УО «Полоцкий государственный университет»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ №6

к выполнению лабораторной работы
по курсу «Базы данных» для специальности
Программное обеспечение информационных технологий 1-40 01 01

Новополоцк 2016

E-mail: irina.psu@gmail

Методические указания разработала:

Ст. препод. кафедры технологий программирования Бураченок Ирина Брониславовна

ТЕМА: знакомство с основными особенностями программирования в MS SQL Server средствами встроенного языка Transact SQL.

ЦЕЛЬ: Научиться создавать пользовательские скалярные и табличные функции.

Результат обучения:

После успешного завершения занятия пользователь должен:

- иметь понятие о пользовательских функциях;
- уметь создавать собственные (пользовательские) функции, добавляющие и расширяющие функции, предоставляемые системой.

Используемая программа: Microsoft SQL Server 2008.

План занятия:

1. Изучить теоретический материал.
2. Выполнить все задания по ходу лабораторной работы.
3. Выполнить индивидуальное задание.

ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ В TRANSACT-SQL

Понятие о пользовательских функциях

В SQL Server вы можете создавать собственные функции, добавляющие и расширяющие функции, предоставляемые системой. Функции могут получать 0 или более параметров и возвращать скалярное значение или таблицу.

Входные параметры могут быть любого типа, исключая *timestamp, cursor, table.*

Сервер SQL поддерживает три типа функций, определенных пользователем:

Скалярные функции – похожи на встроенные функции.

Функция, возвращающая таблицу – возвращает результат единичного оператора SELECT. Он похож на объект просмотра, но имеет большую эластичность благодаря использованию параметров, и расширяет возможности индексированного объекта просмотра.

Многооператорная функция – возвращает таблицу, созданную одним или несколькими операторами Transact-SQL, чем напоминает хранимые процедуры. В отличие от процедур, на такие функции можно ссылаться в **WHERE** как на объект просмотра.

Создание функции

Создание функций очень похоже на создание процедур и объектов просмотра. Недаром мы рассматриваем все эти темы в одной главе. Для создания функции используется оператор **CREATE FUNCTION**. В зависимости от типа, Объявление будет отличаться.

Далее рассмотрим все три типа объявления.

Скалярная функция:

```
CREATE  FUNCTION [ owner_name. ] function_name
   ( [ { @parameter_name [AS] scalar_parameter_data_type [ = default ] }
     [ ,...n ] ] )
RETURNS scalar_return_data_type
[ WITH < function_option> [ [,] ...n] ]
[ AS ]
BEGIN
   function_body
   RETURN scalar_expression
END
```

Функция, возвращающая таблицу:

```
CREATE FUNCTION [ owner_name. ] function_name
   ( [ { @parameter_name [AS] scalar_parameter_data_type [ = default ] }
    [ ,...n ] ] )
RETURNS TABLE
[ WITH < function_option > [ [,] ...n ] ]

[ AS ]
RETURN [ ( ] select-stmt [ ) ]
```

Многооператорные функции:

```
CREATE FUNCTION [ owner_name. ] function_name
   ( [ { @parameter_name [AS] scalar_parameter_data_type [ = default ] }
    [ ,...n ] ] )
RETURNS @return_variable TABLE < table_type_definition >
[ WITH < function_option > [ [,] ...n ] ]
[ AS ]
BEGIN
   function_body
   RETURN
END
< function_option > ::=
   { ENCRYPTION | SCHEMABINDING }
< table_type_definition > :: =
   ( { column_definition | table_constraint } [ ,...n ] )
```

Скалярные функции в Transact-SQL

Для примера создадим функцию, которая будет возвращать скалярное значение. Например, результат перемножение цены на количество указанного товара. Товар будет идентифицироваться по названию и дате (при условии, что сочетание этих полей дает уникальность).

После оператора **CREATE FUNCTION** мы указываем имя функции. Далее, в скобках идут параметры, которые необходимо передать. Да, параметры должны передаваться через запятую в круглых скобках. В этом объявление отличается от процедур и эту разницу необходимо помнить.

