03 快速学会分析SQL执行效率（下）

更新时间：2019-08-12 11:14:57

> 当你做成功一件事，千万不要等待着享受荣誉，应该再做那些需要的事。
>
> —— 巴斯德

在上一节我们学习了定位慢 SQL 及使用 explain 分析慢 SQL，我们也提到了分析慢 SQL 还有 show profile 和 trace 等方法，本节就重点补充学习这两种方法。

1 show profile 分析慢查询

有时需要确定 SQL 到底慢在哪个环节，此时 explain 可能不好确定。在 MySQL 数据库中，通过 profile，能够更清楚地了解 SQL 执行过程的资源使用情况，能让我们知道到底慢在哪个环节。

> 知识扩展：可以通过配置参数 profiling = 1 来启用 SQL 分析。该参数可以在全局和 session 级别来设置。对于全局级别则作用于整个MySQL 实例，而 session 级别仅影响当前 session 。该参数开启后，后续**执行的 SQL 语句都将记录其资源开销，如 IO、上下文切换、CPU、Memory**等等。根据这些开销进一步分析当前 SQL 从而进行优化与调整。

下面我们来讲一下如何使用 profile 分析慢查询，大致步骤是：确定这个 MySQL 版本是否支持 profile；确定 profile 是否关闭；开启 profile；执行 SQL；查看执行完 SQL 的 query id；通过 query id 查看 SQL 的每个状态及耗时时间。

1.1 确定是否支持 profile

我们进行第一步，用下面命令来判断当前 MySQL 是否支持 profile：

```
mysql> select @@have_profiling;

+------------------+
| @@have_profiling |
+------------------+
| YES              |
+------------------+

1 row in set, 1 warning (0.00 sec)
```

从上面结果中可以看出是YES，表示支持profile的。

1.2 查看 profiling 是否关闭的

进行第二步，用下面命令判断 profiling 参数是否关闭（默认 profiling 是关闭的）：

```
mysql> select @@profiling;

+-------------+
| @@profiling |
+-------------+
|           0 |
+-------------+

1 row in set, 1 warning (0.00 sec)
```

结果显示为 0，表示 profiling 参数状态是关闭的。

1.3 通过 set 开启 profile

```
mysql> set profiling=1;

Query OK, 0 rows affected, 1 warning (0.00 sec)
```

Tips：set 时没加 global，只对当前 session 有效。

1.4 执行 SQL 语句

```
mysql> select * from t1 where b=1000;
```

1.5 确定 SQL 的 query id

通过 show profiles 语句确定执行过的 SQL 的 query id：

```
mysql> show profiles;
+----------+------------+--------------------------------+
| Query_ID | Duration   | Query                          |
+----------+------------+--------------------------------+
|        1 | 0.00063825 | select * from t1 where b=1000 |
+----------+------------+--------------------------------+
1 row in set, 1 warning (0.00 sec)
```

1.6 查询 SQL 执行详情

通过 show profile for query 可看到执行过的 SQL 每个状态和消耗时间：

```
mysql> show profile for query 1;
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| starting             | 0.000115 |
| checking permissions | 0.000013 |
| Opening tables       | 0.000027 |
| init                 | 0.000035 |
| System lock          | 0.000017 |
| optimizing           | 0.000016 |
| statistics           | 0.000025 |
| preparing            | 0.000020 |
| executing            | 0.000006 |
| Sending data         | 0.000294 |
| end                  | 0.000009 |
| query end            | 0.000012 |
| closing tables       | 0.000011 |
| freeing items        | 0.000024 |
| cleaning up          | 0.000016 |
+----------------------+----------+
15 rows in set, 1 warning (0.00 sec)
```

通过以上结果，可以确定 SQL 执行过程具体在哪个过程耗时比较久，从而更好地进行 SQL 优化与调整。

2 trace 分析 SQL 优化器

从前面学到了 explain 可以查看 SQL 执行计划，但是无法知道它为什么做这个决策，如果想确定多种索引方案之间是如何选择的或者排序时选择的是哪种排序模式，有什么好的办法吗？

从 MySQL 5.6 开始，可以使用 trace 查看优化器如何选择执行计划。

通过trace，能够进一步了解为什么优化器选择A执行计划而不是选择B执行计划，或者知道某个排序使用的排序模式，帮助我们更好地理解优化器行为。

如果需要使用，先开启 trace，设置格式为 JSON，再执行需要分析的 SQL，最后查看 trace 分析结果（在 information_schema.OPTIMIZER_TRACE 中）。

