**Лабораторная работа № 1. Знакомство с инструментами СУБД Microsoft SQL Server 2008**

Создавать базы данных можно с использованием инспектора объектов Object Explorer утилиты SQL Server Management Studio, входящей в состав программного обеспечения СУБД Microsoft SQL Server 2008 (MSS), или с помощью языка Transact-SQL. В данной лабораторной работе рассматривается первый вариант.

Я создала базу данных с помощью утилиты SQL Server Management Studio, под названием ПРОДАЖИ.

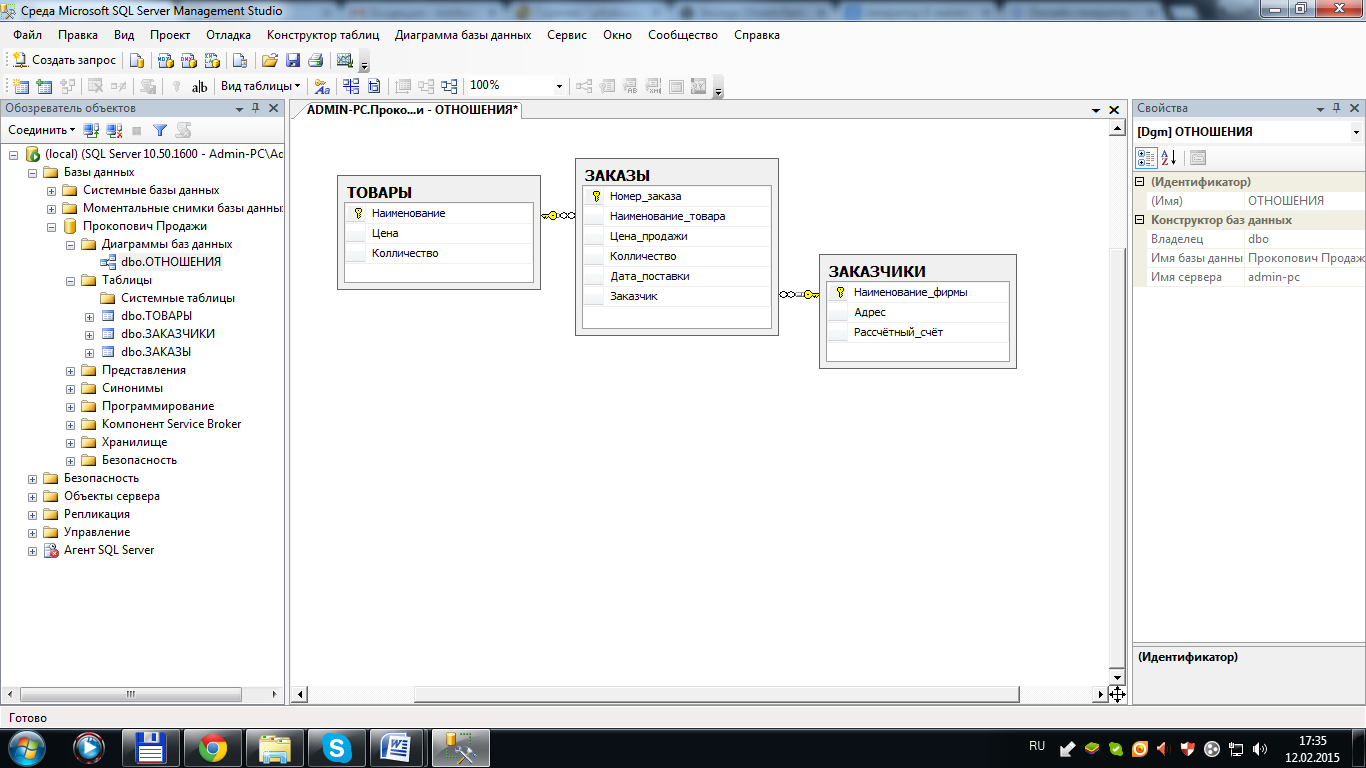
В ней содержится три таблицы.

Первая с именем ТОВАРЫ, содержит поля: Наименование (тип nvarchar(20)), Цена (тип real), Количество (тип int), первый столбец ключевой.

Вторая таблица с именем ЗАКАЗЧИКИ и полями: Наименование*\_*фирмы(тип nvarchar(20)), Адрес (тип nvarchar(50)), Расчетный\_счет (тип nvarchar(15)) , первый столбец ключевой.

Третья таблица с именем ЗАКАЗЫ, поля: Номер\_заказа (тип nvarchar(10)), Наименование\_товара (тип nvarchar(20)), Цена\_продажи (тип real), Количество (тип int), Дата\_поставки (тип date), Заказчик (тип nvarchar(20)), первый столбец ключевой.

При объединении получается такая структура:

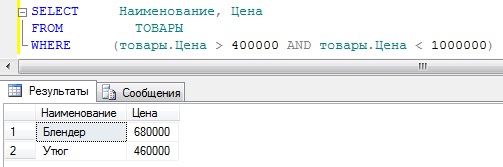


Далее нужно было создать запросы:

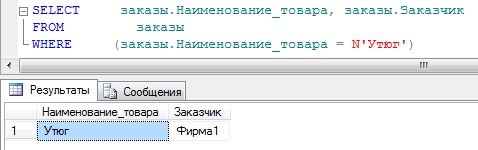
-определить товары, поставки которых должны осуществиться после некоторой даты:



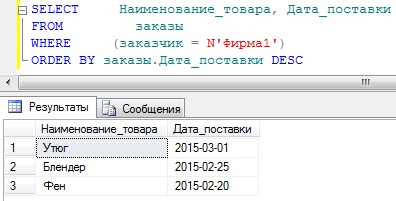
-найти товары, цена которых находится в некоторых пределах:



-определить названия фирм, заказавших конкретный товар:



-найти заказы определенной фирмы по ее названию, отсортировать их по датам поставки:



**Лабораторная работа № 2. Проектирование баз данных. Нормализация.**

В этой лабораторной работе я проектировала свою базу данных, исследовала предметную область с целью определения объектов, нормализовала данные и устанавливала связи между ними. **Нормализация** данных − это процесс, в результате выполнения которого таблицы базы данных проверяются на наличие зависимостей между столбцами таблицы. Если такие зависимости существуют, то таблица разделяется на несколько таблиц. Если одна из этих сгенерированных таблиц все еще содержит зависимости данных, то процесс нормализации должен повторяться, пока не будут устранены все зависимости.

Мой вариант 14: **Грузовые перевозки.** Компания осуществляет перевозки по различным маршрутам. Информационные поля: Название маршрута, Дальность, Количество дней в пути, Оплата, Фамилия водителя, Имя, Отчество, Стаж, Дата отправки, Дата возвращения.

1. Изучила способ приведения информации к первой нормальной форме:

Чтобы таблица соответствовала **1-й нормальной форме**, необходимо, чтобы все значения ее полей были неделимыми и не вычисляемыми, а все записи – уникальными (не должно быть полностью совпадающих строк).

Т1: Название маршрута, Дальность, Количество дней в пути, Оплата, Дата отправки, Дата возвращения;

Т2: Фамилия водителя, Имя, Отчество, Стаж.

Т3: Название маршрута, Дальность, Количество дней в пути, Оплата;

Т4: Дата отправления, Дата возвращения.

1. Изучила способ приведения ко второй нормальной форме:

Чтобы таблица соответствовала **2-й нормальной форме** (2NF), необходимо, чтобы она уже находилась в 1-й нормальной форме и все не ключевые поля полностью зависели от ключевого.

Т1: Номер маршрута, Дальность, Количество дней в пути, Оплата;

Т2: Номер поездки, Номер маршрута, Код водителя, Дата отправления;

Т3: Код водителя ФИО, Фамилия Имя Отчество, Стаж.

1. И перешла к третьей нормальной форме:

Для перехода к **3-й нормальной форме** (3NF), необходимо обеспечить, чтобы все таблицы БД находились во 2-й нормальной форме и все не ключевые поля в таблицах зависели только от ключа таблицы и не зависели друг от друга.

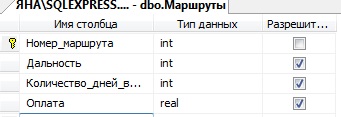
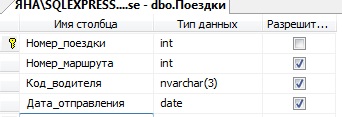
Маршруты: Номер маршрута, Дальность, Количество дней в пути, Оплата;

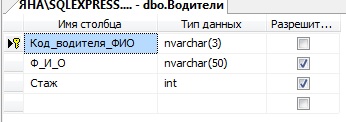
Поездки: Номер поездки, Номер маршрута, Код водителя, Дата отправления;

Водители: Код водителя ФИО, Фамилия Имя Отчество, Стаж.

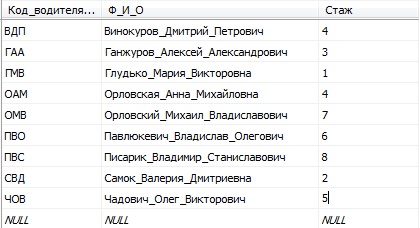
Подчеркнутые поля – ключевые.

1. Определила типы данных для своих полей:



1. Создала свою базу данных Орловская\_My\_Base с помощью утилиты SSMS, установила связи и заполнила таблицы информацией:



**Лабораторная работа № 3. T-SQL – язык реляционной базы данных**

Операторы языка делятся на несколько групп, основными из которых является язык определения данных (**Data Definition Language**, **DDL**) и язык манипулирования данными (**Data Manipulation Language**, **DML**). Язык **DDL** содержит три обобщенных оператора: **create** *объект* (создание объекта базы данных), **alter** *объект* (изменение характери­стик объекта) и **drop** *объект* (удаление существующего объекта). Эти опера­торы создают, изменяют и удаляют объекты базы данных, такие как сама база данных, таблицы, столбцы и индексы.

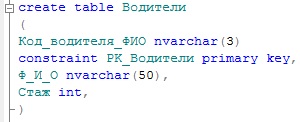
Язык **DML** содержит операторы, которые манипули­руют данными, осуществляя выборку информации (**SELECT**), добавление (**INSERT**), удаление (**DELETE**) и изменение (**UPDATE**).

При записи операторов можно использовать на клавиатуре любой регистр.

1. Я создала свою базу данных с именем Орловская\_MyBase:



1. С помощью оператора CREATE я создала 3 таблицы: Водители, Маршруты, Поездки.



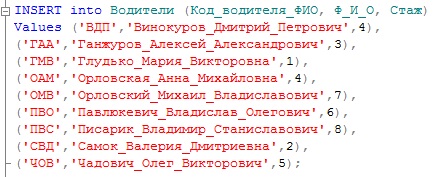
1. С помощью оператора ALTER я внесла некоторые изменения в структуру:

- добавила столбец Дата\_рождения в таблицу Водители (с помощью ключевого слова ADD);

- и удалила столбец с помощью ключевого слова DROP:



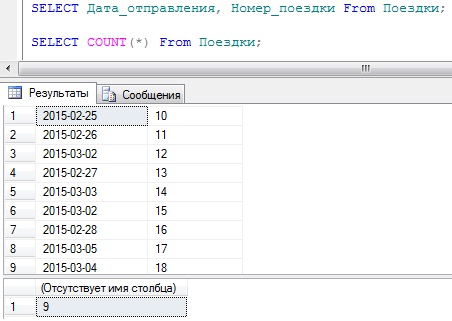
1. С помощью оператора INSERT я внесла строки в таблицы:



1. Операции с SELECT – основной оператор для поиска информации в базе данных.

- Вывод содержимого двух столбцов (Дата\_отправления, Номер\_поездки) из таблицы Поездки;

- Вывод количества строк из таблицы Поездки:



1. С помощью оператора DELETE я удалила из таблицы Поездки строку, в которой значение столбца Номер\_поездки равно 19;

А с помощью UPDATE – изменила своё значение каждая строка столбца Номер\_поездки на 9:



**Лабораторная работа № 4. Настройка базы данных при ее создании.**

1. Создала базу данных CreateORL\_BSTU.
2. Расположить таблицы в файловых группах:

use master

go

create database ORL\_BSTU

on primary

( name = N'ORL\_BSTU\_mdf', filename = N'D:\УЧЁБА\БД\лаб4\ORL\_BSTU\_mdf.mdf',

size = 5Mb, maxsize=10Mb, filegrowth=1Mb),

( name = N'ORL\_BSTU\_ndf', filename = N'D:\УЧЁБА\БД\лаб4\ORL\_BSTU\_ndf.ndf',

size = 5Mb, maxsize=10Mb, filegrowth=10%),

filegroup FG1

( name = N'ORL\_BSTU11\_ndf', filename = N'D:\УЧЁБА\БД\лаб4\ORL\_BSTU11\_ndf.ndf',

size = 10Mb, maxsize=15Mb, filegrowth=1Mb),

( name = N'ORL\_BSTU12\_ndf', filename = N'D:\УЧЁБА\БД\лаб4\ORL\_BSTU12\_ndf.ndf',

size = 2Mb, maxsize=5Mb, filegrowth=1Mb),

filegroup FG2

( name = N'ORL\_BSTU21\_ndf', filename = N'D:\УЧЁБА\БД\лаб4\ORL\_BSTU21\_ndf.ndf',

size = 5Mb, maxsize=10Mb, filegrowth=1Mb),

( name = N'ORL\_BSTU22\_ndf', filename = N'D:\УЧЁБА\БД\лаб4\ORL\_BSTU22\_ndf.ndf',

size = 2Mb, maxsize=5Mb, filegrowth=1Mb)

log on

( name = N'ORL\_BSTU\_log', filename=N'D:\УЧЁБА\БД\лаб4\ORL\_BSTU\_log.ldf',

size=5Mb, maxsize=UNLIMITED, filegrowth=1Mb)

go

1. Аналогичный сценарий я создала и для своей базы ORL\_MyBASE:

use master

go

create database ORL\_MyBase

on primary

( name = N'ORL\_MyBase\_mdf', filename = N'D:\УЧЁБА\БД\лаб4\ORL\_MyBase\_mdf.mdf',

size = 10240Kb, maxsize=UNLIMITED, filegrowth=1024Kb),

( name = N'ORL\_MyBase\_ndf', filename = N'D:\УЧЁБА\БД\лаб4\ORL\_MyBase\_ndf.ndf',

size = 10240KB, maxsize=1Gb, filegrowth=25%),

filegroup FG1

( name = N'ORL\_MyBase\_fg1\_1', filename = N'D:\УЧЁБА\БД\лаб4\ORL\_MyBase\_fgq-1.ndf',

size = 10240Kb, maxsize=1Gb, filegrowth=25%),

( name = N'ORL\_MyBase\_fg1\_2', filename = N'D:\УЧЁБА\БД\лаб4\ORL\_MyBase\_fgq-2.ndf',

size = 10240Kb, maxsize=1Gb, filegrowth=25%)

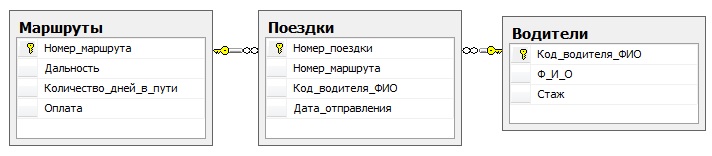
log on

( name = N'ORL\_MyBase\_log', filename=N'D:\УЧЁБА\БД\лаб4\ORL\_MyBase\_log.ldf',

size=10240Kb, maxsize=2048Gb, filegrowth=10%)

go

1. И с помощью SSMS построила диаграмму:



**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).

1. SELECT \* FROM Маршруты;

- этот запрос позволяет сформировать результирующий набор, содержащий все строки и столбцы таблицы Маршруты.

1. SELECT Код\_преподавателя, Код\_кафедры from TEACHER;

- выводит все строки столбцов Код\_преподавателя и Код\_кафедры таблицы TEACHER.

3. SELECT Дата\_отправления, Код\_водителя\_ФИО from Поездки where Дата\_отправления='2015-02-25';

4. SELECT Distinct Количество\_дней\_в\_пути FROM Маршруты;

Секция DISTINCT позволяет не выводить повторяющиеся строки.

5. SELECT Дата\_отправления, Код\_водителя\_ФИО FROM Поездки Order by Дата\_отправления Desc;

Секция ORDER BY позволяет отсортировать результаты по дате отправления.

6. SELECT Distinct Дата\_отправления, Код\_водителя\_ФИО from Поездки where Дата\_отправления BETWEEN '2015-02-28' and '2015-03-02';

Предикат BETWEEN позволяет выбирать строки с заданным диапазоном значений.

7. SELECT Distinct Ф\_И\_О, Стаж from Водители where Стаж IN ('1', '2', '3');

- вывод инициал водителей, у которых стаж равен 1, 2 или 3 годам.

8. SELECT Ф\_И\_О, Стаж from Водители where Ф\_И\_О LIKE 'П%';

- вывод инициал водителей, у которых фамилия начинается с буквы П.

9. Создание временной таблицы:

use ORL\_MyBase;

CREATE table #Водители

( ФИО nvarchar(100), Дата\_рождения date );

INSERT INTO #Водители (ФИО, Дата\_рождения)

values ( 'Хартанович Вадим Александрович','11.03.1991'),

( 'Горбач Никита Владимирович', '07.12.1989'),

( 'Зыков Алексей Дмитриевич', '12.10.1985'),

( 'Борисевич Николай Анатольевич', '09.11.1985'),

( 'Медведев Иван Андреевич', '04.07.1979'),

( 'Шенец Михаил Викторович', '08.01.1968'),

('Шитик Денис Игоревич', '02.08.1987');

И пару запросов к нему:

SELECT \* FROM #Водители;

SELECT ФИО, Дата\_рождения from #Водители where ФИО Like 'Ш%';

**Лабораторная работа № 6. Многотабличные SELECT-запросы**

Соединение таблиц INNER JOIN (внутреннее соединение) наиболее часто используемый вид соединения реляционных таблиц (таблицы связанные между собой).

1. SELECT AUDITORIUM.Код\_типа\_аудитории, AUDITORIUM\_TYPE.Наименование\_типа\_аудитории

From AUDITORIUM, AUDITORIUM\_TYPE

where AUDITORIUM.Код\_типа\_аудитории = AUDITORIUM\_TYPE.Код\_типа\_аудитории

And AUDITORIUM\_TYPE.Наименование\_типа\_аудитории Like '%компьютер%';

1. SELECT

Case

when ( PROGRESS.Оценка = 6) then 'шесть'

when ( PROGRESS.Оценка = 7) then 'семь'

else 'восемь'

end AS Оценка,

STUDENT.ФИО, SUBJECT.Наименование\_дисциплины, PULPIT.Наименование\_кафедры, FACULTY.Наименование\_факультета, GROUPS.Код\_специальности

From ((((STUDENT Inner Join PROGRESS

On STUDENT.Код\_студента=PROGRESS.Код\_студента and PROGRESS.Оценка between 6 and 8)

Inner Join SUBJECT on SUBJECT.Код\_дисциплины=PROGRESS.Код\_дисциплины)

Inner Join PULPIT on SUBJECT.Код\_кафедры=PULPIT.Код\_кафедры)

Inner Join FACULTY on FACULTY.Код\_факльтета =PULPIT.Код\_факультета)

Inner Join GROUPS on GROUPS.Код\_факультета =FACULTY.Код\_факультета

ORDER BY PROGRESS.Оценка;

- перечень студентов, получивших на экзамене оценки от 6 до 8 (оценки должны быть записаны прописью).

