2019
05
08-10
北京新云南皇冠假日酒店

数据风云十年变迁

DTCC 第十届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2019

IT168 ChinaUnix ITPUB

MySQL SQL执行计划分析与优化

徐春阳 2019/05/10

CONTENTS

01 读懂最简单的SQL

- 深入学习的基础.
- 一个最简单的SQL的执行计划但含有丰富的信息.

1.1 范围查找range与等值查找ref

```
mysql>  explain select * from   score where  sid>=0 and   sid<=1   \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: range
possible_keys: sid
          key: sid
      key_len: 5
          ref: NULL
         rows: 13
     filtered: 100.00
        Extra: Using index condition
1 row in set, 1 warning (0.01 sec)
```

```
mysql> explain select * from   score where  sid=1 \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: ref
possible_keys: sid
          key: sid
      key_len: 5
          ref: const
         rows: 13
     filtered: 100.00
        Extra: NULL
1 row in set, 1 warning (0.00 sec)
```

对比观察：
Key相同
Type不同

1.2 范围查找range与全表扫描all

```
mysql>  explain select * from   score where  sid>=2 and   sid<=3   \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: ALL
possible_keys: sid
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 91
     filtered: 28.57
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

```
mysql> explain   select sid  from score  where sid in (7,6) \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: range
possible_keys: sid
          key: sid
      key_len: 5
          ref: NULL
         rows: 26
     filtered: 100.00
        Extra: Using where; Using index
1 row in set, 1 warning (0.00 sec)
```

为什么where 条件为sid in(7,6)是范围查找？
而sid=1的查询为ref查找？

1.3 全索引查找index

```
mysql> explain select sid from    score  \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: index
possible_keys: NULL
          key: sid
      key_len: 10
          ref: NULL
         rows: 91
     filtered: 100.00
        Extra: Using index
1 row in set, 1 warning (0.00 sec)
```

02 “聪明的”SQL优化器

- 优化器到底有多聪明？
- 为什么可以这么聪明？

2.1 为什么改变查找方式?

```
mysql>  explain select * from   score where  sid>=0 and   sid<=1   \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: range
possible_keys: sid
          key: sid
      key_len: 5
          ref: NULL
         rows: 13
     filtered: 100.00
        Extra: Using index condition
1 row in set, 1 warning (0.01 sec)
```

```
mysql>  explain select * from   score where  sid>=2 and   sid<=3
\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: ALL
possible_keys: sid
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 91
     filtered: 28.57
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

2.1.1 改变查找方式（执行计划）原因

因为优化器认为走全表扫描比走索引查找的成本更低

正确！但**成本**如何**预算**？

执行计划会浮动？同一个SQL，在不同环境或者同一环境却发生变化？
SQL优化器有点”个性，随意”。 但计算机里的逻辑都是客观的。

记下这个疑问后面待续

2.2 未执行就”知道”rows的值?

```
mysql> explain   select sid  from score  where sid in (7,6) \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: range
possible_keys: sid
          key: sid
      key_len: 5
          ref: NULL
         rows: 26
     filtered: 100.00
        Extra: Using where; Using index
1 row in set, 1 warning (0.00 sec)
```

```
mysql> explain   select sid  from score  where sid in (7,8) \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: range
possible_keys: sid
          key: sid
      key_len: 5
          ref: NULL
         rows: 14
     filtered: 100.00
        Extra: Using where; Using index
1 row in set, 1 warning (0.01 sec)
```

rows的值如何计算的？准吗？

2.3 未执行就”知道”filtered的值?

```
mysql> explain select * from   score where  sid=2  and score  >50 \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: ref
possible_keys: sid,idx_score
          key: sid
      key_len: 5
          ref: const
         rows: 13
     filtered: 100.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

```
mysql> explain select * from   score where  sid=2  and score  >80
\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: ref
possible_keys: sid,idx_score
          key: sid
      key_len: 5
          ref: const
         rows: 13
     filtered: 28.57
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

Filtered值准吗？

```
mysql> explain select * from   score where  sid=6  and score  >80 \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: ref
possible_keys: sid,idx_score
          key: sid
      key_len: 5
          ref: const
         rows: 13
     filtered: 28.57
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

返回结果行数=13*28.57% ？

```
mysql> explain select * from   score where  sid=2  and score  >80
\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: ref
Possible_keys: sid,idx_score
          key: sid
      key_len: 5
          ref: const
         rows: 13
     filtered: 28.57
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

Filtered值为什么是28.57？

2.4 如何计算filtered值

filtered (JSON name: filtered)
The filtered column indicates an estimated percentage of table rows that will be filtered by the table condition. That is, rows shows the estimated number of rows examined and rows × filtered / 100 shows the number of rows that will be joined with previous tables.