开启该功能，会对 MySQL 性能有所影响，因此只建议分析问题时临时开启。

下面一起来看下 trace 的使用方法。使用讲解 explain 时创建的表t1做实验。

首先构造如下 SQL (表示取出表 t1 中 a 的值大于 900 并且 b 的值大于 910 的数据，然后按照 a 字段排序)：

```
select * from t1 where a >900 and b > 910 order  by a;
```

我们首先用 explain 分析下执行计划：

```
mysql> explain select * from t1 where a >900 and b > 910 order  by a;
+----+-------------+-------+------------+-------+---------------+-------+---------+------+------+----------+-----------------------------------------------------------+
| id | select_type | table | partitions | type  | possible_keys | key   | key_len | ref  | rows | filtered | Extra                                                     |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+------+----------+-----------------------------------------------------------+
|  1 | SIMPLE      | t1    | NULL       | range | idx_a,idx_b   | idx_b | 5       | NULL |   90 |    10.00 | Using index condition; Using where; Using filesort |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+------+----------+-----------------------------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

通过上面执行计划中 key 这个字段可以看出，该语句使用的是 b 字段的索引 idx_b。实际表 t1 中，a、b 两个字段都有索引，为什么条件中有这两个索引字段却偏偏选了 b 字段的索引呢？这时就可以使用 trace 进行分析。大致步骤如下：

```
mysql> set session optimizer_trace="enabled=on",end_markers_in_json=on;
/* optimizer_trace="enabled=on" 表示开启 trace；end_markers_in_json=on 表示 JSON 输出开启结束标记 */
```

```
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t1 where a >900 and b > 910 order  by a;
+------+------+------+
| id   | a    | b    |
+------+------+------+
|    1 |    1 |    1 |
|    2 |    2 |    2 |

......

| 1000 | 1000 | 1000 |
+------+------+------+
1000 rows in set (0.00 sec)

mysql> SELECT * FROM information_schema.OPTIMIZER_TRACE\G
*************************** 1. row ***************************
QUERY: select * from t1 where a >900 and b > 910 order  by a    --SQL语句
TRACE: {
  "steps": [
    {
      "join_preparation": {    --SQL准备阶段
        "select#": 1,
        "steps": [
          {
            "expanded_query": "/* select#1 */ select `t1`.`id` AS `id`,`t1`.`a` AS `a`,`t1`.`b` AS `b`,`t1`.`create_time` AS `create_time`,`t1`.`update_time` AS `u
pdate_time` from `t1` where ((`t1`.`a` > 900) and (`t1`.`b` > 910)) order by `t1`.`a`"
          }
        ] /* steps */
      } /* join_preparation */
    },
    {
      "join_optimization": {   --SQL优化阶段
        "select#": 1,
        "steps": [
          {
            "condition_processing": {    --条件处理
              "condition": "WHERE",
              "original_condition": "((`t1`.`a` > 900) and (`t1`.`b` > 910))",        --原始条件
              "steps": [
                {
                  "transformation": "equality_propagation",
                  "resulting_condition": "((`t1`.`a` > 900) and (`t1`.`b` > 910))"   --等值传递转换
                },
                {
                  "transformation": "constant_propagation",
                  "resulting_condition": "((`t1`.`a` > 900) and (`t1`.`b` > 910))"       --常量传递转换
                },
                {
                  "transformation": "trivial_condition_removal",
                  "resulting_condition": "((`t1`.`a` > 900) and (`t1`.`b` > 910))"        --去除没有的条件后的结构
                }
              ] /* steps */
            } /* condition_processing */
          },
          {
            "substitute_generated_columns": {
            } /* substitute_generated_columns */   --替换虚拟生成列
          },
          {
            "table_dependencies": [  --表依赖详情
              {
                "table": "`t1`",
                "row_may_be_null": false,
                "map_bit": 0,
                "depends_on_map_bits": [
                ] /* depends_on_map_bits */
              }
            ] /* table_dependencies */
```

