Оглавление

[**Лабораторная работа №1** 2](#_Toc7620214)

[**Лабораторная работа №2** 7](#_Toc7620215)

[**Лабораторная работа №3** 9](#_Toc7620216)

[**Лабораторная работа №4** 15](#_Toc7620217)

[**Лабораторная работа №5** 18](#_Toc7620218)

[**Лабораторная работа №6** 22](#_Toc7620219)

[**Лабораторная работа №7** 29](#_Toc7620220)

[**Лабораторная работа №8** 32](#_Toc7620221)

[**Лабораторная работа №9** 40](#_Toc7620222)

[**Лабораторная работа №10** 44](#_Toc7620223)

[**Лабораторная работа № 12.** 49](#_Toc7620224)

[**Лабораторная работа № 11.** 54](#_Toc7620225)

[**Лабораторная работа № 13.** 59](#_Toc7620226)

[**Лабораторная работа № 14.** 65](#_Toc7620227)

[**Лабораторная работа № 15.** 71](#_Toc7620228)

[**Лабораторная работа № 16.** 76](#_Toc7620229)

[**Лабораторная работа № 17.** 78](#_Toc7620230)

## **Лабораторная работа №1**

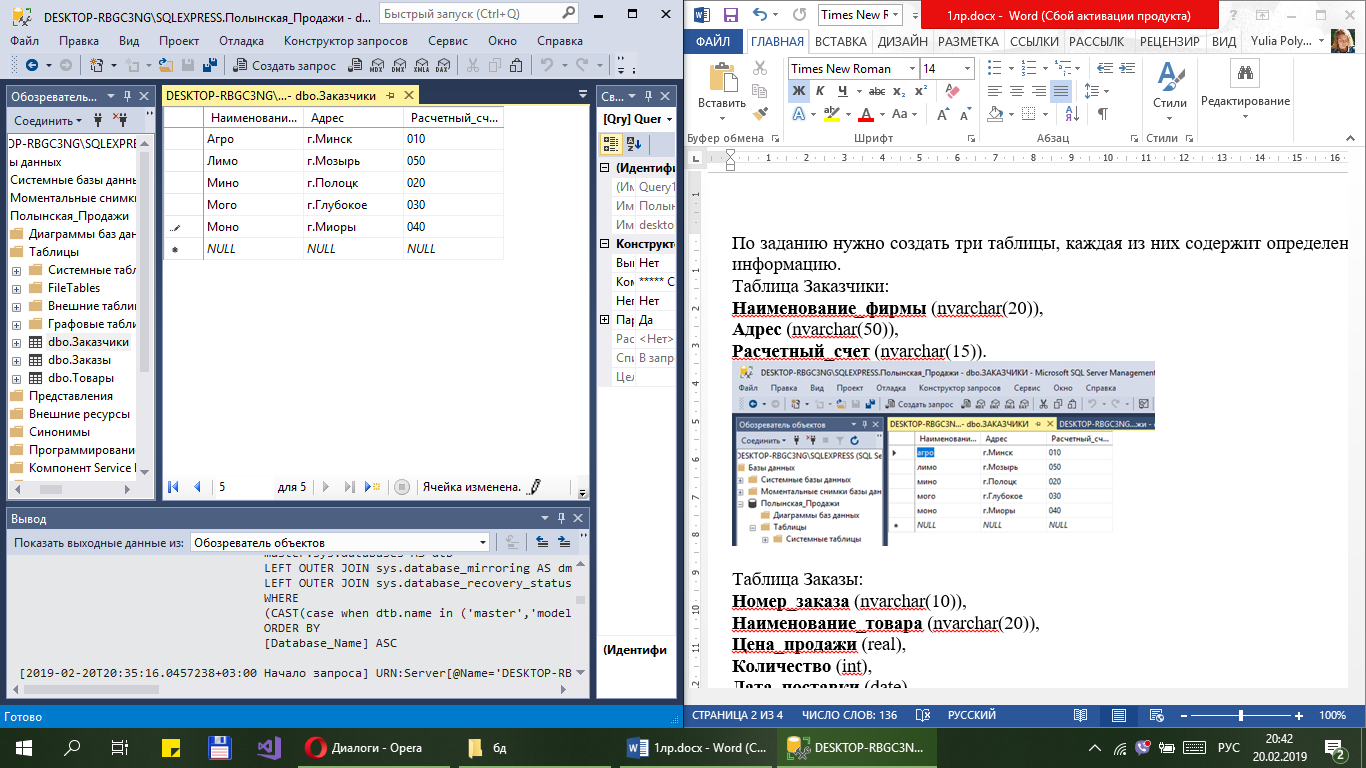
**Знакомство с инструментами СУБД**

По заданию нужно создать три таблицы, каждая из них содержит определенную информацию.  
Таблица Заказчики:

**Наименование*\_*фирмы** (nvarchar(20)),

**Адрес** (nvarchar(50)),

**Расчетный\_счет** (nvarchar(15)).

  
  
Таблица Заказы**:**

**Номер\_заказа** (nvarchar(10)),

**Наименование\_товара** (nvarchar(20)),

**Цена\_продажи** (real),

**Количество** (int),

**Дата\_поставки** (date),

**Заказчик** (nvarchar(20)).

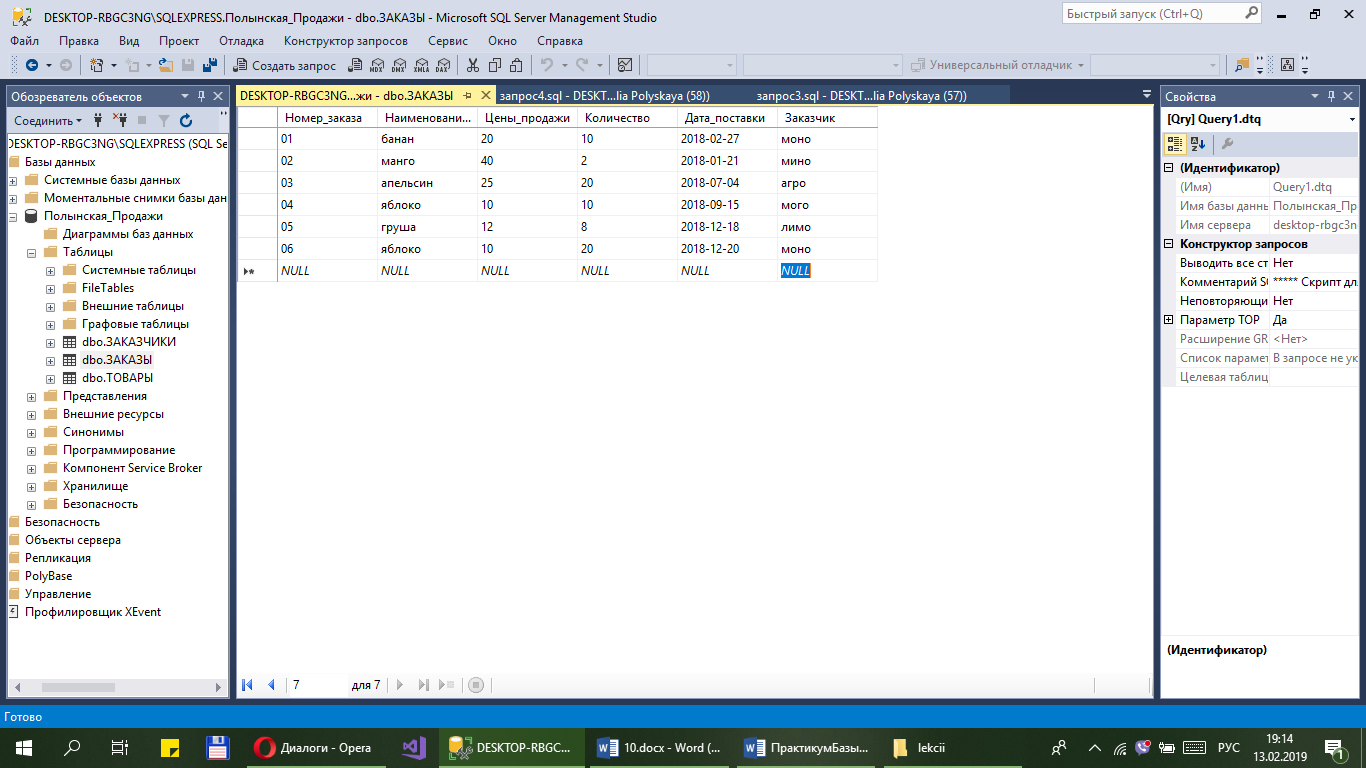
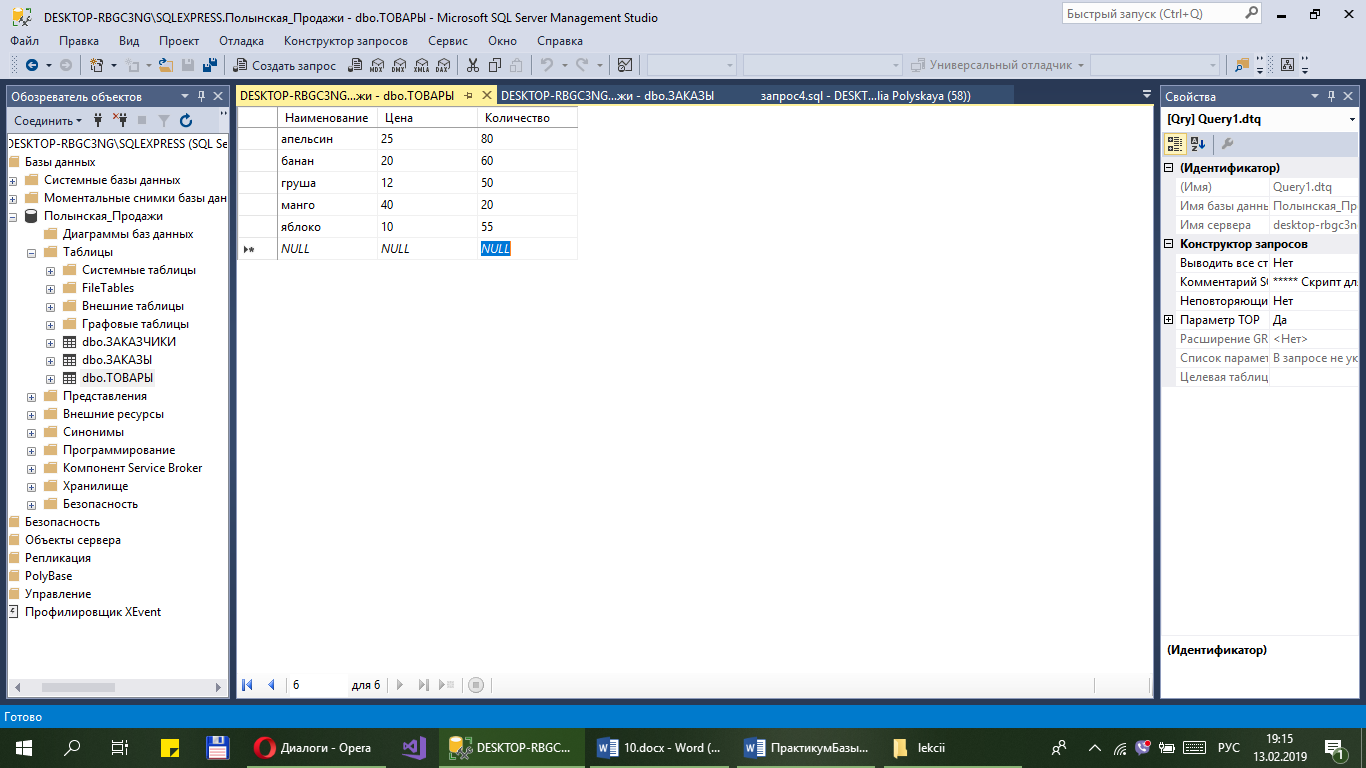


Таблица Товары:

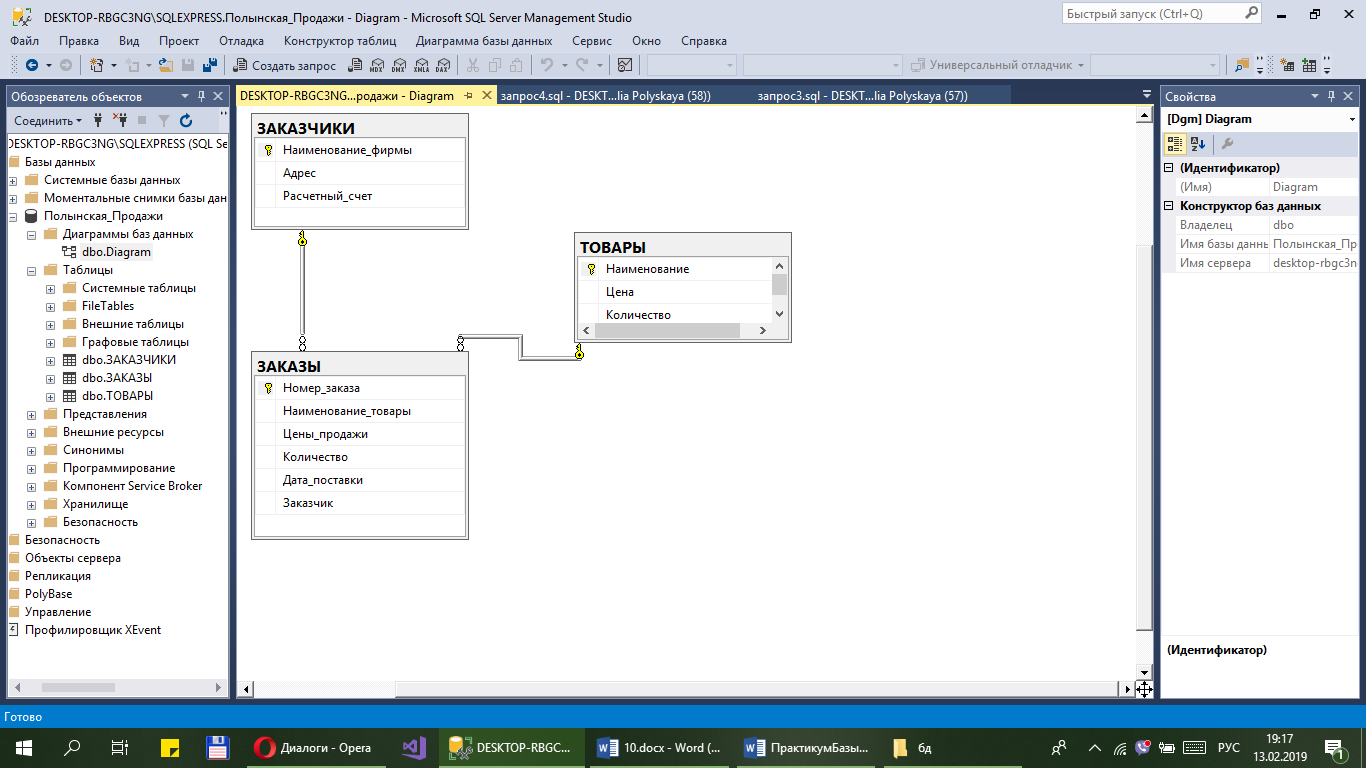
**Наименование** (nvarchar(20))**,**

**Цена** (real)**,**

**Количество** (int).

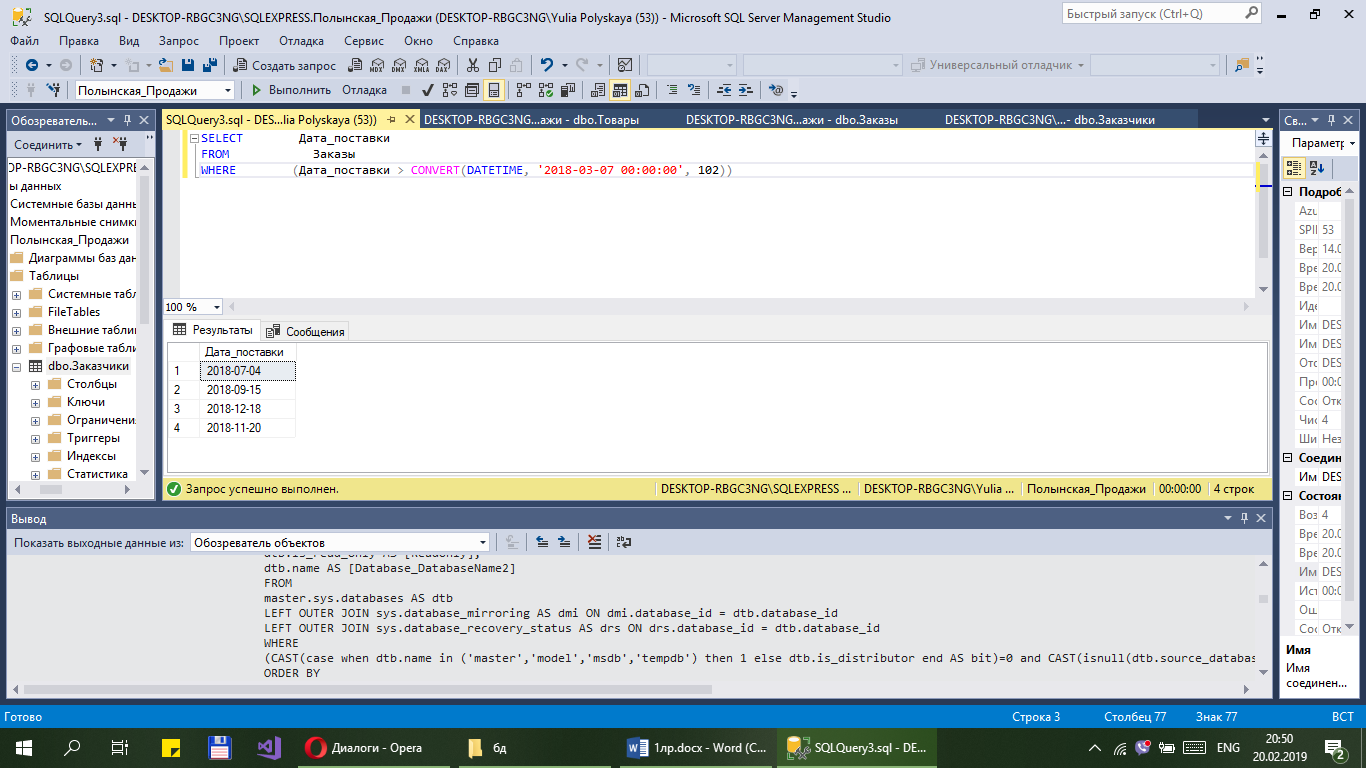
Установить связи между таблицами ЗАКАЗЧИКИ и ЗАКАЗЫ по полям **Наименование\_фирмы** и **Заказчик.**

Установить связи между таблицами ТОВАРЫ и ЗАКАЗЫ по полям **Наименование** и **Наименование\_ товара**.

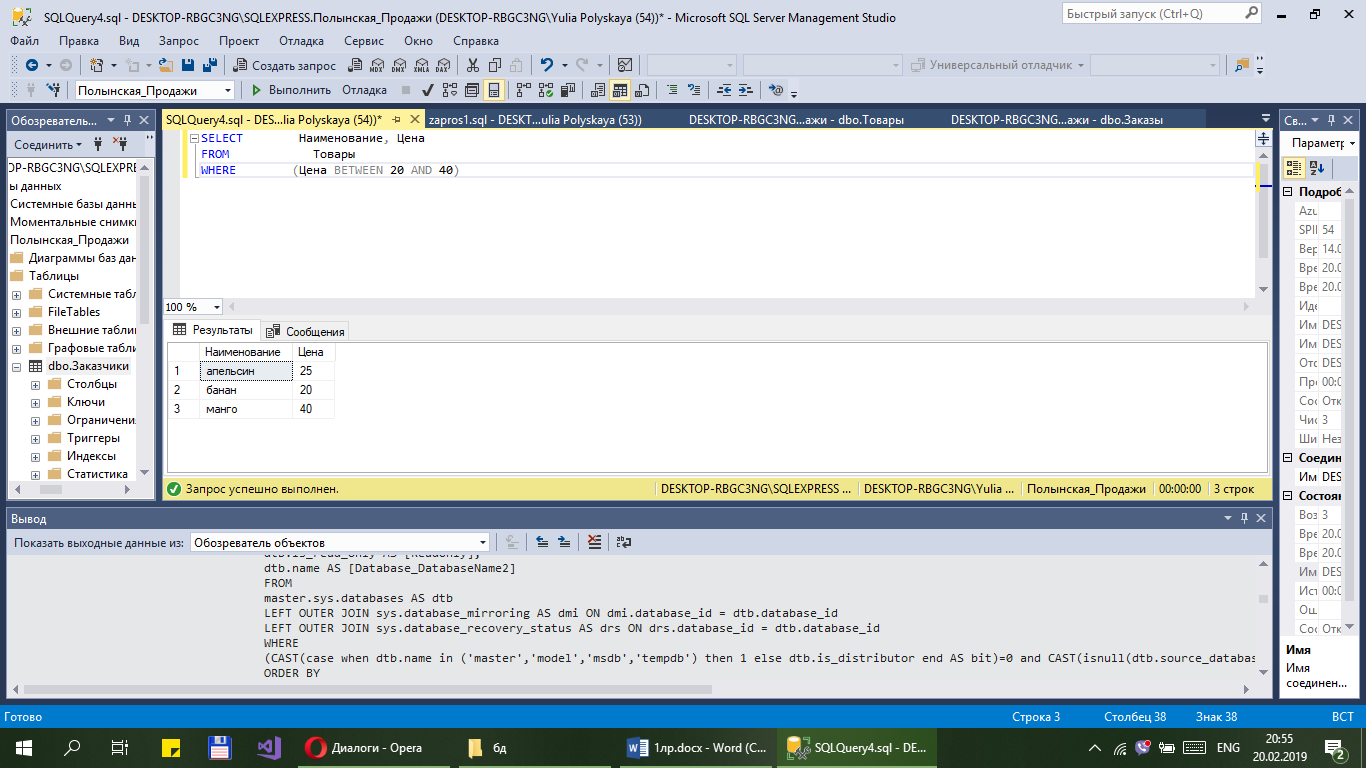


Сформировать следующие запросы и проанализировать результаты:

– определить товары, поставки которых должны осуществиться после некоторой даты;



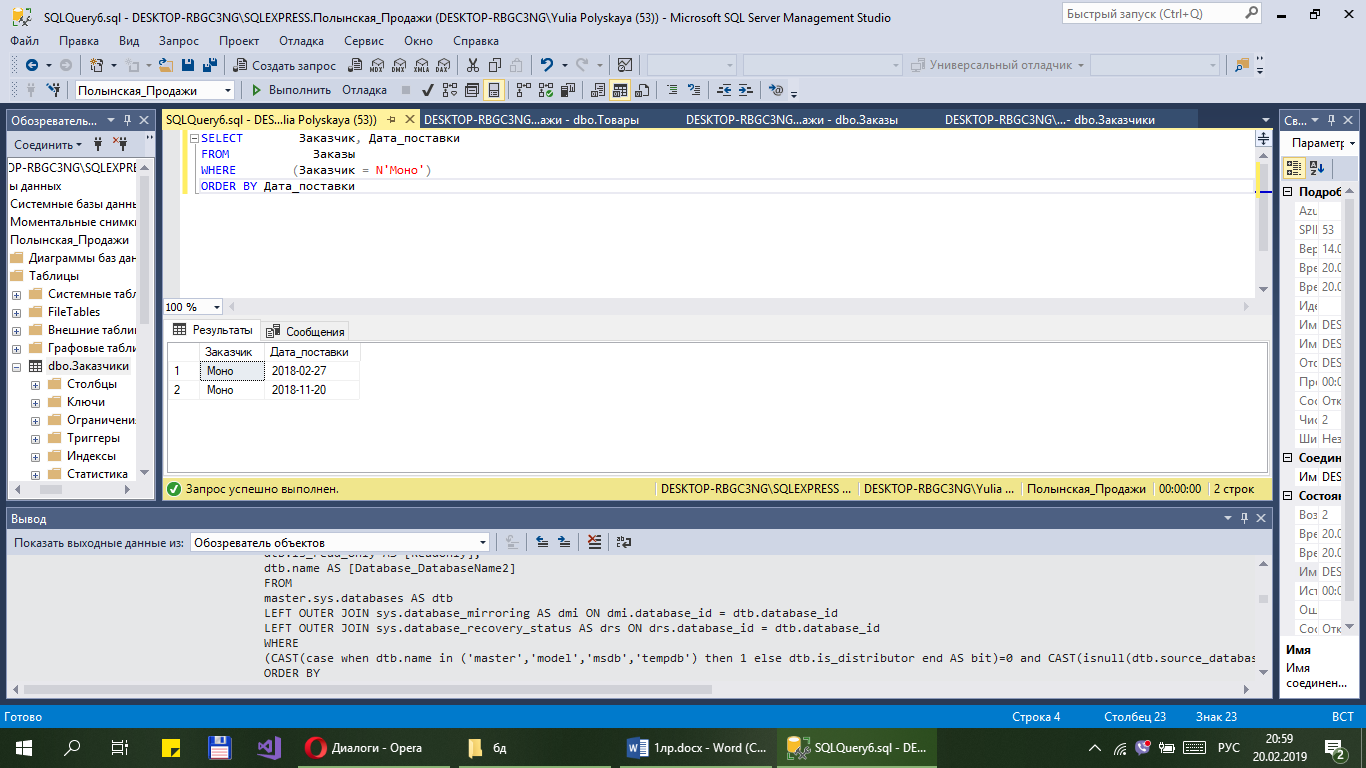
– найти товары, цена которых находится в некоторых пределах;



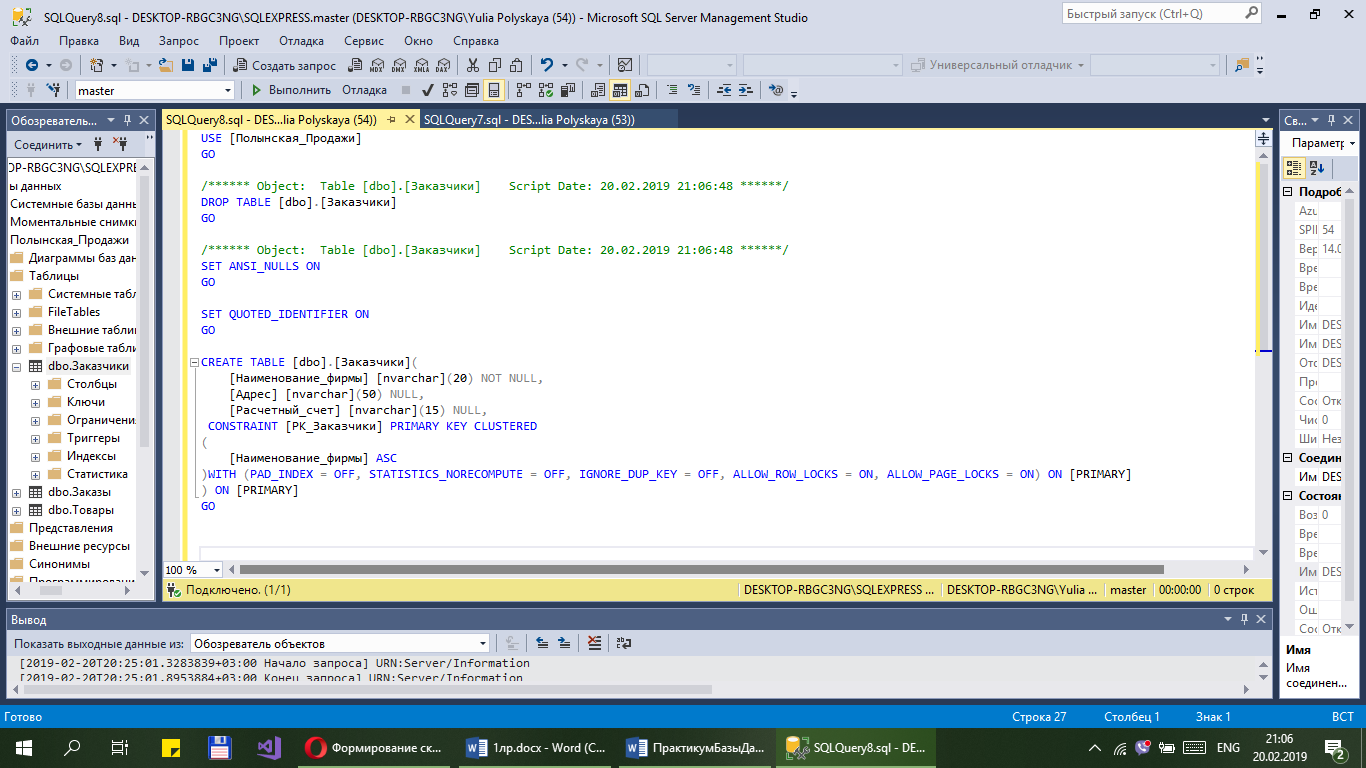
– определить названия фирм, заказавших конкретный товар;

