# 概述

explain可用来分析SQL的执行计划。格式如下：

{EXPLAIN | DESCRIBE | DESC}

tbl\_name [col\_name | wild]

{EXPLAIN | DESCRIBE | DESC}

[explain\_type]

{explainable\_stmt | FOR CONNECTION connection\_id}

{EXPLAIN | DESCRIBE | DESC} ANALYZE select\_statement

explain\_type: {

FORMAT = format\_name

}

format\_name: {

TRADITIONAL

| JSON

| TREE

}

explainable\_stmt: {

SELECT statement

| TABLE statement

| DELETE statement

| INSERT statement

| REPLACE statement

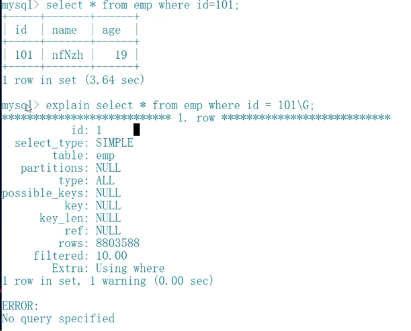
| UPDATE statement

}

# 分析

用来查看索引是否正在被使用，并且输出其使用的索引的信息。

在未添加索引时：



在添加索引后：



具体包含的参数：

## id

id：SELECT标识符，这是SELECT的查询序列号，也就是一条语句中，该SELECT是第几次出现，在此语句中，SELECT就只有一个，所以是1。

数字越大越先执行；而对于相同id的行，则表示从上往下依次执行。

## selecy\_type

selecy\_type：所使用的SELECT查询类型，SIMPLE表示为简单的SELECT，不使用UNION或子查询，就为简单的SELECT。

查询类型，有如下几种取值：

|  |  |
| --- | --- |
| 查询类型 | 作用 |
| SIMPLE | 简单查询（未使用UNION或子查询） |
| PRIMARY | |  | | --- | | 最外层的查询 | |
| UNION | 在UNION中的第二个和随后的SELECT被标记为UNION。如果UNION被FROM子句中的子查询包含，那么它的第一个SELECT会被标记为DERIVED。 |
| DEPENDENT UNION | UNION中的第二个或后面的查询，依赖了外面的查询 |
| UNION RESULT | UNION的结果 |
| SUBQUERY | 子查询中的第一个 SELECT |
| DEPENDENT SUBQUERY | 子查询中的第一个 SELECT，依赖了外面的查询 |
| DERIVED | 用来表示包含在FROM子句的子查询中的SELECT，MySQL会递归执行并将结果放到一个临时表中。MySQL内部将其称为是Derived table（派生表），因为该临时表是从子查询派生出来的 |
| DEPENDENT DERIVED | 派生表，依赖了其他的表 |
| MATERIALIZED | 物化子查询 |
| UNCACHEABLE SUBQUERY | 子查询，结果无法缓存，必须针对外部查询的每一行重新评估 |
| UNCACHEABLE UNION | UNION属于UNCACHEABLE SUBQUERY的第二个或后面的查询 |

## table

table：数据表的名字，他们按照被读取的先后顺序排序。

如果SQL定义了别名，则展示表的别名。

## partitions

partitions：当前查询匹配记录的分区。

对于未分区的表，返回null。

## type

type：指定本数据表和其他数据表之间的关联关系，该表中所有符合检索值的记录都会被取出来和上一个表中取出来的记录作联合。

连接类型，有如下几种取值，性能从好到坏排序如下：

1、system：该表只有一行（相当于系统表），system是const类型的特例

2、const：针对主键或唯一索引的等值查询扫描, 最多只返回一行数据。const 查询速度非常快, 因为它仅仅读取一次即可。

3、eq\_ref：当使用了索引的全部组成部分，并且索引是PRIMARY KEY或UNIQUE NOT NULL 才会使用该类型，性能仅次于system及const。

-- 多表关联查询，单行匹配

SELECT \* FROM ref\_table,other\_table

WHERE ref\_table.key\_column=other\_table.column;

-- 多表关联查询，联合索引，多行匹配

SELECT \* FROM ref\_table,other\_table

WHERE ref\_table.key\_column\_part1=other\_table.column

AND ref\_table.key\_column\_part2=1;

4、ref：当满足索引的最左前缀规则，或者索引不是主键也不是唯一索引时才会发生。如果使用的索引只会匹配到少量的行，性能也是不错的。

-- 根据索引（非主键，非唯一索引），匹配到多行

SELECT \* FROM ref\_table WHERE key\_column=expr;

