MySQL_Exercise_06_Common_Pitfalls_of_Grouped_Queries

July 12, 2020

1 MySQL Exercise 6: Common Pitfalls of Gouped Queries

There are two main reasons grouped queries can cause problems, especially in MySQL:

1) MySQL gives the user the benefit of the doubt, and assumes we don't make (at least some kinds of) mistakes. Unfortunately, we do make those mistakes.

2) We commonly think about data as spreadsheets that allow you make calculations across rows and columns, and that allow you to keep both raw and aggregated data in the same spreadsheet. Relational databases don't work that way.

The way these issues cause problems are:

1) When we are working with a MySQL database, we incorrectly interpret nonsensical output from illogical queries, or

2) When we are working with a non-MySQL database platform, we struggle with trying to make queries that will never work because they ask for both aggregated and non-aggregated data.

In this lesson, we will learn what these issues look like.

1.1 1. Misinterpretations due to Aggregation Mismatches

Begin by loading the SQL library, connecting to the Dognition database, and setting the Dognition database as the default.

[1]:
```
%load_ext sql
%sql mysql://studentuser:studentpw@localhost/dognitiondb
%sql USE dognitiondb
```

```
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

[1]: []

Imagine that we would like to retrieve, for each breed_type in the Dognition database, the number of unique dog_guids associated with that breed_type and their weight. Let's try to write a query that reflects that request:

```
SELECT breed_type, COUNT(DISTINCT dog_guid) AS NumDogs, weight
FROM dogs
GROUP BY breed_type;
```

Now take a look at the output:

[2]:
```
%%sql SELECT breed_type, COUNT(DISTINCT dog_guid) AS NumDogs, weight
FROM dogs
GROUP BY breed_type;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
4 rows affected.
```

[2]:
```
[('Cross Breed', 5568, 0),
 ("Mixed Breed/ Other/ I Don't Know", 9499, 50),
 ('Popular Hybrid', 1160, 70),
 ('Pure Breed', 18823, 50)]
```

You immediately notice a few things: (1) the query accurately represents the fields I said I wanted; (2) the query executed without errors! Wonderful! (3) Cross Breed dogs weigh 0 pounds; and (4) the grammar of the sentence describing what I said I wanted seems a little confusing: "We would like to retrieve, for *each breed_type* in the Dognition database, *the number of* unique dog_guids associated with that breed_type and *their weight*."

All of these things you noticed are related. Let's address them in reverse order.

What's wrong with the sentence I wrote? One of the things I said I wanted was *the number of* unique dog_guids. This is a single number. I also said I wanted "their weight." "Their" implies many weight measurements, not one measurement. In order to make my grammar correct, I need my description of dog_guids and weight to either both be singular or both be plural. To make the logic behind the sentence make sense, I have to do a similar thing: either dog_guids and weight both need to be aggregated or dog_guids and weight both need to be non-aggregated.

It's useful to remember that SQL output is always a table. How could you construct a valid table that would have columns for aggregate counts and individual weight measurements at the same time? The answer is, you can't. One option is to disaggregate the count so that you have one column with dog_guids and another column with weight measurements for each dog_guid. The only other option is to aggregate the weight measurements so that you have one column with the total count of dog_guids and another column with the average (or some other kind of summary aggregation) weight measurement for the group the count represents.

That brings us to the next phenomenon we observed: Cross Breed dogs weigh 0 pounds. Well, unless the laws of gravity and physics have changed, that's not possible. Something strange must be happening in the weight field.

We've established that the question I posed and the query I executed don't make logical sense, yet the MySQL query did run! If there is no way to make a tablular output that fits what I asked for, what is MySQL outputting for us?

It turns out that MySQL resolves my poor query by choosing its own way to "summarize" the unaggregated field, which in our case is "weight." Rather than run an aggregation function that takes all the values in the weight column into account, though, it unpredictably populates the weight output column with one value from all the possible weight values within a given breed_type subset. Which value it chooses will be different depending on the original order of the raw data and the configuration of a database. This flexibility is very convenient when you know that all the values in a non-aggregated column are the same for the subsets of the data that correspond to the variable by which you are grouping. In fact, the visualization software Tableau (which is based in SQL language) recognized how frequently this type of situation arises and came up with a custom solution for its customers. Tableau incorprated an aggregation-like function called "ATTR" into its interface to let users say "I'm using an aggregation function here because SQL says I have to, but I know that this is a situation where all of the rows in each group will have the same value."

Tableau's approach is helpful because it forces users to acknowledge that a field in a query is supposed to be aggregated, and Tableau's formulas will crash if all the rows in a group do not have the same value. MySQL doesn't force users to do this. MySQL trusts users to know what they are doing, and will provide an output even if all the rows in a group do not have the same value. Unfortunately, this approach can cause havoc if you aren't aware of what you are asking MySQL to do and aren't familiar with your data.

Let's see a couple more first-hand examples of this tricky GROUP BY behavior. Let's assume you want to know the number of each kind of test completed in different months of the year.

You execute the following query:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
FROM complete_tests
GROUP BY test_name
ORDER BY test_name ASC, Month ASC;
```

