**Лабораторная работа № 8. 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.

Проанализировать результаты.

DECLARE

@ch char(1) = 'c',

@vch varchar(3) = 'var',

@date datetime,

@time time = '17:28:00',

@i int,

@smi smallint,

@tint tinyint = 2,

@num numeric(12, 5);

SET @date = '03-19-22 17:28:00';

SET @smi = (SELECT COUNT(\*) FROM GROUPS) + @tint;

SELECT @num = 12345.678;

SELECT

@date [@datetime],

@time [@time],

@smi [@smi],

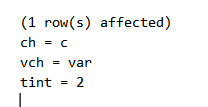
@num [@num];

PRINT 'ch = ' + CAST(@ch AS varchar(10));

PRINT 'vch = ' + CAST(@vch AS varchar(10));

PRINT 'tint = ' + CAST(@tint AS varchar(10));





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

@totalCapacity int = (

SELECT SUM(AUDITORIUM.AUDITORIUM\_CAPACITY)

FROM AUDITORIUM

),

@lessThanAverage int,

@averageCapacity float(10),

@totalAuditoriums int;

IF @totalCapacity > 200

BEGIN

SELECT

@averageCapacity = (

SELECT CAST(AVG(AUDITORIUM.AUDITORIUM\_CAPACITY) AS float(10))

FROM AUDITORIUM

),

@totalAuditoriums = (

SELECT COUNT(\*)

FROM AUDITORIUM

);

SET @lessThanAverage = (

SELECT COUNT(\*)

FROM AUDITORIUM

WHERE AUDITORIUM.AUDITORIUM\_CAPACITY < @averageCapacity

);

SELECT

@totalCapacity [Total\_Capacity],

@averageCapacity [Average\_Capacity],

@totalAuditoriums [Auditoriums\_Count],

@lessThanAverage [Less\_Than\_Average],

CAST((100.0 \* @lessThanAverage / @totalAuditoriums) AS float(2)) [Less\_Percentage]

END

ELSE

BEGIN

PRINT 'Total capacity: ' + @totalCapacity;

END

.



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).

Проанализировать результат.

PRINT '@@ROWCOUNT = ' + CAST(@@ROWCOUNT AS varchar(10));

PRINT '@@VERSION = ' + CAST(@@VERSION AS varchar(10));

PRINT '@@SPID = ' + CAST(@@SPID AS varchar(10));

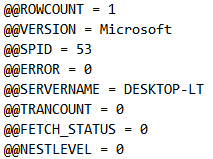
PRINT '@@ERROR = ' + CAST(@@ERROR AS varchar(10));

PRINT '@@SERVERNAME = ' + CAST(@@SERVERNAME AS varchar(10));

PRINT '@@TRANCOUNT = ' + CAST(@@TRANCOUNT AS varchar(10));

PRINT '@@FETCH\_STATUS = ' + CAST(@@FETCH\_STATUS AS varchar(10));

PRINT '@@NESTLEVEL = ' + CAST(@@NESTLEVEL AS varchar(10));



1. 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.);

− поиск студентов, у которых день рождения в следующем месяце, и определение их возраста;

− поиск дня недели, в который студенты некоторой группы сдавали экзамен по СУБД.

DECLARE @x int = 2, @t int = 3, @z float;

IF (@t > @x) SET @z = POWER(SIN(@t), 2);

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

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

PRINT 'x = ' + CAST(@x AS varchar(10));

PRINT 't = ' + CAST(@t AS varchar(10));

PRINT 'z = ' + CAST(@z AS varchar(10));

DECLARE @FIO varchar(50) = 'Макейчик Татьяна Леонидовна';

SET @FIO = (

SELECT

SUBSTRING(@FIO, 1, CHARINDEX(' ', @FIO)) +

SUBSTRING(@FIO, CHARINDEX(' ', @FIO) + 1, 1) + '.'+

SUBSTRING(@FIO, CHARINDEX(' ', @FIO, CHARINDEX(' ', @FIO) + 1) + 1, 1) + '.'

