12.19.2 GROUP BY Modifiers

The `GROUP BY` clause permits a `WITH ROLLUP` modifier that causes summary output to include extra rows that represent higher-level (that is, super-aggregate) summary operations. `ROLLUP` thus enables you to answer questions at multiple levels of analysis with a single query. For example, `ROLLUP` can be used to provide support for OLAP (Online Analytical Processing) operations.

Suppose that a `sales` table has `year`, `country`, `product`, and `profit` columns for recording sales profitability:

```
CREATE TABLE sales
(
    year    INT,
    country VARCHAR(20),
    product VARCHAR(32),
    profit  INT
);
```

To summarize table contents per year, use a simple `GROUP BY` like this:

```
mysql> SELECT year, SUM(profit) AS profit
       FROM sales
       GROUP BY year;
+------+--------+
| year | profit |
+------+--------+
| 2000 |   4525 |
| 2001 |   3010 |
+------+--------+
```

The output shows the total (aggregate) profit for each year. To also determine the total profit summed over all years, you must add up the individual values yourself or run an additional query. Or you can use `ROLLUP`, which provides both levels of analysis with a single query. Adding a `WITH ROLLUP` modifier to the `GROUP BY` clause causes the query to produce another (super-aggregate) row that shows the grand total over all year values:

```
mysql> SELECT year, SUM(profit) AS profit
       FROM sales
```

```
       GROUP BY year WITH ROLLUP;
+------+--------+
| year | profit |
+------+--------+
| 2000 |   4525 |
| 2001 |   3010 |
| NULL |   7535 |
+------+--------+
```

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

`ROLLUP` has a more complex effect when there are multiple `GROUP BY` columns. In this case, each time there is a change in value in any but the last grouping column, the query produces an extra super-aggregate summary row.

For example, without `ROLLUP`, a summary of the `sales` table based on `year`, `country`, and `product` might look like this, where the output indicates summary values only at the year/country/product level of analysis:

```
mysql> SELECT year, country, product, SUM(profit) AS profit
       FROM sales
       GROUP BY year, country, product;
+------+---------+------------+--------+
| year | country | product    | profit |
+------+---------+------------+--------+
| 2000 | Finland | Computer   |   1500 |
| 2000 | Finland | Phone      |    100 |
| 2000 | India   | Calculator |    150 |
| 2000 | India   | Computer   |   1200 |
| 2000 | USA     | Calculator |     75 |
| 2000 | USA     | Computer   |   1500 |
| 2001 | Finland | Phone      |     10 |
| 2001 | USA     | Calculator |     50 |
| 2001 | USA     | Computer   |   2700 |
| 2001 | USA     | TV         |    250 |
+------+---------+------------+--------+
```

With `ROLLUP` added, the query produces several extra rows:

```
mysql> SELECT year, country, product, SUM(profit) AS profit
       FROM sales
       GROUP BY year, country, product WITH ROLLUP;
+------+---------+------------+--------+
| year | country | product    | profit |
```

```
+------+---------+------------+--------+
| 2000 | Finland | Computer   |   1500 |
| 2000 | Finland | Phone      |    100 |
| 2000 | Finland | NULL       |   1600 |
| 2000 | India   | Calculator |    150 |
| 2000 | India   | Computer   |   1200 |
| 2000 | India   | NULL       |   1350 |
| 2000 | USA     | Calculator |     75 |
| 2000 | USA     | Computer   |   1500 |
| 2000 | USA     | NULL       |   1575 |
| 2000 | NULL    | NULL       |   4525 |
| 2001 | Finland | Phone      |     10 |
| 2001 | Finland | NULL       |     10 |
| 2001 | USA     | Calculator |     50 |
| 2001 | USA     | Computer   |   2700 |
| 2001 | USA     | TV         |    250 |
| 2001 | USA     | NULL       |   3000 |
| 2001 | NULL    | NULL       |   3010 |
| NULL | NULL    | NULL       |   7535 |
+------+---------+------------+--------+
```

Now the output includes summary information at four levels of analysis, not just one:

- Following each set of product rows for a given year and country, an extra super-aggregate summary row appears showing the total for all products. These rows have the `product` column set to `NULL`.
- Following each set of rows for a given year, an extra super-aggregate summary row appears showing the total for all countries and products. These rows have the `country` and `products` columns set to `NULL`.
- Finally, following all other rows, an extra super-aggregate summary row appears showing the grand total for all years, countries, and products. This row has the `year`, `country`, and `products` columns set to `NULL`.