-- 多表关联查询，单个索引，多行匹配

SELECT \* FROM ref\_table,other\_table WHERE ref\_table.key\_column=other\_table.column;

-- 多表关联查询，联合索引，多行匹配

SELECT \* FROM ref\_table,other\_table WHERE ref\_table.key\_column\_part1=other\_table.column AND ref\_table.key\_column\_part2=1;

注：

最左前缀原则，指的是索引按照最左优先的方式匹配索引。比如创建了一个组合索引(column1, column2, column3)，那么，如果查询条件是：

● WHERE column1 = 1、WHERE column1= 1 AND column2 = 2、WHERE column1= 1 AND column2 = 2 AND column3 = 3 都可以使用该索引；

● WHERE column1 = 2、WHERE column1 = 1 AND column3 = 3就无法匹配该索引。

5、fulltext：全文索引

6、ref\_or\_null：该类型类似于ref，但是MySQL会额外搜索哪些行包含了NULL。这种类型常见于解析子查询

SELECT \* FROM ref\_table

WHERE key\_column=expr OR key\_column IS NULL;

7、index\_merge：此类型表示使用了索引合并优化，表示一个查询里面用到了多个索引

8、unique\_subquery：该类型和eq\_ref类似，但是使用了IN查询，且子查询是主键或者唯一索引。例如：

value IN (SELECT primary\_key FROM single\_table WHERE some\_expr)

9、index\_subquery：和unique\_subquery类似，只是子查询使用的是非唯一索引

value IN (SELECT key\_column FROM single\_table WHERE some\_expr)

10、range：范围扫描，表示检索了指定范围的行，主要用于有限制的索引扫描。比较常见的范围扫描是带有BETWEEN子句或WHERE子句里有>、>=、<、<=、IS NULL、<=>、BETWEEN、LIKE、IN()等操作符。

SELECT \* FROM tbl\_name

WHERE key\_column BETWEEN 10 and 20;

SELECT \* FROM tbl\_name

WHERE key\_column IN (10,20,30);

11、index：全索引扫描，和ALL类似，只不过index是全盘扫描了索引的数据。当查询仅使用索引中的一部分列时，可使用此类型。有两种场景会触发：

（1）如果索引是查询的覆盖索引，并且索引查询的数据就可以满足查询中所需的所有数据，则只扫描索引树。此时，explain的Extra 列的结果是Using index。index通常比ALL快，因为索引的大小通常小于表数据。

（2）按索引的顺序来查找数据行，执行了全表扫描。此时，explain的Extra列的结果不会出现Uses index。

（3）ALL：全表扫描，性能最差。

## key

key：实际选用的索引。

## possible\_keys

possible\_keys：展示当前查询可以使用哪些索引，这一列的数据是在优化过程的早期创建的，因此有些索引可能对于后续优化过程是没用的。

## key\_len

key\_len显示了MySQL使用索引的长度（也就是使用索引的个数），索引使用的字节数。由于存储格式，当字段允许为NULL时，索引的长度就是NULL，key\_len比不允许为空时大1字节。

key\_len计算公式： <https://www.cnblogs.com/gomysql/p/4004244.html>

## ref

ref：表示将哪个字段或常量和key列所使用的字段进行比较。

如果ref是一个函数，则使用的值是函数的结果。要想查看是哪个函数，可在EXPLAIN语句之后紧跟一个SHOW WARNING语句。

## rows

rows：MySQL估算会扫描的行数，数值越小越好。

## filtered

filtered：表示符合查询条件的数据百分比，最大100。用rows × filtered可获得和下一张表连接的行数。

例如rows = 1000，filtered = 50%，则和下一张表连接的行数是500。

注：

● 在MySQL 5.7之前，想要显示此字段需使用explain extended命令；

● MySQL.5.7及更高版本，explain默认就会展示filtered

## Extra

展示有关本次查询的附加信息，取值如下：

1、Child of 'table' pushed join@1

此值只会在NDB Cluster下出现。

2、const row not found

例如查询语句SELECT ... FROM tbl\_name，而表是空的

3、Deleting all rows

对于DELETE语句，某些引擎（例如MyISAM）支持以一种简单而快速的方式删除所有的数据，如果使用了这种优化，则显示此值

4、Distinct

查找distinct值，当找到第一个匹配的行后，将停止为当前行组合搜索更多行

5、FirstMatch(tbl\_name)

