User Defined Functions

By **Doug Carpenter** on **12 October 2000** | 6 Comments | Tags: User Defined Functions, Functions

This article covers all the basics of User Defined Functions. It discusses how (and why) to create them and when to use them. It talks about scalar, inline table-valued and multi-statement table-valued functions. (**This article has been updated through SQL Server 2005.**)

With SQL Server 2000, Microsoft has introduced the concept of User-Defined Functions that allow you to define your own T-SQL functions that can accept zero or more parameters and return a single scalar data value or a table data type.

What Kind of User-Defined Functions can I Create?

There are three types of User-Defined functions in SQL Server 2000 and they are Scalar, Inline Table-Valued and Multi-statement Table-valued.

How do I create and use a Scalar User-Defined Function?

A Scalar user-defined function returns one of the scalar data types. Text, ntext, image and timestamp data types are not supported. These are the type of user-defined functions that most developers are used to in other programming languages. You pass in 0 to many parameters and you get a return value. Below is an example that is based in the data found in the NorthWind Customers Table.

```
CREATE FUNCTION whichContinent
(@Country nvarchar(15))
RETURNS varchar(30)
AS
BEGIN
declare @Return varchar(30)
select @return = case @Country
when 'Argentina' then 'South America'
when 'Belgium' then 'Europe'
when 'Brazil' then 'South America'
when 'Canada' then 'North America'
when 'Denmark' then 'Europe'
when 'Finland' then 'Europe'
when 'France' then 'Europe'
else 'Unknown'
end

return @return
end
```

Because this function returns a scalar value of a varchar(30) this function could be used anywhere a varchar(30) expression is allowed such as a computed column in a table, view, a T-SQL select list item. Below are some of the examples that I was able to use after creating the above function definition. Note that I had to reference the dbo in the function name.

```
print dbo.WhichContinent('USA')

select dbo.WhichContinent(Customers.Country), customers.*
from customers
```

```
create table test
(Country varchar(15),
Continent as (dbo.WhichContinent(Country)))

insert into test (country)
values ('USA')

select * from test

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

Country          Continent
---------------  ------------------------------
USA              North America
```

Stored procedures have long given us the ability to pass parameters and get a value back, but the ability to use it in such a variety of different places where you cannot use a stored procedure make this a very powerful database object. Also notice the logic of my function is not exactly brain surgery. But it does encapsulate the business rules for the different continents in one location in my application. If you were to build this logic into T-SQL statements scattered throughout your application and you suddenly noticed that you forgot a country (like I missed Austria!) you would have to make the change in every T-SQL statement where you had used that logic. Now, with the SQL Server User-Defined Function, you can quickly maintain this logic in just one place.

How do I create and use an Inline Table-Value User-Defined Function?

An Inline Table-Value user-defined function returns a table data type and is an exceptional alternative to a view as the user-defined function can pass parameters into a T-SQL select command and in essence provide us with a parameterized, non-updateable view of the underlying tables.

```
CREATE FUNCTION CustomersByContinent
(@Continent varchar(30))
RETURNS TABLE
AS
RETURN
  SELECT dbo.WhichContinent(Customers.Country) as continent,
         customers.*
  FROM customers
  WHERE dbo.WhichContinent(Customers.Country) = @Continent
GO

SELECT * from CustomersbyContinent('North America')
SELECT * from CustomersByContinent('South America')
SELECT * from customersbyContinent('Unknown')
```

Note that the example uses another function (WhichContinent) to select out the customers specified by the parameter of this function. After creating the user-defined function, I can use it in the FROM clause of a T-SQL command unlike the behavior found when using a stored procedure which can also return record sets. Also note that I do not have to reference the dbo in my reference to this function. However, when using SQL Server built-in functions that return a table, you must now add the prefix :: to the name of the function.

Example from Books Online: `Select * from ::fn_helpcollations()`

How do I create and use a Multi-statement Table-Value User-Defined Function?

