MySQL Blog Archive

For the latest blogs go to blogs.oracle.com/mysql

MySQL EXPLAIN ANALYZE

Posted on Oct 17, 2019 by Norvald H. Ryeng
Category: Optimizer
Tags: optimizer, performance, sql

MySQL 8.0.18 was just released, and it contains a brand new feature to analyze and understand how queries are executed: EXPLAIN ANALYZE.

What is it?

EXPLAIN ANALYZE is a profiling tool for your queries that will show you where MySQL spends time on your query and why. It will plan the query, instrument it and execute it while counting rows and measuring time spent at various points in the execution plan. When execution finishes, EXPLAIN ANALYZE will print the plan and the measurements instead of the query result.

This new feature is built on top of the regular EXPLAIN query plan inspection tool, and can be seen as an extension of the EXPLAIN FORMAT=TREE that was added earlier in MySQL 8.0. In addition to the query plan and estimated costs, which a normal EXPLAIN will print, EXPLAIN ANALYZE also prints the actual costs of individual iterators in the execution plan.

How do I use it?

As an example, we'll use data from the Sakila Sample Database and a query that lists the total amount each staff member has rung up in August 2005. The query is straight forward:

```
1  SELECT first_name, last_name, SUM(amount) AS total
2  FROM staff INNER JOIN payment
3    ON staff.staff_id = payment.staff_id
4       AND
5       payment_date LIKE '2005-08%'
6  GROUP BY first_name, last_name;
7
8  +------------+-----------+----------+
9  | first_name | last_name | total    |
10 +------------+-----------+----------+
11 | Mike       | Hillyer   | 11853.65 |
12 | Jon        | Stephens  | 12218.48 |
13 +------------+-----------+----------+
14 2 rows in set (0,02 sec)
```

There are only two people, Mike and Jon, and we get the total amount in August 2005 for each of them.

An EXPLAIN FORMAT=TREE will show us the query plan and cost estimates:

```
1  EXPLAIN FORMAT=TREE
2  SELECT first_name, last_name, SUM(amount) AS total
3  FROM staff INNER JOIN payment
4    ON staff.staff_id = payment.staff_id
5       AND
6       payment_date LIKE '2005-08%'
7  GROUP BY first_name, last_name;
8
9  -> Table scan on <temporary>
10     -> Aggregate using temporary table
11         -> Nested loop inner join  (cost=1757.30 rows=1787)
12             -> Table scan on staff  (cost=3.20 rows=2)
13             -> Filter: (payment.payment_date like '2005-08%')  (cost=117.43 rows=894)
14                 -> Index lookup on payment using idx_fk_staff_id (staff_id=staff.staff_id)  (cost=117.43 rows=8043)
```

But it doesn't tell us if those estimates are correct, or on which operations in the query plan the time is actually spent. EXPLAIN ANALYZE will do that:

```
1  EXPLAIN ANALYZE
2  SELECT first_name, last_name, SUM(amount) AS total
3  FROM staff INNER JOIN payment
4    ON staff.staff_id = payment.staff_id
5       AND
6       payment_date LIKE '2005-08%'
7  GROUP BY first_name, last_name;
8
9  -> Table scan on <temporary>  (actual time=0.001..0.001 rows=2 loops=1)
10     -> Aggregate using temporary table  (actual time=58.104..58.104 rows=2 loops=1)
11         -> Nested loop inner join  (cost=1757.30 rows=1787) (actual time=0.816..46.135 rows=5687 loops=1)
12             -> Table scan on staff  (cost=3.20 rows=2) (actual time=0.047..0.051 rows=2 loops=1)
13             -> Filter: (payment.payment_date like '2005-08%')  (cost=117.43 rows=894) (actual time=0.464..22.767 rows=284
14                 -> Index lookup on payment using idx_fk_staff_id (staff_id=staff.staff_id)  (cost=117.43 rows=8043) (actu
```

There are several new measurements here:

- Actual time to get first row (in milliseconds)
- Actual time to get all rows (in milliseconds)
- Actual number of rows read
- Actual number of loops

Let's look at a specific example, the cost estimates and actual measurements of the filtering iterator that picks out sales in August 2005 (line 13 in the EXPLAIN ANALYZE output above).

```
1 Filter: (payment.payment_date like '2005-08%')
2 (cost=117.43 rows=894)
3 (actual time=0.464..22.767 rows=2844 loops=2)
```

Our filter has an estimated cost of 117.43 and is estimated to return 894 rows. These estimates are made by the query optimizer before the query is executed, based on the available statistics. This information is also present in the EXPLAIN FORMAT=TREE output.

We'll start at the end with the number of loops. The number of loops for this filtering iterator is 2. What does that mean? To understand this number, we have to look what's above the filtering iterator in the query plan. On line 11 there's a nested loop join, and on line 12 there's a table scan on the staff table. This means that we're doing a nested loop join where we scan the staff table, and for each row in that table, we look up the corresponding entries in the payment table using an index lookup and a filtering on the payment date. Since there are two rows in the staff table (Mike and Jon), we get two loop iterations on the filtering, and on the index lookup on line 14.

For many people, the most interesting new information provided by EXPLAIN ANALYZE is the actual time, "0.464..22.767", which means that it took on average 0.464 ms to read the first row, and 22.767 ms to read all rows. On average? Yes, because of the looping, we have to time this iterator twice, and the numbers reported are the averages of all loop iterations. This means that the actual execution time of the filtering is twice these numbers. So if we look at the time to receive all rows one level up, in the nested loop iterator (line 11), it's 46.135 ms, a bit more than twice the time for one run of the filtering iterator.

The time reflects the time of the whole subtree rooted at the filtering operator, i.e., the time to read the rows using the index lookup iterator and then evaluate the condition that the payment date was in August 2005. If we look at the index loop iterator (line 14), we see that the corresponding numbers are 0.450 and 19.988 ms, respectively. That means that most of the time was spent reading the rows using index lookup, and that the actual filtering was relatively cheap compared to reading the data.

The actual number of rows read was 2844, while the estimate was 894 rows. So the optimizer missed by a factor 3. Again, because of the looping, both the estimated and actual numbers are averages over all loop iterations. If we look at the schema, there is no index or histogram on the payment_date column, so the statistics provided to the optimizer to compute the selectivity of the filter is limited. For an example where better statistics result in more accurate estimates, we can again look at the index lookup iterator. We see that the index has provided much more accurate statistics: an estimate of 8043 rows vs. 8024 actual rows read. That's pretty good. This happens because indexes come with extra statistics that just isn't there for non-indexed columns.

Search

Categories

- Backup (3)
- Character Sets (16)
- Connectors / Languages (14)
- Data Dictionary (11)
- Document Store, JSON (32)
- GIS (15)
- High Availability / Replication (182)
- InnoDB (57)
- Miscellaneous (84)
- Monitoring (18)
- NDB (69)
- Optimizer (56)
- Performance (5)
- Router (4)
- Security (57)
- Shell (30)
- Thread Pool (1)
- Upgrading (15)
- Windows / .NET (21)
- Workbench (46)

- News Announcements (200)
- Release Announcements (529)

So what can you do with this information? Analyzing queries and understanding why they perform poorly takes some practice. But some simple hints to get you started are:

- If you wonder what it's taking so long, look at the timing. Where does the executor spend its time?
- If you wonder why the optimizer chose that plan, look at the row counters. A large difference (i.e., a couple of orders of magnitude or more) between the estimated number of rows and the actual number of rows is a sign that you should look closer at it. The optimizer chooses its plan based on the estimate, but looking at the actual execution may tell you that another plan would have been better.

That's it! Another tool in the MySQL query analysis toolbox:

- To examine the query plan: EXPLAIN FORMAT=TREE
- To profile the query execution: EXPLAIN ANALYZE
- To understand plan choices: Optimizer trace

I hope you enjoyed this quick tour of the new feature, and that EXPLAIN ANALYZE will help you profile and understand your slow queries.