2.4.2 filtered值的计算逻辑

calculate_condition_filter函数

范围优化器是否已对执行计划所选访问方法未使用的谓词进行行估计，如果是，则范围优化器的估计值将用作这些字段的过滤效果的计算。
这样做是因为范围优化器比索引统计更准确。

是

```
const float selectivity=
        static_cast<float>(table->quick_rows[keyno]) /
        static_cast<float>(tab->records());
filter*= std::min(selectivity, 1.0f);
```

Quick_row[keyno]的值来自于优化器对where条件中列进行范围查找时预计将输出多少行。具体值来自函数check_quick_select

```
/*
    If the range optimizer has made row estimates for predicates that  are not used by the chosen access method, the estimate from the  range optimizer is used as filtering effect for those fields. We  do this because the range optimizer is more accurate than index  statistics.
  */
```

内容来自mysql5.7.18源代码注释

```
/*
    Get filtering effect for predicates that are not already
    reflected in 'filter'. The below call gets this filtering effect
    based on index statistics and guesstimates.
  */
  filter*=
    tab->join()->where_cond->get_filtering_effect(tab->table_ref->map(),
                                                  used_tables,
                                                  &table->tmp_set,
                                          static_cast<double>(tab->records()));
```

因此,filtered值计算不一定来自索引统计信息，也可能来自范围优化器对满足该范围的行数的预估。范围优化器比索引统计更准确。

2.4.3 filtered计算案例分析

```
mysql> explain select * from score where sid=2 and  score>80 \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: ref
possible_keys: sid,idx_score
          key: sid
      key_len: 5
          ref: const
         rows: 13
     filtered: 28.57
        Extra: Using where
1 row in set, 1 warning (7 min 15.94 sec)
```

```
mysql> select count(*) from score where score>80;
+----------+
| count(*) |
+----------+
|       26 |
+----------+
1 row in set (25.27 sec)

mysql> select count(*) from score ;
+----------+
| count(*) |
+----------+
|       91 |
+----------+
1 row in set (5.06 sec)
```

26/91=0.2857142..

2.4.4 filtered计算调试

```
(gdb) p  rows
$10 = 26
(gdb) bt
#0  check_quick_select (param=param@entry=0x7f9f02d66c70, idx=idx@entry=1, index_only=index_only@entry=false,
    tree=tree@entry=0x7f9ec8957138, update_tbl_stats=update_tbl_stats@entry=true, mrr_flags=mrr_flags@entry=0x7f9f02d66ad0,
    bufsize=bufsize@entry=0x7f9f02d66af0, cost=cost@entry=0x7f9f02d66bd0) at /mysql/mysql-5.7.18/sql/opt_range.cc:10082
#1  0x0000000000dd753d in get_key_scans_params (cost_est=0x7f9f02d66b90, update_tbl_stats=true, index_read_must_be_used=false,
    tree=0x7f9ec89570b8, param=0x7f9f02d66c70) at /mysql/mysql-5.7.18/sql/opt_range.cc:5812
#2  test_quick_select (thd=thd@entry=0x7f9ec8000ae0, keys_to_use=..., prev_tables=prev_tables@entry=0,
    limit=limit@entry=18446744073709551615, force_quick_range=force_quick_range@entry=false,
    interesting_order=interesting_order@entry=st_order::ORDER_NOT_RELEVANT, tab=tab@entry=0x7f9ec8955e10, cond=0x7f9ec80066a8,
    needed_reg=needed_reg@entry=0x7f9ec8955e50, quick=quick@entry=0x7f9f02d68f30) at /mysql/mysql-5.7.18/sql/opt_range.cc:3066
#3  0x0000000000c80fa5 in get_quick_record_count (limit=18446744073709551615, tab=0x7f9ec8955e10, thd=0x7f9ec8000ae0)
    at /mysql/mysql-5.7.18/sql/sql_optimizer.cc:5947
#4  JOIN::estimate_rowcount (this=this@entry=0x7f9ec8955828) at /mysql/mysql-5.7.18/sql/sql_optimizer.cc:5694
#5  0x0000000000c8720f in JOIN::make_join_plan (this=this@entry=0x7f9ec8955828) at /mysql/mysql-5.7.18/sql/sql_optimizer.cc:5051
#6  0x0000000000c88afc in JOIN::optimize (this=0x7f9ec8955828) at /mysql/mysql-5.7.18/sql/sql_optimizer.cc:368
#7  0x0000000000ccd185 in st_select_lex::optimize (this=this@entry=0x7f9ec8005540, thd=thd@entry=0x7f9ec8000ae0)
    at /mysql/mysql-5.7.18/sql/sql_select.cc:1009
#8  0x0000000000ccd365 in handle_query (thd=thd@entry=0x7f9ec8000ae0, lex=lex@entry=0x7f9ec8002c28,
    result=result@entry=0x7f9ec8007448, added_options=added_options@entry=0, removed_options=removed_options@entry=0)
    at /mysql/mysql-5.7.18/sql/sql_select.cc:164
#9  0x00000000007678af in execute_sqlcom_select (thd=thd@entry=0x7f9ec8000ae0, all_tables=0x7f9ec8006930)
```

