Creating and Managing User Defined Functions in SQL Server 2008

ASK A QUESTION

Vishal Nayan Sep 02 2011 Article

3 3 69.6k

What are UDF:

SQL server provides list of many predefined functions that are built in to the T-SQL language. The supplied functions helps extend the capabilities of T-SQL, providing the ability to perform string manipulation, mathematical calculations, data type conversion etc. but often we need something which is not provided using these functions. So we can create stored procedure to perform custom processing, but the problem is that we can't use the result of stored procedure in WHERE or SELECT list, for this type of scenario we need UDF.

Why to use User Defined Functions:

The main benefit of UDF is that we are not just limited to SQL provided functions. We can write our own functions to meet our specific needs or to simplify complex SQL codes.

Let's take an example:

SQL getdate() returns current system date and time. It always includes both data and time components. We want to get just date and have the time always set to midnight. One solution is to to the conversion like below;

```
select convert(datetime,CONVERT(date,getdate()))
```

But the problem is that when we want to have date with time always set to midnight, we need to do this conversion. Solution is to make a UDF for this.

```
create function getonlydate()
returns datetime
as
begin
    return(select convert(datetime,convert(date,getdate())
end
go
```

Don't miss Vishal Nayan's next article

Vishal Nayan

278 4.9k 2m

+ Follow

```
select dbo.getonlydate()
```

Let us see how we can use this UDF in other SQL statements.

Let us create a table Order

```
CREATE TABLE Orders (
OrderID int IDENTITY (1, 1) NOT NULL Primary Key,
CustomerID nchar (5) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
EmployeeID int NULL ,
OrderDate datetime NULL default dbo.getonlydate(),
RequiredDate datetime NULL ,
ShippedDate datetime NULL
)
```

Let us INSERT values in this table using the UDF function we created.

```
INSERT Orders(CustomerID,EmployeeID,RequiredDate)
values('BERGS',3,dbo.getonlydate() +7)
```

Let us UPDATE values in this table using the UDF function we created.

```
UPDATE Orders set ShippedDate = dbo.getonlydate()
      where OrderID=1
SELECT OrderDate,RequiredDate,ShippedDate
FROM orders
WHERE orderdate = dbo.getonlydate()
```

```
Orderdate               Requireddate            Shippeddate

---------------------------------------------------

2011-05-01 00:00:00.000     2011-05-08 00:00:00.000     2011-05-01 00:00:00.000
```

Types of User Defined Functions:

1. Scalar functions
2. Inline table valued function
3. Multistatement table valued functions.

For all examples shared below I have used Pubs database. Y attach .mdf file in your SQL Sever 2008.

http://www.microsoft.com/downloads/en/details.aspx?FamilyId=06616212-0356-46A0-8DA2-EEBC53A68034&displaylang=en

1. They are like standard built in functions provided
2. They return scalar values that can be used anywhere a constant expression can be used.
3. They typically take one or more arguments and returns a value of a specified data types.
4. Every T-SQL function must return a result using the RETURN statement.

Example:

The following two functions are variations of a function that returns the average price for a specified type of book from the titles table:

```
CREATE FUNCTION AverageBookPrice (@booktype varchar(12) = '%')
RETURNS money
AS
BEGIN
DECLARE @Avg money
     SELECT @Avg = AVG(price)
     FROM titles
     WHERE type  like @booktype
RETURN @Avg
END
GO

CREATE FUNCTION   AverageBookPrice2 (@booktype varchar(12) ='%')
RETURNS money
AS
BEGIN
RETURN (SELECT AVG(PRICE)
           FROM TITLES
           WHERE TYPE LIKE @booktype)

END
```

SQL Server doesn't allow aggregate functions in a WHERE clause unless they are contained in a subquery.

The AvgBookPrice() function lets you compare against the av

```
SELECT  title_id, type, price from titles
where price > dbo.AverageBookPrice('popular_comp')
```

```
titleid type price
-----------------------
PC1035 popular_comp 22.95
PS1372 psychology 21.59
```

EXEC statement. The following commands are functionally equivalent:

```
declare @avg1 money,
@avg2 money,
@avg3 money
select @avg1 = dbo.AverageBookPrice('popular_comp')
set @avg2 = dbo.AverageBookPrice('popular_comp')
exec @avg3 = dbo.AverageBookPrice 'popular_comp'

select @avg1 as avg1, @avg2 as avg2, @avg3 as avg3
go
```

```
Result is below
avg1 avg2 avg3
----------------------------------
21.475 21.475 21.475
```

B) Table Value Function:

- A table-valued user-defined function returns a rowset instead of a single scalar value.
- Can be invoked in the FROM clause of a SELECT statement, just as we would a table or view.
- A table-valued function can almost be thought of as a view that accepts parameters, so the result set is determined dynamically.
- A table valued function specifies the keyword TABLE in its RETURNS clause.
- They are of two types.

1) Inline table valued function

A) An inline table-valued function specifies only the TABLE keyword in the RETURNS clause, Without table definition information.

B) The code inside the function is a single RETURN statement that invokes a SELECT statement.

Example:

```
CREATEFUNCTION AveragePriceByType(@price money = 0.0)
RETURNStable
AS
RETURN (SELECTtype,avg(isnull(price,0))as avg_pric
        FROM titles
        GROUP BYtype
        HAVINGavg(isnull(price,0))> @price )

select* from AveragePriceByType(15.0)
```

```
type averageprice
```

2) Multi statement table valued function:

a) Multistatement table-valued functions differ from inline functions in two major ways

-> The RETURNS clause specifies a table variable and its definition.
-> The body of the function contains multiple statements, at least one of which populates the table variable with data values.

b) The scope of the table variable is limited to the function in which it is defined.

c) Within the function in which a table variable is defined, that table variable can be treated like a regular table. You can perform any SELECT, INSERT, UPDATE, or DELETE statement on the rows in a table variable, except for SELECT INTO.

