Theory: GROUP BY statement

30 minutes 0 / 5 problems solved

Earlier we've learned how useful aggregation functions can be for solving simple analytical tasks. Now imagine that we need to compute an aggregate value not for the whole table or one specific category but rather for many separate categories. For example, there is a table named *stocks* with historical data on prices.

stock_name	price	datetime
WTI	89.8	2020-05-17 11:00
FB	26.3	2020-04-11 10:23
WTI	18.9	2019-01-18 23:02
WTI	20.9	2019-01-18 23:02
FB	15.6	NULL
DJI	52.7	2004-05-28 21:09
FB	63.7	1998-03-31 04:18

It is feasible but quite inconvenient to execute a query like this for each category:

```
SELECT MAX(price)
FROM stocks
WHERE stock_name = 'WTI';
```

Even more than that, what if we want to get corresponding results for all stocks in the form of a table and use it for further computations?
In order to simplify such a workload **GROUP BY** statement was introduced in SQL. In this topic, you will get a perspective on how it works.

§1. General form

That's how we can use GROUP BY for the task:

```
SELECT stock_name, MAX(price) as maximum
FROM stocks
GROUP BY stock_name;
```

The output would look like this:

stock_name	maximum
WTI	89.8
FB	63.7
DJI	52.7

What's so special about this query? In the GROUP BY clause, we specify the name of a column from the table. Every unique value from this column will get its own result of every utilized aggregate function from SELECT block. Rows corresponding to this value will be taken as an input. It's possible to group things by a computed value that is not currently present in a table; for example, we can introduce groups based on length of stock_name using GROUP BY LENGTH(stock_name).

Nothing forbids us to use more than one aggregate function. Returned values can be completely independent of each other:

Current topic:

GROUP BY statement

Topic depends on:

× Aggregate functions

Table of contents:

```
SELECT stock_name, MIN(datetime) as moment, MAX(price) as maximum
FROM stocks
GROUP BY stock_name;
```

stock_name	moment	maximum
WTI	2019-01-18 23:02	89.8
FB	1998-03-31 04:18	63.7
DJI	NULL	52.7

Here we see that NULL value forms a separate category because it is considered to be a unique value.
If there are several columns in the GROUP BY clause, each unique combination of values from these columns will be aggregated separately.

```
SELECT stock_name, datetime, MAX(price)
FROM stocks
GROUP BY stock_name, datetime;
```

stock_name	datetime	price
WTI	2020-05-17 11:00	89.8
FB	2020-04-11 10:23	26.3
WTI	2019-01-18 23:02	20.9
FB	NULL	15.6
DJI	2004-05-28 21:09	52.7
FB	1998-03-31 04:18	63.7

Two rows for WTI, 2019-01-18 23:02 got merged into one with the maximum price of 20.9. In terms of result, grouping query without any aggregate functions equals to:

```
SELECT DISTINCT stock_name, datetime
FROM stocks;
```

If a column is not mentioned in GROUP BY clause and there is at least one aggregate function being used in SELECT, this column can not be used in SELECT portion of the query without being wrapped in an aggregate function.

§2. HAVING keyword

Queries with a GROUP BY statement are regular in a way that they allow to filter rows with WHERE statement and to order them with ORDER BY statement. There is another clause that is especially helpful with grouping tasks – HAVING. If WHERE accepts conditions on values that certain cells have, HAVING does the same but for values of already computed aggregations. For example, let's select "stock-datetime" pairs with maximum price above 50:

```
SELECT stock_name, datetime, MAX(price) as maximum
FROM stock
GROUP BY stock_name, datetime
HAVING MAX(price) > 50;
```

stock_name	datetime	maximum
WTI	2020-05-17 11:00	89.8
DJI	2004-05-28 21:09	52.7
FB	1998-03-31 04:18	63.7

A fair question would be "why can't we utilize WHERE for the filtering"? The reason for that is the order of evaluation of the statements:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

All the conditions that you put in HAVING have to relate to aggregation functions. However, besides that there are no other special restrictions. In most relational database management systems it is not allowed to use aliases for columns in HAVING clause but in MySQL it is perfectly fine to write "HAVING maximum > 50".

§3. Conclusion

Summing up, the template for queries with grouping is the following:

```
SELECT column_name [, list_of_other_columns]
     , aggregation [, list_of_aggregations]
FROM table_name
[WHERE list_of_conditions]
GROUP BY column_name [, list_of_other_columns]
[HAVING list_of_aggregate_conditions]
[ORDER BY list_of_columns/aliases];
```

GROUP BY is arguably one of the most common SQL statements. It would be quite hard to find a relatively serious SQL code base without grouping queries. Now let's go straight to the tasks to check how you've comprehended this topic.

Report a typo

38 users liked this theory. **3** didn't like it. **What about you?**

Start practicing

Comments (5) Hints (0) Useful links (1) Show discussion