2.4.4’ filtered计算调试

```
1369            const float selectivity=
1370              static_cast<float>(table->quick_rows[keyno]) /
1371              static_cast<float>(tab->records());
1372            // Cannot possible access more rows than there are in the table
1373            filter*= std::min(selectivity, 1.0f);
1374          }
(gdb) p selectivity
$12 = <optimized out>
(gdb) n
493       ha_rows records() const { return m_qs->records(); }
(gdb) n
1370              static_cast<float>(table->quick_rows[keyno]) /
(gdb) n
1373            filter*= std::min(selectivity, 1.0f);
(gdb) p selectivity
$13 = 0.285714298
(gdb) bt
#0  calculate_condition_filter (tab=tab@entry=0x7f9ec896f900, keyuse=keyuse@entry=0x7f9ec896fd40,
    used_tables=18446744073709551614, fanout=fanout@entry=13, is_join_buffering=is_join_buffering@entry=false)
    at /mysql/mysql-5.7.18/sql/sql_planner.cc:1373
#1  0x0000000000ca65cc in Optimize_table_order::best_access_path (this=this@entry=0x7f9f02d692b0, tab=tab@entry=0
    remaining_tables=remaining_tables@entry=1, idx=idx@entry=0, disable_jbuf=true, disable_jbuf@entry=false,
    prefix_rowcount=<optimized out>, pos=pos@entry=0x7f9ec896fa98) at /mysql/mysql-5.7.18/sql/sql_planner.cc:1149
#2  0x0000000000caae3a in Optimize_table_order::best_extension_by_limited_search (this=this@entry=0x7f9f02d692b0,
    remaining_tables=remaining_tables@entry=1, idx=idx@entry=0, current_search_depth=62)
    at /mysql/mysql-5.7.18/sql/sql_planner.cc:2635
#3  0x0000000000cabccc in Optimize_table_order::greedy_search (this=this@entry=0x7f9f02d692b0,
```

2.5 为什么改变查找方式（回顾）

```
mysql>  explain select * from   score where  sid>=0 and   sid<=1   \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: range
possible_keys: sid
          key: sid
      key_len: 5
          ref: NULL
         rows: 13
     filtered: 100.00
        Extra: Using index condition
1 row in set, 1 warning (0.01 sec)
```

```
mysql>  explain select * from   score where  sid>=2 and   sid<=3
\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: ALL
possible_keys: sid
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 91
     filtered: 28.57
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

2.5.1 执行计划计算成本

1. 计算全表扫描时的成本：

```
if (!records)
  records++;                                  /* purecov: inspected */
double scan_time=
  cost_model->row_evaluate_cost(static_cast<double>(records)) + 1;
Cost_estimate cost_est= head->file->table_scan_cost();
cost_est.add_io(1.1);
cost_est.add_cpu(scan_time);
```



```
(gdb) p cost_est
$31 = {io_cost = 2.1000000000000001, cpu_cost = 19.199999999999999, import_cost = 0, mem_cost = 0}
```

```
(gdb) p (0.20000000000000001*91)+1
$12 = 19.199999999999999
```

select count(*) from score结果为91

2. 计算索引查找时的成本：

```
*cost= read_cost(keyno, static_cast<double>(n_ranges),
                 static_cast<double>(total_rows));
  cost->add_cpu(cost_model->row_evaluate_cost(
    static_cast<double>(total_rows)) + 0.01);
```


```
*cost= read_cost(keyno, static_cast<double>(n_ranges),
                 static_cast<double>(total_rows));
  cost->add_cpu(cost_model->row_evaluate_cost(
    static_cast<double>(total_rows)) + 0.01);
```


```
(gdb) p *cost
$53 = {io_cost = 27, cpu_cost = 5.21, import_cost = 0, mem_cost = 0}
(gdb) p 26*0.20000000000000001+0.01
$55 = 5.21
```

select count(*) from score where sid>=2 and sid<=3 的结果为26

2.5.2 查找方式成本比较

```
mysql>  explain select * from   score where  sid>=0 and   sid<=1   \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: range
possible_keys: sid
          key: sid
      key_len: 5
          ref: NULL
         rows: 13
     filtered: 100.00
        Extra: Using index condition
1 row in set, 1 warning (0.01 sec)
```

select count(*) from score where sid>=0 and   sid<=1 结果等于13

走索引成本计算：
io_cost=14 , cpu_cost= 13*0.20000000000000001+0.01
总成本14+2.6=16.6

走全表扫描成本计算： io_cost = 2.1000000000000001,
cpu_cost = 19.199999999999999
总成本:2.1+19.19=21.29

03 JOIN的方式

- Nested loop join 是MySQL唯一的join方式
- 可怕的、低效的Nested loop join ?
- 如何利用Nested loop join解决大表的关联 ?

3.1 Nested-Loop Join Algorithms

T1，T2，T3关联

Table Access Type
t1 range
t2 ref
t3 ALL

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    for each row in t3 {
      if row satisfies join conditions, send to client
    }
  }
}
```

最常见的遍历+嵌套算法

是否注意到T3的访问方式为all,也就是全表扫描。根据Nested loop join的计算方法，外层循环一次，表T3就需要全表扫描一次。假如T1表有N1行，T2表有N2 行， T3最多可能需要被全表扫描N1*N2 次。 非常恐怖！！！，但MySQL的开发者会关心这个问题。

