# Функции T-SQL

SQL Server содержит набор встроенных функций и предоставляет возможность создавать пользовательские функции (User Defined Function – UDF). Различают *детерминированные* и *недетерминированные* функции.

Функция является детерминированной, если при одном и том же заданном входном значении она всегда возвращает один и тот же результат. Например, встроенная функция DATEADD является детерминированной; она возвращает новое значение даты, добавляя интервал к указанной части заданной даты.

Функция является недетерминированной, если она может возвращать различные значения при одном и том же заданном входном значении. Например, встроенная функция GETDATE является недетерминированной; при каждом вызове она возвращает различные значения даты и времени компьютера, на котором запущен экземпляр SQL Server.

Все функции конфигурации, курсора, метаданных, безопасности и системные статистические – недетерминированные. Список этих функций приводится в справочнике. Функции, вызывающие недетерминированные функции и расширенные хранимые процедуры, также считаются недетерминированными.

Пользовательские функции в зависимости от типа данных возвращаемых ими значений могут быть *скалярными* и *табличными*. Табличные пользовательские функции бывают двух типов: *подставляемые* и *многооператорные*.

Скалярные пользовательские функции обычно используются в списке столбцов инструкции SELECT и в предложении WHERE. Табличные пользовательские функции обычно используются в предложении FROM, и их можно соединять с другими таблицами и представлениями.

**Создание и вызов скалярной функции**

Для создания скалярной функции используется инструкция CREATE FUNCTION, имеющая следующий синтаксис:

CREATE FUNCTION [ *имя-схемы*. ] *имя-функции* ( [ *список-объявлений-параметров* ] )
RETURNS *скалярный-тип-данных*
[ WITH *список-опций-функций* ]
[ AS ]
BEGIN
*тело-функции*
RETURN *скалярное-выражение*
END [ ; ]

где

a) Список объявлений параметров является необязательным, но наличие скобок обязательно.
b) Объявление параметра в списке объявлений параметров имеет вид:

*@имя-параметра* [ AS ] [ *имя-схемы*. ] *тип-данных* [ = *значение-по-умолчанию* ] [ READONLY ]

c) Тип данных параметра – любой тип данных, включая определяемые пользователем типы данных CLR и определяемые пользователем табличные типы, за исключением скалярного типа **timestamp** и нескалярных типов **cursor** и **table**.
d) Значение по умолчанию -
e) Необязательное ключевое слово 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.
f) Возвращаемое значение может быть любого скалярного типа данных, включая определяемые пользователем типы данных CLR, за исключением типа данных **timestamp**.
g) В качестве опций функции используются:
ENCRIPTION — SQL Server шифрует определение функции.
SCHEMABINDING - привязывает функцию к схеме базовой таблицы или таблиц. Если аргумент SCHEMABINDING указан, нельзя изменить базовую таблицу или таблицы таким способом, который может повлиять на определение функции.
RETURNS NULL ON NULL INPUT или| CALLED ON NULL INPUT - ...

Предложение EXECUTE AS - указывает контекст безопасности, в котором может быть выполнена функция. Например, EXECUTE AS CALLER указывает, что функция будет выполнена в контексте пользователя, который ее вызывает. Также могут быть указаны параметры SELF, OWNER и *имя-пользователя*.

h) Операторы BEGIN и END, которыми ограничивается тело функции, являются обязательными.
i) Тело функции представляет собой ряд инструкций T-SQL, которые в совокупности вычисляют скалярное выражение.

Синтаксис вызова скалярных функций схож с синтаксисом, используемым для встроенных функций T-SQL:

*имя-схемы.имя-функции* ( [ *список-параметров* ] )

где
a) Имя схемы (имя владельца) для скалярной функции является обязательным.
b) Нельзя использовать синтаксис с именованными параметрами (*@имя-параметра = значение*).
c) Нельзя не указывать (опускать) параметры, но можно применять ключевое слово DEFAULT для указания значения по умолчанию.

Для скалярной функции можно использовать инструкцию EXECUTE:

EXECUTE *@возвращаемое-значение = имя-функции* ( [ *список-параметров* ] )

где
a) Не нужно указывать имя схемы (имя владельца).
b) Можно использовать именованные параметры (*@имя-параметра = значение*).
c) Если используются именованные параметры, они не обязательно должны следовать в том порядке, в котором указаны в объявлении функции, но необходимо указать все параметры; нельзя опускать ссылку на параметр для использования значения по умолчанию.

