# MySQL优化实践

## 概述

1. 避免全表扫描，在where和order by 涉及的列上建立索引。
2. 避免where字句中对字段进行null值判断，否则将导致引擎放弃使用索引而进行全表扫描。

例: select id from t where num is null 可以在num列设置默认为0 查询条件改成 num=0。

1. 避免where字句中使用!= 或<> 操作符，否则引擎将放弃使用索引而进行全表扫描。
2. 避免 where 字句中使用 or否则引擎将放弃使用索引而进行全表扫描。可以把or 改成 union all

例: select id from t where num=10 or num =20 改成: select id from t where num=10 union all select id from t where num =20

1. in 和 not in 也要慎用否则会导致全表扫描，对于连续的数值能用between就不要用 in。
2. Like查询 用“%like%” 前后都有匹配符的 无法优化查询 。可以用fulltext.
3. 如果在 where 子句中使用参数，也会导致全表扫描。因为SQL只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然 而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：

select id from t wherenum=@num

可以改为强制查询使用索引：

select id from t with(index(索引名)) wherenum=@num。

8. 应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描 例：

select id from t where num/2=100

应改为:

select id from t where num=100\*2

9．不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。

10.在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能 的让字段顺序与索引顺序相一致。

11.很多时候用 exists 代替 in 是一个好的选择：

select num from a where num in(select num from b)

用下面的语句替换：

select num from a where exists(select 1 from b where num=a.num)

12.并不是所有索引对查询都有效，SQL是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL查询可能不会去利用索引，如一表中有字段sex，male、female几乎各一半，那么即使在sex上建了索引也对查询效率起不了作用。

14.索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有必要。

15.应尽可能的避免更新 clustered 索引数据列，因为 clustered 索引数据列的顺序就是表记录的物理存储顺序，一旦该列值改变将导致整个表记录的顺序的调整，会耗费相当大的资源。若应用系统需要频繁更新 clustered 索引数据列，那么需要考虑是否应将该索引建为 clustered 索引。

16.尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

17.尽可能的使用 varchar/nvarchar 代替 char/nchar ，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

18.任何地方都不要使用 select \* from t ，用具体的字段列表代替“\*”，不要返回用不到的任何字段。

19 .避免频繁创建和删除临时表，以减少系统表资源的消耗。

20.临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使用导出表。

21.在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，然后insert。

22.如果使用到了临时表，在[存储过程](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fblog%2Eknowsky%2Ecom%2F251432%2Ehtm&p=baidu&c=news&n=10&t=tpclicked3_hc&q=sayyescpr&k=%B4%E6%B4%A2%B9%FD%B3%CC&k0=%B4%E6%B4%A2%B9%FD%B3%CC&k1=asp%2Enet&k2=%D4%B4%B4%FA%C2%EB&k3=%B9%DC%C0%ED%CF%B5%CD%B3&k4=%BF%CD%BB%A7%B6%CB&k5=%B0%EC%B9%AB%D7%D4%B6%AF%BB%AF&sid=2e8d6e4946fbc995&ch=0&tu=u1366390&jk=fb80596b19ab0ac8&cf=1&fv=11&stid=9&urlid=0)的最后务必将所有的临时表显式删除，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。

23.尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。

24.使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。

25. 与临时表一样，游标并不是不可使用。对小型数据集使用 FAST\_FORWARD 游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时 间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。

26.在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ，在结束时设置 SET NOCOUNT OFF 。无需在执行存储过程和触发器的每个语句后向[客户端](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fblog%2Eknowsky%2Ecom%2F251432%2Ehtm&p=baidu&c=news&n=10&t=tpclicked3_hc&q=sayyescpr&k=%BF%CD%BB%A7%B6%CB&k0=%BF%CD%BB%A7%B6%CB&k1=%B0%EC%B9%AB%D7%D4%B6%AF%BB%AF&k2=sql&k3=mysql&k4=%CA%FD%BE%DD%BF%E2&k5=server&sid=2e8d6e4946fbc995&ch=0&tu=u1366390&jk=fb80596b19ab0ac8&cf=1&fv=11&stid=9&urlid=0)发送 DONE\_IN\_PROC 消息。

