Database System

User Defined Functions

Muhammad Tariq Mahmood
tariq@koreatech.ac.kr
School of Computer Science and Engineering
Korea University of Technology and Education

User-Defined Functions

- User-defined functions
  - Like functions in programming languages, SQL Server user-defined functions are routines that accept parameters, perform an action, such as a complex calculation, and return the result of that action as a value.
  - The return value can either be a single scalar value or a result set.

- User-defined functions
  - allow modular programming.
  - allow faster execution
  - can reduce network traffic

Valid statements in a function

- The types of statements that are valid in a function include:
  - DECLARE statements
  - Assignments of values
  - Cursor operations
  - Control-of-flow statements
  - SELECT statements.
  - UPDATE, INSERT, and DELETE statements
  - EXECUTE statements

Types of User-defined functions

- Scalar Function
  - User-defined scalar functions return a single data value of the type defined in the RETURNS clause.
  - For an inline scalar function, there is no function body; the scalar value is the result of a single statement.
  - For a multistatement scalar function, the function body, defined in a BEGIN...END block, contains a series of Transact-SQL statements that return the single value.
- Table-Valued Functions
  - User-defined table-valued functions return a **table** data type.
  - For an inline table-valued function, there is no function body; the table is the result set of a single SELECT statement.

User-Defined Functions
Scalar Functions

- The CREATE FUNCTION statement allows you to create custom scalar functions that behave like the built-in scalar functions.

- Syntax

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type
    [ = default ] [ READONLY ] }
    [ ,...n ]
  ]
)
RETURNS return_data_type
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN scalar_expression
    END
[ ; ]
```

User-Defined Functions
Scalar Functions

- Example-1

```
create function dbo.CalculateArea(@radius as float)
returns float
as
begin

  return PI()* power(@radius,2);

end;
```

Scalar Functions

Example-2

```
IF OBJECT_ID('dbo.GetAge') IS NOT NULL DROP FUNCTION
dbo.GetAge;
GO

CREATE FUNCTION dbo.GetAge(@birthdate AS DATE)

RETURNS INT
AS
BEGIN

RETURN DATEDIFF(year, @birthdate, sysdatetime());

END;
GO
```

Scalar Functions

Example-3

```
if exists (SELECT * FROM sys.objects
          WHERE object_id =
OBJECT_ID(N'[fn_RectangleArea]'))
          drop function fn_RectangleArea
go

CREATE FUNCTION fn_RectangleArea
   (@Width int,
@Height int )
RETURNS int
AS
BEGIN

   RETURN ( @Width * @Height )
END
GO
```

User-Defined Functions
Scalar Functions

- Example-4

```
CREATE FUNCTION ReverseCustName(@string varchar(100))
RETURNS varchar(100)
AS
BEGIN
    DECLARE @custName varchar(100)
    -- Implementation left as exercise for users.
    RETURN @custName
END
```

User-Defined Functions

- ALTER FUNCTION (Transact-SQL)
  - Alters an existing Transact-SQL or CLR function that was previously created by executing the CREATE FUNCTION statement, without changing permissions and without affecting any dependent functions, stored procedures, or triggers.

- Syntax

```
ALTER FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type
    [ = default ] }
    [ ,...n ]
  ]
)
RETURNS return_data_type
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN scalar_expression
    END
[ ; ]
```

User-Defined Functions

- DROP FUNCTION (Transact-SQL)
  - Removes one or more user-defined functions from the current database. User-defined functions are created by using CREATE FUNCTION and modified by using ALTER FUNCTION.
- Syntax