1. SELECT isnull (TEACHER.ФИО\_преподавателя, '\*\*\*') AS Преподаватель,

PULPIT.Наименование\_кафедры AS Кафедра

FROM PULPIT Left Outer JOIN TEACHER

ON PULPIT.Код\_кафедры = TEACHER.Код\_кафедры

- если на кафедре нет преподавателя, то в столбце Преподаватель должна быть выведена строка \*\*\*.

Встроенная функция **isnull** принимает два параметра и проверяет их значения на NULL слева направо. Функция возвращает первое значение, не равное NULL.

1. SELECT AUDITORIUM.Код\_типа\_аудитории, AUDITORIUM\_TYPE.Наименование\_типа\_аудитории

From AUDITORIUM CROSS Join AUDITORIUM\_TYPE

where AUDITORIUM.Код\_типа\_аудитории = AUDITORIUM\_TYPE.Код\_типа\_аудитории;

При использовании соединения CROSS JOIN каждая строка одной таблицы соединяется с каждой строкой другой таблицы.

Далее для своей базы:

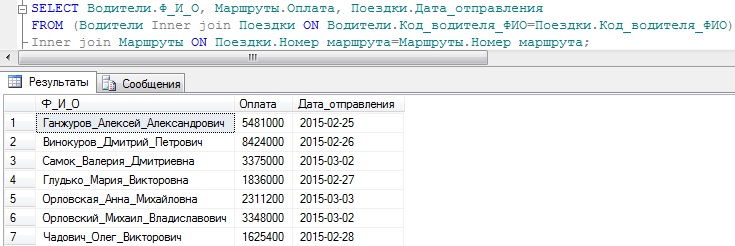
1.SELECT Водители.Ф\_И\_О, Поездки.Дата\_отправления

From Водители Inner Join Поездки

On Водители.Код\_водителя\_ФИО = Код\_водителя\_ФИО

And Водители.Ф\_И\_О Like '%Александр%';

2.



3.SELECT

Водители.Ф\_И\_О, Водители.Стаж, Маршруты.Оплата, Поездки.Дата\_отправления

FROM (Водители Inner join Поездки ON Водители.Код\_водителя\_ФИО=Поездки.Код\_водителя\_ФИО)

Inner join Маршруты ON Поездки.Номер\_маршрута=Маршруты.Номер\_маршрута

ORDER BY

(Case when (Водители.Стаж=7) then 1

when (Водители.Стаж=5) then 2

else 3

end);

**Лабораторная работа № 7. Использование подзапросов**

**Подзапрос** – это SELECT-запрос, который выполняется в рамках другого запроса. Подзапросы могут применяться в секции WHERE. Подзапросы бывают двух видов: коррелируемые и независимые. *Коррелируемый* подзапрос зависит от внешнего запроса и выполняется для каждой строки результирующего набора. *Независимый* подзапрос не зависит от внешнего запроса и выполняется только один раз, но результат его выполнения подставляется в каждую строку результирующего набора. В SELECT-списке допускается применять только такие подзапросы, которые формируют скалярный результирующий набор (набор, состоящий из одной строки и одного столбца).

1. SELECT

FACULTY.Код\_факультета, PULPIT.Наименование\_кафедры,PROFESSION.Наименование\_специальности FROM FACULTY, PULPIT, PROFESSION

Where FACULTY.Код\_факультета=PULPIT.Код\_факультета and FACULTY.Код\_факультета=PROFESSION.Код\_факультета and

Наименование\_специальности In (Select Наименование\_специальности FROM PROFESSION

Where (Наименование\_специальности Like '%технология%' or Наименование\_специальности Like '%технологии%'))

- использовать в секции WHERE предикат IN c некоррелированным подзапросом к таблице **PROFESSION.**

**Для своей базы:**

SELECT Поездки.Дата\_отправления,Маршруты.Количество\_дней\_в\_пути,Водители.Ф\_И\_О

FROM Поездки,Маршруты,Водители

Where Поездки.Номер\_маршрута=Маршруты.Номер\_маршрута and

Ф\_И\_О In (Select Ф\_И\_О FROM Водители

Where (Ф\_И\_О Like '%Влад%') )

2. SELECT FACULTY.Код\_факультета, PULPIT.Наименование\_кафедры, PROFESSION.Наименование\_специальности

FROM (FACULTY Inner JOIN PULPIT ON FACULTY.Код\_факультета = PULPIT.Код\_факультета)

Inner JOIN PROFESSION ON FACULTY.Код\_факультета = PROFESSION.Код\_факультета

Where Наименование\_специальности In (Select Наименование\_специальности FROM PROFESSION

Where (Наименование\_специальности Like '%технология%' or Наименование\_специальности Like '%технологии%'))

- тот же подзапрос в конструкции INNER JOIN секции FROM внешнего запроса.

3. SELECT FACULTY.Код\_факультета, PULPIT.Наименование\_кафедры, PROFESSION.Наименование\_специальности

FROM (FACULTY Inner JOIN PULPIT ON FACULTY.Код\_факультета = PULPIT.Код\_факультета)

Inner JOIN PROFESSION ON FACULTY.Код\_факультета = PROFESSION.Код\_факультета

Where (Наименование\_специальности Like '%технология%' or Наименование\_специальности Like '%технологии%')

- без использования подзапроса.

4. SELECT distinct Количество\_дней\_в\_пути,Оплата from Маршруты a

where Дальность = (select top(1) Дальность from Маршруты aa

where aa.Количество\_дней\_в\_пути = a.Количество\_дней\_в\_пути )

order by Оплата desc;

- оплата по убыванию (коррелируемый подзапрос).

5. SELECT Наименование\_факультета from FACULTY

Where not exists (select \* from PULPIT

Where PULPIT.Код\_факультета = FACULTY.Код\_факультета)

- Операция EXISTS формирует значение «истина», если результирующий набор подзапроса содержит хотя бы одну строку, в противоположном случае − значение «ложь».

6. SELECT

(select avg(Оценка) from PROGRESS

where Код\_дисциплины like 'ОАиП' ) [ОАиП],

(select avg(Оценка) from PROGRESS

where Код\_дисциплины like 'СУБД' ) [СУБД],

(select avg(Оценка) from PROGRESS

where Код\_дисциплины like 'БД' )[БД]

7. SELECT Код\_дисциплины, Оценка from PROGRESS

Where Оценка >=all (select max(Оценка) from PROGRESS

where Код\_дисциплины like 'ОАиП');

8. SELECT Код\_дисциплины, Оценка from PROGRESS

Where Оценка >any (select max(Оценка) from PROGRESS

where Код\_дисциплины like 'ОАиП').

**Лабораторная работа № 8. Группировка данных**

Основное назначение группировки с помощью секции GROUP BY – разбиение множества строк, сформированного секциями FROM и WHERE, на группы в соответствии со значениями в заданных столбцах, а также выполнение вычислений над группами строк с помощью наиболее часто используемых функций: AVG (вычисление среднего значения), COUNT (вычисление количества строк), MAX (вычисление максимального значения), MIN (вычисление минимального значения), SUM (вычисление суммы значений).

При использовании секции GROUP BY в SELECT-списке допускается указывать только те столбцы, по которым осу-ществляется группировка.

1. SELECT AUDITORIUM.Код\_типа\_аудитории,

MAX(Вместимость) [Максимальная вместимость], MIN(Вместимость) [Минимальная вместимость],

AVG(Вместимость) [Средняя вместимость], SUM(Вместимость) [Суммарная вместимость],

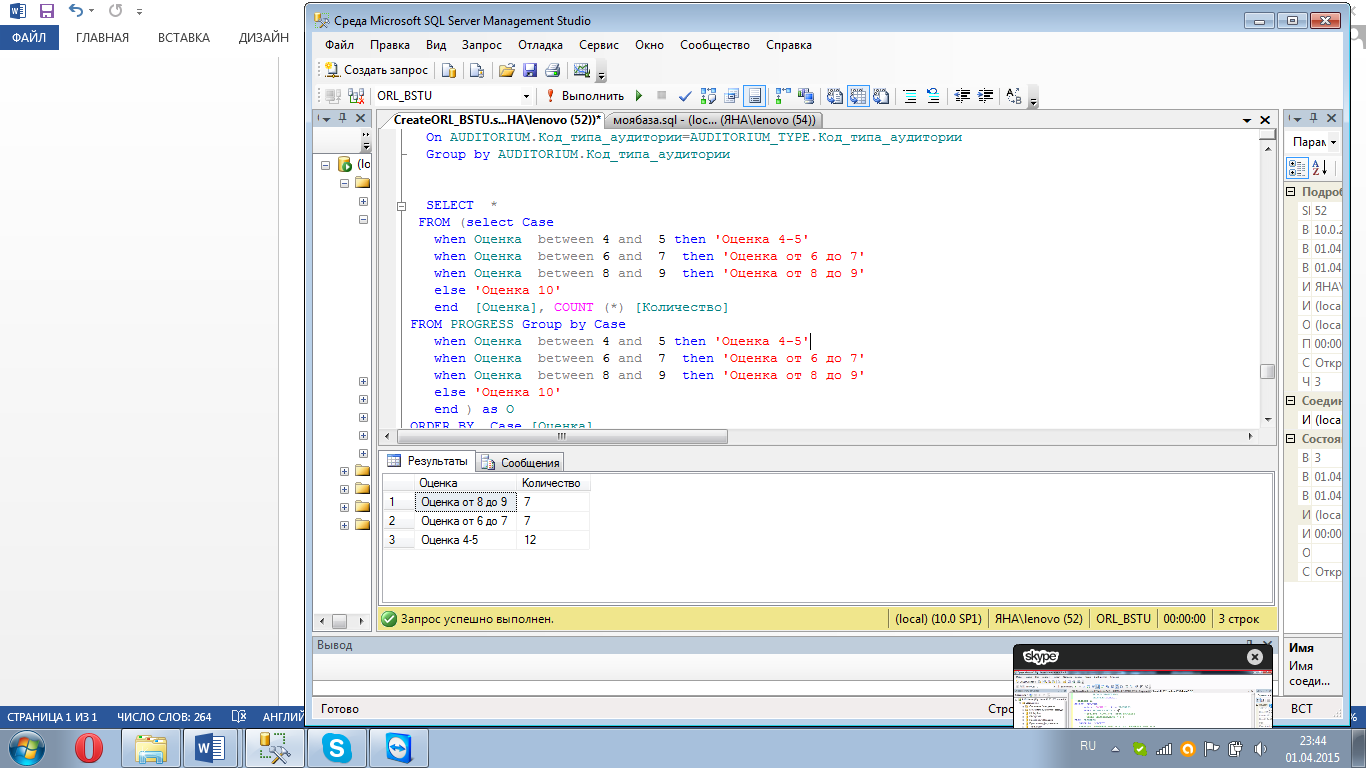
COUNT(\*) [Общее количество] FROM AUDITORIUM Inner Join AUDITORIUM\_TYPE

On AUDITORIUM.Код\_типа\_аудитории=AUDITORIUM\_TYPE.Код\_типа\_аудитории

Group by AUDITORIUM.Код\_типа\_аудитории;



2. SELECT \*

 FROM (select Case

when Оценка between 4 and 5 then 'Оценка 4-5'

when Оценка between 6 and 7 then 'Оценка от 6 до 7'

when Оценка between 8 and 9 then 'Оценка от 8 до 9'

else 'Оценка 10'

end [Оценка], COUNT (\*) [Количество]

FROM PROGRESS Group by Case

when Оценка between 4 and 5 then 'Оценка 4-5'

when Оценка between 6 and 7 then 'Оценка от 6 до 7'

when Оценка between 8 and 9 then 'Оценка от 8 до 9'

else 'Оценка 10' end ) as 0

ORDER BY Case [Оценка]

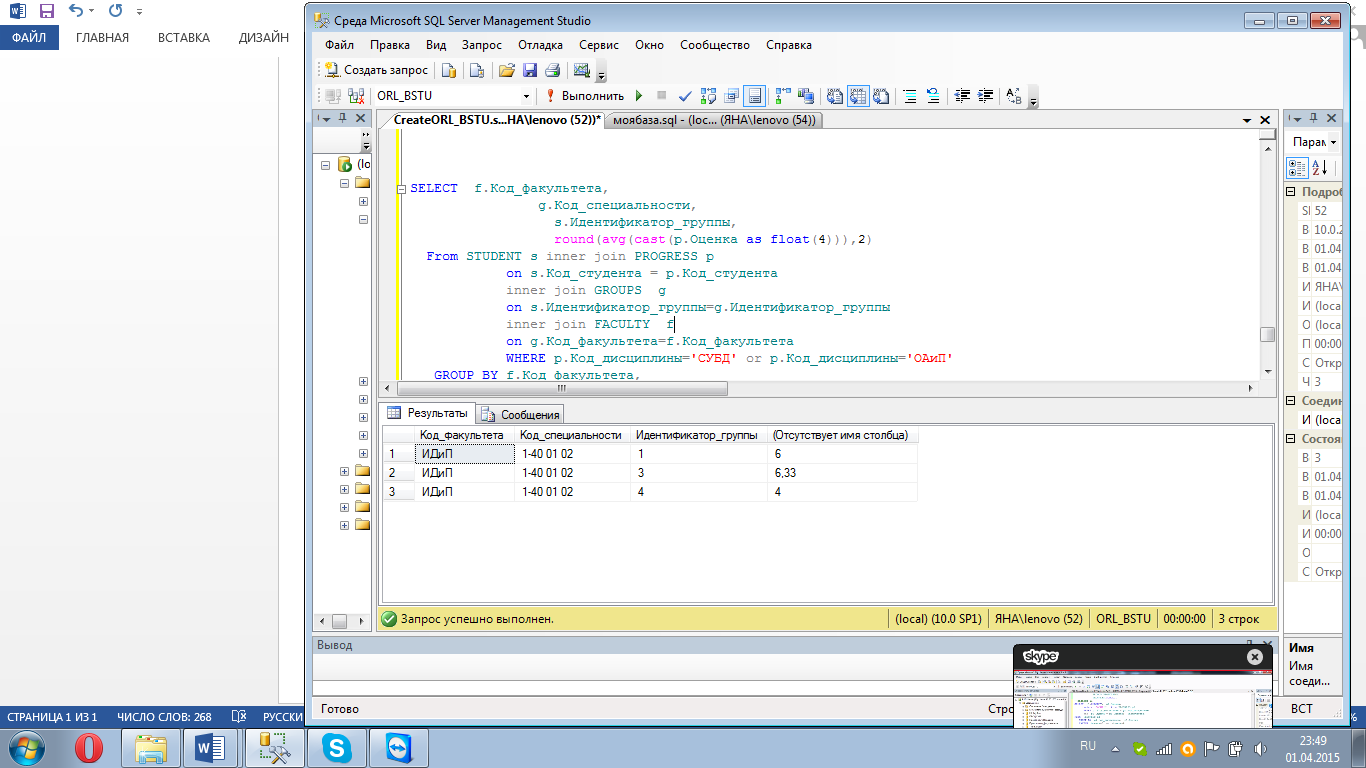
when 'оценка 4-5' then 3

when 'оценка от 6 до 7' then 2

when 'оценка от 8 до 9' then 1 else 0 end

3. Встроенная функция CAST используется в запросе значения цены в вещественное число.

Функция ROUND обеспечивает расчет значений с точностью до двух знаков после запятой.

SELECT f.Код\_факультета,  g.Код\_специальности, s.Идентификатор\_группы,

round(avg(cast(p.Оценка as float(4))),2)

From STUDENT s inner join PROGRESS p

on s.Код\_студента = p.Код\_студента

inner join GROUPS g

on s.Идентификатор\_группы=g.Идентификатор\_группы

inner join FACULTY f on g.Код\_факультета=f.Код\_факультета

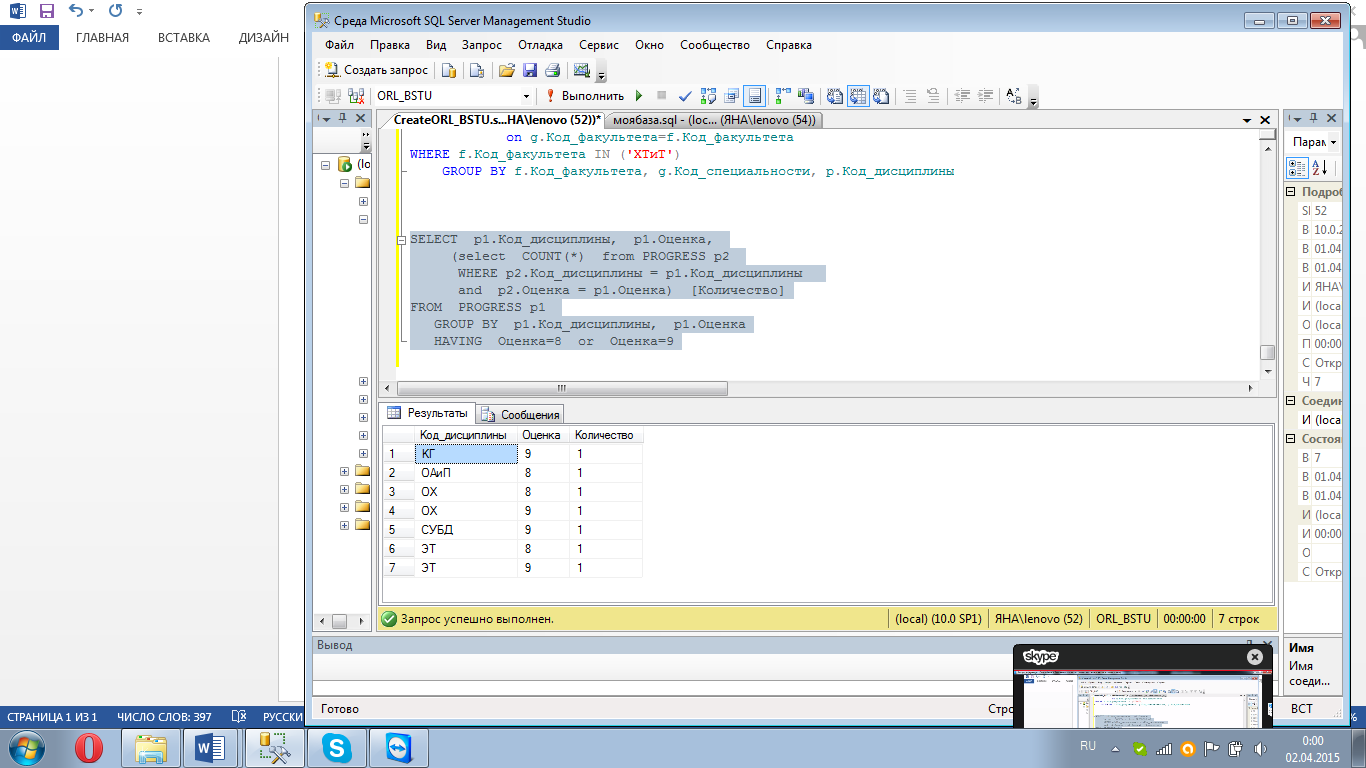
WHERE p.Код\_дисциплины='СУБД' or p.Код\_дисциплины='ОАиП'

GROUP BY f.Код\_факультета, g.Код\_специальности, s.Идентификатор\_группы;

4. Конструкции ROLLUP и CUBE применяются в секции GROUP BY и служат для вычисления значений агрегатных функций для подмножеств строк. Конструкция CUBE возвращает любую возможную комби¬нацию групп и итоговых строк.