Далее указывается ключевое слово **RETURNS**, за которым идет описание типа возвращаемого значения. Для скалярной функции это могут быть любые типы (строки, числа, даты и т.д.).

Код, который должна выполнять функция пишется между ключевыми словами **BEGIN** (начало) и **END** (конец).

В коде можно использовать любые операторы Transact-SQL, которые мы изучали ранее. Итак, объявление функции в упрощенном виде можно описать следующим образом:

```
CREATE FUNCTION GetSumm (@name varchar(50), @date datetime)
    RETURNS numeric(10,2)
        BEGIN
            -- Код функции
        END
```

Между ключевыми словами **BEGIN** и **END** выполняется следующий код:

```
-- Объявление переменной
DECLARE @Summ numeric(10,2)
-- Выполнение запроса на выборку суммы
    SELECT @Summ = Цена*Количество
    FROM Товары
    WHERE [Название товара]=@name AND Дата=@date;
-- Возврат результата
RETURN @Summ
```

В первой строке объявляется переменная **@Summ**. Она нужна для хранения промежуточного результата расчетов. Далее выполняется запрос **SELECT**, в котором происходит поиск строки по дате и названию товара в таблице товаров. В найденной строке перемножаются поля цены и количества, и результат записывается в переменную **@Summ**.

Обратите внимание, что в конце запроса стоит знак точки с запятой. Каждый запрос должен заканчиваться этим символом, но в большинстве примеров мы этим пренебрегали, но в функции отсутствие символа ";" может привести к ошибке.

В последней строке возвращаем результат. Для этого нужно написать ключевое слово **RETURN**, после которого пишется возвращаемое значение или переменная. В данном случае, возвращаться будет содержимое переменной **@Summ**.

Так как функция скалярная, то и возвращаемое значение должно быть скалярным и при этом соответствовать типу, описанному после ключевого слова **RETURNS**.

Пример кода создание скалярной функции:

```
CREATE FUNCTION GetSumm (@name VARCHAR(50), @date DATETIME)
    RETURNS NUMERIC(10,2)
        BEGIN
        DECLARE @Summ NUMERIC(10,2)
            SELECT @Summ = Цена*Количество
            FROM Товары
            WHERE [Название товара]=@name AND Дата=@date;
        RETURN @Summ
        END
```

Использование функций

Как выполнить такую функцию? Да также, как и многие другие системные функции (например, **GETDATE()**). Например, следующий пример использует функцию в операторе **SELECT**:

```
SELECT dbo.GetSumm('Картофель', '03.03.2015')
```

В этом примере, оператор **SELECT** возвращает результат выполнения функции **GetSumm**. Функция принадлежит пользователю **dbo**, поэтому перед именем функции указано имя владельца. После имени в скобках должны быть перечислены параметры в том же порядке, что и при объявлении функции. В данном примере запрашиваются затраты на картофель, купленный 3.3.2015.

Выполните следующий запрос и убедитесь, что он вернул тот же результат, что и созданная нами функция:

```
SELECT Цена*Количество
FROM Товары
WHERE [Название товара]='Картофель' AND Дата='03.03.2015'
```

Функции можно использовать не только в операторе SELECT, но и напрямую, присваивая значение переменной.

Например:

```
DECLARE @Summ NUMERIC(10,2)
SET @Summ=dbo.GetSumm('Картофель', '03.03.2015')
PRINT @Summ
```

В этом примере мы объявили переменную **@Summ** типа **numeric(10,2)**. Именно такой тип возвращает функция. В следующей строке переменной присваивается результат выполнения **Summ**, с помощью **SET**.

Давайте посмотрим, что произойдет, если передать функции такие параметры, при которых запрос функции вернет более одной строки. В нашей таблице товаров сочетание даты и название не дает уникальности, потому что мы ее нарушили. Первичного ключа в таблице также нет, и среди товаров предположим есть четыре строки, которые имеют свои точные копии. Это нарушает правило уникальности строк в реляционных базах, но очень наглядно показывает, что в реальной жизни нарушать его нельзя.

Итак, предположим, что в таблице зафиксированы две покупки хлеба на дату 1.1.2015. Запросим у созданной функции сумму:

```
SELECT dbo.GetSumm('Хлеб', '01.01.2015')
```

Результатом будет только одно число, хотя строки две. А какую строку из двух вернул сервер? Никто точно сказать не может, потому что они обе одинаковые и без единого различия. Поэтому сервер скорей всего вернул первую из строк.

Функция, возвращающая таблицу

В следующем примере мы создаем функцию, которая будет возвращать в качестве результата таблицу. В качестве примера, создадим функцию, которая будет возвращать таблицу товаров, и для каждой строки рассчитаем произведение колонок количества и цены:

```
CREATE FUNCTION GetPrice()
    RETURNS TABLE
      AS
            RETURN
    (
    SELECT Дата, [Название товара], Цена, Количество,
Цена*Количество AS Сумма
    FROM Товары
    )
```

Начало функции такое же, как у скалярной – указываем оператор **CREATE FUNCTION** и имя функции. Специально функция создана без параметров, чтобы вы увидели, как это делается. Несмотря на то, что параметров нет, после имени должны идти круглые скобки, в которых не надо ничего писать. Если не указать скобок, то сервер вернет ошибку и функция не будет создана.

Разница есть и в секции **RETURNS**, после которой указывается тип **TABLE**, что говорит о необходимости вернуть таблицу. После этого идет ключевое слово **AS** и **RETURN**, после которого должно идти возвращаемое значение. Для функции данного типа в секции **RETURN** нужно в скобках указать запрос, результат которого и будет возвращаться функцией.

Когда пишете запрос, то все его поля должны содержать имена. Если одно из полей не имеет имени, то результатом выполнения оператора **CREATE FUNCTION** будет ошибка. В нашем примере последнее поле является результатом перемножения полей "Цена" и "Количество", а такие поля не имеют имени, поэтому мы его задаем с помощью ключевого слова **AS**.

Посмотрим, как можно использовать такую функцию с помощью оператора **SELECT**:

```
SELECT *
FROM GetPrice()
```

Так как мы используем простой оператор **SELECT**, то мы можем и ограничивать вывод определенными строками, с помощью ограничений в секции **WHERE**. Например, в следующем примере выбираем из результата функции только те строки, в которых поле "Количество" содержит значение 1:

```
SELECT *
FROM GetPrice()
WHERE Количество=1
```

Функция возвращает в качестве результата таблицу, которую вы можете использовать как любую другую таблицу базы данных. Давайте создадим пример, в котором можно будет увидеть использование функции в связи с таблицами. Для начала создадим функцию, которая будет возвращать идентификатор работников таблицы **tbPeoples** и объединенные в одно поле ФИО:

```
CREATE FUNCTION GetPeoples()
    RETURNS TABLE
    AS
      RETURN
      (
      SELECT idPeoples, vcFamil+' '+vcName+' '+vcSurName AS FIO
      FROM tbPeoples
      )
```

Функция возвращает нам идентификатор строки, с помощью которого мы легко можем связать результат с таблицей телефонов. Попробуем сделать это с помощью простого SQL запроса:

```
SELECT *
FROM GetPeoples() p, tbPhoneNumbers pn
WHERE p.idPeoples=pn.idPeoples
```

Как видите, функции, возвращающие таблицы очень удобны. Они больше, чем процедуры похожи на объекты просмотра, но при этом позволяют принимать параметры. Таким образом, можно сделать так, чтобы сама функция возвращала нам только то, что нужно. Вьюшки такого не могут делать по определению. Чтобы получить нужные данные, вьюшка должна выполнить свой **SELECT** запрос, а потом уже во внешнем запросе мы пишем еще один оператор **SELECT**, с помощью которого ограничивается вывод до необходимого. Таким образом, выполняется два запроса **SELECT**, что для большой таблицы достаточно накладно. Функция же может сразу вернуть только то, что нужно.

Рассмотрим пример, функция **GetPeoples** у нас возвращает все строки таблицы. Чтобы получить только нужную фамилию, нужно писать запрос типа:

```
SELECT *
FROM GetPeoples()
WHERE FIO LIKE 'ПОЧЕЧКИН%'
```

В этом случае будут выполняться два запроса: этот и еще один внутри функции. Но если передавать фамилию в качестве параметра в функцию и там сделать секцию WHERE, то можно обойтись и одним запросом SELECT:

```
CREATE FUNCTION GetPeoples1(@Famil VARCHAR(50))
    RETURNS TABLE
    AS
      RETURN
      (
      SELECT idPeoples, vcFamil+' '+vcName+' '+vcSurName AS FIO
      FROM tbPeoples
      WHERE vcFamil=@Famil
      )
```

Многооператорная функция, возвращающая таблицу

Все функции, созданные ранее могут возвращать таблицу, сгенерированную только одним оператором **SQL**. А как же тогда сделать возможность выполнять несколько операций? Например, вы можете захотеть выполнять дополнительные проверки входных параметров для обеспечения безопасности. Проверки лишними не бывают, особенно входных данных и особенно, если эти входные данные указываются пользователем.

Следующий пример показывает, как создать функцию, которая может вернуть в качестве результата таблицу, и при этом, в теле функции могут выполняться несколько операторов:

```
CREATE FUNCTION имя (параметры)
    RETURNS  имя_переменной TABLE
      --(описание вида таблицы, в которой будет представлен результат)
    AS
      BEGIN
      -- Выполнение любого количества операций
      RETURN
      END
```

Это упрощенный вид создания процедуры. Более полный вид мы рассматривали ранее, а сейчас объявление упрощено.

Объявление больше похоже на создание скалярных функций. Первая строка без изменений. В секции **RETURNS** объявляется переменная, которая имеет тип **TABLE**. После этого, в скобках нужно описать поля результирующей таблицы. После ключевого слова **AS** идёт пара операторов **BEGIN** и **END**, между которыми может выполняться какое угодно количество операций. Выполнение операций заканчивается ключевым словом **RETURN**.

Вот тут есть одно отличие от скалярных функций – после **RETURN** мы указывали имя переменной, значение которой должно стать результатом. В данном случае ничего указывать не надо. Мы уже объявили переменную в секции **RETURNS** и описали формат этой переменной. В теле функции мы можем и должны наполнить эту переменную значениями и именно это попадет в результат.

Теперь посмотрим на пример создания функции:

```
CREATE FUNCTION getFIO()
    RETURNS  @ret TABLE
    (idPeoples INT PRIMARY KEY, vcFIO VARCHAR(100))
    AS
      BEGIN
            INSERT @ret
            SELECT idPeoples, vcFamil+' '+vcName+' '+vcSurName
            FROM tbPeoples;
      RETURN
      END
```

В данном примере в качестве результата объявлена переменная **@ret**, которая является таблицей из двух полей **"idPeoples"** типа **int** и **"vcFIO"** типа **varchar** длинной в 50 символов. В теле функции в эту таблицу записываются значения из таблицы **tbPeoples** и выполняется оператор **RETURN**, завершающий выполнение функции.

В использовании, такая функция ничем не отличается от рассмотренных ранее. Например, следующий запрос выбирает все данные, которые возвращает функция:

```
SELECT *
FROM GetFIO()
```

Опции функций

При создании функций могут использоваться следующие опции SCHEMABINDING (привязать к схеме) и/или ENCRYPTION (шифровать текст функции). Если вторая опция нам уже известна по вьюшкам и процедурам (позволяет шифровать исходный код функции в системных таблицах), то вторая встречается впервые, но при этом предоставляет удобное средство защиты данных.

Если функция создана с опцией SCHEMABINDING, то объекты базы данных, на которые ссылается функция, не могут быть изменены (с использованием оператора ALTER) или удалены (с помощью оператора DROP). Например, следующая функция использует таблицу tbPeoples и при этом используется опция SCHEMABINDING:

```
CREATE FUNCTION GetPeoples2(@Famil VARCHAR(50))
    RETURNS TABLE
      WITH SCHEMABINDING
    AS
      RETURN
      (
      SELECT idPeoples, vcFamil+' '+vcName+' '+vcSurName AS FIO
      FROM dbo.tbPeoples
      WHERE vcFamil=@Famil
      )
```

Функция может быть связанной со схемой, только если следующие ограничения истины:

- все функции, объявленные пользователем и просмотрщики на которые ссылается функция, также связаны со схемой с помощью опции **SCHEMABINDING**;
- объекты, на которые ссылается функция, должны использовать имя из двух частей именования: **owner.objectname**. При создании функции **GetPeoples2** ссылка на таблицу указана именно в таком формате – **dbo.tbPeoples**;
- функция и объекты должны быть расположены в одной базе данных;
- пользователь, который создает функцию, имеет право доступа ко всем объектам, на которые ссылается функция.

Создайте функцию и попробуйте после этого удалить таблицу **tbPeoples**.

```
DROP TABLE tbPeoples
```

В ответ на это сервер выдаст сообщение с ошибкой о том, что объект не может быть удален, из-за присутствия внешнего ключа. Даже если избавиться от ключа, удаление будет невозможно, потому что на таблицу ссылается функция, привязанная к схеме.

Чтобы увидеть сообщение без удаления ключа, давайте добавим к таблице колонку, а потом попробуем ее удалить:

```
-- Добавим колонку
ALTER TABLE dbo.tbPeoples
        ADD  vcTemp VARCHAR(30) NOT NULL default ' '


-- Попробуем ее удалить
ALTER TABLE dbo.tbPeoples
        DROP COLUMN vcTemp
```

Создание пройдет успешно, а вот во время удаления произойдет ошибка, с сообщением о том, что существует ограничение, которое зависит от колонки. Мы же не создавали никаких ограничений, а просто добавили колонку и попытались ее удалить. Ограничение уже давно существует, но не на отдельную колонку, а на все колонки таблицы и это ограничение создано функцией **GetPeoples2**, которая связана со схемой.

Изменение функций

Вы можете изменять функцию с помощью оператора **ALTER FUNCTION**. Общий вид для каждого варианта функции отличается. Давайте рассмотрим каждый из них.

Общий вид команды изменения скалярной функции:

```
ALTER FUNCTION [ owner_name. ] function_name
   ( [ { @parameter_name scalar_parameter_data_type [ = default ] } [ ,...n ] ] )

RETURNS scalar_return_data_type
[ WITH < function_option> [,...n] ]
[ AS ]
BEGIN
   function_body
   RETURN scalar_expression
END
```

Общий вид изменения функции, возвращающей таблицу/

```
ALTER FUNCTION [ owner_name. ] function_name
   ( [ { @parameter_name scalar_parameter_data_type [ = default ] } [ ,...n ] ] )
RETURNS TABLE
[ WITH < function_option > [ ,...n ] ]
[ AS ]
RETURN [ ( ] select-stmt [ ) ]
```

Общий вид команды изменения функции с множеством операторов, возвращающей таблицу.

```
ALTER FUNCTION [ owner_name. ] function_name
   ( [ { @parameter_name scalar_parameter_data_type [ = default ] } [ ,...n ] ] )
RETURNS @return_variable TABLE < table_type_definition >
[ WITH < function_option > [ ,...n ] ]
[ AS ]
BEGIN
   function_body
   RETURN
END
< function_option > ::=
   { ENCRYPTION | SCHEMABINDING }
< table_type_definition > :: =
   ( { column_definition | table_constraint } [ ,...n ] )
```

Следующий пример показывает упрощенный вариант команды, изменяющей функцию:

```
ALTER FUNCTION dbo.tbPeoples
          AS
           -- Новое тело функции
```

Удаление функций

Если вы внимательно читали в лекциях об объектах просмотра и функциях, то не трудно догадаться, как можно удалить функцию. Конечно же для этого используется оператор **DROP FUNCTION**:

```
DROP FUNCTION dbo.GetPeoples2
```

Создайте все разновидности изученных функций для вашего варианта задания

Продемонстрируйте Вашу работу преподавателю!

КОНТРОЛЬНЫЕ ВОПРОСЫ:

1. Что называется, пользовательской функцией?
2. Приведите пример синтаксиса создания функции скалярного типа.
3. Приведите пример синтаксиса создания функции, возвращающей таблицу.
4. Приведите пример синтаксиса создания многооператорной функции.