T-SQL CASE

Developer Network

This document is provided "as-is". Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. This document does not provide you with any legal rights to any intellectual property in any Microsoft product or product name. You may copy and use this document for your internal, reference purposes. You may modify this document for your internal, reference purposes.© 2016 Microsoft. All rights reserved. Terms of Use (https://msdn.microsoft.com/cc300389.aspx) | Trademarks (http://www.microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx)

Table Of Contents

Chapter 1

CASE (Transact-SQL)

Updated: October 20, 2016

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Evaluates a list of conditions and returns one of multiple possible result expressions.

The CASE expression has two formats:

- The simple CASE expression compares an expression to a set of simple expressions to determine the result.
- The searched CASE expression evaluates a set of Boolean expressions to determine the result.

Both formats support an optional ELSE argument.

CASE can be used in any statement or clause that allows a valid expression. For example, you can use CASE in statements such as SELECT, UPDATE, DELETE and SET, and in clauses such as select_list, IN, WHERE, ORDER BY, and HAVING.

Transact-SQL Syntax Conventions

Syntax

```
-- Syntax for SQL Server and Azure SQL Database

Simple CASE expression:
CASE input_expression
     WHEN when_expression THEN result_expression [ ...n ]
```

```
     [ ELSE else_result_expression ]
END
Searched CASE expression:
CASE
     WHEN Boolean_expression THEN result_expression [ ...n ]
     [ ELSE else_result_expression ]
END
```

```
-- Syntax for Azure SQL Data Warehouse and Parallel Data Warehouse

CASE
     WHEN when_expression THEN result_expression [ ...n ]
     [ ELSE else_result_expression ]
END
```

Arguments

input_expression
Is the expression evaluated when the simple CASE format is used. *input_expression* is any valid expression.

WHEN *when_expression*
Is a simple expression to which *input_expression* is compared when the simple CASE format is used. *when_expression* is any valid expression. The data types of *input_expression* and each *when_expression* must be the same or must be an implicit conversion.

THEN *result_expression*
Is the expression returned when *input_expression* equals *when_expression* evaluates to TRUE, or *Boolean_expression* evaluates to TRUE. *result expression* is any valid expression.

ELSE *else_result_expression*
Is the expression returned if no comparison operation evaluates to TRUE. If this argument is omitted and no comparison operation evaluates to TRUE, CASE returns NULL. *else_result_expression* is any valid expression. The data types of *else_result_expression* and any *result_expression* must be the same or must be an implicit conversion.

WHEN *Boolean_expression*
Is the Boolean expression evaluated when using the searched CASE format. *Boolean_expression* is any valid Boolean expression.

Return Types

Returns the highest precedence type from the set of types in *result_expressions* and the optional *else_result_expression*. For more information, see Data Type Precedence (Transact-SQL).

Return Values

Simple CASE expression:

The simple CASE expression operates by comparing the first expression to the expression in each WHEN clause for equivalency. If these expressions are equivalent, the expression in the THEN clause will be returned.

- Allows only an equality check.
- In the order specified, evaluates input_expression = when_expression for each WHEN clause.
- Returns the *result_expression* of the first *input_expression* = *when_expression* that evaluates to TRUE.
- If no *input_expression* = *when_expression* evaluates to TRUE, the SQL Server Database Engine returns the *else_result_expression* if an ELSE clause is specified, or a NULL value if no ELSE clause is specified.

Searched CASE expression:

- Evaluates, in the order specified, *Boolean_expression* for each WHEN clause.
- Returns *result_expression* of the first *Boolean_expression* that evaluates to TRUE.
- If no *Boolean_expression* evaluates to TRUE, the Database Engine returns the *else_result_expression* if an ELSE clause is specified, or a NULL value if no ELSE clause is specified.

Remarks

SQL Server allows for only 10 levels of nesting in CASE expressions.

The CASE expression cannot be used to control the flow of execution of Transact-SQL statements, statement blocks, user-defined functions, and stored procedures. For a list of control-of-flow methods, see Control-of-Flow Language (Transact-SQL).

The CASE statement evaluates its conditions sequentially and stops with the first condition whose condition is satisfied. In some situations, an expression is evaluated before a CASE statement receives the results of the expression as its input. Errors in evaluating these expressions are possible. Aggregate expressions that appear in WHEN arguments to a CASE statement are evaluated first, then provided to the CASE statement. For example, the following query produces a divide by zero error when producing the value of the MAX aggregate. This occurs prior to evaluating the CASE expression.

