mysqlsql分析

课程内容

1. sql语句的错误使用
2. SQL语句优化之索引优化指标
3. 优化器的执行过程与分析
4. explain与profile结果如何分析

1. sql语句的错误使用

学员问题：可不可以多次查询优化查询效率 （针对join 或者 in ）

注意这个问题并没有很直接的正确答案，问题本身是模棱两可的 既可以多次查询也可以一起；

对于SQL的错误使用有两种情况就是过强和过弱 这是常常会可能出现的问题：

比如查询粉丝最多的前十个用户的文章总数

粉丝关注表：user_fans

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	0	
user_id	int(11)	NO		MUL	NULL

fans_id	int(11)	NO		MUL	NULL

查询的方法

方案一

```
select user_id from (
select
user_id,count(*) c
from
user_fans
group by
user_id
order by c desc limit 0,5
) as fans_count;

然后再通过循环查询出用户的id信息并输出

for (fans_count)

select count(*) from article where user_id = ?
```

对比：

1. 从MySQL执行的query数量来看第一种为 1 + 10 = 11 条，第二种为 1 + 1 =2
2. 对应的交互数量11 ： 2
3. io操作对比假设一条SQL就是一个IO，第一种最少有11次IO，第二种小余或等于11次IO；如果数据非常离散就是11次；
4. 复杂度：前缀的SQL不管只看第二条SQL；第一种简单一些，第二种就增加了group by
5. 数据库对于结果的返回地中是11次，第二种是2次，但是第二种方法中的第二次结果是第一次的10倍

6. 从应用程序的数据处理来看，第二种比第一种多了一个拼接photo_id的过程

根据上面的点做对比分析：

1. 由于MySQL不管客户端每次调教的query是相同还是不同，都需要进行完全解析这个动作主要消耗的资源师数据库主机的CPU，那么这里第一种方案和第二 种方案消耗CPU的比例是11:2. query语句的解析动作在整个query语句执行过程中的整体消耗的CPU比例是比较多的；
2. 网络交互对比：11:2
3. IO比 <= 1:1
4. 在数据量少的情况下分组性能问题不大

所以相对来说第二种会更好一点-》 这个情况就是过度的弱化SQL 或者说数据库的处理能力

但是如果说根据第二种方法把两条SQL合在一起呢？

```
select count(*) from article where user_id in (
select user_id from (
select
user_id,count(*) c
from
user_fans
group by
user_id
order by c desc limit 0,5
) as fans_count;
) group by user_id;
```

当然是可以执行，但是目前的这个SQL复杂度同比之前任何一条SQL都要复杂 还不是一点点的问题.....

从数据库的交互来说 2 ： 1，但是复杂来说不是一点点的大

之前其实有提过SQL语句会通过解析器分解为不同的令牌，任何生成解析树提供给优化器使用并对SQL进行

计算和优化，再去执行该计划，但是因为SQL的复杂度增加，对于优化器来说分析的难度也会增加 -- 执行的计算会更多。

粉丝关注及其用户信息统计：user_stat

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	0	
user_id	int(11)	NO		MUL	NULL
fans_count	int(11)	NO		MUL	NULL

2. explain如何分析

explain在之前的课程中我们已经有讲解，这里就不做说明.....

本次分析依旧使用我们之前课程的sql案例如下：

```
explain select * from staffs where sex = 1  and age = 20 and name = 'starsky';
explain select * from customers1 where city = "长沙" and gender = 0;
```

结果如下：

```
1  explain select * from staffs where sex = 1  and age = 20 and name = 'starsky';
```

信息 结果1 概况 状态

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	staffs	(Null)	ALL	(Null)	(Null)	(Null)	(Null)	5	20	Using where

