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.

Transact-SQL Syntax Conventions

Syntax

```
Scalar Functions
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
[ ; ]

Inline Table-valued Functions
ALTER FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type
    [ = default ] }
    [ ,...n ]
  ]
)
RETURNS TABLE
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    RETURN [ ( ] select_stmt [ ) ]
[ ; ]

Multistatement Table-valued Functions
ALTER FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type
    [ = default ] }
    [ ,...n ]
  ]
)
RETURNS @return_variable TABLE <table_type_definition>
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN
    END
```

```
[ ; ]

CLR Functions
ALTER FUNCTION [ schema_name. ] function_name
( { @parameter_name [AS] [ type_schema_name. ] parameter_data_type
    [ = default ] }
    [ ,...n ]
)
RETURNS { return_data_type | TABLE <clr_table_type_definition> }
    [ WITH <clr_function_option> [ ,...n ] ]
    [ AS ] EXTERNAL NAME <method_specifier>
[ ; ]

<method_specifier>::=
    assembly_name.class_name.method_name

Function Options
<function_option>::=
{
    [ ENCRYPTION ]
  | [ SCHEMABINDING ]
  | [ RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT ]
  | [ EXECUTE_AS_Clause ]
}

<clr_function_option>::=
}
    [ RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT ]
  | [ EXECUTE_AS_Clause ]
}

Table Type Definitions
<table_type_definition>:: =
( { <column_definition> <column_constraint>
  | <computed_column_definition> }
    [ <table_constraint> ] [ ,...n ]
)

<clr_table_type_definition>:: =
( { column_name data_type } [ ,...n ] )

<column_definition>::=
{
    { column_name data_type }
    [ [ DEFAULT constant_expression ]
      [ COLLATE collation_name ] | [ ROWGUIDCOL ]
    ]
    | [ IDENTITY [ (seed , increment ) ] ]
    [ <column_constraint> [ ...n ] ]
}
<column_constraint>::=
{
    [ NULL | NOT NULL ]
    { PRIMARY KEY | UNIQUE }
      [ CLUSTERED | NONCLUSTERED ]
        [ WITH FILLFACTOR = fillfactor
        | WITH ( < index_option > [ , ...n ] )
```

```
      [ ON { filegroup | "default" } ]
  | [ CHECK ( logical_expression ) ] [ ,...n ]
}

<computed_column_definition>::=
column_name AS computed_column_expression

<table_constraint>::=
{
    { PRIMARY KEY | UNIQUE }
      [ CLUSTERED | NONCLUSTERED ]
      ( column_name [ ASC | DESC ] [ ,...n ] )
        [ WITH FILLFACTOR = fillfactor
        | WITH ( <index_option> [ , ...n ] )
  | [ CHECK ( logical_expression ) ] [ ,...n ]
}

<index_option>::=
{
    PAD_INDEX = { ON | OFF }
  | FILLFACTOR = fillfactor
  | IGNORE_DUP_KEY = { ON | OFF }
  | STATISTICS_NORECOMPUTE = { ON | OFF }
  | ALLOW_ROW_LOCKS = { ON | OFF }
  | ALLOW_PAGE_LOCKS ={ ON | OFF }
}
```

Arguments

---

schema_name

Is the name of the schema to which the user-defined function belongs.

function_name

Is the user-defined function to be changed.

Note

Parentheses are required after the function name even if a parameter is not specified.

@ parameter_name

Is a parameter in the user-defined function. One or more parameters can be declared.

A function can have a maximum of 2,100 parameters. The value of each declared parameter must be supplied by the user when the function is executed, unless a default for the parameter is defined.

Specify a parameter name by using an at sign (**@**) as the first character. The parameter name must comply with the rules for identifiers. Parameters are local to the function; the same parameter names can be used in other functions. Parameters

can take the place only of constants; they cannot be used instead of table names, column names, or the names of other database objects.