);

PRINT 'Initials: ' + @FIO;

DECLARE @nextMonth int = MONTH(GETDATE()) + 1;

SELECT

STUDENT.NAME,

STUDENT.BDAY

FROM STUDENT

WHERE MONTH(STUDENT.BDAY) = @nextMonth;

SELECT DISTINCT

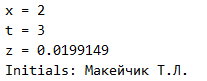
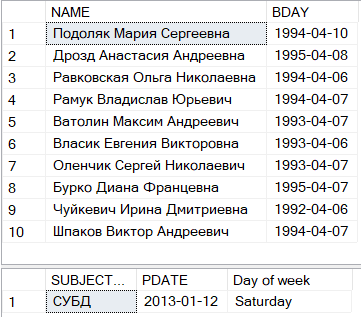
PROGRESS.SUBJECT\_T,

PROGRESS.PDATE,

DATENAME(DW, PROGRESS.PDATE) [Day of week]

FROM PROGRESS

WHERE PROGRESS.SUBJECT\_T = 'СУБД';



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

DECLARE @lectureAuditories int = (

SELECT COUNT(\*)

FROM AUDITORIUM

WHERE AUDITORIUM.AUDITORIUM\_TYPE LIKE 'ЛК%'

);

IF(@lectureAuditories > 5)

BEGIN

PRINT 'Count of lecture auditories > 5';

END

ELSE

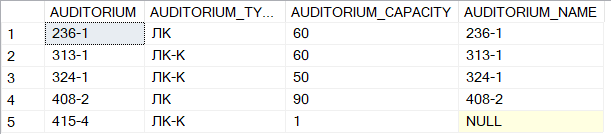
BEGIN

SELECT \*

FROM AUDITORIUM

WHERE AUDITORIUM.AUDITORIUM\_TYPE LIKE 'ЛК%';

END



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

SELECT \*

FROM (

SELECT

CASE

when (PROGRESS.NOTE IN (9, 10)) then 'perfect'

when (PROGRESS.NOTE IN (7, 8)) then 'good'

when (PROGRESS.NOTE IN (5, 6)) then 'not bad'

else 'bad'

END [MARK],

COUNT(\*) [COUNT]

FROM PROGRESS

GROUP BY CASE

when (PROGRESS.NOTE IN (9, 10)) then 'perfect'

when (PROGRESS.NOTE IN (7, 8)) then 'good'

when (PROGRESS.NOTE IN (5, 6)) then 'not bad'

ELSE 'bad'

END

) AS T

ORDER BY

CASE [MARK]

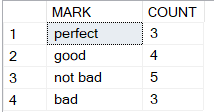
when 'perfect' then 1

when 'good' then 2

when 'not bad' then 3

else 4

END;



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

CREATE TABLE #TEST (

ID int NOT NULL IDENTITY(1,1) PRIMARY KEY,

STRING varchar(50)

);

DECLARE @index int = 0;

WHILE (@index < 10)

BEGIN

INSERT #TEST(STRING) VALUES

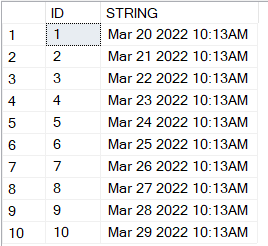
(CAST(@index + GETDATE() AS varchar(50)));

SET @index = @index + 1;

END;

SELECT \*

FROM #TEST;



1. Разработать скрипт, демонстрирующий использование оператора RETURN.

WHILE (@index > 0)

BEGIN

INSERT #TEST(STRING) VALUES

(CAST(@index + GETDATE() AS varchar(50)));

SET @index = @index + 1;

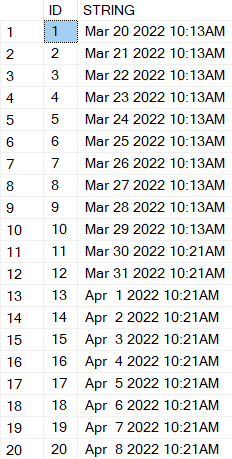
IF(@index >= 20)

RETURN;

END

SELECT \*

FROM #TEST;



1. 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

SELECT \*

FROM AUDITORIUM

WHERE AUDITORIUM.AUDITORIUM = 123;

END TRY

BEGIN CATCH

PRINT 'Error code: ' + CAST(ERROR\_NUMBER() AS varchar(10))

PRINT 'Error message: ' + ERROR\_MESSAGE()

PRINT 'Error line: ' + CAST(ERROR\_LINE() AS varchar(10))

PRINT 'Proceedure name: ' + CAST(ERROR\_PROCEDURE() AS varchar(10))

PRINT 'Error serverity: ' + CAST(ERROR\_SEVERITY() AS varchar(10))

PRINT 'Error state: ' + CAST(ERROR\_STATE() AS varchar(10))

END CATCH