当前使用了半连接FirstMatch策略，

详见https://mariadb.com/kb/en/firstmatch-strategy/，

翻译https://www.cnblogs.com/abclife/p/10895624.html

6、Full scan on NULL key

子查询中的一种优化方式，在无法通过索引访问null值的时候使用

7、Impossible HAVING

HAVING子句始终为false，不会命中任何行

8、Impossible WHERE

WHERE子句始终为false，不会命中任何行

9、Impossible WHERE noticed after reading const tables

MySQL已经读取了所有const（或system）表，并发现WHERE子句始终为false

10、LooseScan(m..n)

当前使用了半连接LooseScan策略，

详见 https://mariadb.com/kb/en/loosescan-strategy/，

翻译 http://www.javacoder.cn/?p=39

11、No matching min/max row

没有任何能满足例如 SELECT MIN(...) FROM ... WHERE condition 中的condition的行

12、no matching row in const table

对于关联查询，存在一个空表，或者没有行能够满足唯一索引条件

13、No matching rows after partition pruning

对于DELETE或UPDATE语句，优化器在partition pruning（分区修剪）之后，找不到要delete或update的内容

14、No tables used

当此查询没有FROM子句或拥有FROM DUAL子句时出现。例如：explain select 1

15、Not exists

MySQL能对LEFT JOIN优化，在找到符合LEFT JOIN的行后，不会为上一行组合中检查此表中的更多行。例如：

SELECT \* FROM t1 LEFT JOIN t2 ON t1.id=t2.id

WHERE t2.id IS NULL;

假设t2.id定义成了NOT NULL ，此时，MySQL会扫描t1，并使用t1.id的值查找t2中的行。如果MySQL在t2中找到一个匹配的行，它会知道t2.id永远不会为NULL，并且不会扫描t2中具有相同id值的其余行。也就是说，对于t1中的每一行，MySQL只需要在t2中只执行一次查找，而不考虑在t2中实际匹配的行数。

在MySQL 8.0.17及更高版本中，如果出现此提示，还可表示形如 NOT IN (subquery) 或 NOT EXISTS (subquery) 的WHERE条件已经在内部转换为反连接。这将删除子查询并将其表放入最顶层的查询计划中，从而改进查询的开销。

通过合并半连接和反联接，优化器可以更加自由地对执行计划中的表重新排序，在某些情况下，可让查询提速。你可以通过在EXPLAIN语句后紧跟一个SHOW WARNING语句，并分析结果中的Message列，从而查看何时对该查询执行了反联接转换。

注：

两表关联只返回主表的数据，并且只返回主表与子表没关联上的数据，这种连接就叫反连接

16、Plan isn't ready yet

使用了EXPLAIN FOR CONNECTION，当优化器尚未完成为在指定连接中为执行的语句创建执行计划时， 就会出现此值。

17、Range checked for each record (index map: N)

MySQL没有找到合适的索引去使用，但是去检查是否可以使用range或index\_merge来检索行时，会出现此提示。index map N索引的编号从1开始，按照与表的SHOW INDEX所示相同的顺序。索引映射值N是指示哪些索引是候选的位掩码值。例如0x19（二进制11001）的值意味着将考虑索引1、4和5。

示例：下面例子中，name是varchar类型，但是条件给出整数型，涉及到隐式转换。图中t2也没有用到索引，是因为查询之前我将t2中name字段排序规则改为utf8\_bin导致的链接字段排序规则不匹配。

explain select a.\* from t1 a left join t2 b

on t1.name = t2.name

where t2.name = 2;

18、Recursive

出现了递归查询。

19、Rematerialize

用得很少，使用类似如下SQL时，会展示Rematerialize

SELECT ... FROM t, LATERAL (derived table that refers to t) AS dt ...