1) GROUP BY ROLLUP (f.Код\_факультета, g.Код\_специальности, p.Код\_дисциплины);

2) GROUP BY CUBE(f.Код\_факультета, g.Код\_специальности, p.Код\_дисциплины);

5. 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 ре-зультирующий набор, если логическое выражение принимает значение «истина».

SELECT p1.Код\_дисциплины, p1.Оценка,

(select COUNT(\*) from PROGRESS p2

WHERE p2.Код\_дисциплины = p1.Код\_дисциплины

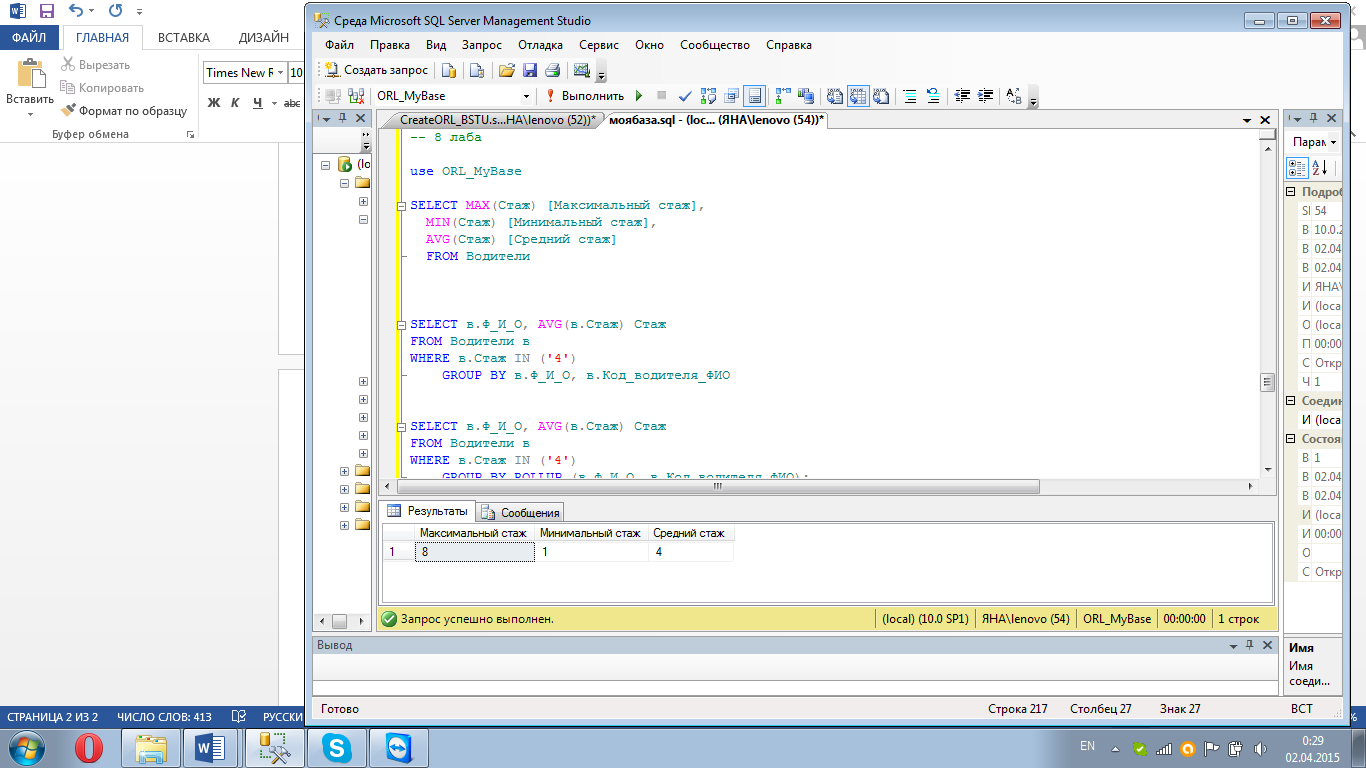
and p2.Оценка = p1.Оценка) [Количество]

FROM PROGRESS p1

GROUP BY p1.Код\_дисциплины, p1.Оценка

HAVING Оценка=8 or Оценка=9;

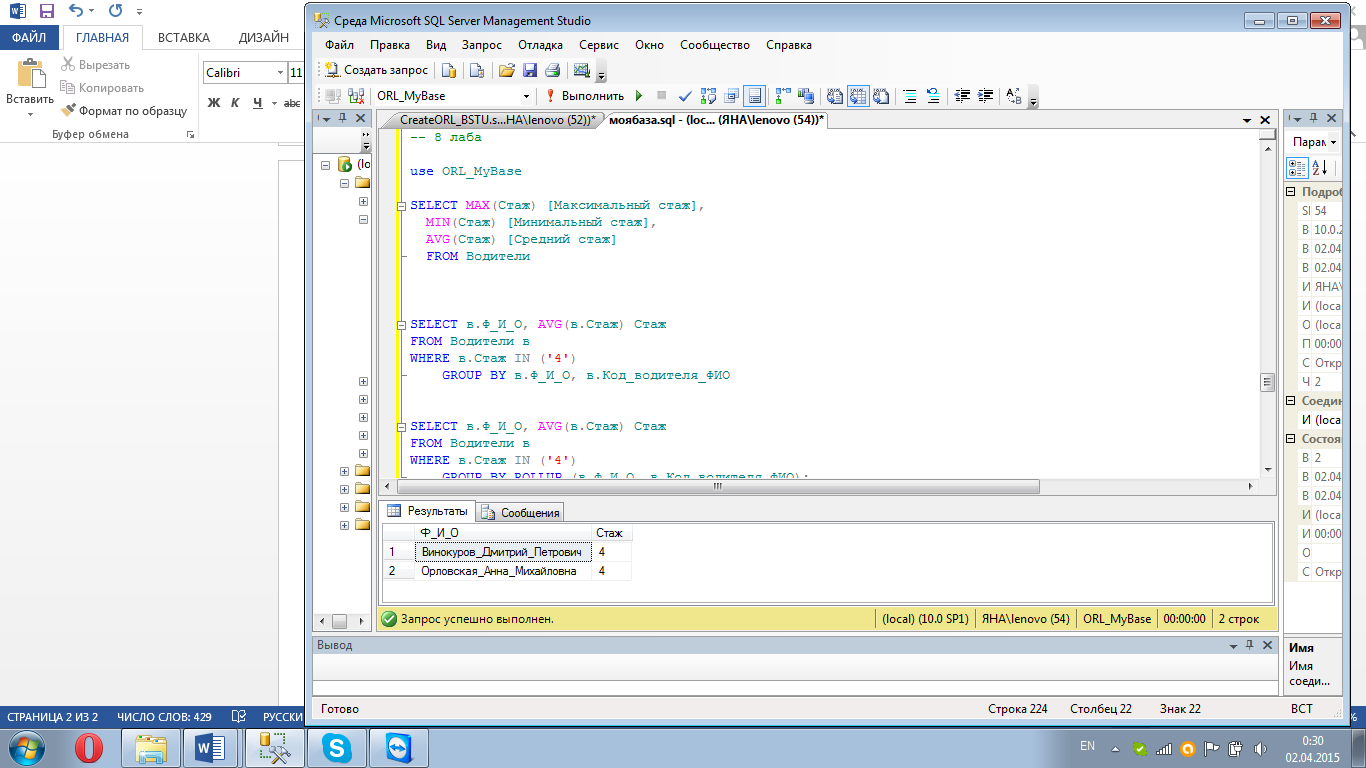
**Для своей базы:**

1. SELECT MAX(Стаж) [Максимальный стаж],

MIN(Стаж) [Минимальный стаж],

AVG(Стаж) [Средний стаж]

FROM Водители;

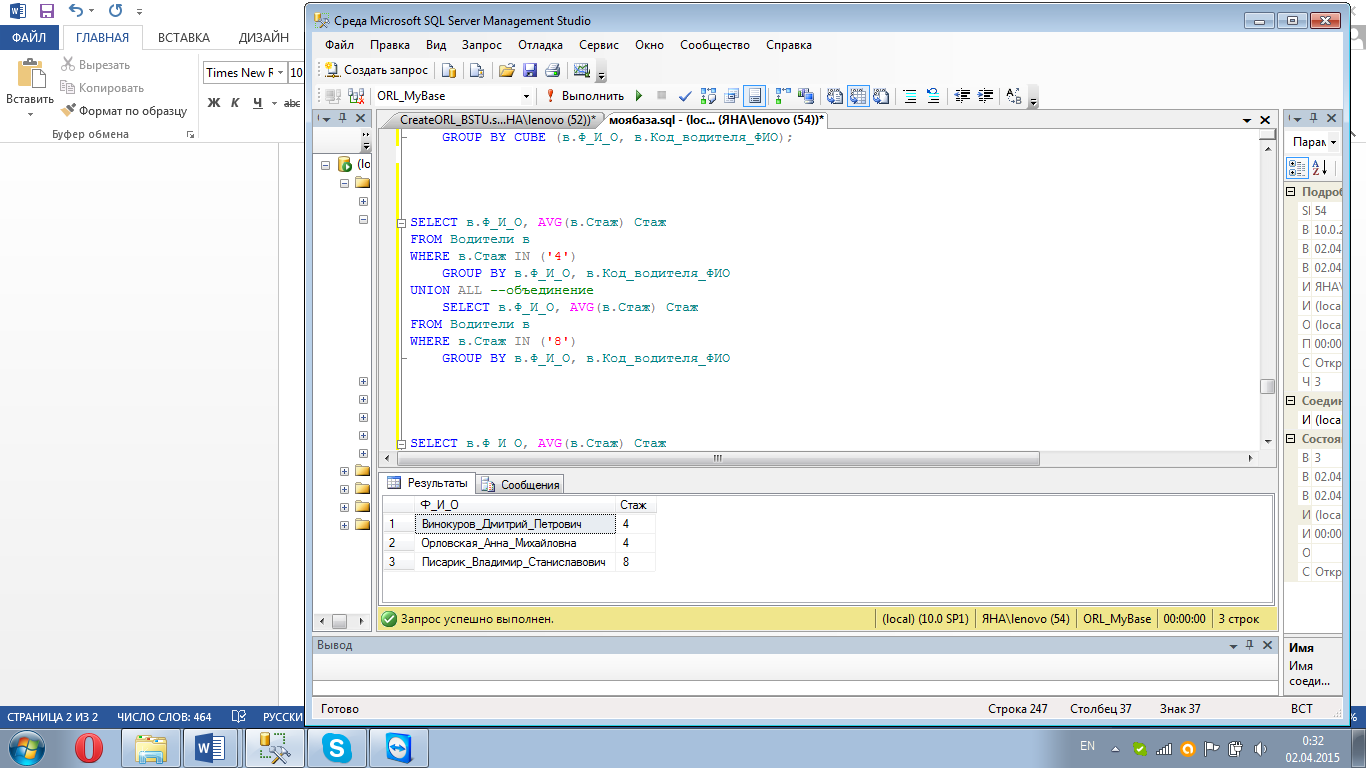
2. SELECT в.Ф\_И\_О, AVG(в.Стаж) Стаж

FROM Водители в

WHERE в.Стаж IN ('4')

GROUP BY в.Ф\_И\_О, в.Код\_водителя\_ФИО;

3. SELECT в.Ф\_И\_О, AVG(в.Стаж) Стаж

FROM Водители в

WHERE в.Стаж IN ('4')

GROUP BY в.Ф\_И\_О, в.Код\_водителя\_ФИО

UNION ALL --объединение

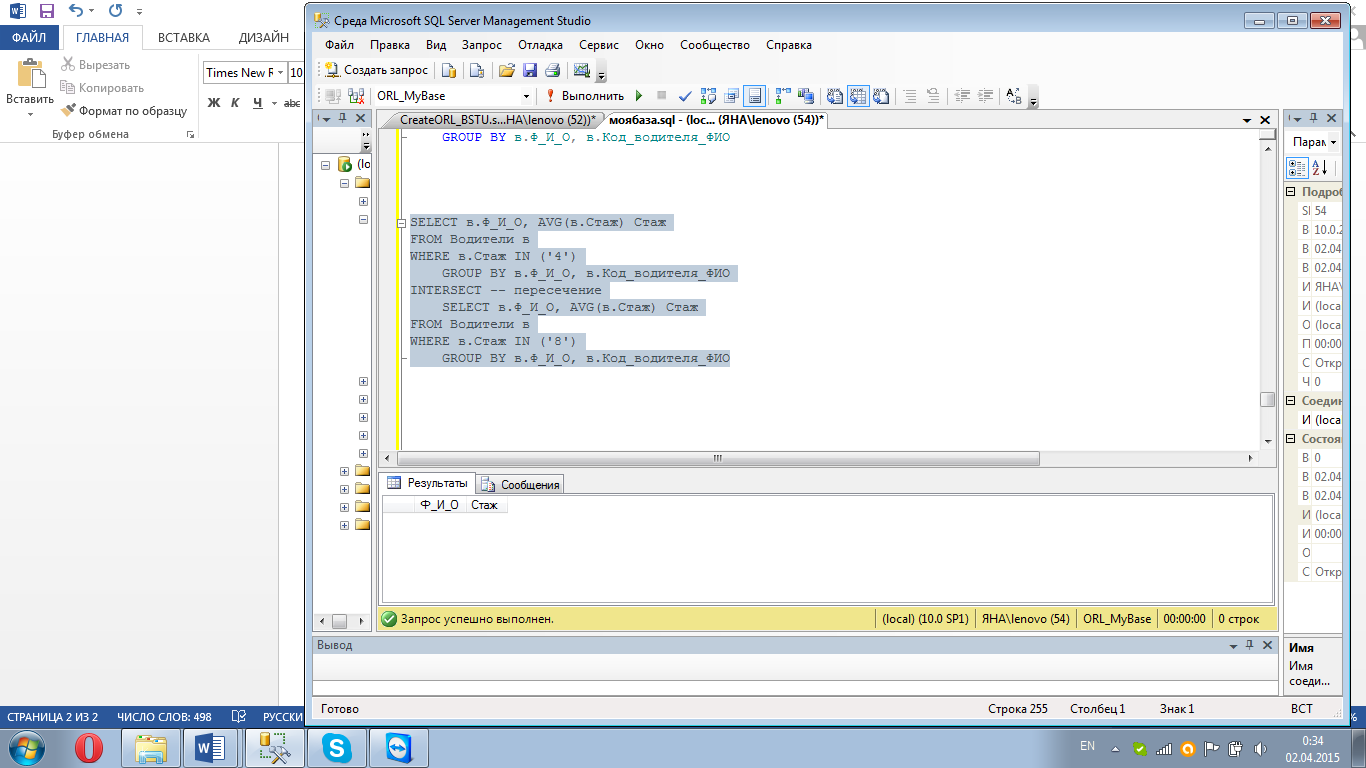
SELECT в.Ф\_И\_О, AVG(в.Стаж) Стаж

FROM Водители в

WHERE в.Стаж IN ('8')

GROUP BY в.Ф\_И\_О, в.Код\_водителя\_ФИО;

4. SELECT в.Ф\_И\_О, AVG(в.Стаж) Стаж

FROM Водители в

WHERE в.Стаж IN ('4')

GROUP BY в.Ф\_И\_О, в.Код\_водителя\_ФИО

INTERSECT -- пересечение

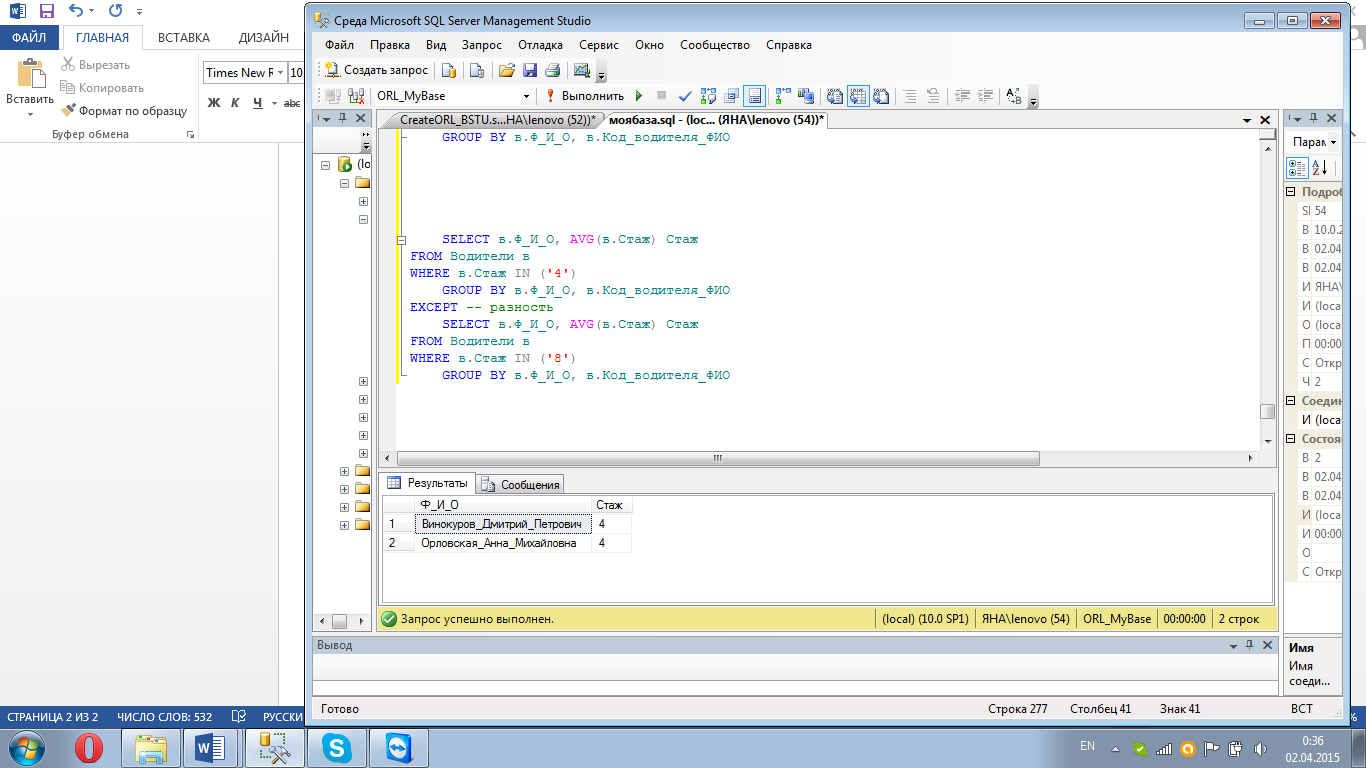
SELECT в.Ф\_И\_О, AVG(в.Стаж) Стаж

FROM Водители в

WHERE в.Стаж IN ('8')

