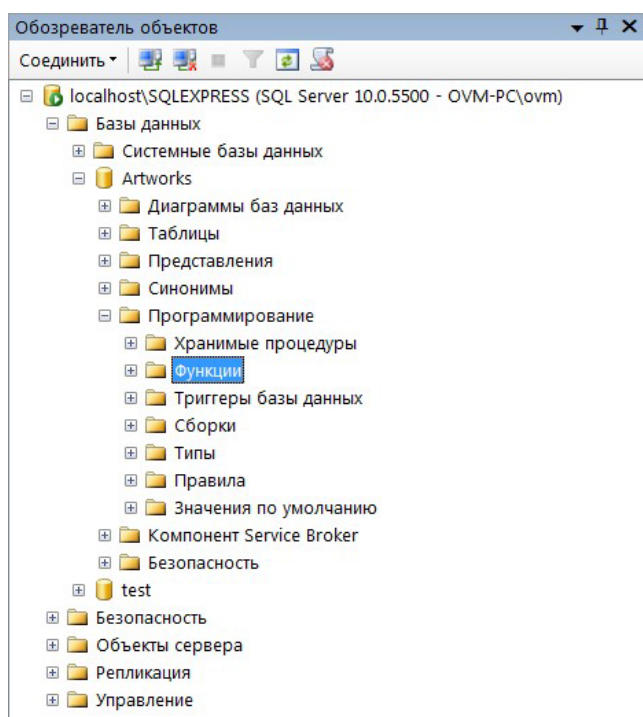


Рис.7.1

Начнём с создания скалярных пользовательских функций. Для создания новой скалярной пользовательской функции в обозревателе объектов щёлкните правой кнопкой мыши по папке «Функции» и в появившемся меню выберите

пункт «Создать → Скалярная функция...». Появится окно новой скалярной пользовательской функции (рис.7.2).

```

-- Template generated from Template Explorer using:
-- Create Scalar Function (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- This block of comments will not be included in
-- the definition of the function.
=====
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
=====
-- Author: <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
=====
CREATE FUNCTION <Scalar_Function_Name, sysname, FunctionName>
(
  -- Add the parameters for the function here
  <@Param1, sysname, @p1> <Data_Type_For_Param1, , int>
)
RETURNS <Function_Data_Type, ,int>
AS
BEGIN
  -- Declare the return variable here
  DECLARE <@ResultVar, sysname, @Result> <Function_Data_Type, ,int>

  -- Add the T-SQL statements to compute the return value here
  SELECT <@ResultVar, sysname, @Result> = <@Param1, sysname, @p1>

  -- Return the result of the function
  RETURN <@ResultVar, sysname, @Result>
END

```

Рис.7.2

Синтаксис скалярной пользовательской функции похож на синтаксис хранимой процедуры. Но есть и существенные отличия (рис.6.2):

1. Область определения имени функции (Scalar_Function_Name);
2. Параметры, передаваемые в функцию (@param1). Определение параметров аналогично определению параметров в хранимой процедуре;
3. Тип данных значения возвращаемого функцией;
4. Область объявления переменных, используемых внутри функции. Объявление переменных имеет следующий синтаксис: DECLARE @Имя переменной <Тип данных>
5. Тело самой пользовательской функции, содержит команды языка T-SQL;
6. Команда RETURN возвращающая результат выполнения функции.

Имеет следующий синтаксис:

RETURN @Имя переменной с результатом

Переменная должна быть того же типа данных, который был указан в пункте 3.

Создадим скалярную пользовательскую функцию, вычисляющую среднее трёх величин. В окне новой пользовательской функции наберите код представленный на рис. 7.3.


```
SQLQuery2.sql - localhost\... (52)*
-- Create Scalar Function (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- This block of comments will not be included in
-- the definition of the function.
=====
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
=====
-- Author:      <Author,Name>
-- Create date: <Create Date, ,>
-- Description: <Description, ,>
=====
CREATE FUNCTION FuncMeanValue
(
    -- Add the parameters for the function here
    @Value1 int, @Value2 int, @Value3 int
)
RETURNS real
AS
BEGIN
    -- Declare the return variable here
    DECLARE @Result real

    -- Add the T-SQL statements to compute the return value here
    SELECT @Result = (@Value1 + @Value2 + @Value3) / 3

    -- Return the result of the function
    RETURN @Result
END
GO
```

Рис.7.3

Рассмотрим более подробно код данной скалярной пользовательской функции:

1. CREATEFUNCTIONFuncMeanValue определяет имя создаваемой функции как FuncMeanValue;
2. @Value1, @Value2, @Value3 определяют три параметра процедуры Value1, Value2 и Value3. Данным параметрам можно присвоить целые числа (тип данных int);
3. RETURNSreal показывает, что функция возвращает вещественные числа (тип данных real);
4. DECLARE @Resultreal – объявляется переменная @Result для хранения результата работы функции вещественного типа (тип данных real);
5. SELECT @Result = (@Value1 + @Value2 + @Value3) / 3 вычисляет среднее и помещает результат в переменную @Result;
6. RETURN @Result возвращает значение переменной @Result.

Для создания функции, выполним вышеописанный код, нажав кнопку «Выполнить» на панели инструментов. В нижней части окна с кодом появится сообщение «Выполнение команд успешно завершено.».

Для проверки работы созданной скалярной пользовательской функции необходимо создать новый пустой запрос, нажав на кнопку «Создать запрос» на панели инструментов. В появившемся окне с пустым запросом наберите

команду `SELECT dbo.FuncMeanValue (3, 5, 4)` и нажмите кнопку «Выполнить» на панели инструментов (рис.7.4).