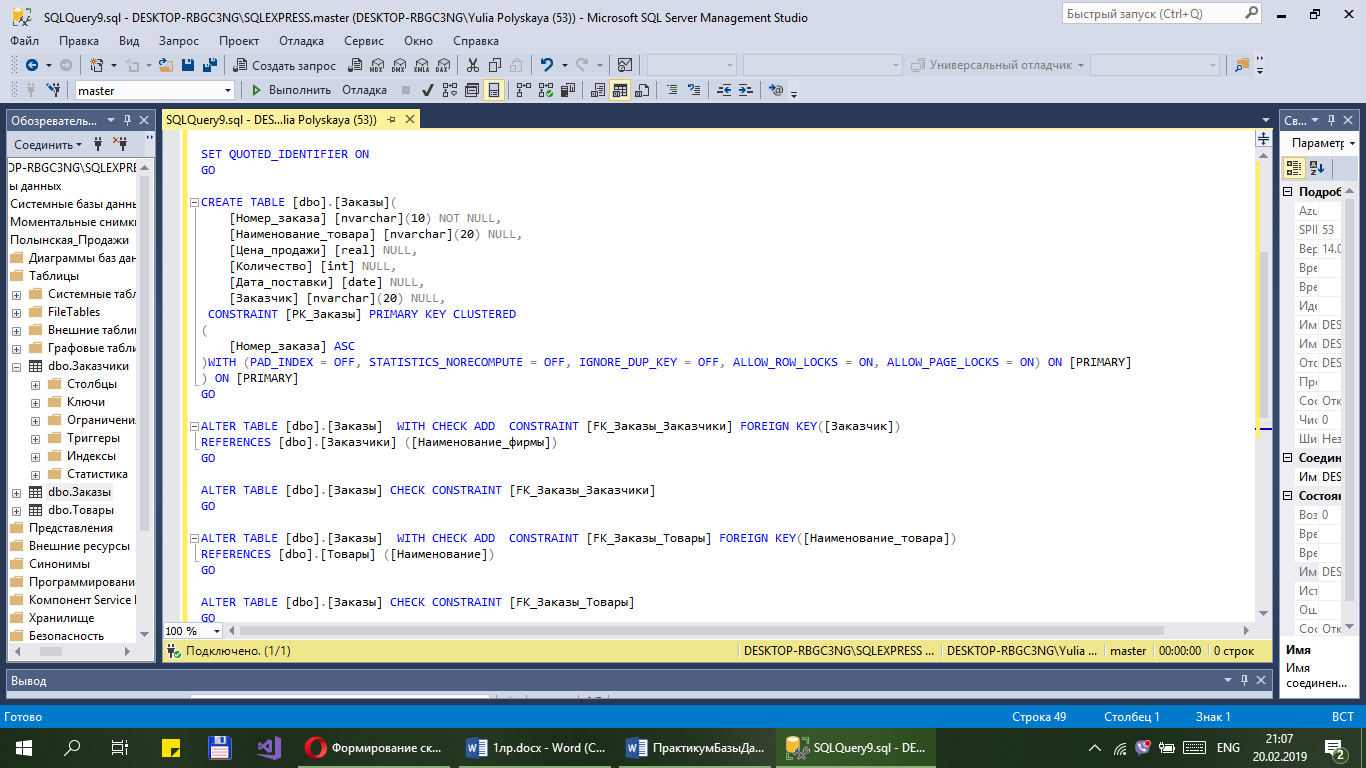


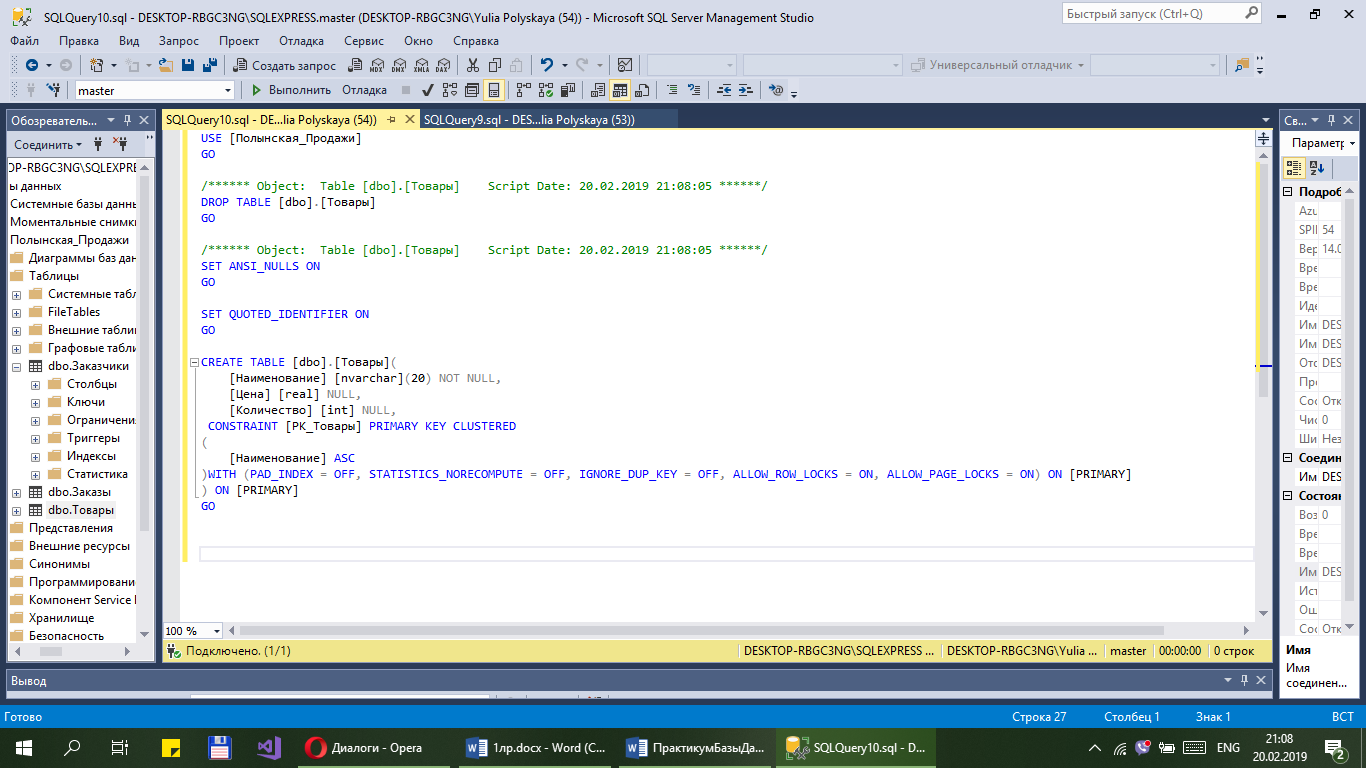
– найти заказы определенной фирмы по ее названию, отсортировать их по датам поставки.



Скрипты:







## **Лабораторная работа №2**

**Вариант 3.**

**Задание.**

**Фирма по продаже запчастей.** Организация занимается продажей запасных частей для автомобилей. Некоторые из поставщиков могут поставлять одинаковые детали (один и тот же артикул). Информационные поля: Код поставщика, Название, Адрес, Телефон, Название детали, Количество деталей на складе, Артикул, Цена, Примечание, Количество заказанных деталей, Дата заказа.

**Выполнение**:

Таблица ПОСТАВЩИКИ:



Таблица ЗАКАЗЧИКИ:

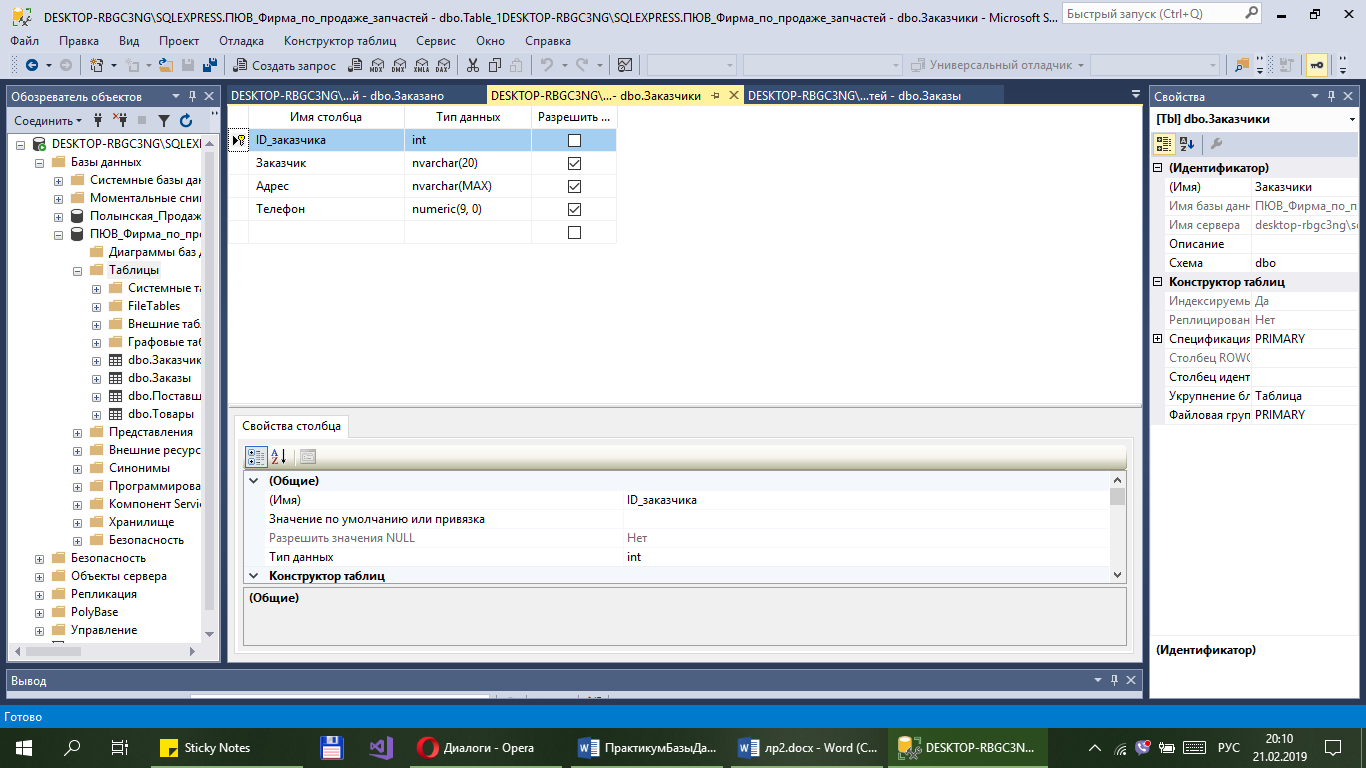


Таблица ТОВАРЫ:

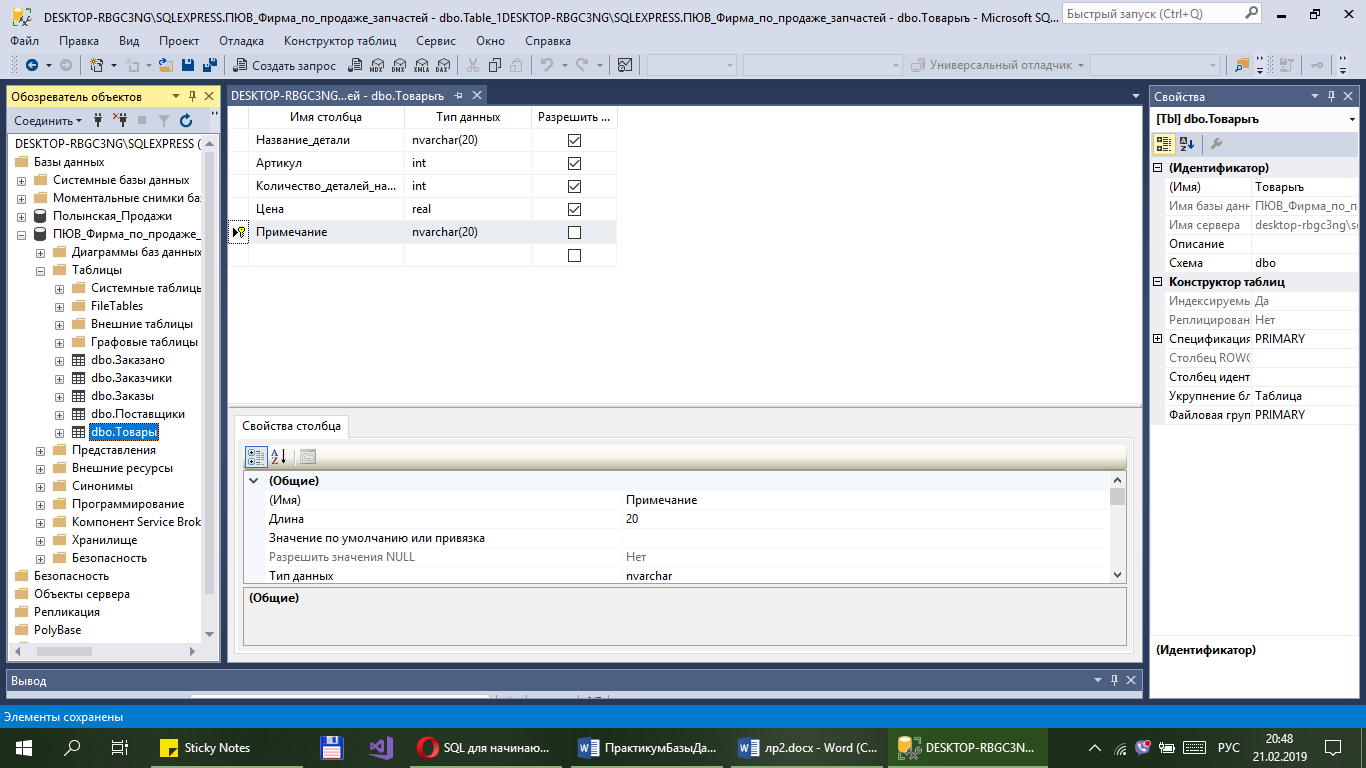


Таблица ЗАКАЗЫ:



Таблица ЗАКАЗАНО:

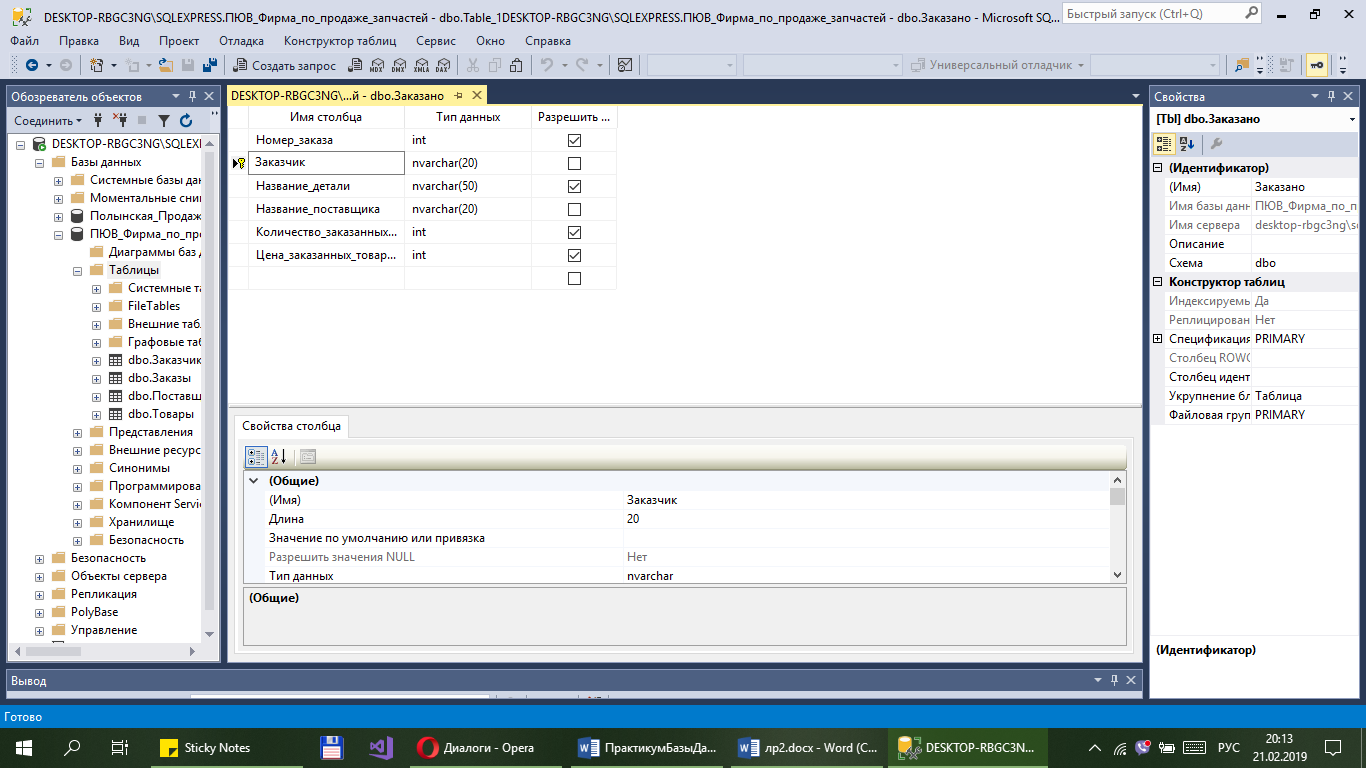
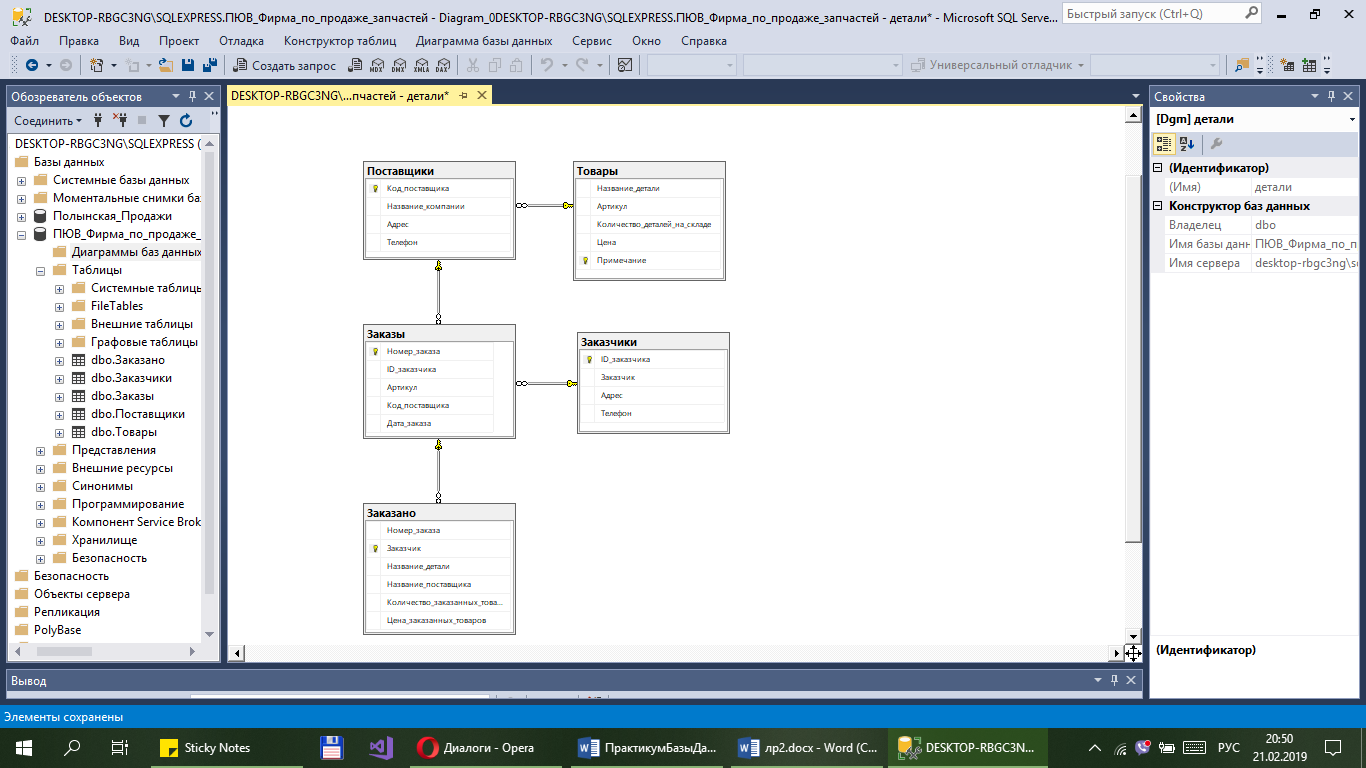


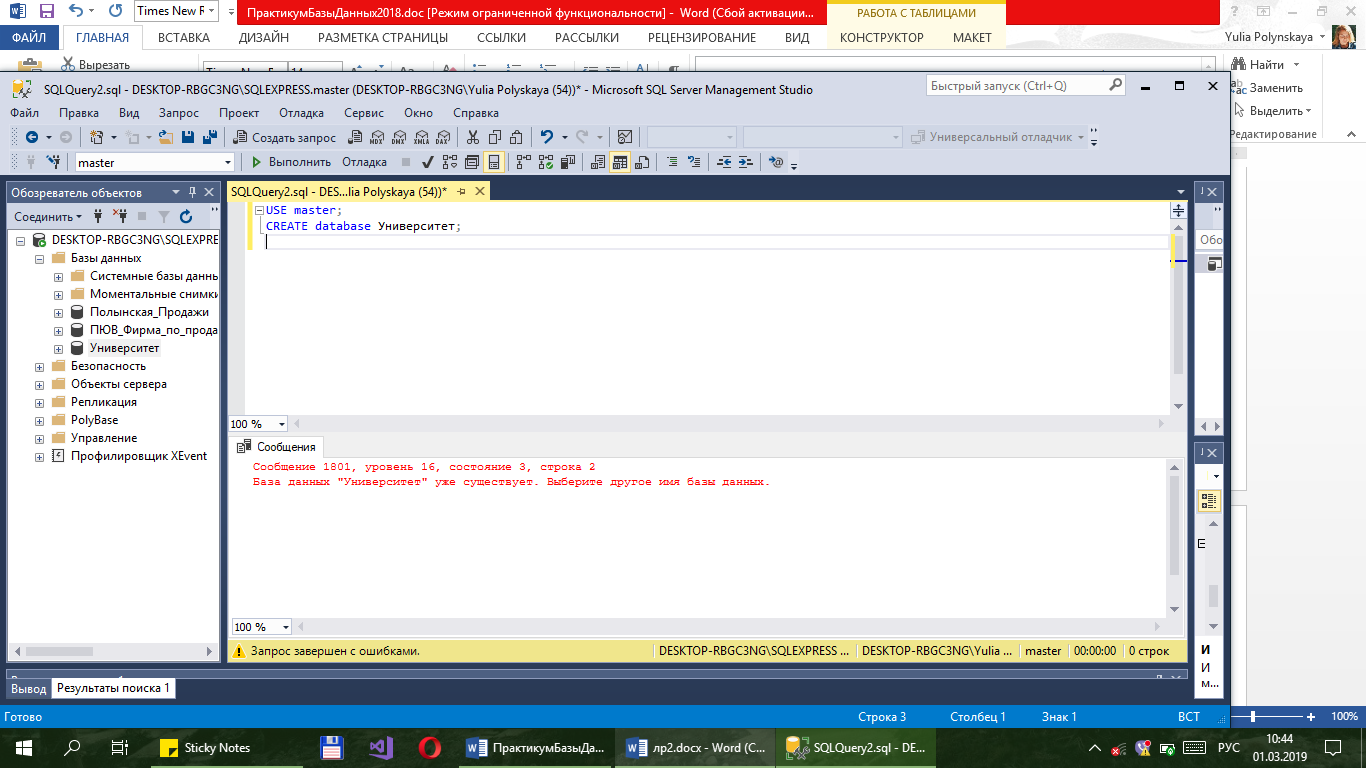
Диаграмма:



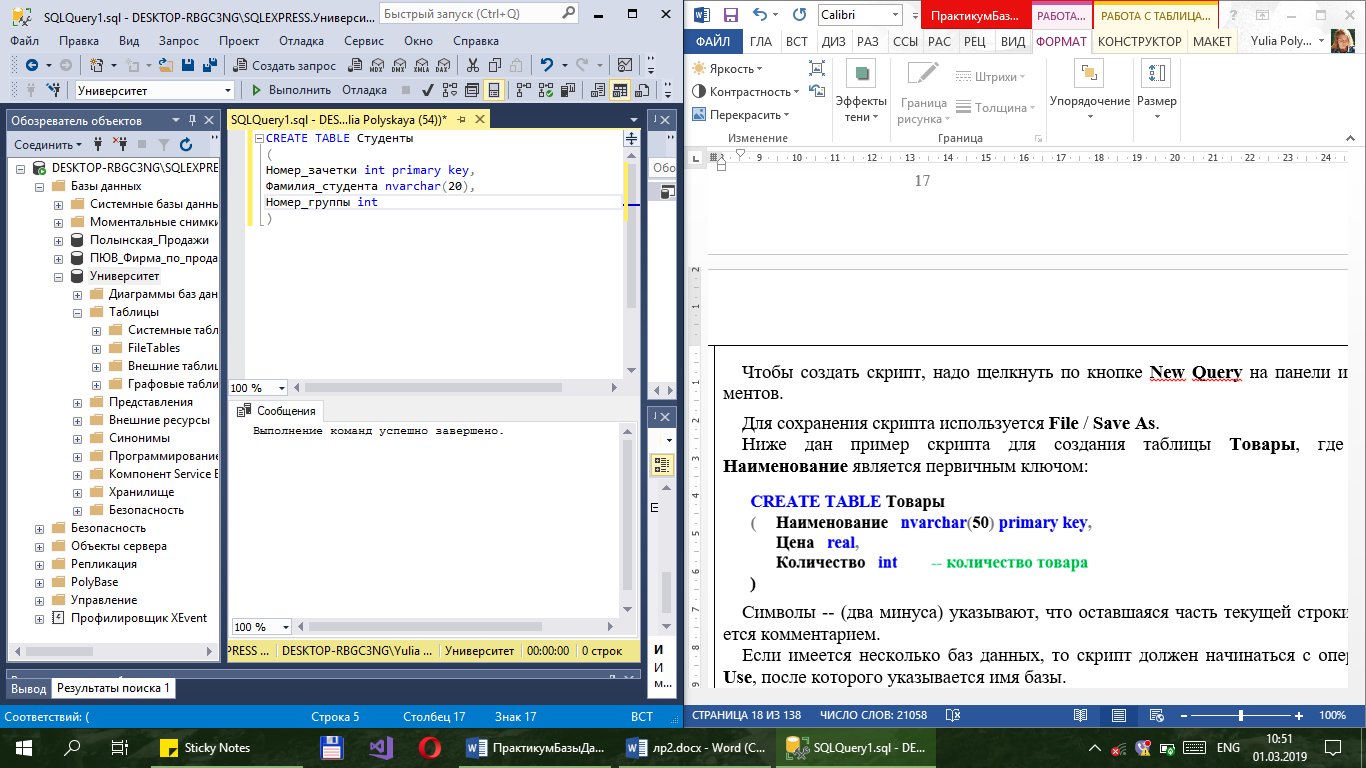
## **Лабораторная работа №3**

T-SQL − язык реляционной базы данных

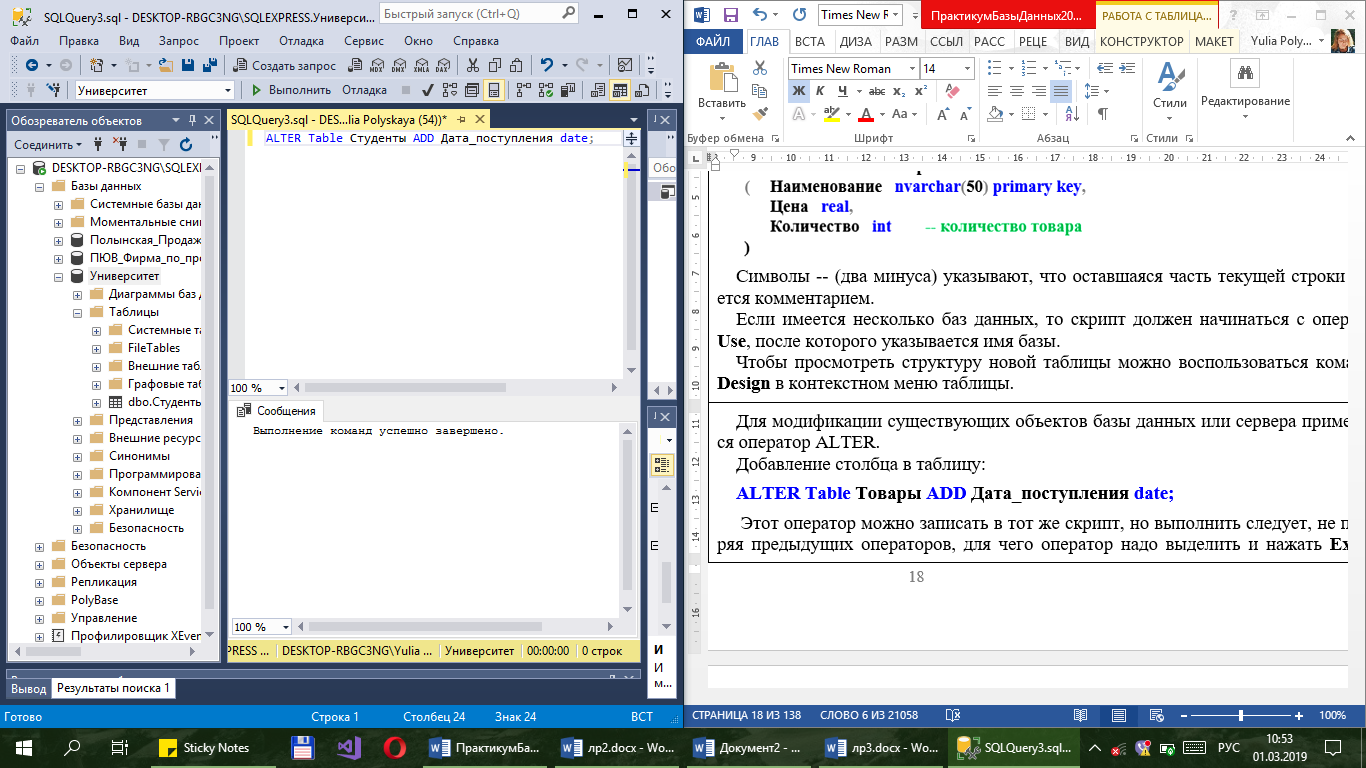
Создать с помощью языка **SQL** базу данных с именем, которое состоит из фамилии студента и слова **UNIVER**.