GROUP BY в.Ф\_И\_О, в.Код\_водителя\_ФИО;

5. SELECT в.Ф\_И\_О, AVG(в.Стаж) Стаж

FROM Водители в

WHERE в.Стаж IN ('4')

GROUP BY в.Ф\_И\_О, в.Код\_водителя\_ФИО

EXCEPT -- разность

SELECT в.Ф\_И\_О, AVG(в.Стаж) Стаж

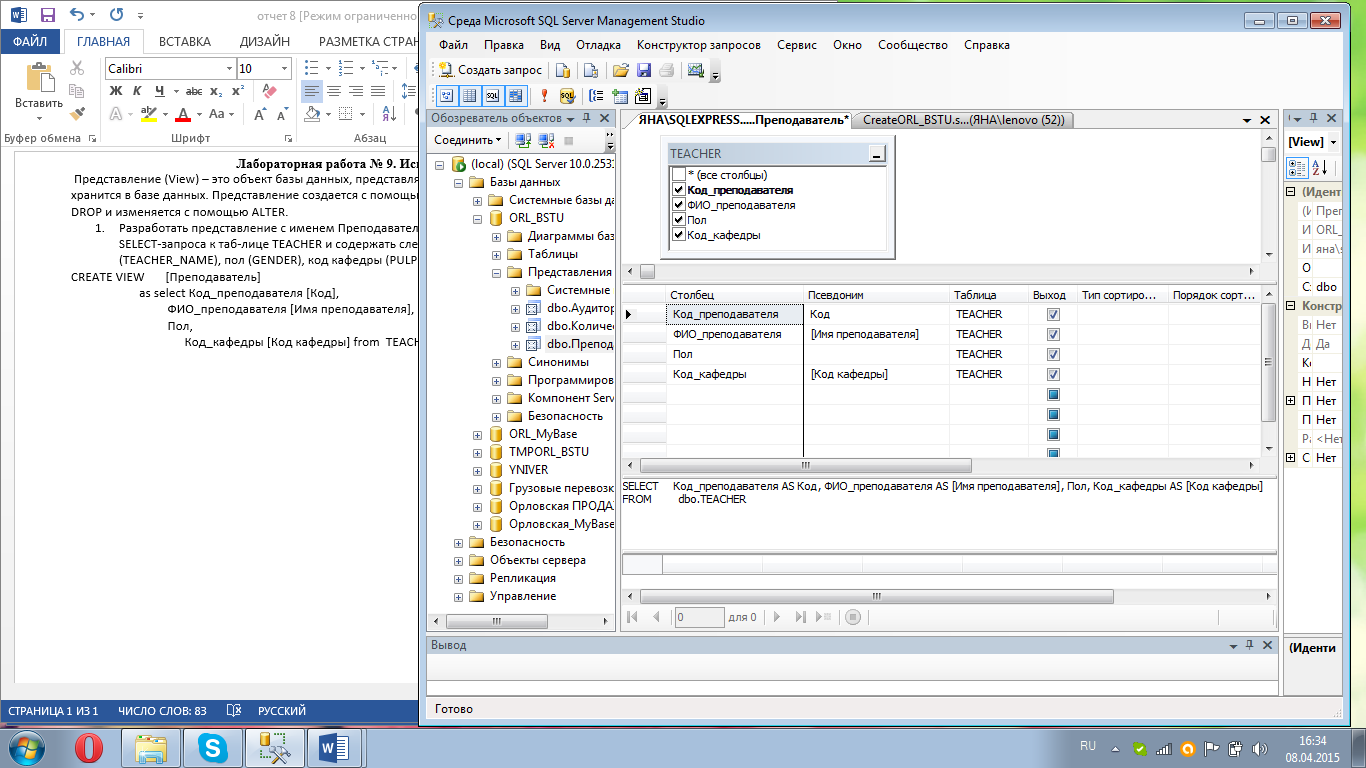
FROM Водители в

WHERE в.Стаж IN ('8')

GROUP BY в.Ф\_И\_О, в.Код\_водителя\_ФИО;

**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).

CREATE VIEW [Преподаватель]

as select Код\_преподавателя [Код],

ФИО\_преподавателя [Имя преподавателя], Пол,

Код\_кафедры [Код кафедры] from TEACHER;

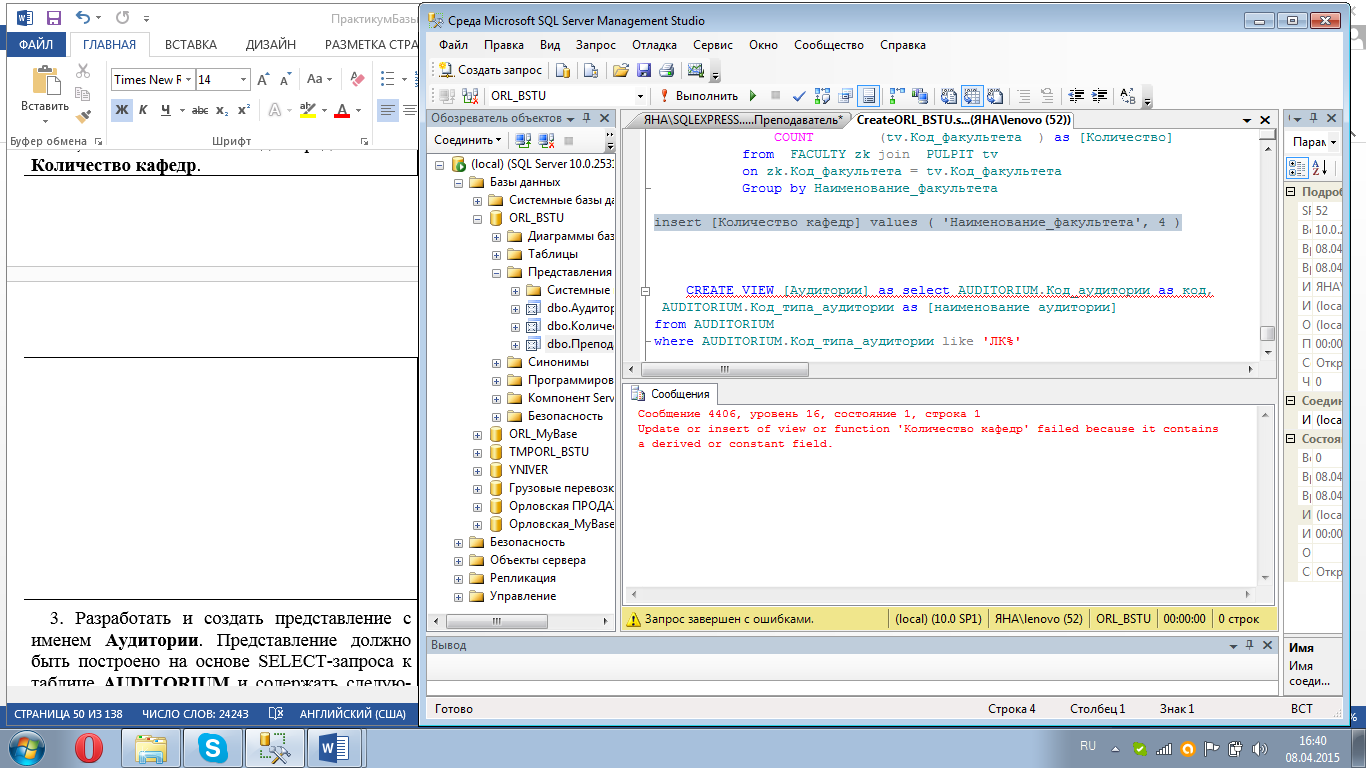
1. Разработать и создать представление с именем Количество кафедр, к таблицам FACULTY и PULPIT. Представление должно содержать столбцы: факультет (FACULTY.FACULTY\_ NAME), количество кафедр (вычисляется на основе строк таблицы PULPIT). Объяснить невозможность выполнения операторов INSERT, UPDATE и DELETE для представления Количество кафедр.

CREATE VIEW [Количество кафедр] as SELECT zk.Наименование\_факультета [Факультет],

COUNT (tv.Код\_факультета) as [Количество]

from FACULTY zk join PULPIT tv on zk.Код\_факультета = tv.Код\_факультета

Group by Наименование\_факультета;

** insert [Количество кафедр] values ( 'Наименование\_факультета', 4 )**

При создании представлений, позволяющих выполнять операции INSERT, DELETE и UPDATE, базовый SELECT-запрос должен удовлетворять правилам:

 запрос не должен содержать секцию группировки GROUP BY;

 запрос не должен применять агрегатные функции, опции DISTINCT и TOP, операторы UNION, INTERSECT и EXCEPT;

 в SELECT-списке запроса не должно быть вычисляемых значений;

 в секции FROM запроса должна указываться только одна таблица.

В приведенном выше примере представление содержит группировку GROUP BY и в секции FROM две таблицы, поэтому и происходит ошибка.

1. Разработать и создать представление с именем Аудитории, к таблице AUDITORIUM и содержать следующие столбцы: код (AUDITORIUM), наименование аудитории (AUDITORIUM\_NAME). Представление должно отоб-ражать только лекционные аудитории (в столбце AUDITORIUM\_ TYPE строка, начинающаяся с символа ЛК). Представление должно допускать выполнение оператора INSERT, UPDATE и DELETE.

CREATE VIEW [Аудитории] as

select AUDITORIUM.Код\_аудитории as код, AUDITORIUM.Код\_типа\_аудитории as [наименование аудитории]

from AUDITORIUM where AUDITORIUM.Код\_типа\_аудитории like 'ЛК%';

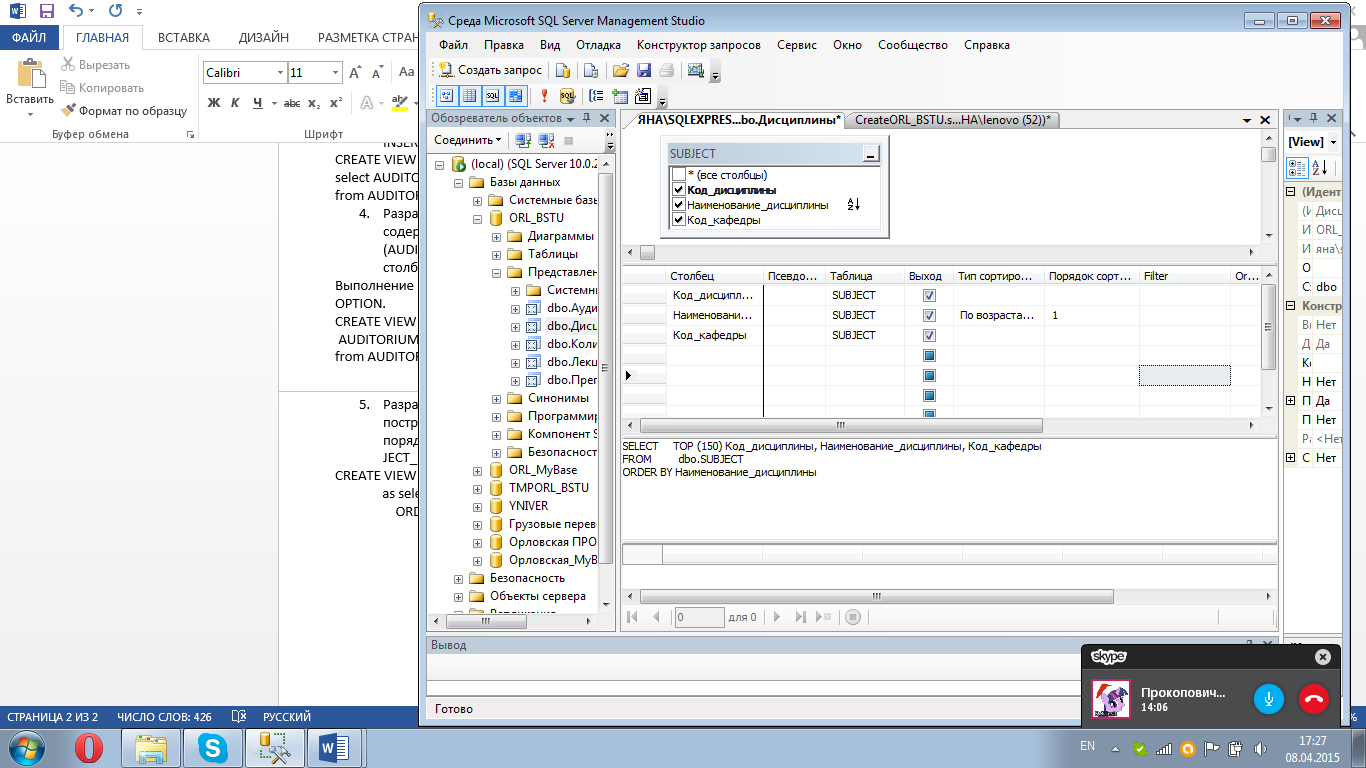
1. Разработать и создать представление с именем Лекционные\_аудитории, к таблице AUDITORIUM и содержать следующие столбцы: код (AUDITORIUM), наименование аудитории (AUDITORIUM\_NAME). Представление должно отображать только лекционные аудитории (в столбце AUDITORIUM\_TYPE строка, начинающаяся с символов ЛК).

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

CREATE VIEW [Лекционные аудитории] as select AUDITORIUM.Код\_аудитории as код,

AUDITORIUM.Код\_типа\_аудитории as [наименование аудитории]

from AUDITORIUM where AUDITORIUM.Код\_типа\_аудитории like 'ЛК%' WITH CHECK OPTION;

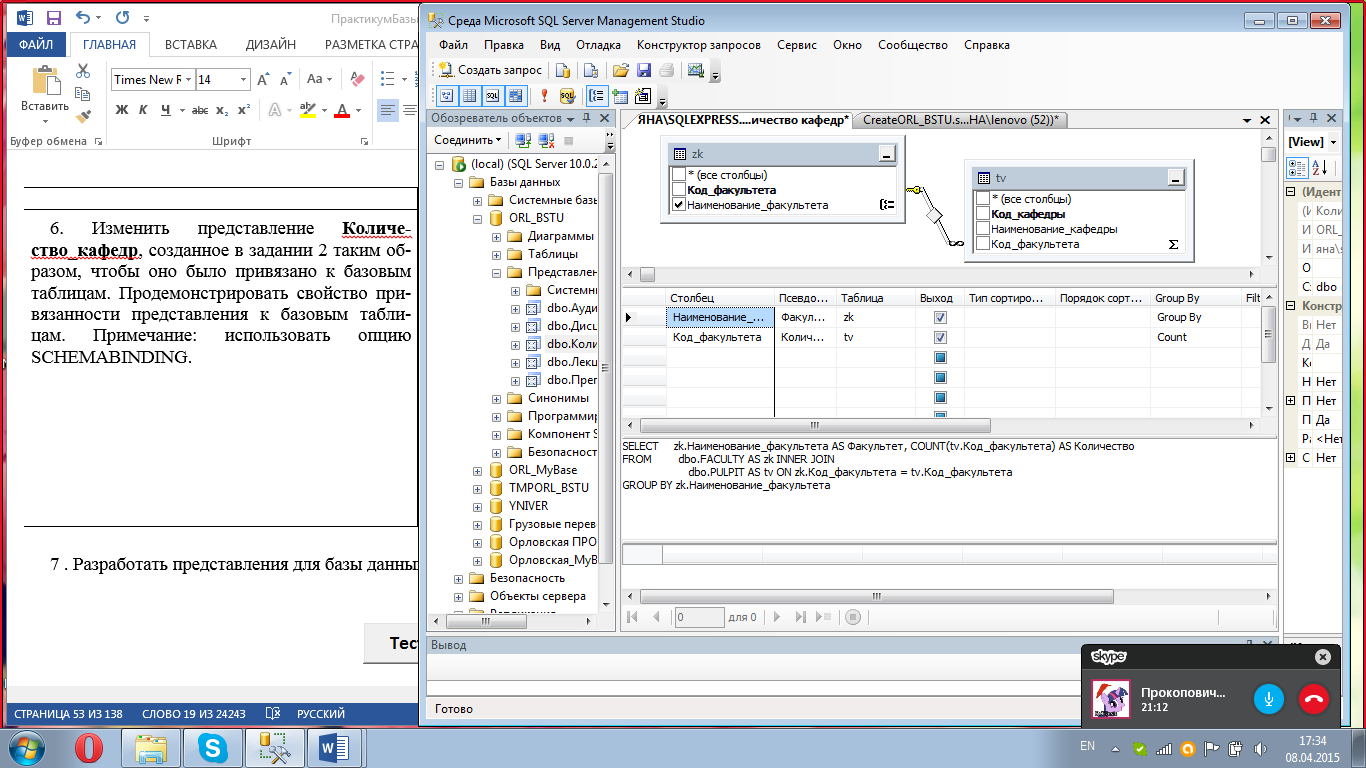
1. Разработать и создать представление с именем Дисциплины, к таблице SUBJECT, отображать все дисци-плины в алфавитном порядке и содержать следующие столбцы: код (SUBJECT), наименование дисциплины (SUB-JECT\_NAME) и код кафедры (PULPIT). Примечание: использовать секции TOP и ORDER BY.