Transact-SQL

```
WITH Data (value) AS
(
SELECT 0
UNION ALL
SELECT 1
)
SELECT
```

```
   CASE
      WHEN MIN(value) <= 0 THEN 0
      WHEN MAX(1/value) >= 100 THEN 1
   END
FROM Data ;
```

You should only depend on order of evaluation of the WHEN conditions for scalar expressions (including non-correlated sub-queries that return scalars), not for aggregate expressions.

Examples

A. Using a SELECT statement with a simple CASE expression

Within a `SELECT` statement, a simple `CASE` expression allows for only an equality check; no other comparisons are made. The following example uses the `CASE` expression to change the display of product line categories to make them more understandable.

```
USE AdventureWorks2012;
GO
SELECT   ProductNumber, Category =
      CASE ProductLine
         WHEN 'R' THEN 'Road'
         WHEN 'M' THEN 'Mountain'
         WHEN 'T' THEN 'Touring'
         WHEN 'S' THEN 'Other sale items'
         ELSE 'Not for sale'
      END,
   Name
FROM Production.Product
ORDER BY ProductNumber;
GO
```

B. Using a SELECT statement with a searched CASE expression

Within a `SELECT` statement, the searched `CASE` expression allows for values to be replaced in the result set based on comparison values. The following example displays the list price as a text comment based on the price range for a product.

```
USE AdventureWorks2012;
GO
SELECT   ProductNumber, Name, "Price Range" =
      CASE
         WHEN ListPrice =  0 THEN 'Mfg item - not for resale'
```

```
          WHEN ListPrice < 50 THEN 'Under $50'
          WHEN ListPrice >= 50 and ListPrice < 250 THEN 'Under $250'
          WHEN ListPrice >= 250 and ListPrice < 1000 THEN 'Under $1000'
          ELSE 'Over $1000'
      END
FROM Production.Product
ORDER BY ProductNumber ;
GO
```

C. Using CASE in an ORDER BY clause

The following examples uses the CASE expression in an ORDER BY clause to determine the sort order of the rows based on a given column value. In the first example, the value in the `SalariedFlag` column of the `HumanResources.Employee` table is evaluated. Employees that have the `SalariedFlag` set to 1 are returned in order by the `BusinessEntityID` in descending order. Employees that have the `SalariedFlag` set to 0 are returned in order by the `BusinessEntityID` in ascending order. In the second example, the result set is ordered by the column `TerritoryName` when the column `CountryRegionName` is equal to 'United States' and by `CountryRegionName` for all other rows.

```
SELECT BusinessEntityID, SalariedFlag
FROM HumanResources.Employee
ORDER BY CASE SalariedFlag WHEN 1 THEN BusinessEntityID END DESC
        ,CASE WHEN SalariedFlag = 0 THEN BusinessEntityID END;
GO
```

```
SELECT BusinessEntityID, LastName, TerritoryName, CountryRegionName
FROM Sales.vSalesPerson
WHERE TerritoryName IS NOT NULL
ORDER BY CASE CountryRegionName WHEN 'United States' THEN TerritoryName
         ELSE CountryRegionName END;
```

D. Using CASE in an UPDATE statement

The following example uses the CASE expression in an UPDATE statement to determine the value that is set for the column `VacationHours` for employees with `SalariedFlag` set to 0. When subtracting 10 hours from `VacationHours` results in a negative value, `VacationHours` is increased by 40 hours; otherwise, `VacationHours` is increased by 20 hours. The OUTPUT clause is used to display the before and after vacation values.

```
USE AdventureWorks2012;
GO
UPDATE HumanResources.Employee
SET VacationHours =
    ( CASE
         WHEN ((VacationHours - 10.00) < 0) THEN VacationHours + 40
         ELSE (VacationHours + 20.00)
       END
    )
OUTPUT Deleted.BusinessEntityID, Deleted.VacationHours AS BeforeValue,
       Inserted.VacationHours AS AfterValue
WHERE SalariedFlag = 0;
```