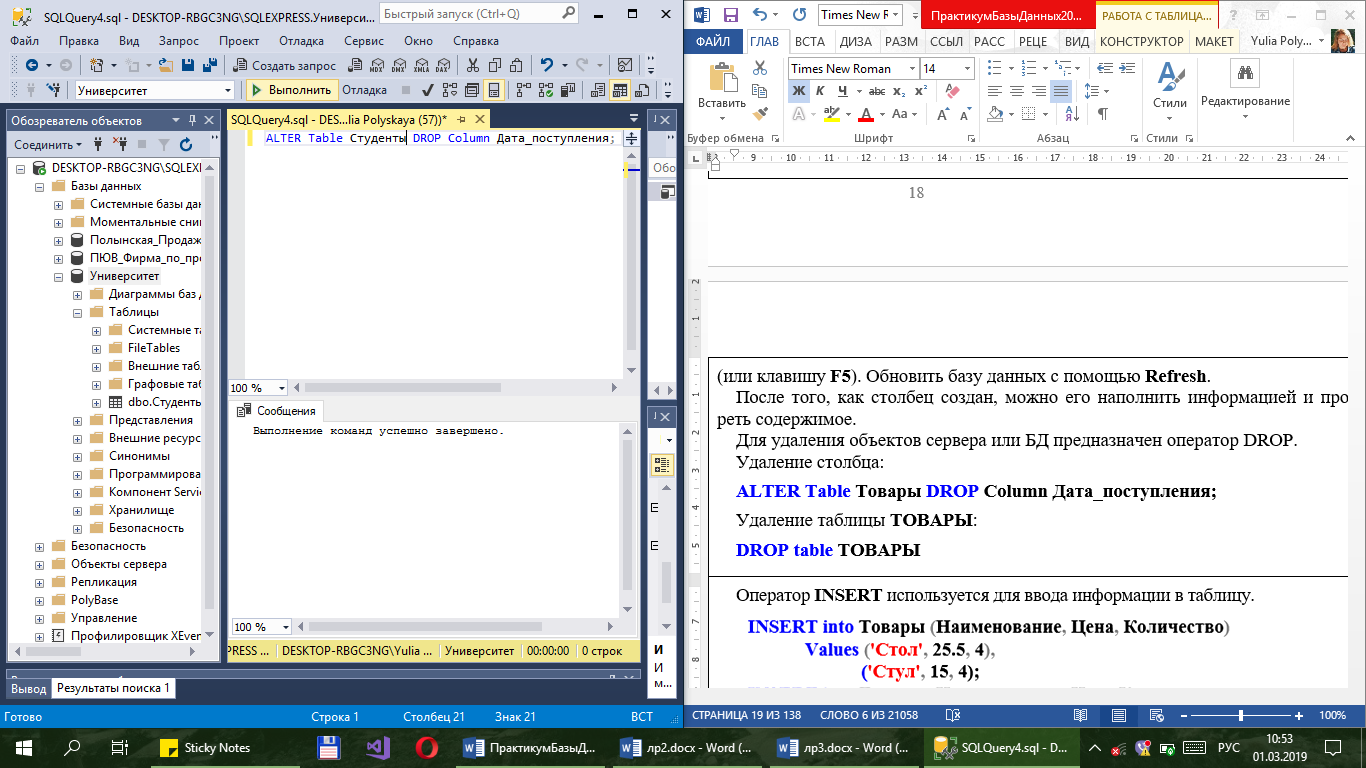
Создать скрипт для формирования таблицы **STUDENT**, содержащей три столбца (номер зачетки, фамилию студента, номер группы) с помощью оператора **CREATE**



Внести изменения в таблицу с помощью оператора **ALTER**

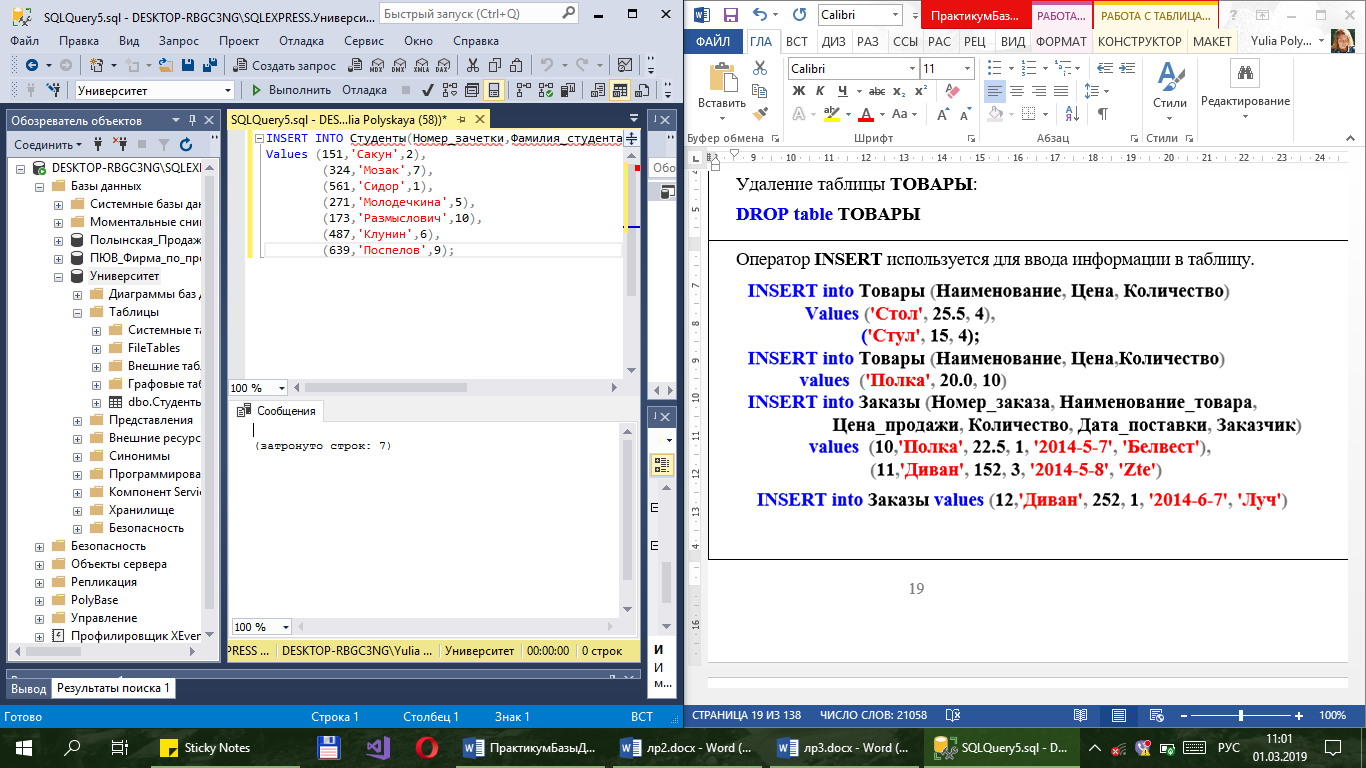


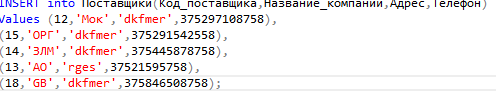


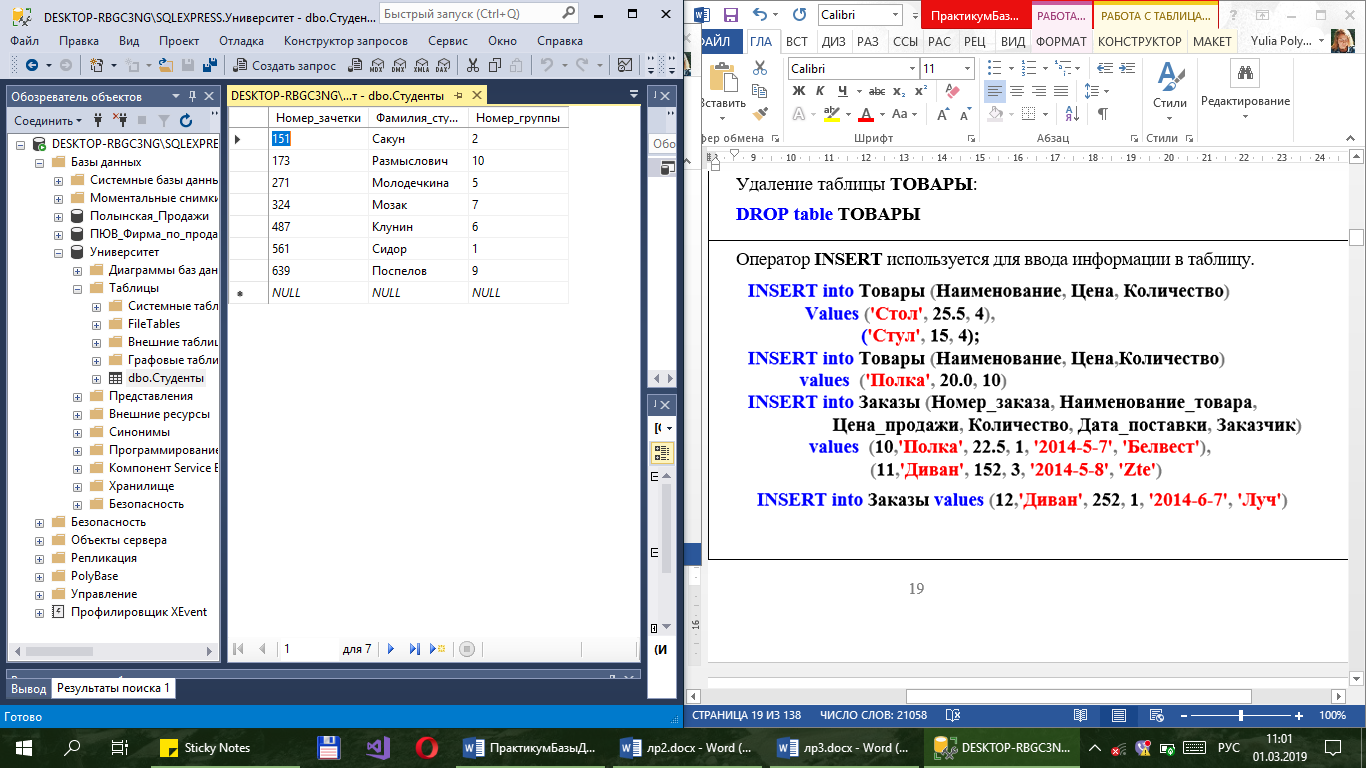




С помощью оператора **INSERT** ввести семь строк в таблицу **STUDENT**







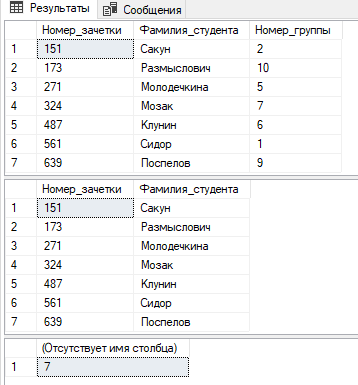
Использовать оператор **SELECT** для выбора всех строк и столбцов таблицы **STUDENT**.

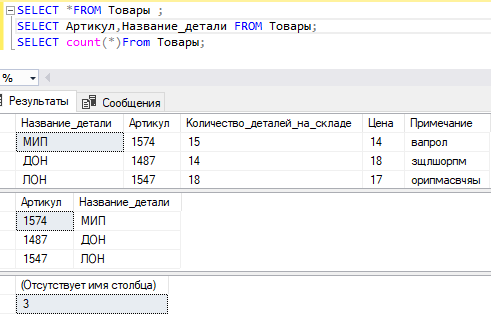
Написать оператор **SELECT**, выбирающий все строки для одного столбца **NAME**.

Написать оператор **SELECT**, выбирающий все строки для двух столбцов (**NZach, NAME**) таблицы.

Подсчитать количество строк в таблице.







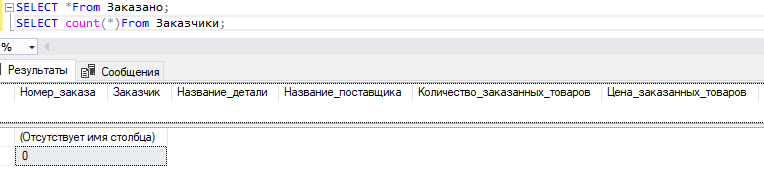
С помощью **SQL Server Management Studio** удалить таблицу **STUDENT** и создать заново

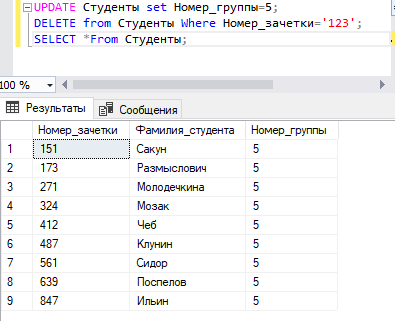


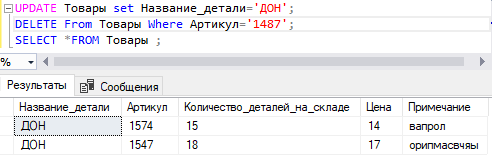
С помощью оператора **UPDATE** изменить значение столбца **GRUP** на **5** во всех записях таблицы. Удалить строку, в которой значение столбца **NZach** равно одному из введенных чисел.



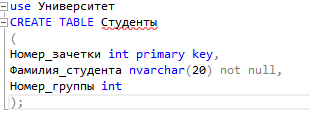


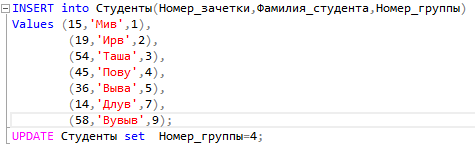


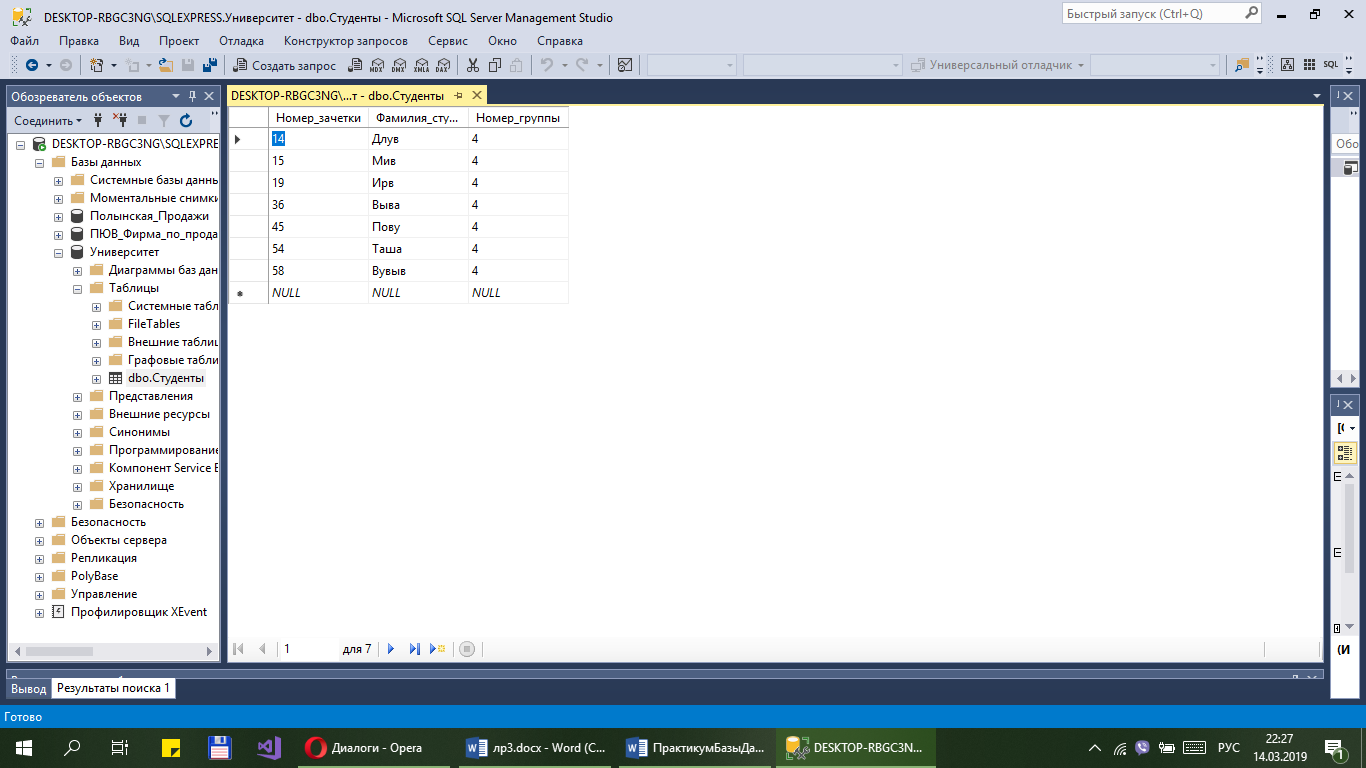




С помощью оператора **CREATE** создать таблицу **STUDENT**,в которойстолбец **NZach** первичный ключ, столбец **NAME** не может принимать значение **NULL**.







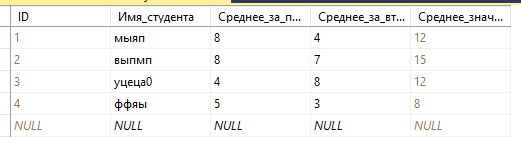
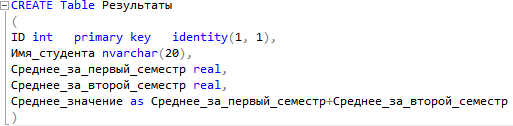
Удалить таблицу **STUDENT**. Создать ее снова, но добавить новый столбец с именем **POL** (пол) с ограничением **CHECK**.Продемонстрировать действие ограничений с помощью операторов **INSERT** и **UPDATE**. Удалить таблицу **STUDENT**. 





Создать таблицу **RESULTS,** которая будет содержать поля **ID (**использовать **IDENTITY), STUDENT\_NAME, AVER\_VALUE (**среднее значение**)**. Поле, содержащее среднее значение, сделать вычисляемым на основе полей результатов учащегося по двум или более предметам. Продемонстрировать вставку значений в такую таблицу и результаты.

**История транзакций,базу можно откатить назад**

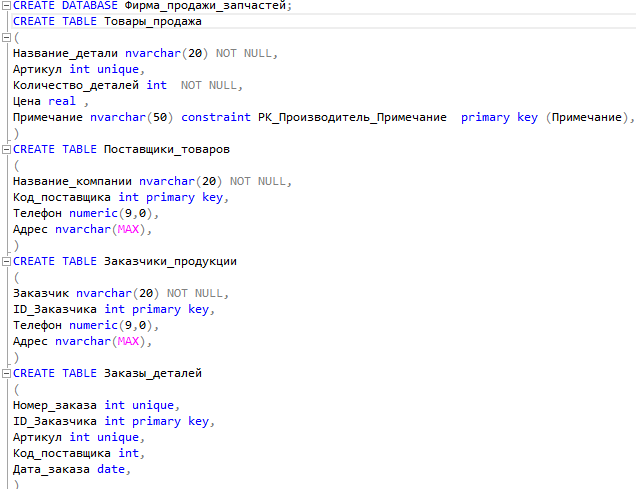


## **Лабораторная работа №4**

**. Настройка базы данных при ее создании**

Разработать аналогичные сценарии для создания базы данных **Фирма по продаже запчастей.**

Разработать сценарий на языке T-SQL для создания и заполнения таблиц. Использовать нужные ограничения целостности из тех, что приведены в таблице справа.

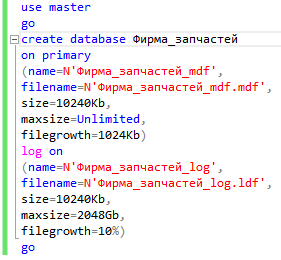


Внести изменения в сценарий

БД представляет собой набор файлов операционной системы трех типов: первичный файл (расширение **mdf**), вторичные файлы (**ndf**) и файлы журнала транзакций (**log**).

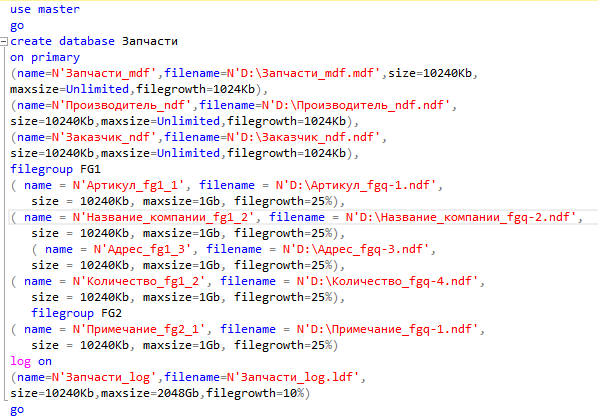
Все файлы БД, кроме файлов журнала транзакций, распределены по файловым группам. Файловые группы – это поименованный набор файлов БД.

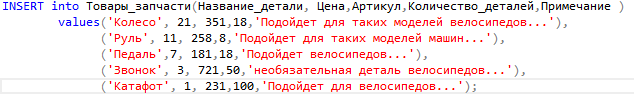
Файловые группы используются в основном для упрощения администрирования БД. Существуют операторы (например, копирования и восстановления БД), позволяющие рассматривать файловую группу как единое целое и выполнять операции не для каждого файла по отдельности, а сразу для файловой группы.



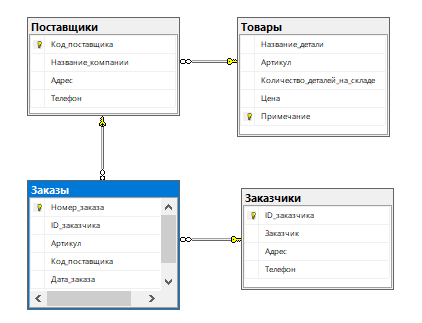
Откорректировать сценарий п. 4 таким образом, чтобы созданные таблицы располагались в файловых группах в соответствии с таблицей:

|  |  |
| --- | --- |
| PRIMARY | FACULTY |
| AUDITORIUM\_TYPE |
| PROFESSION |
| G1 | PULPIT |
| SUBJECT |
| GROUPS |
| TEACHER |
| G2 | AUDITORIUM |
| Файловая группа | Таблица |





С помощью **SSMS** построить диаграмму БД



## **Лабораторная работа №5**

**. Использование оператора SELECT**

Оператор SELECT позволяет выбрать информацию из одной или нескольких таблиц. Оператор состоит из нескольких секций: SELECT, INTO, FROM, WHERE, GROUP BY, HAVING, ORDER BY.

Кроме того, в секции SELECT могут использоваться опции TOP и/или DISTINCT, а в секции ORDER BY опции ASC (возрастание) или DESC (убывание).

В первой строке в результате выполнения оператора SELECT будет сформирован результирующий набор, состоящий из всех строк таблицы **Товары**, причем строки будут содержать все столбцы этой таблицы.

Второй оператор SELECT сформирует результирующий набор, состоящий из одного столбца с именем **Название\_детали** таблицы **Товары**.

Третий и четвертый операторы выведут сообщения, состоящие из одной строки, содержимое которой определяется значением выражений, указанных после ключевого слова **SELECT**.

Как правило, секция WHERE содержит логическое выражение, которое вычисляется для каждой строки. В том случае, если выражение принимает истинное значение – строка отбирается.

Вывод наименований товаров, название которых Мобильный.

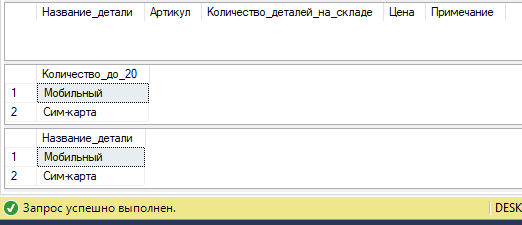
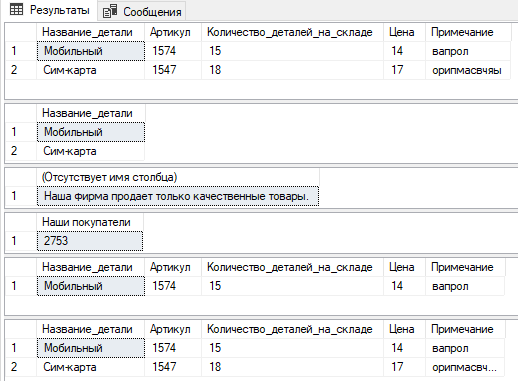
Вывод наименований товаров, цена которых равна 14 или 17:

Вывод наименований товаров, количество которых меньше 15, при этом столбец результатов озаглавлен **Количество\_до\_20**.

Если наименование поля содержит символ пробела, то оно заключается в квадратные скообки

Определяются наименования детали из таблицы **Товары**. Секция DISTINCT позволяет не выводить повторяющиеся строки:





Выводим номера заказов и артикул из таблицы **Заказы**, дата заказа которых не превышает **2016-02-27**. Секция ORDER BY позволяет отсортировать результаты по наименованию товаров.

Вывод первых трех строк из столбца **Название\_детали** таблицы **Товары** (опция TOP ограничивает количество результирующих строк).

Вывод трех строк столбцов **ID\_Заказчика,Артикул** и **Дата\_заказа** из таблицы **Заказы**. Результаты отсортированы по убыванию, повторяющиеся строки не выводятся

В секции WHERE, помимо операций сравнения, логических связок **AND**, **OR** и **NOT**, могут использоваться предикаты (выражения, значениями которых могут быть только «истина» и «ложь»).

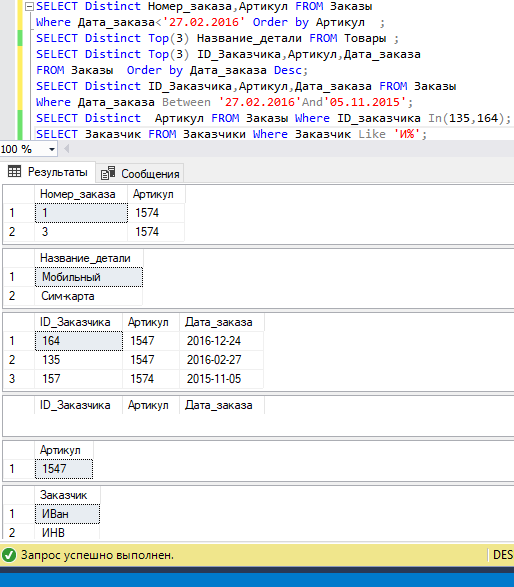
Предикат BETWEEN позволяет выбирать строки с заданным диапазоном значений.