Question 1: What does the Month column represent in this output? Take a look and see what you think:

[3]:
```
%%sql SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS␣
 ↪Num_Completed_Tests
FROM complete_tests
GROUP BY test_name
ORDER BY test_name ASC, Month ASC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
40 rows affected.
```

[3]:
```
[('1 vs 1 Game', 6, 255),
 ('3 vs 1 Game', 5, 368),
 ('5 vs 1 Game', 5, 620),
 ('Arm Pointing', 2, 11452),
 ('Cover Your Eyes', 2, 7250),
 ('Delayed Cup Game', 2, 5271),
 ('Different Perspective', 11, 89),
 ('Expression Game', 10, 124),
```

```
('Eye Contact Game', 2, 14545),
('Eye Contact Warm-up', 2, 16238),
('Foot Pointing', 2, 9751),
('Impossible Task Game', 4, 532),
('Impossible Task Warm-up', 4, 539),
('Inferential Reasoning Game', 2, 4980),
('Inferential Reasoning Warm-up', 2, 5098),
('Memory versus Pointing', 2, 6525),
('Memory versus Smell', 2, 6163),
('Navigation Game', 3, 927),
('Navigation Learning', 3, 1054),
('Navigation Warm-up', 3, 1299),
('Numerosity Warm-Up', 5, 382),
('One Cup Warm-up', 2, 6785),
('Physical Reasoning Game', 2, 4255),
('Physical Reasoning Warm-up', 2, 4761),
('Self Control Game', 9, 132),
('Shaker Game', 2, 77),
('Shaker Warm-Up', 2, 81),
('Shared Perspective', 11, 83),
('Slide', 8, 147),
('Smell Game', 7, 168),
('Stair Game', 12, 112),
('Switch', 8, 137),
('Treat Warm-up', 2, 11737),
('Turn Your Back', 2, 7428),
('Two Cup Warm-up', 2, 6681),
('Warm-Up', 6, 873),
('Watching', 2, 8808),
('Watching - Part 2', 2, 7044),
('Yawn Game', 2, 19612),
('Yawn Warm-up', 2, 20863)]
```

Now try a similar query, but GROUP BY Month instead of test_name:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
FROM complete_tests
GROUP BY Month
ORDER BY Month ASC, test_name ASC;
```

Qustion 2: What does test_name mean in this case? Try it out:

[5]:
```
%%sql SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS␣
 ↪Num_Completed_Tests
FROM complete_tests
GROUP BY Month
ORDER BY Month ASC, test_name ASC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
```

