**Лабораторная работа № 12. Разработка хранимых процедур**

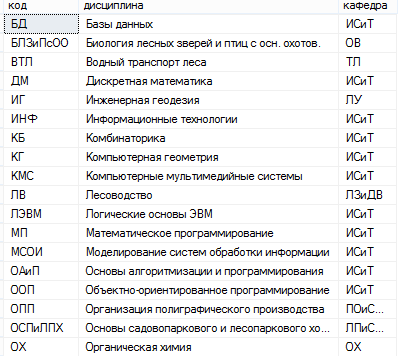
**КУРНОСЕНКО СОФЬЯ**

**Хранимая процедура** – это поименованный код на языке Transact-SQL. Хранимая процедура 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 целочисленное значение, которое возвращается к точке вызова оператором RETURN, либо один или более результирующих 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, DML и TCL-операторов, конструкций TRY/CATCH, курсоров, временных таблиц.

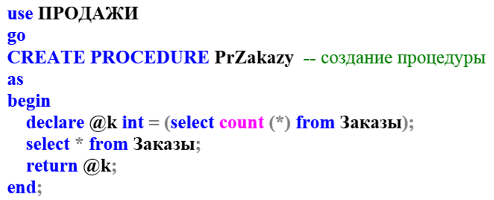
1.Разработать хранимую процедуру без параметров с именем **PSUBJECT**. Процедура формирует результирующий набор на основе таблицы **SUBJECT**, аналогичный набору, представленному на рисунке:

****

К точке вызова процедура должна возвращать количество строк, выведенных в результирующий набор.

***→ Теория (методичка)***

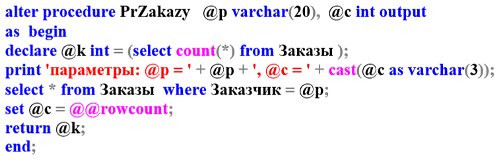
Процедура ниже предназначена для подсчета количества строк в таблице **Заказы** и вывода ее содержимого:



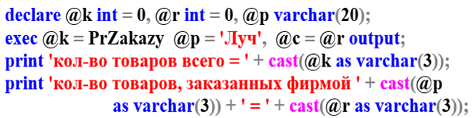
Обращение к процедуре осуществляется оператором EXEC:



С помощью оператора ALTER можно изменить процедуру:



Здесь процедура принимает два параметра: входной параметр @p и выходной (ключевое слово OUTPUT) параметр @c. Процедура формирует значение выходного параметра @c, выводит в стандартный поток сообщение (оператор PRINT), а также формирует результирующий набор. Вызов процедуры:



***→ Выполнение заданий***

-- создание процедуры

go -- команда CREATE PROCEDURE должна вызываться в отдельном пакете

create procedure PSUBJECT

as

begin

select SUBJECT as [код],

SUBJECT\_NAME as [дисциплина],

PULPIT as [кафедра]

from SUBJECT

return @@rowcount; -- число обработанных строк

end;

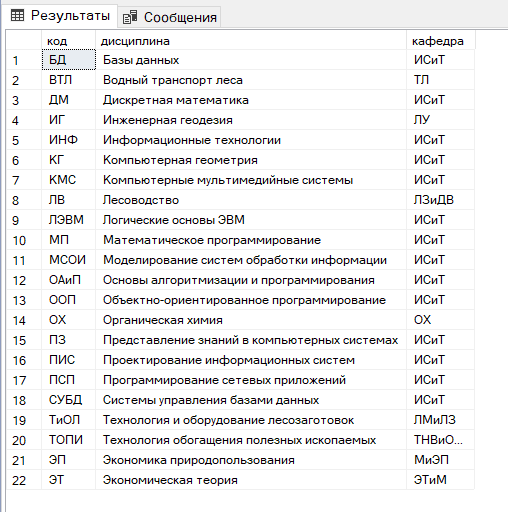
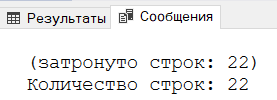
-- выполнение

go

declare @result\_1 int;

exec @result\_1 = PSUBJECT;

print 'Количество строк: ' + convert(varchar(20), (@result\_1));