3.2 Block Nested-Loop Join Algorithm

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    store used columns from t1, t2 in join buffer
    if buffer is full {
      for each row in t3 {
        for each t1, t2 combination in join buffer {
          if row satisfies join conditions, send to client
        }
      }
      empty join buffer
    }
  }
}
if buffer is not empty {
  for each row in t3 {
    for each t1, t2 combination in join buffer {
      if row satisfies join conditions, send to client
    }
  }
}
```

只有当join buffer 满了之后，才会对表t3进行全表扫描：将从表T3取出来的每一行跟join buffer的记录比较，检查是否满足条件。所以join buffer越大，对表T3 的扫描次数就越少。公式如下：

(*S* * *C*)/join_buffer_size + 1

S：the size of each stored t1, t2 combination in the join buffer

C: the number of combinations in the join buffer

内容来源于mysql 5.7 refman

3.3 Block Nested-Loop与NLJ区别及使用场景

```
mysql> explain select  s.name,sc.cid,sc.score
    -> from student s , score sc  where sc.sid=s.id \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: s
   partitions: NULL
         type: ALL
possible_keys: PRIMARY
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 21
     filtered: 100.00
        Extra: NULL
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: sc
   partitions: NULL
         type: ref
possible_keys: sid
          key: sid
      key_len: 5
          ref: xcytest.s.id
         rows: 12
     filtered: 100.00
        Extra: NULL
2 rows in set, 1 warning (0.00 sec)
```

```
mysql> alter table score drop key "sid";
mysql> alter table  student  drop primary key ;
mysql> explain select  s.name,sc.cid,sc.score
    -> from student s , score sc  where sc.sid=s.id \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: s
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 21
     filtered: 100.00
        Extra: NULL
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: sc
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 91
     filtered: 10.00
        Extra: Using where; Using join buffer (Block Nested Loop)
2 rows in set, 1 warning (0.01 sec)
```

第一个表访问方式不变(don't care)，第二个表的访问方式从ref变成了all,然后join方式变成了block nested loop

有没有遇到过关联中的内表走索引查找有时可能比全表扫描慢,而且可能是慢非常多的情况？

04 子查询

- 可能有曾听说过在MySQL上不建议使用子查询
- 可曾试想过MySQL对子查询是如何处理的，通过嵌套方式层层处理？
- 但不是所有的子查询是真正的子查询
- .....

4 子查询优化方式

The MySQL query optimizer has different strategies available to evaluate subqueries:

For IN (or =ANY) subqueries, the optimizer has these choices:

Semi-join

Materialization

EXISTS strategy

For NOT IN (or <>ALL) subqueries, the optimizer has these choices:

Materialization

EXISTS strategy

For derived tables, the optimizer has these choices (which also apply to view references):

Merge the derived table into the outer query block

Materialize the derived table to an internal temporary table

内容来自mysql 5.7refman

4.1 子查询变成内连接

```
mysql> explain   select  * from student where   id   in  ( select sid from  score where cid=2) \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: score
   partitions: NULL
         type: index
possible_keys: sid
          key: sid
      key_len: 10
          ref: NULL
         rows: 91
     filtered: 10.00
        Extra: Using where; Using index
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: student
   partitions: NULL
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: xcytest.score.sid
         rows: 1
     filtered: 100.00
        Extra: NULL
2 rows in set, 1 warning (0.03 sec)
```

```
mysql> show warnings;
select "xcytest"."student"."id" AS "id",
"xcytest"."student"."name" AS "name"
from "xcytest"."score" join "xcytest"."student"
where (("xcytest"."student"."id" = "xcytest"."score"."sid") and
("xcytest"."score"."cid" = 2))
```

为什么这个子查询可以转变成了内连接？
转换成内连接的好处是什么？

4.1’ 子查询变成内连接

```
mysql> explain  select sid,cid,score from  score sc where  sc.sid in ( select  id from
student   ) and sc.score >60 \G
*************************** 1. row *******
           id: 1
  select_type: SIMPLE
        table: student
   partitions: NULL
         type: index
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: NULL
         rows: 21
     filtered: 100.00
        Extra: Using index
*************************** 2. row *******
           id: 1
  select_type: SIMPLE
        table: sc
   partitions: NULL
         type: ref
possible_keys: sid,idx_score
          key: sid
      key_len: 5
          ref: xcytest.student.id
         rows: 13
     filtered: 100.00
        Extra: Using where
```

```
mysql> show warnings;
select "xcytest"."sc"."sid" AS "sid",
"xcytest"."sc"."cid" AS "cid",
"xcytest"."sc"."score" AS "score“
 from "xcytest"."student" join "xcytest"."score" "sc“
 where (("xcytest"."sc"."sid" = "xcytest"."student"."id")
and ("xcytest"."sc"."score" > 60))
```

为什么这个子查询也转变成了内连接？

子查询转换成半连接条件

