链滴

【转】MySQL EXPLAIN 命令详解学习

作者：honglan

原文链接：https://ld246.com/article/1495764695034

来源网站：链滴

MySQL EXPLAIN 命详解
MySQL 的 EXPLAIN 命令用于 SQL 语句的查询执行计划(QEP)。这条命令的输出结果能够让我了解 MySQL 优化器是如何执行
SQL 语句的。这条命令并没有提供任何调整建议，但它能够提供重要的信息帮助你做出调优决策。</>
1 语法
MySQL 的 EXPLAIN 语法可以运行在 SELECT 语句或者特定表上。如果作用在表上，那么此命令等于 DESC 表命令。UPDATE
和 DELETE 命令也需要进行性能改进，当这些命令不是直接在表的主码上运行时，为了确保最优化的引使用率，需要把它们改
写成 SELECT 语句(以便对它们执行 EXPLAIN 命令)。请看下面的示例：

UPDATE table1
SET col1 = X, col2 = Y
WHERE id1 = 9
AND dt >= '2010-01-01';

这个 UPDATE 语句可以被重写成为下面这样的 SELECT 语句：

SELECT col1, col2
FROM table1
WHERE id1 = 9
AND dt >= '2010-01-01';

在 5.6.10 版本里面,是可以直接对 dml 语句进行 explain 分析操作的.
MySQL 优化器是基于开销来工作的，它并不提供任何的 QEP 的位置。这意味着 QEP 是在每条QL 语句执行的时候动态地计
算出来的。在 MySQL 存储过程中的 SQL 语句也是在每次执行时计算 QEP 的。存储过程缓存仅仅解查询树。
2 各列详解
MySQL EXPLAIN 命令能够为 SQL 语句中的每个表生成以下信息：

mysql> EXPLAIN SELECT * FROM inventory WHERE item_id = 16102176\G;
********************* 1. row ************************
id: 1
select_type: SIMPLE
table: inventory
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 787338
Extra: Using where

这个 QEP 显示没有使用任何索引(也就是全表扫描)并且处理了大量的行来满足查询。对同样一条ELECT 语句，一个优化过的 QEP 如下所示：

********************* 1. row ************************
id: 1
select_type: SIMPLE
table: inventory

type: ref
possible_keys: item_id
key: item_id
key_len: 4
ref: const
rows: 1
Extra:

在这个 QEP 中，我们看到使用了一个索引，且估计只有一行数据将被获取。
**QEP 中每个行的所有列表如下所示：
**□ id
□ select_type
□ table
□ partitions(这一列只有在 EXPLAIN PARTITIONS 语法中才会出现)
□ possible_keys
□ key
□ key_len
□ ref
□ rows
□ filtered(这一列只有在 EXPLAINED EXTENDED 语法中才会出现)
□ Extra
这些列展示了 SELECT 语句对每一个表的 QEP。一个表可能和一个物理模式表或者在 SQL 执行生成的内部临时表(例如从子查询或者合并操作会产生内部临时表)相关联。
可以参考 MySQL Reference Manual 获得更多信息：http://dev.mysql.com/doc/refman/5.5/en/explain-output.htm 。
2.1 key
key 列指出优化器选择使用的索引。一般来说 SQL 查询中的每个表都仅使用一个索引。也存在索引并的少数例外情况，如给定表上用到了两个或者更多索引。
下面是 QEP 中 key 列的示例：
key: item_id
key: NULL
key: first, last
SHOW CREATE TABLE 命令是最简单的查看表和索引列细节的方式。和 key 列相关的列还包括 possi le_keys、rows 以及 key_len。
2.2 ROWS
rows 列提供了试图分析所有存在于累计结果集中的行数目的 MySQL 优化器估计值。QEP 很容易描这个很困难的统计量。
查询中总的读操作数量是基于合并之前行的每一行的 rows 值的连续积累而得出的。这是一种嵌套行法。
以连接两个表的 QEP 为例。通过 id=1 这个条件找到的第一行的 rows 值为 1，这等于对第一个做了一次读操作。第二行是
通过 id=2 找到的，rows 的值为 5。这等于有 5 次读操作符合当前 1 的积累量。参考两个表，读操的总数目是 6。在另一个 QEP
中，第一 rows 的值是 5，第二 rows 的值是 1。这等于第一个表有 5 次读操作，对 5 个积累量中每都有一个读操作。因此两个表
总的读操作的次数是 10(5+5)次。
最好的估计值是 1，一般来说这种情况发生在当寻找的行在表中可以通过主键或者唯一键找到的候。
在下面的 QEP 中，外面的嵌套循环可以通过 id=1 来找到，其估计的物理行数是 1。第二个循环处理 10 行。

********** 1. row **********

id: 1
select_type: SIMPLE
table: p
type: const
possible_keys: PRIMARY
key: PRIMARY
key_len: 4
ref: const
rows: 1
Extra:
*********************** 2. row ***********************
id: 1
select_type: SIMPLE
table: c
type: ref
possible_keys: parent_id
key: parent_id
key_len: 4
ref: const
rows: 10
Extra:

可以使用 SHOW STATUS 命令来查看实际的行操作。这个命令可以提供最佳的确认物理行操作方式。请看下面的示例：

mysql> SHOW SESSION STATUS LIKE 'Handler_read%';
+-----------------------+-------+
| Variable_name | Value |
+-----------------------+-------+
| Handler_read_first | 0 |
| Handler_read_key | 0 |
| Handler_read_last | 0 |
| Handler_read_next | 0 |
| Handler_read_prev | 0 |
| Handler_read_rnd | 0 |
| Handler_read_rnd_next | 11 |
+-----------------------+-------+
7 rows in set (0.00 sec)

在下一个 QEP 中，通过 id=1 找到的外层嵌套循环估计有 160 行。第二个循环估计有 1 行。</
>

*********************** 1. row ***********************
id: 1
select_type: SIMPLE
table: p
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 160
Extra:
*********************** 2. row ***********************

id: 1
select type: SIMPLE
table: c
type: ref
possible_keys: PRIMARY,parent_id
key: parent_id
key_len: 4
ref: test.p.parent_id
rows: 1
Extra: Using where

通过 SHOW STATUS 命令可以查看实际的行操作，该命令表明物理读操作数量大幅增加。请看面的示例：

mysql> SHOW SESSION STATUS LIKE 'Handler_read%';
+---------------------------------------+---------+
| Variable_name | Value |
+---------------------------------------+---------+
| Handler_read_first | 1 |
| Handler_read_key | 164 |
| Handler_read_last | 0 |
| Handler_read_next | 107 |
| Handler_read_prev | 0 |
| Handler_read_rnd | 0 |
| Handler_read_rnd_next | 161 |
+---------------------------------------+---------+
相关的 QEP 列还包括 key 列。

2.3 possible_keys possible_keys 列指出优化器为查询选定的索引。 一个会列出大量可能的索引例如多于 3 个)的 QEP 意味着备选索引数量太多了，同时也可能提示存在一个无效的单列索引。 可用第 2 章详细介绍过的 SHOW INDEXES 命令来检查索引是否有效且是否具有合适的基数。 为查询定 QEP 的速度也会影响到查询的性能。如果发现有大量的可能的索引，则意味着这些索引没有被使到。 相关的 QEP 列还包括 key 列。 2.4 key_len key_len 列定义了用于 SQL 语句的连接条件的键长度。此列值对于确认索引的有效性以及多列索引中用到的列的数目很重要。 此列的一些示例值如下示： 此列的一些示例值如下所示： key_len: 4 // INT NOT NULL key_len: 5 // INT NULL key_len: 0 // CHAR(30) NOT NULL key_len: 32 // VARCHAR(30) NOT NULL key_len: 92 // VARCHAR(30) NULL CHARSET=utf8 从这些示例中可以看出，是否可以为空、可变长度的列以及 key_len 列的值和用在连接和 WHERE 条件中的索引的列 有关。索引中的其他列会在 ORDER BY 或者 GROUP BY 句中被用到。下面这个来自于著名的开源博客软件 WordPress 的表展示了 如何以最佳方式使用带有义好的表索引的 SQL 语句：

CREATE TABLE <code>wp_posts</code> (
<code>ID</code> bigint(20) unsigned NOT NULL AUTO_INCREMENT,
<code>post_date</code> datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
<code>post_status</code> varchar(20) NOT NULL DEFAULT 'publish' ,
<code>post_type</code> varchar(20) NOT NULL DEFAULT 'post',
PRIMARY KEY (<code>ID</code>),
KEY <code>type_status_date</code>(<code>post_type</code>,<code>post_status</code>,<code>post_date</code>,<code>ID</code>)
) DEFAULT CHARSET=utf8

CREATE TABLE <code>wp_posts</code> (
<code>ID</code> bigint(20) unsigned NOT NULL AUTO_INCREMENT,
<code>post_date</code> datetime NOT NULL DEFAULT '0000-00-00 00:00:00',

<code>post_status</code> varchar(20) NOT NULL DEFAULT 'publish' ,
<code>post_type</code> varchar(20) NOT NULL DEFAULT 'post',
PRIMARY KEY (<code>ID</code>),
KEY <code>type_status_date</code>(<code>post_type</code>,<code>post_status</code>,<code>post_date</code>,<code>ID</code>)
) DEFAULT CHARSET=utf8

这个表的索引包括 post_type、post_status、post_date 以及 ID 列。下面是一个演示索引列用的 SQL 查询： EXPLAIN SELECT ID, post_title FROM wp_posts WHERE post_type='post' AND post_date > '2010-06-01'；这个查询的 QEP 返回的 key_len 是 62。这说明只有 post_type 列上索引用到了(因为(20×3)+2=62)。尽管查询在 WHERE 语句 中使用了 post_type 和 post_date 列，只有 post_type 部分被用到了。其他索引没有被使用的原因是 MySQL 只能使用定义索引的 最左边分。为了更好地利用这个索引，可以修改这个查询来调整索引的列。请看下面的示例：

mysql> EXPLAIN SELECT ID, post_title
-> FROM wp_posts
-> WHERE post_type='post'
-> AND post_status='publish'
-> AND post_date > '2010-06-01';

在 SELECT 查询的添加一个 post_status 列的限制条件后，QEP 显示 key_len 的值为 132，这味着 post_type、post_status、post_date 三列(62+62+8，(20×3)+2，(20×3)+2，8)都被用到了此外，这个索引的主码列 ID 的定义是使用 MyISAM 存储索 引的遗留痕迹。当使用 InnoDB 存储引时，在非主码索引中包含主码列是多余的，这可以从 key_len 的用法看出来。 相关的 QEP 列还包括有 Using index 值的 Extra 列。 2.5 table table 列是 EXPLAIN 命令输出结果中的一个单独行的唯标识符。这个值可能是表名、表的别名或者一个为查询产生临时表 的标识符，如派生表、子查询或集。下面是 QEP 中 table 列的一些示例： table: item table: table: 表中 N 和 M 的值参考了另一个合 id 列值的 table 行。相关的 QEP 列还有 select_type
** 2.6 select_type**
select_type 列提供了各种表示 table 列引用的使用方式的类型。最常见的值包括 SIMPLE、PRIMAR、DERIVED 和 UNION。其他可能
的值还有 UNION RESULT、DEPENDENT SUBQUERY、DEPENDENT UNION、UNCACHEABLE UION 以及 UNCACHEABLE QUERY。
1. SIMPLE
对于不包含子查询和其他复杂语法的简单查询，这是一个常 见的类型。
2. PRIMARY
这是为更复杂的查询而创建的首要表(也就是最外层的表)。这个类型通常可以在 DERIVED 和 UNION 类型混合使用时见到。
3. DERIVED
当一个表不是一个物理表时，那么就被叫做 DERIVED。下面的 SQL 语句给出了一个 QEP 中 DERIVE select-type 类型的
示例：
mysql> EXPLAIN SELECT MAX(id)
-> FROM (SELECT id FROM users WHERE first = 'west') c;
4. DEPENDENT SUBQUERY
这个 select-type 值是为使用子查询而定义的。下面的 SQL 语句提供了这个值：
mysql> EXPLAIN SELECT p.*
-> FROM parent p
-> WHERE p.id NOT IN (SELECT c.parent_id FROM child c);
5. UNION
这是 UNION 语句其中的一个 SQL 元素。
6. UNION RESULT
这是一系列定义在 UNION 语句中的表的返回结果。当 select_type 为这个值时，经常可以看到 table 的值是，