Note
ANSI_WARNINGS is not honored when passing parameters in a stored procedure, user-defined function, or when declaring and setting variables in a batch statement. For example, if a variable is defined as char(3), and then set to a value larger than three characters, the data is truncated to the defined size and the INSERT or UPDATE statement succeeds.

[type_schema_name.] parameter_data_type

Is the parameter data type and optionally, the schema to which it belongs. For Transact-SQL functions, all data types, including CLR user-defined types, are allowed except the timestamp data type. For CLR functions, all data types, including CLR user-defined types, are allowed except text, ntext, image, and timestamp data types. The nonscalar types cursor and table cannot be specified as a parameter data type in either Transact-SQL or CLR functions.

If type_schema_name is not specified, the SQL Server 2005 Database Engine looks for the parameter_data_type in the following order:

- The schema that contains the names of SQL Server system data types.
- The default schema of the current user in the current database.
- The **dbo** schema in the current database.

[=default]

Is a default value for the parameter. If a default value is defined, the function can be executed without specifying a value for that parameter.

Note
Default parameter values can be specified for CLR functions except for varchar(max) and varbinary(max) data types.

When a parameter of the function has a default value, the keyword DEFAULT must be specified when calling the function to retrieve the default value. This behavior is different from using parameters with default values in stored procedures in which omitting the parameter also implies the default value.

return_data_type

Is the return value of a scalar user-defined function. For Transact-SQL functions, all data types, including CLR user-defined types, are allowed except the timestamp data type. For CLR functions, all data types, including CLR user-defined types, are allowed except text, ntext, image, and timestamp data types. The nonscalar types

cursor and table cannot be specified as a return data type in either Transact-SQL or CLR functions.

function_body

Specifies that a series of Transact-SQL statements, which together do not produce a side effect such as modifying a table, define the value of the function. function_body is used only in scalar functions and multistatement table-valued functions.

In scalar functions, function_body is a series of Transact-SQL statements that together evaluate to a scalar value.

In multistatement table-valued functions, function_body is a series of Transact-SQL statements that populate a TABLE return variable.

scalar_expression

Specifies that the scalar function returns a scalar value.

TABLE

Specifies that the return value of the table-valued function is a table. Only constants and **@**local_variables can be passed to table-valued functions.

In inline table-valued functions, the TABLE return value is defined through a single SELECT statement. Inline functions do not have associated return variables.

In multistatement table-valued functions, **@**return_variable is a TABLE variable used to store and accumulate the rows that should be returned as the value of the function. **@**return_variable can be specified only for Transact-SQL functions and not for CLR functions.

select-stmt

Is the single SELECT statement that defines the return value of an inline table-valued function.

EXTERNAL NAME <method_specifier>assembly_name.class_name.method_name

Specifies the method of an assembly to bind with the function. assembly_name must match an existing assembly in SQL Server in the current database with visibility on. class_name must be a valid SQL Server identifier and must exist as a class in the assembly. If the class has a namespace-qualified name that uses a period (**.**) to separate namespace parts, the class name must be delimited by using brackets

(**[ ]**) or quotation marks (" "). method_name must be a valid SQL Server identifier and must exist as a static method in the specified class.

Note

By default, SQL Server cannot execute CLR code. You can create, modify, and drop database objects that reference common language runtime modules; however, you cannot execute these references in SQL Server until you enable the clr enabled option. To enable the option, use sp_configure.

Note

This option is not available in a contained database.

<table_type_definition>**(** { <column_definition> <column_constraint> | <computed_column_definition> } [<table_constraint>] [**,**...n]**)**

Defines the table data type for a Transact-SQL function. The table declaration includes column definitions and column or table constraints.

< clr_table_type_definition > **(** { column_namedata_type } [**,**...n] **)**

Defines the table data types for a CLR function. The table declaration includes only column names and data types.

<function_option>::= and <clr_function_option>::=

Specifies the function will have one or more of the following options.