```
1011    DBUG_PRINT("info", ("Checking if subq can be converted to semi-join"));
1012    /*
1013      Check if we're in subquery that is a candidate for flattening into a
1014      semi-join (which is done in flatten_subqueries()). The requirements are:
1015        1. Subquery predicate is an IN/=ANY subquery predicate
1016        2. Subquery is a single SELECT (not a UNION)
1017        3. Subquery does not have GROUP BY
1018        4. Subquery does not use aggregate functions or HAVING
1019        5. Subquery predicate is (a) in an ON/WHERE clause, and (b) at
1020        the AND-top-level of that clause.
1021        6. Parent query block accepts semijoins (i.e we are not in a subquery of
1022        a single table UPDATE/DELETE (TODO: We should handle this at some
1023        point by switching to multi-table UPDATE/DELETE)
1024        7. We're not in a confluent table-less subquery, like "SELECT 1".
1025        8. No execution method was already chosen (by a prepared statement)
1026        9. Parent select is not a confluent table-less select
1027        10. Neither parent nor child select have STRAIGHT_JOIN option.
1028    */
```

子查询转换成半连接条件代码

```
1029    if (semijoin_enabled(thd) &&
1030        in_predicate &&                                  // 1
1031        !is_part_of_union() &&                           // 2
1032        !group_list.elements &&                          // 3
1033        !m_having_cond && !with_sum_func &&              // 4
1034        (outer->resolve_place == st_select_lex::RESOLVE_CONDITION ||      // 5a
1035         outer->resolve_place == st_select_lex::RESOLVE_JOIN_NEST) &&     // 5a
1036        !outer->semijoin_disallowed &&                   // 5b
1037        outer->sj_candidates &&                          // 6
1038        leaf_table_count &&                              // 7
1039        in_predicate->exec_method ==
1040                        Item_exists_subselect::EXEC_UNSPECIFIED &&   // 8
1041        outer->leaf_table_count &&                       // 9
1042        !((active_options() | outer->active_options()) &
1043         SELECT_STRAIGHT_JOIN))                          //10
1044    {
1045      DBUG_PRINT("info", ("Subquery is semi-join conversion candidate"));
```

半连接进一步转换为内连接

```
/**
  Pull tables out of semi-join nests based on functional dependencies

  @param join  The join where to do the semi-join table pullout

  @return False if successful, true if error (Out of memory)

  @details
    Pull tables out of semi-join nests based on functional dependencies,
    ie. if a table is accessed via eq_ref(outer_tables).
    The function may be called several times, the caller is responsible
    for setting up proper key information that this function acts upon.

    NOTE
    Table pullout may make uncorrelated subquery correlated. Consider this
    example:

     ... WHERE oe IN (SELECT it1.primary_key WHERE p(it1, it2) ... )

    here table it1 can be pulled out (we have it1.primary_key=oe which gives
    us functional dependency).
Making the subquery (i.e. its semi-join nest) correlated prevents us from
    using Materialization or LooseScan to execute it. */

static bool pull_out_semijoin_tables(JOIN *join)
```

4.2 利用MATERIALIZED处理半连接

```
mysql> explain  select  * from student where  id  in  ( select sid from score) \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: student
   partitions: NULL
         type: ALL
Possible_keys: PRIMARY
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 21
     filtered: 100.00
        Extra: Using where
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: <subquery2>
   partitions: NULL
         type: eq_ref
possible_keys: <auto_key>
          key: <auto_key>
      key_len: 5
          ref: xcytest.student.id
         rows: 1
     filtered: 100.00
        Extra: NULL
*************************** 3. row ***************************
           id: 2
  select_type: MATERIALIZED
        table: score
   partitions: NULL
         type: index
possible_keys: sid
          key: sid
      key_len: 10
          ref: NULL
```

```
mysql> show warnings;
select "xcytest"."student"."id" AS "id",
"xcytest"."student"."name" AS "name"
from "xcytest"."student" semi join
(“xcytest"."score")
where ("<subquery2>"."sid" =
xcytest"."student"."id")
```

通过MATERIALIZED的方式对子查询<subquery2>生成临时表,然后跟外表进行1vs1等值连接eq_ref。

4.3 利用loosescan处理半连接

```
mysql> explain select * from xcytest     where  a in  ( select     b  from xcytestb       ) ;
+----+-------------+---------+-----------+------+---------------+-----+---------+-----+------+----------+-
-----------------------------------------+
| id | select_type | table     | partitions | type  | possible_keys | key  | key_len | ref  | rows |
filtered | Extra                                  |
+----+-------------+---------+-----------+------+---------------+-----+---------+-----+------+----------+-
-----------------------------------------+
|  1 | SIMPLE      | xcytestb | NULL       | index | b             | b    | 5       | NULL |    2 |   100.00
| Using index; LooseScan                 |
|  1 | SIMPLE      | xcytest  | NULL       | ALL   | PRIMARY       | NULL | NULL    | NULL |    2
|    50.00 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+---------+-----------+------+---------------+-----+---------+-----+------+----------+-
-----------------------------------------+
```

4.4 其他处理半连接策略

the firstmatch, loosescan, duplicateweedout, and materialization flags enable finer control over the permitted semi-join strategies

-----请参考mysql 5.7 refman

4.5 非常特殊的“子查询”