CREATE VIEW Дисциплины

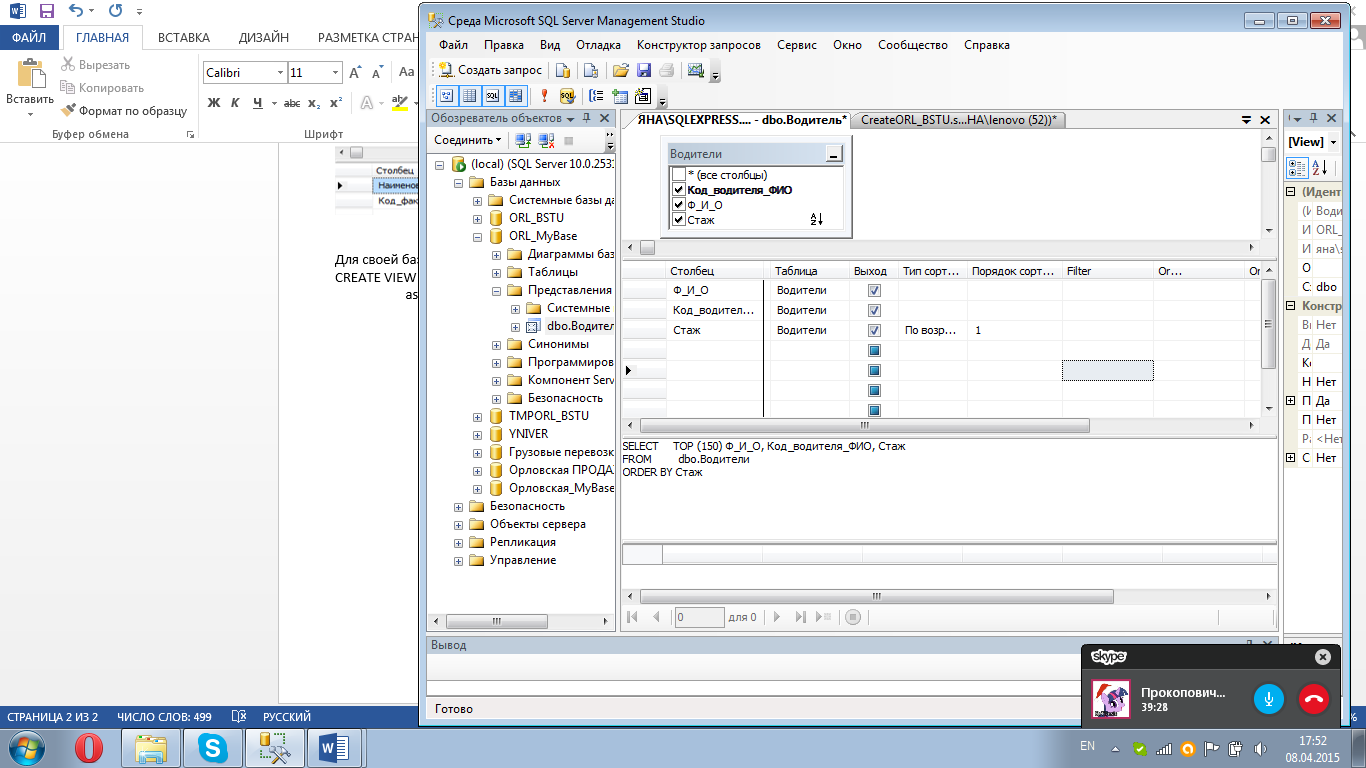
as select TOP 150 Код\_дисциплины, Наименование\_дисциплины, Код\_кафедры from SUBJECT

ORDER BY Наименование\_дисциплины;

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

Опция SCHEMABINDING устанавливает запрещение на операции с таблицами и представлениями, которые могут привести к нарушению ра-ботоспособности представления.

При использовании опции SCHEMABINDING требуется использовать в SELECT-запросе для имен таблиц и представлений двухкомпонентный формат (в имени присутствует точка).

**Для своей базы:**

CREATE VIEW [Водитель]

as select Код\_водителя\_ФИО [Код],

Ф\_И\_О [Имя водителя],

Стаж from Водители;

Немного модифицируем ( по возрастанию):

Alter VIEW Водитель

as select TOP 150 Ф\_И\_О, Код\_водителя\_ФИО, Стаж from Водители

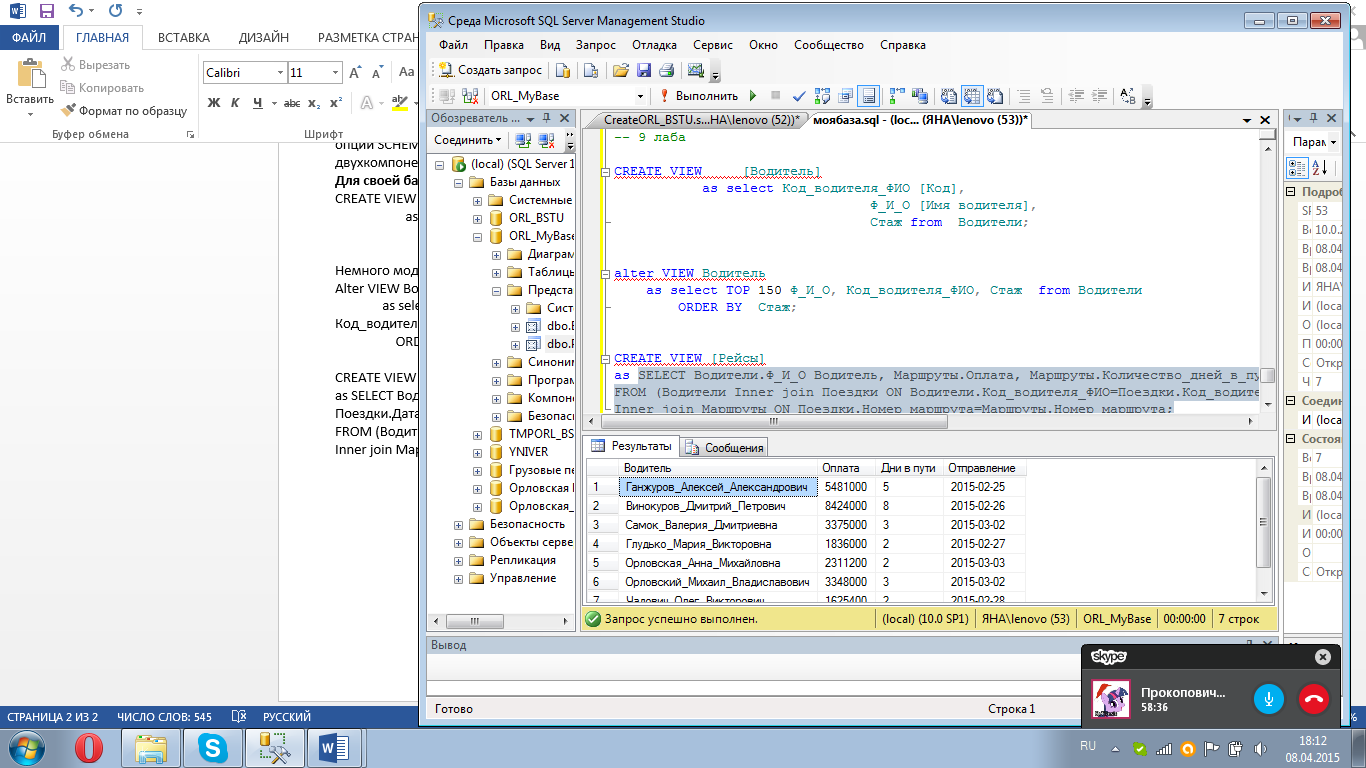
ORDER BY Стаж;

CREATE VIEW [Рейсы]

as SELECT Водители.Ф\_И\_О Водитель, Маршруты.Оплата, Маршруты.Количество\_дней\_в\_пути [Дни в пути], Поездки.Дата\_отправления [Отправление]

FROM (Водители Inner join Поездки ON Водители.Код\_водителя\_ФИО=Поездки.Код\_водителя\_ФИО)

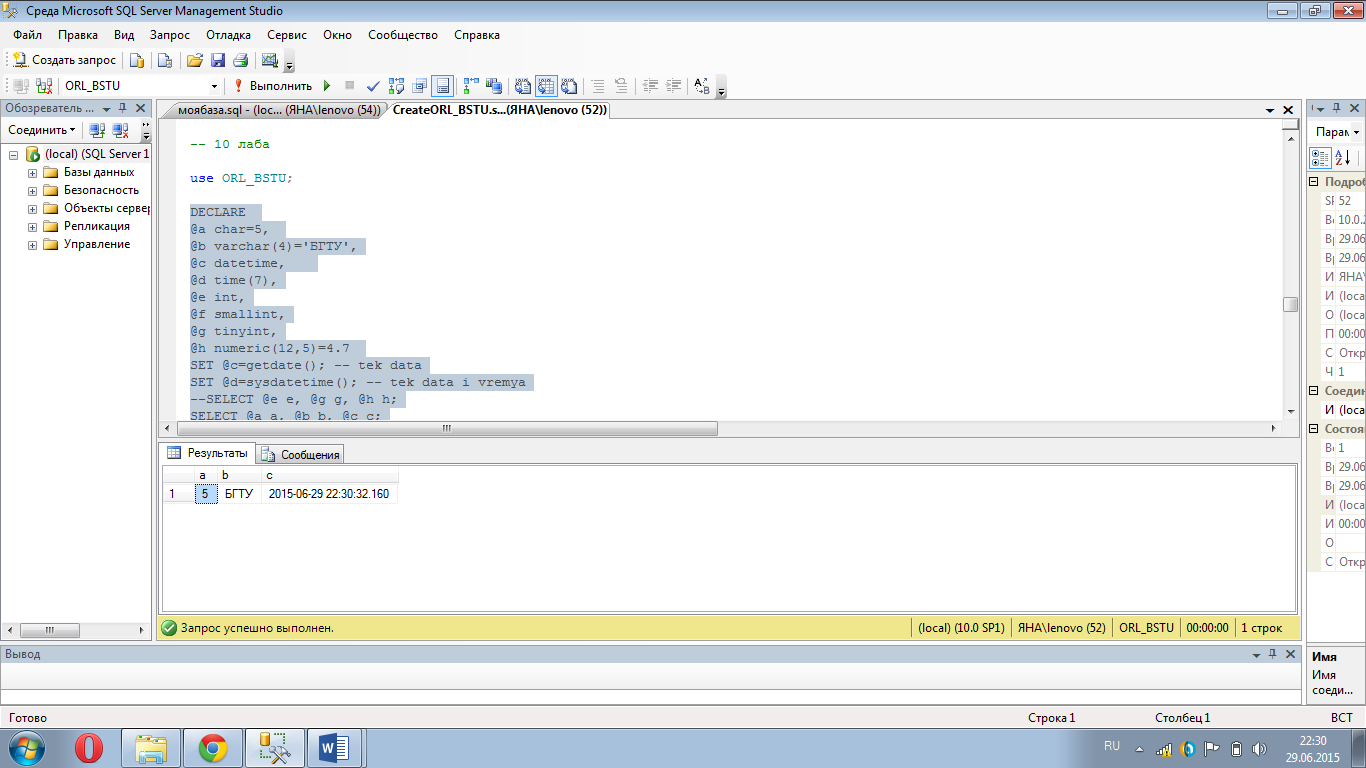
Inner join Маршруты ON Поездки.Номер\_маршрута=Маршруты.Номер\_маршрута;



**Лабораторная работа № 10. Основы программирования на T-SQL**

Для объявления переменных, используемых в программах, предназначен оператор DECLARE. Имя переменной должно начинаться с символа @. С помощью оператора SET можно переменной присвоить значение и выполнять вычисления. Оператор SELECT позволяет нескольким переменным присвоить значения. Функция CAST используется для преобразования типов.

DECLARE

@a char=5,

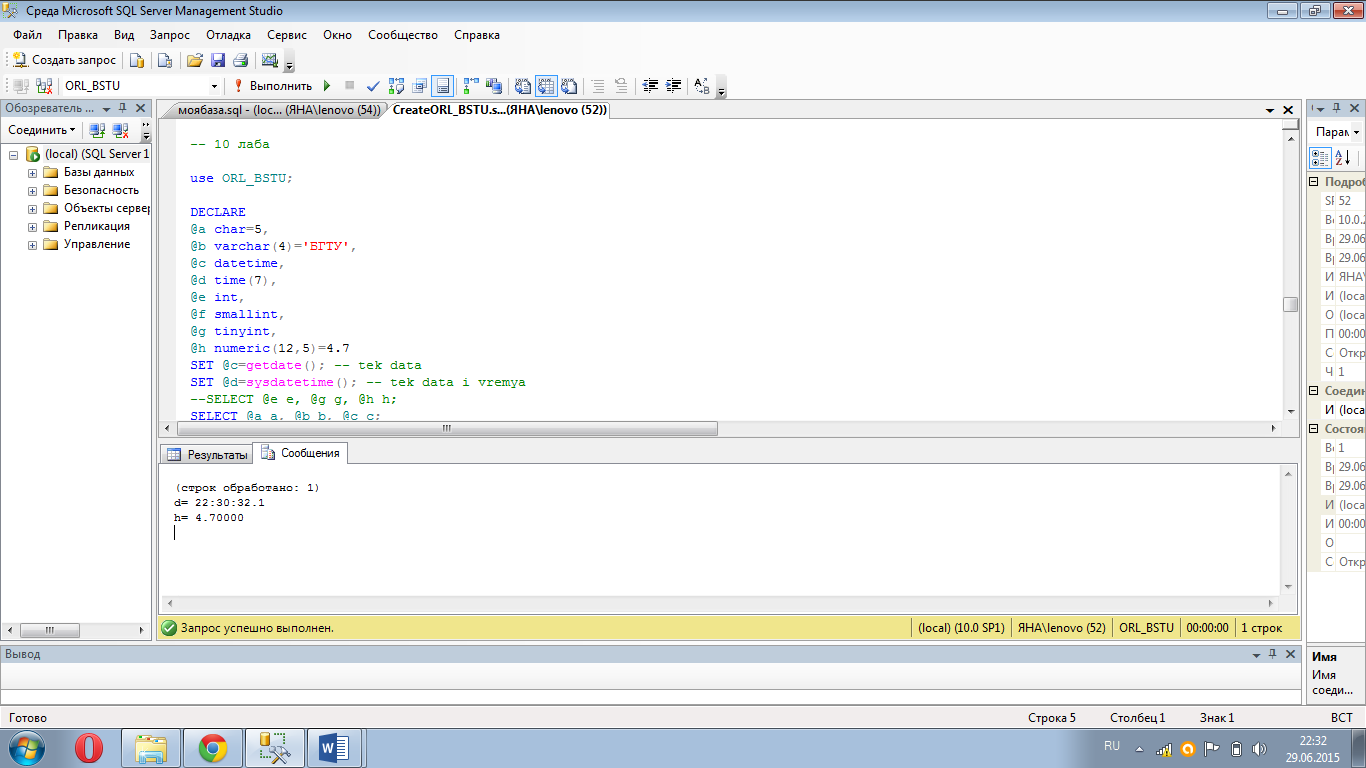
@b varchar(4)='БГТУ',

@c datetime,

@d time(7),

@e int,

@f smallint,

@g tinyint,

@h numeric(12,5)=4.7

SET @c=getdate(); -- tek data

SET @d=sysdatetime(); -- tek data i vremya

--SELECT @e e, @g g, @h h;

SELECT @a a, @b b, @c c;

print 'd= ' +cast(@d as varchar(10));

print 'h= ' +cast(@h as varchar(10));

DECLARE @a1 numeric(8,3)=(select CAST(sum(Вместимость)

as numeric(8,3))from AUDITORIUM),

@a2 real, @a3 numeric(8,3), @a4 real, @a5 numeric(8,3)

If @a1>200

begin

select @a2=(select CAST(count(\*) as numeric(8,3)) from AUDITORIUM),

@a3=(select CAST(AVG(Вместимость)as numeric(8,3)) from AUDITORIUM);

SET @a4=(select cast(count(\*) as numeric(8,3)) from AUDITORIUM where Вместимость<@a3)

select @a5=(select cast(count(\*) as numeric(8,3))/@a2\*100 from AUDITORIUM where Вместимость<@a3)

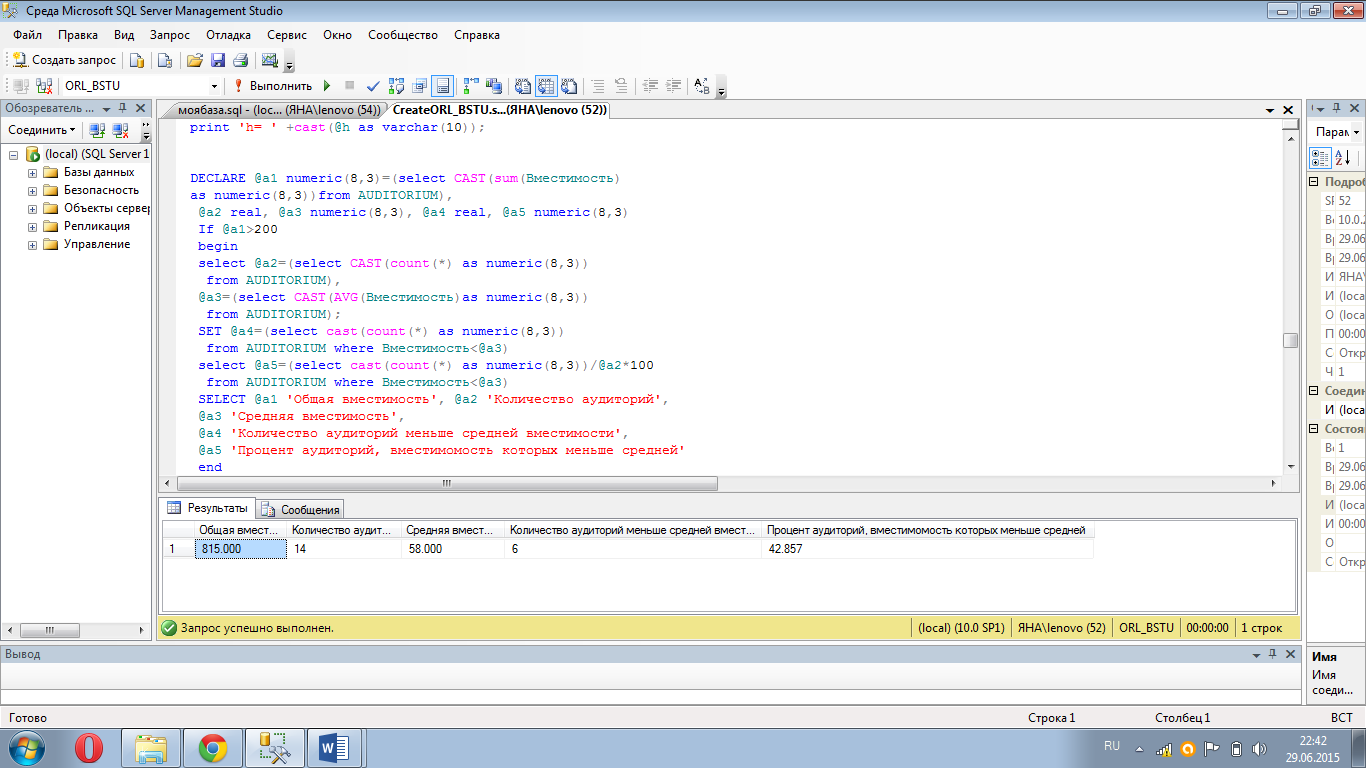
SELECT @a1 'Общая вместимость', @a2 'Количество аудиторий',