27.尽量避免大事务操作，提高系统并发能力。

28.尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理

## 优化 Order by

在另一些情况下，MySQL无法使用索引来满足 ORDER BY，尽管它会使用索引来找到记录来匹配 WHERE 子句。这些情况如下：   
1. 对不同的索引键做 ORDER BY ：   
 SELECT \* FROM t1 ORDER BY key1, key2;   
2. 在非连续的索引键部分上做 ORDER BY：   
 SELECT \* FROM t1 WHERE key2=constant ORDER BY key\_part2;   
3. 同时使用了 ASC 和 DESC：   
 SELECT \* FROM t1 ORDER BY key\_part1 DESC, key\_part2 ASC;   
4. 用于搜索记录的索引键和做 ORDER BY 的不是同一个：  
 SELECT \* FROM t1 WHERE key2=constant ORDER BY key1;   
5. 有很多表一起做连接，而且读取的记录中在 ORDER BY 中的字段都不全是来自第一个非 常数的表中（也就是说，在 EXPLAIN 分析的结果中的第一个表的连接类型不是 const）。   
6. 使用了不同的 ORDER BY 和 GROUP BY 表达式。  
7. 表索引中的记录不是按序存储。例如，HASH 和 HEAP 表就是这样。

1、ORDER BY + LIMIT组合的索引优化。如果一个SQL语句形如：

|  |
| --- |
| SELECT [column1],[column2],.... FROM [TABLE] ORDER BY [sort] LIMIT [offset],[LIMIT]; |

这个SQL语句优化比较简单，在[sort]这个栏位上建立索引即可。

2、WHERE + ORDER BY + LIMIT组合的索引优化，形如：

|  |
| --- |
| SELECT [column1],[column2],.... FROM [TABLE] WHERE [columnX] = [value] ORDER BY [sort] LIMIT[offset],[LIMIT]; |

这个语句，如果你仍然采用第一个例子中建立索引的方法，虽然可以用到索引，但是效率不高。更高效的方法是建立一个联合索引(columnX,sort)

3、WHERE + IN + ORDER BY + LIMIT组合的索引优化，形如：

|  |
| --- |
| SELECT [column1],[column2],.... FROM [TABLE] WHERE [columnX] IN ([value1],[value2],...) ORDER BY[sort] LIMIT [offset],[LIMIT]; |

这个语句如果你采用第二个例子中建立索引的方法，会得不到预期的效果（仅在[sort]上是using index，WHERE那里是using where;using filesort），理由是这里对应columnX的值对应多个。

这个语句怎么优化呢？我暂时没有想到什么好的办法，看到网上有便宜提供的办法，那就是将这个语句用UNION分拆，然后建立第二个例子中的索引：

|  |
| --- |
| SELECT [column1],[column2],.... FROM [TABLE] WHERE [columnX]=[value1] ORDER BY [sort] LIMIT[offset],[LIMIT]  UNION  SELECT [column1],[column2],.... FROM [TABLE] WHERE [columnX]=[value2] ORDER BY [sort] LIMIT[offset],[LIMIT]  UNION  …… |

但经验证,这个方法根本行不通,效率反而更低,测试时对于大部分应用强制指定使用排序索引效果更好点

4、不要再WHERE和ORDER BY的栏位上应用表达式(函数)，比如：

|  |
| --- |
| SELECT \* FROM [table] ORDER BY YEAR(date) LIMIT 0,30; |

5、WHERE+ORDER BY多个栏位+LIMIT，比如

|  |
| --- |
| SELECT \* FROM [table] WHERE uid=1 ORDER x,y LIMIT 0,10; |

对于这个语句，大家可能是加一个这样的索引(x,y,uid)。但实际上更好的效果是(uid,x,y)。这是由MySQL处理排序的机制造成的。

SELECT msg\_id, msg\_content FROM message ORDER BY gmt\_create desc LIMIT 100, 20;

