



User-Defined Functions



User-Define Functions

- A user-defined function is a collection of T-SQL statements
 - accepts parameters
 - performs an action and
 - returns a result
- Cannot be used to perform actions that produce a **side effect** such as modifying a table
- Can be invoked from a query - like system functions
 - `SELECT dbo.functionName (paramList)`
- `ALTER FUNCTION` is used to modify the function and `DROP FUNCTION` to delete it



User-Define Functions (cont.)

- User-Defined functions are used to create a reusable routine:
 - In Transact-SQL statements such as SELECT
 - In the definition of another user-defined function
 - To parameterize a view or improve the functionality of a view
 - To define a column in a table (Derived-Column)
 - To define a CHECK constraint on a column
- Limitations
 - Cannot return multiple result sets.
 - Use a stored procedure if you need to return multiple result sets.
 - Cannot use @ERROR or RAISERROR.
 - Cannot call a stored procedure.



Components of a UDF

- Have a two-part structure
 - Header and
 - Body.
- The header defines:
 - Function name
 - Input parameter name and data type
 - Return parameter data type and optional name
- The body defines the action, or logic, the function is to perform



Parameters

- A function takes zero or more input parameters
- The return value can either be a single scalar value or a result set (table)
- Optional parameters can be used
 - The DEFAULT keyword is used when the function is called to retrieve the default value
 - This is different from stored procedures in which omitting the parameter also implies the default value.



Types of Functions

- Three type of functions
 - 1. Scalar Functions
 - 2. Table Valued Functions
 - A. Inline Table-Valued Function
 - B. Multi-statement Table-Valued Function



Scalar Functions - Syntax

```
CREATE FUNCTION          function_name
    ( [ @parameterName dataType [ = default ] [ ,...n ] ] )
    RETURNS returnType
AS
BEGIN
    function_body
    RETURN scalar_expression
END
```



Scalar Functions (cont.)

- Returns a single data value of the type defined in the RETURNS clause.
- The return type can be any data type except text, ntext, image, cursor, and timestamp
- Using Scalar Functions
 - Can be used anywhere that a scalar expression of the same data type is allowed
 - Invoked by using at least the two-part name of the function



Scalar Functions – Example 1

```
CREATE FUNCTION          extractLastName ( @fullName varchar(100) )
    RETURNS varchar(50)
AS
BEGIN
    DECLARE @tempFullName varchar(100)
    DECLARE @lastName varchar(50)
    DECLARE @spcPosition int

    SET @tempFullName = LTrim(RTrim(@fullName))
    SET @spcPosition = CHARINDEX( ' ', @tempFullName)

    IF(@spcPosition > 0)
        SET @lastName = SUBSTRING(@tempFullName, @spcPosition + 1
                                   , LEN(@tempFullName))
    ELSE
        SET @lastName = ""
    RETURN(@lastName);
END
```



Scalar Functions – Example 2

```
CREATE FUNCTION      getTotalSalary (@depId int)
    RETURNS decimal(8,2)
AS
BEGIN
    DECLARE          @totSalary decimal(8,2)
    SELECT            @totSalary = SUM(Salary)
    FROM              Employee
    WHERE             DepID = @depId

    RETURN            @totSalary
END
```



Scalar Functions – Example 3

```
CREATE FUNCTION      getMyKindOfDate(@theDate date)
    RETURNS varchar(20)
AS
BEGIN
    RETURN DATENAME(D, (@theDate )
        + CASE
            WHEN DAY(@theDate) IN (1, 21, 31) THEN 'st'
            WHEN DAY(@theDate) IN (2, 22) THEN 'nd'
            WHEN DAY(@theDate) IN (3, 23) THEN 'rd'
            ELSE 'th'
        END
        + ' '
        + DATENAME(MONTH, (@theDate )+ ' '
        + DATENAME(yy, (@theDate )
END
```



Calling Scalar Functions

```
SELECT      dbo.GetFullName(fName, LName) AS  
EmployeeName  
FROM        Employee  
WHERE       dbo.getNetSalary(Salary) > 5000.00  
  
UPDATE      Employee  
SET         Salary = dbo.CalculateNewSalary(EmpID)
```

- `SELECT dbo.funcTotalSalary(1)`
- `SELECT dbo.detDefaultDate()`
- `SELECT dbo.getMyKindOfDate(GETDATE())`
- `SELECT dbo.getFirstName('Abebe Kassa')`



Scalar UDF – Example 4

```
CREATE FUNCTION avSalary (@jobType VARCHAR(50))
RETURNS FLOAT
AS
BEGIN
    RETURN (      SELECT  AVG(Salary) FROM Employee
                  WHERE Occupation = @jobType    )
END
```

- - using the function

```
SELECT      EmployeeID, FirstName
FROM        Employee
WHERE       Salary >= dbo. avSalary(Occupation)
```