@a3 'Средняя вместимость',

@a4 'Количество аудиторий меньше средней вместимости',

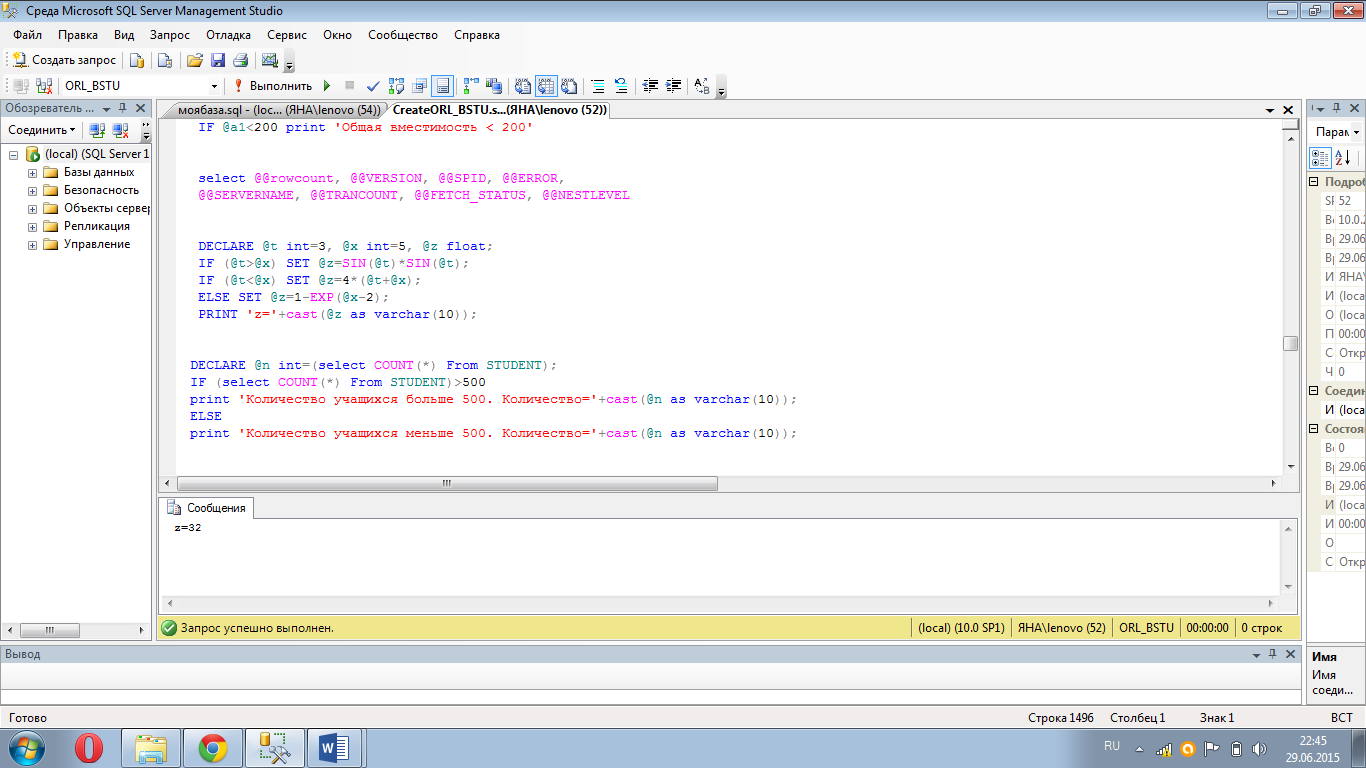
@a5 'Процент аудиторий, вместимомость которых меньше средней'

End IF @a1<200 print 'Общая вместимость < 200'



Вычисление функции:

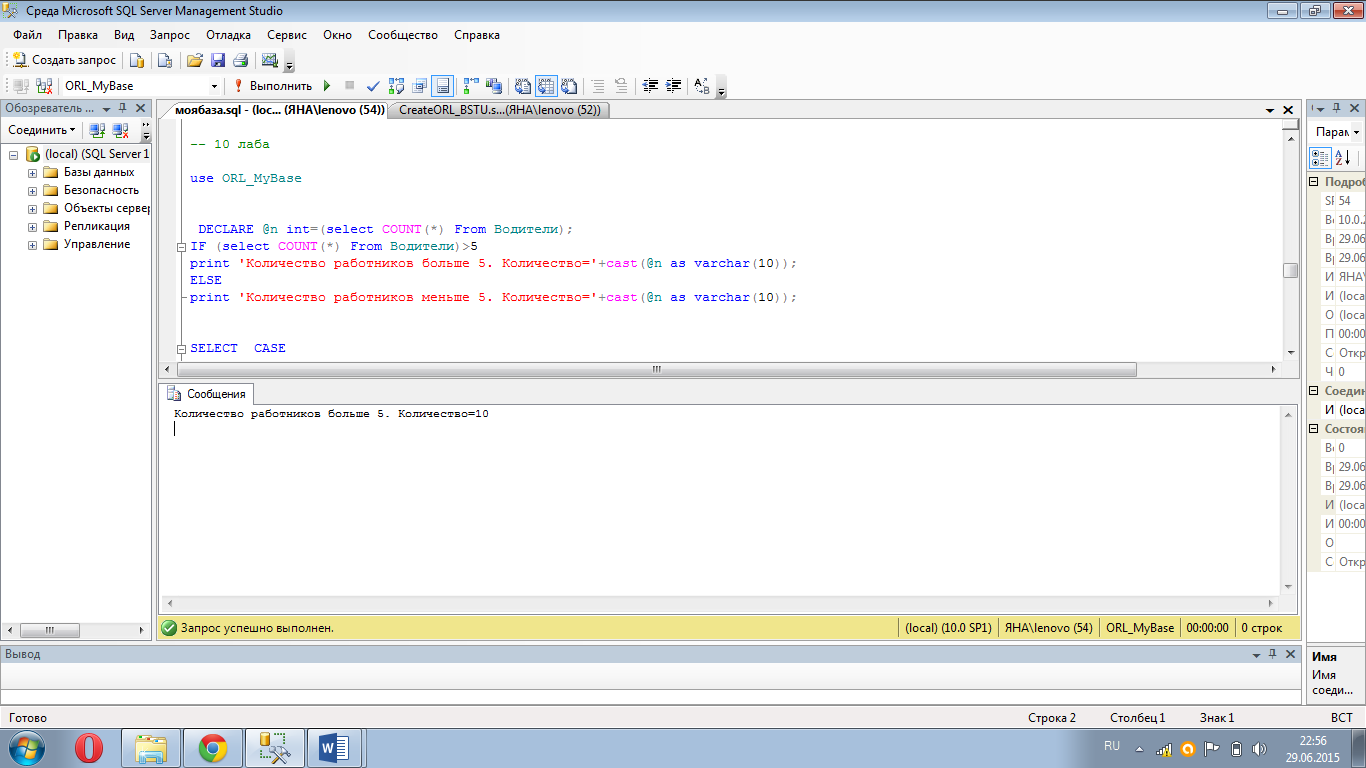
DECLARE @t int=3, @x int=5, @z float;

 IF (@t>@x) SET @z=SIN(@t)\*SIN(@t);

IF (@t<@x) SET @z=4\*(@t+@x);

ELSE SET @z=1-EXP(@x-2);

PRINT 'z='+cast(@z as varchar(10));

Для своей базы. Подсчет работников:

DECLARE @n int=(select COUNT(\*) From Водители);

IF (select COUNT(\*) From Водители)>5

print 'Количество работников больше 5. Количество='+cast(@n as varchar(10));

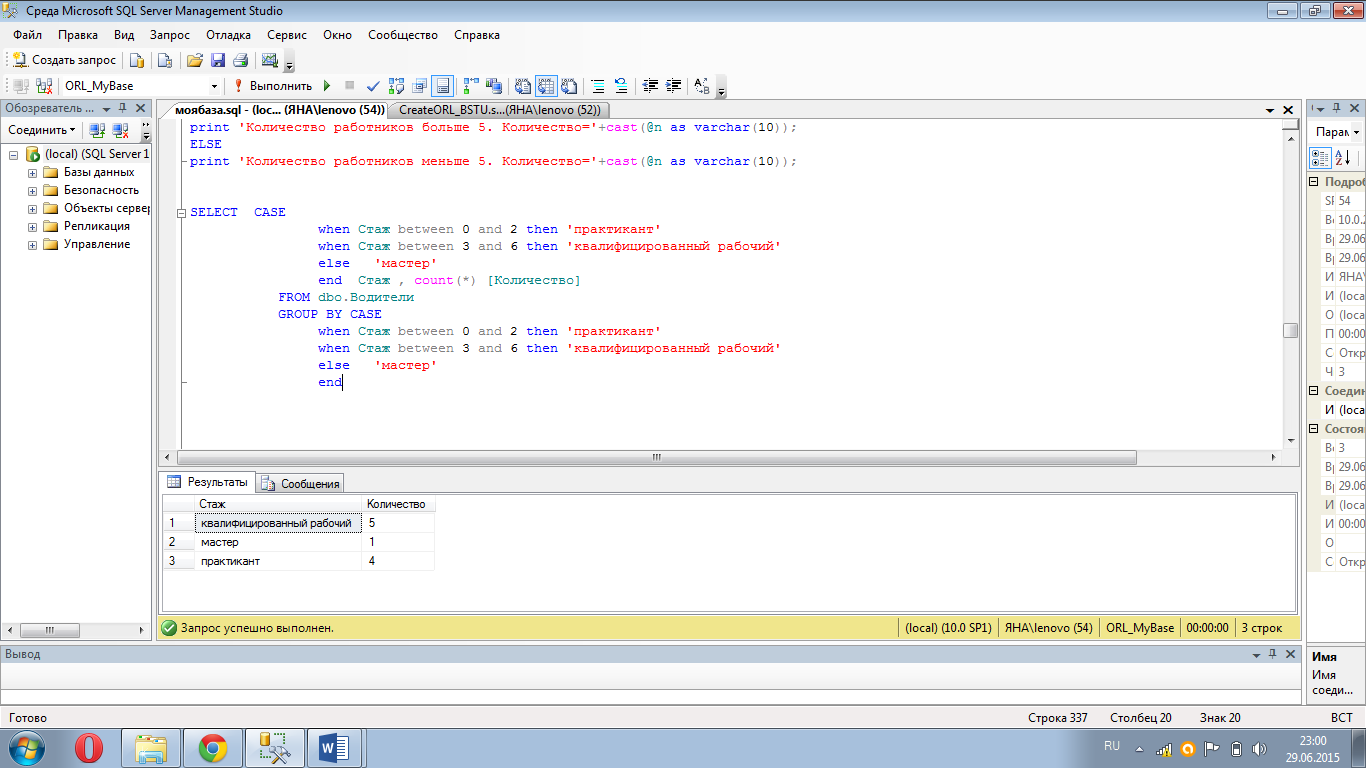
ELSE

print 'Количество работников меньше 5. Количество='+cast(@n as varchar(10));

И пределение их квалификации:

SELECT CASE

when Стаж between 0 and 2 then 'практикант'

 when Стаж between 3 and 6 then 'квалифицированный рабочий'

else 'мастер'

end Стаж , count(\*) [Количество]

FROM dbo.Водители

GROUP BY CASE

when Стаж between 0 and 2 then 'практикант'

when Стаж between 3 and 6 then 'квалифицированный рабочий'

else 'мастер'

end

**Лабораторная работа № 11. Создание и применение индексов**

Индекс – это объект базы данных, позволяющий ускорить поиск в определенной таблице. Как и любой другой объект базы данных, индекс может быть создан с помощью оператора CREATE, модифицирован с помощью ALTER и удален с помощью оператора DROP. Для одной таблицы возможно построение нескольких индексов. Индексы бывают кластеризованные, некластеризованные, уникальные, неуникальные и др.

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

CREATE table #EXPLRE

( TIND int, TFIELD varchar(100) );

go

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

declare @i int=0;

while @i<1000

begin

insert #EXPLRE(TIND, TFIELD)

values (FLOOR(5000\*RAND()),REPLICATE('stroka',10));

if (@i%1=0) print @i;

set @i=@i+1;

end;

go

select COUNT(\*) [kolichestvo strok] from #EXPLRE;

CREATE clustered index #EXPLRE\_CL on #EXPLRE(TIND asc) – кластеризованный

CREATE index #EX\_NONCLU on #EXPLR(TIND, CC) -- составной неуникальный, некластеризованный

CREATE index #EX\_TKEY on #EX(TKEY) INCLUDE (CC); -- некластеризованный покрытия

CREATE index #EXP\_WHERE on #EXP(TK) where (TK>=5000 and TK<15000); -- некластеризованный фильтруемый

Для своей:

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

exec SP\_HELPINDEX'Водители';

SELECT

Водители.Ф\_И\_О, Водители.Стаж, Маршруты.Оплата, Поездки.Дата\_отправления

FROM (Водители Inner join Поездки ON Водители.Код\_водителя\_ФИО=Поездки.Код\_водителя\_ФИО)

Inner join Маршруты ON Поездки.Номер\_маршрута=Маршруты.Номер\_маршрута

ORDER BY

(Case when (Водители.Стаж=7) then 1

when (Водители.Стаж=5) then 2

else 3

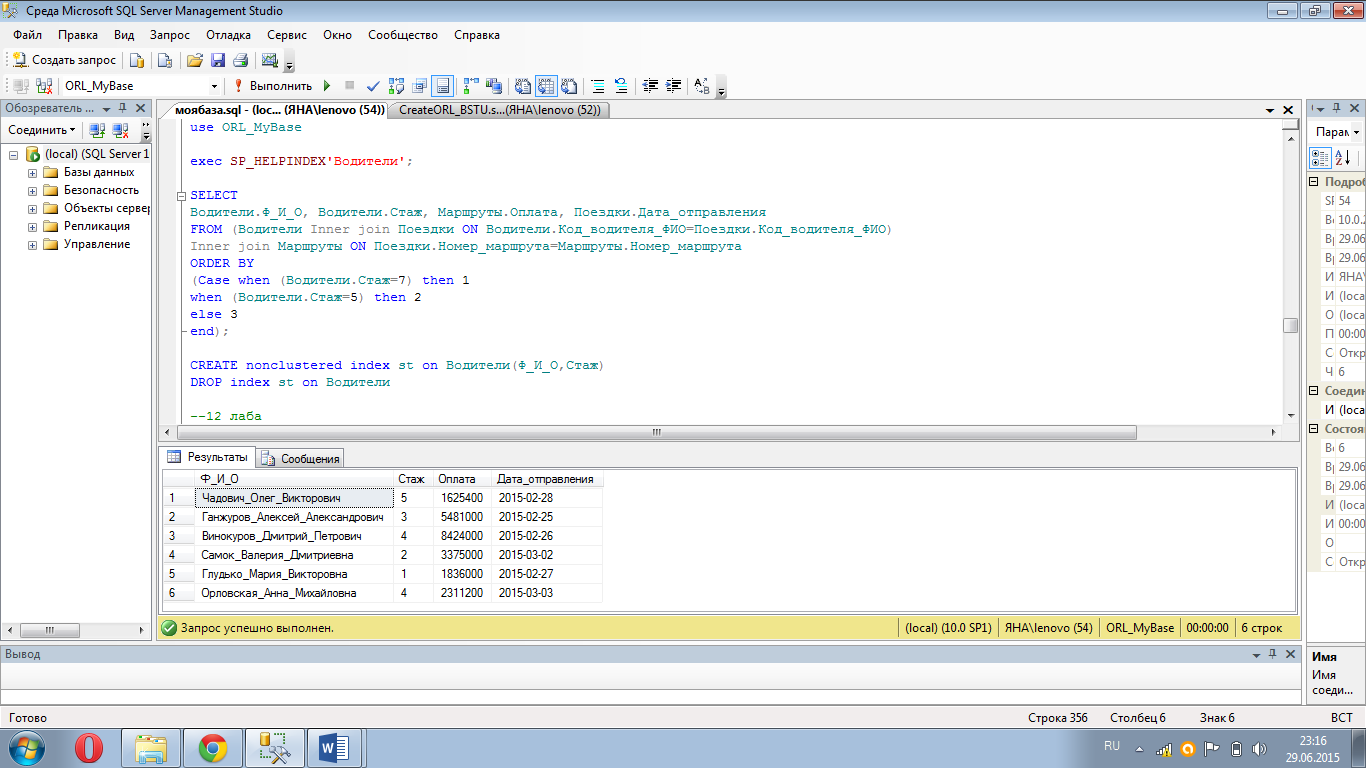
end);

Создадим некластеризованный индекс:

CREATE nonclustered index st on Водители(Ф\_И\_О,Стаж)

Удалим индекс:

DROP index st on Водители



**Лабораторная работа № 12. Обработка результатов запросов с помощью курсоров**

Курсор является программной конструкцией, которая служит для хранения результата запроса и дает возможность пользователю обрабатывать строки результирующего набора запись за записью. Курсоры бывают локальные и гло-бальные (по умолчанию), статические и динамические (по умолчанию). Работа с курсором осуществляется в следующей последовательности: курсор объявляется в операторе DECLARE, открывается с помощью оператора OPEN, с помощью оператора FETCH считывается одна или несколько строк результирующего набора. Результат каждого считывания про-веряется с помощью системной функции @@FETCH\_STATUS. Затем курсор закрывается с помощью оператора CLOSE. Если курсор глобальный, то он должен быть освобожден с помощью оператора DEALLOCATE.