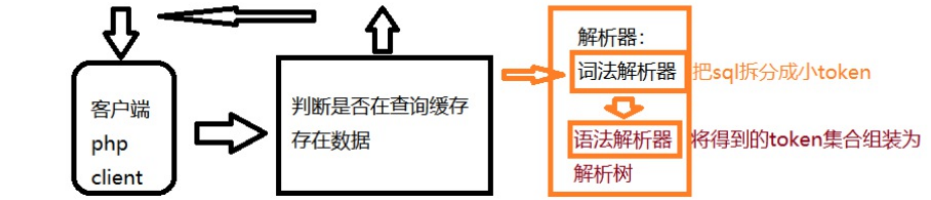
```
3  explain select * from customersls where city = "长沙" and gender = 0;
```

信息 结果1 概况 状态

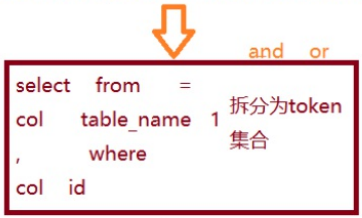
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	customer	(Null)	ALL	(Null)	(Null)	(Null)	(Null)	5089626	1	Using where

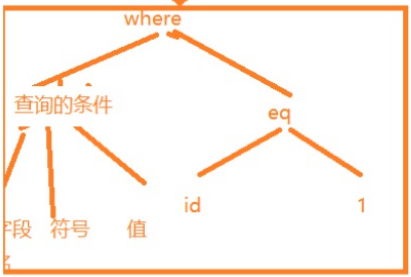
3. 优化器的执行过程与分析

SQL语句的执行流程：跳过线程分配，权限判断



select col1,col2 from table_name where id=1





优化器的重要功能：

1. 确认查询表（多对多的查询会选择合适的表做为查询依赖）
2. 对于数据表扫描查询字段（确定好查询的顺序）join
3. 重写where从句，去掉无意义的查询操作，尽可能去更改限制条件，减少查询范围
4. 判断索引的使用
   a. 是否覆盖了所有的字段
   b. 是否在where上使用，是否group order上使用
5. (join) 连接方式是否内还是外
6. (join) 尝试简化子查询=>合并视图

3.1 开启OPTIMIZER_TRACE

```
set optimizer_trace="enabled=on";--开启trace查看优化器的结果
set end_markers_in_json=on;--增加注释

select id,`name`,city,monthsalary,gender from customers1 where city="长沙" and gender=0 and
monthsalary=99;
select * from information_schema.optimizer_trace \G;
```

3.2 OPTIMIZER_TRACE结果

```
QUERY：查询语句
TRACE：QUERY字段对应语句的跟踪信息
MISSING_BYTES_BEYOND_MAX_MEM_SIZE：跟踪信息过长时，被截断的跟踪信息的字节数。
INSUFFICIENT_PRIVILEGES：执行跟踪语句的用户是否有查看对象的权限。当不具有权限时，该列信
息为1且TRACE字段为空，一般在调用带有SQL SECURITY DEFINER的视图或者是存储过程的情况下，
会出现此问题。
```

> join_preparation

join_preparation段落展示了准备阶段的执行过程。

```
{
  "join_preparation": {
    "select#": 1,
    "steps": [
      {
        -- 对比下原始语句，可以知道，这一步做了个格式化。
```



```
        "expanded_query": "/* select#1 */ select `salaries`.`emp_no` AS
`emp_no`,`salaries`.`salary` AS `salary`,`salaries`.`from_date` AS
`from_date`,`salaries`.`to_date` AS `to_date` from `salaries` where
((`salaries`.`from_date` = '1986-06-26') and (`salaries`.`to_date` = '1987-06-26'))"
      }
    ]
    /* steps */
  }
  /* join_preparation */
}
```

> join_optimization join_optimization展示了优化阶段的执行过程，是分析OPTIMIZER TRACE的重点。这段内容超级长，而且分了好多步骤，不妨按照步骤逐段分析：