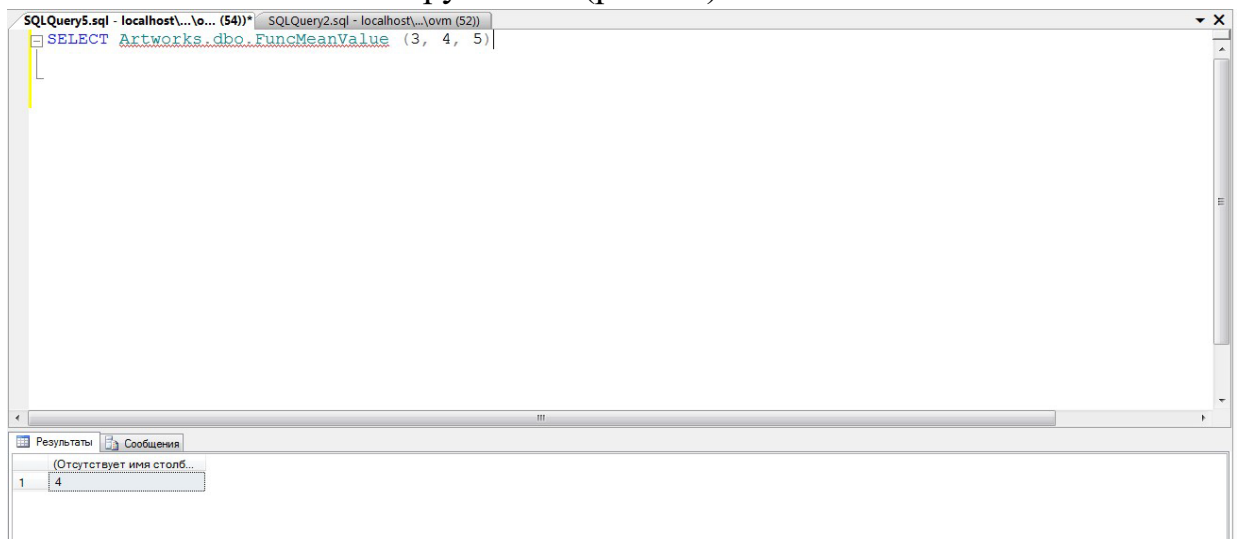


Рис.7.4

В нижней части окна с кодом появится результат выполнения новой скалярной пользовательской функции.

Теперь перейдём к созданию табличных пользовательских функций. Для создания табличной пользовательской функции в обозревателе объектов щёлкните правой кнопкой мыши по папке «Функции» и в появившемся меню выберите пункт «Создать → Встроенная функция, возвращающая табличное значение...». Появится окно новой табличной пользовательской функции (рис.7.5).

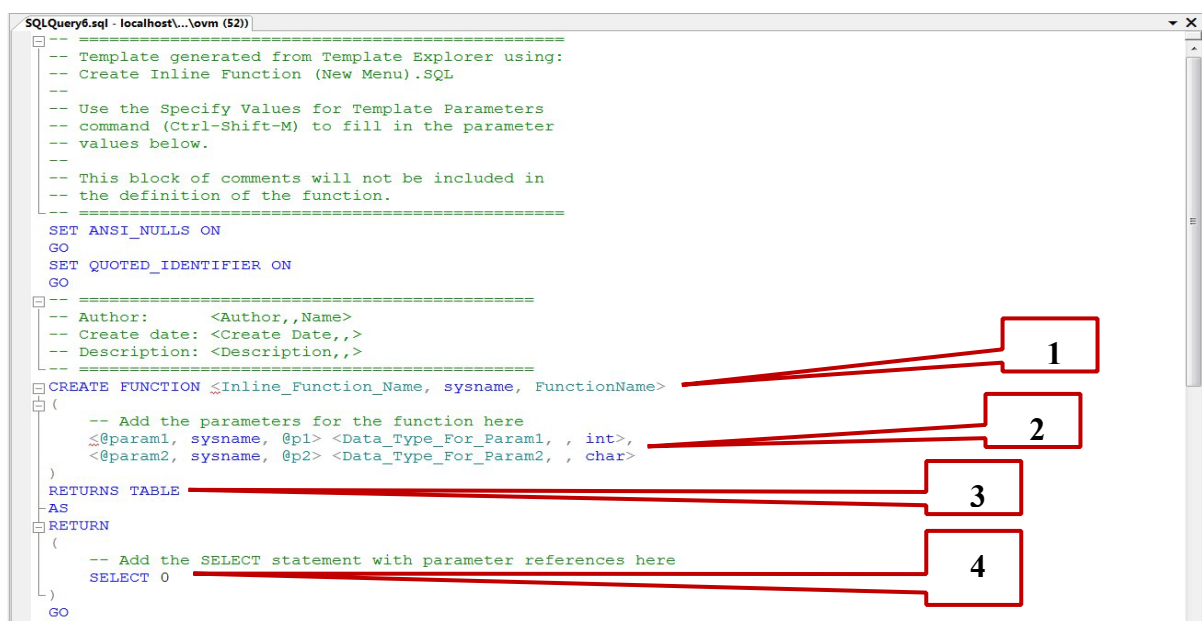
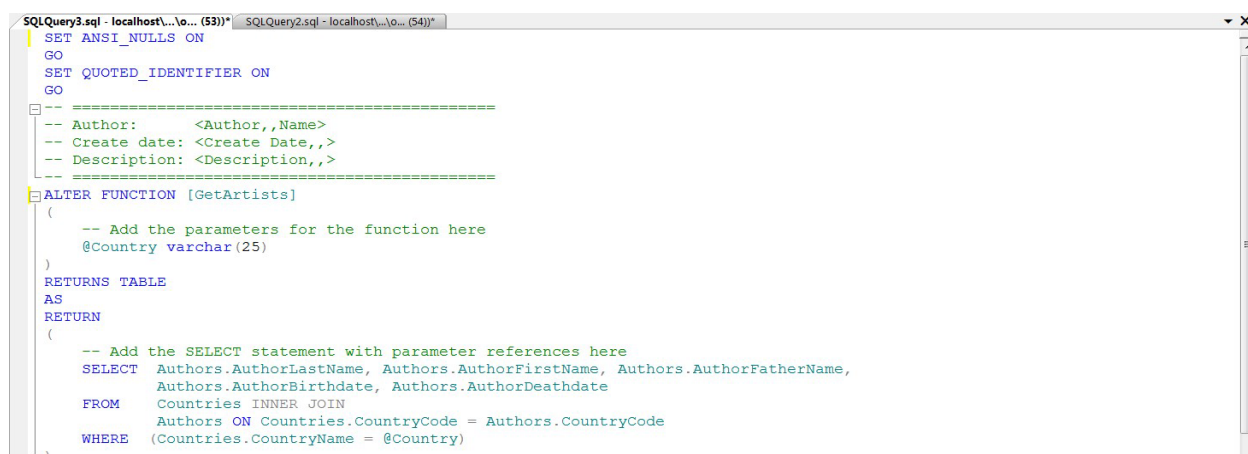