**Примеры создания и вызова скалярных функций**

```
USE dbSPJ
GO
IF OBJECT_ID (N'dbo.AveragePrice', N'FN') IS NOT NULL
    DROP FUNCTION dbo.AveragePrice
GO
CREATE FUNCTION dbo.AveragePrice()
RETURNS smallmoney
WITH SCHEMABINDING
AS
BEGIN
      RETURN (SELECT AVG(Price) FROM dbo.R)
END
GO
IF OBJECT_ID (N'dbo.PriceDifference', N'FN') IS NOT NULL
    DROP FUNCTION dbo.PriceDifference
GO
CREATE FUNCTION dbo.PriceDifference(@Price smallmoney)
RETURNS smallmoney
AS
BEGIN
      RETURN @Price - dbo.AveragePrice()
END
GO
SELECT Pname, Price, dbo.AveragePrice() AS Average, dbo.PriceDifference(Price) AS Difference
FROM R
WHERE City='Смоленск'
GO
```

**Создание и вызов подставляемой табличной функции**

Синтаксис создания подставляемой табличной функции выглядит так:

CREATE FUNCTION [ *имя-схемы*. ] *имя-функции* ( [ *список-объявлений-параметров* ] )
RETURNS TABLE
[ WITH *список-опций-функций* ]
[ AS ]
        RETURN [ ( ] *выражение-выборки* [ ) ]
END [ ; ]

Тело подставляемой табличной функции фактически состоит из единственной инструкции SELECT.

**Пример создания и вызова подставляемой табличной функции**

```
USE dbSPJ
GO
IF OBJECT_ID (N'dbo.FullSPJ', N'FN') IS NOT NULL
    DROP FUNCTION dbo.FullSPJ
GO
CREATE FUNCTION dbo.FullSPJ()
RETURNS TABLE
AS
RETURN (SELECT S.Sname, P.Pname, J.Jname, SPJ.Qty
        FROM S
                    INNER JOIN SPJ ON S.Sno=SPJ.Sno
                    INNER JOIN P ON P.Pno=SPJ.Pno
                    INNER JOIN J ON J.Jno=SPJ.Jno)
GO
SELECT *
FROM dbo.FullSPJ()
GO
```

**Создание и вызов многооператорной табличной функции**

Синтаксис создания многооператорной табличной функции выглядит так:

CREATE FUNCTION [ *имя-схемы*. ] *имя-функции* ( [ *список-объявлений-параметров* ] )
RETURNS *@имя-возвращаемой-переменной* TABLE *определение-таблицы*
[ WITH *список-опций-функций* ]
[ AS ]
BEGIN
        *тело-функции*
        RETURN
END [ ; ]

Подобно скалярным функциям, в многооператорной табличной функции команды T-SQL располагаются внутри блока BEGIN-END. Поскольку блок может содержать несколько инструкций SELECT, в предложении RETURNS необходимо явно определить таблицу, которая будет возвращаться. Поскольку оператор RETURN в многооператорной табличной функции всегда возвращает таблицу, заданную в предложении RETURNS, он должен выполняться без аргументов.

**Пример создания и вызова многооператорной табличной функции**

```
USE Northwind
GO
CREATE FUNCTION dbo.fnGetReports ( @EmployeeID AS int )
RETURNS @Reports TABLE
       (
            EmployeeID    int   NOT NULL,
            ReportsToID   int    NULL
       )
```

```
AS
BEGIN

/* Объявляем переменную @Employee для хранения ID текущего служащего (чтобы случайно не
включить его дважды). */

DECLARE @Employee AS int

/* Добавляем текущего служащего и его начальника в рабочую таблицу @Reports. Суть в том, что
необходима некоторая начальная строка  из-за рекурсивной природы функции. */

INSERT INTO @Reports
   SELECT EmployeeID, ReportsTo
   FROM Employees
   WHERE EmployeeID = @EmployeeID

/* Подготавливаемся к выполнению рекурсивного вызова функции. */

SELECT @Employee = MIN(EmployeeID)
FROM Employees
WHERE ReportsTo = @EmployeeID

/* Рекурсивный вызов функции! */

WHILE @Employee IS NOT NULL
   BEGIN
      INSERT INTO @Reports
           SELECT *
           FROM fnGetReports(@Employee)

           SELECT @Employee = MIN(EmployeeID)
           FROM Employees
           WHERE EmployeeID > @Employee
              AND ReportsTo = @EmployeeID
   END
RETURN
END
GO
```

Для исходных данных

```
1      Davolio      Nancy        2
2      Fuller       Andrew       NULL
3      Leverling    Janet        2
4      Peacock      Margaret     2
5      Buchanan     Steven       2
6      Suyama       Michael      5
7      King         Robert       5
8      Callahan     Laura        2
9      Dodsworth    Anne         5
```

вызов функции

```
SELECT * FROM dbo.fnGetReports(5)
GO
```

даст следующие результаты

```
5      2
```

```
6       5
7       5
9       5
```

Какие результаты будут получены при таком вызове функции?

```
DECLARE @EmployeeID int

SELECT @EmployeeID = EmployeeID
    FROM Employees
    WHERE LastName = 'Fuller' AND FirstName = 'Andrew'

SELECT Emp.EmployeeID, Emp.LastName, Emp.FirstName, Mgr.LastName AS ReportsTo
FROM Employees AS Emp
       JOIN dbo.fnGetReports(@EmployeeID) AS gr ON gr.EmployeeID = Emp.EmployeeID
       JOIN Employees AS Mgr ON Mgr.EmployeeID = gr.ReportsToID
```