ENCRYPTION

Indicates that the Database Engine encrypts the catalog view columns that contains the text of the ALTER FUNCTION statement. Using ENCRYPTION prevents the function from being published as part of SQL Server replication. ENCRYPTION cannot be specified for CLR functions.

SCHEMABINDING

Specifies that the function is bound to the database objects that it references. This condition will prevent changes to the function if other schema bound objects are referencing it.

The binding of the function to the objects it references is removed only when one of the following actions occurs:

- The function is dropped.
- The function is modified by using the ALTER statement with the SCHEMABINDING option not specified.

For a list of conditions that must be met before a function can be schema bound, see CREATE FUNCTION (Transact-SQL).

RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT

Specifies the **OnNULLCall** attribute of a scalar-valued function. If not specified, CALLED ON NULL INPUT is implied by default. This means that the function body executes even if NULL is passed as an argument.

If RETURNS NULL ON NULL INPUT is specified in a CLR function, it indicates that SQL Server can return NULL when any of the arguments it receives is NULL, without actually invoking the body of the function. If the method specified in <method_specifier> already has a custom attribute that indicates RETURNS NULL ON NULL INPUT, but the ALTER FUNCTION statement indicates CALLED ON NULL INPUT, the ALTER FUNCTION statement takes precedence. The **OnNULLCall** attribute cannot be specified for CLR table-valued functions.

EXECUTE AS Clause

Specifies the security context under which the user-defined function is executed. Therefore, you can control which user account SQL Server uses to validate permissions on any database objects referenced by the function.

Note
EXECUTE AS cannot be specified for inline user-defined functions.

For more information, see EXECUTE AS Clause (Transact-SQL).

< column_definition >::=

Defines the table data type. The table declaration includes column definitions and constraints. For CLR functions, only column_name and data_type can be specified.

column_name

Is the name of a column in the table. Column names must comply with the rules for identifiers and must be unique in the table. column_name can consist of 1 through 128 characters.

data_type

Specifies the column data type. For Transact-SQL functions, all data types, including CLR user-defined types, are allowed except timestamp. For CLR functions, all data types, including CLR user-defined types, are allowed except text, ntext, image, char, varchar, varchar(max), and timestamp.The nonscalar type cursor cannot be specified as a column data type in either Transact-SQL or CLR functions.

DEFAULT constant_expression

Specifies the value provided for the column when a value is not explicitly supplied during an insert. constant_expression is a constant, NULL, or a system function value. DEFAULT definitions can be applied to any column except those that have the IDENTITY property. DEFAULT cannot be specified for CLR table-valued functions.

COLLATE collation_name

Specifies the collation for the column. If not specified, the column is assigned the default collation of the database. Collation name can be either a Windows collation name or a SQL collation name. For a list of and more information, see Windows Collation Name (Transact-SQL) and SQL Server Collation Name (Transact-SQL).

The COLLATE clause can be used to change the collations only of columns of the char, varchar, nchar, and nvarchar data types.

COLLATE cannot be specified for CLR table-valued functions.

ROWGUIDCOL

Indicates that the new column is a row global unique identifier column. Only one uniqueidentifier column per table can be designated as the ROWGUIDCOL column. The ROWGUIDCOL property can be assigned only to a uniqueidentifier column.

The ROWGUIDCOL property does not enforce uniqueness of the values stored in the column. It also does not automatically generate values for new rows inserted into the table. To generate unique values for each column, use the NEWID function on INSERT statements. A default value can be specified; however, NEWID cannot be specified as the default.

IDENTITY

Indicates that the new column is an identity column. When a new row is added to the table, SQL Server provides a unique, incremental value for the column. Identity columns are typically used together with PRIMARY KEY constraints to serve as the unique row identifier for the table. The IDENTITY property can be assigned to tinyint, smallint, int, bigint, decimal(p,0), or numeric(p,0) columns. Only one identity column can be created per table. Bound defaults and DEFAULT constraints cannot be used with an identity column. You must specify both the seed and increment or neither. If neither is specified, the default is (1,1).