Рис.7.5

Табличная пользовательская функция состоит из следующих разделов:

1. Область определения имени функции (Inline_Function_Name);
2. Параметры, передаваемые в функцию (@param1, @param2);
3. RETURNSTABLE показывает, что функция возвращает таблицу;
4. Тело самой пользовательской функции состоит из команды SELECT языка TSQL.

Рассмотрим создание табличной пользовательской функции, выбирающей информацию о художниках по стране их проживания. В окне новой пользовательской функции (рис. 7.5) наберем следующий код (рис. 7.6):



```

SQLQuery3.sql - localhost\...\o... (53)*  SQLQuery2.sql - localhost\...\o... (54)*
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
=====
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
=====
ALTER FUNCTION [GetArtists]
(
    -- Add the parameters for the function here
    @Country varchar(25)
)
RETURNS TABLE
AS
RETURN
(
    -- Add the SELECT statement with parameter references here
    SELECT  Authors.AuthorLastName, Authors.AuthorFirstName, Authors.AuthorFatherName,
            Authors.AuthorBirthdate, Authors.AuthorDeathdate
    FROM    Countries INNER JOIN
            Authors ON Countries.CountryCode = Authors.CountryCode
    WHERE   (Countries.CountryName = @Country)
)

```

Рис.7.6

Из кода функции видно, что она принимает один параметр типа varchar (название страны) и реализуется запросом:

```

SELECT Authors.AuthorLastName, Authors.AuthorFirstName,
Authors.AuthorFatherName, Authors.AuthorBirthdate,
Authors.AuthorDeathdate
FROM Countries
    INNER JOIN Authors
    ON Countries.CountryCode = Authors.CountryCode
WHERE (Countries.CountryName = @Country)

```

Из таблицы «Authors» выбираются фамилии, имена, отчества, даты рождения и смерти художников, при этом таблица «Authors» соединяется (INNER JOIN) с таблицей «Countries» по столбцу «CountryCode». Условием соединения является равенство содержимого поля «CountryCode» передаваемому в функцию параметру. Результат выполнения этой функции приведен на рис. 7.7.

The screenshot shows a SQL query window with the following text:

```
select * from GetArtists('Россия')
```

Below the query window, the results are displayed in a table with the following columns: AuthorLastName, AuthorFirstName, AuthorFatherName, AuthorBirthdate, and AuthorDeathdate. The results are as follows:

	AuthorLastName	AuthorFirstName	AuthorFatherName	AuthorBirthdate	AuthorDeathdate
1	Айвазовский	Иван	Константинович	1817-07-17	1900-04-19
2	Шишкин	Иван	Иванович	1832-01-25	1898-03-08
3	Репин	Илья	Ефимович	1844-08-05	1930-09-29
4	Коровин	Константин	Алексеевич	1861-12-05	1939-09-11
5	Шагал	Марк	Захарович	1887-07-06	1985-03-28
6	Рерих	Николай	Константинович	1874-10-09	1947-12-13
7	Полenov	Василий	Дмитриевич	1844-06-01	1927-07-18
8	Кандинский	Василий	Васильевич	1866-12-16	1944-12-13

Рис. 7.7

Работа с табличной функцией осуществляется так же, как и с обыкновенной таблицей.

Задание

Создать функции в соответствии с заданием для своего варианта.

ЛАБОРАТОРНАЯ РАБОТ 8.

МАНИПУЛИРОВАНИЕ ДАННЫМИ. ТРИГГЕРЫ

ЗАДАНИЕ

3. Создать аналогичные приведенным примерам триггеры по своей БД
4. Сохранить каждый триггер в отдельный запрос или сделать скриншот выполнения запроса на создание и отобразить это в отчете по лабораторной работе).

Триггеры

Триггеры являются одной из разновидностей хранимых процедур. Их исполнение происходит при выполнении для таблицы какого-либо оператора языка манипулирования данными (DML). Триггеры используются для проверки целостности данных, а также для отката транзакций.

Триггер – это откомпилированная SQL-процедура, исполнение которой обусловлено наступлением определенных событий внутри реляционной базы данных. Применение триггеров большей частью весьма удобно для пользователей базы данных. И все же их использование часто связано с дополнительными затратами ресурсов на операции ввода/вывода. В том случае, когда тех же результатов (с гораздо меньшими непроизводительными затратами ресурсов) можно добиться с помощью хранимых процедур или прикладных программ, применение триггеров нецелесообразно.