The `NULL` indicators in each super-aggregate row are produced when the row is sent to the client. The server looks at the columns named in the `GROUP BY` clause following the leftmost one that has changed value. For any column in the result set with a name that matches any of those names, its value is set to `NULL`. (If you specify grouping columns by column position, the server identifies which columns to set to `NULL` by position.)

Because the `NULL` values in the super-aggregate rows are placed into the result set at such a late stage in query processing, you can test them as `NULL` values only in the select list or `HAVING` clause. You cannot test them as `NULL` values in join conditions or the `WHERE` clause to determine which rows to select. For

example, you cannot add `WHERE product IS NULL` to the query to eliminate from the output all but the super-aggregate rows.

The `NULL` values do appear as `NULL` on the client side and can be tested as such using any MySQL client programming interface. However, at this point, you cannot distinguish whether a `NULL` represents a regular grouped value or a super-aggregate value. To test the distinction, use the `GROUPING()` function, described later.

Previously, MySQL did not allow the use of `DISTINCT` or `ORDER BY` in a query having a `WITH ROLLUP` option. This restriction is lifted in MySQL 8.0.12 and later. (Bug #87450, Bug #86311, Bug #26640100, Bug #26073513)

For `GROUP BY ... WITH ROLLUP` queries, to test whether `NULL` values in the result represent super-aggregate values, the `GROUPING()` function is available for use in the select list, `HAVING` clause, and (as of MySQL 8.0.12) `ORDER BY` clause. For example, `GROUPING(year)` returns 1 when `NULL` in the `year` column occurs in a super-aggregate row, and 0 otherwise. Similarly, `GROUPING(country)` and `GROUPING(product)` return 1 for super-aggregate `NULL` values in the `country` and `product` columns, respectively:

```
mysql> SELECT
         year, country, product, SUM(profit) AS profit,
         GROUPING(year) AS grp_year,
         GROUPING(country) AS grp_country,
         GROUPING(product) AS grp_product
       FROM sales
       GROUP BY year, country, product WITH ROLLUP;
+------+---------+------------+--------+----------+-------------+-------------+
| year | country | product    | profit | grp_year | grp_country | grp_product |
+------+---------+------------+--------+----------+-------------+-------------+
| 2000 | Finland | Computer   |   1500 |        0 |           0 |           0 |
| 2000 | Finland | Phone      |    100 |        0 |           0 |           0 |
| 2000 | Finland | NULL       |   1600 |        0 |           0 |           1 |
| 2000 | India   | Calculator |    150 |        0 |           0 |           0 |
| 2000 | India   | Computer   |   1200 |        0 |           0 |           0 |
| 2000 | India   | NULL       |   1350 |        0 |           0 |           1 |
| 2000 | USA     | Calculator |     75 |        0 |           0 |           0 |
| 2000 | USA     | Computer   |   1500 |        0 |           0 |           0 |
| 2000 | USA     | NULL       |   1575 |        0 |           0 |           1 |
| 2000 | NULL    | NULL       |   4525 |        0 |           1 |           1 |
| 2001 | Finland | Phone      |     10 |        0 |           0 |           0 |
| 2001 | Finland | NULL       |     10 |        0 |           0 |           1 |
| 2001 | USA     | Calculator |     50 |        0 |           0 |           0 |
| 2001 | USA     | Computer   |   2700 |        0 |           0 |           0 |
| 2001 | USA     | TV         |    250 |        0 |           0 |           0 |
```

```
| 2001 | USA     | NULL       |   3000 |        0 |           0 |           1 |
| 2001 | NULL    | NULL       |   3010 |        0 |           1 |           1 |
| NULL | NULL    | NULL       |   7535 |        1 |           1 |           1 |
+------+---------+------------+--------+----------+-------------+-------------+
```