Сценарий, формирующий список дис-циплин на кафедре ИСиТ:

DECLARE @kaf char(20), @k char(300)=' ';

DECLARE dis CURSOR local for select Код\_дисциплины from SUBJECT

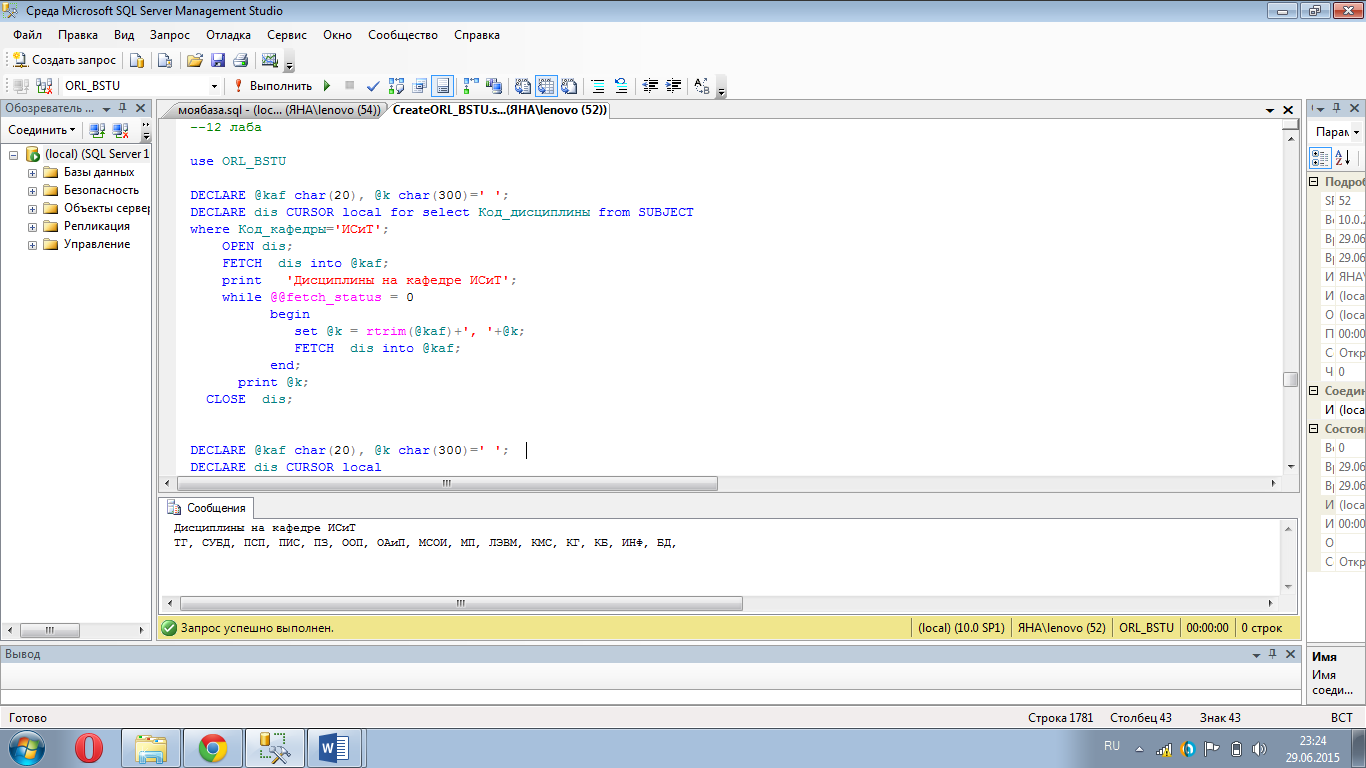
where Код\_кафедры='ИСиТ';

OPEN dis;

FETCH dis into @kaf;

print 'Дисциплины на кафедре ИСиТ';

while @@fetch\_status = 0

 begin

set @k = rtrim(@kaf)+', '+@k;

FETCH dis into @kaf;

end;

print @k;

CLOSE dis;

DECLARE di CURSOR global

declare c\_teacher cursor local static

declare c\_teacher cursor local dynamic

DECLARE @kaf char(20), @k char(300)=' ';

DECLARE dis CURSOR local for select Дата\_отправления from Поездки

OPEN dis;

FETCH dis into @kaf;

print 'Рейсы';

while @@fetch\_status = 0

begin

set @k = rtrim(@kaf)+', '+@k;

FETCH dis into @kaf;

end;

print @k;

CLOSE dis;



**Лабораторная работа № 13. Особенности использования транзакций**

Транзакция - это механизм базы данных, позволяющий таким образом объединять несколько операторов, изменя-ющих базу данных, чтобы при выполнении этой совокупности операторов они или все выполнились или все не выполни-лись.

Основные свойства транзакции: атомарность (операторы изменения БД, включенные в транзакцию, либо выполнят-ся все, либо не выполнится ни один); согласованность (транзакция должна фиксировать новое согласованное состояние БД); изолированность (отсутствие взаимного влияния параллельных транзакций на результаты их выполнения); долго-вечность (изменения в БД, выполненные и зафиксированные транзакцией, могут быть отменены только с помощью новой транзакции).

Сценарий, демонстрирующий работу в режиме неявной транзакции:

declare @c int, @flag char = 'c'; -- оператор фиксации commit или отката rollback?

SET IMPLICIT\_TRANSACTIONS ON -- включ. режим неявной транзакции

-- create table X(K int ); -- начало транзакции

insert X values (1),(2),(3);

set @c = (select count(\*) from X);

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

if @flag = 'c' commit; -- завершение транзакции: фиксация

else rollback; -- завершение транзакции: откат

SET IMPLICIT\_TRANSACTIONS OFF -- выключ. режим неявной транзакции

-- действует режим автофиксации

if exists (select \* from SYS.OBJECTS -- таблица X есть?

where OBJECT\_ID= object\_id(N'DBO.X') )

print 'таблица X есть';

else print 'таблицы X нет'

Сценарий, демонстрирующий работу в режиме явной транзакции:

go

begin try

begin transaction -- начало явной транзакции

delete SUBJECT where Код\_дисциплины='ДМ';

insert SUBJECT values ('ТГ', 'Теория графов','ИСиТ' ),

('КБ', 'Комбинаторика','ИСиТ' );

commit transaction; -- фиксация транзакции

end try

begin catch

print 'ошибка: '+ case

when error\_number() = 547 and patindex('%FK\_SUBJECT%', error\_message()) > 0

then 'нет кафедры с таким кодом'

when error\_number() = 2627 and patindex('%PK\_SUBJECT%', error\_message()) > 0

then 'дублирование кода информации'

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

end;

if @@trancount > 0 rollback transaction ; -- откат транзакции

end catch;

declare @point varchar(32); -- макс. длина имени 32

begin try

begin tran -- начало явной транзакции

delete SUBJECT where Код\_дисциплины='ДМ';

set @point = 'p1'; save tran @point; -- контрольная точка p1

insert SUBJECT values ('ТГ', 'Теория графов','ИСиТ' );

set @point = 'p2'; save tran @point; -- контрольная точка p2

insert SUBJECT values ('КБ', 'Комбинаторика','ИСиТ' );

commit tran; -- фиксация транзакции

end try

begin catch

print 'ошибка: '+ case

when error\_number() = 547 and patindex('%FK\_SUBJECT%', error\_message()) > 0

then 'нет кафедры с таким кодом'

when error\_number() = 2627 and patindex('%PK\_SUBJECT%', error\_message()) > 0

then 'дублирование информации'

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

end;

if @@trancount > 0

begin

print 'контрольная точка: '+ @point;

rollback tran @point; -- откат к контрольной точке

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

end;

end catch;

-- A ---

set transaction isolation level READ UNCOMMITTED -- неподтвержденное чтение

begin transaction

-------------------------- t1 ------------------

select @@SPID, 'insert FACULTY' 'результат', \* from FACULTY

where Код\_факультета='ИТ';

select @@SPID, 'update FACULTY' 'результат', \* from PULPIT

where Код\_факультета='ИТ';

commit;

-------------------------- t2 -----------------

--- B --

begin transaction

select @@SPID

insert FACULTY values ('ИТ', 'Информационных Технологий');

update PULPIT set Код\_факультета = 'ИТ'

where Код\_кафедры= 'ИСиТ'

-------------------------- t1 --------------------

-------------------------- t2 --------------------

rollback;

-- A ---

set transaction isolation level READ COMMITTED -- неповторяющееся чтение

begin transaction

select count(\*) from PULPIT

where Код\_факультета = 'ИТ';

-------------------------- t1 ------------------

-------------------------- t2 -----------------

select 'update PULPIT' 'результат', count(\*)

from PULPIT where Код\_факультета = 'ИТ';

commit;

--- B ---

begin transaction

-------------------------- t1 --------------------

update PULPIT set Код\_факультета = 'ИТ'

where Код\_кафедры = 'ИСиТ'

commit;

-- A ---

set transaction isolation level REPEATABLE READ -- фантомное чтение

begin transaction

select Код\_преподавателя from TEACHER where Код\_кафедры = 'ПОиСОИ';

-------------------------- t1 ------------------

-------------------------- t2 -----------------

select case

when Код\_преподавателя = 'ПТР' then 'insert TEACHER' else ' '

end 'результат', Код\_преподавателя from TEACHER where Код\_кафедры = 'ПОиСОИ';

commit;

--- B ---

begin transaction

-------------------------- t1 --------------------

insert TEACHER values ('ПТР', 'Петров Петр Петрович', 'м', 'ПОиСОИ');

commit;

-------------------------- t2 --------------------

Для своей базы:

go

begin try

begin transaction -- начало явной транзакции

delete Поездки where Код\_водителя\_ФИО='ОМВ';

delete Водители where Код\_водителя\_ФИО='ОМВ';

insert Водители values ('ОЯВ', 'Орловская Яна Владимировна',1 ),

('ВАВ', 'Винокурова Анастасия Владимировна',1 );

commit transaction; -- фиксация транзакции

end try

begin catch

print 'ошибка: '+ case

when error\_number() = 547 and patindex('%FK\_Поездки%', error\_message()) > 0

then 'нет такого водителя'

when error\_number() = 2627 and patindex('%PK\_Водители%', error\_message()) > 0

then 'дублирование кода информации'

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

end;

if @@trancount > 0 rollback transaction ; -- откат транзакции

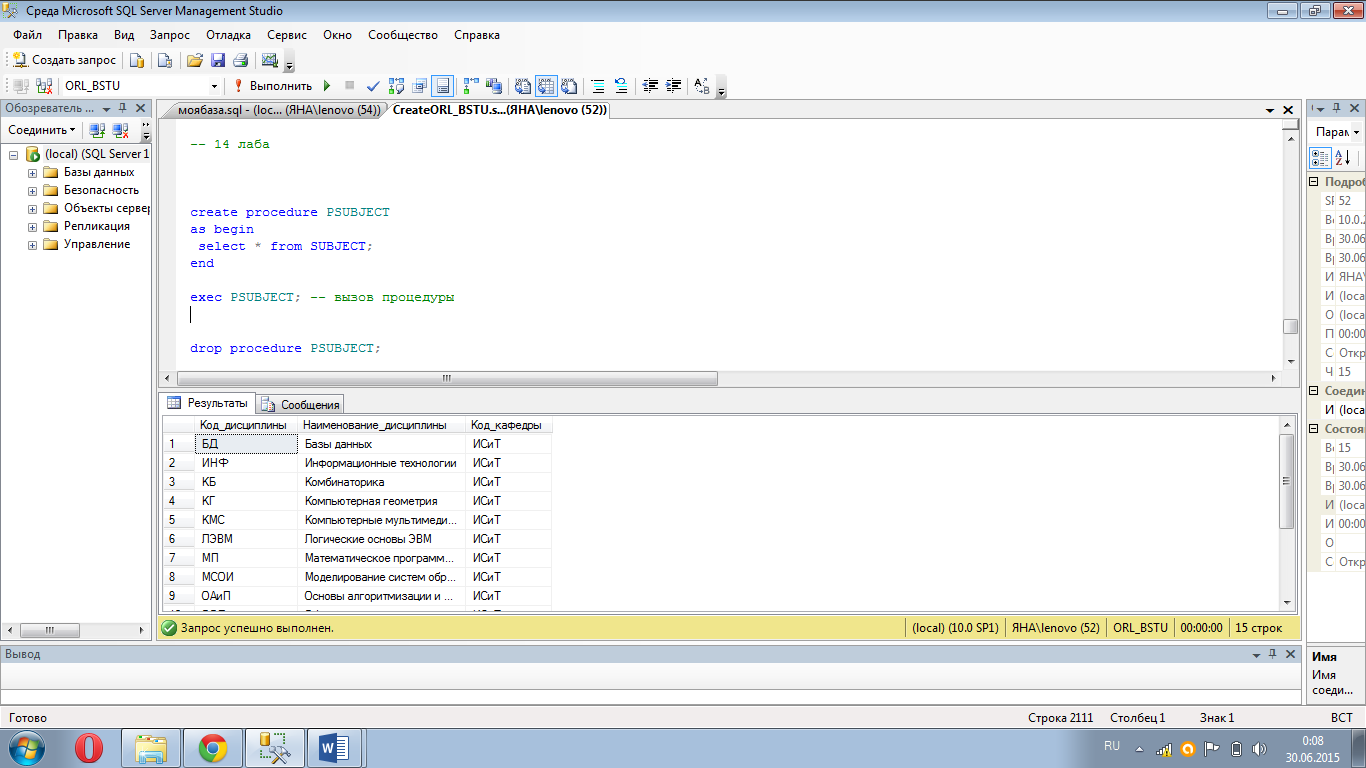
end catch;

**Лабораторная работа № 14. Разработка хранимых процедур**

Хранимая процедура – это объект базы данных, представляющий собой поименованный код T-SQL. Хранимая про-цедура может быть создана с помощью CREATE, изменена с помощью ALTER и удалена с помощью оператора DROP. Процедура может принимать входные и формировать выходные параметры. Результатом ее выполнения может быть це-лочисленное значение, которое возвращается к точке вызова оператором RETURN, либо один или более результирую-щих наборов, сформированных операторами SELECT, либо содержимое стандартного выходного потока, полученного при выполнении операторов PRINT. Вызов процедуры осуществляется оператором EXECUTE (EXEC).

В хранимых процедурах допускается применение основных DDL и всех DML и TCL-операторов, конструкций TRY/CATCH, курсоров, временных таблиц.

Создание процедуры:

create procedure PSUBJECT

as begin

select \* from SUBJECT;

end

exec PSUBJECT; -- вызов процедуры

Удаление процедуры:

drop procedure PSUBJECT;

alter procedure PSUBJECT -- модификация процедуры

@p varchar(20),

@c int output

as begin

declare @rc int = (select COUNT(\*) from SUBJECT);

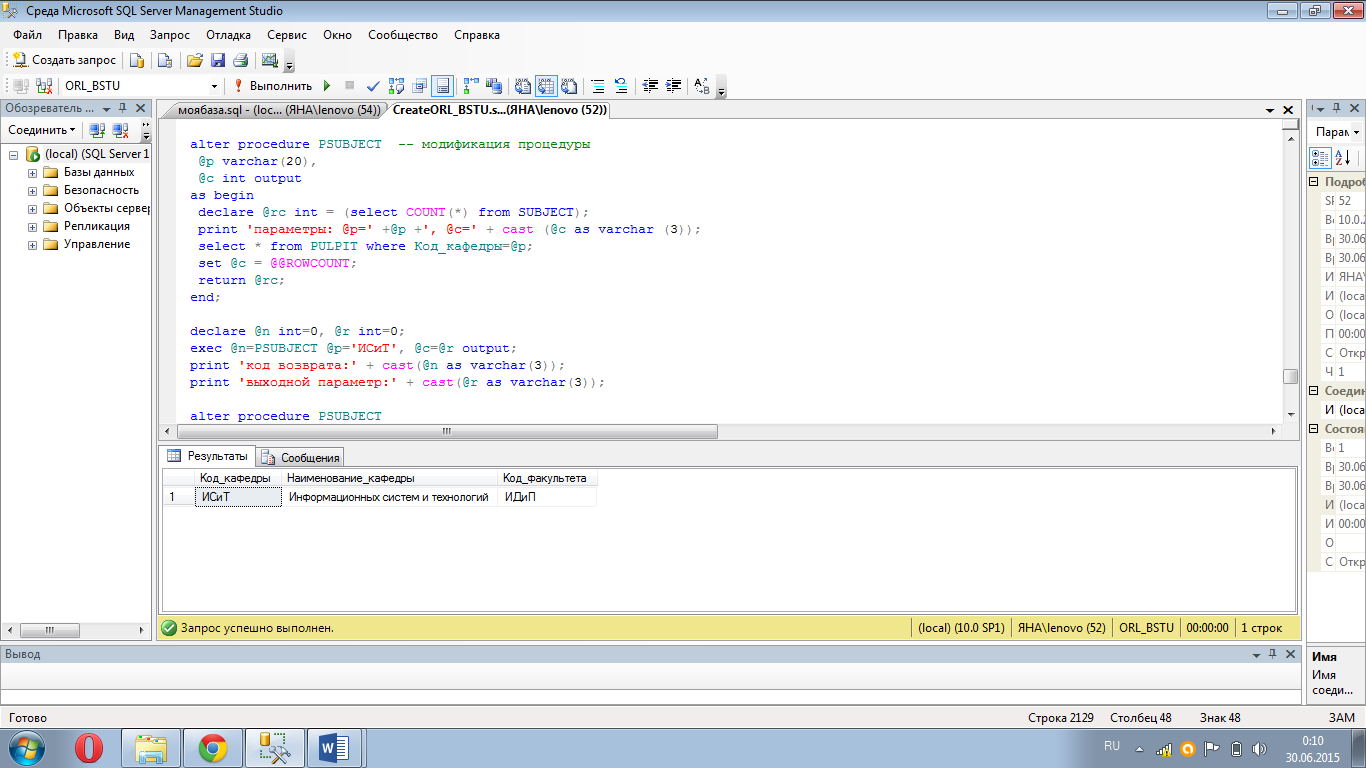
print 'параметры: @p=' +@p +', @c=' + cast (@c as varchar (3));

select \* from PULPIT where Код\_кафедры=@p;

set @c = @@ROWCOUNT;

return @rc;

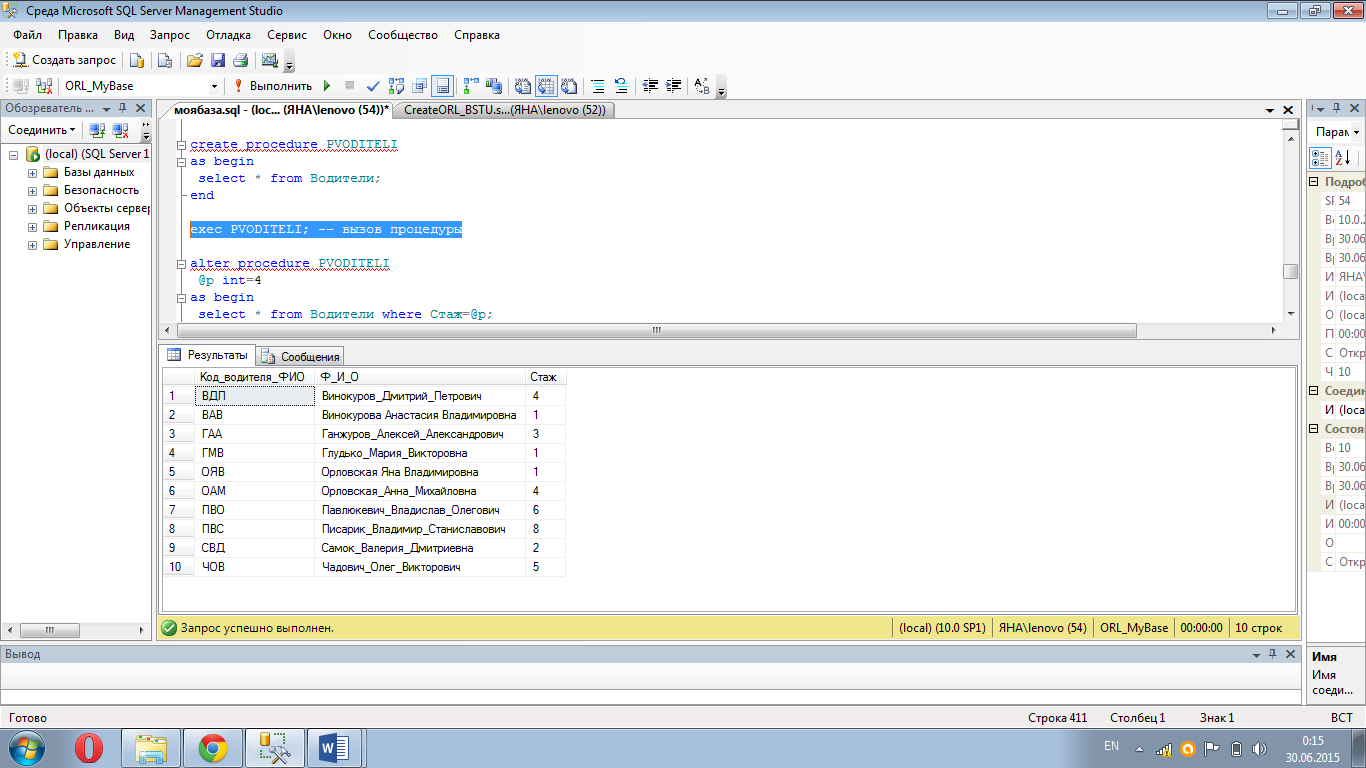
end;

declare @n int=0, @r int=0;

exec @n=PSUBJECT @p='ИСиТ', @c=@r output;

print 'код возврата:' + cast(@n as varchar(3));

print 'выходной параметр:' + cast(@r as varchar(3));