Триггеры – особый инструмент SQL-сервера, используемый для поддержания целостности данных в базе данных. С помощью ограничений целостности, правил и значений по умолчанию не всегда можно добиться нужного уровня функциональности. Часто требуется реализовать сложные алгоритмы проверки данных, гарантирующие их достоверность и реальность. Кроме того, иногда необходимо отслеживать изменения

значений таблицы, чтобы нужным образом изменить связанные данные. Триггеры можно рассматривать как своего рода фильтры, вступающие в действие после выполнения всех операций в соответствии с правилами, стандартными значениями и т.д.

Триггер представляет собой специальный тип хранимых процедур, запускаемых сервером автоматически при попытке изменения данных в таблицах, с которыми триггеры связаны. Каждый триггер привязывается к конкретной таблице. Все производимые им модификации данных рассматриваются как одна транзакция. В случае обнаружения ошибки или нарушения целостности данных происходит откат этой транзакции. Тем самым внесение изменений запрещается. Отменяются также все изменения, уже сделанные триггером.

Создает триггер только владелец базы данных. Это ограничение позволяет избежать случайного изменения структуры таблиц, способов связи с ними других объектов и т.п.

Триггер представляет собой весьма полезное и в то же время опасное средство. Так, при неправильной логике его работы можно легко уничтожить целую базу данных, поэтому триггеры необходимо очень тщательно отлаживать.

В отличие от обычной подпрограммы, триггер выполняется неявно в каждом случае возникновения триггерного события, к тому же он не имеет аргументов. Приведение его в действие иногда называют запуском триггера. С помощью триггеров достигаются следующие цели:

- проверка корректности введенных данных и выполнение сложных ограничений целостности данных, которые трудно, если вообще возможно, поддерживать с помощью ограничений целостности, установленных для таблицы;

- выдача предупреждений, напоминающих о необходимости выполнения некоторых действий при обновлении таблицы, реализованном определенным образом;

□ накопление аудиторской информации посредством фиксации сведений о внесенных изменениях и тех лицах, которые их выполнили;

□ поддержка репликации.

Ниже приведен синтаксис создания триггера:

```
CREATE
TRIGGER trigger_name trigger_time trigger_event ON tbl_name
FOR EACH ROW trigger_stmt
```

trigger_name — название триггера; **trigger_time** — время срабатывания триггера:

BEFORE — перед событием; AFTER — после события.

trigger_event — Событие:

insert — событие активируется операторами insert, dataload, replace;

update — событие активируется оператором update

delete — событие активируется операторами delete, replace.

Операторы DROPTABLE и TRUNCATE не активируют выполнение триггера

tbl_name — название таблицы;

trigger_stmt - выражение, которое выполняется при активации триггера

Рассмотрим несколько простых примеров создания триггеров для существующей базы данных. В рамках лабораторной работы необходимо создать и протестировать любых три триггера (из рассмотренных примеров ниже, изучив их работу)

Пример 1. Триггер на добавление записи в таблицу USERS.

Данный триггер в случае успешного добавления данных выводит в «Запись добавлена»

```
CREATE TRIGGER INSERT_INDICATION --
определение имени функции
```

```

ON USERS          --для какой таблицы создается
                  триггер
AFTER INSERT      --когда выполнять триггер
--INSERT - при создании записи в таблице,
--DELETE - при удалении записи в таблице,
--UPDATE - при изменении записи в таблице,
--AFTER - после выполнения операции,
--INSTEADOF - вместо выполнения операции
AS
BEGIN --тело триггера
SET NOCOUNT ON;
PRINT 'Запись добавлена'
END
GO

```

Тестирование работы триггера

```

INSERT INTO USERS VALUES
('Громова', 'Валерьевна', 'Анна', '55-66-89', NULL,
NULL, 'active'),
('Кремнева', 'Александровна', 'Александра',
'9058956458', 'cremneva@mail.ru', NULL, 'passive')

```

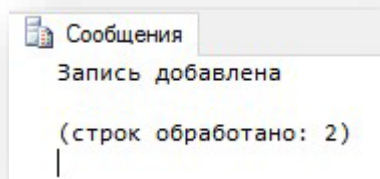


Рис. 8.1 – Результат работы триггера

Пример 2. Триггер на изменение записи в таблицу USERS

```

CREATE TRIGGER UPDATE_INDICATION

```



```
ON USERS
AFTER UPDATE
AS
BEGIN
SET NOCOUNT ON;
PRINT 'Запись изменена'
END
```

Тестирование работы триггера

```
UPDATE USERS
SET URL = 'cremnera.tomsk.ru'
WHERE SURNAME = 'Кремнева'
```

Пример 3. Триггер на удаление записи из таблицы USERS

```
CREATE TRIGGER DELETE_INDICATION
ON USERS
AFTER DELETE
AS
BEGIN
SET NOCOUNT ON;
PRINT 'Запись удалена'
END
GO
```

Тестирование работы
триггера

```
DELETE FROM USERS
WHERE SURNAME = 'Громова'
```

Пример 4. Триггер, демонстрирующий откат

```
CREATE TRIGGER ROLLBACK_EXAMPLE
ON ORDERS
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;
IF (SELECT NUMBER FROM inserted) < 1
ROLLBACK
PRINT 'Вы не можете создать заказ с количеством
меньше 1 '
END
GO
```

Тестированиеработытриггера

```
INSERT INTO ORDERS VALUES (2, '2005-01-06
12:39:38', 0,
```

20)

Пример 5. Триггер на изменение количества товаров при их заказе. Количество проданного товара должно быть не меньше, чем его остаток из таблицы PRODUCTS

```
CREATE TRIGGER NUMBER_UPDATE
ON ORDERS
AFTER INSERT
AS
DECLARE @X INT, @Y INT
BEGIN
SET NOCOUNT ON;
IF NOT EXISTS (SELECT * FROM inserted
```

```

WHERE inserted.NUMBER<= ALL (SELECT PRODUCTS.COUNT
FROM PRODUCTS WHERE inserted.ID_PRODUCT =
PRODUCTS.ID_PRODUCT))
BEGIN
ROLLBACK TRAN
PRINT 'откат! товара нет '
END
SELECT @Y = O.ID_PRODUCT, @X=O.NUMBER
FROM inserted O
UPDATE PRODUCTS
SET PRODUCTS.COUNT = PRODUCTS.COUNT - @X
WHERE PRODUCTS.ID_PRODUCT = @Y
END
GO

```

Тестированиеработытриггера

```

INSERT INTO ORDERS VALUES ( 4, '2005-01-04
18:39:38', 12,

```

28)

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе были приведены практические примеры по созданию выборок и триггеров.