Вывести **ID\_Заказчика,Артикул** и **Дата\_заказа**, которые должны быть доставлены между **2015-11-05** до **2016-02-27**.

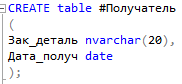
Ключевое слово LIKE осуществляет сравнение полей с заданным шаблоном, при этом в шаблоне символ **%** обозначает любое количество любых символов, ключевое слово IN отбирает строки, в которых значение анализируемого поля совпадает с одним из значений, записанных справа от IN в круглых скобках.

Вывод **Артикул**, **ID\_Заказчика** которых равна 135 или 164. Повторяющиеся наименования не выводятся.

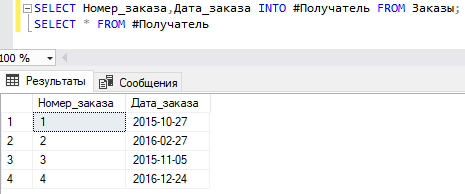
Вывод товаров, наименования которых начинаются с буквы **И.**



Локальные временные таблицы имеют имена, начинающиеся с символа # и доступны только пользователю, ее создавшему



С помощью оператора **SELECT** перенести данные во временную таблицу. Основное отличие временных таблиц от постоянных в том, что они хранятся в системной базе данных TEMPDB и не могут иметь внешние ключи.



## **Лабораторная работа №6**

**Многотабличные SELECT-запросы**

1.На основе таблиц **AUDITORIUM\_ TYPE** и **AUDITORIUM** сформировать перечень кодов аудиторий (столбец **AUDITORUM.AUDITORIUM**) и соответствующих им наименований типов аудиторий (столбец **AUDITORIUM\_ TYPE.AUDITORIUM\_ TYPENAME**). Примечание: использовать соединение таблиц INNER JOIN.

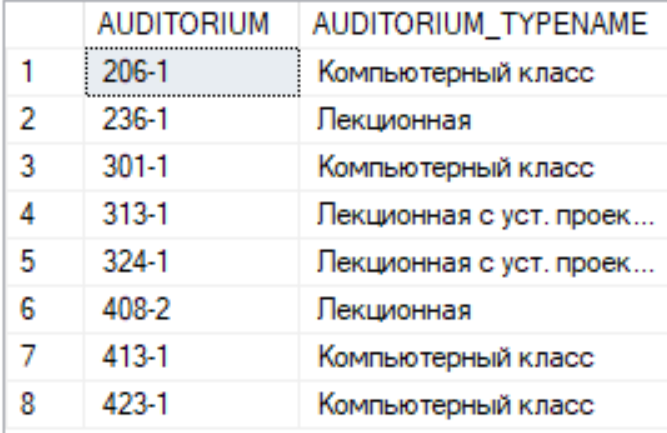
SELECT AUDITORIUM.AUDITORIUM, AUDITORIUM\_TYPE.AUDITORIUM\_TYPENAME

FROM AUDITORIUM

INNER JOIN

AUDITORIUM\_TYPE

ON AUDITORIUM.AUDITORIUM\_TYPE = AUDITORIUM\_TYPE.AUDITORIUM\_TYPE;



2.На основе таблиц **AUDITORIUM\_TYPE** и **AUDITORIUM** сформировать перечень кодов аудиторий (столбец **AUDITORIUM.AUDITORIUM**) и соответствующих им наименований типов аудиторий (столбец **AUDITORIUM\_ TYPE.AUDITO-RIUM\_TYPENAME**).

При этом следует выбрать только те аудитории, в наименовании которых присутствует подстрока **компьютер**.

Примечание: использовать соединение таблиц INNER JOIN и предикат LIKE.

SELECT AUDITORIUM.AUDITORIUM, AUDITORIUM\_TYPE.AUDITORIUM\_TYPENAME

FROM AUDITORIUM

INNER JOIN

AUDITORIUM\_TYPE

ON AUDITORIUM.AUDITORIUM\_TYPE = AUDITORIUM\_TYPE.AUDITORIUM\_TYPE

WHERE AUDITORIUM\_TYPENAME LIKE '%компьютер%';

3.Написать два SELECT-запроса, формирующих результирующие наборы аналогичные запросам из заданий 1 и 2, но без применения INNER JOIN.

SELECT AUDITORIUM.AUDITORIUM, AUDITORIUM\_TYPE.AUDITORIUM\_TYPENAME

FROM AUDITORIUM, AUDITORIUM\_TYPE

WHERE AUDITORIUM.AUDITORIUM\_TYPE = AUDITORIUM\_TYPE.AUDITORIUM\_TYPE;

SELECT AUDITORIUM.AUDITORIUM, AUDITORIUM\_TYPE.AUDITORIUM\_TYPENAME

FROM AUDITORIUM, AUDITORIUM\_TYPE

WHERE AUDITORIUM.AUDITORIUM\_TYPE = AUDITORIUM\_TYPE.AUDITORIUM\_TYPE

AND AUDITORIUM\_TYPENAME LIKE '%компьютер%';

4.На основе таблиц **PRORGESS**, **STUDENT**, **GROUPS**, **SUBJECT**, **PULPIT** и **FACULTY** сформировать перечень студентов, получивших экзаменационные оценки (столбец **PROGRESS.NOTE**) от 6 до 8.

Результирующий набор должен содержать столбцы: **Факультет**, **Кафедра**, **Специальность**, **Дисциплина**, **Имя Студента**, **Оценка**. В столбце **Оценка** должны быть записаны экзаменационные оценки прописью: **шесть**, **семь**, **восемь**.

SELECT FACULTY.FACULTY\_NAME [Факультет], PULPIT.PULPIT\_NAME [Кафедра],

PROFESSION.PROFESSION\_NAME [Специальность], PROGRESS.SUBJECT [Дисциплина],

STUDENT.NAME [Имя студента], PROGRESS.NOTE [Оценка],

CASE

WHEN (PROGRESS.NOTE = 6) then 'шесть'

WHEN (PROGRESS.NOTE = 7) then 'семь'

WHEN (PROGRESS.NOTE = 8) then 'восемь'

END [ОценкаС]

FROM

PROGRESS

INNER JOIN STUDENT

ON PROGRESS.IDSTUDENT = STUDENT.IDSTUDENT,

FACULTY

INNER JOIN PULPIT

ON PULPIT.FACULTY = FACULTY.FACULTY

INNER JOIN PROFESSION

ON FACULTY.FACULTY = PROFESSION.FACULTY

WHERE PROGRESS.NOTE BETWEEN 6 AND 8

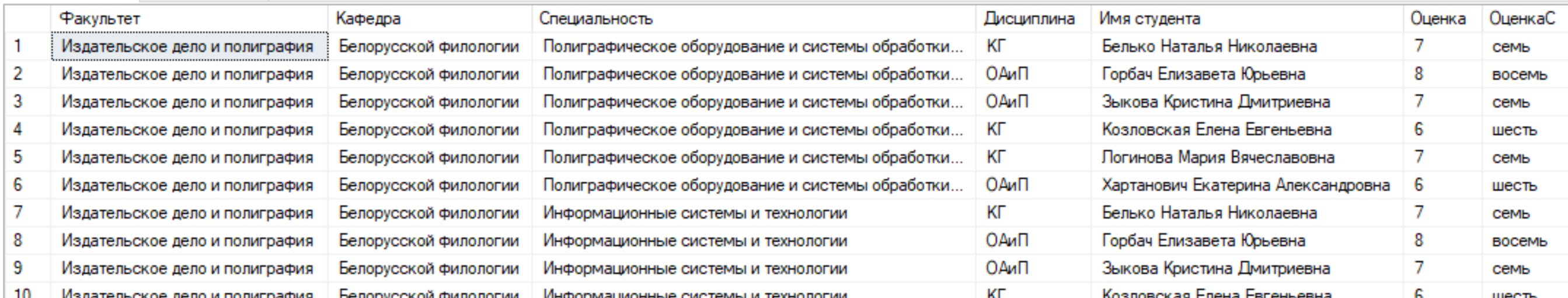
ORDER BY FACULTY.FACULTY,

PULPIT.PULPIT,

PROFESSION.PROFESSION,

STUDENT.NAME ASC,

PROGRESS.NOTE DESC;



6.На основе таблиц **PULPIT** и **TEACHER** получить полный перечень кафедр (столбец **PULPIT.PULPIT\_ NAME)** и преподавателей (столбец **TEACHER.TEA-CHER\_NAME**) на этих кафедрах. Результирующий набор должен содержать два столбца: **Кафедра** и **Преподаватель**. Если на кафедре нет преподавателей, то в столбце **Преподаватель** должна быть выведена строка **\*\*\*.**

Примечание: использовать соединение таблиц LEFT OUTER JOIN и функцию **isnull**.

SELECT FACULTY.FACULTY\_NAME [Факультет], PULPIT.PULPIT\_NAME [Кафедра],

PROFESSION.PROFESSION\_NAME [Специальность], PROGRESS.SUBJECT [Дисциплина],

STUDENT.NAME [Имя студента], PROGRESS.NOTE [Оценка],

CASE

WHEN (PROGRESS.NOTE = 6) then 'шесть'

WHEN (PROGRESS.NOTE = 7) then 'семь'

WHEN (PROGRESS.NOTE = 8) then 'восемь'

END [ОценкаС]

FROM

PROGRESS

INNER JOIN STUDENT

ON PROGRESS.IDSTUDENT = STUDENT.IDSTUDENT,

FACULTY

INNER JOIN PULPIT

ON PULPIT.FACULTY = FACULTY.FACULTY

INNER JOIN PROFESSION

ON FACULTY.FACULTY = PROFESSION.FACULTY

WHERE PROGRESS.NOTE BETWEEN 6 AND 8

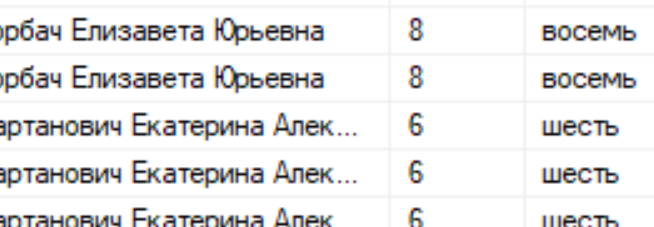
ORDER BY (CASE

WHEN (PROGRESS.NOTE = 6) THEN 3

WHEN (PROGRESS.NOTE = 8) THEN 2

WHEN (PROGRESS.NOTE = 7) THEN 1

END);



**7.** В запросе, реализующем пункт 6, поменять порядок таблиц в выражении LEFT OUTER JOIN. Переписать запрос таким образом, чтобы получился аналогичный результат, но применялось соединение таблиц RIGHT OUTER JOIN.

SELECT PULPIT.PULPIT\_NAME [Кафедра], ISNULL(TEACHER.TEACHER\_NAME, '\*\*\*') [Преподаватель]

FROM TEACHER

LEFT OUTER JOIN PULPIT

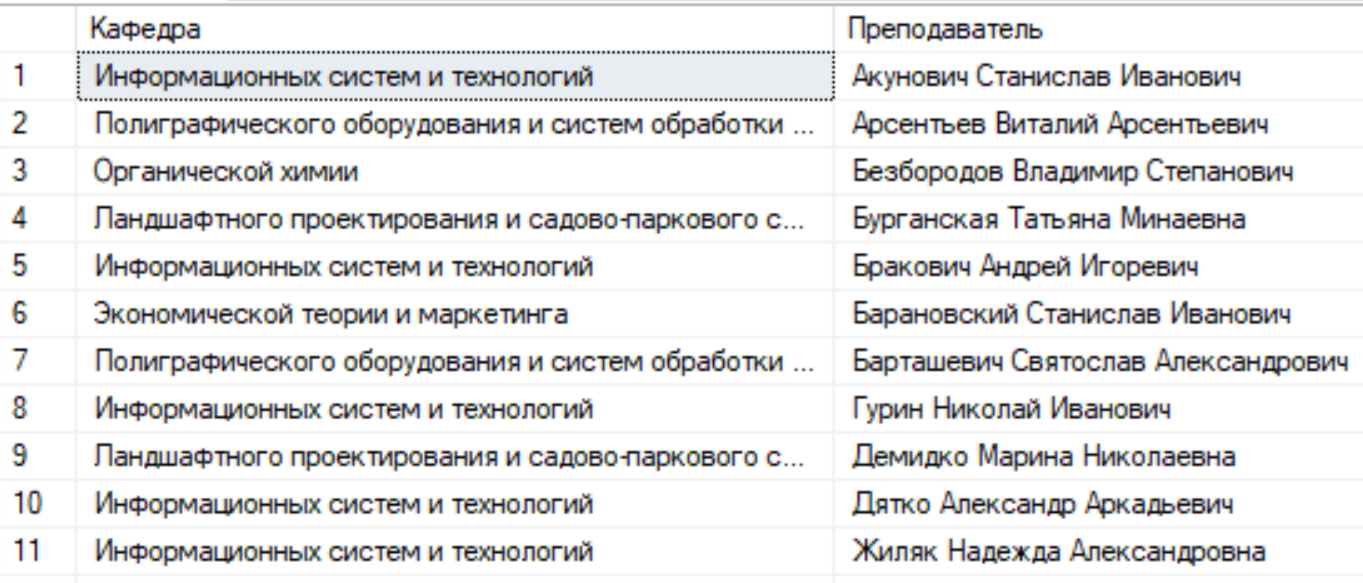
ON PULPIT.PULPIT = TEACHER.PULPIT;

SELECT PULPIT.PULPIT\_NAME [Кафедра], ISNULL(TEACHER.TEACHER\_NAME, '\*\*\*') [Преподаватель]

FROM PULPIT

RIGHT OUTER JOIN TEACHER

ON PULPIT.PULPIT = TEACHER.PULPIT;



8.Показать на примере, что соединение FULL OUTER JOIN двух таблиц:

− является коммутативной операцией;

− является объединением LEFT OUTER JOIN и RIGHT OUTER JOIN соединений этих таблиц;

− включает соединение INNER JOIN этих таблиц.

Примечание: создать две таблицы, заполнить их данными. Разработать SELECT-запросы, реализующие задания.

Создать три новых запроса:

− запрос, результат которого содержит данные левой (в операции FULL OUTER JOIN) таблицы и не содержит данные правой;

− запрос, результат которого содержит данные правой таблицы и не содержащие данные левой;

− запрос, результат которого содержит данные правой таблицы и левой таблиц;

Примечание: использовать в запросах выражение IS NULL и IS NOT NULL.

CREATE DATABASE TEST;

USE TEST;

CREATE TABLE T1

(

T1F1 NVARCHAR(50),

T1F2 INT

);

CREATE TABLE T2

(

T2F1 NVARCHAR(50),

T2F2 INT

);

INSERT INTO T1

VALUES ('111', 111),

('222', 222),

('333', 333),

('444', 444),

('555', 555);

INSERT INTO T2

VALUES ('111', 111),

('222', 22),

('333', 33),

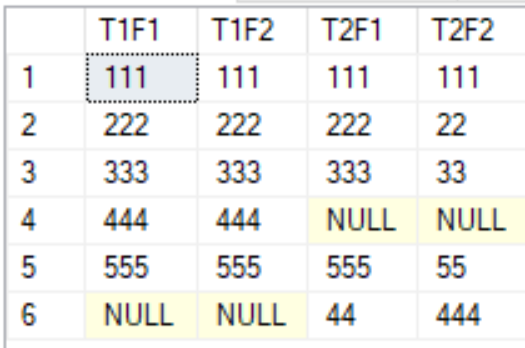
('44', 444),

('555', 55);

SELECT \* FROM T1

FULL OUTER JOIN T2

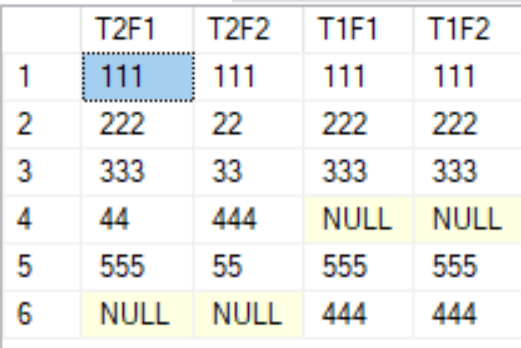
ON T1F1 = T2F1;



SELECT \* FROM T2

FULL OUTER JOIN T1

ON T1F1 = T2F1;



SELECT \* FROM T1

FULL OUTER JOIN T2

ON T1F2 = T2F2;

SELECT \* FROM T1

LEFT OUTER JOIN T2

ON T1F2 = T2F2;

SELECT \* FROM T1

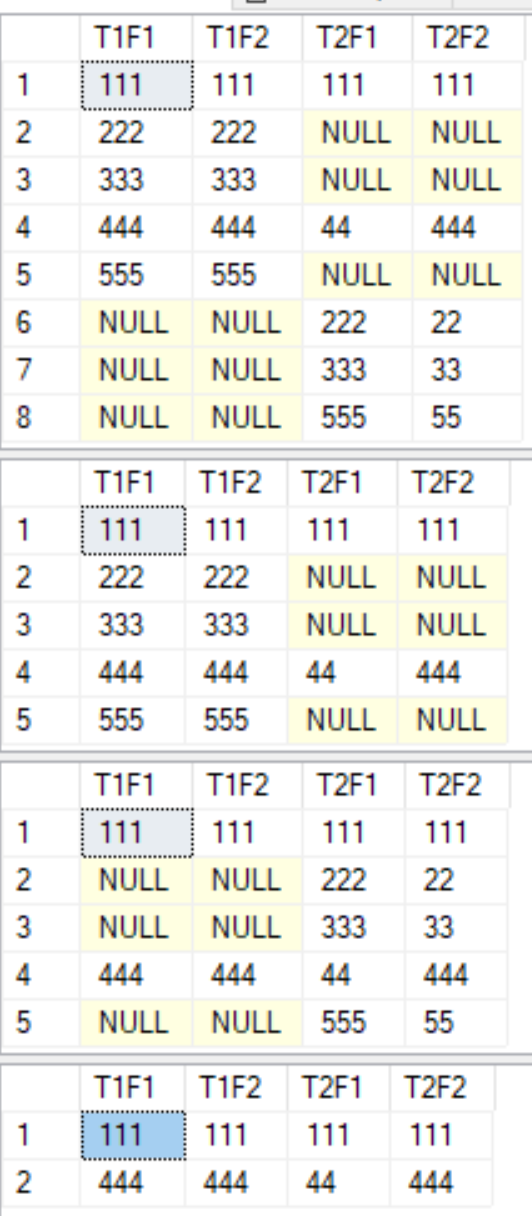
RIGHT OUTER JOIN T2

ON T1F2 = T2F2;

SELECT \* FROM T1

INNER JOIN T2

ON T1F2 = T2F2;



SELECT T1F1, T1F2

FROM T1

FULL OUTER JOIN T2

ON T1F2 = T2F2;

SELECT T2F1, T2F2

FROM T1

FULL OUTER JOIN T2

ON T1F2 = T2F2;

SELECT \*

FROM T1

FULL OUTER JOIN T2

ON T1F2 = T2F2;



SELECT ISNULL(T1F1, '\*\*\*'), ISNULL(T1F2, '\*\*\*')

9.Разработать SELECT-запрос на основе CROSS JOIN-соединения таблиц **AUDITORIUM\_TYPE** и **AUDITORIUM**, формирующего результат, аналогичный результату, полученному при выполнении запроса в задании 1.

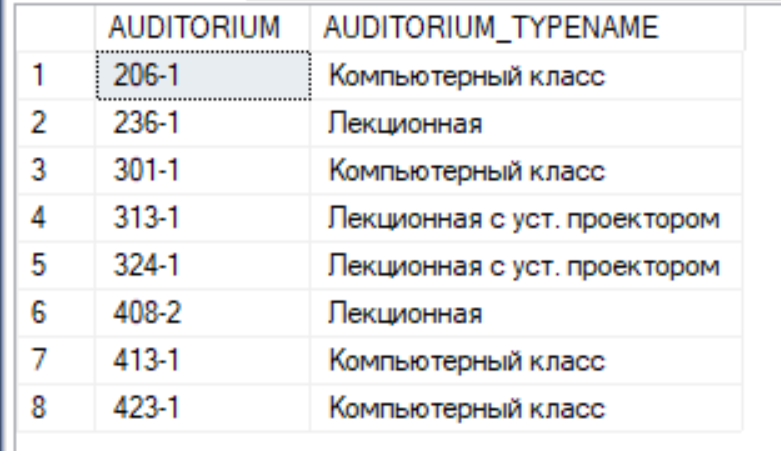
USE UNIVER

SELECT AUDITORIUM.AUDITORIUM, AUDITORIUM\_TYPE.AUDITORIUM\_TYPENAME

FROM AUDITORIUM

CROSS JOIN AUDITORIUM\_TYPE

WHERE AUDITORIUM.AUDITORIUM\_TYPE = AUDITORIUM\_TYPE.AUDITORIUM\_TYPE;



## **Лабораторная работа №7**

[**Использование подзапросов**](#Лаб7)

**1.** На основе таблиц **FACULTY**, **PULPIT** и **PROFESSION** сформировать список наименований кафедр (столбец **PULPIT\_NAME**), которые находятся на факультете (таблица **FACULTY**), обеспечивающем подготовку по специальности, в наименовании (столбец **PROFESSION\_ NAME**) которого содержится слово ***технология*** или ***технологии***. Примечание: использовать в секции WHERE предикат IN c некоррелированным подзапросом к таблице **PROFESSION**.

SELECT PULPIT.PULPIT\_NAME,FACULTY.FACULTY,PROFESSION.PROFESSION\_NAME

FROM FACULTY INNER JOIN PULPIT

ON FACULTY.FACULTY=PULPIT.FACULTY

INNER JOIN PROFESSION

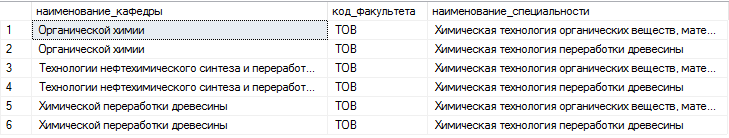
ON PROFESSION.FACULTY=FACULTY.FACULTY

WHERE PROFESSION.PROFESSION\_NAME

IN( SELECT PROFESSION\_NAME FROM PROFESSION

WHERE FACULTY LIKE'%технологии%'

OR FACULTY LIKE'%технология%');



**2.** Переписать запрос пункта 1 таким образом, чтобы тот же подзапрос был записан в конструкции INNER JOIN секции FROM внешнего запроса. При этом результат выполнения запроса должен быть аналогичным результату исходного запроса.

SELECT PULPIT\_NAME,FACULTY.FACULTY,PROFESSION.PROFESSION\_NAME

FROM(SELECT PROFESSION\_NAME FROM PROFESSION

WHERE PROFESSION\_NAME LIKE'%технологии%'

OR PROFESSION\_NAME LIKE'%технология%')AS T

INNER JOIN PROFESSION

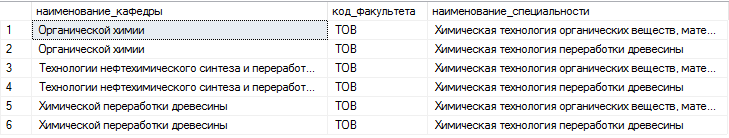
ON T.PROFESSION\_NAME=PROFESSION.PROFESSION\_NAME

INNER JOIN FACULTY

ON PROFESSION.FACULTY=FACULTY.FACULTY

INNER JOIN PULPIT

ON PULPIT.FACULTY=FACULTY.FACULTY;



**3.** Переписать запрос, реализующий 1 пункт без использования подзапроса. Примечание: использовать соединение INNER JOIN трех таблиц.

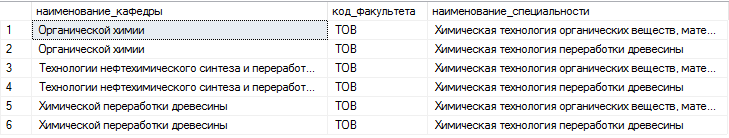
SELECT PULPIT\_NAME,FACULTY.FACULTY,PROFESSION.PROFESSION\_NAME

FROM FACULTY INNER JOIN PULPIT

ON FACULTY.FACULTY=PULPIT.FACULTY

INNER JOIN PROFESSION

ON PROFESSION.PROFESSION\_NAME LIKE'%технологии%'

 OR .PROFESSION\_NAME LIKE'%технология%';

**4.** На основе таблицы **AUDITORIUM** сформировать список аудиторий самых больших вместимостей (столбец **AUDITORIUM\_CAPACITY**) для каждого типа аудитории (**AUDITORIUM\_TYPE**). При этом результат следует отсортировать в порядке убывания вместимости. Примечание: использовать коррелируемый подзапрос c секциями TOP (ограничивает кол-во результирующих строк) и ORDER BY.

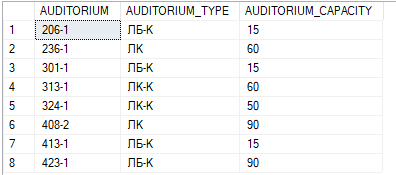
SELECT AUDITORIUM,AUDITORIUM\_TYPE, AUDITORIUM\_CAPACITY FROM AUDITORIUM T1

WHERE AUDITORIUM=(

SELECT TOP(1) AUDITORIUM FROM AUDITORIUM T2

WHERE T1.AUDITORIUM=T2.AUDITORIUM

ORDER BY AUDITORIUM\_CAPACITY DESC);



**5.** На основе таблиц **FACULTY** и **PULPIT** сформировать список наименование факультетов (столбец **FACULTY\_NAME**) на котором нет ни одной кафедры (таблица **PULPIT**). Примечание: использовать предикат EXISTS и коррелированный подзапрос

SELECT FACULTY\_NAME FROM FACULTY

WHERE NOT EXISTS( SELECT \* FROM PULPIT

WHERE FACULTY.FACULTY=PULPIT.FACULTY);



**6.** На основе таблицы **PROGRESS** сформировать строку, содержащую средние значения оценок (столбец **NOTE**) по дисциплинам, имеющим следующие коды: **ОАиП**, **БД** и **СУБД**. Примечание: использовать три некоррелированных подзапроса в списке SELECT; в подзапросах применить агрегатные функции AVG.

SELECT (SELECT AVG(NOTE) FROM PROGRESS WHERE SUBJECT='ОАИП')[ОАИП],

(SELECT AVG(NOTE) FROM PROGRESS WHERE SUBJECT='КГ')[КГ],

(SELECT AVG(NOTE) FROM PROGRESS WHERE SUBJECT='СУБД')[СУБД];



**7.** Разработать SELECT-запрос, демонстрирующий принцип применения ALL совместно с подзапросом.

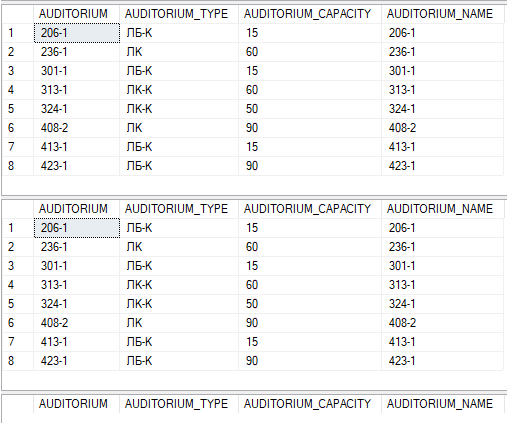
SELECT \* FROM AUDITORIUM

WHERE AUDITORIUM\_CAPACITY >=ALL(SELECT AUDITORIUM\_CAPACITY FROM AUDITORIUM WHERE AUDITORIUM LIKE '%4');

8. Разработать SELECT-запрос, демонстрирующий принцип применения ANY совместно с подзапросом.

SELECT \* FROM AUDITORIUM

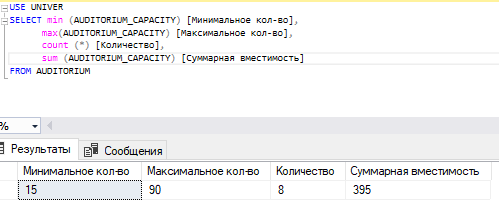
WHERE AUDITORIUM\_CAPACITY >=ANY(SELECT AUDITORIUM\_CAPACITY FROM AUDITORIUM WHERE AUDITORIUM LIKE '%4');



## **Лабораторная работа №8**

**Группировка данных**

1. На основе таблицы **AUDITORIUM** разработать SELECT-запрос, вычисляющий максимальную, минимальную и среднюю вместимость аудиторий, суммарную вместимость всех аудиторий и общее количество аудиторий.



