ШАГ
компьютерная
АКАДЕМИЯ

Программирование
и администрирование
СУБД

Microsoft®
SQL Server

Урок №3

Пользовательские функции

Содержание

1. Пользовательские функции

В SQL Server существует большой набор стандартных функций, которыми вы уже не раз пользовались. И этого набора для обеспечения функциональности базы данных иногда может быть недостаточно. Поэтому SQL Server предоставляет возможность создавать свои собственные пользовательские функции (user-defined functions).

Типы пользовательских функций:

1. скалярные (scalar functions) – это функции, которые возвращают одно скалярное значение, то есть число, строка и т. п.
2. встроенные однотаблични или подставляемые табличные (inline table-valued functions) – это функции, которые возвращают результат в виде таблицы. Возвращаются они одним оператором SELECT. Причем, если в результате создается таблица, то имена ее полей являются псевдонимами полей при выборке данных.
3. многооператорные функции (multistatement table-valued functions) – это функции, при определении которой задаются новые имена полей и типы.

Кроме того функции разделяют по детерминизму. Детерминизм функции определяется постоянством ее результатов. Функция является детерминированной (deterministic function), если при одном и том же заданном входном значении она всегда возвращает один

и тот же результат. Например, встроенная функция DATEADD () является детерминированной, поскольку добавление трех дней к дате 5 мая 2010 г. Всегда дает дату 8 мая 2010, или функция COS(), которая возвращает косинус указанного угла.

Функция является недетерминированной (nondeterministic function), если она может возвращать разные значения при одном и том же заданном входном значении. Например, встроенная функция GETDATE() является недетерминированной, поскольку при каждом вызове она возвращает разные значения.

Детерминизм пользовательской функции не зависит от того, является она скалярной или табличной – функции обоих этих типов могут быть как детерминированными, так и недетерминированными.

От детерминированности функции зависит, можно ли проиндексировать ее результат, а также можно ли определить кластеризованный индекс на представление, которое ссылается на эту функцию. Например, недетерминированные функции не могут быть использованы для создания индексов или расчетных полей. Что касается кластеризованного индекса, то он не может быть создан для представления, если оно обращается к недетерминированной функции (независимо от того, используется она в индексе или нет).

Пользовательские функции имеют ряд преимуществ, среди которых основным является повышение производительности выполнения, поскольку функции, как и хранимые процедуры, кэшируют код и повторно используют план выполнения.

Итак, рассмотрим типы функций по порядку. Начнем со скалярных. Синтаксис их объявления следующий:

```
CREATE FUNCTION [ схема. ] имя_функции
               ( [ @параметр [ AS ] [ схема. ] тип
                     [ = значение_по_умолчанию | default ]
                     [ READONLY ]
                 ][,... n]
                )
RETURNS тип_возвращаемого_значения
[ WITH {   [ ENCRYPTION ]
    | [ SCHEMABINDING ]
    | [ RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT ]
    | [ EXECUTE AS контекст_безопасности]
   }
  [ ,...n ]
]
[ AS ]
BEGIN
  тело_функции
  RETURN (возвращаемое_скалярное_значение)
END
```

Как видно из синтаксиса, после имени функции, необходимо в скобках перечислить необходимые входные параметры, если они предусмотрены. Поскольку все параметры являются локальными переменными, то перед именем необходимо ставить символ @, после чего указывается его тип (*допускаются все типы данных, включая типы данных CLR, кроме timestamp* (!)). В случае необходимости можно задать значение по умолчанию для каждого из входных параметров или указать ключевое слово default. Если определено значение default, тогда параметру присваивается значение по умолчанию для данного типа.

Ключевое слово ***READONLY*** указывает на то, что параметр не может быть обновлен или изменен при определении функции. Заметим, что если тип параметра является пользовательским табличным типом, тогда обязательно указать ключевое слово READONLY.

Далее следует указать тип возвращаемого скалярного значения для функции (***returns***). Допускаются все типы данных, кроме нескалярних типов cursor и table, а также типы rowversion (***timestamp***), text, ntext или image. Их лучше заменить типами uniqueidentifier и binary (8).

Параметр WITH задает дополнительные характеристики для входных аргументов. С ключевым словом ENCRIPTION, SCHEMABINDING и EXECUTE AS вы уже знакомы, поэтому на них останавливаться не будем. Хотя здесь есть несколько замечаний:

- параметр SCHEMABINDING нельзя указывать для функций CLR и функций, которые ссилаються на псевдонимы типов данных;
- параметр EXECUTE AS нельзя указывать для встроенных пользовательских функций.

Параметр *RETURNS / CALLED* может быть представлен одним из двух значений:

- CALLED ON NULL INPUT (по умолчанию) означает, что функция выполняется и в случаях, если в качестве аргумента передано значение NULL.
- RETURNS NULL ON NULL INPUT указаный для функций CLR и означает, что функция может вернуть NULL значения, если один из аргументов равен NULL. При этом код самой функции SQL Server не вызывает.

Тело функции размещается в середине блоков BEGIN.. END, который обязательно должен содержать оператор RETURN для возврата результата. При написании кода тела функции следует помнить, что здесь существует ряд ограничений, основным из которых является запрет изменять состояние произвольного объекта базы данных или же базу данных.

Кстати, рекурсивные функции также поддерживаются. Допускается до 32 уровней вложенности.

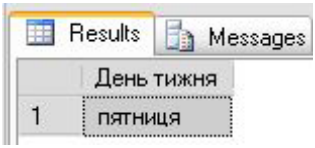
Вызвать скалярную функцию можно одним из двух способов: с помощью оператора ***select*** или ***execute***.

```
SELECT имя_функции (параметр1 [,... n])
EXEC[UTE] @переменная = имя_функции параметр1 [,... n]
```

Напишем функцию, которая возвращает день недели по указанной в качестве параметра дате.

```
create function DayOfWeek (@day datetime)
returns nvarchar(15)
as
begin
  declare @wday nvarchar(15)
  if(datename(dw, @day)='Monday')
     set @wday = 'понедельник'
  if(datename(dw, @day)='Friday')
     set @wday = 'пятница'
  else
     set @wday = 'другой'
  return @wday
end;
-- вызов
select DayOfWeek(GETDATE()) as 'День недели';
```

Результат:



Встроенные табличные функции подчиняются тем же правилам, что и скалярные. Их отличие от последних заключается в том, что они возвращают результат в виде таблицы. Табличные функции являются неплохой альтернативой представлениям и хранимым процедурам. Например, недостатком представлений является то, что они не могут принимать параметры, которые иногда необходимо передать. Хранимые процедуры в свою очередь могут принимать параметры, но не могут быть использованы в выражении *FROM* оператора *SELECT*, что несколько усложняет обработку результатов. Табличные функции решают вышеописанные проблемы.

Итак, синтаксис однотабличной функции выглядит так:

```
CREATE FUNCTION [ схема. ] имя_функции
                ( [ @параметр [ AS ] [ схема. ] тип
                        [ = значение_по_умолчанию | default ]
                        [ READONLY ]
                  ][,... n]
                 )
RETURNS TABLE
[ WITH {   [ ENCRYPTION ]
    | [ SCHEMABINDING ]
    | [ RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT ]
    | [ EXECUTE AS контекст_безопасности]
   }
  [ ,...n ]
]
```

```
[ AS ]
BEGIN
  тело_функции
  RETURN [ ( ] оператор SELECT[ ) ]
END
```

Вызывается табличная функция следующим образом:

```
select * from имя_функции (параметр1 [,... n])
```

Например, напишем функцию, которая выводит названия книг и количество магазинов, которые их продают.

```
create function countShops ()
returns table
as
return (select b.NameBook as 'Book Title',
    count(sh.id) as 'Number of Shops'
    from books b, shops sh, sales s
    where s.id_book = b.id and s.id_shop = sh.id
    group by b.NameBook);
go
-- call the function
select * from countShops();
```

Результат:

Results | Messages

	Book Title	Number Of Shops
1	Applied Microsoft .NET Framework Programming	1
2	ASP.Net: The Complete Reference	1
3	CLR via C#	1
4	Java: A Beginner Guide	1
5	Java: The Complete Reference	2
6	Pro Wpf 4.5 in C#: Windows Presentation Foundati...	1
7	Programming ASP.NET Core (Developer Reference)	1
8	Ring Around the Sun	1
9	Swing: A Beginner Guide	1
10	The Art of Computer Programming, vol.1	1
11	Time is the Simplest Thing	2
12	Windows Runtime via C#	1

Синтаксис многооператорной функции:

```
CREATE FUNCTION [ схема. ] имя_функции
                ( [ @параметр [ AS ] [ схема. ] тип
                        [ = значение_по_умолчанию
| default ]
                        [ READONLY ]
                  ][,... n]
                 )
RETURNS @возвращаемая_таблица
  TABLE структура_таблицы
[ WITH {   [ ENCRYPTION ]
    | [ SCHEMABINDING ]
    | [ RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT ]
    | [ EXECUTE AS контекст_безопасности]
   }
  [ ,...n ]
]
[ AS ]
BEGIN
  тело_функции
  RETURN
END
```

Отметим, что в многооператорной функции возвращаемая таблица не обязательно создается оператором select. Отсюда происходит и название функции. Здесь можно, например, выполнять предварительную обработку данных и создавать временную таблицу, после чего доработать ее и вернуть новую таблицу в вызывающую программу.

Следует также быть внимательным, поскольку блок тела функции может содержать несколько операторов SELECT. В таком случае, в выражении RETURNS нужно явно определить таблицу, которая будет возвращаться. Кроме того, поскольку оператор RETURN

в многооператорной табличной функции всегда возвращает таблицу, которая задана в RETURNS, то он должен выполняться без аргументов. Например, **RETURN**, а не **RETURN @myTable**.

Вызывается многооператорная табличная функция подобно встроенной табличной, то есть с помощью оператора SELECT:

```
select * from имя_функции (параметр1 [,... n])
```

В качестве примера напишем многооператорную табличную функцию, которая возвращает название магазина (-ов), который продал наибольшее количество книг.

Данный процесс можно разделить на **два этапа:**

- **первый** – создадим временную таблицу, которая возвращает название книги и количество магазинов, которые их продают;
- **второй** – получаем магазин, продавший максимальное количество книг.

```
create function bestShops()
returns @tableBestShops table (ShopName varchar(30)
not null,  BookCount int not null)
as
begin
-- creating a temporary table that returns the shop name
-- and number of books in the shop
declare @tmpTable table (id_book int not null,
  bookNumber int not null)
insert @tmpTable
  select b.id as 'Identifier', count(s.id_shop) as 'Number
  Of Shops'
```

```
  from books b, sales s
  where s.id_book=b.id
  group by b.id
insert @tableBestShops
  select sh.Shop as 'Shop Name',
     'Number Of Books' = max(tt.bookNumber)
  from @tmpTable tt, sales s, Shops sh
  where tt.id_book=s.id_book and
     s.id_shop = sh.id
  group by sh.Shop
return
end;
go
-- call the function
select * from bestShops();
```

Результат:

Results | Messages

	ShopName	BookCount
1	HashTag	2
2	Rare Books	2
3	Smith&Brown	2

Изменить существующую функцию пользователя можно с помощью оператора **ALTER FUNCTION**:

```
-- скалярная функция
ALTER FUNCTION [ схема. ] имя_функции
               ( [ @параметр [ AS ] [ схема. ] тип
                      [ = значение_по_умолчанию | default ]
                      [ READONLY ]
                 ][,... n]
```

```
                )
RETURNS тип_возвращаемого_значения
[ WITH {   [ ENCRYPTION ]
    | [ SCHEMABINDING ]
    | [ RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT ]
    | [ EXECUTE AS контекст_безопасности]
   }
  [ ,...n ]
]
[ AS ]
BEGIN
  тело_функции
  RETURN (возвращаемое_скалярное_значение)
END
-- встроенная однотабличная функция
ALTER FUNCTION [ схема. ] имя_функции
               ( [ @параметр [ AS ] [ схема. ] тип
                       [ = значение_по_умолчанию | default ]
                       [ READONLY ]
                 ][,... n]
                )
RETURNS TABLE
[ WITH {   [ ENCRYPTION ]
    | [ SCHEMABINDING ]
    | [ RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT ]
    | [ EXECUTE AS контекст_безопасности]
   }
  [ ,...n ]
]
[ AS ]
BEGIN
  тело_функции
  RETURN [ ( ] оператор SELECT[ ) ]
END
-- многооператорная функция
ALTER FUNCTION [ схема. ] имя_функции
               ( [ @параметр [ AS ] [ схема. ] тип
                       [ = значение_по_умолчанию
```

```
| default ]
                          [ READONLY ]
                    ][,... n]
                   )
RETURNS @возвращаемая_таблица
  TABLE структура_таблицы
[ WITH {   [ ENCRYPTION ]
    | [ SCHEMABINDING ]
    | [ RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT ]
    | [ EXECUTE AS контекст_безопаснгости]
   }
  [ ,...n ]
]
[ AS ]
BEGIN
  тело_функции
  RETURN
END
```

Несмотря на то, что с помощью оператора ALTER FUNCTION можно изменить функцию практически полностью, использовать данный оператор для преобразования скалярной функции в табличную или наоборот запрещено. С помощью ALTER FUNCTION также нельзя превращать функцию T-SQL в функцию среды CLR или наоборот.

Удаление пользовательской функции осуществляется оператором **DROP FUNCTION**:

```
DROP FUNCTION [ схема. ] имя_функции [ ,...n ]
```

Получить список пользовательских функций можно из системного представления sys.sql_modules, а список параметров каждой из них размещается в представлении **sys.parameters**. Список пользовательских функций CLR

размещается по другому адресу, а именно в системном представлении **sys.assembly_modules**.

Урок №3
Пользовательские функции