```
mysql> explain select * from testa  a where a.id in  (select  b.id from testb  b where  b.id=100);
+----+-------------+-------+-----------+-----+---------------+-----+---------+-----+------+----------+-----------------------
----------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra
|
+----+-------------+-------+-----------+-----+---------------+-----+---------+-----+------+----------+-----------------------
----------+
|  1 | SIMPLE      | NULL  | NULL       | NULL | NULL          | NULL | NULL    | NULL | NULL |     NULL | no
matching row in const table |
+----+-------------+-------+-----------+-----+---------------+-----+---------+-----+------+----------+-----------------------
----------+
1 row in set, 1 warning (0.00 sec)
```


在分析阶段就知道了这个查询将为空？

4.6 常量表转换

```
(gdb) bt
#0  ha_innobase::index_read (this=0x7f067c058d00, buf=0x7f067c059110 "\377",
key_ptr=0x7f067c06e930 "\004", key_len=4, find_flag=HA_READ_KEY_EXACT)
#1  0x0000000000f9fba4 in handler::index_read_map (this=0x7f067c058d00, buf=0x7f067c059110 "\377",
key=0x7f067c06e930 "\004", keypart_map=1, find_flag=HA_READ_KEY_EXACT)
……
#4  0x000000000154eb1b in read_const (table=0x7f067c058350, ref=0x7f067c073448)
#5  0x000000000154e5fe in join_read_const_table (tab=0x7f067c0732e0, pos=0x7f067c0735c0)
……..
#7  0x0000000001579ad5 in JOIN::make_join_plan (this=0x7f067c072d40)
#8  0x000000000156e576 in JOIN::optimize (this=0x7f067c072d40)
#9  0x00000000015e5ee6 in st_select_lex::optimize (this=0x7f067c053f70, thd=0x7f067c01bf40) #10
0x00000000015e4642 in handle_query (thd=0x7f067c01bf40, lex=0x7f067c01e238,
result=0x7f067c06dc38, added_options=0, removed_options=0) at /mysqldata/mysql-
5.7.18/sql/sql_select.cc:164
```

4.7 必须手工优化的子查询

例如：

select * from （select id from testa union select id from testb ）out_a_b where out_a_b.id=8;

select * from （select id from testa union all select id from testb ）out_a_b where out_a_b.id=8;

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

无法转化成半连接的子查询

4.7.1 手工优化子查询方法

1.将子查询上拉，将子查询跟外层表平级,将嵌套查询变成关联查询。 ------消除子查询

2. 将查询的条件下沉，使子查询能够快速执行且生成的结果集变小。------快速物理化子查询

05 视图/派生表

- 采用跟外表合并的方式优化
- 谨慎使用包含union/union all的视图/派生表

5.1 视图/派生表的优化方式

For derived tables, the optimizer has these choices (which also apply to view references):

Merge the derived table into the outer query block

Materialize the derived table to an internal temporary table

5.2 视图/派生表优化函数

```
/**
  Merge a derived table or view into a query block.
  If some constraint prevents the derived table from being merged then do
  nothing, which means the table will be prepared for materialization later.

  After this call, check is_merged() to see if the table was really merged.

  @param thd           Thread handler
  @param derived_table Derived table which is to be merged.

  @return false if successful, true if error
*/

bool SELECT_LEX::merge_derived(THD *thd, TABLE_LIST *derived_table)
{
  DBUG_ENTER("SELECT_LEX::merge_derived");

  if (!derived_table->is_view_or_derived() || derived_table->is_merged())
    DBUG_RETURN(false);

  SELECT_LEX_UNIT *const derived_unit= derived_table->derived_unit();

  // A derived table must be prepared before we can merge it
  DBUG_ASSERT(derived_unit->is_prepared());

  LEX *const lex= parent_lex;

  // Check whether the outer query allows merged views
  if ((master_unit() == lex->unit && 。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。
```

5.3 派生表/视图能合并的前提条件

```
bool st_select_lex_unit::is_mergeable() const
{
  if (is_union())
    return false;

  SELECT_LEX *const select= first_select();
  Item *item;
  List_iterator<Item> it(select->fields_list);
  while ((item= it++))
  {
    if (item->has_subquery() && item->used_tables())
      return false;
  }
  return !select->is_grouped() &&
         !select->having_cond() &&
         !select->is_distinct() &&
         select->table_list.elements > 0 &&
         !select->has_limit() &&
         thd->lex->set_var_list.elements == 0;
}
```

5.4 视图的good案例

```
create view v_student_score  as select  name,sid,cid,score  from score
sc,student st
where st.id=sc.sid  ;
```

仅对视图查询：

```
select * from  v_student_score where name='xu';
```

视图跟另外一个表join

```
select vsc.name,vsc.sid,c.name, vsc.score
from  v_student_score vsc , course  c where vsc.name='xu'   and vsc.cid=c.id ;
```