2.На основе таблиц **AUDITORIUM** и **AUDITORIUM\_TYPE** разработать запрос, вычисляющий для каждого типа аудиторий максимальную, минимальную, среднюю вместимость аудиторий, суммарную вместимость всех аудиторий и общее количество аудиторий данного типа.

Результирующий набор должен содержать столбец с наименованием типа аудиторий (столбец **AUDITORIUM\_TYPE.AUDITORIUM\_TYPENAME**) и столбцы с вычисленными величинами. Использовать внутреннее соединение таблиц, секцию GROUP BY и агрегатные функции.

SELECT AUDITORIUM\_TYPENAME,

max(A.AUDITORIUM\_CAPACITY) [мах Вместимость],

min(A.AUDITORIUM\_CAPACITY) [min Вместимость],

avg(A.AUDITORIUM\_CAPACITY) [средняя Вместимость],

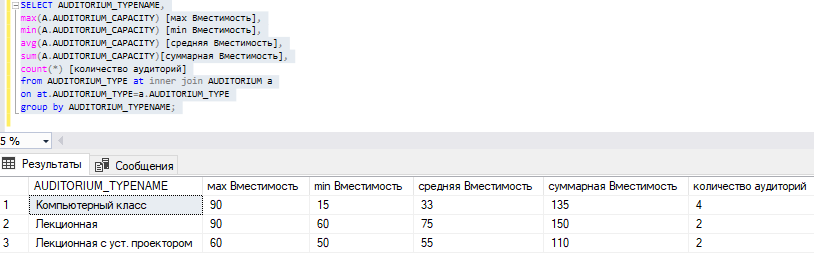
sum(A.AUDITORIUM\_CAPACITY)[суммарная Вместимость],

count(\*) [количество аудиторий]

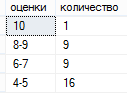
from AUDITORIUM\_TYPE at inner join AUDITORIUM a

on at.AUDITORIUM\_TYPE=a.AUDITORIUM\_TYPE

group by AUDITORIUM\_TYPENAME;



3.Результат запроса на основе таблицы **PROGRESS** содержит количество экзаменационных оценок в заданном интервале. Разработать запрос, формирующий аналогичный результат. При этом учесть, что сортировка строк должна осуществляться в порядке, обратном величине оценки; сумма значений в столбце **количество** должна быть равна количеству строк в таблице **PROGRESS**. Использовать подзапрос в секции FROM, в подзапросе применить GROUP BY, сортировку осуществить во внешнем запросе, в секции GROUP BY, в SELECT-списке подзапроса и в ORDER BY внешнего запроса применить CASE.



SELECT \* FROM

(SELECT CASE

WHEN P.NOTE =10 THEN '10'

WHEN P.NOTE BETWEEN 8 AND 9 THEN '8-9'

WHEN P.NOTE BETWEEN 6 AND 7 THEN '6-7'

WHEN P.NOTE BETWEEN 4 AND 5 THEN '4-5'

END [ОЦЕНКИ], COUNT (\*)[КОЛИЧЕСТВО]

FROM PROGRESS AS P

GROUP BY CASE

WHEN P.NOTE =10 THEN '10'

WHEN P.NOTE BETWEEN 8 AND 9 THEN '8-9'

WHEN P.NOTE BETWEEN 6 AND 7 THEN '6-7'

WHEN P.NOTE BETWEEN 4 AND 5 THEN '4-5'

END) AS SLCT1

ORDER BY CASE [ОЦЕНКИ]

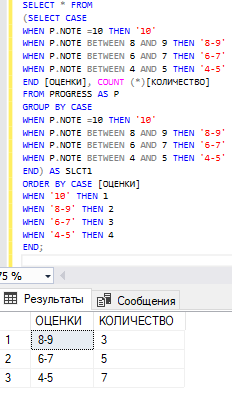
WHEN '10' THEN 1

WHEN '8-9' THEN 2

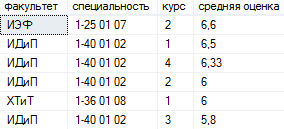
WHEN '6-7' THEN 3

WHEN '4-5' THEN 4

END;



4. Результат SELECT-запроса на основе таблиц **FACULTY**, **GROUPS**, **STUDENT** и **PROGRESS** содержит среднюю экзаменационную оценку для каждого курса каждой специальности. Строки отсортированы в порядке убывания средней оценки.



Разработать SELECT-запрос, формирующий аналогичный результат. При этом следует учесть, что средняя оценка должна рассчитываться с точностью до двух знаков после запятой. Использовать внутреннее соединение таблиц, агрегатную функцию AVG и встроенные функции CAST и ROUND.

SELECT

F.FACULTY\_NAME AS [Факультет],

G.PROFESSION AS [Специальность],

(2014-G.YEAR\_FIRST) AS [Курс],

ROUND(AVG(CAST(PR.NOTE AS FLOAT(4))),2) AS [Средняя оценка]-вещественное каст, точность раунд

FROM

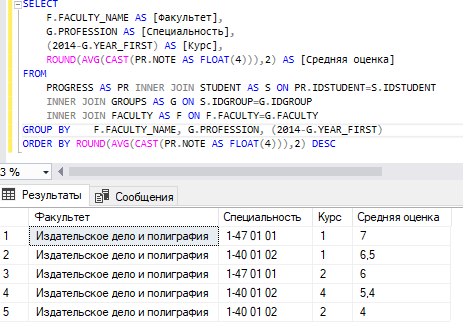
PROGRESS AS PR INNER JOIN STUDENT AS S ON PR.IDSTUDENT=S.IDSTUDENT

INNER JOIN GROUPS AS G ON S.IDGROUP=G.IDGROUP

INNER JOIN FACULTY AS F ON F.FACULTY=G.FACULTY

GROUP BY F.FACULTY\_NAME, G.PROFESSION, (2014-G.YEAR\_FIRST)

ORDER BY ROUND(AVG(CAST(PR.NOTE AS FLOAT(4))),2) DESC



5.Переписать SELECT-запрос, разработанный в задании 4 так, чтобы в расчете среднего значения оценок использовались оценки только по дисциплинам с кодами **БД** и **ОАиП**. Использовать WHERE.

SELECT

F.FACULTY\_NAME AS [Факультет],

G.PROFESSION AS [Специальность],

(2014-G.YEAR\_FIRST) AS [Курс],

ROUND(AVG(CAST(PR.NOTE AS FLOAT(4))),2) AS [Средняя оценка]

FROM PROGRESS AS PR INNER JOIN STUDENT AS S ON PR.IDSTUDENT=S.IDSTUDENT

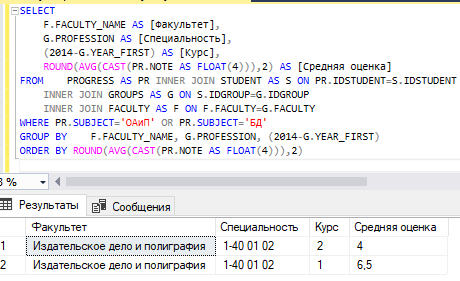
INNER JOIN GROUPS AS G ON S.IDGROUP=G.IDGROUP

INNER JOIN FACULTY AS F ON F.FACULTY=G.FACULTY

WHERE PR.SUBJECT='ОАиП' OR PR.SUBJECT='БД'

GROUP BY F.FACULTY\_NAME, G.PROFESSION, (2014-G.YEAR\_FIRST)

ORDER BY ROUND(AVG(CAST(PR.NOTE AS FLOAT(4))),2)



6. На основе таблиц **FACULTY**, **GROUPS**, **STUDENT** и **PROGRESS** разработать SELECT-запрос, в котором выводятся специальность, дисциплины и средние оценки при сдаче экзаменов на факультете ТОВ. Использовать группировку по полям FACULTY, PROFESSION, SUBJECT.

Добавить в запрос конструкцию **ROLLUP** и проанализировать результат.

Формирует промежуточные итоги для каждого указанного элемента и общий итог. Строки со значением NULL и есть промежутрочные итоги по отделам,а самая последняя – общий итог.

SELECT FA.FACULTY, G.PROFESSION, P.SUBJECT, AVG(P.NOTE)[СРЕДНЯЯ ОЦЕНКА]

FROM

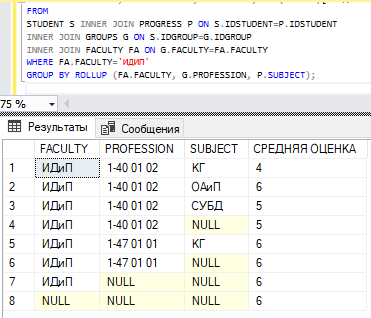
STUDENT S INNER JOIN PROGRESS P ON S.IDSTUDENT=P.IDSTUDENT

INNER JOIN GROUPS G ON S.IDGROUP=G.IDGROUP

INNER JOIN FACULTY FA ON G.FACULTY=FA.FACULTY

WHERE FA.FACULTY='ИДИП'

GROUP BY ROLLUP (FA.FACULTY, G.PROFESSION, P.SUBJECT);



**7.** Выполнить исходный SELECT-запрос пункта 6 с использованием **CUBE**-группировки. Проанализировать результат.

Формирует результаты для всех возможных перекрестных вычислений. Группировка и промежуточные итоги выполнены и для факультета, и для специальности.

SELECT FA.FACULTY, G.PROFESSION, P.SUBJECT, AVG(P.NOTE)[СРЕДНЯЯ ОЦЕНКА]

FROM

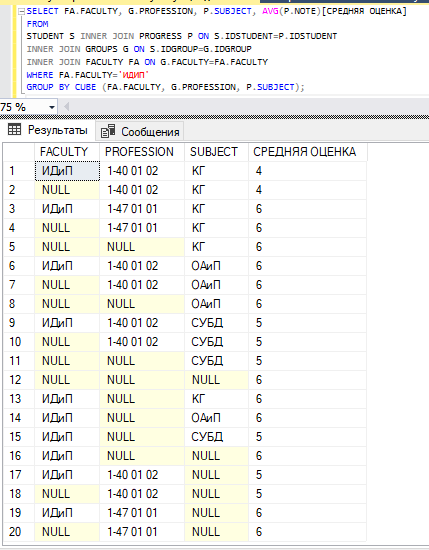
STUDENT S INNER JOIN PROGRESS P ON S.IDSTUDENT=P.IDSTUDENT

INNER JOIN GROUPS G ON S.IDGROUP=G.IDGROUP

INNER JOIN FACULTY FA ON G.FACULTY=FA.FACULTY

WHERE FA.FACULTY='ИДИП'

GROUP BY CUBE (FA.FACULTY, G.PROFESSION, P.SUBJECT);



8. На основе таблиц **GROUPS**, **STUDENT** и **PROGRESS** разработать SELECT-зчапрос, в котором определяются результаты сдачи экзаменов.

В запросе должны отражаться специальности, дисциплины, средние оценки студентов на факультете ИДиП.

Отдельно разработать запрос, в котором определяются результаты сдачи экзаменов на факультете ХТиТ.

Объединить результаты двух запросов с использованием операторов UNION и UNION ALL. Объяснить результаты.

Оператор **UNION** выполняет теоретико-множественную операцию объединения, т.е. результатом является множество строк, в котором строки не могут повторяться. Если требуется механическое объединение строк, можно применить оператор **UNION ALL**.

**UNION-только уникальные строки**

SELECT G.PROFESSION [СПЕЦИАЛЬНОСТЬ], P.SUBJECT [ДИСЦИПЛИНА], ROUND(AVG(P.NOTE),2)[СРЕДНЯЯ ОЦЕНКА]

FROM PROGRESS P INNER JOIN STUDENT S ON P.IDSTUDENT=S.IDSTUDENT

INNER JOIN GROUPS G ON S.IDGROUP=G.IDGROUP

WHERE G.FACULTY='ИДИП'

GROUP BY G.PROFESSION,P.SUBJECT

UNION

SELECT G.PROFESSION [СПЕЦИАЛЬНОСТЬ], P.SUBJECT [ДИСЦИПЛИНА], ROUND(AVG(P.NOTE),2)[СРЕДНЯЯ ОЦЕНКА]

FROM PROGRESS P INNER JOIN STUDENT S ON P.IDSTUDENT=S.IDSTUDENT

INNER JOIN GROUPS G ON S.IDGROUP=G.IDGROUP

WHERE G.FACULTY='ХТИТ'

GROUP BY G.PROFESSION,P.SUBJECT;

GO

**UNION ALL-все строки и возможные дубли**

SELECT G.PROFESSION [СПЕЦИАЛЬНОСТЬ], P.SUBJECT [ДИСЦИПЛИНА], ROUND(AVG(P.NOTE),2)[СРЕДНЯЯ ОЦЕНКА]

FROM PROGRESS P INNER JOIN STUDENT S ON P.IDSTUDENT=S.IDSTUDENT

INNER JOIN GROUPS G ON S.IDGROUP=G.IDGROUP

WHERE G.FACULTY='ИДИП'

GROUP BY G.PROFESSION,P.SUBJECT

UNION ALL

SELECT G.PROFESSION [СПЕЦИАЛЬНОСТЬ], P.SUBJECT [ДИСЦИПЛИНА], ROUND(AVG(P.NOTE),2)[СРЕДНЯЯ ОЦЕНКА]

FROM PROGRESS P INNER JOIN STUDENT S ON P.IDSTUDENT=S.IDSTUDENT

INNER JOIN GROUPS G ON S.IDGROUP=G.IDGROUP

WHERE G.FACULTY='ХТИТ'

GROUP BY G.PROFESSION,P.SUBJECT;



9.Получить пересечение двух множеств строк, созданных в результате выполнения запросов пункта 8. Примечание: использовать оператор INTERSECT.

Результатом оператора **INTERSECT** является набор строк, являющийся пересечением двух исходных наборов строк.

---INTERSECT

---Возвращает все различные значения, входящие в результаты выполнения запросов, указанных как слева, так и справа от оператора INTERSECT.

SELECT G.PROFESSION [СПЕЦИАЛЬНОСТЬ], P.SUBJECT [ДИСЦИПЛИНА], ROUND(AVG(P.NOTE),2)[СРЕДНЯЯ ОЦЕНКА]

FROM PROGRESS P INNER JOIN STUDENT S ON P.IDSTUDENT=S.IDSTUDENT

INNER JOIN GROUPS G ON S.IDGROUP=G.IDGROUP

WHERE G.FACULTY IN ('ИДИП', 'ХТИТ')

GROUP BY G.PROFESSION,P.SUBJECT

INTERSECT

SELECT G.PROFESSION [СПЕЦИАЛЬНОСТЬ], P.SUBJECT [ДИСЦИПЛИНА], ROUND(AVG(P.NOTE),2)[СРЕДНЯЯ ОЦЕНКА]

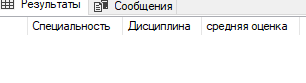
FROM PROGRESS P INNER JOIN STUDENT S ON P.IDSTUDENT=S.IDSTUDENT

INNER JOIN GROUPS G ON S.IDGROUP=G.IDGROUP

WHERE G.FACULTY='ХТИТ'

GROUP BY G.PROFESSION,P.SUBJECT;

GO



10.Получить разницу между множеством строк, созданных в результате запросов пункта 8. Примечание: использовать оператор EXCEPT.

**EXCEPT** –только те строки, которые есть в первом наборе, но нет во втором.

---EXCEPT

---Возвращает все различные значения, возвращенные запросом, указанным слева от оператора EXCEPT, но отсутствующие в результатах выполнения правого запроса.

SELECT G.PROFESSION [СПЕЦИАЛЬНОСТЬ], P.SUBJECT [ДИСЦИПЛИНА], ROUND(AVG(P.NOTE),2)[СРЕДНЯЯ ОЦЕНКА]

FROM PROGRESS P INNER JOIN STUDENT S ON P.IDSTUDENT=S.IDSTUDENT

INNER JOIN GROUPS G ON S.IDGROUP=G.IDGROUP

WHERE G.FACULTY IN ('ИДИП', 'ХТИТ')

GROUP BY G.PROFESSION,P.SUBJECT

EXCEPT

SELECT G.PROFESSION [СПЕЦИАЛЬНОСТЬ], P.SUBJECT [ДИСЦИПЛИНА], ROUND(AVG(P.NOTE),2)[СРЕДНЯЯ ОЦЕНКА]

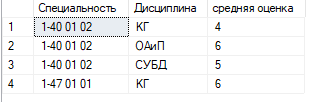
FROM PROGRESS P INNER JOIN STUDENT S ON P.IDSTUDENT=S.IDSTUDENT

INNER JOIN GROUPS G ON S.IDGROUP=G.IDGROUP

WHERE G.FACULTY='ХТИТ'

GROUP BY G.PROFESSION,P.SUBJECT

GO



11.На основе таблицы **PROGRESS** определить для каждой дисциплины количество студентов, получивших оценки 8 и 9.

Примечание: использовать группировку по двум столбцам, секция HAVING, сортировку по дисциплине и оценкам.

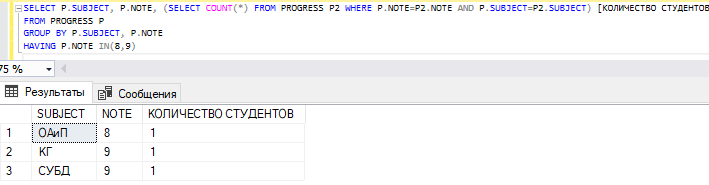
Логическое выражение, указанное в секции HAVING вычисляется для каждой строки результирующего набора, сформированного секцией GROUP BY. Как и в случае с секцией WHERE строка отбирается в результирующий набор, если логическое выражение принимает значение «истина».

SELECT P.SUBJECT, P.NOTE, (SELECT COUNT(\*) FROM PROGRESS P2 WHERE P.NOTE=P2.NOTE AND P.SUBJECT=P2.SUBJECT) [КОЛИЧЕСТВО СТУДЕНТОВ]

FROM PROGRESS P

GROUP BY P.SUBJECT, P.NOTE

HAVING P.NOTE IN(8,9)



## **Лабораторная работа №9**

**Использование представлений**

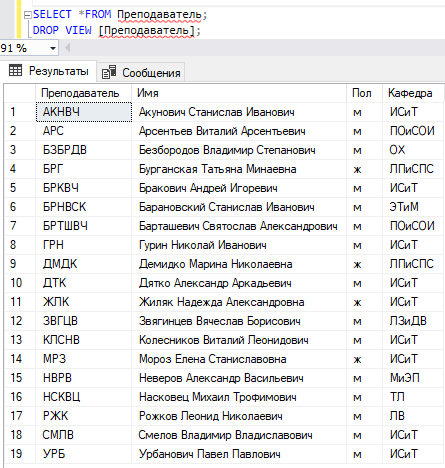
**Представление** (View) – это объект базы данных, представляющий собой *поименованный* SELECT-запрос, который хранится в базе данных. Представление создается с помощью оператора CREATE, удаляется с помощью оператора DROP и изменяется с помощью ALTER.

1. Разработать представление с именем **Преподаватель**. Представление должно быть построено на основе SELECT-запроса к таблице **TEACHER** и содержать следующие столбцы: **код** (**TEACHER**), **имя преподавателя** (**TEACHER\_NAME**), **пол** (**GENDER**), **код кафедры** (**PULPIT**).

Секцию ORDER BY можно использовать только совместно с опцией TOP; не допускается применение секции INTO, COMPUTE и COMPUTE BY; все столбцы результирующего набора должны быть поименованы.

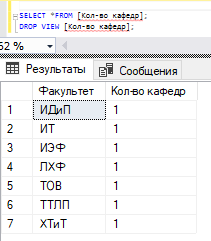






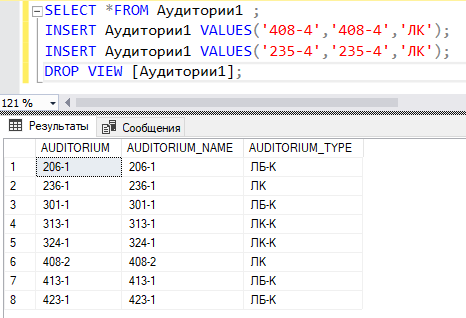
2.Разработать и создать представление с именем **Количество кафедр**. Представление должно быть построено на основе SELECT-запроса к таблицам **FACULTY** и **PULPIT**. Представление должно содержать следующие столбцы: **факультет** (**FACULTY.FACULTY\_ NAME**), **количество кафедр** (вычисляется на основе строк таблицы **PULPIT**).





1. Разработать и создать представление с именем **Аудитории**. Представление должно быть построено на основе таблицы **AUDITORIUM** и содержать столбцы: **код** (**AUDITORIUM**), **наименование аудитории** (**AUDITORIUM\_NAME**)**.** Представление должно отображать только лекционные аудитории (в столбце **AUDITORIUM\_ TYPE** строка, начинающаяся с символа **ЛК**) и допускать выполнение оператора INSERT, UPDATE и DELETE.



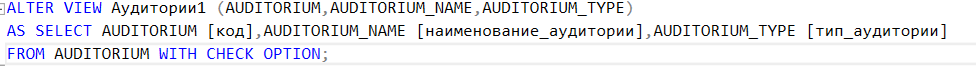


4.Разработать и создать представление с именем **Лекционные\_аудитории**. Представление должно быть построено на основе SELECT-запроса к таблице **AUDITORIUM** и содержать следующие столбцы: **код** (**AUDITORIUM**), **наименование аудитории** (**AUDITORIUM\_NAME**)**.** Представление должно отображать только лекционные аудитории (в столбце **AUDITORIUM\_TYPE** строка, начинающаяся с символов **ЛК**).

Выполнение INSERT и UPDATE допускается, но с учетом ограничения, задаваемого опцией WITH CHECK OPTION.

Флаг WITH CHECK OPTION запрещает пользователям вставку и обновление записей,

 не входящих в представление. Не выполнится, т.к. не удовлетворяет условиям



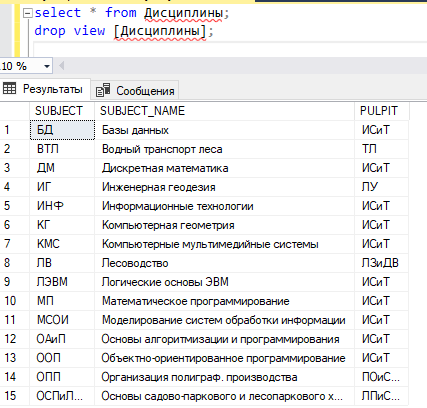
5.Разработать и создать представление с именем **Дисциплины**. Представление должно быть построено на основе SELECT-запроса к таблице **SUBJECT**, отображать все дисциплины в алфавитном порядкеи содержать следующие столбцы: **код** (**SUBJECT**), **наименование дисциплины** (**SUBJECT\_NAME**) и **код кафедры** (**PULPIT**). Примечание: использовать секции TOP и ORDER BY.

create view [Дисциплины] (SUBJECT, SUBJECT\_NAME, PULPIT)

as select top 15 s.SUBJECT [код], s.SUBJECT\_NAME [Наименование\_кафедры], s.PULPIT [код\_кафедры] from SUBJECT s order by s.SUBJECT;

select \* from Дисциплины;

drop view [Дисциплины];



6.Изменить представление **Количество\_кафедр**, созданное в задании 2 так, чтобы оно было привязано к базовым таблицам. Продемонстрировать свойство привязанности представления к базовым таблицам. Примечание: использовать опцию SCHEMABINDING.

--Запрещает операции, которые могут нарушить работоспособность представления

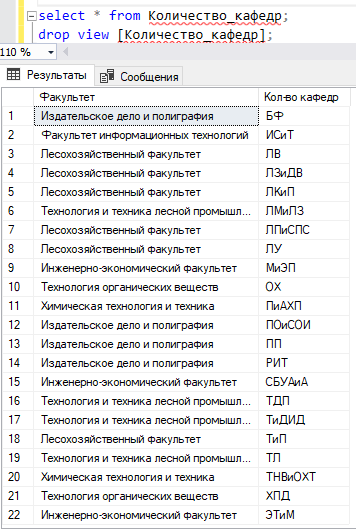
create view [Количество\_кафедр] with schemabinding

as select f.FACULTY\_NAME [Факультет], p.PULPIT [Кол-во кафедр]

from dbo.FACULTY f join dbo.PULPIT p on f.FACULTY = p.FACULTY;

select \* from Количество\_кафедр;

drop view [Количество\_кафедр];



## **Лабораторная работа №10**

**Основы программирования на T-SQL**

1. Разработать T-SQL-скрипт, в котором:

− объявить переменные типа char, varchar, datetime, time, int, smallint, tinint, numeric(12, 5);

− первые две переменные проинициализировать в операторе объявления;

− присвоить произвольные значения следующим двум переменным с помощью оператора SET, одной из этих переменных присвоить значение, полученное в результате запроса SELECT;

− одну из переменных оставить без инициализации и не присваивать ей значения, оставшимся переменным присвоить некоторые значения с помощью оператора SELECT;

− значения одной половины переменных вывести с помощью оператора SELECT,значения другой половины переменных распечатать с помощью оператора PRINT.

DECLARE @CH CHAR(3)='ME',

@VCH VARCHAR(15)='YOU&ME',

@DT DATETIME,

@T TIME,

@I INT,

@SI SMALLINT,

@TI TINYINT,

@NY NUMERIC(12,5);

SET @DT=GETDATE();

SET @T='23:45';

SELECT @T T

SELECT @I=CAST(SUM(AUDITORIUM\_CAPACITY)AS INT)FROM AUDITORIUM;

SELECT @SI=CAST(AVG(NOTE)AS SMALLINT)FROM PROGRESS;

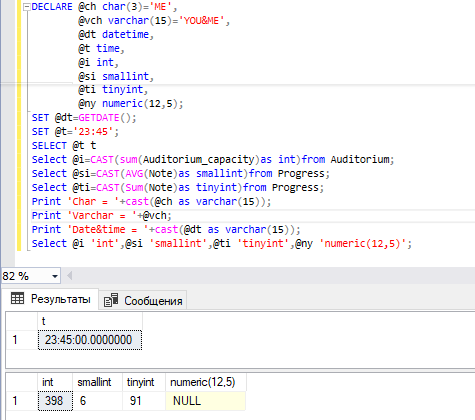
SELECT @TI=CAST(SUM(NOTE)AS TINYINT)FROM PROGRESS;

PRINT 'CHAR = '+CAST(@CH AS VARCHAR(15));

PRINT 'VARCHAR = '+@VCH;

PRINT 'DATE&TIME = '+CAST(@DT AS VARCHAR(15));

SELECT @I 'INT',@SI 'SMALLINT',@TI 'TINYINT',@NY 'NUMERIC(12,5)';



1. Разработать скрипт, в котором определяется общая вместимость аудиторий. Когда общая вместимость превышает 200, то вывести количество аудиторий, среднюю вместимость аудиторий, количество аудиторий, вместимость которых меньше средней, и процент таких аудиторий. Когда общая вместимость аудиторий меньше 200, то вывести сообщение о размере общей вместимости.

DECLARE @AUDCAP NUMERIC(10,2) = (SELECT SUM(AUDITORIUM\_CAPACITY) AS INT FROM AUDITORIUM),

@KOLCAP NUMERIC(10,2), @KOLCAPL NUMERIC(10,2), @AVGCAP NUMERIC(10,2),@ALLCAP NUMERIC(10,2), @PROC NUMERIC(10,2);

IF @AUDCAP>200

BEGIN

SELECT @KOLCAP=CAST(COUNT(AUDITORIUM)AS NUMERIC(10,2)) FROM AUDITORIUM;

SELECT @AVGCAP=CAST(AVG(AUDITORIUM\_CAPACITY)AS NUMERIC(10,2))FROM AUDITORIUM;

SELECT @KOLCAPL=CAST(COUNT(AUDITORIUM)AS NUMERIC(10,2)) FROM AUDITORIUM WHERE AUDITORIUM\_CAPACITY<@AVGCAP;

SET @PROC=(@KOLCAPL/@KOLCAP) \* 100;

PRINT 'КОЛИЧЕСТВО АУДИТОРИЙ = '+ CAST(@KOLCAP AS VARCHAR(10));

PRINT 'СРЕДНЯЯ ВМЕСТИМОСТЬ= '+ CAST(@AVGCAP AS VARCHAR(10));

PRINT 'КОЛИЧЕСТВО АУДИТОРИЙ С ВМЕСТИМОСТЬЮ НИЖЕ СРЕДНЕЙ = '+ CAST(@KOLCAPL AS VARCHAR(10));

PRINT 'ИХ ПРОЦЕНТ = '+ CAST(@PROC AS VARCHAR(10)) + '%';

END

ELSE

SELECT @ALLCAP=CAST(SUM(AUDITORIUM\_CAPACITY)AS NUMERIC(10,2))FROM AUDITORIUM;

PRINT 'ОБЩАЯ ВМЕСТИМОСТЬ'+ CAST(@ALLCAP AS VARCHAR(10));



1. Разработать T-SQL-скрипт, который выводит на печать глобальные переменные:

− @@ROWCOUNT (число обработанных строк);

− @@VERSION (версия SQL Server);

− @@SPID (возвращает системный идентификатор процесса, назначенный сервером текущему подключению);

− @@ERROR (код последней ошибки);

− @@SERVERNAME (имя сервера);

− @@TRANCOUNT (возвращает уровень вложенности транзакции);

− @@FETCH\_STATUS (проверка результата считывания строк результирующего набора);

− @@NESTLEVEL (уровень вложенности текущей процедуры).

