MySQLTUTORIAL

MySQL ROLLUP

Summary: in this tutorial, you will learn how to use the MySQL `ROLLUP` clause to generate subtotals and grand totals.

Setting up a sample table

The following statement creates a new table (https://www.mysqltutorial.org/mysql-create-table/) named `sales` that stores the order values summarized by product lines and years. The data comes from the `products`, `orders`, and `orderDetails` tables in the sample database (https://www.mysqltutorial.org/mysql-sample-database.aspx).

```
CREATE TABLE sales
SELECT
    productLine,
    YEAR(orderDate) orderYear,
    SUM(quantityOrdered * priceEach) orderValue
FROM
    orderDetails
        INNER JOIN
    orders USING (orderNumber)
        INNER JOIN
    products USING (productCode)
GROUP BY
    productLine ,
    YEAR(orderDate);
```

The following query returns all rows from the `sales` table:

```
SELECT * FROM sales;
```

MySQL ROLLUP Overview

A grouping set is a set of columns to which you want to group. For example, the following query creates a grouping set denoted by `(productline)`

```
SELECT
    productline,
    SUM(orderValue) totalOrderValue
FROM
    sales
GROUP BY
    productline;
```

The following query creates an empty grouping set denoted by the `()`:

```
SELECT
    SUM(orderValue) totalOrderValue
FROM
    sales;
```

If you want to generate two or more grouping sets together in one query, you may use the `UNION ALL` (https://www.mysqltutorial.org/sql-union-mysql.aspx) operator as follows:

```
SELECT
    productline,
    SUM(orderValue) totalOrderValue
FROM
    sales
GROUP BY
    productline
UNION ALL
SELECT
    NULL,
    SUM(orderValue) totalOrderValue
FROM
    sales;
```

Here's the query output:

Because the `UNION ALL` requires all queries to have the same number of columns, we added `NULL` in the select list of the second query to fulfill this requirement.

The `NULL` in the `productLine` column identifies the grand total super-aggregate line.

This query is able to generate the total order values by product lines and also the grand total row. However, it has two problems:

1. The query is quite lengthy.
2. The performance of the query may not be good since the database engine has to internally execute two separate queries and combine the result sets into one.

To fix these issues, you can use the `ROLLUP` clause.

The `ROLLUP` clause is an extension of the `GROUP BY (https://www.mysqltutorial.org/mysql-group-by.aspx)` clause with the following syntax:

```
SELECT
    select_list
FROM
    table_name
GROUP BY
    c1, c2, c3 WITH ROLLUP;
```

The `ROLLUP` generates multiple grouping sets based on the columns or expressions specified in the `GROUP BY` clause. For example:

```
SELECT
    productLine,
    SUM(orderValue) totalOrderValue
FROM
    sales
GROUP BY
    productline WITH ROLLUP;
```

Here is the output:

As clearly shown in the output, the `ROLLUP` clause generates not only the subtotals but also the grand total of the order values.

If you have more than one column specified in the `GROUP BY` clause, the `ROLLUP` clause assumes a hierarchy among the input columns.

For example:

```
GROUP BY c1, c2, c3 WITH ROLLUP
```

The `ROLLUP` assumes that there is the following hierarchy:

```
c1 > c2 > c3
```

And it generates the following grouping sets:

```
(c1, c2, c3)
(c1, c2)
(c1)
()
```

And in case you have two columns specified in the `GROUP BY` clause:

```
GROUP BY c1, c2 WITH ROLLUP
```

then the `ROLLUP` generates the following grouping sets:

```
(c1, c2)
(c1)
()
```

See the following query example:

```
SELECT
    productLine,
    orderYear,
    SUM(orderValue) totalOrderValue
FROM
    sales
GROUP BY
    productline,
    orderYear
WITH ROLLUP;
```

Here is the output:

The `ROLLUP` generates the subtotal row every time the product line changes and the grand total at the end of the result.

The hierarchy in this case is:

```
productLine > orderYear
```

If you reverse the hierarchy, for example:

```
SELECT
    orderYear,
    productLine,
    SUM(orderValue) totalOrderValue
FROM
```

```
    sales
GROUP BY
    orderYear,
    productline
WITH ROLLUP;
```

The following picture shows the output:

The `ROLLUP` generates the subtotal every time the year changes and the grand total at the end of the result set.

The hierarchy in this example is:

```
orderYear > productLine
```

The GROUPING() function

To check whether `NULL` in the result set represents the subtotals or grand totals, you use the `GROUPING()` function.

The `GROUPING()` function returns 1 when `NULL` occurs in a supper-aggregate row, otherwise, it returns 0.

The `GROUPING()` function can be used in the select list, `HAVING` clause, and (as of MySQL 8.0.12) `ORDER BY` clause.

Consider the following query:

```
SELECT
    orderYear,
    productLine,
    SUM(orderValue) totalOrderValue,
    GROUPING(orderYear),
    GROUPING(productLine)
FROM
    sales
GROUP BY
    orderYear,
    productline
WITH ROLLUP;
```

The following picture shows the output:

The `GROUPING(orderYear)` returns 1 when `NULL` in the `orderYear` column occurs in a super-aggregate row, 0 otherwise.

Similarly, the `GROUPING(productLine)` returns 1 when `NULL` in the `productLine` column occurs in a super-aggregate row, 0 otherwise.

We often use `GROUPING()` function to substitute meaningful labels for super-aggregate `NULL` values instead of displaying it directly.

The following example shows how to combine the `IF()` (https://www.mysqltutorial.org/mysql-if-function.aspx) function with the `GROUPING()` function to substitute labels for the super-aggregate `NULL` values in `orderYear` and `productLine` columns:

```
SELECT
    IF(GROUPING(orderYear),
        'All Years',
        orderYear) orderYear,
```

```
    IF(GROUPING(productLine),
        'All Product Lines',
        productLine) productLine,
    SUM(orderValue) totalOrderValue
FROM
    sales
GROUP BY
    orderYear ,
    productline
WITH ROLLUP;
```

The output is:

In this tutorial, you have learned how to use the MySQL `ROLLUP()` to generate multiple grouping sets considering a hierarchy between columns specified in the `GROUP BY` clause.