这说明匹配的 id 行是这个集合的一部分。下面的 SQL 产生了一个 UNION 和 UNION RESULT select type：
mysql> EXPLAIN SELECT p.* FROM parent p WHERE p.val
LIKE 'a%'
-> UNION
-> SELECT p.* FROM parent p WHERE p.id > 5;
** 2.7 partitions**
partitions 列代表给定表所使用的分区。这一列只会在 EXPLAIN
PARTITIONS 语句中出现。
** 2.8 Extra**
Extra 列提供了有关不同种类的 MySQL 优化器路径的一系列
额外信息。Extra 列可以包含多个值，可以有很多不同的取值，并
且这些值还在随着 MySQL 新版本的发布而进一步增加。下面给
出常用值的列表。你可以从下面的地址找到更全面的值的列表：
http://dev.mysql.om/doc/refman/5.5/en/explain-output.html。
1. Using where
这个值表示查询使用了 where 语句来处理结果——例如执行
全表扫描。如果也用到了索引，那么行的限制条件是通过获取必
要的数据之后处理读缓冲区来实现的。
2. Using temporary
这个值表示使用了内部临时(基于内存的)表。一个查询可能
用到多个临时表。有很多原因都会导致 MySQL 在执行查询期间
创建临时表。两个常见的原因是在来自不同表的列上使用了
DISTINCT，或者使用了不同的 ORDER BY 和 GROUP BY 列。
想了解更多内容可以访问 http://forge.mysql.com/wik/Overview_
of_query_execution_and_use_of_temp_tables。
可以强制指定一个临时表使用基于磁盘的 MyISAM 存储引
擎。这样做的原因主要有两个：
 内部临时表占用的空间超过 min(tmp_table_size，max_
heap_table_size)系统变量的限制
 使用了 TEXT/BLOB 列
3. Using filesort
这是 ORDER BY 语句的结果。这可能是一个 CPU 密集型的过程。
可以通过选择合适的索引来改进性能，用索引来为查询结果排序。详细过程请参考第 4 章。
4. Using index
这个值重点强调了只需要使用索引就可以满足查询表的要求，不需要直接访问表数据。请参考第 5 章详细示例来理解这
个值。
5. Using join buffer
这个值强调了在获取连接条件时没有使用索引，并且需要连接缓冲区来存储中间结果。
如果出现了这个值，那应该注意，根据查询的具体情况可能需要添加索引来改进性能。
6. Impossible where
这个值强调了 where 语句会导致没有符合条件的行。请看下面的示例：
mysql> EXPLAIN SELECT * FROM user WHERE 1=2;
7. Select tables optimized away
这个值意味着仅通过使用索引，优化器可能仅从聚合函数结果中返回一行。请看下面的示例：
8. Distinct
这个值意味着 MySQL 在找到第一个匹配的行之后就会停止搜索其他行。
9. Index merges
当 MySQL 决定要在一个给定的表上使用超过一个索引的时候，就会出现以下格式中的一个，详细说

使用的索引以及合并的类型。
 Using sort_union(...)
 Using union(...)
 Using intersect(...)
** 2.9 id**
id 列是在 QEP 中展示的表的连续引用。
** 2.10 ref**
ref 列可以被用来标识那些用来进行索引比较的列或者常量。
** 2.11 filtered**
filtered 列给出了一个百分比的值，这个百分比值和 rows 列的值一起使用，可以估计出那些将要和 QP 中的前一个表进行连
接的行的数目。前一个表就是指 id 列的值比当前表的 id 小的表。这一列只有在 EXPLAIN EXTENDED 语句中才会出现。
** 2.12 type**
type 列代表 QEP 中指定的表使用的连接方式。下面是最常用的几种连接方式：
 const 当这个表最多只有一行匹配的行时出现 system 这是 const 的特例，当表只有一个 row 时会现
 eq_ref 这个值表示有一行是为了每个之前确定的表而读取的
 ref 这个值表示所有具有匹配的索引值的行都被用到
 range 这个值表示所有符合一个给定范围值的索引行都被用到
 ALL 这个值表示需要一次全表扫描其他类型的值还有 fulltext 、ref_or_null 、index_merge 、unique_subquery、index_subquery 以及 index。
想了解更多信息可以访问 http://dev.mysql.com/doc/refman/5.5/en/explain-output.html。
** 3 解释 EXPLAIN 输出结果**
理解你的应用程序(包括技术和实现可能性)和优化 SQL 语句同等重要。下面给出一个从父子关系中获孤立的父辈记录的商
业需求的例子。这个查询可以用三种不同的方式构造。尽管会产生相同的结果，但 QEP 会显示三种同的路径。

mysql> EXPLAIN SELECT p.*
-> FROM parent p
-> WHERE p.id NOT IN (SELECT c.parent_id FROM child
c)\G
*********************** 1. row ***********************
id: 1
select type: PRIMARY
table: p
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 160
Extra: Using where
*********************** 2. row ***********************
id: 2
select_type: DEPENDENT SUBQUERY
table: c
type: index_subquery
possible_keys: parent_id
key: parent_id
key_len: 4

ref: func
rows: 1
Extra: Using index
2 rows in set (0.00 sec)

EXPLAIN SELECT p.* FROM parent p LEFT JOIN child c ON p.id = c.parent_id WHERE c.chil
_id IS NULL\G
*********************** 1. row ***********************
id: 1
select_type: SIMPLE
table: p
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 160
Extra:
*********************** 2. row ***********************
id: 1
select_type: SIMPLE
table: c
type: ref
possible_keys: parent_id
key: parent_id
key_len: 4
ref: test.p.id
rows: 1
Extra: Using where; Using index; Not exists
2 rows in set (0.00 sec)