PRINT 'ЧИСЛО ОБРАБОТАННЫХ СТРОК --> '+CAST(@@ROWCOUNT AS VARCHAR(50));

PRINT 'ВЕРСИЯ СЕРВЕРА --> '+CAST(@@VERSION AS VARCHAR(50));

PRINT 'СИСТЕМНЫЙ ИДЕНТИФИКАТОР ПРОЦЕССА, НАЗНАЧЕННЫЙ СЕРВЕРОМ ТЕКУЩЕМУ ПОДКЛЮЧЕНИЮ --> '+CAST(@@SPID AS VARCHAR(50));

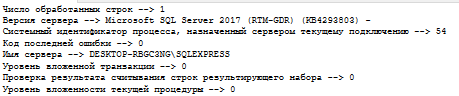
PRINT 'КОД ПОСЛЕДНЕЙ ОШИБКИ --> '+CAST(@@ERROR AS VARCHAR(50));

PRINT 'ИМЯ СЕРВЕРА --> '+CAST(@@SERVERNAME AS VARCHAR(50));

PRINT 'УРОВЕНЬ ВЛОЖЕННОЙ ТРАНЗАКЦИИ --> '+CAST(@@TRANCOUNT AS VARCHAR(50));

PRINT 'ПРОВЕРКА РЕЗУЛЬТАТА СЧИТЫВАНИЯ СТРОК РЕЗУЛЬТИРУЮЩЕГО НАБОРА --> '+CAST(@@FETCH\_STATUS AS VARCHAR(50));

PRINT 'УРОВЕНЬ ВЛОЖЕННОСТИ ТЕКУЩЕЙ ПРОЦЕДУРЫ --> '+CAST( @@NESTLEVEL AS VARCHAR(50));



4. Разработать T-SQL-скрипты, выполняющие:

− вычисление значений переменной **z**



для различных значений исходных данных;

− преобразование полного ФИО студента в сокращенное (например, Макейчик Татьяна Леонидовна в Макейчик Т. Л.);

--4.1

DECLARE @T41 FLOAT = 4.1,@X41 FLOAT =7.2 ,@Z41 FLOAT;

IF (@T41>@X41) SET @Z41=SIN(@T41)\*SIN(@T41)

IF (@T41<@X41) SET @Z41=4\*(@T41+@X41)

ELSE SET @Z41=1-EXP(@X41-2);

PRINT 'T= '+CAST(@T41 AS VARCHAR(10));

PRINT 'X= '+CAST(@X41 AS VARCHAR(10));

PRINT 'Z= '+CAST(@Z41 AS VARCHAR(10));

DECLARE @T42 FLOAT = 4.1,@X42 FLOAT =3.2 ,@Z42 FLOAT;

IF (@T42>@X42) SET @Z41=SIN(@T42)\*SIN(@T42)

IF (@T42<@X42) SET @Z41=4\*(@T42+@X42)

ELSE SET @Z42=1-EXP(@X42-2);

PRINT 'T= '+CAST(@T42 AS VARCHAR(10));

PRINT 'X= '+CAST(@X42 AS VARCHAR(10));

PRINT 'Z= '+CAST(@Z42 AS VARCHAR(10));

DECLARE @T43 FLOAT = 4.1,@X43 FLOAT =4.1 ,@Z43 FLOAT;

IF (@T43>@X43) SET @Z43=SIN(@T43)\*SIN(@T43)

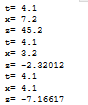
IF (@T43<@X43) SET @Z43=4\*(@T43+@X43)

ELSE SET @Z43=1-EXP(@X43-2);

PRINT 'T= '+CAST(@T43 AS VARCHAR(10));

PRINT 'X= '+CAST(@X43 AS VARCHAR(10));

PRINT 'Z= '+CAST(@Z43 AS VARCHAR(10));



--4.2

DECLARE @FAM VARCHAR(10)='BUGAENKO',@NAME VARCHAR(10)='VIKTORYA',@OTCH VARCHAR(10)='VYACHESLAVOVNA';

PRINT @FAM+' '+@NAME+' '+@OTCH;

PRINT @FAM+' '+CONVERT(VARCHAR(1),@NAME)+'. '+CONVERT(VARCHAR(1),@OTCH)+'. ';



5. Продемонстрировать конструкцию IF… ELSE на примере анализа данных таблиц базы данных **Х\_UNIVER**.

DECLARE @X5 INT = (SELECT COUNT(\*) FROM TEACHER);

IF @X5 > 50 PRINT '@X5 > 50';

ELSE PRINT '@X5 < 50';

IF @X5 > 25 PRINT '@X5 > 25';

ELSE PRINT '@X5 < 25';

IF @X5 > 5 PRINT '@X5 > 5';

ELSE PRINT '@X5 < 5';

PRINT '@X5 = '+CAST(@X5 AS VARCHAR(5));



6. Разработать сценарий, в котором с помощью CASE анализируются оценки, полученные студентами некоторого факультета при сдаче экзаменов

SELECT \*

FROM (SELECT CASE

WHEN NOTE BETWEEN 4 AND 6 THEN '4-6'

WHEN NOTE BETWEEN 7 AND 8 THEN '7-8'

ELSE '9-10'

END [ПРЕДЕЛЫ ОЦЕНОК], COUNT (\*) [КОЛИЧЕСТВО]

FROM PROGRESS GROUP BY CASE

WHEN NOTE BETWEEN 4 AND 6 THEN '4-6'

WHEN NOTE BETWEEN 7 AND 8 THEN '7-8'

ELSE '9-10'

END) AS T

ORDER BY CASE [ПРЕДЕЛЫ ОЦЕНОК]

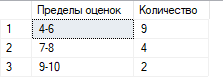
WHEN '9-10' THEN 3

WHEN '7-8' THEN 2

WHEN '4-6' THEN 1

ELSE 0

END



7. Создать временную локальную таблицу из трех столбцов и 10 строк, заполнить ее и вывести содержимое. Использовать оператор WHILE.

CREATE table #7

( Строка varchar(50) default 'Число',

Номер int,

Число int);

DECLARE @i7 int=0; ---заполнение

WHILE @i7<10

begin

INSERT #7 (Номер, Число)

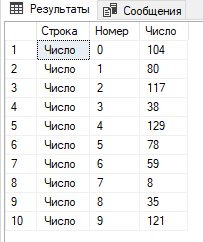
Values(@i7,floor(150\*rand()));

SET @i7=@i7+1;

end;

Select \* from #7

Drop table #7



8. Разработать скрипт, демонстрирующий использование оператора RETURN.

DECLARE @X INT = 1

PRINT @X+2

PRINT @X+3

RETURN

PRINT @X+3



9. Разработать сценарий с ошибками, в котором используются для обработки ошибок блоки TRY и CATCH. Применить функции ERROR\_NUMBER (код последней ошибки), ERROR\_ES-SAGE (сообщение об ошибке), ERROR\_LINE (код последней ошибки), ERROR\_PROCEDURE (имя процедуры или NULL), ERROR\_SEVERITY (уровень серьезности ошибки), ERROR\_ STATE (метка ошибки). Проанализировать результат.

BEGIN TRY

UPDATE DBO.STUDENT SET IDGROUP = '2.5' WHERE IDSTUDENT= '1000'

END TRY

BEGIN CATCH

PRINT 'Номер ошибки '

PRINT ERROR\_NUMBER()

PRINT ERROR\_MESSAGE()

PRINT 'Ошибка в строке '

PRINT ERROR\_LINE()

PRINT 'Уровень опасности '

PRINT ERROR\_SEVERITY()

PRINT 'Метка ошибки '

PRINT ERROR\_STATE()

END CATCH



## **Лабораторная работа № 12.**

**Обработка результатов запросов с помощью курсоров**

1.Разработать сценарий, формирующий список дисциплин на кафедре ИСиТ. В отчет должны быть выведены краткие названия (поле SUBJECT) из таблицы SUBJECT в одну строку через запятую. Использовать встроенную функцию RTRIM.

USE UNIVER;

GO

DECLARE @X CHAR(100), @Y CHAR(100);

DECLARE ISIT CURSOR FOR SELECT SUBJECT FROM SUBJECT

WHERE PULPIT = 'ИСиТ';

OPEN ISIT;

FETCH ISIT INTO @X;

PRINT 'Кафедра ИСиТ';

SET @Y=CAST (@X AS VARCHAR (10))

WHILE @@FETCH\_STATUS = 0

BEGIN

SET @Y = RTRIM(@X)+', '+@Y;

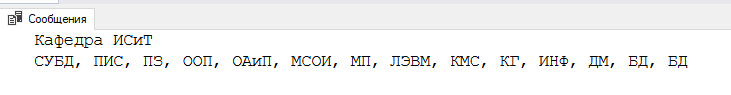
FETCH ISIT INTO @X;

END;

PRINT @Y;

CLOSE ISIT;

DEALLOCATE ISIT;



2. Разработать сценарий, демонстрирующий отличие глобального курсора от локального на примере базы данных X\_UNIVER.

**Локальный курсор:**

DECLARE TEACHER CURSOR LOCAL

FOR SELECT TEACHER, TEACHER\_NAME FROM TEACHER;

DECLARE @N CHAR(10), @C VARCHAR(100);

OPEN TEACHER;

FETCH TEACHER INTO @N, @C;

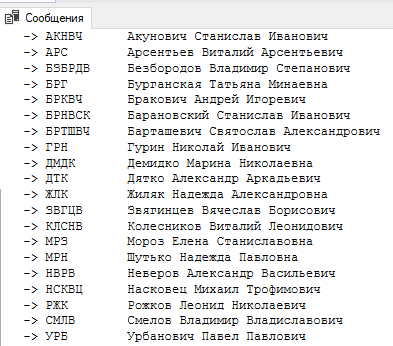
WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT '-> ' + @N+ ' ' +@C;

FETCH TEACHER INTO @N, @C;

END



--- teacher уже не существует:

DECLARE @N CHAR(10), @C VARCHAR(100);

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT '--> ' + @N+ ' ' +@C;

FETCH TEACHER INTO @N, @C;

END

---Глобальный курсор:

DECLARE TEACHER CURSOR GLOBAL

FOR SELECT TEACHER, TEACHER\_NAME FROM TEACHER;

DECLARE @N CHAR(10), @C VARCHAR(100);

OPEN TEACHER;

FETCH TEACHER INTO @N, @C;

WHILE @@FETCH\_STATUS = 0

BEGIN

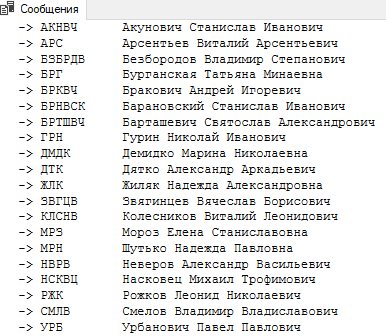
PRINT '-> ' + @N+ ' ' +@C;

FETCH TEACHER INTO @N, @C;

END

CLOSE TEACHER;

GO



DECLARE @N CHAR(10), @C VARCHAR(100);

OPEN TEACHER;

FETCH TEACHER INTO @N, @C;

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT '--> ' + @N+ ' ' +@C;

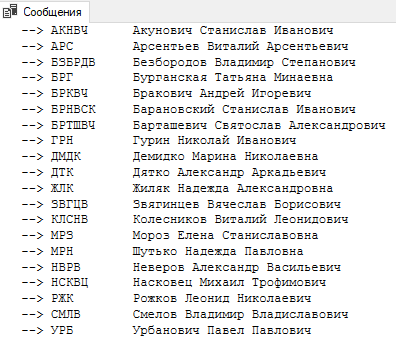
FETCH TEACHER INTO @N, @C;

END

CLOSE TEACHER;

DEALLOCATE TEACHER;

GO



3. Разработать сценарий, демонстрирующий отличие статических курсоров от динамических на примере базы данных X\_UNIVER.

---Статический курсор:

DECLARE @N CHAR(10), @C CHAR(100);

DECLARE TEACHER CURSOR STATIC/DYNAMIC

FOR SELECT TEACHER, TEACHER\_NAME

FROM DBO.TEACHER;

OPEN TEACHER;

PRINT 'Количество строк : '+CAST(@@CURSOR\_ROWS AS VARCHAR(5));

GO

 STATIC

DYNAMIC

----------------------------------------------------------------

UPDATE TEACHER SET TEACHER\_NAME='Имя Фамилия Отчество' WHERE GENDER = 'м';

DELETE TEACHER WHERE TEACHER = 'МИА';

INSERT TEACHER(TEACHER, TEACHER\_NAME, GENDER,

PULPIT)

VALUES ('ЧДИ','Черняк Дарья Игоревна', 'ж', 'ИСиТ');

DECLARE @N CHAR(10), @C CHAR(100);

FETCH TEACHER INTO @N, @C;

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT @N + ' '+ @C;

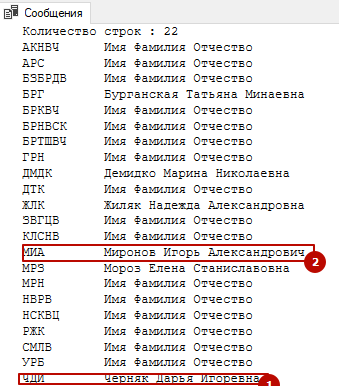
FETCH TEACHER INTO @N, @C;

END;

CLOSE TEACHER;

DEALLOCATE TEACHER;

GO



4. Разработать сценарий, демонстрирующий свойства навигации в результирующем наборе курсора с атрибутом SCROLL на примере базы данных X\_UNIVER.

Использовать все известные ключевые слова в операторе FETCH.

DECLARE @N INT, @P CHAR(20), @PN VARCHAR(100), @F CHAR(10);

DECLARE PULPIT CURSOR LOCAL DYNAMIC SCROLL

FOR SELECT ROW\_NUMBER() OVER (ORDER BY PULPIT) N, PULPIT, PULPIT\_NAME, FACULTY

FROM DBO.PULPIT;

OPEN PULPIT;

FETCH PULPIT INTO @N, @P, @PN, @F;

PRINT 'следующая строка: ' + CAST(@N AS VARCHAR(3)) + '.Кафедра ' + @P + @PN + ' Факультет: ' + @F;

FETCH LAST FROM PULPIT INTO @N, @P, @PN, @F;

PRINT 'последняя строка: ' + CAST(@N AS VARCHAR(3)) + '.Кафедра ' + @P + @PN + ' Факультет: ' + @F;

FETCH PRIOR FROM PULPIT INTO @N, @P, @PN, @F;

PRINT 'предыдущая строка от текущей: ' + CAST(@N AS VARCHAR(3)) + '.Кафедра ' + @P + @PN + ' Факультет: ' + @F;

FETCH ABSOLUTE 3 FROM PULPIT INTO @N, @P, @PN, @F;

PRINT 'третья строка от начала: ' + CAST(@N AS VARCHAR(3)) + '.Кафедра ' + @P + @PN + ' Факультет: ' + @F;

FETCH ABSOLUTE -3 FROM PULPIT INTO @N, @P, @PN, @F;

PRINT 'третья строка от конца: ' + CAST(@N AS VARCHAR(3)) + '.Кафедра ' + @P + @PN + ' Факультет: ' + @F;

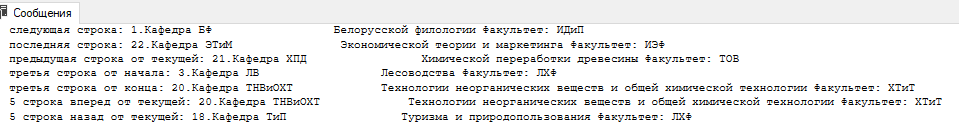
FETCH RELATIVE 5 FROM PULPIT INTO @N, @P, @PN, @F;

PRINT '5 строка вперед от текущей: ' + CAST(@N AS VARCHAR(3)) + '.Кафедра ' + @P + @PN + ' Факультет: ' + @F;

FETCH RELATIVE -5 FROM PULPIT INTO @N, @P, @PN, @F;

PRINT '5 строка назад от текущей: ' + CAST(@N AS VARCHAR(3)) + '.Кафедра ' + @P + @PN + ' Факультет: ' + @F;

CLOSE PULPIT;



5. Создать курсор, демонстрирующий применение конструкции CURRENT OF в секции WHERE с использованием операторов UPDATE и DELETE.

DECLARE @N5 CHAR(10), @C5 CHAR(100);

DECLARE TEACHER5 CURSOR LOCAL DYNAMIC

FOR SELECT TEACHER, TEACHER\_NAME

FROM DBO.TEACHER FOR UPDATE;

OPEN TEACHER5;

FETCH TEACHER5 INTO @N5, @C5;

UPDATE TEACHER SET TEACHER\_NAME='ФИО' WHERE CURRENT OF TEACHER5;

---DELETE TEACHER WHERE CURRENT OF TEACHER5;

CLOSE TEACHER5;

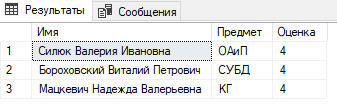


6. Разработать SELECT-запрос, с помощью которого из таблицы PROGRESS удаляются строки, содержащие информацию о студентах, получивших оценки ниже 4 (использовать объединение таблиц PROGRESS, STUDENT, GROUPS).

Разработать SELECT-запрос, с помощью которого в таблице PROGRESS для студента с конкретным номером IDSTUDENT корректируется оценка (увеличивается на единицу).

SELECT STUDENT.NAME AS [Имя], PROGRESS.SUBJECT AS [Предмет], PROGRESS.NOTE AS [Оценка] from STUDENT, PROGRESS, GROUPS

WHERE STUDENT.IDGROUP=GROUPS.IDGROUP AND PROGRESS.IDSTUDENT=STUDENT.IDSTUDENT AND PROGRESS.NOTE<=4



SELECT STUDENT.NAME AS [Имя], PROGRESS.SUBJECT AS [Предмет], PROGRESS.NOTE AS [Оценка] from STUDENT, PROGRESS, GROUPS

WHERE STUDENT.IDGROUP=GROUPS.IDGROUP AND STUDENT.IDSTUDENT=1015



DECLARE @A NVARCHAR(100), @B CHAR(10), @C INT;

DECLARE NOTES CURSOR LOCAL DYNAMIC

FOR SELECT STUDENT.NAME AS [Имя], PROGRESS.SUBJECT AS [Предмет], PROGRESS.NOTE AS [Оценка] FROM STUDENT, PROGRESS, GROUPS

WHERE STUDENT.IDGROUP=GROUPS.IDGROUP AND STUDENT.IDSTUDENT=1015 FOR UPDATE;

OPEN NOTES;

FETCH NOTES INTO @A, @B, @C;

WHILE @@FETCH\_STATUS=0

BEGIN

UPDATE PROGRESS SET NOTE=NOTE+1 WHERE CURRENT OF NOTES;

END;

CLOSE NOTES;

GO

DECLARE @A NVARCHAR(100), @B CHAR(10), @C INT;

DECLARE NOTES CURSOR LOCAL DYNAMIC

FOR SELECT STUDENT.NAME AS [Имя], PROGRESS.SUBJECT AS [Предмет], PROGRESS.NOTE AS [Оценка] FROM STUDENT, PROGRESS, GROUPS

WHERE STUDENT.IDGROUP=GROUPS.IDGROUP AND PROGRESS.IDSTUDENT=STUDENT.IDSTUDENT AND PROGRESS.NOTE<=4 FOR UPDATE;

OPEN NOTES;

FETCH NOTES INTO @A, @B, @C;

DELETE PROGRESS WHERE CURRENT OF NOTES;

CLOSE NOTES;

GO



## **Лабораторная работа № 11.**

**Создание и применение индексов**

1.С помощью SSMS определить все индексы, которые имеются в БД **UNIVER**. Определить, какие из них являются кластеризованными, а какие некластеризованными.

С помощью системной процедуры SP\_HELPINDEX можно получить перечень индексов, связанных с заданной таблицей:

exec sp\_helpindex 'AUDITORIUM'

exec sp\_helpindex 'AUDITORIUM\_TYPE'

exec sp\_helpindex 'FACULTY'

exec sp\_helpindex 'GROUPS'

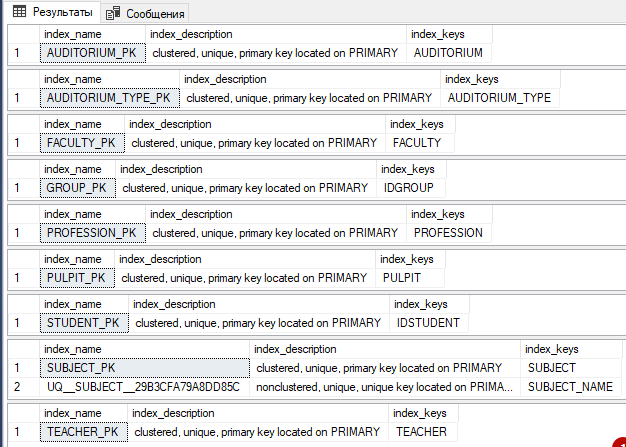
exec sp\_helpindex 'PROFESSION'

exec sp\_helpindex 'PULPIT'

exec sp\_helpindex 'STUDENT'

exec sp\_helpindex 'SUBJECT'

exec sp\_helpindex 'TEACHER'



2.Создать временную локальную таблицу. Заполнить ее данными (не менее 1000 строк).

Разработать SELECT-запрос. Получить план запроса и определить его стоимость.

Создать кластеризованный индекс, уменьшающий стоимость SELECT-запроса.

--- кластеризованные индексы

CREATE table #local

( number int, symbol varchar(100) );

GO

SET nocount on; --не выводить сообщения о вводе строк

DECLARE @i2 int=0;

WHILE @i2<1000

begin

INSERT #local(number, symbol)

values(floor(20000\*RAND()), REPLICATE('строка', 10));

IF(@i2 % 100=0) print @i2; --вывести сообщение

SET @i2=@i2+1;

end;

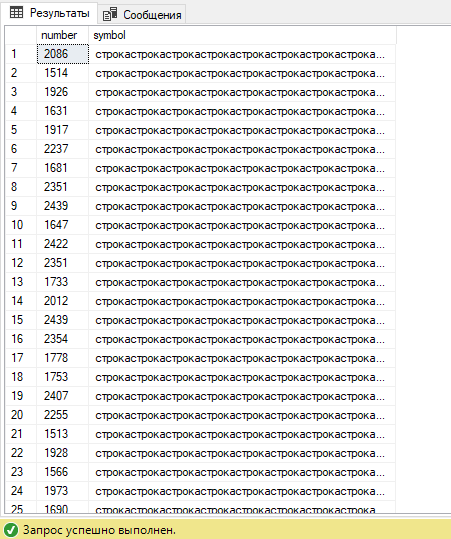
GO

SELECT \* from #local WHERE number between 1500 and 2500 ORDER BY symbol

checkpoint; --фиксация БД

DBCC DROPCLEANBUFFERS; --очистить буферный кэш

CREATE clustered index #local\_CL on #local(symbol asc) ---кластеризованный индекс



3.Создать временную локальную таблицу. Заполнить ее данными (1000 строк или больше).

Разработать SELECT-запрос. Получить план запроса и определить его стоимость.

Создать *некластеризованный* неуникальный *составной* индекс.

Оценить процедуры поиска информации.

--- некластеризованные индексы

CREATE TABLE #tmp3

( i\_1 int IDENTITY(0,1), i\_2 int, i\_3 varchar(10))

SET NOCOUNT ON; --не выводить сообщения о вводе строк

DECLARE @i INT = 0;

WHILE (@i<10000)

BEGIN

INSERT #tmp3(i\_2, i\_3) VALUES (FLOOR(666\*rand()), 'Database');

SET @i=@i+1;

END;

SELECT COUNT(\*) FROM #tmp3

SELECT \* FROM #tmp3

---некластеризованный составной неуникальный

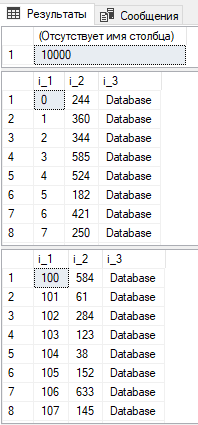
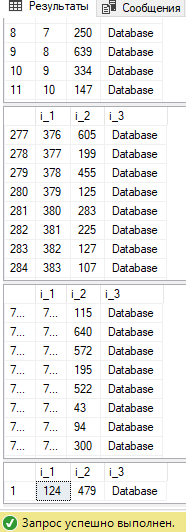
CREATE INDEX #tmp3\_NOCL ON #tmp3(i\_1, i\_2);

SELECT \* FROM #tmp3 WHERE i\_1 BETWEEN 100 AND 900 AND i\_2<10000;

SELECT \* FROM #tmp3 ORDER BY i\_1, i\_2;

SELECT \* FROM #tmp3 WHERE i\_1=124 AND i\_2<1000; ---индекс применится(фиксированное значение)

GO

4.Создать временную локальную таблицу. Заполнить ее данными (не менее 10000 строк).

Разработать SELECT-запрос. Получить план запроса и определить его стоимость.

Создать *некластеризованный индекс покрытия*, уменьшающий стоимость SELECT-запроса.

--- некластеризованный индекс покрытия

CREATE TABLE #tmp4

( i\_1 int IDENTITY(0,1), i\_2 int, i\_3 varchar(10))

SET NOCOUNT ON; --не выводить сообщения о вводе строк

DECLARE @i INT = 0;

WHILE (@i<10000)

BEGIN

INSERT #tmp4(i\_2, i\_3) VALUES (FLOOR(666\*rand()), 'Database');

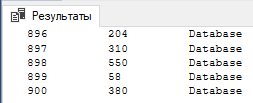
SET @i=@i+1;

END;

SELECT \* FROM #tmp4 WHERE i\_1 BETWEEN 100 AND 900;

CREATE index #tmp4\_X on #tmp4(i\_2) INCLUDE (i\_1)-- позволяет включить в состав индексной строки значения одного или нескольких неиндексируемых столбцов.

GO



5.Создать и заполнить временную локальную таблицу.

Разработать SELECT-запрос, получить план запроса и определить его стоимость.

Создать *некластеризованный фильтруемый индекс*, уменьшающий стоимость SELECT-запроса.

--- некластеризованный фильтруемый индекс

CREATE TABLE #tmp5

( i\_1 int IDENTITY(0,1), i\_2 int, i\_3 varchar(10))

SET NOCOUNT ON; --не выводить сообщения о вводе строк

DECLARE @i INT = 0;

WHILE (@i<10000)

BEGIN

INSERT #tmp5(i\_2, i\_3) VALUES (FLOOR(666\*rand()), 'Database');

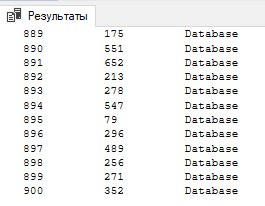
SET @i=@i+1;

END;

GO

SELECT \* FROM #tmp5 WHERE i\_1 BETWEEN 100 AND 900;--фильтрация

CREATE index #tmp5\_X on #tmp5(i\_2) INCLUDE (i\_1)



6.Создать и заполнить временную локальную таблицу.

Создать некластеризованный индекс. Оценить уровень *фрагментации индекса*.

Разработать сценарий на T-SQL, выполнение которого приводит к уровню фрагментации индекса выше 90%. Оценить уровень фрагментации индекса.

Выполнить процедуру *реорганизации* индекса, оценить уровень фрагментации.

Выполнить процедуру *перестройки* индекса и оценить уровень фрагментации индекса.