****** ******

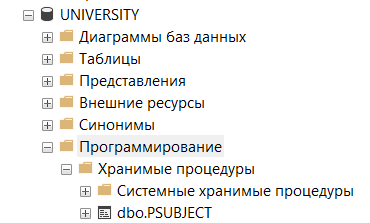
2.Найти процедуру **PSUBJECT** с помощью обозревателя объектов (Object Explorer) SSMS и через контекстное меню создать сценарий на изменение процедуры оператором ALTER.

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**).

***→ Выполнение заданий***

Можно увидеть хранимые процедуры в папке «Программирование» БД:



-- вносим изменения в скрипт процедуры

go

alter procedure [dbo].[PSUBJECT]

@p varchar(20) = null, -- входной параметр, по умолчанию равен null

@c int output -- выходной параметр

as

begin

select SUBJECT as [код],

SUBJECT\_NAME as [дисциплина],

PULPIT as [кафедра]

from SUBJECT

where SUBJECT = @p;

set @c = @@rowcount; -- кол-во обработанных строк (строк в результирующем наборе)

return (select count(\*) from SUBJECT); -- кол-во строк в таблице SUBJECT

end;

-- использование процедуры

go

set nocount on;

declare @result\_2 int,

@p varchar(20),

@outputParam int;

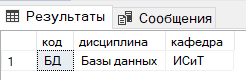
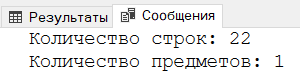
exec @result\_2 = PSUBJECT

@p = 'БД',

@c = @outputParam output;

print 'Количество строк: ' + cast(@result\_2 as varchar);

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

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

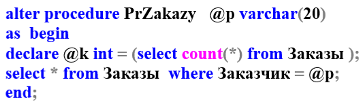
Изменить процедуру **PSUBJECT** таким образом, чтобы она не содержала выходного параметра.

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

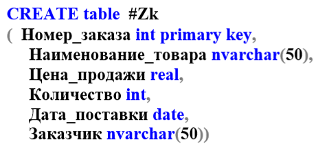
***→ Теория (методичка)***

Процедуры, формирующие результирующий набор и не имеющие выходных параметров, могут быть применены в операторе INSERT в качестве источника строк для добавления в некоторую таблицу.

Пусть с помощью ALTER внесены изменения в процедуру:



Временная таблица может быть создана с помощью операторов:



Операторы INSERT добавляют строки во временную таблицу #Zk:

**INSERT #Zk exec PrZakazy @p = 'Zte';**

**INSERT #Zk exec PrZakazy @p = 'Белвест';**

Просмотреть содержимое временной таблицы можно с помощью оператора SELECT:

**select \* from #Zk**

***→ Выполнение заданий***

-- вносим изменения в скрипт процедуры

go

alter procedure [dbo].[PSUBJECT]

@p varchar(20) = null

as

begin

select SUBJECT as [код],

SUBJECT\_NAME as [дисциплина],

PULPIT as [кафедра]

from SUBJECT

where SUBJECT = @p;

end;

-- создаем временную таблицу

create table #SUBJECT (

код char(10),

дисциплина varchar(100),

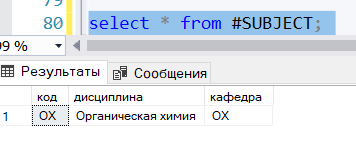
кафедра char(20),

);

-- использование процедуры для добавления строк

insert #SUBJECT exec PSUBJECT 'ох';

select \* from #SUBJECT;

******

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.

***→ Теория (методичка)***

Процедура, представленная ниже, осуществляет вставку строки в таблицу **Товары** и обработку ошибки, если таковая появится:



Обращение к процедуре:

**declare @rc int;**

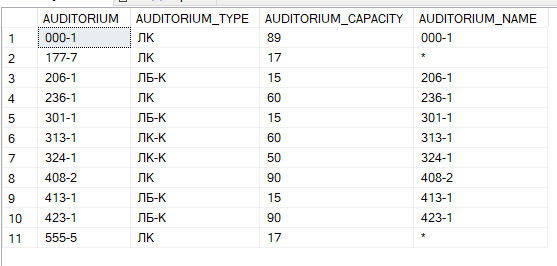
**exec @rc = TovaryInsert @t = 'Планшет', @cn = 160, @kl = 90;**

**print 'код ошибки : ' + cast(@rc as varchar(3));**

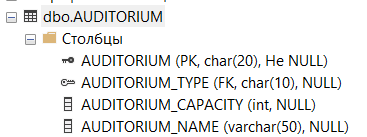
Если в операторе EXEC допустить ошибку, например, ввести повторно наименование уже имеющегося товара или неправильно использовать имена переменных, то будет выдано сообщение об ошибке.

***→ Выполнение заданий***

Строки в таблице AUDITORIUM:



Типы и ограничения столбцов:



Процедура:

go

create procedure PAUDITORIUM\_INSERT

@a char(20),

@n varchar(50),

@c int = 0,

@t char(10)

as

begin try

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

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

return 1;

end try

begin catch

print 'При выполнении процедуры произошла ошибка.';

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

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

print 'Текст ошибки: ' + error\_message();

return -1;

end catch;

Исполнение процедуры с ошибкой:

declare @result\_4 int;

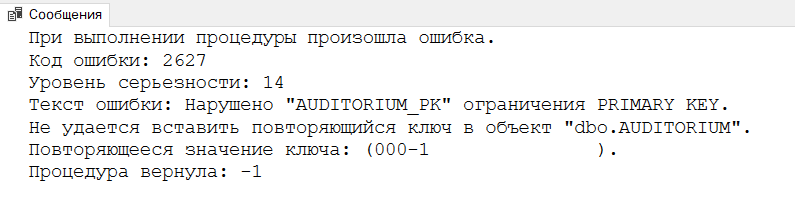
exec @result\_4 = PAUDITORIUM\_INSERT '000-1',

'000-1',

36,

'ЛК';

print 'Процедура вернула: ' + cast(@result\_4 as varchar)



Без ошибки:

declare @result\_4 int;

exec @result\_4 = PAUDITORIUM\_INSERT '222-1',

'222-1',

36,

'ЛК';

print 'Процедура вернула: ' + cast(@result\_4 as varchar)

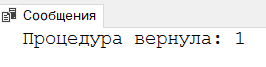
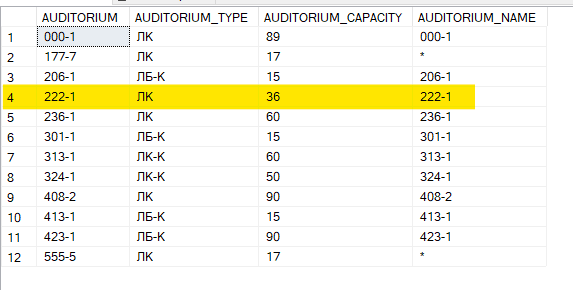
******

Таблица:

******

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.

***→ Теория (методичка)***

Пусть требуется разработать процедуру, которая выводит отчет о товарах, заказанных конкретным заказчиком:

**create procedure Zkz\_REPORT @p CHAR(50)**

**as**

**declare @rc int = 0;**

**begin try**

**declare @tv char(20), @t char(300) = ' ';**

**declare ZkTov CURSOR for**

**select Наименование\_товара from Заказы where Заказчик = @p;**

**if not exists (select Наименование\_товара**

**from Заказы where Заказчик = @p)**

**raiserror('ошибка', 11, 1);**

**else**

**open ZkTov;**

**fetch ZkTov into @tv;**

**print 'Заказанные товары';**

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

**begin**

**set @t = rtrim(@tv) + ', ' + @t;**

**set @rc = @rc + 1;**

**fetch ZkTov into @tv;**

**end;**

**print @t;**

**close ZkTov;**

**return @rc;**

**end try**

**begin catch**

**print 'ошибка в параметрах'**

**if error\_procedure() is not null**

**print 'имя процедуры : ' + error\_procedure();**

**return @rc;**

**end catch;**

Здесь для формирования сообщения об ошибке применяется встроенная функция RAISERROR, которая содержит три параметра: текстовое сообщение об ошибке, уровень серьезности ошибки и метку. Если уровень серьезности равен 11, то управление передается в блок обработки ошибок. Вызов процедуры:

**declare @rc int;**

**exec @rc = Zkz\_REPORT @p = 'Луч';**

**print 'количество товаров = ' + cast(@rc as varchar(3));**

***→ Выполнение заданий***

go

-- создаем процедуру

create procedure SUBJECT\_REPORT @p char(10)

as

begin try

declare @sub varchar(10),

@allSubjects varchar(200) = '',

@amountOfSubjects int = 0;

-- создаем курсор

declare subjectsCur cursor local

for

select SUBJECT from SUBJECT

where PULPIT = @p;

-- если нет дисциплин на кафедре @p, кидаем ошибку

if not exists (select SUBJECT from SUBJECT

where PULPIT = @p)

raiserror('ошибка в параметрах', 11, 1);

-- считываем дисциплины

print 'Дисциплины на кафедре ' + ltrim(rtrim(@p)) + ': ';

open subjectsCur

fetch subjectsCur into @sub

while @@fetch\_status = 0

begin

set @allSubjects = rtrim(@sub) + ', ' + @allSubjects;

set @amountOfSubjects += 1;

fetch subjectsCur into @sub;

end;

close subjectsCur;

-- последним символом будет запятая - удаляем её

print substring(@allSubjects, 1, len(@allSubjects) - 1);

return @amountOfSubjects;

end try

begin catch

print 'Произошла ошибка.';

print 'Текст ошибки: ' + error\_message();

if error\_procedure() is not null

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

return -1;

end catch;

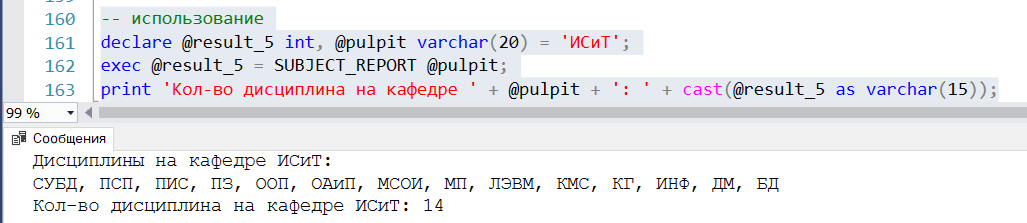
-- использование

declare @result\_5 int, @pulpit varchar(20) = 'Gtl';

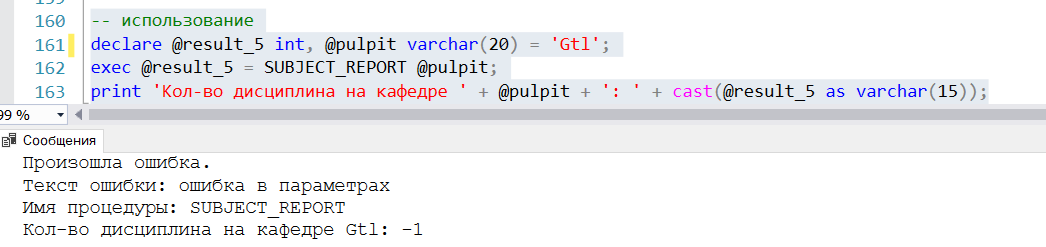
exec @result\_5 = SUBJECT\_REPORT @pulpit;

print 'Кол-во дисциплина на кафедре ' + @pulpit + ': ' + cast(@result\_5 as varchar(15));

Без ошибки:



С ошибкой:



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.