Лабораторная работа № 9

Создание пользователей для доступа к серверу через утилиту MicrosoftSQLServerManagementStudio

Создадим новую учетную запись для нашей базы данных **University**. Для этого выберите в Обозревателе объектов раздел **Безопасность/Имена входа**. Добавьте новое имя входа – **Proba**, установите опцию **Проверка подлинности SQL Server**, присвойте свой пароль, примените к выбранной базе данных, установите язык по умолчанию – русский.

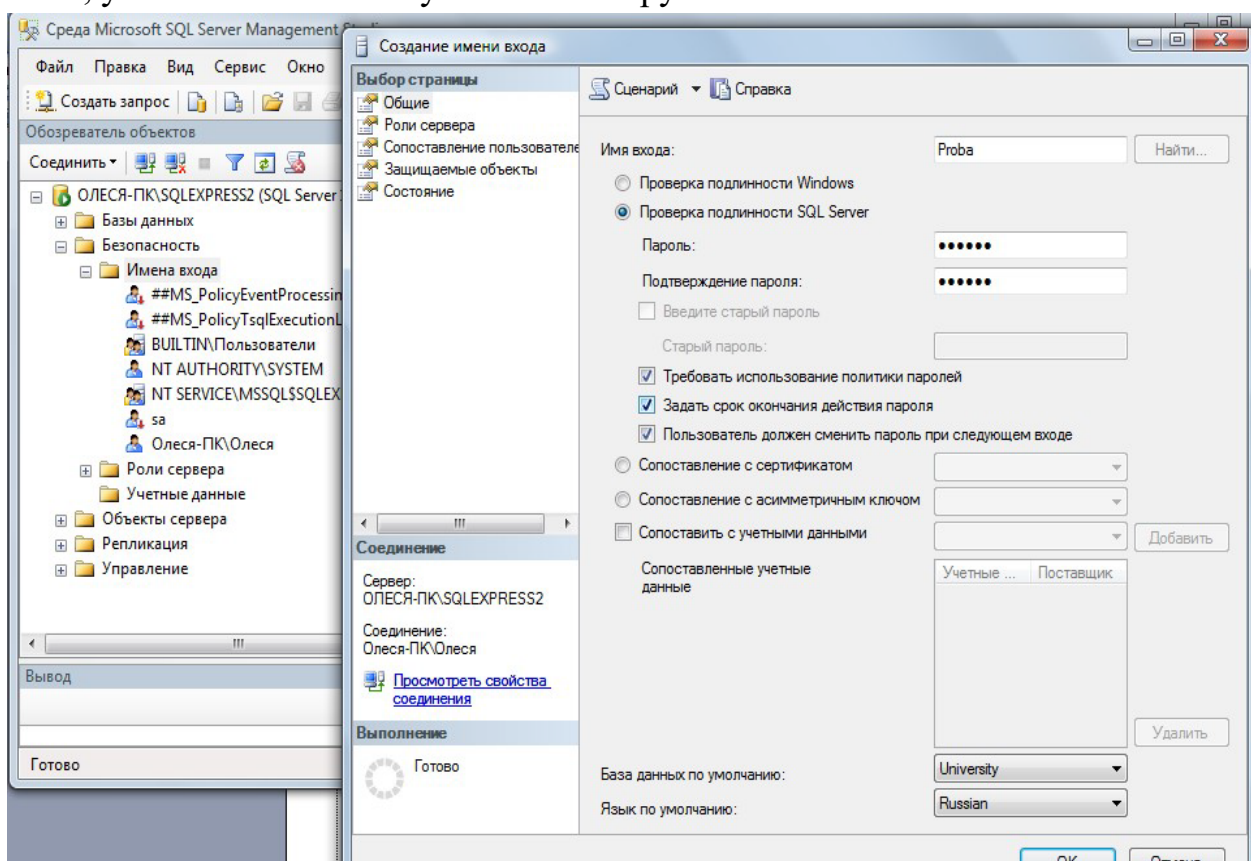


Рис. 9.1. Раздел Безопасность (Security) для работы с пользователями и создание нового пользователя (при SQLServer аутентификации нужно снять галочки с **Enforcepasswordpolicy**)

Прежде чем добавлять нового пользователя посмотрите его назначенные серверные роли. Для этого в этом же окне выберите раздел **Роли сервера**. Установите для пользователя **Proba** роль **sysadmin**.