```
DROP FUNCTION { [ schema_name. ] function_name } [ ,...n ]
```

- DROP FUNCTION will fail if
  - there are Transact-SQL functions or views in the database that reference this function
  - there are computed columns, CHECK constraints, or DEFAULT constraints that reference the function.

Table-valued Functions

- User-defined functions that return a **table** data type are referred to as table-valued functions.
- These functions can be powerful alternatives to views.
- A table-valued user-defined function can be used where table or view expressions are allowed in Transact-SQL queries. While views are limited to a single SELECT statement, user-defined functions can contain additional statements that allow more powerful logic than is possible in views.
- A table-valued user-defined function can also replace stored procedures that return a single result set.
- The table returned by a user-defined function can be referenced in the FROM clause of a Transact-SQL statement, but stored procedures that return result sets cannot.

Table-Valued Functions

- Inline Table-Valued Function Syntax

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type
    [ = default ] [ READONLY ] }
    [ ,...n ]
  ]
)
RETURNS TABLE
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    RETURN [ ( ] select_stmt [ ) ]
[ ; ]
```

Table-Valued Functions

Example-1

```
CREATE FUNCTION ProductsCostingMoreThan(@cost money)
RETURNS TABLE
AS
RETURN
    SELECT ProductID, UnitPrice
    FROM Products
    WHERE UnitPrice > @cost
```

Table-Valued Functions

- Multistatement Table-valued Function
- Syntax

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type
    [ = default ] [READONLY] }
    [ ,...n ]
  ]
)
RETURNS @return_variable TABLE <table_type_definition>
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN
    END
[ ; ]
```

Table-Valued Functions

- Example-1

```
CREATE FUNCTION DatesBetween(@startDate date, @endDate date)
RETURNS @dates TABLE (DateValue date NOT NULL)
AS
BEGIN
   WHILE (@startDate <= @endDate) BEGIN
      INSERT INTO @dates VALUES (@startDate);
      SET @startDate = DATEADD(day, 1, @startDate);
   END;

   RETURN;
END;
```

Table-Valued Functions

- Example-2

```
create function FN(@Str varchar(max))
  returns
  @Names table(name varchar(25))
  as
  begin

     declare @ln as int;
      set @ln=len(@Str);

      while (charindex(',', @str) > 0)
      begin
      insert into @Names values(substring(@str, 1, charindex(',', @str) - 1));
      set  @str = substring(@str, charindex(',', @str) + 1, @ln);
      end;
      insert into @Names values(@str);

      return;
  end;
```

Getting Information about Database Objects

- Consider the following system tables and functions
  - **Sys.objects**: Contains a row for each user-defined, schema-scoped object that is created within a database
  - sys.tables: Returns a row for each user table in SQL Server
  - sys.columns: Returns a row for each column of an object that has columns, such as views or tables
  - sys.parameters: Contains a row for each parameter of an object that accepts parameters.
  - sys.types: Contains a row for each system and user-defined type
  - sys.triggers: Contains a row for each object that is a trigger, with a type of TR or TA

Getting Information about Database Objects

- Returning all the user-defined functions in a database

```
USE <database_name>;
GO
SELECT name AS function_name
   ,SCHEMA_NAME(schema_id) AS schema_name
   ,type_desc
   ,create_date
   ,modify_date
FROM sys.objects
WHERE type_desc LIKE '%FUNCTION%';
GO
```

Getting Information about Database Objects

- Returning all the objects that have been modified in the last N days

```
USE <database_name>;
GO
SELECT name AS object_name
   ,SCHEMA_NAME(schema_id) AS schema_name
   ,type_desc
   ,create_date
   ,modify_date
FROM sys.objects
WHERE modify_date > GETDATE() - <n_days>
ORDER BY modify_date;
GO
```

Getting Information about Database Objects

- Returning the parameters for a specified stored procedure or function

```
USE <database_name>;
GO
SELECT SCHEMA_NAME(schema_id) AS schema_name
    ,o.name AS object_name
    ,o.type_desc
    ,p.parameter_id
    ,p.name AS parameter_name
    ,TYPE_NAME(p.user_type_id) AS parameter_type
    ,p.max_length
    ,p.precision
    ,p.scale
    ,p.is_output
FROM sys.objects AS o
INNER JOIN sys.parameters AS p ON o.object_id = p.object_id
WHERE o.object_id = OBJECT_ID('<schema_name.object_name>')
ORDER BY schema_name, object_name, p.parameter_id;
GO
```