> condition_processing 该段用来做条件处理，主要对WHERE条件进行优化处理。

```
"condition_processing": {
  "condition": "WHERE",
  "original_condition": "((`salaries`.`from_date` = '1986-06-26') and (`salaries`.`to_date`
= '1987-06-26'))",
  "steps": [
    {
      "transformation": "equality_propagation",
      "resulting_condition": "(multiple equal('1986-06-26', `salaries`.`from_date`) and
multiple equal('1987-06-26', `salaries`.`to_date`))"
    },
    {
      "transformation": "constant_propagation",
      "resulting_condition": "(multiple equal('1986-06-26', `salaries`.`from_date`) and
multiple equal('1987-06-26', `salaries`.`to_date`))"
```



```
    },
    {
      "transformation": "trivial_condition_removal",
      "resulting_condition": "(multiple equal(DATE'1986-06-26', `salaries`.`from_date`) and
multiple equal(DATE'1987-06-26', `salaries`.`to_date`))"
    }
  ] /* steps */
} /* condition_processing */
```

其中：

```
  condition：优化对象类型。WHERE条件句或者是HAVING条件句
  original_condition：优化前的原始语句
  steps：主要包括三步，分别是quality_propagation（等值条件句转换），
  constant_propagation（常量条件句转换），trivial_condition_removal（无效条件移除的转换）
  transformation：转换类型句
  resulting_condition：转换之后的结果输出
```

> substitute_generated_columns substitute_generated_columns用于替换虚拟生成列

```
"substitute_generated_columns": {
} /* substitute_generated_columns */
```

> table_dependencies 分析表之间的依赖关系

```
{
  "table_dependencies": [
```



```
    {
      "table": "`salaries`",
      "row_may_be_null": false,
      "map_bit": 0,
      "depends_on_map_bits": [
      ] /* depends_on_map_bits */
    }
  ] /* table_dependencies */
}
```

其中：

table：涉及的表名，如果有别名，也会展示出来
row_may_be_null：行是否可能为NULL，这里是指JOIN操作之后，这张表里的数据是不是可能为NULL。如果语句中使用了LEFT JOIN，则后一张表的row_may_be_null会显示为true
map_bit：表的映射编号，从0开始递增
depends_on_map_bits：依赖的映射表。主要是当使用STRAIGHT_JOIN强行控制连接顺序或者LEFT JOIN/RIGHT JOIN有顺序差别时，会在depends_on_map_bits中展示前置表的map_bit值。

ref_optimizer_key_uses 列出所有可用的ref类型的索引。如果使用了组合索引的多个部分（例如本例，用到了index(from_date, to_date) 的多列索引），则会在ref_optimizer_key_uses下列出多个元素，每个元素中会列出ref使用的索引及对应值。

```
{
  "ref_optimizer_key_uses": [
    {
      "table": "`salaries`",
      "field": "from_date",
      "equals": "DATE'1986-06-26'",
```



```
      "null_rejecting": false
    },
    {
      "table": "`salaries`",
      "field": "to_date",
      "equals": "DATE'1987-06-26'",
      "null_rejecting": false
    }
  ] /* ref_optimizer_key_uses */
}
```