Для своей базы:

create procedure PVODITELI

as begin

select \* from Водители;

end

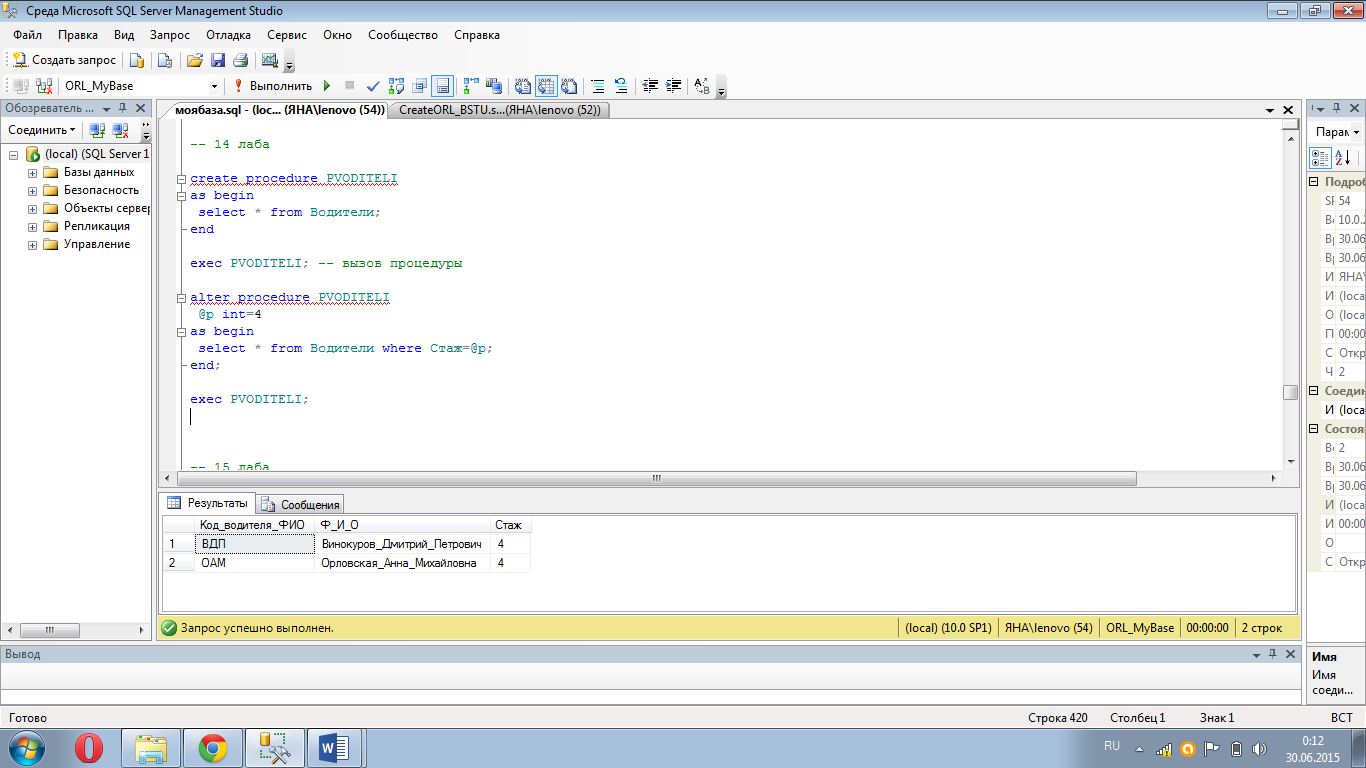
exec PVODITELI; -- вызов процедуры

alter procedure PVODITELI

@p int=4

as begin

select \* from Водители where Стаж=@p;

end;

**Лабораторная работа № 15. Разработка и использование функций**

Функция – это объект БД, представляющий собой поименованный код T-SQL. Для создания, удаления и изменения функций надо использовать операторы CREATE, DROP и ALTER соответственно. Отличие функций от хранимых про-цедур в ограничениях, накладываемых на код функции, в форме представления результата работы, а также в способе вызова. В функции не допускается применение DDL-операторов, DML-операторов, изменяющих БД (INSERT, DELETE, UPDATE), конструкций TRY/CATCH, а также использование транзакций.

Результатом выполнения функции является возвращаемое к точке вызова значение. Если функция возвращает един-ственное значение (число, строка, дата, время и пр.), то она называется скалярной. Функция, возвращающая таблицу, называется табличной. В зависимости от структуры кода, различают встроенные функции и многооператорные таблич-ные функции.

create function COUNT\_STUDENTS(@faculty varchar(20)) returns int -- скалярная функция

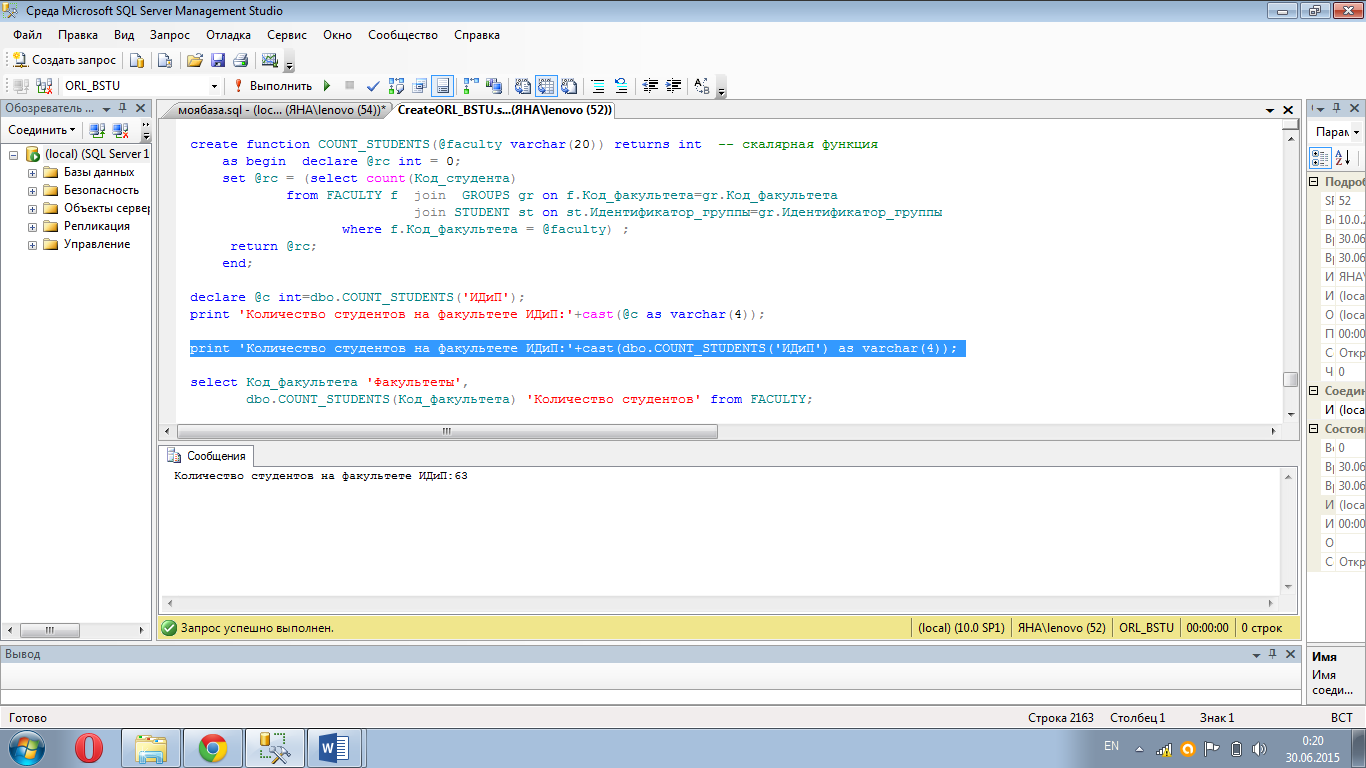
as begin declare @rc int = 0;

set @rc = (select count(Код\_студента)

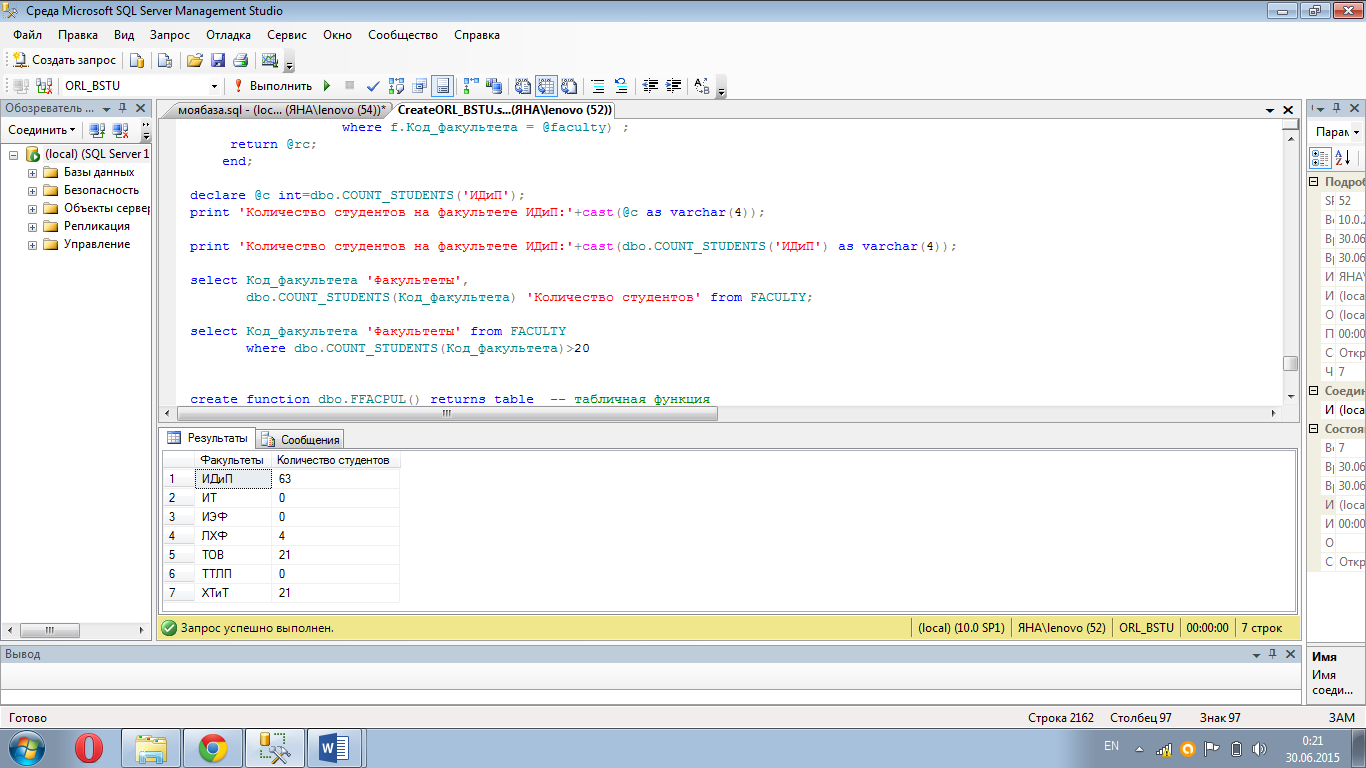
from FACULTY f join GROUPS gr on f.Код\_факультета=gr.Код\_факультета

join STUDENT st on st.Идентификатор\_группы=gr.Идентификатор\_группы

where f.Код\_факультета = @faculty) ;

 return @rc;

end;

declare @c int=dbo.COUNT\_STUDENTS('ИДиП');

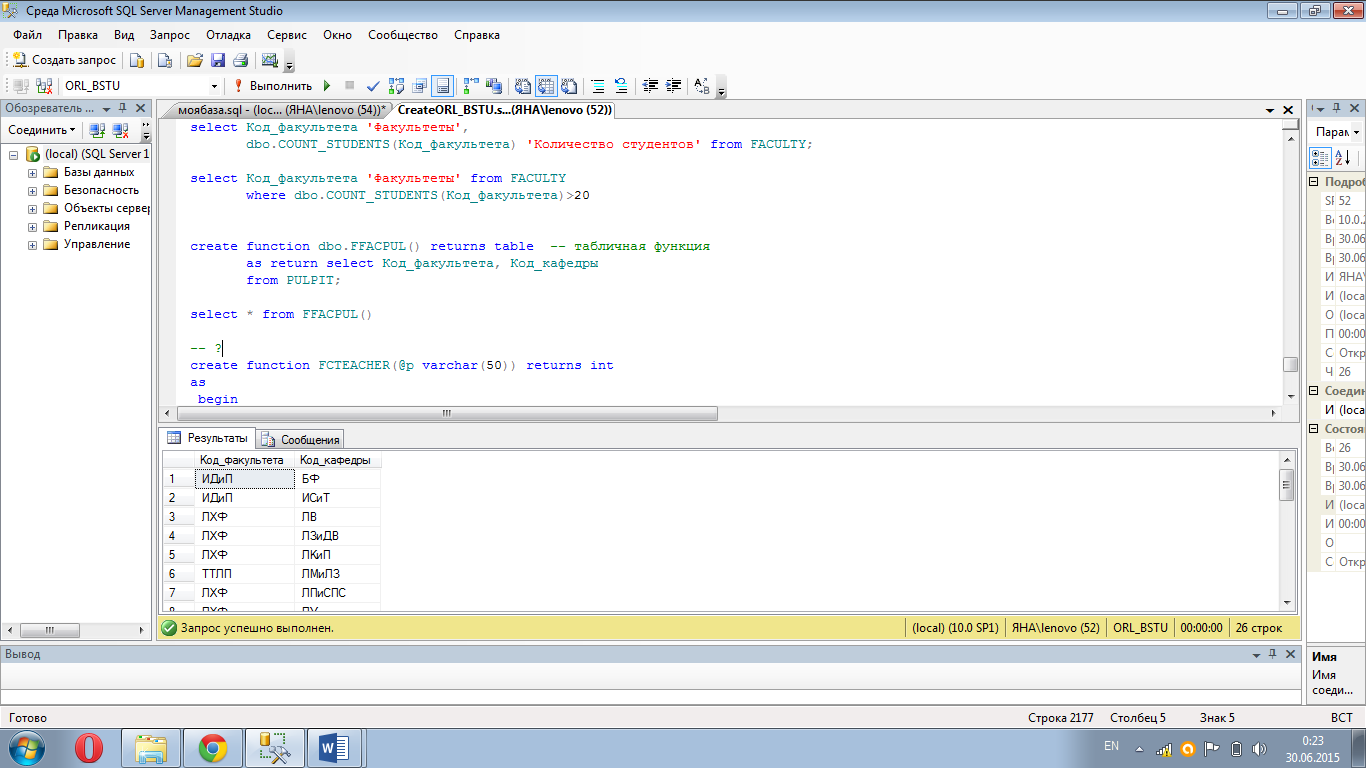
print 'Количество студентов на факультете ИДиП:'+cast(@c as varchar(4));

select Код\_факультета 'Факультеты',

dbo.COUNT\_STUDENTS(Код\_факультета) 'Количество студентов' from FACULTY;

select Код\_факультета 'Факультеты' from FACULTY

where dbo.COUNT\_STUDENTS(Код\_факультета)>20

create function dbo.FFACPUL() returns table -- табличная функция

as return select Код\_факультета, Код\_кафедры

from PULPIT;

select \* from FFACPUL()

Для своей базы:

create function COUNT\_VODITELI(@vod varchar(20)) returns int -- скалярная функция

as begin declare @rc int = 0;

set @rc = (select count(Код\_водителя\_ФИО) from Маршруты m join Поездки p on m.Номер\_маршрута=p.Номер\_маршрута

where m.Номер\_маршрута = @vod) ;

return @rc;

end;

drop function COUNT\_VODITELI

declare @c int=dbo.COUNT\_VODITELI('7');

print 'Количество водителей с номером маршрута 7:'+cast(@c as varchar(4));

или

print 'Количество водителей с номером маршрута 7:'+cast(dbo.COUNT\_VODITELI('7') as varchar(4));



create function dbo.TABLPOIZDKI() returns table -- табличная функция

as return select Номер\_маршрута, Дата\_отправления

from Поездки;

select \* from TABLPOIZDKI()

**Лабораторная работа № 16. Применение DML-триггеров**

Триггер – это особый вид хранимой процедуры, предназначенной для обработки событий в БД. Поддерживается два типа триггеров: DDL-триггеры и DML-триггеры. Для каждого типа определено свое семейство событий, обработку ко-торых триггер этого типа может выполнять.

create table TR\_MARSHRUTS

( ID int identity,

STMT varchar(20) check (STMT in ('INS','DEL','UPD')),

TRNAME varchar(50),

CC varchar(300) )

Триггер реагирующий на вставку:

create trigger TR\_MARSHRUTS\_INS

on Маршруты after insert

as declare @a1 int, @a2 int, @a3 int, @a4 real, @in varchar(300);

print 'Insert operation';

set @a1=(select[Номер\_маршрута] from INSERTED);

set @a2=(select[Дальность] from INSERTED);

set @a3=(select[Количество\_дней\_в\_пути] from INSERTED);

set @a4=(select[Оплата] from INSERTED);

set @in=@a1+''+@a2+''+@a3+''+@a4;

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

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

return;

go

insert into Маршруты(Номер\_маршрута,Дальность,Количество\_дней\_в\_пути,Оплата)

values(11,230,4,3250600);

select \* from TR\_MARSHRUTS

select \* from Маршруты

drop table TR\_MARSHRUTS

**Лабораторная работа № 17. Использование XML**

XML (Extensible Markup Language) – расширяемый язык разметки. XML-формат часто используется для обмена дан-ными между компонентами информационных систем. При работе с базами данных важными являются две задачи: пре-образование табличных данных в XML-структуры и преобразование XML-структур в строки реляционной таблицы.

select \* from Маршруты where Количество\_дней\_в\_пути='5'

select \* from Маршруты where Количество\_дней\_в\_пути='4' for xml raw

select \* from Маршруты where Количество\_дней\_в\_пути='3' for xml auto

select v.Ф\_И\_О, p.Дата\_отправления

from Водители v join Поездки p

on v.Код\_водителя\_ФИО=p.Код\_водителя\_ФИО

where p.Дата\_отправления in ('2015-02-26','2015-02-25') order by p.Дата\_отправления

for xml auto