CREATE TABLE #tmp

( i\_1 int IDENTITY(0,1), i\_2 int, i\_3 varchar(10))

SET NOCOUNT ON; --не выводить сообщения о вводе строк

DECLARE @i INT = 0;

WHILE (@i<1000)

BEGIN

INSERT #tmp(i\_2, i\_3) VALUES (FLOOR(666\*rand()), 'Database');

SET @i=@i+1;

END;

GO

CREATE INDEX #TMP6\_X ON #tmp(i\_2)

Go

---информация о степени фрагментации индекса:

SELECT name [Индекс], avg\_fragmentation\_in\_percent [Фрагментация (%)] FROM SYS.dm\_db\_index\_physical\_stats(DB\_iD(N'TEMPDB'), OBJECT\_ID(N'#tmp'), NULL,NULL,NULL) ss

JOIN .tempbd.sys.indexes ii on ss.object\_id = ii.object\_id and ss.index\_id = ii.index\_id

where name is not null;

INSERT top(1000) #tmp(i\_2, i\_3) select i\_2, i\_3 from #tmp;

ALTER INDEX #TMP ON #tmp REORGANIZE;--реорганизация

ALTER INDEX #TMP ON #tmp REBUILD WITH (ONLINE=OFF)--перестройка

go



7.Разработать пример, демонстрирующий применение параметра FILLFACTOR при создании некластеризованного индекса. Параметр FILLFACTOR указывает процент заполнения индексных страниц нижнего уровня

---7---

DROP INDEX #TMP6\_X on #tmp6

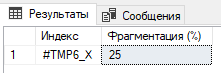
CREATE index #TMP6\_X on #tmp6(i\_2)with (fillfactor = 65);

INSERT top(50)percent into #tmp6(i\_2, i\_3) select i\_2, i\_3 from #tmp6;

SELECT name [Индекс], avg\_fragmentation\_in\_percent [Фрагментация (%)] FROM SYS.dm\_db\_index\_physical\_stats(DB\_iD(N'TEMPDB'), OBJECT\_ID(N'#tmp6'), NULL,NULL,NULL) ss

JOIN tempbd.sys.indexes ii on ss.object\_id = ii.object\_id and ss.index\_id = ii.index\_id

where name is not null;



## **Лабораторная работа № 13.**

**Особенности использования транзакций**

1. Разработать сценарий, демонстрирующий работу в режиме *неявной* транзакции.

Проанализировать пример, приведенный справа, в котором создается таблица Х, и создать сценарий для другой таблицы.

---1--- РЕЖИМ НЕЯВНОЙ ТРАНЗАКЦИИ

set nocount on

if exists (select \* from SYS.OBJECTS where OBJECT\_ID=object\_id(N'DBO.tempUka')) drop table tempUka;

declare @x1 int, @flag char = 'c'; -- если поменять с на r, то таблица не сохранится

SET IMPLICIT\_TRANSACTIONS ON -- включение режима неявной транзакции

create table tempUka(x2 int ); --начало

insert tempUka values (1),(2),(3),(4),(5);

set @x1 = (select count(\*) from tempUka);

print 'количество строк в таблице tempUka: ' + cast(@x1 as varchar(2));

if @flag = 'c' commit;

else rollback;

SET IMPLICIT\_TRANSACTIONS OFF -- выключение режима неявной транзакции

if exists (select \* from SYS.OBJECTS where OBJECT\_ID= object\_id(N'DBO.tempUka')) print 'таблица tempUka есть';

else print 'таблицы tempUka нет'



2.Разработать сценарий, демонстрирующий свойство *атомарности* *явной* транзакции на примере базы данных X\_UNIVER.

В блоке CATCH предусмотреть выдачу соответствующих сообщений об ошибках.

Опробовать работу сценария при использовании различных операторов модификации таблиц.

---2--- атомарность явной транзакции

begin try

begin tran -- начало явной транзакции

insert FACULTY values ('ПиМ', 'Факультет print-технологий и медиакоммуникаций');

commit tran; -- фиксация транзакции

end try

begin catch

print 'ошибка: '+ case

when error\_number() = 2627 and patindex('%FACULTY\_PK%', error\_message()) > 0 then 'дублирование товара'

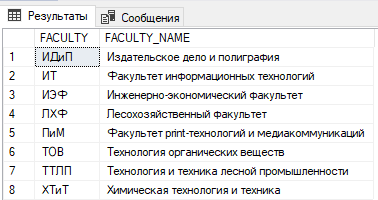
else 'неизвестная ошибка: '+ cast(error\_number() as varchar(5))+ error\_message()

end;

if @@trancount > 0 rollback tran; --уровень вложенности транзакции (если значение больше нуля, то транзакция не завершена). + откат(rollback)

end catch;

select \* from FACULTY;



3.Разработать сценарий, демонстрирующий применение оператора SAVE TRAN на примере базы данных X\_UNIVER.

В блоке CATCH предусмотреть выдачу соответствующих сообщений об ошибках.

Опробовать работу сценария при использовании различных контрольных точек и различных операторов модификации таблиц.

--- ОПЕРАТОР SAVETRAN

declare @point varchar(32);

begin try

begin tran --начало явной транзакции

set @point = 'p\_1'; save tran @point; -- контрольная точка p\_1

insert STUDENT(IDGROUP, NAME, BDAY, INFO, FOTO) values (19,'Иван', '1998-01-23', NULL, NULL),(19,'Василий', '1997-02-02', NULL, NULL),(19,'Дмитрий', '1997-02-01', NULL, NULL),(19,'Виктория', '1995-02-03', NULL, NULL);

set @point = 'p\_2'; save tran @point; -- контрольная точка p\_2

insert STUDENT(IDGROUP, NAME, BDAY, INFO, FOTO) values (19, 'Вася Пупкин', '1998-04-01', NULL, NULL);

commit tran;

end try

begin catch

print 'ошибка: '+ case

when error\_number() = 2627 and patindex('%STUDENT\_PK%', error\_message()) > 0 then 'дублирование студента'

else 'неизвестная ошибка: '+ cast(error\_number() as varchar(5)) + error\_message()

end;

if @@trancount > 0 -- если транзакция не завершена

begin

print 'контрольная точка: '+ @point;

rollback tran @point; -- откат к последней контрольной точке

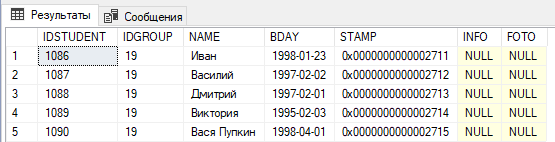
commit tran; -- фиксация изменений, выполненных до контрольной точки

end;

end catch;

select \* from STUDENT where IDGROUP=19;

delete STUDENT where IDGROUP=19;



4.Разработать два сценария A и B на примере базы данных X\_UNIVER.

Сценарий A представляет собой явную транзакцию с уровнем изолированности READ UNCOMMITED, сценарий B – явную транзакцию с уровнем изолированности READ COMMITED (по умолчанию).

Сценарий A должен демонстрировать, что уровень READ UNCOMMITED допускает неподтвержденное, неповторяющееся и фантомное чтение.

--- +неподтвержденное, +неповторяющееся, +фантомное чтение

---A

set transaction isolation level READ UNCOMMITTED

begin transaction --системный идентификатор процесса

---t1

select @@SPID[идентификатор процесса], 'insert FACULTY' 'result', \*from FACULTY

where FACULTY='ТОВ';

select @@SPID[идентификатор процесса], 'update FACULTY' 'result', FACULTY, FACULTY\_NAME from FACULTY

where FACULTY ='ТОВ';

commit;

---t2

---B

begin transaction ---read commited по умолчанию

select @@SPID[идентификатор процесса]

insert FACULTY values ('АТФ', 'Автотракторный факультет');

---delete FACULTY where FACULTY = 'АТФ';

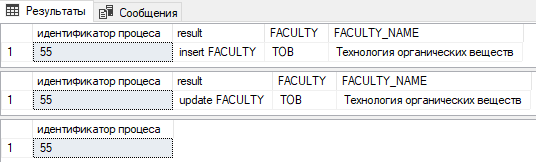
update FACULTY set FACULTY='АФ'

where FACULTY\_NAME='Автотракторный факультет';

---t1

---t2

rollback;



5.Разработать два сценария A и B на примере базы данных X\_UNIVER.

Сценарии A и В представляют собой явные транзакции с уровнем изолированности READ COMMITED.

Сценарий A должен демонстрировать, что уровень READ COMMITED не допускает неподтвержденного чтения, но при этом возможно неповторяющееся и фантомное чтение.

---A -неподтверждённое +неповторяющееся +фантомное

set transaction isolation level READ COMMITTED

begin transaction

select count(\*) from PROFESSION

where PROFESSION\_NAME='Химические процессы';

---t1

---t2

select 'update PROFESSION' 'result', count(\*)

from PROFESSION where PROFESSION\_NAME='Химические процессы';

commit;

---B

begin transaction

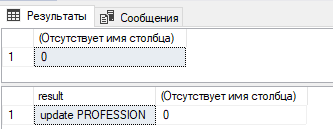
---t1

update PROFESSION set PROFESSION\_NAME='экономист'

where PROFESSION\_NAME='Химические процессы';

commit;

---t2



6.Разработать два сценария A и B на примере базы данных X\_UNIVER.

Сценарий A представляет собой явную транзакцию с уровнем изолированности REPEATABLE READ. Сценарий B – явную транзакцию с уровнем изолированности READ COMMITED.

Сценарий A должен демонстрировать, что уровень REAPETABLE READ не допускает неподтвержденного чтения и неповторяющегося чтения, но при этом возможно фантомное чтение.

---A -неподтвержденное, - неповторяющееся, +фантомное

set transaction isolation level REPEATABLE READ

begin transaction

select FACULTY from PROFESSION where PROFESSION='1-48 01 02';

---t1

---t2

select case

when FACULTY='ХТиТ' then 'insert PROFESSION' else ''

end 'result', PROFESSION\_NAME from PROFESSION where FACULTY='ХТиТ';

commit;

---B

begin transaction

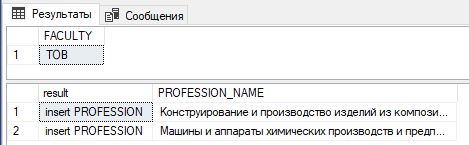
---t1

insert PROFESSION values('1-46 03 77', 'ИТ', 'Дизайн электронных и веб изданий', 'программист-дизайнер');

---delete PROFESSION where PROFESSION = '1-46 03 77';

commit;

---t2



7.Разработать два сценария A и B на примере базы данных X\_UNIVER.

Сценарий A представляет собой явную транзакцию с уровнем изолированности SERIALIZABLE.

Сценарий B – явную транзакцию с уровнем изолированности READ COMMITED.

Сценарий A должен демонстрировать отсутствие фантомного, неподтвержденного и неповторяющегося чтения

USE UNIVER

---A snapshot изолированность (отсутствие взаимного влияния параллельных транзакций на результаты их выполнения)

alter database UNIVER set allow\_snapshot\_isolation on

set transaction isolation level SNAPSHOT

begin transaction

select PROFESSION from PROFESSION where PROFESSION='1-48 01 02';

---t1

---delete PROFESSION where PROFESSION='1-48 01 09';

insert PROFESSION values('1-48 01 09','ТОВ', 'какое-то название', 'какая-то специализация');

---t2

select PROFESSION from PROFESSION where PROFESSION='1-48 01 09';

commit;

---B

begin transaction

---t1

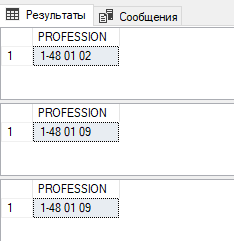
delete PROFESSION where PROFESSION = '1-48 01 09';

insert PROFESSION values('1-48 01 09','ТОВ', 'Химическая технология органических веществ, материалов и изделий','инженер-химик-технолог');

select PROFESSION from PROFESSION where PROFESSION = '1-48 01 09';

commit;

---t2



USE UNIVER

---A -фантомное -неподтвержденное – неповторяющееся

set transaction isolation level SERIALIZABLE

begin transaction

---delete PROFESSION where PROFESSION='1-46 01 99';

insert PROFESSION values('1-46 01 99', 'ТТЛП','Технологии и техники лесной промышленности', 'Лесоинженерное дело инженер-технолог');

---t1

select PROFESSION from PROFESSION where PROFESSION = '1-48 01 02';

---t2

commit;

---B

begin transaction

delete PROFESSION where PROFESSION = '1-46 01 99';

insert PROFESSION values('1-46 01 99', 'ТТЛП','Технологии и техники лесной промшленности', 'Лесоинженерное дело инженер-технолог');

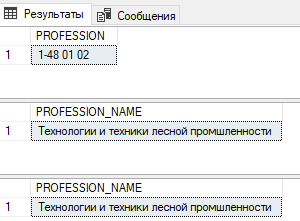
select PROFESSION\_NAME from PROFESSION where PROFESSION = '1-46 01 99';

---t1

commit;

select PROFESSION\_NAME from PROFESSION where PROFESSION = '1-46 01 99';

---t2



8.Разработать сценарий, демонстрирующий свойства *вложенных* транзакций, на примере базы данных X\_UNIVER.

USE UNIVER

-- ВЛОЖЕННЫЕ ТРАНЗАКЦИИ

-- Транзакция, выполняющаяся в рамках другой транзакции, называется вложенной.

-- оператор COMMIT вложенной транзакции действует только на внутренние операции вложенной транзакции;

-- оператор ROLLBACK внешней транзакции отменяет зафиксированные операции внутренней транзакции;

-- оператор ROLLBACK вложенной транзакции действует на операции внешней и внутренней транзакции,

-- а также завершает обе транзакции;

-- уровень вложенности транзакции можно определить с помощью системной функции @@TRANCOUT.

alter database UNIVER set allow\_snapshot\_isolation on

select (select count(\*) from dbo.PULPIT where FACULTY = 'ИДиП') 'Кафедры ИДИПа',

(select count(\*) from FACULTY where FACULTY.FACULTY = 'ИДиП') 'ИДИП';

select \* from PULPIT

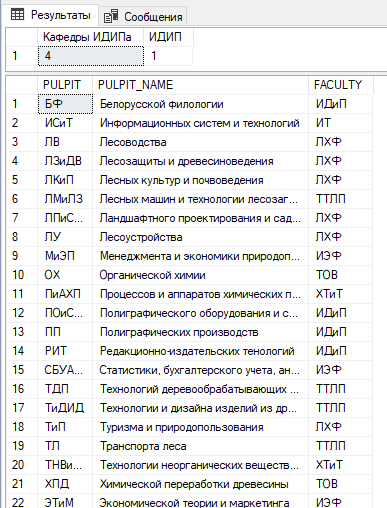
begin tran ----внешняя транзакция

begin tran ----внутренняя транзакция

update PULPIT set PULPIT\_NAME='Кафедра ИДиПа' where PULPIT.FACULTY = 'ИДиП';

commit; ----внутренняя транзакция

if @@TRANCOUNT > 0 rollback; ----внешняя транзакция



## **Лабораторная работа № 14.**

**Разработка хранимых процедур**

1.Разработать хранимую процедуру без параметров с именем **PSUBJECT**. Процедура формирует результирующий набор на основе таблицы **SUBJECT**, аналогичный набору, представленному на рисунке:

****

create procedure PSUBJECT1

as begin

declare @x\_1 int = (select count(\*) from "SUBJECT");

select SUBJECT [КОД], SUBJECT\_NAME [ДИСЦИПЛИНА], PULPIT [КАФЕДРА] from "SUBJECT";

return @x\_1;

end;

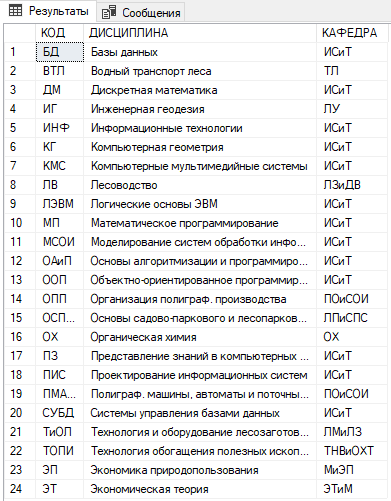
declare @y\_1 int;

exec @y\_1 = PSUBJECT1; --вызов процедуры

print 'Количество предметов: ' + cast(@y\_1 as varchar(3));

go

go



2.Найти процедуру **PSUBJECT** с помощью обозревателя объектов (Object Explorer) SSMS и через контекстное меню создать сценарий на изменение процедуры оператором ALTER.

Изменить процедуру **PSUBJECT**, созданную в задании 1, таким образом, чтобы она принимала два параметра с именами **@p** и **@c**. Параметр **@p** является входным, имеет тип VARCHAR(20) и значение по умолчанию NULL. Параметр **@с** является выходным, имеет тип INT.

Процедура **PSUBJECT** должна формировать результирующий набор, аналогичный набору, представленному на рисунке выше, но при этом содержать строки, соответствующие коду кафедры, заданному параметром **@p**. Кроме того, процедура должна формировать значение выходного параметра **@с**, равное количеству строк в результирующем наборе, а также возвращать значение к точке вызова, равное общему количеству дисциплин (количеству строк в таблице **SUBJECT**).

alter procedure PSUBJECT1

@p varchar(20), --входной параметр

@c int output --выходной

as begin

select \* from "SUBJECT" where "SUBJECT" = @p;

set @c = @@rowcount;

return @c;

end;

declare @x\_2 int;

exec @x\_2 = PSUBJECT1 @p = 'СУБД', @c = @x\_2 output;

print 'Количество предметов: ' + cast(@x\_2 as varchar(3));

go



3.Создать временную локальную таблицу с именем **#SUBJECT**. Наименование и тип столбцов таблицы должны соответствовать столбцам результирующего набора процедуры **PSUBJECT**, разработанной в задании 2.

Изменить процедуру **PSUBJECT** таким образом, чтобы она не содержала выходного параметра.

Применив конструкцию INSERT… EXECUTE с модифицированной процедурой **PSUBJECT**, добавить строки в таблицу **#SUBJECT**.

alter procedure PSUBJECT1

@p varchar(20) --входной параметр

as begin

select \* from "SUBJECT" where "SUBJECT" = @p;

end;

create table #SUBJECT(Код\_предмета varchar(20), Название\_предмета varchar(100), Кафедра varchar(20));

insert #SUBJECT exec PSUBJECT1 @p = 'БД';

insert #SUBJECT exec PSUBJECT1 @p = 'СУБД';

select \* from #SUBJECT;

go



4.Разработать процедуру с именем **PAUDITORIUM\_INSERT**. Процедура принимает четыре входных параметра: **@a**, **@n**, **@c** и **@t**. Параметр **@a** имеет тип CHAR(20), параметр **@n** имеет тип VARCHAR(50), параметр **@c** имеет тип INT и значение по умолчанию **0**, параметр **@t** имеет тип CHAR(10).

Процедура добавляет строку в таблицу **AUDITORIUM**. Значения столбцов **AUDITORIUM**, **AUDITORIUM\_NAME, AUDITORIUM\_CAPACITY** и **AUDITORIUM\_TYPE** добавляемой строки задаются соответственно параметрами **@a**, **@n**, **@c** и **@t**.

Процедура **PAUDITORIUM\_INSERT** должна применять механизм TRY/CATCH для обработки ошибок. В случае возникновения ошибки, процедура должна формировать сообщение, содержащее код ошибки, уровень серьезности и текст сообщения в стандартный выходной поток.

Процедура должна возвращать к точке вызова значение **-1** в том случае, если произошла ошибка и **1**, если выполнение успешно.

Опробовать работу процедуры с различными значениями исходных данных, которые вставляются в таблицу.

create procedure PAUDITORIUM\_INSERT

@a char(20),

@n varchar(50),

@c int = 0,

@t char(10)

as begin

begin try

insert into AUDITORIUM(AUDITORIUM, AUDITORIUM\_TYPE, AUDITORIUM\_CAPACITY, AUDITORIUM\_NAME)

values(@a, @n, @c, @t);

return 1;

end try

begin catch

print 'Номер ошибки: ' + cast(error\_number() as varchar(6));

print 'Сообщение: ' + error\_message();

print 'Уровень: ' + cast(error\_severity() as varchar(6));

print 'Метка: ' + cast(error\_state() as varchar(8));

print 'Номер строки: ' + cast(error\_line() as varchar(8));

if error\_procedure() is not null

print 'Имя процедуры: ' + error\_procedure();

return -1;

end catch;

end;

declare @p int;

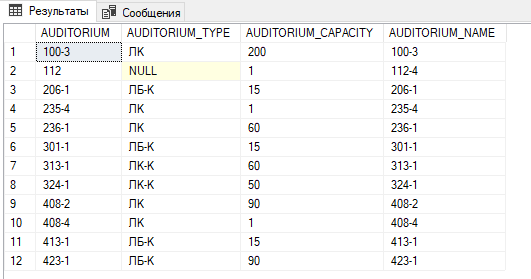
exec @p = PAUDITORIUM\_INSERT @a = '100-3', @n = 'ЛК', @c = 200, @t = '100-3';

print 'Код ошибки: ' + cast(@p as varchar(3));

go

select \* from AUDITORIUM

--drop procedure PAUDITORIUM\_INSERT



5.Разработать процедуру с именем **SUBJECT\_REPORT**, формирующую в стандартный выходной поток отчет со списком дисциплин на конкретной кафедре. В отчет должны быть выведены краткие названия (поле SUBJECT) из таблицы SUBJECT в одну строку через запятую (использовать встроенную функцию RTRIM). Процедура имеет входной параметр с именем **@p** типа CHAR(10), который предназначен для указания кода кафедры.

В том случае, если по заданному значению **@p** невозможно определить код кафедры, процедура должна генерировать ошибку с сообщением **ошибка в параметрах**.

Процедура **SUBJECT\_REPORT** должна возвращать к точке вызова количество дисциплин, отображенных в отчете.

create procedure SUBJECT\_REPORT1

@p char(10) = 0

as begin

declare @x int;

begin try

declare @с char(10), @r varchar(100) = '';

declare sbjct cursor for

select "SUBJECTS" from "SUBJECTS" where PULPIT = @p;

if not exists(select "SUBJECTS" from "SUBJECTS" where PULPIT = @p)

raiserror('Ошибка', 11, 1); --11 - обработка ошибок

else open sbjct;

fetch sbjct into @с;

print 'Предметы: ';

while @@fetch\_status = 0

begin

set @r = rtrim(@с) + ', ' + @r;

set @x = @x + 1;

fetch sbjct into @с;

end

print @r;

close sbjct;

return @x;

end try

begin catch

print 'Ошибка в параметрах'

if error\_procedure() is not null

print 'Имя процедуры: ' + error\_procedure();

print 'Номер строки: ' + cast(error\_line() as varchar(8));

return @x;

end catch;

end;

declare @y int;

exec @y = SUBJECT\_REPORT1 @p ='ОХ';

print 'Количество предметов: ' + cast(@y as varchar(3));

go

drop procedure SUBJECT\_REPORT1

go



6.Разработать процедуру с именем **PAUDITORIUM\_INSERTX**. Процедура принимает пять входных параметров: **@a**, **@n**, **@c**, **@t** и **@tn**.

Параметры **@a**, **@n**, **@c**, **@t** аналогичны параметрам процедуры **PAUDITORIUM\_INSERT**. Дополнительный параметр **@tn** является входным, имеет тип VARCHAR(50), предназначен для ввода значения в столбец **AUDITORIUM\_TYPE**.**AUDITORIUM\_TYPENAME**.

Процедура добавляет две строки. Первая строка добавляется в таблицу **AUDITORIUM\_TYPE**. Значения столбцов **AUDITORIUM\_TYPE** и **AUDITORIUM\_ TYPENAME** добавляемой строки задаются соответственно параметрами **@t** и **@tn**. Вторая строка добавляется путем вызова процедуры **PAUDITORIUM\_INSERT**.

Добавление строки в таблицу **AUDITORIUM\_TYPE** и вызов процедуры **PAUDITORIUM\_INSERT** должны выполняться в рамках одной явной транзакции с уровнем изолированности SERIALIZABLE.

В процедуре должна быть предусмотрена обработка ошибок с помощью механизма TRY/CATCH. Все ошибки должны быть обработаны с выдачей соответствующего сообщения в стандартный выходной поток.

Процедура **PAUDITORIUM\_INSERTX** должна возвращать к точке вызова значение **-1** в том случае, если произошла ошибка и **1**, если выполнения процедуры завершилось успешно.

create procedure PAUDITORIUM\_INSERTX @a char(20), @n varchar(50), @c int = 0, @t char(10), @tn varchar(50)

as begin

declare @rc int = 1;

begin try

set transaction isolation level serializable;

begin tran

insert into AUDITORIUM\_TYPE(AUDITORIUM\_TYPE, AUDITORIUM\_TYPENAME) values(@n, @tn);

exec @rc = PAUDITORIUM\_INSERT @a, @n, @c, @t;

commit tran;

return @rc;

end try

begin catch

print 'Номер ошибки: ' + cast(error\_number() as varchar(6));

print 'Сообщение: ' + error\_message();

print 'Уровень: ' + cast(error\_severity() as varchar(6));

print 'Метка: ' + cast(error\_state() as varchar(8));

print 'Номер строки: ' + cast(error\_line() as varchar(8));

if error\_procedure() is not null

print 'Имя процедуры: ' + error\_procedure();

if @@trancount > 0 rollback tran ;

return -1;

end catch;

end;

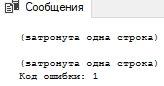
declare @k3 int;

exec @k3 = PAUDITORIUM\_INSERTX '201', @n = 'КB', @c = 101, @t = '201', @tn = 'Поточк';

print 'Код ошибки: ' + cast(@k3 as varchar(3));

--drop procedure PAUDITORIUM\_INSERTX

--процедура изменит оценку студенту в зав-сти от средней оценки в группе. если выше средняя, то + 1 балл, ниже - -1



## **Лабораторная работа № 15.**

**Разработка и использование функций**

1. Разработать *скалярную* функцию с именем **COUNT\_STUDENTS**, которая вычисляет количество студентов на факультете, код которого задается параметром типа VARCHAR(20) с именем **@faculty**. Использовать внутреннее соединение таблиц FACULTY, GROUPS, STUDENT. Опробовать работу функции.

Внести изменения в текст функции с помощью оператора ALTER с тем, чтобы функция принимала второй параметр **@prof** типа VARCHAR(20),обозначающий специальность студентов. Для параметров определить значения по умолчанию NULL. Опробовать работу функции с помощью SELECT-запросов.

create function COUNT\_STUDENTS1(@faculty varchar(20)) returns int

as begin

declare @x1 int = 0;

set @x1 = (select count(IDSTUDENT)

from FACULTY inner join GROUPS

on FACULTY.FACULTY = GROUPS.FACULTY inner join STUDENT

on GROUPS.IDGROUP = STUDENT.IDGROUP

where FACULTY.FACULTY LIKE @faculty);

return @x1;

end;

--drop function COUNT\_STUDENTS

go

declare @col int = dbo.COUNT\_STUDENTS1('ИТ');

print 'Количество студентов: ' + cast(@col as varchar(4));

go



alter function COUNT\_STUDENTS1(@faculty varchar(20) = null, @prof varchar(20) = '1-89 02 02') returns int

as begin

declare @x2 int = 0;

set @x2 = (select count(IDSTUDENT)

from FACULTY inner join "GROUPS"

on FACULTY.FACULTY = "GROUPS".FACULTY inner join STUDENT

on "GROUPS".IDGROUP = STUDENT.IDGROUP

where FACULTY.FACULTY = @faculty and "GROUPS".PROFESSION = @prof);

return @x2;

end;

go

declare @col int = dbo.COUNT\_STUDENTS1('ЛХФ', '1-89 02 02');

print 'Количество студентов: ' + cast(@col as varchar(4));

go



2. Разработать *скалярную* функцию с именем **FSUBJECTS,** принимающую параметр **@p** типа VARCHAR(20), значение которого задает код кафедры (столбец **SUBJECT.PULPIT**).

Функция должна возвращать строку типа VARCHAR(300) с перечнем дисциплин в отчете.

Создать и выполнить сценарий, который создает отчет, аналогичный представленному ниже.

Примечание: использовать локальный статический курсор на основе SELECT-запроса к таблице **SUBJECT**.



create function FSUBJECTS(@p varchar(20)) returns varchar(300)

as begin

declare @x varchar(10), @d varchar(300) = 'Дисциплины: ';

declare sub cursor local static for select "SUBJECTS" from "SUBJECTS" where PULPIT LIKE @p;

open sub;

fetch sub into @x;

while @@fetch\_status = 0

begin

set @d = @d + ', ' + rtrim(@x);

fetch sub into @x;

end;

return @d;

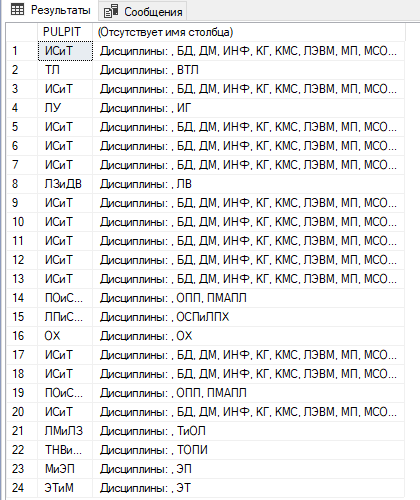
end;

--DROP FUNCTION FSUBJECTS

go

select PULPIT, dbo.FSUBJECTS(PULPIT) from "SUBJECTS";

go



3. Разработать *табличную* функцию **FFACPUL**, результаты работы которой продемонстрированы на рисунке ниже.

