mysql优化

# 设置高速缓存

## 设置高速缓存

### 查看高速缓存是否可用

SHOW VARIABLES LIKE ‘have\_query\_cache’;

### 设置和查询高速缓存大小

SET GLOBAL query\_cache\_size = 41984;

SHOW VARIABLES LIKE ‘query\_cache\_size’;

+------------------+-------+

| Variable\_name    | Value |

+------------------+-------+

| query\_cache\_size | 41984 |

+------------------+-------+

### 缓存开启的方式

mysql> SET SESSION query\_cache\_type = ON;

如果查询缓存大小设置为大于0，query\_cache\_type变量影响其工作方式。这个变量可以设置为下面的值：

0或OFF：将阻止缓存或查询缓存结果。

1或ON：将允许缓存，以SELECT SQL\_NO\_CACHE开始的查询语句除外。

2或DEMAND：仅对以SELECT SQL\_CACHE开始的那些查询语句启用缓存。

另外：

设置query\_cache\_type变量的GLOBAL值将决定更改后所有连接客户端的缓存行为。具体客户端可以通过设置query\_cache\_type变量的会话值控制它们本身连接的缓存行为。

例如，一个客户可以禁用自己的查询缓存，方法如下：

mysql> SET SESSION query\_cache\_type = OFF;

SHOW VARIABLES LIKE 'query\_cache\_size';#显示缓存大小

SET SESSION query\_cache\_type = OFF;#关闭缓存

### 设置缓存结果的最大值最小值

SET GLOBAL query\_cache\_limit=10485760; #10M

SET GLOBAL query\_cache\_min\_res\_unit=41984;

### 查询高速缓冲状态和维护

可以使用下面的语句检查MySQL服务器是否提供查询缓存功能：

mysql> SHOW VARIABLES LIKE 'have\_query\_cache';

+------------------+-------+

| Variable\_name | Value |

+------------------+-------+

| have\_query\_cache | YES |

+------------------+-------+

FLUSH QUERY CACHE：语句来清理查询缓存碎片以提高内存使用性能。该语句不从缓存中移出任何查询。

RESET QUERY CACHE：语句从查询缓存中移出所有查询。FLUSH TABLES语句也执行同样的工作。

SHOW STATUS：为了监视查询缓存性能，使用SHOW STATUS查看缓存状态变量，例如：

mysql> SHOW STATUS LIKE 'Qcache%';

+-------------------------+--------+

| Qcache\_free\_blocks | 36 |

| Qcache\_free\_memory | 138488 |

| Qcache\_hits | 79570 |

| Qcache\_inserts | 27087 |

| Qcache\_lowmem\_prunes | 3114 |

| Qcache\_not\_cached | 22989 |

| Qcache\_queries\_in\_cache | 415 |

| Qcache\_total\_blocks | 912 |

+-------------------------+--------+

QCACHE\_free\_blocks：空闲内存块的数量。

QCACHE\_free\_memory：空闲内存内存的数量。

QCACHE\_hits：查询缓存被访问的次数。

QCACHE\_inserts：加入到缓存的查询数量。

QCACHE\_lowmem\_prunes：由于内存较少从缓存删除的查询数量。

QCACHE\_not\_cached：非缓存查询数(不可缓存，或由于query\_cache\_type设定值未缓存)。

Qcache\_queries\_in\_cache：登记到缓存内的查询的数量。

Qcache\_total\_blocks：查询缓存内的总块数。

## 高速缓存语句要求

下面的两个查询被查询缓存认为是不相同的：

SELECT \* FROM tbl\_name

Select \* from tbl\_name

查询必须是完全相同的(逐字节相同)才能够被认为是相同的。

## 不缓存的语句

如果一个查询包含下面函数中的任何一个，它不会被缓存：

BENCHMARK()

CONNECTION\_ID()

CURDATE()

CURRENT\_DATE()

CURRENT\_TIME()

CURRENT\_TIMESTAMP()

CURTIME()

DATABASE()

带一个参数的ENCRYPT()

FOUND\_ROWS()

GET\_LOCK()

LAST\_INSERT\_ID()

LOAD\_FILE()

MASTER\_POS\_WAIT()

NOW()

RAND()

RELEASE\_LOCK()

SYSDATE()

不带参数的UNIX\_TIMESTAMP()

USER()

# EXPLAIN

## 查看表的索引

SHOW INDEX FROM tbl\_name;

## 创建索引

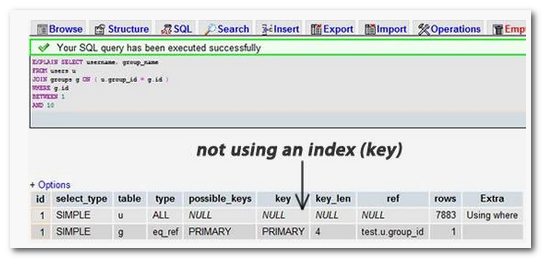
ALTER TABLE 表名 ADD INDEX 索引名 (索引列) ;

## 说明

使用 EXPLAIN 关键字可以让你知道MySQL是如何处理你的SQL语句的。这可以帮你分析你的查询语句或是表结构的性能瓶颈。

EXPLAIN 的查询结果还会告诉你你的索引主键被如何利用的，你的数据表是如何被搜索和排序的……等等，等等。

挑一个你的SELECT语句（推荐挑选那个最复杂的，有多表联接的），把关键字EXPLAIN加到前面。然后，你会看到一张表格。下面的这个示例中，我们忘记加上了group\_id索引，并且有表联接：



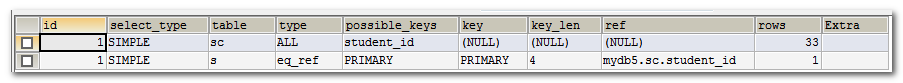
当我们为 group\_id 字段加上索引后：



我们可以看到，前一个结果显示搜索了 7883 行，而后一个只是搜索了两个表的 9 和 16 行。查看rows列可以让我们找到潜在的性能问题。

## 参数





id：这是SELECT的查询序列号。

select\_type：SELECT类型，可以为以下任何一种：

SIMPLE：简单SELECT(不使用UNION或子查询)

PRIMARY：最外面的SELECT

UNION：UNION中的第二个或后面的SELECT语句

DEPENDENT UNION：UNION中的第二个或后面的SELECT语句，取决于外面的查询

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，取决于外面的查询

DERIVED:导出表的SELECT(FROM子句的子查询)

table:输出的行所引用的表。

type:联接类型。下面给出各种联接类型，按照从最佳类型到最坏类型进行排序：

system表仅有一行(=系统表)。

const表最多有一个匹配行，它将在查询开始时被读取。

eq\_ref比较的时候，“=”前后的变量都加了索引。

ref:前面的表加了索引。

index：该联接类型与ALL相同，只是索引树被扫描。

ALL：对于每个来自于先前的表的行组合，进行完整的表扫描。

possible\_keys：possible\_keys列指出MySQL能使用哪个索引在该表中找到行。

如果该列是NULL，则没有相关的索引。在这种情况下，可以通过检查WHERE子句看是否它引用某些列或适合索引的列来提高你的查询性能。

key:显示MySQL实际决定使用的索引。如果没有选择索引，键是NULL。

key\_len:显示MySQL决定使用的索引长度。如果索引是NULL，则长度为NULL。

ref：显示使用哪个列或常数与key一起从表中选择行。

rows：显示MySQL认为它执行查询时必须检查的行数。

Extra：该列包含MySQL解决查询的详细信息。下面解释了该列可以显示的不同的文本字符串：

Distinct：MySQL发现第1个匹配行后，停止为当前的行组合搜索更多的行。

Not exists：MySQL能够对查询进行LEFT JOIN优化，发现1个匹配LEFT JOIN标准的行后，不再为前面的的行组合在该表内检查更多的行。

range checked for each record (index map: #)：MySQL没有发现好的可以使用的索引，但发现如果来自前面的表的列值已知，可能部分索引可以使用。对前面的表的每个行组合，MySQL检查是否可以使用range或index\_merge访问方法来索取行。