> rows_estimation 顾名思义，用于估算需要扫描的记录数。

```
{
  "rows_estimation": [
    {
      "table": "`salaries`",
      "range_analysis": {
        "table_scan": {
          "rows": 2838216,
          "cost": 286799
        } /* table_scan */,
        "potential_range_indexes": [
          {
            "index": "PRIMARY",
            "usable": false,
            "cause": "not_applicable"
          },
          {
            "index": "salaries_from_date_to_date_index",
            "usable": true,
            "key_parts": [
```



```
          "from_date",
          "to_date",
          "emp_no"
        ] /* 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 */,
    "skip_scan_range": {
      "potential_skip_scan_indexes": [
        {
          "index": "salaries_from_date_to_date_index",
          "usable": false,
          "cause": "query_references_nonkey_column"
        }
      ] /* potential_skip_scan_indexes */
    } /* skip_scan_range */,
    "analyzing_range_alternatives": {
      "range_scan_alternatives": [
        {
          "index": "salaries_from_date_to_date_index",
          "ranges": [
            "0xda840f <= from_date <= 0xda840f AND 0xda860f <= to_date <= 0xda860f"
          ] /* ranges */,
          "index_dives_for_eq_ranges": true,
          "rowid_ordered": true,
          "using_mrr": false,
          "index_only": false,
          "rows": 86,
          "cost": 50.909,
```



```
            "chosen": true
          }
        ] /* range_scan_alternatives */,
        "analyzing_roworder_intersect": {
          "usable": false,
          "cause": "too_few_roworder_scans"
        } /* analyzing_roworder_intersect */
      } /* analyzing_range_alternatives */,
      "chosen_range_access_summary": {
        "range_access_plan": {
          "type": "range_scan",
          "index": "salaries_from_date_to_date_index",
          "rows": 86,
          "ranges": [
            "0xda840f <= from_date <= 0xda840f AND 0xda860f <= to_date <= 0xda860f"
          ] /* ranges */
        } /* range_access_plan */,
        "rows_for_plan": 86,
        "cost_for_plan": 50.909,
        "chosen": true
      } /* chosen_range_access_summary */
    } /* range_analysis */
  }
 ] /* rows_estimation */
}
```

其中：

```
table：表名

range_analysis:
  table_scan：如果全表扫描的话，需要扫描多少行（row，2838216），以及需要的代价（cost，
286799）
```

六星教育 SIXSTAREDU.COM
11-六星教育-MYSQL-11-MYSQLSQL分析

potential_range_indexes：列出表中所有的索引并分析其是否可用。如果不可用的话，会列出不可用的原因是什么；如果可用会列出索引中可用的字段；
setup_range_conditions：如果有可下推的条件，则带条件考虑范围查询
group_index_range：当使用了GROUP BY或DISTINCT时，是否有合适的索引可用。当未使用GROUP BY或DISTINCT时，会显示chosen=false，cause=not_group_by_or_distinct；如使用了GROUP BY或DISTINCT，但是多表查询时，会显示chosen=false，cause =not_single_table。其他情况下会尝试分析可用的索引（potential_group_range_indexes）并计算对应的扫描行数及其所需代价
skip_scan_range：是否使用了skip scan

TIPS skip_scan_range是MySQL 8.0的新特性，感兴趣的可详见
https://blog.csdn.net/weixin_43970890/article/details/89494915

analyzing_range_alternatives：分析各个索引的使用成本
range_scan_alternatives：range扫描分析
index：索引名
ranges：range扫描的条件范围
index_dives_for_eq_ranges：是否使用了index dive，该值会被参数eq_range_index_dive_limit变量值影响。
rowid_ordered：该range扫描的结果集是否根据PK值进行排序
using_mrr：是否使用了mrr
index_only：表示是否使用了覆盖索引
rows：扫描的行数
cost：索引的使用成本
chosen：表示是否使用了该索引
analyzing_roworder_intersect：分析是否使用了索引合并（index merge），如果未使用，会在cause中展示原因；如果使用了索引合并，会在该部分展示索引合并的代价。
chosen_range_access_summary：在前一个步骤中分析了各类索引使用的方法及代价，得出了一定的中间结果之后，在summary阶段汇总前一阶段的中间结果确认最后的方案

range_access_plan：range扫描最终选择的执行计划。
type：展示执行计划的type，如果使用了索引合并，则会显示index_roworder_intersect
index：索引名
rows：扫描的行数
ranges：range扫描的条件范围