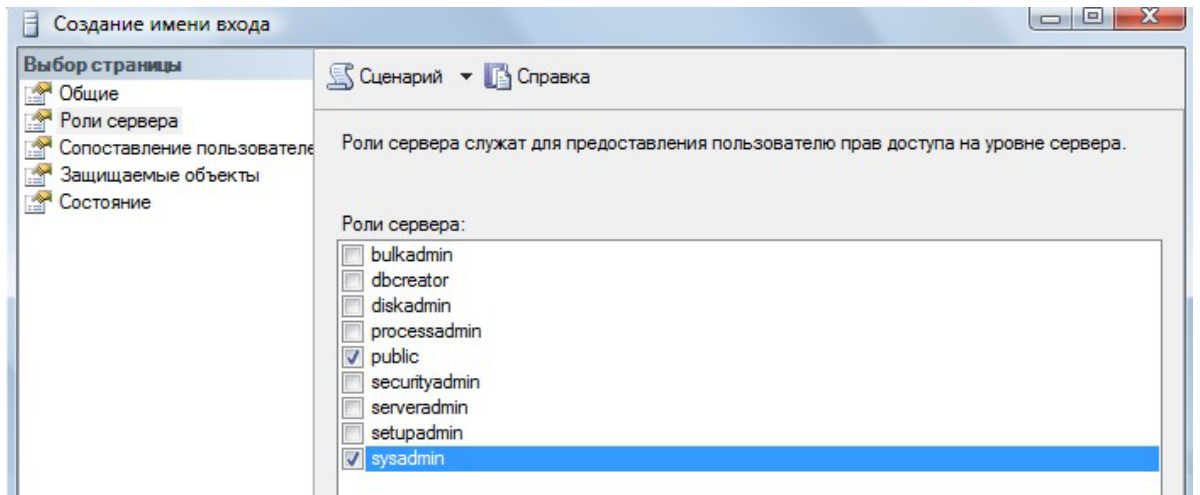


Рис. 9.2. Настройка серверной роли для нового пользователя (весь список серверных ролей с их привилегиями в конце работы)

Далее просмотрите раздел **Сопоставление пользователя**. Установите для базы данных **University** пользователя **Proba** права доступа **Db_owner**, означающие, что пользователь может выполнять **любые действия с БД**. Ниже перечислены все возможные варианты прав доступа.

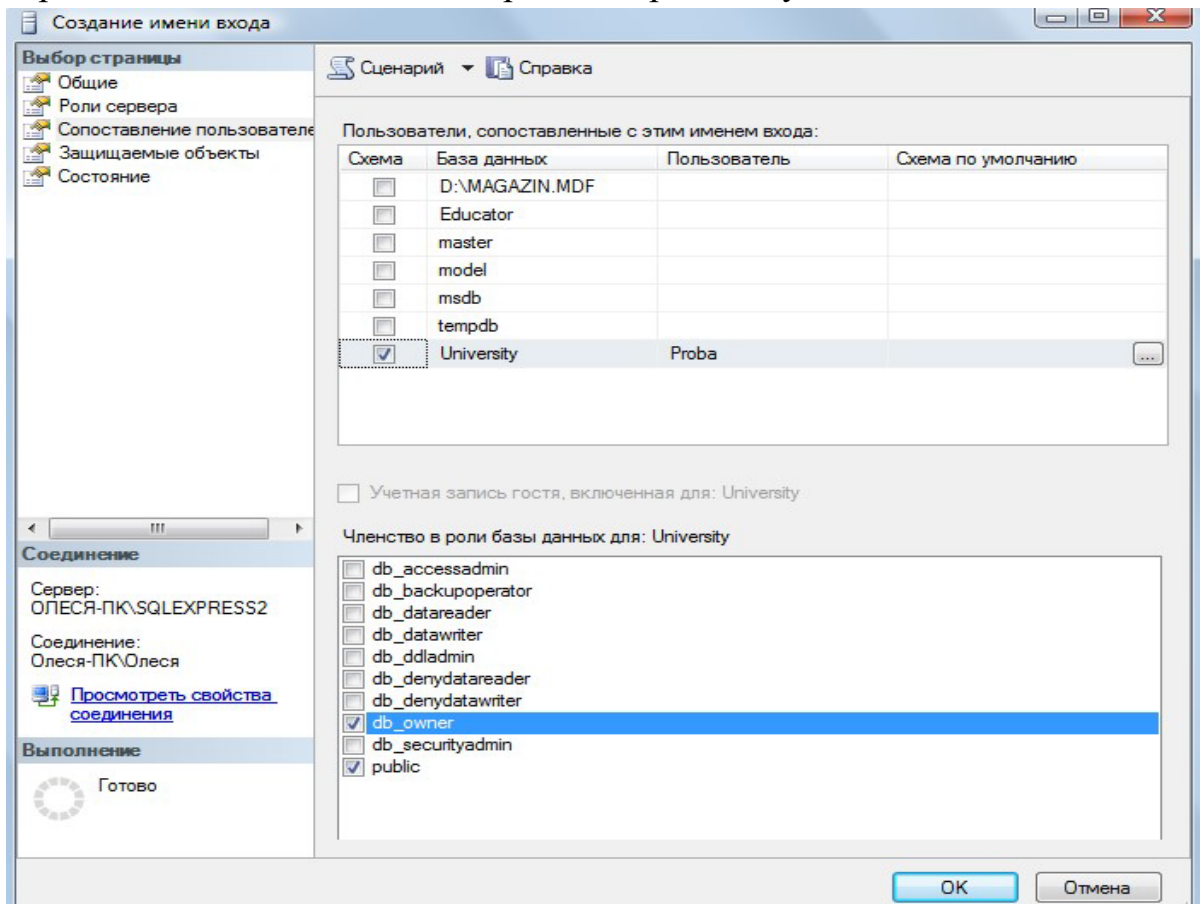


Рис. 9.3. Настройка роли базы данных для нового пользователя (весь список ролей баз данных с их привилегиями ниже)

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_securityadmin – 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

Далее перейдите на раздел **Состояние**. Установите опции **Разрешение к подключению к ядру СУБД** – предоставить и **имя входа** включить.