IDENTITY cannot be specified for CLR table-valued functions.

seed

Is the integer value to be assigned to the first row in the table.

increment

Is the integer value to add to the seed value for successive rows in the table.

< column_constraint >::= and < table_constraint>::=

Defines the constraint for a specified column or table. For CLR functions, the only constraint type allowed is NULL. Named constraints are not allowed.

NULL | NOT NULL

Determines whether null values are allowed in the column. NULL is not strictly a constraint but can be specified just like NOT NULL. NOT NULL cannot be specified for CLR table-valued functions.

PRIMARY KEY

Is a constraint that enforces entity integrity for a specified column through a unique index. In table-valued user-defined functions, the PRIMARY KEY constraint can be created on only one column per table. PRIMARY KEY cannot be specified for CLR table-valued functions.

UNIQUE

Is a constraint that provides entity integrity for a specified column or columns through a unique index. A table can have multiple UNIQUE constraints. UNIQUE cannot be specified for CLR table-valued functions.

CLUSTERED | NONCLUSTERED

Indicate that a clustered or a nonclustered index is created for the PRIMARY KEY or UNIQUE constraint. PRIMARY KEY constraints use CLUSTERED, and UNIQUE constraints use NONCLUSTERED.

CLUSTERED can be specified for only one constraint. If CLUSTERED is specified for a UNIQUE constraint and a PRIMARY KEY constraint is also specified, the PRIMARY KEY uses NONCLUSTERED.

CLUSTERED and NONCLUSTERED cannot be specified for CLR table-valued functions.

CHECK

Is a constraint that enforces domain integrity by limiting the possible values that can be entered into a column or columns. CHECK constraints cannot be specified for CLR table-valued functions.

logical_expression

Is a logical expression that returns TRUE or FALSE.

<computed_column_definition>::=

Specifies a computed column. For more information about computed columns, see CREATE TABLE (Transact-SQL).

column_name

Is the name of the computed column.

computed_column_expression

Is an expression that defines the value of a computed column.

<index_option>::=

Specifies the index options for the PRIMARY KEY or UNIQUE index. For more information about index options, see CREATE INDEX (Transact-SQL).

PAD_INDEX = { ON | OFF }

Specifies index padding. The default is OFF.

FILLFACTOR = fillfactor

Specifies a percentage that indicates how full the Database Engine should make the leaf level of each index page during index creation or change. fillfactor must be an integer value from 1 to 100. The default is 0.

IGNORE_DUP_KEY = { ON | OFF }

Specifies the error response when an insert operation attempts to insert duplicate key values into a unique index. The IGNORE_DUP_KEY option applies only to insert operations after the index is created or rebuilt. The default is OFF.

STATISTICS_NORECOMPUTE = { ON | OFF }

Specifies whether distribution statistics are recomputed. The default is OFF.

ALLOW_ROW_LOCKS = { ON | OFF }

Specifies whether row locks are allowed. The default is ON.

ALLOW_PAGE_LOCKS = { ON | OFF }

Specifies whether page locks are allowed. The default is ON.

Remarks

ALTER FUNCTION cannot be used to change a scalar-valued function to a table-valued function, or vice versa. Also, ALTER FUNCTION cannot be used to change an inline function to a multistatement function, or vice versa. ALTER FUNCTION cannot be used to change a Transact-SQL function to a CLR function or vice-versa.

The following Service Broker statements cannot be included in the definition of a Transact-SQL user-defined function:

- BEGIN DIALOG CONVERSATION
- END CONVERSATION
- GET CONVERSATION GROUP
- MOVE CONVERSATION
- RECEIVE
- SEND

Permissions

Requires ALTER permission on the function or on the schema. If the function specifies a user-defined type, requires EXECUTE permission on the type.