***→ Теория (методичка)***

Для базы данных ПРОДАЖИ представленная ниже процедура осуществляет добавление информации в таблицу **Заказы** и **Товары**:

**use ПРОДАЖИ**

**go**

**create procedure TovaryInsert\_X**

**@a int, @b NVARCHAR(50), @c REAL, @d INT = null,**

**@e date, @f NVARCHAR(50)**

**as declare @rc int=1;**

**begin try**

**set transaction isolation level SERIALIZABLE;**

**begin tran**

**insert into Заказы (Номер\_заказа, Наименование\_товара,**

**Цена\_продажи, Количество, Дата\_поставки, Заказчик)**

**values (@a, @b, @c, @d, @e, @f)**

**exec @rc=TovaryInsert @b, @c, @d;**

**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;**

Вызов процедуры:

**declare @rc int;**

**exec @rc = TovaryInsert\_X @a = 20, @b = 'Стол', @c = 78,**

**@d = 10, @e = '01.12.2014', @f = 'Луч';**

**print 'код ошибки=' + cast(@rc as varchar(3));**

***→ Выполнение заданий***

-- создание процедуры

go

create procedure PAUDITORIUM\_INSERTX

@a char(20),

@n varchar(50),

@c int = 0,

@t char(10),

@tn varchar(50)

as

begin try

set tran isolation level SERIALIZABLE

begin tran

declare @res int = 1;

insert into AUDITORIUM\_TYPE (AUDITORIUM\_TYPE, AUDITORIUM\_TYPENAME)

values(@t, @tn);

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

commit tran

return @res;

end try

begin catch

print 'Произошла ошибка.';

print 'Текст ошибки: ' + error\_message();

if error\_procedure() is not null

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

if @@trancount > 0

rollback;

return -1;

end catch;

-- использование

set nocount on

go

declare @result\_6 int;

declare @aud char(20) = '000-0',

@aud\_name varchar(50) = '000-0',

@aud\_cap int = 100,

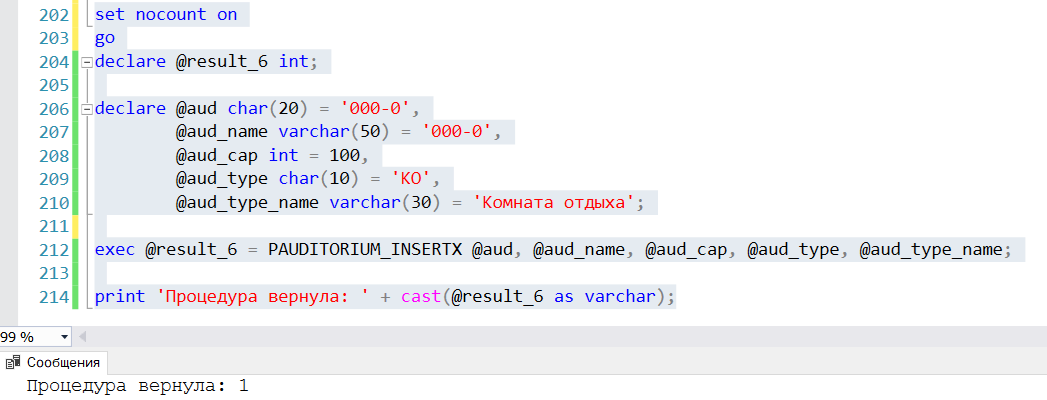
@aud\_type char(10) = 'КО',

@aud\_type\_name varchar(30) = 'Комната отдыха';

exec @result\_6 = PAUDITORIUM\_INSERTX @aud, @aud\_name, @aud\_cap, @aud\_type, @aud\_type\_name;

print 'Процедура вернула: ' + cast(@result\_6 as varchar);

Без ошибки:



С ошибкой:

