Хранимые функции Transact-SQL

С SQL Server вы можете создавать ваши 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.

1. Создание хранимой функции

Для создания функции используется оператор CREATE FUNCTION. В зависимости от типа, объявление будет отличаться. 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 ] )

2. Скалярные функции в Transact-SQL

Давайте для примера создадим функцию, которая будет возвращать скалярное значение. Например, результат перемножение цены на количество указанного товара. Товар будет идентифицироваться по названию и дате.

Итак, посмотрим сначала на код 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

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 нашей 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 у нас выполняется следующий код:

-- Объявление переменной

DECLARE @Summ numeric(10,2)

-- Выполнение запроса на выборку суммы

SELECT @Summ = Цена\*Количество

FROM Товары

WHERE [эазвание 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.

3. Использование функций

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

SELECT dbo.GetSumm('Картофель', '03.03.2005')

В этом примере, оператор SELECT возвращает результат выполнения функции GetSumm. Функция принадлежит пользователю dbo, поэтому перед именем я указал владельца. После имени в скобках должны быть перечислены параметры в том же порядке, что и при объявлении функции. В данном примере я запрашиваю затраты на картофель, купленный 03.03.2005.

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='03.03.2005'

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', '03.03.2005')

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 у меня 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, в моей таблице есть две покупки хлеба 01.01.2005-го числа. Попробую запросить у функцию сумму:

SELECT dbo.GetSumm('Хлеб', '01.01.2005')

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.

4. 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. Я специально создал эту функцию без параметров, чтобы вы увидели, как это делается. не смотря 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 таблице, 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

)

5. Многооператорная функция возвращающая таблицу

Все функции, созданные в разделе 4 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 бывает, 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.

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 идет 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()