```
mysql> explain select vsc.name,vsc.sid,c.name, vsc.score
    -> from  v_student_score vsc , course  c where vsc.name='xu'   and vsc.cid=c.id ;
+----+-------------+-------+------------+------+---------------+------+---------+---------------------------+------+----------+------------------------------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref                       | rows | filtered | Extra                              |
+----+-------------+-------+------------+------+---------------+------+---------+---------------------------+------+----------+------------------------------------+
|  1 | SIMPLE      | st    | NULL       | ALL  | PRIMARY       | NULL | NULL    | NULL                      |   21 |    10.00 | Using where                        |
|  1 | SIMPLE      | c     | NULL       | ALL  | PRIMARY       | NULL | NULL    | NULL                      |   13 |   100.00 | Using join buffer (Block Nested Loop) |
|  1 | SIMPLE      | sc    | NULL       | ref  | sid           | sid  | 10      | xcytest.st.id,xcytest.c.Id |    1 |   100.00 | NULL                               |
+----+-------------+-------+------------+------+---------------+------+---------+---------------------------+------+----------+------------------------------------+
3 rows in set, 1 warning (0.00 sec)

mysql> show warnings;
+-------+------+---------------------------------------------------------------------------------------------------------------------------------+
| Level | Code | Message                                                                                                                         |
+-------+------+---------------------------------------------------------------------------------------------------------------------------------+
| Note  | 1003 | /* select#1 */ select "st"."name" AS "name","xcytest"."sc"."sid" AS "sid","xcytest"."c"."name" AS "name","xcytest"."sc"."score" AS "score" from "xcytest"."score" "sc" join "xcytest"."student" "st" join "xcytest"."course" "c" where (("xcytest"."sc"."sid" = "st"."id") and ("xcytest"."sc"."cid" = "xcytest"."c"."Id") and ("st"."name" = 'xu')) |
+-------+------+---------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

5.5 视图的bad案例

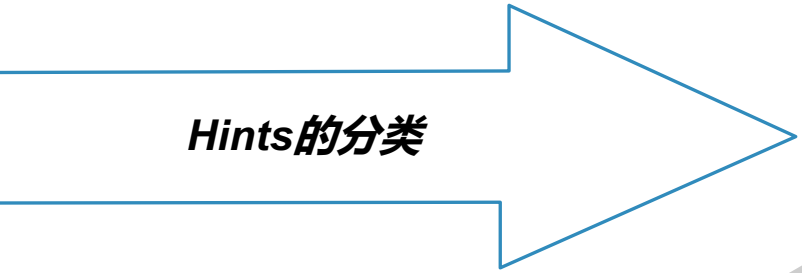
```
mysql> create view     v_test  as select * from testa union all  select * from testb;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from v_test where id=2;
+----+------+------+------+
| id | c2   | c3   | c4   |
+----+------+------+------+
|  2 | c22  | c31  | c41  |
|  2 | c22  | c31  | c41  |
+----+------+------+------+
2 rows in set (0.01 sec)

mysql> explain    select * from v_test where id=2;
+----+-------------+-----------+------------+------+---------------+------------+---------+-------+------+----------+-------+
| id | select_type | table     | partitions | type | possible_keys | key        | key_len | ref   | rows | filtered | Extra |
+----+-------------+-----------+------------+------+---------------+------------+---------+-------+------+----------+-------+
|  1 | PRIMARY     | <derived2> | NULL       | ref  | <auto_key0>   | <auto_key0> | 4       | const |    4 |   100.00 | NULL  |
|  2 | DERIVED     | testa      | NULL       | ALL  | NULL          | NULL       | NULL    | NULL  |   21 |   100.00 | NULL  |
|  3 | UNION       | testb      | NULL       | ALL  | NULL          | NULL       | NULL    | NULL  |   21 |   100.00 | NULL  |
+----+-------------+-----------+------------+------+---------------+------------+---------+-------+------+----------+-------+
3 rows in set, 1 warning (0.00 sec)
```

06 “愚蠢”的优化器以及纠正

- 优化器作SQL执行计划时，不保证绝对不犯错
- 优化器作执行计划时，是局部最优，而不是全路径最优
- 主要原因filtered值是一个分布概率估算，无法精确某个具体值数量
- 。。。。。。。

解决方法:告诉优化器，按照您的指令去执行SQL

6.1 优化器hints



指定索引

指定关联时谁做外表

提示优化器选用什么策略

6.1.1 使用指定的索引查找

```
mysql> explain select * from   score where  sid=2  and score  >80
    -> ;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | score | NULL       | ref  | sid,idx_score | sid  | 5       | const |   13 |    28.57 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.01 sec)
```

```
mysql> explain  select * from score  force index(idx_score)  where  sid=2  and score  >80;
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-----------------------------+
| id | select_type | table | partitions | type  | possible_keys | key       | key_len | ref  | rows | filtered | Extra                       |
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-----------------------------+
|  1 | SIMPLE      | score | NULL       | range | idx_score     | idx_score | 5       | NULL |   26 |    14.29 | Using index condition; Using where |
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-----------------------------+
1 row in set, 1 warning (0.07 sec)
```