E. Using CASE in a SET statement

The following example uses the CASE expression in a SET statement in the table-valued function `dbo.GetContactInfo`. In the **AdventureWorks2012** database, all data related to people is stored in the `Person.Person` table. For example, the person may be an employee, vendor representative, or a customer. The function returns the first and last name of a given `BusinessEntityID` and the contact type for that person.The CASE expression in the SET statement determines the value to display for the column `ContactType` based on the existence of the `BusinessEntityID` column in the `Employee`, `Vendor`, or `Customer` tables.

```
USE AdventureWorks2012;
GO
CREATE FUNCTION dbo.GetContactInformation(@BusinessEntityID int)
    RETURNS @retContactInformation TABLE
(
BusinessEntityID int NOT NULL,
FirstName nvarchar(50) NULL,
LastName nvarchar(50) NULL,
ContactType nvarchar(50) NULL,
    PRIMARY KEY CLUSTERED (BusinessEntityID ASC)
)
AS
-- Returns the first name, last name and contact type for the specified
contact.
BEGIN
    DECLARE
        @FirstName nvarchar(50),
        @LastName nvarchar(50),
        @ContactType nvarchar(50);

    -- Get common contact information
    SELECT
        @BusinessEntityID = BusinessEntityID,
@FirstName = FirstName,
```

```
        @LastName = LastName
    FROM Person.Person
    WHERE BusinessEntityID = @BusinessEntityID;

    SET @ContactType =
        CASE
            -- Check for employee
            WHEN EXISTS(SELECT * FROM HumanResources.Employee AS e
                WHERE e.BusinessEntityID = @BusinessEntityID)
                THEN 'Employee'

            -- Check for vendor
            WHEN EXISTS(SELECT * FROM Person.BusinessEntityContact AS bec
                WHERE bec.BusinessEntityID = @BusinessEntityID)
                THEN 'Vendor'

            -- Check for store
            WHEN EXISTS(SELECT * FROM Purchasing.Vendor AS v
                WHERE v.BusinessEntityID = @BusinessEntityID)
                THEN 'Store Contact'

            -- Check for individual consumer
            WHEN EXISTS(SELECT * FROM Sales.Customer AS c
                WHERE c.PersonID = @BusinessEntityID)
                THEN 'Consumer'
        END;

    -- Return the information to the caller
    IF @BusinessEntityID IS NOT NULL
    BEGIN
        INSERT @retContactInformation
        SELECT @BusinessEntityID, @FirstName, @LastName, @ContactType;
    END;

    RETURN;
END;
GO

SELECT BusinessEntityID, FirstName, LastName, ContactType
FROM dbo.GetContactInformation(2200);
GO
SELECT BusinessEntityID, FirstName, LastName, ContactType
FROM dbo.GetContactInformation(5);
```

F. Using CASE in a HAVING clause

The following example uses the CASE expression in a HAVING clause to restrict the rows returned by the SELECT statement. The statement returns the maximum hourly rate for each job title in the `HumanResources.Employee` table. The HAVING clause restricts the titles to those that are held by men with a maximum pay rate greater than 40 dollars or women with a maximum pay rate greater than 42 dollars.

```
USE AdventureWorks2012;
GO
SELECT JobTitle, MAX(ph1.Rate)AS MaximumRate
FROM HumanResources.Employee AS e
JOIN HumanResources.EmployeePayHistory AS ph1 ON e.BusinessEntityID =
ph1.BusinessEntityID
GROUP BY JobTitle
HAVING (MAX(CASE WHEN Gender = 'M'
        THEN ph1.Rate
        ELSE NULL END) > 40.00
     OR MAX(CASE WHEN Gender  = 'F'
        THEN ph1.Rate
        ELSE NULL END) > 42.00)
ORDER BY MaximumRate DESC;
```

Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

G. Using a SELECT statement with a CASE expression

Within a SELECT statement, the CASE expression allows for values to be replaced in the result set based on comparison values. The following example uses the CASE expression to change the display of product line categories to make them more understandable. When a value does not exist, the text “Not for sale’ is displayed.

```
-- Uses AdventureWorks

SELECT   ProductAlternateKey, Category =
      CASE ProductLine
         WHEN 'R' THEN 'Road'
         WHEN 'M' THEN 'Mountain'
         WHEN 'T' THEN 'Touring'
         WHEN 'S' THEN 'Other sale items'
         ELSE 'Not for sale'
      END,
   EnglishProductName
FROM dbo.DimProduct
ORDER BY ProductKey;
```

H. Using CASE in an UPDATE statement

The following example uses the CASE expression in an UPDATE statement to determine the value that is

set for the column `VacationHours` for employees with `SalariedFlag` set to 0. When subtracting 10 hours from `VacationHours` results in a negative value, `VacationHours` is increased by 40 hours; otherwise, `VacationHours` is increased by 20 hours.

```
-- Uses AdventureWorks

UPDATE dbo.DimEmployee
SET VacationHours =
    ( CASE
         WHEN ((VacationHours - 10.00) < 0) THEN VacationHours + 40
         ELSE (VacationHours + 20.00)
       END
    )
WHERE SalariedFlag = 0;
```

See Also

Expressions (Transact-SQL)
SELECT (Transact-SQL)
COALESCE (Transact-SQL)
IIF (Transact-SQL)
CHOOSE (Transact-SQL)

© 2016 Microsoft