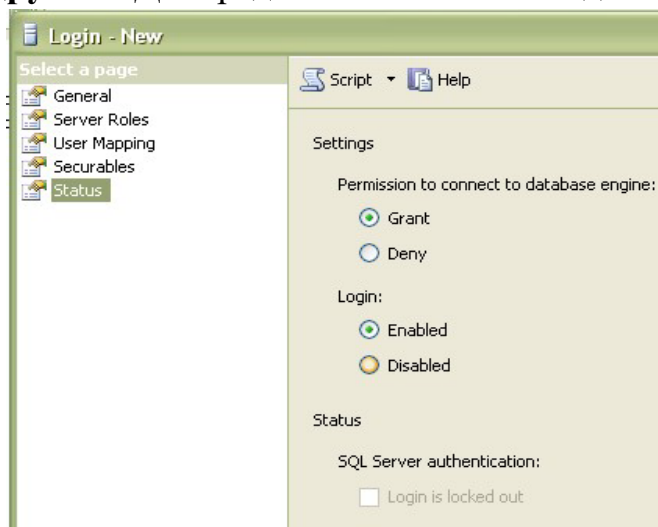


Рис. 9.4. Разблокирование создаваемой учетной записи

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

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

Создание ролей программно

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

Создадим для нашего примера роли декана (**ДЕКАН**) и студента (**STUDENT**).

Пример создания роли декана:

```
USEUniversity --сделатьтекущейБДUniversity  
EXECsp_addrole 'ДЕКАН'
```

Эти операторы набрать на странице, вызванной нажатием кнопки **<Создать запрос>**.

Для запуска команд на выполнение нажать .

Сохраните запрос.

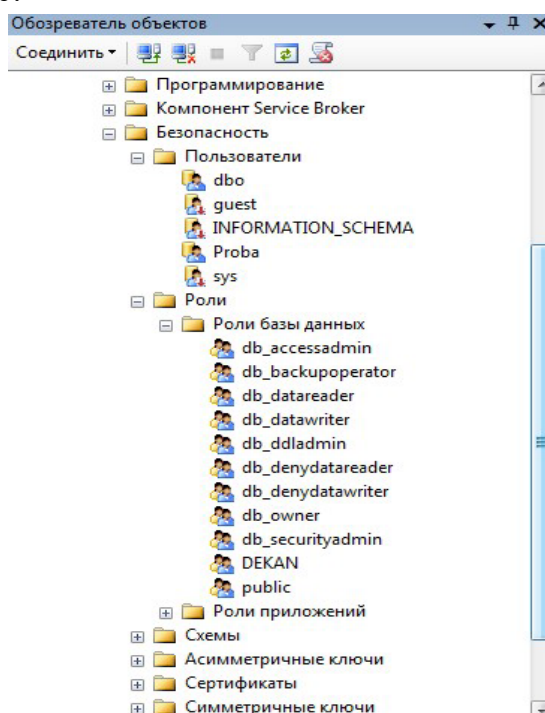


Рис.9.5. – Обозреватель объектов

Повторный запуск тех же команд сгенерирует ошибки типа «**В БД уже существует роль ДЕКАН**».

Чтобы посмотреть, что роль добавлена, откройте вкладку **Безопасность/Роли/Роли базы данных**.

Пример создания роли студента:

```
USEUniversity --сделатьтекущейБДUniversity  
EXECsp_addrole 'STUDENT'
```

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>  
[WITHgrantoption];
```

Атрибут **WITHGRANTOPTION** 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 (невыполнять):

```
GRANT select, update (Sales, num) ON Sales_data TO user1  
WITHGRANTOPTION
```

Пользователь, предоставивший привилегию другому, называется **грантор** (grantor — поставитель). 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.

Для примера (немного забегаая вперед) создадим таблицу **Discuplinu**. Без объяснения синтаксиса выполните следующий sql-запрос:

```
USEUniversity --сделатьтекущейБДuniversitycreatetableDiscuplinu (  
Kod_Discuplinuint NOT NULL primary key,  
name_Discuplinuchar(30) NULL, kol_chasovint NULL  
);
```

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 **ДЕКАН**. 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.

Самостоятельно просмотрите свойства роли базы данных **Student**. Просмотрите назначенные общие свойства, защищаемые объекты и расширенные свойства.

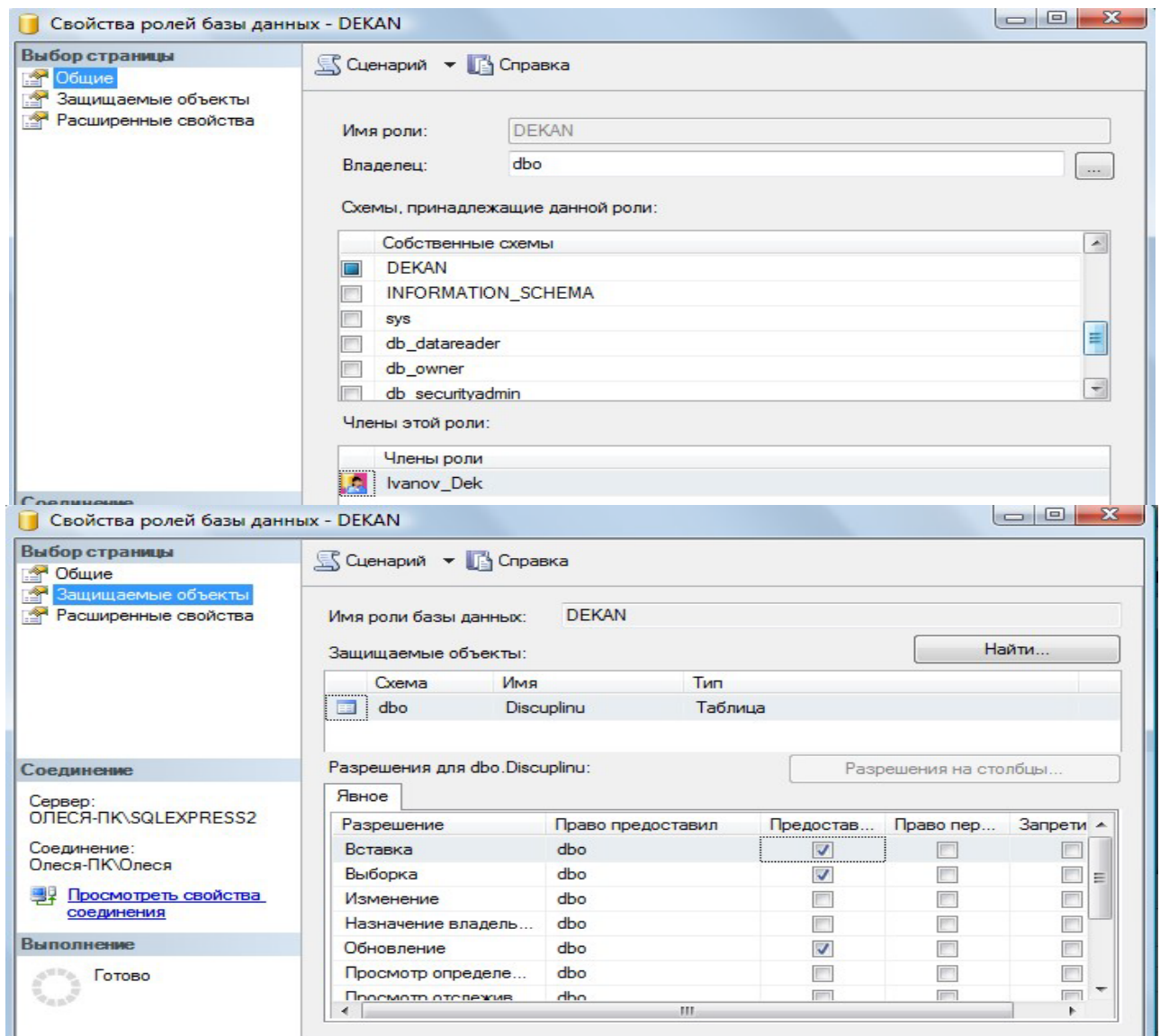


Рис.9.6. Свойства ролей базы данных

Оператор отмены привилегий

Синтаксис отмены привилегий:

REVOKE [withgrantoption]

< привилегии>,...

ON< объект >,...

FROM<имя_пользователя>;

Предложение **withgrantoption** сохраняет за пользователем перечисленные привилегии, но отменяет его право передавать их кому-либо другому.

Пример:

REVOKE SELECT ON Disciplinu FROM STUDENT

Выполните команду.

Оператор изымания роли у пользователя

Revoke<список ролей>**from**<список пользователей>.

Пример:

useUniversity

EXECsp_droprolemember 'STUDENT', 'Petrov_Stud' Выполните команду и просмотрите результат.

Задание

- Создать файл базы данных с помощью sql-команды.
- Создать резервную копию базы данных.
- Определить **2-3** должностных лица, которые смогут работать с таблицами БД. Для каждого должностного лица определить набор привилегий, которыми он может пользоваться.
- В утилите **SQLServerManagementStudio** создать под каждое должностное лицо соответствующую роль, наделить эту роль определенными привилегиями. Далее создать по одному пользователю на каждую должность и присвоить им соответствующие роли.
- Сохранить последовательно SQL-операторы с указанием заданий в файле с названием **ФамилияСтудента_Лаб_9**.
- Создать текстовый отчет, в котором отобразить sql-команды разработанных запросов и скриншоты результатов работы из СУБД **SQLServerManagement Studio**.

Использованные источники

1. Лабораторные работы «Работы с SQL». – Режим доступа: https://portal.tpu.ru/SHARED/s/SKIRNEVSKIY/academic/discipline/Tab2/LB02.Physical_model_0.pdf
2. В.Ю. Кара-Ушанов. SQL - язык реляционных баз данных: учебное пособие – Екатеринбург: Издательство Урал ун-та, 2016. - 156 с.

Список литературы

1. Кодд Э.Ф. Реляционная модель данных для больших совместно используемых банков данных [Электронный ресурс]/Э.Ф. Кодд; пер. с англ. М. Р. Когаловского//СУБД. 1995. № 1. Режим доступа: <http://citforum.ru/database/classics/codd/>. Загл. с экрана.
2. Дейт К. Дж. Введение в системы баз данных: пер с англ./ К. Дж. Дейт. 6-е изд. Киев; М.; СПб.: Изд. дом «Вильямс», 1999. 848 с.
3. Коннолли Т. Базы данных: проектирование, реализация и сопровождение. Теория и практика : [пер. с англ.]/Т. Коннолли, К. Бегг, А. Страчан. 2-е изд.М.: Изд. дом «Вильямс», 2000. 1120 с.
4. Пушников А.Ю. Введение в системы управления базами данных. Часть 1. Реляционная модель данных [Электронный ресурс]: учеб. пособие/А.Ю. Пушников. Уфа: изд. Башкир. ун-та, 1999. 108с. [Электронный ресурс]. Режим доступа: <http://www.citforum.ru/database/dblearn/index.shtml>. Загл. с экрана.
5. Кириллов В. В. Основы проектирования реляционных баз данных [Электронный ресурс]/В.В. Кириллов. СПб.: Изд-во С.-Петербур. гос. ин-та точной механики и оптики (техн. ун-т) Режим доступа: <http://citforum.ru/database/dbguide/>. Загл. с экрана.
6. Грабер М. Введение в SQL/М. Грабер.М.: Лори, 2010. 228 с.
7. Дунаев В. В. Базы данных. Язык SQL/В. В. Дунаев. СПб.: БХВ-Петербург, 2006. 288 с.
8. Форта Б. SQL за 10 минут/Б. Форта. 4-е изд.М.: Изд. дом «Вильямс», 2014. 288 с.