6.1.2 指定表关联的顺序

```
mysql> explain  select   sc.score,cid, s.name from  student s,score sc where sc.sid=s.id   and score >80;
+----+-------------+-------+------------+--------+---------------+---------+---------+-----------------+------+----------+-------------+
| id | select_type | table | partitions | type   | possible_keys | key     | key_len | ref             | rows | filtered | Extra       |
+----+-------------+-------+------------+--------+---------------+---------+---------+-----------------+------+----------+-------------+
|  1 | SIMPLE      | sc    | NULL       | ALL    | sid,idx_score | NULL    | NULL    | NULL            |   91 |    28.57 | Using where |
|  1 | SIMPLE      | s     | NULL       | eq_ref | PRIMARY       | PRIMARY | 4       | learning.sc.sid |    1 |   100.00 | NULL        |
+----+-------------+-------+------------+--------+---------------+---------+---------+-----------------+------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)
```

explain select sc.score,cid, s.name from student s STRAIGHT_JOIN score sc where sc.sid=s.id and score >80;

explain select STRAIGHT_JOIN sc.score,cid, s.name from student s,score sc where sc.sid=s.id and score >80;

```
mysql> explain select  STRAIGHT_JOIN sc.score,cid, s.name from  student s,score sc where sc.sid=s.id   and score >80;
+----+-------------+-------+------------+------+---------------+------+---------+--------------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref          | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+--------------+------+----------+-------------+
|  1 | SIMPLE      | s     | NULL       | ALL  | PRIMARY       | NULL | NULL    | NULL         |   21 |   100.00 | NULL        |
|  1 | SIMPLE      | sc    | NULL       | ref  | sid,idx_score | sid  | 5       | learning.s.id |   13 |    28.57 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+--------------+------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)
```

6.1.3 指定优化器采用什么优化策略

```
mysql> explain  select * from student  s where s.id in ( select sid from score where  score >60);
+----+--------------+-------------+------------+--------+---------------+------------+---------+--------------+------+----------+-------------+
| id | select_type  | table       | partitions | type   | possible_keys | key        | key_len | ref          | rows | filtered | Extra       |
+----+--------------+-------------+------------+--------+---------------+------------+---------+--------------+------+----------+-------------+
|  1 | SIMPLE       | s           | NULL       | ALL    | PRIMARY       | NULL       | NULL    | NULL         |   21 |   100.00 | Using where |
|  1 | SIMPLE       | <subquery2> | NULL       | eq_ref | <auto_key>    | <auto_key> | 5       | learning.s.id |    1 |   100.00 | NULL        |
|  2 | MATERIALIZED | score       | NULL       | ALL    | sid,idx_score | NULL       | NULL    | NULL         |   91 |   100.00 | Using where |
+----+--------------+-------------+------------+--------+---------------+------------+---------+--------------+------+----------+-------------+
3 rows in set, 1 warning (0.00 sec)

mysql> show warnings;
+-------+------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Level | Code | Message                                                                                                                                                                                                  |
+-------+------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Note  | 1003 | /* select#1 */ select "learning"."s"."id" AS "id","learning"."s"."name" AS "name" from "learning"."student" "s" semi join ("learning"."score") where (("<subquery2>"."sid" = "learning"."s"."id") and ("learning"."score"."score" > 60)) |
+-------+------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

当前 SQL 默认使用materialized 子查询来处理半连接

6.1.3’ 提示优化器采用指定的优化策略

```
mysql> explain SELECT /*+ SEMIJOIN(@subq1 FIRSTMATCH) */ * from  student  s
    -> WHERE s.id IN (SELECT /*+ QB_NAME(subq1) */ sid from score where  score >60);
+----+-------------+-------+------------+------+---------------+------+---------+--------------+------+----------+----------------------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref          | rows | filtered | Extra                      |
+----+-------------+-------+------------+------+---------------+------+---------+--------------+------+----------+----------------------------+
|  1 | SIMPLE      | s     | NULL       | ALL  | PRIMARY       | NULL | NULL    | NULL         |   21 |   100.00 | NULL                       |
|  1 | SIMPLE      | score | NULL       | ref  | sid,idx_score | sid  | 5       | learning.s.id |  13 |   100.00 | Using where; FirstMatch(s) |
+----+-------------+-------+------------+------+---------------+------+---------+--------------+------+----------+----------------------------+
2 rows in set, 1 warning (0.00 sec)

Mysql> show  warnings;
+-------+------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Level | Code | Message                                                                                                                                                                                            |
+-------+------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Note  | 1003 | /* select#1 */ select /*+ SEMIJOIN(@”subq1” FIRSTMATCH) */ “learning”.”s”.”id” AS “id”,”learning”.”s”.”name” AS “name” from “learning”.”student” “s” semi join (“learning”.”score”) where ((“learning”.”score”.”sid” = “learning”.”s”.”id”) and (“learning”.”score”.”score” > 60)) |
+-------+------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
```

指定当前SQL 使用firstMatch策略来处理半连接

9.9.2 Optimizer Hints