Multiple column index vs multiple indexes with MySQL 5.6

多列索引 vs 多个索引

Posted on: January 3, 2014 | **By:**

Stephane Combaudon

有关索引的一个问题：我们应该使用多列索引，还是多个单列索引?

A question often comes when talking about indexing: should we use multiple column indexes or multiple indexes on single columns? Peter Zaitsev wrote about it back in 2008 and the conclusion then was that a multiple column index is most often the best solution. But with all the recent optimizer improvements, is there anything different with MySQL 5.6?

2008年时的结论是：多列索引通常是最佳解决方案

Setup

For this test, we will use these 2 tables (same structure as in Peter's post):

Shell

```
CREATE TABLE t1000merge (
  id int not null auto_increment primary key,
  i int(11) NOT NULL,
  j int(11) NOT NULL,
  val char(10) NOT NULL,
  KEY i (i),
  KEY j (j)
) ENGINE=InnoDB;

CREATE TABLE t1000idx2 (
  id int not null auto_increment primary key,
  i int(11) NOT NULL,
  j int(11) NOT NULL,
  val char(10) NOT NULL,
  KEY ij (i,j)
) ENGINE=InnoDB;
```

多个单列索引

多列索引

Tables were populated with 1M rows for this test, i and j have 1000 distinct values (independent of each other). The buffer pool is large enough to hold all data and indexes.

We will look at this query on MySQL 5.5.35 and MySQL 5.6.15:

```
Shell
1 SELECT sum(length(val)) FROM T WHERE j=2 AND i BETWEEN 100 and 200
```

Why this specific query? With MySQL 5.5, for t1000idx2, the optimizer estimates that the index on `(i,j)` is not selective enough and it falls back to a full table scan. While for t1000merge, the index on `(j)` is an obvious good candidate to filter efficiently. 有效地筛选

Consequently this query has a better response on t1000merge (**0.01s**) than on t1000idx2 (**0.45s**).

索引条件下推（ICP）

On MySQL 5.6, this query is a good candidate for index condition pushdown (ICP), so we can reasonably hope that response time for t1000idx2 will improve.

t1000idx2表的响应时间会提高

ICP: FORCE INDEX to the rescue

Unfortunately the optimizer still prefers the full table scan which gives us the same bad response time:

```
Shell
1 mysql5.6> EXPLAIN SELECT sum(length(val)) FROM t1000idx2 WHERE j=2 AND i BETWE
2 +----+-------------+-----------+------+---------------+------+---------+------
3 | id | select_type | table     | type | possible_keys | key  | key_len | ref
4 +----+-------------+-----------+------+---------------+------+---------+------
5 |  1 | SIMPLE      | t1000idx2 | ALL  | ij            | NULL | NULL    | NULL
6 +----+-------------+-----------+------+---------------+------+---------+------
```

And what if we use `FORCE INDEX`?

```
Shell
1 mysql5.6 > EXPLAIN SELECT sum(length(val)) FROM  t1000idx2 FORCE INDEX(ij) WHE
2 +----+-------------+-----------+-------+---------------+------+---------+-----
3 | id | select_type | table     | type  | possible_keys | key  | key_len | ref
4 +----+-------------+-----------+-------+---------------+------+---------+-----
5 |  1 | SIMPLE      | t1000idx2 | range | ij            | ij   | 8       | NULL
6 +----+-------------+-----------+-------+---------------+------+---------+-----
```

This time ICP is used (see "Using index condition" in the Extra field)!

And the difference in response time is impressive:

– Without `FORCE INDEX` (full table scan): **0.45s**

– With `FORCE INDEX` (multiple column index + index condition pushdown): **0.04s**, a 10x

使用"强制使用索引"语法（多列索引 + 索引条件下推）：0.04s，10倍性能提升

注意：若需要使用"强制使用索引"语法，极大可能说明索引存在优化空间！
本例索引优化：KEY ji (j,i)

improvement!

Additional thoughts 其他想法

It is interesting to see that the optimizer fails to find the best execution plan for this simple query. The optimizer trace sheds some light:

```shell
mysql> SET optimizer_trace="enabled=on";
mysql> SELECT sum(length(val)) FROM  T  WHERE j=2 AND i BETWEEN 100 and 200;
mysql> SELECT * FROM INFORMATION_SCHEMA.OPTIMIZER_TRACE;
[...]
"range_analysis": {
                    "table_scan": {
                      "rows": 1000545,
                      "cost": 202835
                    },
```

This is the estimated cost for a full table scan. 这是一个全表扫描的预估成本

Now we will see how the optimizer estimates the cost of the range scan using the `ij` index:

```shell
[...]
"range_scan_alternatives": [
                        {
                          "index": "ij",
                          "ranges": [
                            "100 <= i <= 200"
                          ],
                          "index_dives_for_eq_ranges": true,
                          "rowid_ordered": false,
                          "using_mrr": false,
                          "index_only": false,
                          "rows": 188460,
                          "cost": 226153,
                          "chosen": false,
                          "cause": "cost"
                        }
                      ]
[...]
```

At this stage the optimizer does not know if ICP can be used. This probably explains why the cost of the range scan is overestimated.

If we look at the optimizer trace for the query with the `FORCE INDEX` hint, ICP is only detected after the range scan is chosen:

```shell
[...]
"refine_plan": [
                {
```

```
                "table": "`t1000idx2` FORCE INDEX (`ij`)",
                "pushed_index_condition": "((`t1000idx2`.`j` = 2) and (`t1000ic
                "table_condition_attached": null,
                "access_type": "range"
            }
[...]
```

Conclusion 结论：多列索引 更加高效！

Multiple column index vs multiple indexes? Having indexes on single columns often lead to the optimizer using the index_merge access type, which is typically not as good as accessing a single index on multiple columns. MySQL 5.6 makes multiple column indexes more efficient than before with index condition pushdown.

But don't forget that the optimizer is not perfect: you may have to use index hints to benefit from this feature.

Stephane Combaudon

Stéphane joined Percona in July 2012, after working as a MySQL DBA for leading French companies such as Dailymotion and France Telecom. In real life, he lives in Paris with his wife and their twin daughters. When not in front of a computer or not spending time with his family, he likes playing chess and hiking.

Tags:

Index Condition Pushdown, MySQL 5.5 vs. MySQL 5.6, MySQL 5.6, Optimizer

Categories:

Insight for DBAs, MySQL

Comments

Jørgen Løland

January 4, 2014 at 2:29 am

Hi Stephane,

Great blog post. Your observations are correct – the effect of ICP is not taken into account when the optimizer decides which access method to use, it is just added later if an applicable index has been chosen. Making the optimizer ICP aware on an earlier stage is on our TODO list but it would be really nice to track this in a public feature request. Can you please one on bugs.mysql.com?

Btw, the best index for this query would be "KEY (j, i)". It would be able to use both keyparts with the range access method and outperform the execution plan using ICP above. I'm sure you're already aware of this and just wanted to make a point 🙂 For details, see http://jorgenloland.blogspot.com/2011/08/mysql-range-access-method-explained.html and http://jorgenloland.blogspot.com/2012/03/index-condition-pushdown-to-rescue.html

使用范围访问方法的所有键部分比使用 ICP 的执行计划还要好

Reply ↓

Morgan Tocker

I've filed a bug report here: http://bugs.mysql.com/bug.php?id=71302

Reply ↓

January 5, 2014 at 10:29 am

Stephane Combaudon **Post author**

January 6, 2014 at 3:35 am

@David: fixed!

@Jørgen: thanks for confirming my observations. And you're right, I forgot to mention which index would be the best one!

@Morgan: thanks!

Reply ↓

Elaine Lindelef

January 6, 2014 at 12:07 pm

We have been seeing an odd situation in replication where occasionally the optimizer is making terribly bad choices on the slave database. If you run the exact same select on the master, it runs fine, and with an appropriate explain. Run it on the slave and the explain is all wrong. The statistics on the two tables are slightly different. Fixing the statistics (either via intervention or self-healing) fixes the problem, and we could add an index hint. But why would the statistics get to be different on master and slave?

Reply ↓

@Elaine: It is a difficult question to answer, since

Morgan Tocker

January 7, 2014 at 11:23 am

not all queries use statistics, but also InnoDB prior to MySQL 5.6 had far less accurate statistics, so versions will matter here.

Stephane is also demonstrating the optimizer trace feature (new to 5.6) which can show why various plan alternatives were not taken. This might help in your case – http://dev.mysql.com/doc/internals/en/optimizer-tracing-typical-usage.html

Reply ↓

Stephane Combaudon **Post author**

January 8, 2014 at 5:29 am

@Elaine: Execution plan instability is typically very hard to understand because before 5.6 and the optimizer trace it was really difficult to know why the optimizer chose plan A and not plan B.

If only a few queries are affected, your best option is to use index hints. If it affects many queries, it probably means that you're doing something uncommon (but not necessarily bad).

Reply ↓