Chapter 10: CASE

The CASE expression is used to implement if-then logic.

Section 10.1: Use CASE to COUNT the number of rows in a column match a condition

Use Case

`CASE` can be used in conjunction with `SUM` to return a count of only those items matching a pre-defined condition. (This is similar to `COUNTIF` in Excel.)

The trick is to return binary results indicating matches, so the "1"s returned for matching entries can be summed for a count of the total number of matches.

Given this table `ItemSales`, let's say you want to learn the total number of items that have been categorized as "Expensive":

Id	ItemId	Price	PriceRating
1	100	34.5	EXPENSIVE
2	145	2.3	CHEAP
3	100	34.5	EXPENSIVE
4	100	34.5	EXPENSIVE
5	145	10	AFFORDABLE

Query

```
SELECT
    COUNT(Id) AS ItemsCount,
    SUM ( CASE
            WHEN PriceRating = 'Expensive' THEN 1
            ELSE 0
          END
        ) AS ExpensiveItemsCount
FROM ItemSales
```

Results:

ItemsCount	ExpensiveItemsCount
5	3

Alternative:

```
SELECT
    COUNT(Id) as ItemsCount,
    SUM (
        CASE PriceRating
            WHEN 'Expensive' THEN 1
            ELSE 0
        END
       ) AS ExpensiveItemsCount
FROM ItemSales
```

Section 10.2: Searched CASE in SELECT (Matches a boolean expression)

The *searched* CASE returns results when a *boolean* expression is TRUE.

(This differs from the simple case, which can only check for equivalency with an input.)

```
SELECT Id, ItemId, Price,
  CASE WHEN Price < 10 THEN 'CHEAP'
       WHEN Price < 20 THEN 'AFFORDABLE'
       ELSE 'EXPENSIVE'
  END AS PriceRating
FROM ItemSales
```

Id	ItemId	Price	PriceRating
1	100	34.5	EXPENSIVE
2	145	2.3	CHEAP
3	100	34.5	EXPENSIVE
4	100	34.5	EXPENSIVE
5	145	10	AFFORDABLE

Section 10.3: CASE in a clause ORDER BY

We can use 1,2,3.. to determine the type of order:

```
SELECT * FROM DEPT
ORDER BY
CASE DEPARTMENT
      WHEN 'MARKETING' THEN  1
      WHEN 'SALES' THEN 2
      WHEN 'RESEARCH' THEN 3
      WHEN 'INNOVATION' THEN 4
      ELSE        5
      END,
      CITY
```

ID	REGION	CITY	DEPARTMENT	EMPLOYEES_NUMBER
12	New England	Boston	MARKETING	9
15	West	San Francisco	MARKETING	12
9	Midwest	Chicago	SALES	8
14	Mid-Atlantic	New York	SALES	12
5	West	Los Angeles	RESEARCH	11
10	Mid-Atlantic	Philadelphia	RESEARCH	13
4	Midwest	Chicago	INNOVATION	11
2	Midwest	Detroit	HUMAN RESOURCES	9

Section 10.4: Shorthand CASE in SELECT

`CASE`'s shorthand variant evaluates an expression (usually a column) against a series of values. This variant is a bit shorter, and saves repeating the evaluated expression over and over again. The `ELSE` clause can still be used, though:

```
SELECT Id, ItemId, Price,
  CASE Price WHEN 5  THEN 'CHEAP'
             WHEN 15 THEN 'AFFORDABLE'
```

```
            ELSE         'EXPENSIVE'
  END as PriceRating
FROM ItemSales
```

A word of caution. It's important to realize that when using the short variant the entire statement is evaluated at each `WHEN`. Therefore the following statement:

```
SELECT
    CASE ABS(CHECKSUM(NEWID())) % 4
        WHEN 0 THEN 'Dr'
        WHEN 1 THEN 'Master'
        WHEN 2 THEN 'Mr'
        WHEN 3 THEN 'Mrs'
    END
```

may produce a `NULL` result. That is because at each `WHEN` `NEWID()` is being called again with a new result. Equivalent to:

```
SELECT
    CASE
        WHEN ABS(CHECKSUM(NEWID())) % 4 = 0 THEN 'Dr'
        WHEN ABS(CHECKSUM(NEWID())) % 4 = 1 THEN 'Master'
        WHEN ABS(CHECKSUM(NEWID())) % 4 = 2 THEN 'Mr'
        WHEN ABS(CHECKSUM(NEWID())) % 4 = 3 THEN 'Mrs'
    END
```

Therefore it can miss all the `WHEN` cases and result as `NULL`.

Section 10.5: Using CASE in UPDATE

sample on price increases:

```
UPDATE ItemPrice
SET Price = Price *
  CASE ItemId
    WHEN 1 THEN 1.05
    WHEN 2 THEN 1.10
    WHEN 3 THEN 1.15
    ELSE 1.00
  END
```

Section 10.6: CASE use for NULL values ordered last

in this way '0' representing the known values are ranked first, '1' representing the NULL values are sorted by the last:

```
SELECT ID
      ,REGION
      ,CITY
      ,DEPARTMENT
      ,EMPLOYEES_NUMBER
  FROM DEPT
  ORDER BY
  CASE WHEN REGION IS NULL THEN 1
  ELSE 0
  END,
  REGION
```

ID	REGION	CITY	DEPARTMENT	EMPLOYEES_NUMBER
10	Mid-Atlantic	Philadelphia	RESEARCH	13
14	Mid-Atlantic	New York	SALES	12
9	Midwest	Chicago	SALES	8
12	New England	Boston	MARKETING	9
5	West	Los Angeles	RESEARCH	11
15	NULL	San Francisco	MARKETING	12
4	NULL	Chicago	INNOVATION	11
2	NULL	Detroit	HUMAN RESOURCES	9

Section 10.7: CASE in ORDER BY clause to sort records by lowest value of 2 columns

Imagine that you need sort records by lowest value of either one of two columns. Some databases could use a non-aggregated `MIN()` or `LEAST()` function for this (`... ORDER BY MIN(Date1, Date2)`), but in standard SQL, you have to use a `CASE` expression.

The `CASE` expression in the query below looks at the `Date1` and `Date2` columns, checks which column has the lower value, and sorts the records depending on this value.

Sample data

Id	Date1	Date2
1	2017-01-01	2017-01-31
2	2017-01-31	2017-01-03
3	2017-01-31	2017-01-02
4	2017-01-06	2017-01-31
5	2017-01-31	2017-01-05
6	2017-01-04	2017-01-31

Query

```
SELECT Id, Date1, Date2
FROM YourTable
ORDER BY CASE
           WHEN COALESCE(Date1, '1753-01-01') < COALESCE(Date2, '1753-01-01') THEN Date1
           ELSE Date2
         END
```

Results

Id	Date1	Date2
1	**2017-01-01**	2017-01-31
3	2017-01-31	**2017-01-02**
2	2017-01-31	**2017-01-03**
6	**2017-01-04**	2017-01-31
5	2017-01-31	**2017-01-05**
4	**2017-01-06**	2017-01-31

Explanation

As you see row with `Id = 1` is first, that because `Date1` have lowest record from entire table `2017-01-01`, row where `Id = 3` is second that because `Date2` equals to `2017-01-02` that is second lowest value from table and so on.

So we have sorted records from `2017-01-01` to `2017-01-06` ascending and no care on which one column `Date1` or `Date2` are those values.