Функция принимает два параметра, задающих код факультета (столбец **FACULTY.FACULTY**) и код кафедры (столбец **PULPIT.PULPIT**). Использует SELECT-запрос c левым внешним соединением между таблицами **FACULTY** и **PULPIT**.

Если оба параметра функции равны NULL, то она возвращает результирующий набор, содержащий список всех кафедр на всех факультетах.

Если задан первый параметр (второй равен NULL), функция возвращает результирующий набор, содержащий список всех кафедр заданного факультета.

Если задан второй параметр (первый равен NULL), функция возвращает результирующий набор, содержащий строку, соответствующую заданной кафедре.

Если заданы два параметра, функция возвращает результирующий набор, содержащий строку, соответствующую заданной кафедре на заданном факультете.

Если по заданным значениям параметров невозможно сформировать строки, функция возвращает пустой результирующий набор.

create function FFACPUL(@f char(10), @p char(10)) returns table

as return

select F.Faculty, F.Faculty\_name, P.PULPIT

from Faculty F left outer join PULPIT P

on F.Faculty = P.FACULTY

where F.Faculty = isnull(@f,F.Faculty)

and P.PULPIT = isnull(@p, P.PULPIT);

-- DROP FUNCTION FFACPUL

go

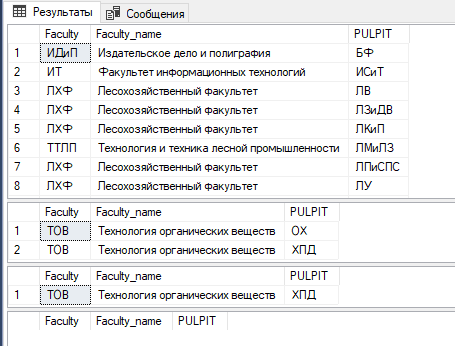
select \* from dbo.FFACPUL(null, null);

select \* from dbo.FFACPUL('ТОВ', null);

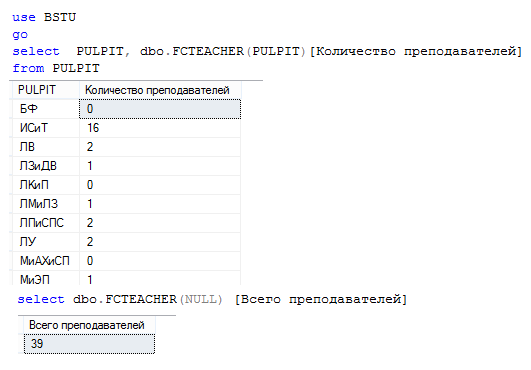
select \* from dbo.FFACPUL(null, 'ХПД');

select \* from dbo.FFACPUL('ТОВ', 'ХПДP');

go



4. На рисунке ниже показан сценарий, демонстрирующий работу *скалярной* функции **FCTEACHER**. Функция принимает один параметр, задающий код кафедры. Функция возвращает количество преподавателей на заданной параметром кафедре. Если параметр равен NULL, то возвращается общее количество преподавателей.

****

Разработать функцию **FCTEACHER**.

create function FCTEACHER(@p varchar(20)) returns int

as begin

declare @x int = (select count(TEACHER) from TEACHER where PULPIT = isnull(@p, PULPIT));

return @x;

end;

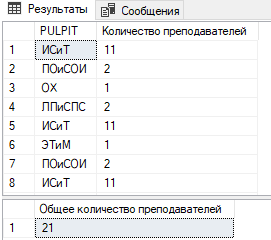
go

-- DROP FUNCTION FCTEACHER

select PULPIT, dbo.FCTEACHER(PULPIT)[Количество преподавателей] from TEACHER;

select dbo.FCTEACHER(null)[Общее количество преподавателей];

go



5.Проанализировать *многооператорную табличную* функцию **FACULTY\_REPORT**, представленную ниже:

**create function FACULTY\_REPORT(@c int) returns @fr table**

**( [Факультет] varchar(50), [Количество кафедр] int, [Количество групп] int,**

**[Количество студентов] int, [Количество специальностей] int )**

**as begin**

**declare cc CURSOR static for**

**select FACULTY from FACULTY**

**where dbo.COUNT\_STUDENTS(FACULTY, default) > @c;**

**declare @f varchar(30);**

**open cc;**

**fetch cc into @f;**

**while @@fetch\_status = 0**

**begin**

**insert @fr values( @f, (select count(PULPIT) from PULPIT where FACULTY = @f),**

**(select count(IDGROUP) from GROUPS where FACULTY = @f), dbo.COUNT\_STUDENTS(@f, default),**

**(select count(PROFESSION) from PROFESSION where FACULTY = @f) );**

**fetch cc into @f;**

**end;**

**return;**

**end;**

Изменить эту функцию так, чтобы количество кафедр, количество групп, количество студентов и количество специальностей вычислялось отдельными скалярными функциями.

create function COUNT\_PULPIT(@p varchar(20)) returns int

as begin

declare @x int = 0;

set @x = (select count(PULPIT) from PULPIT where FACULTY = @p);

return @x;

end;

-- DROP FUNCTION COUNT\_PULPIT

go

create function COUNT\_GROUP(@f varchar(20)) returns int

as begin

declare @rc int = 0;

set @rc = (select count(IDGROUP) from "GROUPS" where FACULTY like @f);

return @rc;

end;

-- DROP FUNCTION COUNT\_GROUP

go

create function COUNT\_PROFESSION(@f varchar(20)) returns int

as begin

declare @rc int = 0;

set @rc = (select count(PROFESSION) from PROFESSION where FACULTY like @f);

return @rc;

end;

-- DROP FUNCTION COUNT\_PROFESSION

go

create function FACULTY\_REPORT(@c int) returns @fr table([Факультет] varchar(50), [Количество кафедр] int, [Количество групп] int, [Количество студентов] int, [Количество специальностей] int)

as begin

declare cc cursor static for select FACULTY from FACULTY where dbo.COUNT\_STUDENTS1(FACULTY, default)> @c;

declare @f varchar(30);

open cc;

fetch cc into @f;

while @@fetch\_status = 0

begin

insert @fr values(@f, dbo.COUNT\_PULPIT(@f), dbo.COUNT\_GROUP(@f), dbo.COUNT\_STUDENTS1(@f, default), dbo.COUNT\_PROFESSION(@f));

fetch cc into @f;

end;

return;

end;

-- DROP FUNCTION FACULTY\_REPORT

go

## **Лабораторная работа № 16.**

**Применение триггеров**

1.С помощью сценария, представленного на рисунке, создать таблицу **TR\_AUDIT**.

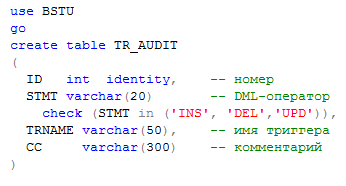
****

Таблица предназначена для добавления в нее строк триггерами.

В столбец **STMT** триггер должен поместить событие, на которое он среагировал, а в столбец **TRNAME −** собственное имя.

Разработать AFTER-триггер с именем **TR\_TEACHER\_INS** для таблицы **TEACHER**, реагирующий на событие **INSERT**. Триггер должен записывать строки вводимых данных в таблицу **TR\_AUDIT**. В столбец **СС** помещаются значения столбцов вводимой строки.

create table TR\_AUDIT

(

ID int identity,

STMT varchar(20)

check (STMT in ('INS', 'DEL', 'UPD')),

TRNAME varchar(50),

CC varchar(300)

)

go

create trigger TR\_TEACHER\_INS

on TEACHER after INSERT

as

declare @x1 char(10), @x2 varchar(100), @x3 char(1), @x4 char(20), @in varchar(300);

print 'Операция вставки';

set @x1 = (select TEACHER from INSERTED);

set @x2= (select TEACHER\_NAME from INSERTED);

set @x3= (select GENDER from INSERTED);

set @x4 = (select PULPIT from INSERTED);

set @in = @x1+' '+ @x2 +' '+ @x3+ ' ' +@x4;

insert into TR\_AUDIT(STMT, TRNAME, CC)

values('INS', 'TR\_TEACHER\_INS', @in);

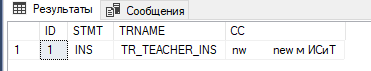
return;

go

select \* from TR\_AUDIT;

insert into TEACHER values('nw', 'new', 'м', 'ИСиТ');

select \* from TR\_AUDIT



2.Создать AFTER-триггер с именем **TR\_TEACHER\_DEL** для таблицы **TEA-CHER**, реагирующий на событие **DELETE**. Триггер должен записывать строку данных в таблицу **TR\_AUDIT** для каждой удаляемой строки. В столбец **СС** помещаются значения столбца **TEACHER** удаляемой строки.

create trigger TR\_TEACHER\_DEL on TEACHER after DELETE

as

declare @x1 char(10), @x2 varchar(100), @x3 char(1), @x4 char(20), @in varchar(300);

print 'Операция удаления';

set @x1 = (select TEACHER from DELETED);

set @x2= (select TEACHER\_NAME from DELETED);

set @x3= (select GENDER from DELETED);

set @x4 = (select PULPIT from DELETED);

set @in = @x1+' '+ @x2 +' '+ @x3+ ' ' +@x4;

insert into TR\_AUDIT(STMT, TRNAME, CC)

values('DEL', 'TR\_TEACHER\_DEL', @in);

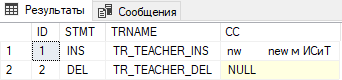
return;

go

delete TEACHER where TEACHER='nw'

select \* from TR\_AUDIT

drop trigger TR\_TEACHER\_DEL



3.Создать AFTER-триггер с именем **TR\_TEACHER\_UPD** для таблицы **TEA-CHER**, реагирующий на событие **UPDATE**. Триггер должен записывать строку данных в таблицу **TR\_AUDIT** для каждой изменяемой строки. В столбец **СС** помещаются значения столбцов изменяемой строки до и после изменения.

alter trigger TR\_TEACHER\_DEL

on TEACHER after UPDATE

as

declare @x1 char(10), @x2 varchar(100), @x3 char(1), @x4 char(20), @in varchar(300);

declare @ins int = (select count(\*) from inserted),

@del int = (select count(\*) from deleted);

print 'Операция обновления';

set @x1 = (select TEACHER from INSERTED);

set @x2= (select TEACHER\_NAME from INSERTED);

set @x3= (select GENDER from INSERTED);

set @x4 = (select PULPIT from INSERTED);

set @in = @x1+' '+ @x2 +' '+ @x3+ ' ' +@x4;

set @x1 = (select TEACHER from deleted);

set @x2= (select TEACHER\_NAME from DELETED);

set @x3= (select GENDER from DELETED);

set @x4 = (select PULPIT from DELETED);

set @in =@in + '' + @x1+' '+ @x2 +' '+ @x3+ ' ' +@x4;

insert into TR\_AUDIT(STMT, TRNAME, CC)

values('UPD', 'TR\_TEACHER\_UPD', @in);

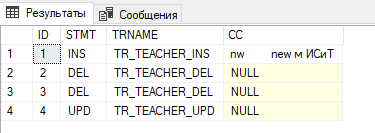
return;

go

update TEACHER set GENDER = 'ж' where TEACHER='nw'

select \* from TR\_AUDIT

delete from TR\_AUDIT where STMT = 'UPD'



4.Создать AFTER-триггер с именем **TR\_TEACHER** для таблицы **TEACHER**, реагирующий на события **INSERT**, **DELETE**, **UPDATE**.

Триггер должен записывать строку данных в таблицу **TR\_AUDIT** для каждой изменяемой строки. В коде триггера определить событие, активизировавшее триггер и поместить в столбец **СС** соответствующую событию информацию.

Разработать сценарий, демонстрирующий работоспособность триггера.

create trigger TR\_TEACHER

on TEACHER after INSERT, DELETE, UPDATE

as declare @x1 char(10), @x2 varchar(100), @x3 char(1), @x4 char(20), @in varchar(300);

declare @ins int = (select count(\*) from inserted),

@del int = (select count(\*) from deleted);

if @ins > 0 and @del = 0 begin print 'Событие: INSERT';

set @x1 = (select TEACHER from INSERTED);

set @x2= (select TEACHER\_NAME from INSERTED);

set @x3= (select GENDER from INSERTED);

set @x4 = (select PULPIT from INSERTED);

set @in = @x1+' '+ @x2 +' '+ @x3+ ' ' +@x4;

insert into TR\_AUDIT(STMT, TRNAME, CC)

values('INS', 'TR\_TEACHER\_INS', @in);

end; else

if @ins = 0 and @del > 0 begin print 'Событие: DELETE';

set @x1 = (select TEACHER from DELETED);

set @x2= (select TEACHER\_NAME from DELETED);

set @x3= (select GENDER from DELETED);

set @x4 = (select PULPIT from DELETED);

set @in = @x1+' '+ @x2 +' '+ @x3+ ' ' +@x4;

insert into TR\_AUDIT(STMT, TRNAME, CC)

values('DEL', 'TR\_TEACHER\_DEL', @in);

end; else

if @ins > 0 and @del > 0 begin print 'Событие: UPDATE';

set @x1 = (select TEACHER from INSERTED);

set @x2= (select TEACHER\_NAME from INSERTED);

set @x3= (select GENDER from INSERTED);

set @x4 = (select PULPIT from INSERTED);

set @in = @x1+' '+ @x2 +' '+ @x3+ ' ' +@x4;

set @x1 = (select TEACHER from deleted);

set @x2= (select TEACHER\_NAME from DELETED);

set @x3= (select GENDER from DELETED);

set @x4 = (select PULPIT from DELETED);

set @in =@in + '' + @x1+' '+ @x2 +' '+ @x3+ ' ' +@x4;

insert into TR\_AUDIT(STMT, TRNAME, CC)

values('UPD', 'TR\_TEACHER\_UPD', @in);

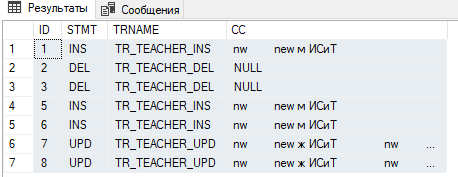
end; return;

delete TEACHER where TEACHER='nw'

insert into TEACHER values('nw', 'new', 'м', 'ИСиТ');

update TEACHER set GENDER = 'ж' where TEACHER='nw'

select \* from TR\_AUDIT



5.Разработать сценарий, который демонстрирует на примере базы данных X\_UNIVER, что проверка ограничения целостности выполняется до срабатывания AFTER-триггера.

alter table AUDITORIUM

add constraint AUDITORIUM\_CAPACITY check(AUDITORIUM\_CAPACITY<=200)

go

update AUDITORIUM set AUDITORIUM\_CAPACITY = 250 where AUDITORIUM = '200';

6.Создать для таблицы **TEACHER** три AFTER-триггера с именами: **TR\_TEACHER\_ DEL1**, **TR\_TEACHER\_DEL2** и **TR\_TEA-CHER\_ DEL3**. Триггеры должны реагировать на событие DELETE и формировать соответствующие строки в таблицу **TR\_AUDIT**. Получить список триггеров таблицы **TEACHER**. Упорядочить выполнение триггеров для таблицы **TEACHER**, реагирующих на событие **DELETE** следующим образом: первым должен выполняться триггер с именем **TR\_TEA-CHER\_DEL3**, последним – триггер **TR\_TEACHER\_DEL2**.

Примечание: использовать системные представления **SYS.TRIGGERS** и **SYS.TRIG-GERS\_ EVENTS**, а также системную процедуру **SP\_SETTRIGGERORDERS**.

Insert into FACULTY(FACULTY) values ('new')

go

create trigger AUD\_AFTER\_DEL1 on FACULTY after DELETE

as print 'AUD\_AFTER\_DEL1';

return;

go

create trigger AUD\_AFTER\_DEL2 on FACULTY after DELETE

as print 'AUD\_AFTER\_DEL2';

return;

go

create trigger AUD\_AFTER\_DEL3 on FACULTY after DELETE

as print 'AUD\_AFTER\_DEL3';

return;

go

---drop trigger AUD\_AFTER\_DEL2

delete FACULTY from FACULTY where FACULTY='new';

Проверить порядок выполнения триггеров можно следующим запросом:

select t.name, e.type\_desc

from sys.triggers t join sys.trigger\_events e on t.object\_id = e.object\_id

where OBJECT\_NAME(t.parent\_id)='FACULTY' and e.type\_desc = 'DELETE' ;

Изменение порядка выполнения триггеров выполняется с помощью системных процедур:

exec SP\_SETTRIGGERORDER @triggername = 'AUD\_AFTER\_DEL3',

@order='First', @stmttype = 'DELETE';

exec SP\_SETTRIGGERORDER @triggername = 'AUD\_AFTER\_DEL2',

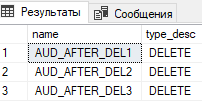
@order='Last', @stmttype = 'DELETE';

select t.name, e.type\_desc

from sys.triggers t join sys.trigger\_events e on t.object\_id = e.object\_id

where OBJECT\_NAME(t.parent\_id)='FACULTY' and e.type\_desc = 'DELETE' ;

GO



7.Разработать сценарий, демонстрирующий на примере базы данных X\_UNIVER утверждение: AFTER-триггер является частью транзакции, в рамках которого выполняется оператор, активизировавший триггер.

create trigger PTran

on PULPIT after INSERT, DELETE, UPDATE

as declare @c int = (select count (\*) from PULPIT);

if (@c >26)

begin

raiserror('Общая количество кафедр не может быть >26', 10, 1);

rollback;

end;

return;

insert into PULPIT(PULPIT) values ('new')

8.Для таблицы **FACULTY** создать **INSTEAD OF**-триггер, запрещающий удаление строк в таблице.

Разработать сценарий, который демонстрирует на примере базы данных X\_UNIVER, что проверка ограничения целостности выполнена, если есть INSTEAD OF-триггер.

С помощью оператора DROP удалить все DML-триггеры, созданные в этой лабораторной работе.

create trigger F\_INSTEAD\_OF

on FACULTY instead of DELETE

as

raiserror (N'Удаление запрещено', 10, 1);

return;

delete FACULTY where FACULTY = 'ИДиП'

drop trigger F\_INSTEAD\_OF

drop trigger PTran

drop trigger TR\_TEACHER

drop trigger TR\_TEACHER\_DEL

9.Создать DDL-триггер, реагирующий на все DDL-события в БД **UNIVER**. Триггер должен запрещать создавать новые таблицы и удалять существующие. Свое выполнение триггер должен сопровождать сообщением, которое содержит: тип события, имя и тип объекта, а также пояснительный текст, в случае запрещения выполнения оператора.

Разработать сценарий, демонстрирующий работу триггера.

create trigger DDL\_First on database

for DDL\_DATABASE\_LEVEL\_EVENTS

as declare @t varchar(50)= EVENTDATA().value('(/EVENT\_INSTANCE/EventType)[1]', 'varchar(50)');

declare @t1 varchar(50)= EVENTDATA().value('(/EVENT\_INSTANCE/ObjectName)[1]', 'varchar(50)');

declare @t2 varchar(50)= EVENTDATA().value('(/EVENT\_INSTANCE/ObjectType)[1]', 'varchar(50)');

if @t1 = 'Progress'

begin

print 'Тип события: '+@t;

print 'Имя объекта: '+@t1;

print 'Тип объекта: '+@t2;

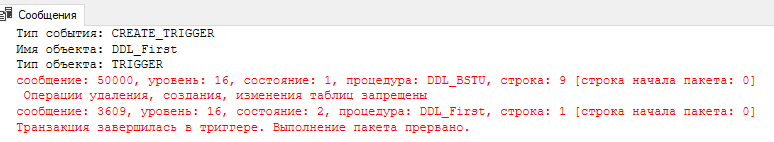
raiserror( N'операции с таблицей Progress запрещены', 16, 1);

rollback;

end;

alter table Progress Drop Column Note;

drop trigger DDL\_First on database



## **Лабораторная работа № 17.**

**Использование XML**

1. Разработать сценарий создания XML-документа в режиме PATH из таблицы **TEACHER** для преподавателей кафедры ИСиТ.

--PATH

select [Кафедра].PULPIT [кафедра],

[Преподаватель].Teacher\_name [имя\_преподавателя],

[Преподаватель].Gender [пол]

from Teacher [Преподаватель] join Pulpit [Кафедра]

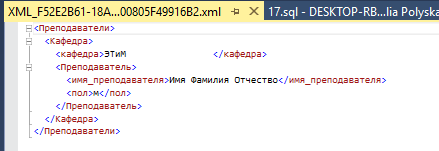
on [Преподаватель].Pulpit = [Кафедра].PULPIT

where [Кафедра].PULPIT in ('ИСИТ')

order by [Кафедра] for xml PATH,

root('Преподаватели'), elements;





--raw

select t.Teacher 'Наименование\_товара',

t.Teacher\_name 'Цена\_товара',

p.Pulpit 'Цена продажи'

from Teacher t join Pulpit p

on t.Pulpit = p.Pulpit

where p.Pulpit = 'ТДП' for xml RAW('Кафедра'),

root('Преподаватели'), elements;



--AUTO

select [Кафедра].PULPIT [кафедра],

[Преподаватель].Teacher\_name [имя\_преподавателя],

[Преподаватель].Gender [пол]

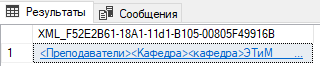
from Teacher [Преподаватель] join Pulpit [Кафедра]

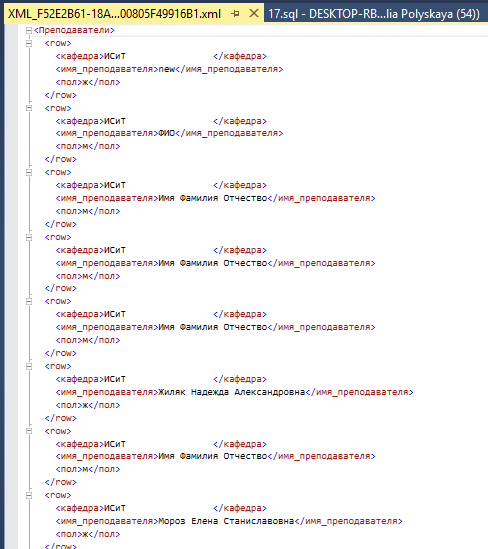
on [Преподаватель].Pulpit = [Кафедра].PULPIT

where [Кафедра].PULPIT in ('ТДП', 'ЭТиМ')

order by [Кафедра] for xml AUTO,

root('Преподаватели'), elements;





2. Разработать сценарий создания XML-документа в режиме AUTO на основе SELECT-запроса к таблицам **AUDITORIUM** и **AUDITORIUM\_TYPE,** которыйсодержит следующие столбцы: наименование аудитории, наименование типа аудитории и вместимость. Найти только лекционные аудитории.

--AUTO

select [Аудитория].AUDITORIUM [аудитория],

[тип\_аудитории].AUDITORIUM\_TYPE [тип\_аудитории],

[Аудитория].AUDITORIUM\_CAPACITY[Вместимость]

from AUDITORIUM\_TYPE [тип\_аудитории] join AUDITORIUM [Аудитория]

on [тип\_аудитории].AUDITORIUM\_TYPE = [Аудитория].AUDITORIUM

where [Аудитория].AUDITORIUM in ('408')

order by [Аудитория] for xml AUTO,

root('ЛК'), elements;



3.Разработать XML-документ, содержащий данные о трех новых учебных дисциплинах, которые следует добавить в таблицу **SUBJECT**.

Разработать сценарий, извлекающий данные о дисциплинах из XML-документа и добавляющий их в таблицу **SUBJECT**.

При этом применить системную функцию **OPENXML** и конструкцию INSERT… SELECT.

declare @h int = 1,

@x varchar(2000) = '<?xml version="1.0" encoding="windows-1251" ?>

<Subject>

<Subject="КГиГ" Subject\_name="Компьютерная геометрия и графика" Pulpit="РИТ" />

<Subject="БД" Subject\_name="Базы данных" Pulpit="РИТ" />

<Subject="ОЗИ" Subject\_name="Основы защиты информации" Pulpit="РИТ" />

</Subject>';

exec sp\_xml\_preparedocument @h output, @x; -- подготовка документа

INSERT SUBJECT select [Subject] ,[Subject\_name], [Pulpit] from openxml(@h, '/UNIVER/SUBJECT', 0)

with([Subject] char(10), [Subject\_name] varchar(100), [Pulpit] char(10) )

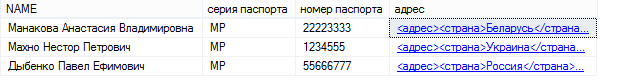
exec sp\_xml\_removedocument @h; -- удаление документа

4.Используя таблицу **STUDENT** разработать XML-структуру, содержащую паспортные данные студента: серию и номер паспорта, личный номер, дата выдачи и адрес прописки.

Разработать сценарий, в который включен оператор INSERT, добавляющий строку с XML-столбцом.

Включить в этот же сценарий оператор UPDATE, изменяющий столбец **INFO** у одной строки таблицы **STUDENT** и оператор SELECT, формирующий результирующий набор, аналогичный представленному на рисунке.

В SELECT-запросе использовать методы QUERY и VALUEXML-типа.



insert into STUDENT(IDGROUP, NAME, BDAY, INFO) values(22, 'Самаль Антон Дмитриевич', '',

'<студент>

<паспорт серия="КН" номер="2170227" дата="12.02.2012" />

<телефон>+375447077796</телефон>

<адрес>

<страна>Беларусь</страна>

<город>Сморгонь</город>

<улица>Западная</улица>

<дом>4</дом>

<квартира>28</квартира>

</адрес>

</студент>');

select \* from STUDENT where NAME = 'Самаль Антон Дмитриевич';

update STUDENT set INFO = '<студент>

<паспорт серия="КН" номер="2170227" дата="12.02.2012" />

<телефон>+375447077796</телефон>

<адрес>

<страна>Беларусь</страна>

<город>Сморгонь</город>

<улица>Западная</улица>

<дом>4</дом>

<квартира>28</квартира>

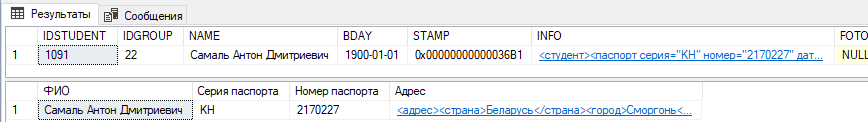
</адрес>

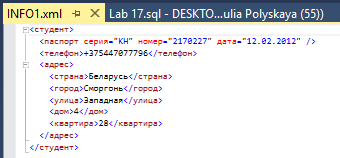
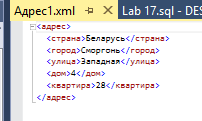
</студент>'

where NAME = 'Самаль Антон Дмитриевич';

select NAME[ФИО], INFO.value('(студент/паспорт/@серия)[1]', 'char(2)')[Серия паспорта], INFO.value('(студент/паспорт/@номер)[1]', 'varchar(20)')[Номер паспорта], INFO.query('/студент/адрес')[Адрес] from STUDENT where NAME = 'Самаль Антон Дмитриевич';

delete STUDENT where NAME = 'Самаль Антон Дмитриевич';



5. Изменить (ALTER TABLE) таблицу **STUDENT** в базе данных **UNIVER** таким образом, чтобы значения *типизированного* столбца с именем **INFO** контролировались коллекцией XML-схем (XML SCHEMACOLLECTION), представленной в правой части.

Разработать сценарии, демонстрирующие ввод и корректировку данных (операторы INSERT и UPDATE) в столбец **INFO** таблицы **STUDENT**, как содержащие ошибки, так и правильные.

Разработать другую XML-схему и добавить ее в коллекцию XML-схем в БД UNIVER**.**

create xml schema collection Student as

N'<?xml version="1.0" encoding="utf-16" ?>

<xs:schema attributeFormDefault="unqualified"

elementFormDefault="qualified"

xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="студент">

<xs:complexType><xs:sequence>

<xs:element name="паспорт" maxOccurs="1" minOccurs="1">

<xs:complexType>

<xs:attribute name="серия" type="xs:string" use="required" />

<xs:attribute name="номер" type="xs:unsignedInt" use="required"/>

<xs:attribute name="дата" use="required" >

<xs:simpleType> <xs:restriction base ="xs:string">

<xs:pattern value="[0-9]{2}.[0-9]{2}.[0-9]{4}"/>

</xs:restriction> </xs:simpleType>

</xs:attribute>

</xs:complexType>

</xs:element>

<xs:element maxOccurs="3" name="телефон" type="xs:unsignedInt"/>

<xs:element name="адрес"> <xs:complexType><xs:sequence>

<xs:element name="страна" type="xs:string" />

<xs:element name="город" type="xs:string" />

<xs:element name="улица" type="xs:string" />

<xs:element name="дом" type="xs:string" />

<xs:element name="квартира" type="xs:string" />

</xs:sequence></xs:complexType> </xs:element>

</xs:sequence></xs:complexType>

</xs:element></xs:schema>';