```
},
{
  "ref_optimizer_key_uses": [
  ] /* ref_optimizer_key_uses */
},
{
  "rows_estimation": [ --预估表的访问成本
    {
      "table": "`t1`",
      "range_analysis": {
        "table_scan": {
          "rows": 1000,       --扫描行数
          "cost": 207.1       --成本
        } /* table_scan */,
        "potential_range_indexes": [    --分析可能使用的索引
          {
            "index": "PRIMARY",
            "usable": false,       --为false，说明主键索引不可用
            "cause": "not_applicable"
          },
          {
            "index": "idx_a",      --可能使用索引idx_a
            "usable": true,
            "key_parts": [
              "a",
              "id"
            ] /* key_parts */
          },
          {
            "index": "idx_b",      --可能使用索引idx_b
            "usable": true,
            "key_parts": [
              "b",
              "id"
            ] /* key_parts */
          }
        ] /* potential_range_indexes */,
        "setup_range_conditions": [
        ] /* setup_range_conditions */,
        "group_index_range": {
          "chosen": false,
          "cause": "not_group_by_or_distinct"
        } /* group_index_range */,
        "analyzing_range_alternatives": { --分析各索引的成本
          "range_scan_alternatives": [
            {
              "index": "idx_a", --使用索引idx_a的成本
              "ranges": [
                "900 < a"   --使用索引idx_a的范围
              ] /* ranges */,
              "index_dives_for_eq_ranges": true, --是否使用index dive（详细描述请看下方的知识扩展）
              "rowid_ordered": false, --使用该索引获取的记录是否按照主键排序
              "using_mrr": false,   --是否使用mrr
              "index_only": false,    --是否使用覆盖索引
              "rows": 100,           --使用该索引获取的记录数
              "cost": 121.01,        --使用该索引的成本
              "chosen": true         --可能选择该索引
            },
            {
              "index": "idx_b",      --使用索引idx_b的成本
              "ranges": [
                "910 < b"
              ] /* ranges */,
              "index_dives_for_eq_ranges": true,
              "rowid_ordered": false,
              "using_mrr": false,
              "index_only": false,
              "rows": 90,
              "cost": 109.01,
```

```
            "chosen": true            --也可能选择该索引
           }
          ] /* range_scan_alternatives */,
          "analyzing_roworder_intersect": { --分析使用索引合并的成本
           "usable": false,
           "cause": "too_few_roworder_scans"
          } /* analyzing_roworder_intersect */
         } /* analyzing_range_alternatives */,
         "chosen_range_access_summary": {  --确认最优方法
          "range_access_plan": {
           "type": "range_scan",
           "index": "idx_b",
           "rows": 90,
           "ranges": [
            "910 < b"
           ] /* ranges */
          } /* range_access_plan */,
          "rows_for_plan": 90,
          "cost_for_plan": 109.01,
          "chosen": true
         } /* chosen_range_access_summary */
        } /* range_analysis */
       }
      ] /* rows_estimation */
    },
    {
      "considered_execution_plans": [  --考虑的执行计划
       {
        "plan_prefix": [
        ] /* plan_prefix */,
        "table": "`t1`",
        "best_access_path": {          --最优的访问路径
         "considered_access_paths": [ --决定的访问路径
          {
           "rows_to_scan": 90,      --扫描的行数
           "access_type": "range",  --访问类型：为range
           "range_details": {
            "used_index": "idx_b"  --使用的索引为：idx_b
           } /* range_details */,
           "resulting_rows": 90,    --结果行数
           "cost": 127.01,          --成本
           "chosen": true,      --确定选择
           "use_tmp_table": true
          }
         ] /* considered_access_paths */
        } /* best_access_path */,
        "condition_filtering_pct": 100,
        "rows_for_plan": 90,
        "cost_for_plan": 127.01,
        "sort_cost": 90,
        "new_cost_for_plan": 217.01,
        "chosen": true
       }
      ] /* considered_execution_plans */
    },
    {
      "attaching_conditions_to_tables": {  --尝试添加一些其他的查询条件
       "original_condition": "((`t1`.`a` > 900) and (`t1`.`b` > 910))",
       "attached_conditions_computation": [
       ] /* attached_conditions_computation */,
       "attached_conditions_summary": [
        {
         "table": "`t1`",
         "attached": "((`t1`.`a` > 900) and (`t1`.`b` > 910))"
        }
       ] /* attached_conditions_summary */
      } /* attaching_conditions_to_tables */
    },
    {
```