20、Scanned N databases

表示在处理INFORMATION\_SCHEMA表的查询时，扫描了几个目录，N的取值可以是0，1或者all。

21、Select tables optimized away

优化器确定：①最多返回1行；②要产生该行的数据，要读取一组确定的行，时会出现此提示。一般在用某些聚合函数访问存在索引的某个字段时，优化器会通过索引直接一次定位到所需要的数据行完成整个查询时展示，例如下面这条SQL。

explain

select min(id)

from t1;

22、Skip\_open\_table， Open\_frm\_only， Open\_full\_table

这些值表示适用于INFORMATION\_SCHEMA表查询的文件打开优化；

23、Skip\_open\_table：无需打开表文件，信息已经通过扫描数据字典获得

24、Open\_frm\_only：仅需要读取数据字典以获取表信息

25、Open\_full\_table：未优化的信息查找。表信息必须从数据字典以及表文件中读取

26、Start temporary, End temporary

表示临时表使用Duplicate Weedout策略

详见 https://mariadb.com/kb/en/duplicateweedout-strategy/

翻译 https://www.cnblogs.com/abclife/p/10895531.html

27、unique row not found

对于形如 SELECT ... FROM tbl\_name 的查询，但没有行能够满足唯一索引或主键查询的条件

28、Using filesort

当Query 中包含 ORDER BY 操作，而且无法利用索引完成排序操作的时候，MySQL Query Optimizer 不得不选择相应的排序算法来实现。数据较少时从内存排序，否则从磁盘排序。Explain不会显示的告诉客户端用哪种排序。官方解释：“MySQL需要额外的一次传递，以找出如何按排序顺序检索行。通过根据联接类型浏览所有行并为所有匹配WHERE子句的行保存排序关键字和行的指针来完成排序。然后关键字被排序，并按排序顺序检索行”

29、Using index

仅使用索引树中的信息从表中检索列信息，而不必进行其他查找以读取实际行。当查询仅使用属于单个索引的列时，可以使用此策略。例如：

explain SELECT id FROM t

30、Using index condition

表示先按条件过滤索引，过滤完索引后找到所有符合索引条件的数据行，随后用 WHERE 子句中的其他条件去过滤这些数据行。通过这种方式，除非有必要，否则索引信息将可以延迟“下推”读取整个行的数据。例如：

注：

MySQL分成了Server层和引擎层，下推指的是将请求交给引擎层处理。

理解这个功能，可创建所以INDEX (zipcode, lastname, firstname)，并分别用如下指令，

SET optimizer\_switch = 'index\_condition\_pushdown=off';

SET optimizer\_switch = 'index\_condition\_pushdown=on';

开或者关闭索引条件下推，并对比：

explain SELECT \* FROM people

WHERE zipcode='95054'

AND lastname LIKE '%etrunia%'

AND address LIKE '%Main Street%';

的执行结果。

index condition pushdown从MySQL 5.6开始支持，是MySQL针对特定场景的优化机制，感兴趣的可以看下 https://blog.51cto.com/lee90/2060449

31、Using index for group-by

数据访问和 Using index 一样，所需数据只须要读取索引，当Query 中使用GROUP BY或DISTINCT 子句时，如果分组字段也在索引中，Extra中的信息就会是 Using index for group-by。详见 “GROUP BY Optimization”[12]

-- name字段有索引

explain SELECT name FROM t1 group by name

32、Using index for skip scan

表示使用了Skip Scan。详见 Skip Scan Range Access Method[13]

33、Using join buffer (Block Nested Loop), Using join buffer (Batched Key Access)

使用Block Nested Loop或Batched Key Access算法提高join的性能。

详见 https://www.cnblogs.com/chenpingzhao/p/6720531.html

34、Using MRR

使用了Multi-Range Read优化策略。

35、Using sort\_union(...), Using union(...), Using intersect(...)

这些指示索引扫描如何合并为index\_merge连接类型。

36、Using temporary

为了解决该查询，MySQL需要创建一个临时表来保存结果。如果查询包含不同列的GROUP BY和 ORDER BY子句，通常会发生这种情况。

-- name无索引

explain SELECT name FROM t1 group by name

37、Using where

如果我们不是读取表的所有数据，或者不是仅仅通过索引就可以获取所有需要的数据，则会出现using where信息

explain SELECT \* FROM t1 where id > 5

38、Using where with pushed condition

仅用于NDB

39、Zero limit

该查询有一个limit 0子句，不能选择任何行

explain SELECT name FROM resource\_template limit 0

# 扩展explain

EXPLAIN可产生额外的扩展信息，可通过在EXPLAIN语句后紧跟一条SHOW WARNING语句查看扩展信息。

注：

● 在MySQL 8.0.12及更高版本，扩展信息可用于SELECT、DELETE、INSERT、REPLACE、UPDATE语句；在MySQL 8.0.12之前，扩展信息仅适用于SELECT语句；

