User Defined Functions in T-SQL

Presented by: Verawaty Situmorang

Stored Procedure (SP) vs. User Define Function (UDF)

- A function is a subprogram written to perform certain computations
- A scalar function returns only a single value (or NULL), whereas a table function returns a (relational) table comprising zero or more rows, each row with one or more columns.
- Functions must return a value (using the RETURN keyword), but for stored procedures this is not compulsory.
- Stored procedures can use RETURN keyword but without any value being passed.

Stored Procedure (SP) vs. User Define Function (UDF)

- **Functions could be used in SELECT statements, provided they don't do any data manipulation. However, procedures cannot be included in SELECT statements.**
- **A function can have only IN parameters, while stored procedures may have OUT or INOUT parameters.**
- **A stored procedure can return multiple values using the OUT parameter or return no value at all.**

User Defined Function

- **UDF:**
  - a body of T-SQL statements
  - pre-compiled and pre-optimized
  - works as a single unit
  - can perform in-line to a query
- **Two types:**
  - those that return a scalar value
  - those that return a table

So..When the developers use UDF rather than SP??

When??

- The ability for a user-defined function to act like a table gives developers the ability to break out complex logic into shorter code blocks. This will generally provides the additional benefit of making the code less complex, and easier to write and maintain.
- If you want to be able to invoke a stored procedure directly from within a query, then rewriting a stored procedure as a user-defined function would be worthwhile.

UDF Returning a Scalar

```
CREATE FUNCTION DayOnly (@date DATETIME)
  RETURNS varchar(10)
AS
BEGIN
  RETURN CONVERT(VARCHAR(10), @date, 101)
END
```

```
SELECT dbo.DayOnly(GETDATE()) AS Today

Results:
----------
03/15/2010
```

Scalar udf must be deterministic

- **Must return the same value for the same input parameters**

```
CREATE FUNCTION fnRandomInt(@max INT)
RETURNS INT
AS
BEGIN
     RETURN CEILING(@max*RAND())
END
```

```
Msg 443
Invalid use of a side-effecting
operator 'rand' within a function.
```

Usage example

```
CREATE TABLE test (id          INT              IDENTITY PRIMARY KEY,
                   testDate    DATETIME         NOT NULL)
GO

-- populate table test:
DECLARE @count INT
SET @count = 1
WHILE @count <= 10
BEGIN
        INSERT test VALUES (DATEADD(MINUTE, @count, GETDATE()))
        SET @count = @count + 1
END
GO

-- no results:
SELECT * FROM test WHERE testDate = GETDATE()
GO

-- this works:
SELECT * FROM test WHERE dbo.DayOnly(testDate) = dbo.DayOnly(GETDATE())
```

Usage example

```
CREATE FUNCTION CubicVolume
-- Input dimensions in centimeters
(
@CubeHeight decimal(4,1),
@CubeLength decimal(4,1),
@CubeWidth decimal(4,1)
)
RETURNS decimal(12,3) -- Cubic Centimeters.
AS
BEGIN
    RETURN ( @CubeLength * @CubeWidth * @CubeHeight )
END
```

```
CREATE TABLE Bricks
   (
    BrickPartNmbr   int PRIMARY KEY,
    BrickColor      nchar(20),
    BrickHeight     decimal(4,1),
    BrickLength     decimal(4,1),
    BrickWidth      decimal(4,1),
    BrickVolume AS
              (
               dbo.CubicVolume(BrickHeight,
                        BrickLength, BrickWidth)
              )
   )
```

UDF Returning a Table

```
USE MovieRental          -- See Davidson, chapter 5
GO

CREATE FUNCTION dbo.fnMovieList()
  RETURNS TABLE
AS
  RETURN (SELECT
               m.Name,
               CONVERT(VARCHAR(10),ReleaseDate, 101) AS [Release Date],
               g.Name AS Genre
           FROM Inventory.Movie m
           INNER JOIN Inventory.Genre g
               ON m.GenreId = g.GenreId)
GO

SELECT * FROM dbo.fnMovieList()
```

UDF with parameter Returning a Table

```
USE MovieRental          -- See Davidson, chapter 5
GO


CREATE FUNCTION dbo.fnMovieSearch(@MovieNamePart NVARCHAR(50))
  RETURNS TABLE
AS
  RETURN (SELECT
               m.Name,
               CONVERT(VARCHAR(10), ReleaseDate, 101) AS [Release Date],
               g.Name AS Genre
           FROM Inventory.Movie m INNER JOIN Inventory.Genre g
                       ON m.GenreId = g.GenreId
           WHERE m.Name LIKE ('%' + @MovieNamePart + '%'))
GO



SELECT * FROM dbo.fnMovieSearch('Maltese')
```

Returning a Table (3)

```
USE MovieRental
GO
CREATE FUNCTION fnOutstandingRentals (@CustID INT)
  RETURNS @OutstandingRentalsTable
                TABLE (
                       CustomerId     INT             NOT NULL,
                       MovieRentalID  INT             NOT NULL,
                       DueReturnDate  SMALLDATETIME   NOT NULL
                      )
AS
BEGIN
  INSERT @OutstandingRentalsTable
                (CustomerId, MovieRentalID, DueReturnDate)
    SELECT CustomerId, MovieRentalID, ReturnDate
    FROM   Rentals.MovieRental
    WHERE  CustomerId = @CustID

  RETURN
END
GO
SELECT * FROM dbo.fnOutstandingRentals(0)
```