六星教育 SIXSTAREDU.COM
11-六星教育-MYSQL-11-MYSQLSQL分析


```
        rows_for_plan：该执行计划的扫描行数
        cost_for_plan：该执行计划的执行代价
        chosen：是否选择该执行计划
```

considered_execution_plans 负责对比各可行计划的开销，并选择相对最优的执行计划。

```
{
  "considered_execution_plans": [
    {
      "plan_prefix": [
      ] /* plan_prefix */,
      "table": "`salaries`",
      "best_access_path": {
        "considered_access_paths": [
          {
            "access_type": "ref",
            "index": "salaries_from_date_to_date_index",
            "rows": 86,
            "cost": 50.412,
            "chosen": true
          },
          {
            "access_type": "range",
            "range_details": {
              "used_index": "salaries_from_date_to_date_index"
            } /* range_details */,
            "chosen": false,
            "cause": "heuristic_index_cheaper"
          }
        ] /* considered_access_paths */
      } /* best_access_path */,
```



```
      "condition_filtering_pct": 100,
      "rows_for_plan": 86,
      "cost_for_plan": 50.412,
      "chosen": true
    }
  ] /* considered_execution_plans */
}
```

其中：

```
plan_prefix：当前计划的前置执行计划。
table：涉及的表名，如果有别名，也会展示出来
best_access_path：通过对比considered_access_paths，选择一个最优的访问路径
considered_access_paths：当前考虑的访问路径
access_type：使用索引的方式，可参考explain中的type字段
index：索引
rows：行数
cost：开销
chosen：是否选用这种执行路径
condition_filtering_pct：类似于explain的filtered列，是一个估算值
rows_for_plan：执行计划最终的扫描行数，由considered_access_paths.rows X
condition_filtering_pct计算获得。
cost_for_plan：执行计划的代价，由considered_access_paths.cost相加获得
chosen：是否选择了该执行计划
attaching_conditions_to_tables
基于considered_execution_plans中选择的执行计划，改造原有where条件，并针对表增加适当的附
加条件，以便于单表数据的筛选。
```

> TIPS 这部分条件的增加主要是为了便于ICP（索引条件下推），但ICP是否开启并不影响这部分内容的构造。 ICP参考文档：https://www.cnblogs.com/Terry-Wu/p/9273177.html


```
{
  "attaching_conditions_to_tables": {
    "original_condition": "((`salaries`.`to_date` = DATE'1987-06-26') and
(`salaries`.`from_date` = DATE'1986-06-26'))",
    "attached_conditions_computation": [
    ] /* attached_conditions_computation */,
    "attached_conditions_summary": [
      {
        "table": "`salaries`",
        "attached": "((`salaries`.`to_date` = DATE'1987-06-26') and (`salaries`.`from_date`
= DATE'1986-06-26'))"
      }
    ] /* attached_conditions_summary */
  } /* attaching_conditions_to_tables */
}
```

其中：

original_condition：原始的条件语句
attached_conditions_computation：使用启发式算法计算已使用的索引，如果已使用的索引的访问类型是ref，则计算用range能否使用组合索引中更多的列，如果可以，则用range的方式替换ref。
attached_conditions_summary：附加之后的情况汇总
table：表名
attached：附加的条件或原语句中能直接下推给单表筛选的条件。

> finalizing_table_conditions 最终的、经过优化后的表条件。

```
{
```



```
  "finalizing_table_conditions": [
    {
      "table": "`salaries`",
      "original_table_condition": "((`salaries`.`to_date` = DATE'1987-06-26') and
(`salaries`.`from_date` = DATE'1986-06-26'))",
      "final_table_condition   ": null
    }
  ] /* finalizing_table_conditions */
}
```

> refine_plan 改善执行计划：

```
{
  "refine_plan": [
    {
      "table": "`salaries`"
    }
  ] /* refine_plan */
}
```

其中：

table：表名及别名

> join_execution join_execution段落展示了执行阶段的执行过程。

```
"join_execution": {
  "select#": 1,
  "steps": [
```


```
  ] /* steps */
}
```