● 在MySQL 5.6及更低版本，需使用EXPLAIN EXTENDED xxx语句；而从MySQL 5.7开始，无需添加EXTENDED关键词。

使用示例：

mysql> EXPLAIN

SELECT t1.a, t1.a IN (SELECT t2.a FROM t2) FROM t1\G

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

id: 1

select\_type: PRIMARY

table: t1

type: index

possible\_keys: NULL

key: PRIMARY

key\_len: 4

ref: NULL

rows: 4

filtered: 100.00

Extra: Using index

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 2

select\_type: SUBQUERY

table: t2

type: index

possible\_keys: a

key: a

key\_len: 5

ref: NULL

rows: 3

filtered: 100.00

Extra: Using index

2 rows in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G

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

Level: Note

Code: 1003

Message: /\* select#1 \*/ select `test`.`t1`.`a` AS `a`,

<in\_optimizer>(`test`.`t1`.`a`,`test`.`t1`.`a` in

( <materialize> (/\* select#2 \*/ select `test`.`t2`.`a`

from `test`.`t2` where 1 having 1 ),

<primary\_index\_lookup>(`test`.`t1`.`a` in

<temporary table> on <auto\_key>

where ((`test`.`t1`.`a` = `materialized-subquery`.`a`))))) AS `t1.a

IN (SELECT t2.a FROM t2)` from `test`.`t1`

1 row in set (0.00 sec)

由于SHOW WARNING的结果并不一定是一个有效SQL，也不一定能够执行（因为里面包含了很多特殊标记）。特殊标记取值如下：

1、<auto\_key>

自动生成的临时表key

2、<cache>(expr)

表达式（例如标量子查询）执行了一次，并且将值保存在了内存中以备以后使用。对于包括多个值的结果，可能会创建临时表，你将会看到 <temporary table> 的字样

3、<exists>(query fragment)

子查询被转换为 EXISTS

4、<in\_optimizer>(query fragment)

这是一个内部优化器对象，对用户没有任何意义

5、<index\_lookup>(query fragment)

使用索引查找来处理查询片段，从而找到合格的行

6、<if>(condition, expr1, expr2)

如果条件是true，则取expr1，否则取expr2

7、<is\_not\_null\_test>(expr)

验证表达式不为NULL的测试

8、<materialize>(query fragment)

使用子查询实现

9、materialized-subquery.col\_name

在内部物化临时表中对col\_name的引用，以保存子查询的结果

10、<primary\_index\_lookup>(query fragment)

使用主键来处理查询片段，从而找到合格的行

11、<ref\_null\_helper>(expr)

这是一个内部优化器对象，对用户没有任何意义

12、/\* select#N \*/ select\_stmt

SELECT与非扩展的EXPLAIN输出中id=N的那行关联

13、outer\_tables semi join (inner\_tables)

半连接操作。inner\_tables展示未拉出的表。

14、<temporary table>

表示创建了内部临时表而缓存中间结果

当某些表是const或system类型时，这些表中的列所涉及的表达式将由优化器尽早评估，并且不属于所显示语句的一部分。但是，当使用FORMAT=JSON时，某些const表的访问将显示为ref。

# 估计查询性能

多数情况下，你可以通过计算磁盘的搜索次数来估算查询性能。对于比较小的表，通常可以在一次磁盘搜索中找到行（因为索引可能已经被缓存了），而对于更大的表，你可以使用B-tree索引进行估算：你需要进行多少次查找才能找到行：log(row\_count) / log(index\_block\_length / 3 \* 2 / (index\_length + data\_pointer\_length)) + 1

在MySQL中，index\_block\_length通常是1024字节，数据指针一般是4字节。比方说，有一个500,000的表，key是3字节，那么根据计算公式 log(500,000)/log(1024/3\*2/(3+4)) + 1 = 4 次搜索。

该索引将需要500,000 \* 7 \* 3/2 = 5.2MB的存储空间（假设典型的索引缓存的填充率是2/3），因此你可以在内存中存放更多索引，可能只要一到两个调用就可以找到想要的行了。

但是，对于写操作，你需要四个搜索请求来查找在何处放置新的索引值，然后通常需要2次搜索来更新索引并写入行。

前面的讨论并不意味着你的应用性能会因为log N而缓慢下降。只要内容被OS或MySQL服务器缓存，随着表的变大，只会稍微变慢。在数据量变得太大而无法缓存后，将会变慢很多，直到你的应用程序受到磁盘搜索约束（按照log N增长）。

为了避免这种情况，可以根据数据的增长而增加key的。对于MyISAM表，key的缓存大小由名为key\_buffer\_size的系统变量控制。