

The CASE expression goes through conditions and returns a value when the first condition is met (like if-then-else statement). If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

Also called CASE STATEMENT

91

 payment csv file: https://bit.ly/41kJLRW

92



**Example:**



**General CASE Syntax**

CASE

WHEN condition1 THEN result1 WHEN condition2 THEN result2 WHEN conditionN THEN resultN ELSE other\_result

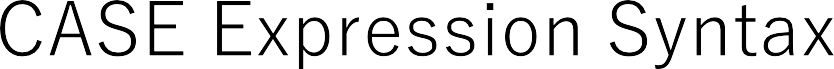
END;

SELECT customer\_id, amount, CASE

WHEN amount > 100 THEN 'Expensive product' WHEN amount = 100 THEN 'Moderate product' ELSE 'Inexpensive product'

END AS ProductStatus FROM payment

93



**Example:** SELECT customer\_id, CASE **amount**



**CASE Expression Syntax**

CASE **Expression**

WHEN value1 THEN result1 WHEN value2 THEN result2 WHEN valueN THEN resultN ELSE other\_result

END;

WHEN 500 THEN 'Prime Customer' WHEN 100 THEN 'Plus Customer' ELSE 'Regular Customer'

END AS CustomerStatus FROM payment

94

The CASE statement and CASE expression in SQL are used for conditional logic within queries. They allow you to execute certain expressions or return specific values based on conditional checks.

CASE Statement

The CASE statement is used to perform conditional logic within SQL statements. It can be used in various parts of a SQL query, such as the SELECT clause, WHERE clause, ORDER BY clause, etc.

Syntax

There are two forms of the CASE statement: the simple CASE and the searched CASE.

1. **Simple CASE**:

 CASE expression  
     WHEN value1 THEN result1  
     WHEN value2 THEN result2  
     ...  
     ELSE default\_result  
 END

1. **Searched CASE**:

 CASE  
     WHEN condition1 THEN result1  
     WHEN condition2 THEN result2  
     ...  
     ELSE default\_result  
 END

**Example with Simple CASE**

Suppose we have an employees table with a job\_title column and we want to categorize employees based on their job titles.

SELECT   
    employee\_id,  
    first\_name,  
    last\_name,  
    job\_title,  
    CASE job\_title  
        WHEN 'Manager' THEN 'Leadership'  
        WHEN 'Developer' THEN 'Technical Staff'  
        WHEN 'Analyst' THEN 'Support Staff'  
        ELSE 'Other'  
    END AS job\_category  
FROM employees;

**Example with Searched CASE**

Here, we categorize employees based on their salary ranges.

SELECT   
    employee\_id,  
    first\_name,  
    last\_name,  
    salary,  
    CASE  
        WHEN salary < 30000 THEN 'Low'  
        WHEN salary BETWEEN 30000 AND 70000 THEN 'Medium'  
        WHEN salary > 70000 THEN 'High'  
        ELSE 'Undefined'  
    END AS salary\_range  
FROM employees;

**CASE Expression**

The CASE expression is used to return a value based on specified conditions. It is typically used in the SELECT statement to create new columns or in the WHERE clause to add conditional logic.

Syntax

The syntax is the same as for the CASE statement:

1. **Simple CASE**:

 SELECT   
     CASE expression  
         WHEN value1 THEN result1  
         WHEN value2 THEN result2  
         ...  
         ELSE default\_result  
     END AS new\_column  
 FROM table\_name;

1. **Searched CASE**:

 SELECT   
     CASE  
         WHEN condition1 THEN result1  
         WHEN condition2 THEN result2  
         ...  
         ELSE default\_result  
     END AS new\_column  
 FROM table\_name;

Real-World Example

Let's assume we have the following orders table:

| **order\_id** | **customer\_id** | **order\_date** | **amount** | **status** |
| --- | --- | --- | --- | --- |
| 1 | 101 | 2023-01-01 | 150.00 | completed |
| 2 | 102 | 2023-01-02 | 200.00 | pending |
| 3 | 103 | 2023-01-03 | 250.00 | shipped |
| 4 | 104 | 2023-01-04 | 300.00 | canceled |

Example of a Simple CASE Expression in a SELECT Statement

Categorize orders based on their status:

SELECT   
    order\_id,  
    customer\_id,  
    amount,  
    status,  
    CASE status  
        WHEN 'completed' THEN 'Closed'  
        WHEN 'pending' THEN 'Open'  
        WHEN 'shipped' THEN 'In Transit'  
        ELSE 'Other'  
    END AS order\_status  
FROM orders;

Example of a Searched CASE Expression in a SELECT Statement

Categorize orders based on the amount:

SELECT   
    order\_id,  
    customer\_id,  
    amount,  
    status,  
    CASE  
        WHEN amount < 200 THEN 'Low Value'  
        WHEN amount BETWEEN 200 AND 300 THEN 'Medium Value'  
        WHEN amount > 300 THEN 'High Value'  
        ELSE 'Unknown'  
    END AS order\_value\_category  
FROM orders;

Example of CASE in a WHERE Clause

Select only high-value completed orders:

SELECT   
    order\_id,  
    customer\_id,  
    amount,  
    status  
FROM orders  
WHERE   
    status = 'completed' AND   
    CASE  
        WHEN amount > 250 THEN TRUE  
        ELSE FALSE  
    END;

Advantages and Use Cases

* **Conditional Logic**: Allows you to embed conditional logic within your SQL statements, making it easier to write complex queries.
* **Data Transformation**: Useful for data transformation, categorizing data, and creating new computed columns based on existing data.
* **Filtering Data**: Can be used in the WHERE clause to filter data based on complex conditions.
* **Improving Readability**: Simplifies complex IF-THEN-ELSE logic into a more readable format within SQL.

Disadvantages

* **Performance**: Depending on the complexity and the database engine, using many CASE statements can impact query performance.
* **Complexity**: Overusing CASE statements in very complex queries can make the query harder to read and maintain.

Use Cases

* **Data Categorization**: Categorizing data based on conditions, such as grouping sales into different revenue bands.
* **Derived Columns**: Creating derived columns that show different values based on specified conditions.
* **Conditional Aggregation**: Using in aggregation queries to apply conditions within aggregate functions.
* **Custom Sorting**: Implementing custom sorting logic within the ORDER BY clause.

By effectively using CASE statements and expressions, you can handle a wide range of conditional logic directly within your SQL queries, making them more powerful and flexible.