```
12 rows affected.
```

```
[5]: [('Delayed Cup Game', 1, 11068),
      ('Yawn Warm-up', 2, 9122),
      ('Yawn Warm-up', 3, 9572),
      ('Physical Reasoning Game', 4, 7130),
      ('Delayed Cup Game', 5, 21013),
      ('Foot Pointing', 6, 23381),
      ('Eye Contact Game', 7, 15977),
      ('Memory versus Smell', 8, 13382),
      ('Yawn Warm-up', 9, 19853),
      ('Yawn Warm-up', 10, 39237),
      ('Inferential Reasoning Warm-up', 11, 12652),
      ('Inferential Reasoning Warm-up', 12, 10859)]
```

It looks like in both of these cases, MySQL is likely populating the unaggregated column with the first value it finds in that column within the first "group" of rows it is examining.

So how do we prevent this from happening?

> The only way to be sure how the MySQL database will summarize a set of data in a SELECT clause is to tell it how to do so with an aggregate function.<mark>

I should have written my original request to read:

"I would like to know, for *each breed type* of dog, *the number of* unique Dog_Guids there are in the Dognition database and *the breed_type's average weight*."

The query that would have reflected this sentence would have executed an aggregate function for both Dog_Guids and weight. The output of these aggregate functions would be unambiguous, and would easily be represented in a single table.

1.2 2. Errors due to Aggregation Mismatches

It is important to note that the issues I described above are the consequence of mismatching aggregate and non-aggregate functions through the GROUP BY clause in MySQL, but other databases manifest the problem in a different way. Other databases won't allow you to run the queries described above at all. When you try to do so, you get an error message that sounds something like:

```
Column 'X' is invalid in the select list because it is not contained in either an aggregate fu
```

Especially when you are just starting to learn MySQL, these error messages can be confusing and infuriating. A good discussion of this problem can be found here:

http://weblogs.sqlteam.com/jeffs/archive/2007/07/20/but-why-must-that-column-be-contained-in-an-aggregate.aspx

As a way to prevent these logical mismatches or error messages, you will often hear a rule that "every non-aggregated field that is listed in the SELECT list *must* be listed in the GROUP BY list." You have just seen that this rule is not true in MySQL, which makes MySQL both more

flexible and more tricky to work with. However, it is a useful rule of thumb for helping you avoid unknown mismatch errors.

1.3 3. By the way, even if you want to, there is no way to intentionally include aggregation mismatches in a single query

You might want to know the total number of unique User_Guids in the Dognition database, and in addition, the total number of unique User_Guids and average weight associated with each breed type. Given that you want to see the information efficiently to help you make decisions, you would like all of this information in one output. After all, that would be easy to do in Excel, given that all of this information could easily be summarized in a single worksheet.

To retrieve this information, you try one of the queries described above. Since you know the rule describing the relationship between fields in the SELECT and GROUP BY clauses, you write:

```
SELECT COUNT(DISTINCT dog_guid), breed_type, AVG(weight) AS avg_weight,
FROM dogs
GROUP BY breed_type;
```

The output to your query gives you four rows with the correct information, but it doesn't give you a count of the entire table without the groups being applied. Surely there must be a way to write a sophisticated query that can put these two pieces of information together for you, right?

Hopefully the discussion in the section above has already made it clear that the answer to this has to be "no." The output of every SQL query is a table. Can you think of a single table that could logically contain aggregated and non-aggregated data? You could put both types of information in an Excel worksheet, but not a single table.

There's yet another more practical reason the information you want can't be selected in a single query. The order of SQL queries is meant to reflect the way we write sentences, but in actuality they are actually executed in a different order than we write them. The cartoon below shows the order we write the queries being sent to the database at the top of the funnel, and the order the database usually executes the queries on the conveyer belt.

This diagram shows you that data are actually grouped before the SELECT expressions are applied. That means that when a GROUP BY expression is included in an SQL query, there is no way to use a SELECT statement to summarize data that cross multiple groups. The data will have already been separated by the time the SELECT statement is applied. The only way to get the information you want is to write two separate queries. This concept can be difficult to understand when you start using SQL for the first time after exclusively using Excel, but soon you will be come accustomed to it.

By the way, this diagram also shows you why some platforms and some queries in some platforms crash when you try to use aliases or derived fields in WHERE, GROUP BY, or HAVING clauses. If the SELECT statement hasn't been run yet, the alias or derived fields won't be available (as a reminder, some database systems–like MySQL–have found ways to overcome this issue). On the other hand, SELECT is executed before ORDER BY clauses. That means most database systems should be able to use aliases and derived fields in ORDER BY clauses.

1.4 Now that you are knowledgeable about the common pitfalls caused by GROUP BY, you are ready to perform one of the most powerful and fundamental utlities of a relational database: JOINS! Watch the next video to learn more about how joins work.

[ ]: