**СОДЕРЖАНИЕ**

[Часть 1. ОСНОВЫ ПРОГРАММИРОВАНИЯ С ПОМОЩЬЮ ВСТРОЕННОГО ЯЗЫКА TRANSACT-SQL В MICROSOFT SQL SERVER 3](#_Toc116471825)

[1.1 Цель работы 3](#_Toc116471826)

[1.2 Исходные данные 3](#_Toc116471827)

[1.3 Используемые программы 3](#_Toc116471828)

[1.4 Задание 3](#_Toc116471829)

[1.5 Теоретические сведения 3](#_Toc116471830)

[1.6 Задание к практической работе (часть 1) 20](#_Toc116471831)

[Часть 2. СОЗДАНИЕ, ИЗМЕНЕНИЕ, ПРИМЕНЕНИЕ И УДАЛЕНИЕ ФУНКЦИЙ И ХРАНИМЫХ ПРОЦЕДУР 22](#_Toc116471832)

[2.1 Цель работы 22](#_Toc116471833)

[2.2 Методические рекомендации для выполнения практических работ 22](#_Toc116471834)

[2.3 Задание к практической работе (часть 2) 24](#_Toc116471835)

**ЛАБОРАТОРНАЯ РАБОТА №4**

# Часть 1. ОСНОВЫ ПРОГРАММИРОВАНИЯ С ПОМОЩЬЮ ВСТРОЕННОГО ЯЗЫКА TRANSACT-SQL В MICROSOFT SQL SERVER

### Цель работы

Изучить используемый в реляционных СУБД встроенный язык программирования Transact-SQL для написания программ в MS SQL Server. Изучить правила построения идентификаторов, правила объявления переменных и их типов. Изучить принципы работы с циклами и ветвлениями. Изучить работу с переменными типа Table. Изучить синтаксис и семантику функций и хранимых процедур Transact– SQL: способов их идентификации, методов задания и спецификации параметров и возвращаемых значений, и вызовов функций и хранимых процедур.

### Исходные данные

Исходными данными является индивидуальное задание и результат предыдущих практических работ.

### Используемые программы

Программа Microsoft SQL Server Management Studio.

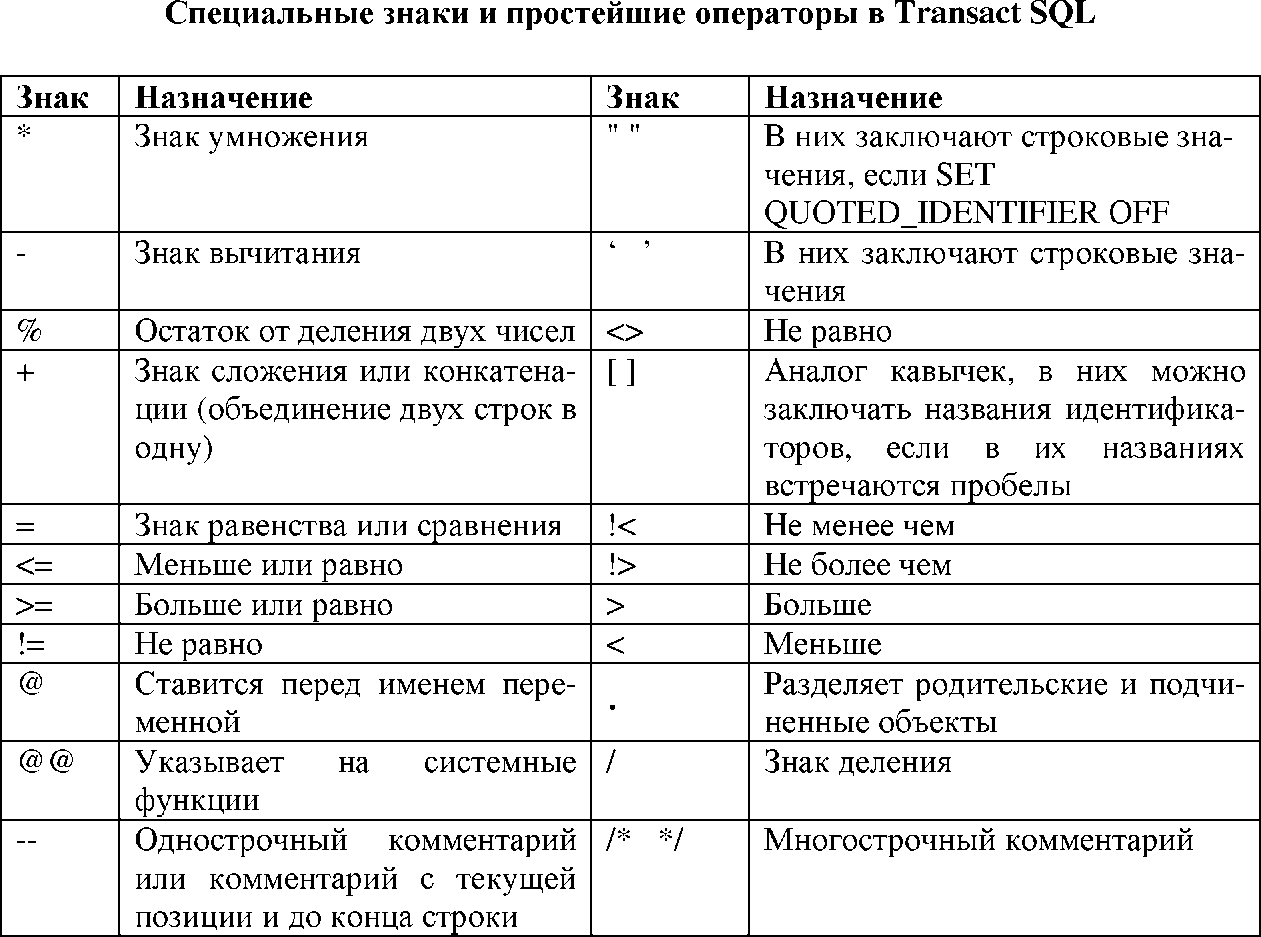
### Задание

Практическую работу следует выполнять в следующем порядке:

1. Знакомство с правилами обозначения синтаксиса команд в справочной системе MS SQL Server (утилита Books Online).
2. Изучение правил написания программ на Transact SQL.
3. Изучение правил построения идентификаторов, правил объявления переменных и их типов.
4. Изучение работы с циклами и ветвлениями.
5. Изучение работы с переменными типа Table.
6. Изучение правил написания хранимых процедур и функций.
7. Проработка всех примеров, анализ результатов их выполнения.
8. Выполнение индивидуальных заданий по вариантам.

Для освоения программирования используем пример базы данных **University,** которая была создана в предыдущих практических работах. При выполнении примеров и заданий обращайте внимание на соответствие названий БД, таблиц и других объектов проекта.

### Теоретические сведения



**Идентификаторы** - это имена объектов, на которые можно ссылаться в программе, написанной на языке Transact SQL. Первый символ может состоять из букв английского алфавита или "**@**", "**#**". Остальные дополнительно из цифр и «$».

Имя идентификатора не должно совпадать с зарезервированным словом.

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

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

**Переменные** используются для сохранения промежуточных данных в хранимых процедурах и функциях. Все переменные считаются локальными. Имя переменной должно начинаться с **@**.

**Объявление переменных**

Синтаксис в обозначениях MS SQL Server:

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

**DECLARE @ имя\_переменной1 тип\_переменной (размер), .,**

**@ имя\_переменнойN тип\_переменной(размер)**

**Пример 1:** Введите в редактор SQL запросов следующее объявление переменных

**DECLARE @a INT, @b numeric(10,2)**

**DECLARE @str CHAR(20)**

**Присвоение значений переменным и вывод значений на экран**

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

**SET @имя\_переменной = значение**

Допишите в редакторе SQL объявление переменных с вводом их значений и выводом результата

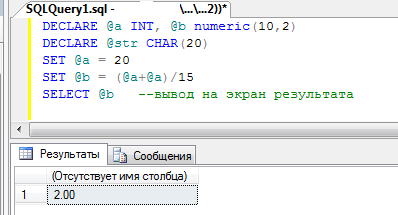
**DECLARE @a INT, @b numeric(10,2)**

**SET @a = 20**

**SET @b = (@a+@a)/15**

**SELECT @ b --вывод на экран результата**

Запустить эту конструкцию на выполнение. У вас появиться следующий результат на экране. Сохраните запрос в папку Лаб4.



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

Рассмотрим пример, в котором переменной присвоим результат выполнения агрегированной функции над таблицами базы данных

**Пример 2**: Вывести количество строк в таблице **Студенты** и присвоить это значение переменной

**DECLARE @a INT**

**SELECT @a = COUNT(\*) FROM Student**

**SELECT @a**

**Пример 3:**

**DECLARE @str CHAR(30)**

**SELECT @str = Sutname FROM Student**

**SELECT @str**

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

**Сочетание ключевых слов SET и SELECT**

Измените код запроса из примера 2 на следующий:

**DECLARE @a INT**

**SET @a = (SELECT COUNT(\*) FROM Authors)**

**SELECT @a**

**Создание временной таблицы через переменную типа TABLE**

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

**Пример 4:** Создание временной таблицы с двумя полями, в которую будет добавлены 2 строки с данными

**DECLARE @mytable TABLE (id INT, myname CHAR(20) DEFAULT 'Иванов Иван')**

**INSERT INTO @mytable(id) VALUES (1)**

**INSERT INTO @mytable(id, myname) VALUES (2,'Игорь Троцкий')**

**SELECT \* FROM @mytable**

Выполните и сохраните запрос

**Пример 5:** Создание временной таблицы с двумя полями, в которую будет добавлены строки, как результат выполнения запроса выборки данных из таблицы **факультет**

**DECLARE @mytable TABLE(id INT, myname CHAR (255) DEFAULT 'Введите название')**

**INSERT @mytable SELECT kod\_faculteta, name\_faculteta FROM facultet SELECT \* FROM @mytable**

Выполните и сохраните запрос.

**Самостоятельно создать на языке P-SQL запросы, с помощью которых:**

**Запрос 1. Подсчитать среднюю зарплату преподавателей (с помощью запроса**

**SELECT**) и умножить ее на значение 123,34, которое необходимо сохранить в отдельной переменной, вывести значение переменной на экран.

**Запрос 2.** Подсчитать суммарное значение всех стипендий у студентов, результат поместить в переменную, вывести значение переменной на экран.

**Запрос 3.** Подсчитать количество кафедр, результат поместить в переменную, вывести значение переменной на экран.

**Запрос 4.** Создать локальную таблицу с названием **TEMP** и полями типа дата/время, длинное целое, строка. Добавить в нее две записи с данными и вывести результат на экран.

**Операторские скобки**

**BEGIN**

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

\*/

**END**

**Условная конструкция IF**

Синтаксис:

**IF условие**

Набор операторов**1 ELSE**

Набор операторов**2**

**Пример 6: Использование операторских скобок и условных конструкций для поиска ответа – количество кафедр больше 10, да или нет**

**DECLARE @a INT**

**DECLARE @str CHAR(30)**

**SET @a = (SELECT COUNT(\*) FROM kafedra)**

**IF @a >10 BEGIN**

**SET @str = 'Количество кафедр больше 10' SELECT @str**

**END ELSE BEGIN**

**SET @str = 'Количество кафедр = ' + str(@a) SELECT @str**

**END**

Выполните и сохраните запрос.

**Самостоятельно создать на языке P-SQL запросы, с помощью которых**

**Запрос 5.** Подсчитать количество факультетов. Если их в таблице от 2 до 4, то ничего не сообщать, в противном случае вывести сообщение вида "В таблице ... факультетов" (вместо многоточия поставить точное количество факультетов).

**Запрос 6.** Подсчитать средний год рождения студентов. Если полученный год в диапазоне от 1980 до 1999, то ничего не сообщать, в противном случае вывести сообщение вида "Средний год рождения = ." (вместо многоточия поставить точный средний год).

**Цикл WHILE**

Синтаксис:

**WHILE Условие**

**Набор операторов1 BREAK**

**Набор опреторов2**

**CONTINUE**

Конструкции **BREAK** и **CONTINUE** являются необязательными.

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

**Пример 7:** Объявление переменной а, проверка в цикле, чтобы значение а не превосходило 100. Переменная а увеличивает свое значение в случайном порядке. Также организуется проверка на условие.

DECLARE @a INT SET @a = 1 WHILE @a <100

**BEGIN**

**PRINT @a -- вывод на экран значения переменной I**

**F (@a>40) AND (@a<50)**

**BREAK --выход и выполнение 1-й команды за циклом**

ELSE

SET @a = @a+rand()\*10

**CONTINUE**

**END**

**PRINT @a**

Выполните и сохраните запрос.

**Самостоятельно создать на языке P-SQL запросы, с помощью которых**

**Запрос 7.** Определить количество записей в таблице кафедра. Пока записей меньше 10, делать в цикле добавление записи в временную таблицу с автоматическим наращиванием значения ключевого поля, а вместо названия кафедры ставить значение 'Имя не известно'.

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

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

Исключением являются **расширенные хранимые процедуры**, которые хранятся в двоичном формате в виде динамически подключаемых библиотек типа

**\*.dll** и создаются с помощью других языков программирования с использованием интерфейса **SQL Server Open Data Services API.** Такие процедуры подключаются, отключаются и выгружаются соответственно командами **sp\_addextendedproc, sp\_dropextendedproc** и **DBCC dlname (FREE),** где dllno me\_имя dll\_библиотеки.

Хранение функций и хранимых процедур в виде исходных модулей языка Transact

SQL на сервере и в соответствующих базах данных позволяет уменьшить размер запроса, посылаемого по сети от клиента на сервер, а, следовательно, и нагрузку на сеть, что повышает общую производительность системы. Это также позволяет упростить сопровождение программных комплексов и внесение изменений в исходный текст модулей, причем большинство изменений не отразится на работоспособности клиентских приложений.

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

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

Таким образом, пользовательские функции и хранимые процедуры становятся объектами той базы данных, в которой они создавались. Поэтому при их создании, если необходимо, требуемую базу данных следует сделать текущей с помощью команды **USE** имя базы данных. Системные же функции хранятся на экземпляре сервера, а системные хранимые процедуры – в базе **MASTER** этого же экземпляра сервера.

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

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

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

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

Аргументы в вызове функции отделяются запятыми.

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

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

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

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

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

* **CREATE FUNCTION,**
* **CREATE PROCEDURE,**
* **ALTER FUNCTION,**
* **ALTER PROCEDURE,**
* **DROP FUNCTION,**
* **DROP PROCEDURE**.

**При создании функции** указывается тип возвращаемого значения и в теле функции обязательно задается команда **RETURN**, за которой следует выражения для вычисления возвращаемого значения.

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

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

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

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

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

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

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

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

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

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

**Создание функций**

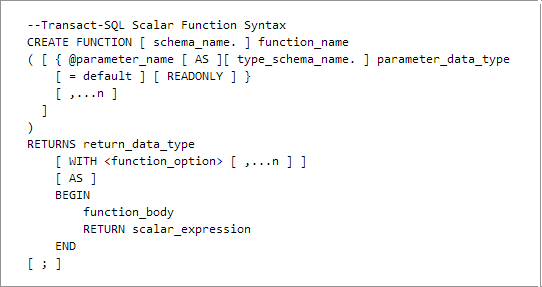
Команда:

**CREATE FUNCTION (Transact-SQL)**

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

1. В инструкциях Transact-SQL, например, SELECT.
2. В приложениях, вызывающих функцию.
3. В определении другой пользовательской функции.
4. Для параметризации представления или улучшения функциональности индексированного представления.
5. Для определения столбца таблицы.
6. Для определения ограничения CHECK на столбец.
7. Для замены хранимой процедуры.

**Синтаксис создания скалярной функции.**



**Аргументы**

**schema\_name** - Имя схемы, к которой принадлежит определяемая пользователем функция.

**function\_name** - Имя определяемой пользователем функции. Имена функций должны удовлетворять правилам построения идентификаторов и должны быть уникальными в пределах базы данных и схемы.

**@parameter\_name** - Аргумент пользовательской функции. Может быть объявлен один или несколько аргументов.

Для функций допускается не более 2 100 параметров. При выполнении функции значение каждого из объявленных параметров должно быть указано пользователем, если для них не определены значения по умолчанию.

[ **type\_schema\_name**. ] **parameter\_data\_type -** Тип данных параметра

**[ =default ] -** Значение по умолчанию для аргумента. Если определено значение default, то функция выполняется даже в том случае, если для данного аргумента значение не указано.

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

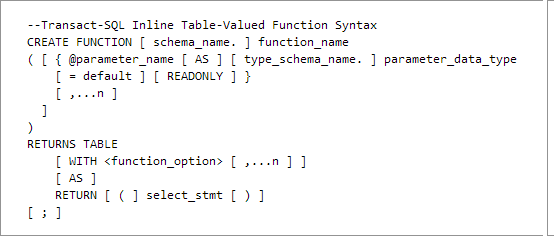
**return\_data\_type** - Возвращаемое значение скалярной функции, определяемой пользователем.

**function\_body** - Указывает серию инструкций Transact-SQL

**scalar\_expression** - Указывает скалярное значение, возвращаемое скалярной функцией.

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

**Синтаксис функции, которая возвращает табличное значение:**



**Аргументы:**

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

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

**select\_stmt -** Одиночная инструкция **SELECT**, определяющая возвращаемое значение встроенной функции с табличным значением.

В функциях допустимы следующие инструкции.

* Инструкции присваивания.
* Инструкции управления потоком, за исключением инструкций TRY...CATCH.
* Инструкции DECLARE, объявляющие локальные переменные и локальные курсоры.
* Инструкции SELECT, которые содержат списки выбора с выражениями, присваивающими значения локальным переменным.
* Операции над локальными курсорами, которые объявляются, открываются, закрываются и освобождаются в теле функции. Допустимы только те инструкции FETCH, которые предложением INTO присваивают значения локальным переменным. Инструкции FETCH, возвращающие данные клиенту, недопустимы.
* Инструкции INSERT, UPDATE и DELETE, которые изменяют локальные табличные переменные.
* Инструкции EXECUTE, вызывающие расширенные хранимые процедуры.

**Ограничения**

* Определяемые пользователем функция не может выполнять действия, изменяющие состояние базы данных.
* Определяемые пользователем функции не могут содержать предложение OUTPUT INTO, целью которого является таблица.
* Определяемые пользователем функции могут быть вложенными, то есть из одной функции может быть вызвана другая. Вложенность определяемых пользователем функций не может превышать 32 уровней.

**Пример 1. Применение скалярной определяемой пользователем функции, вычисляющей месяц.**

В следующем примере показано создание определяемой пользовательской функции **ISOweek,** которая получает в качестве аргумента дату и вычисляет номер месяца, а затем по номеру определяет название месяца. Для правильной работы этой функции перед ее вызовом должна быть выполнена инструкция **SET DATEFIRST 1.**

Следующий пример также показывает использование предложения **EXECUTE AS** для указания контекста безопасности, в котором может быть выполнена хранимая процедура. В этом примере параметр **CALLER** указывает, что процедура будет выполнена в контексте пользователя, который ее вызывает.

В начале, прежде чем создавать функцию, будет выполнена проверка на наличие в базе данных функции под таким же именем. Если функция с таким именем уже была создана ранее, то она будет удалена с помощью команды **DROP FUNCTION** и создана заново **CREATE FUNCTION**.

Откройте sql-редактор. Создайте новый запрос.

**USE University;**

**GO**

**IF OBJECT\_ID (N'dbo.ISOweek', N'FN') IS NOT NULL**

**DROP FUNCTION dbo.ISOweek;**

**GO**

**CREATE FUNCTION dbo.ISOweek (@DATE date) RETURNS CHAR(15)**

**WITH EXECUTE AS CALLER AS**

**BEGIN**

**DECLARE @man int;**

**DECLARE @ISOweek char(15);**

**SET @man= MONTH(@DATE)**

**IF (@man=1) SET @ISOweek='Январь';**

**IF (@man=2) SET @ISOweek='Февраль';**

**IF (@man=3) SET @ISOweek='Март';**

**IF (@man=4) SET @ISOweek='Апрель';**

**IF (@man=5) SET @ISOweek='Май';**

**IF (@man=6) SET @ISOweek='Июнь';**

**IF (@man=7) SET @ISOweek='Июль';**

**IF (@man=8) SET @ISOweek='Август';**

**IF (@man=9) SET @ISOweek='Сентябрь';**

**IF (@man=10) SET @ISOweek='Октябрь';**

**IF (@man=11) SET @ISOweek='Ноябрь';**

**IF (@man=12) SET @ISOweek='Декабрь';**

**RETURN(@ISOweek);**

**END;**

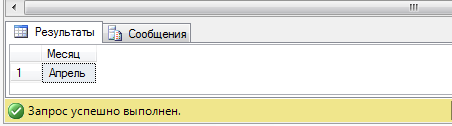
Для того чтобы увидеть результат, выведем на экран вычисление значение функции от произвольной даты, например '12.04.2004'. для этого ниже в sql-редакторе пишем:

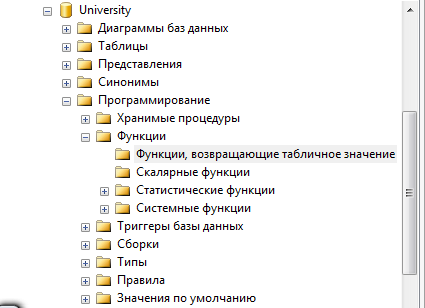
**GO**

**SET DATEFIRST 1;**

**SELECT dbo.ISOweek('12.04.2004') AS 'Месяц';**

Ниже приводится результирующий набор.



Просмотреть все функции, которые пользователь может создавать, откройте в окне обозревателя объектов в базе данных **University** группу **Программирование/Функции/Скалярные функции**.

**Пример 2. Создание пользовательской функции с табличным значением**.

Например, создадим пользовательскую функцию, которая будет разрешена к использованию всеми пользователями с правами роли «**dekan**» (созданной в предыдущих работах). Функция будет возвращать результат в виде **таблицы** – вывод на всех кафедрах суммирующей зарплаты по каждой должности всех преподавателей. При этом функция имеет один параметр **@storeid**, с помощью которого введем ограничение на вычисление, а именно зарплата должна быть больше 100.

Создайте новый запрос:

**USE University;**

**GO**

**IF OBJECT\_ID (N'ufn\_SalesByStore', N'IF') IS NOT NULL**

**DROP FUNCTION dekan.ufn\_SalesByStore;**

**GO**

**CREATE FUNCTION dekan.ufn\_SalesByStore(@storeid int) RETURNS TABLE**

**AS RETURN (**

**SELECT d.Name\_kafedru AS "Кафедра", t.Dolgnost AS "Должность",**

**SUM(t.Salary + t.RISE) AS "Сумма зарплаты" FROM KAFEDRA d, TEACHER t**

**WHERE d.KOD\_kafedru =t.KOD\_kafedru and t.salary>@storeid**

**GROUP BY d.Name\_kafedru, t.Dolgnost**

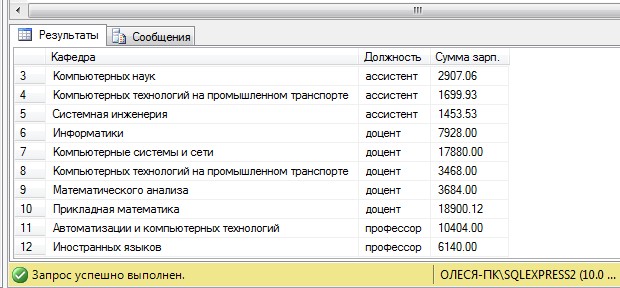
**);**

Для вызова этой функции нужно выполнить следующий запрос:

**GO**

**SELECT \* from dekan.ufn\_SalesByStore(100);**

Ниже приводится результирующий набор.



**Самостоятельно создайте запрос:**

**Запрос 8. С**оздать пользовательскую функцию, которая будет возвращать результат в виде **таблицы, а именно** выводить всех учащихся студентов по кафедрам с указанием курса. При этом функция имеет один параметр **@city**, с помощью которого введем ограничение на вычисление, а именно город проживания должен быть Ростов-на- дону.

**Создание хранимых процедур в Microsoft SQL Server**

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

**Типы хранимых процедур**

Системные хранимые процедуры предназначены для выполнения различных административных действий. Практически все действия по администрированию сервера выполняются с их помощью.

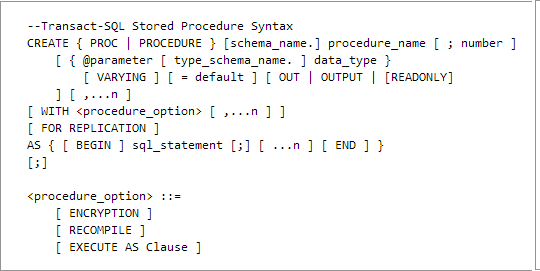
Пользовательские хранимые процедуры реализуют те или иные действия. Хранимые процедуры - полноценный объект базы данных. Вследствие этого каждая хранимая процедура располагается в конкретной базе данных, где и выполняется.

Временные хранимые процедуры существуют лишь некоторое время, после чего автоматически уничтожаются сервером. Они делятся на локальные и глобальные.

**Создание, изменение хранимых процедур**

Создание хранимой процедуры предполагает решение следующих задач: планирование прав доступа. При создании хранимой процедуры следует учитывать, что она будет иметь те же права доступа к объектам базы данных, что и создавший ее пользователь; определение параметров хранимой процедуры, хранимые процедуры могут обладать входными и выходными параметрами; разработка кода хранимой процедуры. Код процедуры может содержать последовательность любых команд SQL, включая вызов других хранимых процедур.

**Синтаксис оператора создания новой или изменения имеющейся хранимой процедуры в обозначениях MS SQL Server:**



Рассмотрим параметры данной команды.

Используя префиксы **sp\_, #, ##,** создаваемую процедуру можно определить в качестве **системной** или **временной**. Как видно из синтаксиса команды, не допускается указывать имя владельца, которому будет принадлежать создаваемая процедура, а также имя базы данных, где она должна быть размещена.

Таким образом, чтобы разместить создаваемую хранимую процедуру в конкретной базе данных, необходимо выполнить команду **CREATE PROCEDURE** в контексте этой базы данных. При обращении из тела хранимой процедуры к объектам той же базы данных можно использовать укороченные имена, т. е. без указания имени базы данных. Когда же требуется обратиться к объектам, расположенным в других базах данных, указание имени базы данных обязательно.

Для передачи входных и выходных данных в создаваемой хранимой процедуре имена параметров должны начинаться с символа **@**. В одной хранимой процедуре можно задать множество параметров **@parameter**, разделенных запятыми. В теле процедуры не должны применяться локальные переменные, чьи имена совпадают с именами параметров этой процедуры.

Наличие ключевого слова **OUTPUT** означает, что соответствующий параметр предназначен для **возвращения данных из хранимой процедуры**. Указание ключевого слова **OUTPUT** предписывает серверу при выходе из хранимой процедуры присвоить текущее значение параметра локальной переменной, которая была указана при вызове процедуры в качестве значения параметра.

Ключевое слово **VARYING** применяется совместно с параметром OUTPUT, имеющим тип CURSOR. Оно определяет, что выходным параметром будет результирующее множество.

Ключевое слово **DEFAULT** представляет собой значение, которое будет принимать соответствующий параметр по умолчанию. Таким образом, при вызове процедуры можно не указывать явно значение соответствующего параметра.

Ключевое слово **AS** размещается в начале собственно тела хранимой процедуры.

В теле процедуры могут применяться практически все команды SQL, объявляться транзакции, устанавливаться блокировки и вызываться другие хранимые процедуры.

Выход из хранимой процедуры можно осуществить посредством команды

**RETURN.**

**Удаление хранимой процедуры**

**DROP PROCEDURE {имя\_процедуры} [,...n]**

**Выполнение хранимой процедуры**

Для выполнения хранимой процедуры используется команда:

**[[ EXEC [ UTE] имя\_процедуры [;номер]**

**[[@имя\_параметра=]{ значение | @имя\_переменной}**

**[OUTPUT ]|[DEFAULT ]][,...n]**

Если вызов хранимой процедуры не является единственной командой в пакете, то присутствие команды **EXECUTE** обязательно. Более того, эта команда требуется для вызова процедуры из тела другой процедуры или триггера.

Использование ключевого слова **OUTPUT** при вызове процедуры разрешается только для параметров, которые были объявлены при создании процедуры с ключевым словом **OUTPUT**.

Когда же при вызове процедуры для параметра указывается ключевое слово **DEFAULT**, то будет использовано значение по умолчанию. Естественно, указанное слово **DEFAULT** разрешается только для тех параметров, для которых определено значение по умолчанию.

Из синтаксиса команды **EXECUTE** видно, что имена параметров могут быть опущены при вызове процедуры. Однако в этом случае пользователь должен указывать значения для параметров в том же порядке, в каком они перечислялись при создании процедуры. Присвоить параметру значение по умолчанию, просто пропустив его при перечислении, нельзя. Если же требуется опустить параметры, для которых определено значение по умолчанию, достаточно явного указания имен параметров при вызове хранимой процедуры. Более того, таким способом можно перечислять параметры и их значения в произвольном порядке.

Отметим, что при вызове процедуры указываются либо имена параметров со значениями, либо только значения без имени параметра. Их комбинирование не допускается.

**Использование RETURN в хранимой процедуре**

Позволяет выйти из процедуры в любой точке по указанному условию, а также позволяет передать результат выполнения процедуры числом, по которому можно судить о качестве и правильности выполнения процедуры.

**Примеры создания процедур**

**Пример 3. Создание процедуры без параметров. Процедура вычисляет кол-во всех ассистентов:**

**CREATE PROCEDURE Count\_Assistent**

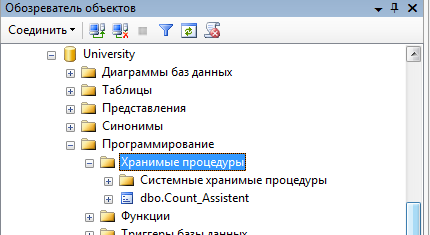
**AS**

**Select count(DOLGNOST) from TEACHER**

**where DOLGNOST='Ассистент'**

**Go**

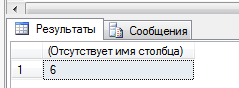
Создайте данную процедуру к базе данных University через утилиту SQL server Management Studio. Выполните запрос. Посмотрите созданную процедуру в разделе Программирование/Хранимые процедуры базы данных University.



Запустите ее с помощью команды:

**EXECUTE Count\_Assistent**

Проверьте результат. Количество ассистентов = 6.



**Пример 4**. Создание процедуры c входным параметром. Например, нужно посчитать кол-во ассистентов, зарплата у которых не более заданного параметра @Sum\_salary:

**CREATE PROCEDURE Count\_Assistent\_Salary @Sum\_salary as Int**

**AS**

**Select count(DOLGNOST) from TEACHER**

**WHERE DOLGNOST='Ассистент' and SALARY>=@Sum\_salary**

**Go**

Создайте данную процедуру. Запустите ее с помощью команды EXEC **Count\_Assistent\_Salary 1500**

Проверьте результат.

**Самостоятельно создать запрос:**

**Запрос 8**. Создать процедуру Count\_Assistent\_Salary\_Title c входными параметрами @Sum\_salary целого типа, @Title строка с длиной 15 символов, которая определяет кол-во преподавателей, должность которых совпадает с параметром @Title и зарплата у которых не менее заданного параметра @Sum\_salary.

Создайте данную процедуру. Запустите ее с помощью команды **EXECUTE Count\_Assistent\_Salary\_Title 1300, '%нт%'** Проверьте результат.

**Пример 6**. Создание процедуры c входными параметрами и выходным параметром:

**CREATE PROCEDURE**

**Count\_Assistent\_Itogo @Sum\_salary Int, @Title Char(15) , @Itogo Int OUTPUT AS**

**Select @Itogo = count(DOLGNOST) from TEACHER**

**WHERE SALARY<=@Sum\_salary AND DOLGNOST LIKE @Title**

**Go**

Создайте данную процедуру. Запустите с помощью набора команд:

**Declare @q As int**

**EXEC Count\_Assistent\_Itogo 2000, '%тент%', @q output select @q**

**Проверьте результат.**

**Пример 7.** Создание процедуры c входными параметрами и RETURN. Пусть процедура проверяет, если номер студента равен параметру @param и одновременно его фамилия 'Петрова', то выдать 1, иначе 2:

**CREATE PROCEDURE checkname @param int AS**

**IF (SELECT SUTFNAME FROM STUDENT WHERE STUDENT\_ID = @param)**

**= 'Петрова'**

**RETURN 1 ELSE RETURN 2**

Создайте данную процедуру. Запустите ее с помощью команд:

**DECLARE @return\_status int**

**EXECUTE @return\_status = checkname 15 SELECT 'Return Status' = @return\_status**

Проверьте результат.

**Пример 8**. Создание процедуры без параметров для уменьшения значения стипендии в таблице Student на 50 грн:

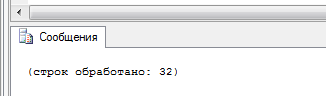
**CREATE PROC update\_proc AS**

**UPDATE STUDENT SET stipend = stipend-50**

Процедура не возвращает никаких данных.

Создайте данную процедуру. Запустите ее с помощью команды

**EXEC update\_proc**

Проверьте результат.

**Пример 9**. Создание процедуры с входным параметром для получения всей информации о конкретном заведующем кафедры:

**CREATE PROC select\_zavkaf @k CHAR(10) AS**

**SELECT \* FROM kafedra WHERE fio\_zavkaf=@k**

Создайте данную процедуру. Запустите ее с помощью команд:

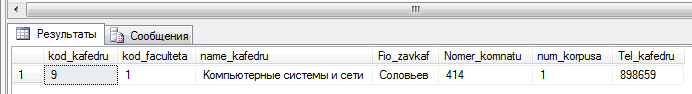
**EXEC select\_zavkaf 'Соловьев'**

или

**select\_author @k= 'Соловьев'**

или

**EXEC select\_author @k='Соловьев'**



**Самостоятельно создайте запрос:**

**Запрос 9.** Создать процедуру update\_proc\_rise с входным параметром и значением по умолчанию @p real = 0.5 для увеличения значения надбавки к зарплате в таблице TEACHER в заданное количество раз:

Процедура не возвращает никаких данных.

Создайте данную процедуру. Запустите ее с помощью команд:

**EXEC update\_proc\_rise 1.5**

или

**EXEC update\_proc\_rise @p = 2**

или

**EXEC update\_proc\_rise --будет использовано значение по умолчанию.**

**Пример 11.** Создание процедуры с входным и выходным параметрами. Создать процедуру для определения количества преподавателей, с датой приема на работу в указанный период:

**CREATE PROC count\_teacher**

**@d1 DATE, @d2 DATE, @c INT OUTPUT**

**AS**

**SELECT @c=count(KOD\_TEACHER) from teacher**

**WHERE Data\_hire BETWEEN @d1 AND @d2 SET @c = ISNULL(@c,0)**

Создайте данную процедуру. Запустите ее с помощью команд:

**DECLARE @c2 INT**

**EXEC count\_teacher '01/01/2006', '31/12/2008', @c2 OUTPUT SELECT @c2**

### Задание к практической работе (часть 1)

Для созданной базы данных, согласно номеру варианта, самостоятельно создать **на языке P-SQL 10 запросов:**

* 1 запрос для создания временной таблицы через переменную типа **TABLE**;
* 2 запроса с использованием условной конструкции **IF**;
* 2 запроса с использованием цикла **WHILE**;
* 1 запрос для создания скалярной функции;
* 1 запрос для создания функции, которая возвращает табличное значение;
* 2 запроса для создания процедуры без параметров;
* 2 запроса для создания процедуры c входным параметром;
* 2 запроса для создания процедуры c входными параметрами и **RETURN**;
* 2 запроса для создания процедуры обновления данных в таблице базы данных

**UPDATE**;

* 2 запроса для создания процедуры извлечения данных из таблиц базы данных

**SELECT**;

Все программные инструкции команд SQL сохранять в файлах с расширением

**\*.sql** в папке **ФИО\_студента/Лаб4**.

Для каждого запроса сформулировать текстовое задание, которое должно быть выполнено к базе данных.

Создать текстовый отчет, в котором отобразить sql-команды разработанных запросов и скриншоты результатов работы из СУБД **SQL Server Management Studio**.

# Часть 2. СОЗДАНИЕ, ИЗМЕНЕНИЕ, ПРИМЕНЕНИЕ И УДАЛЕНИЕ ФУНКЦИЙ И ХРАНИМЫХ ПРОЦЕДУР

### Цель работы

Изучение синтаксиса и семантики функций и хранимых процедур Transact– SQL: способов их идентификации, методов задания и спецификации параметров и возвращаемых значений, кодирования тела и вызовов функций и хранимых процедур, применение команд для создания, изменения и удаления системных и пользовательских как скалярных, так и табличных (с одной Inline или несколькими Multi – statement командами в теле) функций, системных, пользовательских, временных (локальных или глобальных) и расширенных хранимых процедур, а также приобретение навыков программирования, отладки, тестирования и включения в группу или подключения библиотеки функций и хранимых процедур.

### Методические рекомендации для выполнения практических работ

Функции и хранимые процедуры используются в SQL Server для реализации на языке Transact-SQL сложных часто используемых алгоритмов обработки данных или различных административных действий создания учетных записей, получения информации об объектах базы данных, управления свойствами сервера и баз данных, управления подсистемой репликации и автоматизации и т.д. Они хранятся в виде исходного текста и являются программными модулями, существующими независимо от таблиц или каких либо других объектов баз данных. Исключением являются расширенные хранимые процедуры, которые хранятся в двоичном формате в виде динамически подключаемых библиотек типа \*.dll и создаются с помощью других языков программирования с использованием интерфейса SQL Server Open Data Services API. Такие процедуры подключаются, отключаются и выгружаются соответственно командами sp\_addextendedproc, sp\_dropextendedproc и DBCC dlname (FREE), где dllno me\_имя dll\_библиотеки.

Хранение функций и хранимых процедур в виде исходных модулей языка Transact

* SQL на сервере и в соответствующих базах данных позволяет уменьшить размр запроса, посылаемого по сети от клиента на сервер, а, следовательно, и нагрузку на сеть, что повышает общую производительность системы. Это также позволяет упростить сопровождение программных комплексов и внесение изменений в исходный текст модулей, причем большинство изменений не отразится на работоспособности клиентских приложений.

Значительная часть функций и хранимых процедур поставляются в составе SQL Server. Они называются системными, или встроенными (built - in). Кроме того, пользователю предоставляется возможность разрабатывать и включать в свою базу данных собственные, или пользовательские (user-defined) функции и хранимые процедуры, реализующие специальные алгоритмы обработки данных. Таким образом, пользовательские функции и хранимые процедуры становятся объектами той базы данных, в которой они создавались. Поэтому при их создании, если необходимо, требуемую базу данных следует сделать текущей с помощью команды USE имя базы данных. Системные же функции хранятся на экземпляре сервера, а системные хранимые процедуры – в базе master этого же экземпляра сервера.

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

**Локальные хранимые процедуры** должны иметь имя, начинающееся с символа #, и могут быть вызваны только из того соединения, в котором они были созданы. Они автоматически удаляются при отключении пользователя, перезапуске или остановке сервера.

**Глобальные хранимые процедуры** должны иметь имя, начинающееся с символов ##, и доступны для любых соединений с экземпляром сервера, на котором они были созданы. Они удаляются либо при закрытии соединения, в контексте которого они были созданы, либо автоматически – при перезапуске или остановке сервера.

Функции и хранимые процедуры могут быть вызваны клиентскими программами, другими функциями или хранимыми процедурами, а также триггерами. В любом случае необходимо указать имя функции или хранимой процедуры и список аргументов, которые сопоставляются параметрам соответствующей функции или хранимой процедуры при этом типы аргументов и параметров должны совпадать или допускать автоматические преобразования типов. Если для некоторого параметра задано значение по умолчанию и это значение подходит для данного вызова, то соответствующий аргумент может быть опущен. Поскольку функция возвращает значение, она используется в качестве операнда некоторого выражения в виде вызова функций, состоящего из имени этой функции и списка аргументов, заключенного в круглые скобки, при этом в качестве аргументов могут быть любые выражения языка Transact – SQL, дающие в результате значения требуемых типов. Аргументы в вызове функции отделяются запятыми. Если список аргументов пуст, то круглые скобки после имени функции, как правило, задаются.

Исключения составляют некоторые системные функции, для которых круглые скобки не задаются, когда нет аргументов. Хранимые процедуры могут вызываться только командой EXECUTE, или сокращенно EXEC. За этой командой должны быть указаны имя процедуры и через пробел список аргументов, если вызывается процедура с параметрами. Аргументы разделяются запятой. Если для параметра задано значение по умолчанию, то аргумент либо совсем не задается (в конце списка), либо используется слово DEFAULT (в середине списка). Процедура может возвращать результаты только через параметры с ключевым словом OUTPUT, при этом и аргумент должен быть задан с таким же ключевым словом.

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

**CREATE FUNCTION, CREATE PROCEDURE, ALTER FUNCTION, ALTER PROCEDURE, DROP FUNCTION, DROP PROCEDURE.**

При создании функции указывается тип возвращаемого значения и в теле функции обязательно задается команда RETURN, за которой следует выражения для вычисления возвращаемого значения. В теле процедуры использование команды RETURN (конечно, без последующего выражения) вовсе не обязательно. Когда этой команды нет, выход из процедуры будет происходить после исполнения последней команды процедуры.

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

Поскольку каждая из них храниться как отдельный объект, то для указания конца тела не требуется записывать какое-либо специальное ключевое слово или знак. За командами создания функции или хранимой процедуры перечисляются имена параметров, начинающиеся с символа @, и их типы, а также важно значение по умолчанию. Для функции этот список заключается в круглые скобки, после которых записывается ключевое слово RETURNS (возвращает) и тип возвращаемого значения. Для хранимой процедуры круглые скобки не используются, и задавать тип возвращаемого значение не требуется. Для тела функции часто используют ключевое слово begin после ключевого слова as и ключевое слово end в конце тела. Дополнительные опции функции или хранимой процедуры задаются ключевым словом with до начала тела. Например, опция encryption позволяет зашифровать исходный текст функции или хранимой процедуры и сделать его, таким образом, нечитабельным. Опция функции schemabinding запрещает производить какие-либо изменения в объектах базы данных. Опция хранимой процедуры recompile обеспечивает повторную компиляцию исходного текста процедуры при каждом её вызове. Наконец, опция FOR REPLICATION указывает, что данная хранимая процедура будет использоваться при репликации данных. При создании однотипных хранимых процедур можно использовать для них одно групповое имя. В этом случае конкретная процедура в группе идентифицируется своим номером, который должен задаваться как при создании, так и при вызове процедуры сразу же после группового имени и отделяться от него точкой с запятой.

Функции и хранимые процедуры можно создавать также с помощью Enterprise Manager, а хранимые процедуры еще и с помощью мастера Create Stored Procedure Wisard.

В SQL Server можно создавать функции трех классов:

–**Scalar** – возвращают обычное скалярное значение;

–**Inline** – возвращают таблицу динамической структуры, создаваемую единственной командой тела функции SELECT;

–**Multi – statement** – возвращает обычную таблицу заданной структуры, при этом количество команд в теле функции не ограничивается.

### Задание к практической работе (часть 2)

**Задание 1**. Создать функцию для выполнения четырех арифметических операций “+”, “- ”, “\*” и “/” над целыми операндами типа bigint, выполнив кодирование и проверку:

**Кодирование**

CREATE FUNCTION Calculator (@ Opd1 bigint,

@ Opd2 bigint,

@ Oprt char(1) = “\*”) RETURNS bigint

AS BEGINT

DECLARE @ Result bigint SET @ Result =

CASE @ Oprt

WHEN “+” THEN @ Opd1 + @ Opd2 WHEN “-” THEN @ Opd1 - @ Opd2

WHEN “\*” THEN @ Opd1 \* @ Opd2 WHEN “/” THEN @ Opd1 / @ Opd2 ELSE 0

END

Return @ Result END

**Тестирование**

SELECT dbo.Calculator(4,5, ‘+’),

dbo. Calculator(3,7, ‘\*’) – dbo.Calculator(64,4,‘/’)\*2. 9 -11

(1 row (s) affected)

**Задание 2.** Создать функцию, возвращающую таблицу с динамическим набором столбцов, выполнив кодирование и тестирование:

**Кодирование**

CREATE FUNCTION DYNTAB (@ State char(2))

RETURNS Table AS

RETURNS SELECT au\_id, au\_lname, au\_fname FROM authors WHERE state = @ state

**Проверка**

SELECT \* FROM DYNTAB (“CA”)

ORDER BY au\_lname, au\_fname au\_id au\_lname au\_fname

**Задание 3**. Создать функцию, разбивающую входную строку на подстроки, используя в качестве разделителя пробелы, выполнив кодирование и тестирование:

**Кодирование**

CREATE FUNCTION Parse (@ String nvarchar (500))

RETURNS @ tabl TABLE

(Number int IDENTITY (1,1) NOT NULL,

Substr nvarchar (30)) AS

BEGIN

DECLARE @ Str1 nvarchar (500), @ Pos int SET@Str1 = @String

WHILE 1>0 BEGIN

SET@Pos = CHARINDEX(“ ”, @Str1) IF @POS>0

BEGIN

INSERT INTO @tab1

VALUES (SUBSTRING (@Str1,1,@Pos)) END

ELSE BEGIN

INSERT INTO @tab1e VALUES (@Str1) BREAK

END END RETURN END

**Тестирование**

DECLARE @ Test String nvchar (500)

Set @ TestString = ‘”SQL Server 2019” SELECT \* FROM Parse (@ Test String)

Number Substr

1. SQL
2. Server

3 2019

(3 row(s)) affected)

**Задание 4**. Создать указанные три функции с помощью утилиты Enterprise Manager и проверить их синтаксис, щелкнув по кнопке Check Syntax (проверить синтаксис), затем сохранить их как шаблон, щелкнув по кнопке Save as Template (сохранить как шаблон).

**Задание 5.** Создать и отредактировать функцию, используя шаблон, полученный в задании №4.

**Задание 6.** Выбрать вновь созданную функцию и, дважды щелкнув по ней, открыть окно редактирования, а затем отредактировать текст этой функции и права доступа пользователей и ролей к данной функции.

**Задание 7**. Используя утилиту Enterprise Manager, ознакомиться с текстами исходных модулей системных функций (если это возможно), их семантикой и способами вызова в выражения, отображая при этом результаты их работы.

**Задание 8**. С помощью Transact-SQL создать три хранимые процедуры, семантика которых аналогична рассмотренным функциям, введя при этом дополнительный параметр для получения результата работы соответствующей процедуры.

**Задание 9**. Отредактировать указанные хранимые процедуры с помощью Enterprise Manager.

**Задание 10**. Создать эти же процедуры с помощью мастера Create Stored Procedure Wizard и проанализировать их свойства. Протестировать вновь созданные хранимые процедуры. С помощью sp\_help получить справочные данные по одной из этих процедур.