```
          "clause_processing": {
            "clause": "ORDER BY",
            "original_clause": "`t1`.`a`",
            "items": [
              {
                "item": "`t1`.`a`"
              }
            ] /* items */,
            "resulting_clause_is_simple": true,
            "resulting_clause": "`t1`.`a`"
          } /* clause_processing */
        },
        {
          "reconsidering_access_paths_for_index_ordering": {
            "clause": "ORDER BY",
            "index_order_summary": {
              "table": "`t1`",
              "index_provides_order": false,
              "order_direction": "undefined",
              "index": "idx_b",
              "plan_changed": false
            } /* index_order_summary */
          } /* reconsidering_access_paths_for_index_ordering */
        },
        {
          "refine_plan": [          --改进的执行计划
            {
              "table": "`t1`",
              "pushed_index_condition": "(`t1`.`b` > 910)",
              "table_condition_attached": "(`t1`.`a` > 900)"
            }
          ] /* refine_plan */
        }
      ] /* steps */
    } /* join_optimization */
  },
  {
    "join_execution": {            --SQL执行阶段
      "select#": 1,
      "steps": [
        {
          "filesort_information": [
            {
              "direction": "asc",
              "table": "`t1`",
              "field": "a"
            }
          ] /* filesort_information */,
          "filesort_priority_queue_optimization": {
            "usable": false,             --未使用优先队列优化排序
            "cause": "not applicable (no LIMIT)"     --未使用优先队列排序的原因是没有limit
          } /* filesort_priority_queue_optimization */,
          "filesort_execution": [
          ] /* filesort_execution */,
          "filesort_summary": {          --排序详情
            "rows": 90,
            "examined_rows": 90,          --参与排序的行数
            "number_of_tmp_files": 0,     --排序过程中使用的临时文件数
            "sort_buffer_size": 115056,
            "sort_mode": "<sort_key, additional_fields>"   --排序模式（详解请看下方知识扩展）
          } /* filesort_summary */
        }
      ] /* steps */
    } /* join_execution */
  }
] /* steps */
}
MISSING_BYTES_BEYOND_MAX_MEM_SIZE: 0 --该字段表示分析过程丢弃的文本字节大小，本例为0，说明没丢弃任何文本
        INSUFFICIENT_PRIVILEGES: 0    --查看trace的权限是否不足，0表示有权限查看trace详情
```

```
1 row in set (0.00 sec)
----------------------------------------------
----------------------------------------------


mysql> set session optimizer_trace="enabled=off";
/* 及时关闭trace */
```

这里对上方的执行字段详细描述一下：

TRACE 字段中整个文本大致分为三个过程。

- 准备阶段：对应文本中的 join_preparation
- 优化阶段：对应文本中的 join_optimization
- 执行阶段：对应文本中的 join_execution

使用时，重点关注优化阶段和执行阶段。

由此例可以看出：

- 在 trace 结果的 analyzing_range_alternatives 这一项可以看到：使用索引 idx_a 的成本为 121.01，使用索引 idx_b 的成本为 109.01，显然使用索引 idx_b 的成本要低些，因此优化器选择了 idx_b 索引；
- 在 trace 结果的 filesort_summary 这一项可以看到：排序模式为<sort_key, additional_fields>，表示使用的是单路排序，即一次性取出满足条件行的所有字段，然后在 sort buffer 中进行排序。

> 知识扩展：
>
> **知识点一：MySQL 常见排序模式：**
>
> - < sort_key, rowid >双路排序（又叫回表排序模式）：是首先根据相应的条件取出相应的排序字段和可以直接定位行数据的行 ID，然后在 sort buffer 中进行排序，排序完后需要再次取回其它需要的字段；
> - < sort_key, additional_fields >单路排序：是一次性取出满足条件行的所有字段，然后在sort buffer中进行排序；
> - < sort_key, packed_additional_fields >打包数据排序模式：将 char 和 varchar 字段存到 sort buffer 中时，更加紧缩。
>
> 三种排序模式比较：
>
> 第二种模式相对第一种模式，避免了二次回表，可以理解为用空间换时间。由于 sort buffer 有限，如果需要查询的数据比较大的话，会增加磁盘排序时间，效率可能比第一种方式更低。
>
> MySQL 提供了一个参数：max_length_for_sort_data，当“排序的键值对大小” > max_length_for_sort_data 时，MySQL 认为磁盘外部排序的 IO 效率不如回表的效率，会选择第一种排序模式；否则，会选择第二种模式。
>
> 第三种模式主要解决变长字符数据存储空间浪费的问题。
>
> **知识点二：优化器在估计符合条件的行数时有两个选择：**
>
> - index diver：dive 到 index 中利用索引完成元组数的估算；特点是速度慢，但可以得到精确的值；

> - index statistics：使用索引的统计数值，进行估算；特点是速度快，但是值不一定准确。

3 总结

今天我们分享了 show profile 和 trace 的使用方法，我们来对比一下三种分析 SQL 方法的特点：

- explain：获取 MySQL 中 SQL 语句的执行计划，比如语句是否使用了关联查询、是否使用了索引、扫描行数等；
- profile：可以清楚了解到SQL到底慢在哪个环节；
- trace：查看优化器如何选择执行计划，获取每个可能的索引选择的代价。

三种方法各有其适用场景，如果你有其它分析 SQL 的工具，欢迎在留言区分享。

4 问题

在工作中，遇到慢查询你是如何去分析优化的？

5 参考资料

MySQL 5.7官方文档：

https://dev.mysql.com/doc/internals/en/tracing-example.html

}

02 快速学会分析SQL执行效率（上）

04 条件字段有索引，为什么查询也这么慢?