The following example defines the inline table-valued function AveragePricebyType() as a multistatement table-valued function called AveragePricebyType3():

```
CREATE FUNCTION  AveragePricebyType3 (@price money =0.0)
RETURNS @table table(type varchar(12) null,avg_price money null)
AS
BEGIN
    INSERT @table
    SELECT type,avg(isnull(price,0)) as avg_price
    FROM titles
    GROUP BY type
    HAVING avg(isnull(price,0))> @price

RETURN
END
SELECT * FROM AveragePricebyType3(15.0), this also gives same result.
```

```
type averageprice
--------------------------------------
trad_cook 15.9633
-----------------------------------------------------------
```

Big Question: Why use multi-statement table-valued fu functions?

1. Generally, we use multi-statement table-valued functic operations (for example, inserts, updates, or deletes) (returning a result set.

2. We would also use them if we need to perform more c input parameters of the function before invoking the query to populate the table variable.

Types of SQL statements allowed in a function include the following:

- Assignments of values to variables that are local to the function, using the SET command or an assignment select.

- Cursor operations on local cursors that are declared, opened, closed, and de-allocated within the function. FETCH statements must assign values to local variables by using the INTO clause.

- Control-of-flow statements such as IF, ELSE, WHILE, GOTO, and so on, excluding the TRY...CATCH statements.

- UPDATE, INSERT, and DELETE statements that modify table variables defined within the function.

- EXECUTE statements that call an extended stored procedure. (Any results returned by the extended stored procedure are discarded.)

Nesting of User Defined Function: User-defined functions can also call other user-defined functions, with a limit of 32 levels of nesting. Nesting of functions can help improve the modularity and reusability of function code.

```
CREATE FUNCTION dbo.getonlydate3()
RETURNS datetime
as
BEGIN
DECLARE @date datetime
SET @date = dbo.striptime( getdate())
RETURN @date
End
```

How to get information about Functions:

To get information by using the provided system procedures and queries against the INFORMATION_SCHEMA.routines view. The following sections describe these methods.

```
execsp_helptextgetonlydate
```

```
Text
--------------------------------------------------------------------------

create function getonlydate()
returns datetime
as
begin
return(select convert(datetime,convert(date,getdate())))
end
```

In addition to sp_helptext, you can write queries against the display the source code for a function:

```
SELECT routine_definition
from INFORMATION_SCHEMA.routines
where routine_name = 'getonlydate'
and specific_schema = 'dbo'
```

ASK A QUESTION

Conclusion:

User-defined functions in SQL Server 2008 allow you to create reusable routines that can help make your SQL code more straightforward and efficient. Table-valued functions provide a way to create what are essentially parameterized views, and you can include them inline in your queries, just as you would in a table or view.

Hope you enjoyed reading

Cheers.

Functions | Functions in SQL Server 2008 | SQL Server 2008 | UDF | User Defined Functions

Vishal Nayan TOP 500

Vishal is a seasoned professional with hand on experience on Microsoft technologies. He is also a part-time trainer on framework , WPF ,WCF , Silverlight and is active in Mauritius.

http://www.vishalnayan.wordpress.com

278 2m

3 3

Type your comment here and press Enter Key (Minimum 18 characters)

×

Can we give proper example as create new table and create function for that table. It will help freshers like me who want to learn from basis. Waiting fro your quick revert

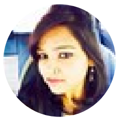
Bhavini Kothari Jun 21, 2017

1458 2 0 0 0 Reply

Nice explanation

Subash Sep 16, 2016

267 5.1k 16.7k



nice

Sonu Chaudhary

562 1.6k 87.4k

Comment Using

Don't miss Vishal Nayan's next article

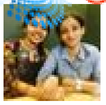
Vishal Nayan

278 4.9k 2m

+ Follow

C#Corner

ASK A QUESTION



Add a comment...

TRENDING UP

01 Making Web Sites Look Like Native Apps Without the App Store

02 Creating Web API With ASP.NET Core Using Visual Studio Code

03 Future Of IoT, Machine Learning, And Artificial Intelligence

04 CRUD Operation Using React, Nodejs, Express, MongoDb

05 Angular, BootStrap And ASP.NET MVC - Part One

06 Progressive Web App - The Latest Technology Disruption Developers Must Master Today

07 How To Do Paging With ASP.NET Web API

08 Angular, Bootstrap And ASP.NET MVC - Part Two

09 Angular, Bootstrap And ASP.NET MVC - Part Seven (Bootstrap Modal Forms)

10 Getting Started With Git Version Control

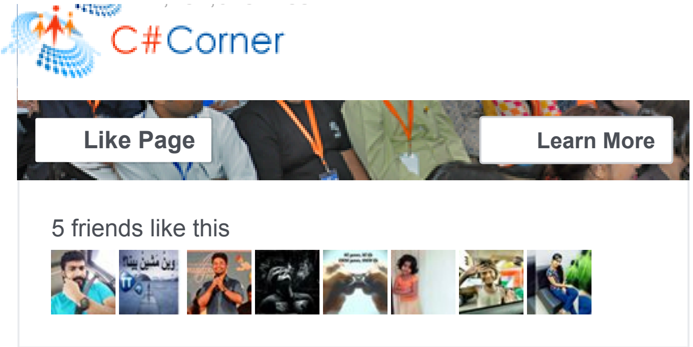
View All ○

Don't miss Vishal Nayan's next article

Vishal Nayan

278 4.9k 2m

+ Follow



C#Corner
Like Page
Learn More
5 friends like this