Instead of displaying the `GROUPING()` results directly, you can use `GROUPING()` to substitute labels for super-aggregate `NULL` values:

```
mysql> SELECT
         IF(GROUPING(year), 'All years', year) AS year,
         IF(GROUPING(country), 'All countries', country) AS country,
         IF(GROUPING(product), 'All products', product) AS product,
         SUM(profit) AS profit
       FROM sales
       GROUP BY year, country, product WITH ROLLUP;
+-----------+---------------+-------------+--------+
| year      | country       | product     | profit |
+-----------+---------------+-------------+--------+
| 2000      | Finland       | Computer    |   1500 |
| 2000      | Finland       | Phone       |    100 |
| 2000      | Finland       | All products |   1600 |
| 2000      | India         | Calculator  |    150 |
| 2000      | India         | Computer    |   1200 |
| 2000      | India         | All products |   1350 |
| 2000      | USA           | Calculator  |     75 |
| 2000      | USA           | Computer    |   1500 |
| 2000      | USA           | All products |   1575 |
| 2000      | All countries | All products |   4525 |
| 2001      | Finland       | Phone       |     10 |
| 2001      | Finland       | All products |     10 |
| 2001      | USA           | Calculator  |     50 |
| 2001      | USA           | Computer    |   2700 |
| 2001      | USA           | TV          |    250 |
| 2001      | USA           | All products |   3000 |
| 2001      | All countries | All products |   3010 |
| All years | All countries | All products |   7535 |
+-----------+---------------+-------------+--------+
```

With multiple expression arguments, `GROUPING()` returns a result representing a bitmask that combines the results for each expression, with the lowest-order bit corresponding to the result for the rightmost expression. For example, `GROUPING(year, country, product)` is evaluated like this:

```
  result for GROUPING(product)
+ result for GROUPING(country) << 1
```

```
+ result for GROUPING(year) << 2
```

The result of such a `GROUPING()` is nonzero if any of the expressions represents a super-aggregate `NULL`, so you can return only the super-aggregate rows and filter out the regular grouped rows like this:

```
mysql> SELECT year, country, product, SUM(profit) AS profit
       FROM sales
       GROUP BY year, country, product WITH ROLLUP
       HAVING GROUPING(year, country, product) <> 0;
+------+---------+---------+--------+
| year | country | product | profit |
+------+---------+---------+--------+
| 2000 | Finland | NULL    |   1600 |
| 2000 | India   | NULL    |   1350 |
| 2000 | USA     | NULL    |   1575 |
| 2000 | NULL    | NULL    |   4525 |
| 2001 | Finland | NULL    |     10 |
| 2001 | USA     | NULL    |   3000 |
| 2001 | NULL    | NULL    |   3010 |
| NULL | NULL    | NULL    |   7535 |
+------+---------+---------+--------+
```

The `sales` table contains no `NULL` values, so all `NULL` values in a `ROLLUP` result represent super-aggregate values. When the data set contains `NULL` values, `ROLLUP` summaries may contain `NULL` values not only in super-aggregate rows, but also in regular grouped rows. `GROUPING()` enables these to be distinguished. Suppose that table `t1` contains a simple data set with two grouping factors for a set of quantity values, where `NULL` indicates something like “other” or “unknown”:

```
mysql> SELECT * FROM t1;
+------+-------+----------+
| name | size  | quantity |
+------+-------+----------+
| ball | small |       10 |
| ball | large |       20 |
| ball | NULL  |        5 |
| hoop | small |       15 |
| hoop | large |        5 |
| hoop | NULL  |        3 |
+------+-------+----------+
```

A simple `ROLLUP` operation produces these results, in which it is not so easy to distinguish `NULL` values in super-aggregate rows from `NULL` values in regular grouped rows:

```
mysql> SELECT name, size, SUM(quantity) AS quantity
       FROM t1
       GROUP BY name, size WITH ROLLUP;
+------+-------+----------+
| name | size  | quantity |
+------+-------+----------+
| ball | NULL  |        5 |
| ball | large |       20 |
| ball | small |       10 |
| ball | NULL  |       35 |
| hoop | NULL  |        3 |
| hoop | large |        5 |
| hoop | small |       15 |
| hoop | NULL  |       23 |
| NULL | NULL  |       58 |
+------+-------+----------+
```

Using `GROUPING()` to substitute labels for the super-aggregate `NULL` values makes the result easier to interpret:

```
mysql> SELECT
         IF(GROUPING(name) = 1, 'All items', name) AS name,
         IF(GROUPING(size) = 1, 'All sizes', size) AS size,
         SUM(quantity) AS quantity
       FROM t1
       GROUP BY name, size WITH ROLLUP;
+-----------+-----------+----------+
| name      | size      | quantity |
+-----------+-----------+----------+
| ball      | NULL      |        5 |
| ball      | large     |       20 |
| ball      | small     |       10 |
| ball      | All sizes |       35 |
| hoop      | NULL      |        3 |
| hoop      | large     |        5 |
| hoop      | small     |       15 |
| hoop      | All sizes |       23 |
| All items | All sizes |       58 |
+-----------+-----------+----------+
```

Other Considerations When using ROLLUP

The following discussion lists some behaviors specific to the MySQL implementation of `ROLLUP`.