Using filesort：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

Using index：从只使用索引树中的信息而不需要进一步搜索读取实际的行来检索表中的列信息。当查询只使用作为单一索引一部分的列时，可以使用该策略。

Using temporary：为了解决查询，MySQL需要创建一个临时表来容纳结果。典型情况如查询包含可以按不同情况列出列的GROUP BY和ORDER BY子句时。

Using where：WHERE子句用于限制哪一个行匹配下一个表或发送到客户。除非你专门从表中索取或检查所有行，如果Extra值不为Using where并且表联接类型为ALL或index，查询可能会有一些错误。

Using sort\_union(...), Using union(...), Using intersect(...)：这些函数说明如何为index\_merge联接类型合并索引扫描。详细信息参见7.2.6节，“索引合并优化”。

Using index for group-by：类似于访问表的Using index方式，Using index for group-by表示MySQL发现了一个索引，可以用来查询GROUP BY或DISTINCT查询的所有列，而不要额外搜索硬盘访问实际的表。并且，按最有效的方式使用索引，以便对于每个组，只读取少量索引条目。详情参见7.2.13节，“MySQL如何优化GROUP BY”。

# 其他优化

## 使用 ENUM 而不是 VARCHAR

ENUM 类型是非常快和紧凑的。在实际上，其保存的是 TINYINT，但其外表上显示为字符串。这样一来，用这个字段来做一些选项列表变得相当的完美。

如果你有一个字段，比如“性别”，“国家”，“民族”，“状态”或“部门”，你知道这些字段的取值是有限而且固定的，那么，你应该使用 ENUM 而不是 VARCHAR。

MySQL也有一个“建议”（见第十条）告诉你怎么去重新组织你的表结构。当你有一个 VARCHAR 字段时，这个建议会告诉你把其改成 ENUM 类型。使用 PROCEDURE ANALYSE() 你可以得到相关的建议。

## 从 PROCEDURE ANALYSE() 取得建议

语法：SELECT \* FROM student LIMIT 1,1 PROCEDURE ANALYSE(1);

PROCEDURE ANALYSE() 会让 MySQL 帮你去分析你的字段和其实际的数据，并会给你一些有用的建议。只有表中有实际的数据，这些建议才会变得有用，因为要做一些大的决定是需要有数据作为基础的。

例如，如果你创建了一个 INT 字段作为你的主键，然而并没有太多的数据，那么，PROCEDURE ANALYSE()会建议你把这个字段的类型改成 MEDIUMINT 。或是你使用了一个 VARCHAR 字段，因为数据不多，你可能会得到一个让你把它改成 ENUM 的建议。这些建议，都是可能因为数据不够多，所以决策做得就不够准。

一定要注意，这些只是建议，只有当你的表里的数据越来越多时，这些建议才会变得准确。

# mysql引擎

MySQL常用的存储引擎为MyISAM、InnoDB、MEMORY、MERGE，其中InnoDB提供事务安全表，其他存储引擎都是非事务安全表。

MyISAM是MySQL的默认存储引擎。MyISAM不支持事务、也不支持外键，但其访问速度快，对事务完整性没有要求。

innoDB存储引擎提供了具有提交、回滚和崩溃恢复能力的事务安全。但是比起MyISAM存储引擎，InnoDB写的处理效率差一些并且会占用更多的磁盘空间以保留数据和索引

MEMORY存储引擎使用存在内存中的内容来创建表。每个MEMORY表只实际对应一个磁盘文件。

MEMORY类型的表访问非常得快，因为它的数据是放在内存中的，并且默认使用HASH索引。但是一旦服务关闭，表中的数据就会丢失掉。

MERGE存储引擎是一组MyISAM表的组合，这些MyISAM表必须结构完全相同。MERGE表本身没有数据，对MERGE类型的表进行查询、更新、删除的操作，就是对内部的MyISAM表进行的。