A Multi-Statement Table-Value user-defined function returns a table and is also an exceptional alternative to a view as the function can support multiple T-SQL statements to build the final result where the view is limited to a single SELECT statement. Also, the ability to pass parameters into a T-SQL select command or a group of them gives us the capability to in essence create a parameterized, non-updateable view of the data in the underlying tables. Within the create function command you must define the table structure that is being returned. After creating this type of user-defined function, I can use it in the FROM clause of a T-SQL command unlike the behavior found when using a stored procedure which can also

return record sets.

```
CREATE FUNCTION dbo.customersbycountry ( @Country varchar(15) )
RETURNS
        @CustomersbyCountryTab table (
                [CustomerID] [nchar] (5), [CompanyName] [nvarchar] (40),
                [ContactName] [nvarchar] (30), [ContactTitle] [nvarchar] (30),
                [Address] [nvarchar] (60), [City] [nvarchar] (15),
                [PostalCode] [nvarchar] (10), [Country] [nvarchar] (15),
                [Phone] [nvarchar] (24), [Fax] [nvarchar] (24)
        )
AS
BEGIN
        INSERT INTO @CustomersByCountryTab
        SELECT  [CustomerID],
                        [CompanyName],
                        [ContactName],
                        [ContactTitle],
                        [Address],
                        [City],
                        [PostalCode],
                        [Country],
                        [Phone],
                        [Fax]
        FROM [Northwind].[dbo].[Customers]
        WHERE country = @Country

        DECLARE @cnt INT
        SELECT @cnt = COUNT(*) FROM @customersbyCountryTab

        IF @cnt = 0
                INSERT INTO @CustomersByCountryTab (
                        [CustomerID],
                        [CompanyName],
                        [ContactName],
                        [ContactTitle],
                        [Address],
                        [City],
                        [PostalCode],
                        [Country],
                        [Phone],
                        [Fax]  )
                VALUES ('','No Companies Found','','','','','','','')

        RETURN
END
GO
SELECT * FROM dbo.customersbycountry('USA')
SELECT * FROM dbo.customersbycountry('CANADA')
SELECT * FROM dbo.customersbycountry('ADF')
```

What are the benefits of User-Defined Functions?

The benefits to SQL Server User-Defined functions are numerous. First, we can use these functions in so many different places when compared to the SQL Server stored procedure. The ability for a function to act like a table (for Inline table and Multi-statement table functions) gives developers the ability to break out complex logic into shorter and shorter code blocks. This will generally give the additional benefit of making the code less complex and easier to write and maintain. In the case of a Scalar User-Defined Function, the ability to use this function anywhere you can use a scalar of the same data type is also a very powerful thing. Combining these advantages with the ability to pass parameters into these database objects makes the SQL Server User-Defined function a very powerful tool.

Summary

So, if you have ever wanted to use the results of a stored procedure as part of a T-SQL command, use parameterized non-updateable views, or encapsulate complex logic into a single database object, the SQL Server 2000 User-Defined function is a new database object

that you should examine to see if its right for your particular environment.

Discuss this article: **6 Comments** so far. Print this Article.

If you like this article you can sign up for our **weekly newsletter**. There's an opt-out link at the bottom of each newsletter so it's easy to unsubscribe at any time.

Email Address: Subscribe

Related Articles

Using REPLACE in an UPDATE statement (31 March 2010)

Returning Complex Data from User-Defined Functions with CROSS APPLY (11 June 2007)

SQL Server 2005: Using OVER() with Aggregate Functions (21 May 2007)

DATEDIFF Function Demystified (20 March 2007)

Using CROSS APPLY in SQL Server 2005 (4 May 2005)

Using the PARSENAME function to split delimited data (10 November 2003)

Using a CSV with an IN sub-select (13 October 2002)

Simulating Constants Using User Defined Functions (24 February 2002)

Other Recent Forum Posts

Replace '' with characters (5 Replies)

Make Column name Date (2 Replies)

Delete Files from directory (1 Reply)

Efficient Singular / Plural child records syntax (3 Replies)

Transaction log restore (2 Replies)

Where Datetime just need time (1 Reply)

help with query - last day of the month (2 Replies)

Cannot connect to server in sql server 2008 (2 Replies)