Prior to MySQL 8.0.12, when you use `ROLLUP`, you cannot also use an `ORDER BY` clause to sort the results. In other words, `ROLLUP` and `ORDER BY` were mutually exclusive in MySQL. However, you still have some control over sort order. To work around the restriction that prevents using `ROLLUP` with `ORDER BY` and achieve a specific sort order of grouped results, generate the grouped result set as a derived table and apply `ORDER BY` to it. For example:

```
mysql> SELECT * FROM
         (SELECT year, SUM(profit) AS profit
         FROM sales GROUP BY year WITH ROLLUP) AS dt
       ORDER BY year DESC;
+------+--------+
| year | profit |
+------+--------+
| 2001 |   3010 |
| 2000 |   4525 |
| NULL |   7535 |
+------+--------+
```

As of MySQL 8.0.12, `ORDER BY` and `ROLLUP` can be used together, which enables the use of `ORDER BY` and `GROUPING()` to achieve a specific sort order of grouped results. For example:

```
mysql> SELECT year, SUM(profit) AS profit
       FROM sales
       GROUP BY year WITH ROLLUP
       ORDER BY GROUPING(year) DESC;
+------+--------+
| year | profit |
+------+--------+
| NULL |   7535 |
| 2000 |   4525 |
| 2001 |   3010 |
+------+--------+
```

In both cases, the super-aggregate summary rows sort with the rows from which they are calculated, and their placement depends on sort order (at the end for ascending sort, at the beginning for descending sort).

`LIMIT` can be used to restrict the number of rows returned to the client. `LIMIT` is applied after `ROLLUP`, so the limit applies against the extra rows added by `ROLLUP`. For example:

```
mysql> SELECT year, country, product, SUM(profit) AS profit
       FROM sales
       GROUP BY year, country, product WITH ROLLUP
       LIMIT 5;
+------+---------+------------+--------+
| year | country | product    | profit |
+------+---------+------------+--------+
| 2000 | Finland | Computer   |   1500 |
| 2000 | Finland | Phone      |    100 |
| 2000 | Finland | NULL       |   1600 |
| 2000 | India   | Calculator |    150 |
| 2000 | India   | Computer   |   1200 |
+------+---------+------------+--------+
```

Using `LIMIT` with `ROLLUP` may produce results that are more difficult to interpret, because there is less context for understanding the super-aggregate rows.

A MySQL extension permits a column that does not appear in the `GROUP BY` list to be named in the select list. (For information about nonaggregated columns and `GROUP BY`, see Section 12.19.3, “MySQL Handling of GROUP BY”.) In this case, the server is free to choose any value from this nonaggregated column in summary rows, and this includes the extra rows added by `WITH ROLLUP`. For example, in the following query, `country` is a nonaggregated column that does not appear in the `GROUP BY` list and values chosen for this column are nondeterministic:

```
mysql> SELECT year, country, SUM(profit) AS profit
       FROM sales
       GROUP BY year WITH ROLLUP;
+------+---------+--------+
| year | country | profit |
+------+---------+--------+
| 2000 | India   |   4525 |
| 2001 | USA     |   3010 |
| NULL | USA     |   7535 |
+------+---------+--------+
```

This behavior is permitted when the `ONLY_FULL_GROUP_BY` SQL mode is not enabled. If that mode is enabled, the server rejects the query as illegal because `country` is not listed in the `GROUP BY` clause. With `ONLY_FULL_GROUP_BY` enabled, you can still execute the query by using the `ANY_VALUE()` function for nondeterministic-value columns:

```
mysql> SELECT year, ANY_VALUE(country) AS country, SUM(profit) AS profit
       FROM sales
       GROUP BY year WITH ROLLUP;
+------+---------+--------+
| year | country | profit |
+------+---------+--------+
| 2000 | India   |   4525 |
| 2001 | USA     |   3010 |
| NULL | USA     |   7535 |
+------+---------+--------+
```

In MySQL 8.0.28 and later, a rollup column cannot be used as an argument to `MATCH()` (and is rejected with an error) except when called in a `WHERE` clause. See Section 12.9, “Full-Text Search Functions”, for more information.