Scalar UDF – Example 5

```
CREATE FUNCTION sales .calcSellingPrice(  
    @quantity INT,  
    @uPrice DEC(10,2),  
    @discount DEC(4,2)  
)  
RETURNS DEC(10,2)  
AS  
BEGIN  
    RETURN @quantity * @uPrice * (1 - @discount)  
END
```

- - using the function

```
SELECT          Order_id  
                , SUM(sales.calcSellingPrice(Quantity, UPrice, Discount)) Amount  
  
FROM            OrderItem  
  
GROUP BY        Order_id  
  
ORDER BY        Amount DESC
```



Scalar UDF – Example 6

```
CREATE FUNCTION funcSalesCount ( @EmployeeID int )  
RETURNS INT  
AS  
BEGIN  
    DECLARE @count int  
  
    SELECT @ count = count(*)  
    FROM Sales  
    WHERE EmployeeID = @EmployeeID  
  
    RETURN @ count  
END
```

- - using the function

```
SELECT          FirstName, LastName  
FROM            Sales JOIN Employee ON Sales.EmployeeID = Employee . EmployeeID  
WHERE           dbo. funcSalesCount(EmployeeID) > 10
```

- - you can do this using GROUP BY !!!!



Inline Table-Valued Function

- An Inline Table-Valued Function specifies a single SELECT statement
 - Can be seen as a VIEW with parameters
- Always returns TABLE
 - The table is the result set of a single SELECT statement
 - There is no function body (no BEGIN ... END)
- Do NOT need the TWO-PART name when using the function



Inline Table-Valued UDF (cont.)

```
CREATE FUNCTION funcEmpByDepartment (@depId int)
RETURNS TABLE
AS
RETURN
(
    SELECT      E.FName, E.LName, E.Salary, D.DepName
    FROM        Employee AS E
               JOIN Department AS D ON D.DepID = E.DepID
    WHERE       D.DepID = @depId
)
```

■ Using the function:

```
SELECT * FROM dbo.funcEmpByDepartment(2)
SELECT FName , Lname FROM dbo.funcEmpByDepartment(2)
```



Inline Table-Valued UDF (cont.)

```
CREATE FUNCTION funcTableColumns(@tableName varchar(100))
RETURNS TABLE
AS
RETURN
(
    SELECT      sc.name
    FROM        SysColumns sc
               JOIN SysObjects so ON sc.id = so.id
WHERE         so.name      = @tableName
)
```

Using the function:

```
SELECT * FROM funcTableColumns('Employee')
```



Inline Table-Valued UDF (cont.)

```
CREATE FUNCTION funContactSearch (@LastName varchar(50))
RETURNS TABLE
AS
RETURN
    (SELECT      LastName + ' ' + FirstName AS Name
     , ea.EmailAddress
    FROM      Person as p JOIN EmailAddress ea
     ON ea.PID = p.PID
    WHERE      LastName Like @LastName + '%');
GO
```

Using the function:

```
SELECT * FROM funContactSearch('Ad')
```



Multi-Statement Table-Valued UDF

- Return a table data type
- A TABLE variable is used
 - To define the structure (columns) of the table
 - To insert the rows that should be returned
- The function body is defined in a BEGIN...END block
- A RETURN statement is used without a return value
- Do NOT need the TWO-PART name when using the function



Multi-Statement Table-Valued UDF

```
CREATE FUNCTION GetEmployeesByDepartment (@myDepID INT)
RETURNS @empList TABLE
(
    EmployeeID int primary key NOT NULL
    , EmpName varchar(100) NOT NULL
    , Department varchar(100) NOT NULL
)
AS
BEGIN
    INSERT @empList
    SELECT E.EmpID, E.FirstName + ' ' + E.LastName, D.DepName
    FROM Employee E JOIN Department D ON E.DepID = D.DepID
    WHERE D.DepID = @myDepID
RETURN
END;
```



Multi-Statement Table-Valued UDF

- Using the function:

```
SELECT      EmployeeID , EmpName , Department
FROM        dbo.GetEmployeesByDepartment(1)
```



Exercise

- Create an inline table valued function that will return the list of all employee names that are working in a department when given the ID of the department
- Write a User Defined Function that can be used to convert the letter grade to numeric
 - (i.e. A=4, B=3, C=2, D=1, otherwise 0)



Exercise (cont.)

- Write an Inline Table-Valued user-defined function that can be used instead of the following function

```
CREATE FUNCTION GetEmpData (@depID INT)
RETURNS @empList TABLE
(
    EmployeeID int primary key NOT NULL
    , EmpName varchar(100) NOT NULL
    , Department varchar(100) NOT NULL
)
AS
BEGIN
    INSERT @empList
    SELECT E.EmpID, E.FirstName + ' ' + E.LastName, D.DepName
    FROM Employee E JOIN Department D ON E.DepID = D.DepID
    WHERE D.DepID = @depID

    RETURN
END
```