 可以写成如下方式

Sql代码

SELECT msg\_id, msg\_content FROM message

INNER JOIN (

    SELECT msg\_id FROM message

    ORDER BY gmt\_create LIMIT 100, 20

    ) AS page USING(msg\_id);

 这样当前查询页的内容就只会在索引中进行，当得到当前页的msg\_id再统一通过一个INNER JOIN得到最终要得到的数据详情，避免了对大量数据详情进行操作的消耗。当然JOIN操作也可以通过子查询实现，不过书中介绍5.6之前版本的[mysql](http://www.2cto.com/database/MySQL/)相比子查询还是优先使用JOIN。

## 优化MySQL配置

一．Query\_cache

1. query\_cache\_type 0为不使用查询缓存 1为使用2根据需要使用。
2. query\_cache\_size 缓存查询结果的大小。
3. query\_cache\_limit 单个查询能共使用的缓冲区大小 缺省为1M。
4. 在mysql命令行可以使用 SHOW STATUS LIKE ‘Qcache%’ ，如果出现Qcache\_lowmen\_prunes的值非常大，表明经常出现缓冲不够的情况，如果Qcache\_hits的值非常大，标识查询缓冲使用频繁，此时应该适当增加缓冲的大小，反之很小的话使用查询缓冲反而影响效率，可以不用查询缓冲。在SELECT语句中加入SQL\_NO\_CACHE
5. Query\_cache\_min\_res\_unit 缓冲区空间的最小单位，缺省为4k。
6. 检查Qcache\_free\_blocks 如果该值非常大，则表明缓冲区碎片多，表明查询结果都比较小，此时需要减小query\_cache\_min\_res\_unit

二．key\_buffer\_size

show global status like ‘key\_read%’

key\_read\_requests 27813678764

key\_reads 6798830

表示 有27813678764 读取索引请求 有 6798830 个请求在内存中没有找到直接从硬盘读取索引

索引未命中内存缓存概率key\_cache\_miss\_rate = key\_reads / key\_read\_requests\*100%

key\_cache\_miss\_rate在0.01%以下的话，key\_buffer\_size分配的过多，可以适当减少。

mysql> show global status like 'key\_blocks\_u%';  
+------------------------+-------------+  
| Variable\_name | Value |  
+------------------------+-------------+  
| Key\_blocks\_unused | 0 |  
| Key\_blocks\_used | 413543 |  
+------------------------+-------------+  
Key\_blocks\_unused 表示未使用的缓存簇(blocks)数，Key\_blocks\_used表示曾经用到的最大的blocks数，比如这台服务器，所有的缓存都用到了，要么 增加key\_buffer\_size，要么就是过渡索引了，把缓存占满了。比较理想的设置：  
Key\_blocks\_used / (Key\_blocks\_unused + Key\_blocks\_used) \* 100% ≈ 80%

1. read\_buffer\_size

据说 read\_buffer\_size 参数只影响以下两种情况的写数据的性能：

* SELECT INTO … OUTFILE ‘fileName‘
  + When writing to the OUTFILE, the writes are buffered before writing to OUTFILE
* When [filesort](http://dev.mysql.com/doc/refman/5.1/en/order-by-optimization.html) is used, during merge buffers and when merged results are written to a temporary file, then writes are buffered

1. read\_rnd\_buffer\_size：

用在 sort 之后，以排序后的顺序读取记录时。如果有许多查询都使用了 ORDER BY 函数，增大这个值可以提高性能。需要注意的是，这个值不象 key\_buffer\_size 和 table\_cache ，它的内存分配是对每个线程的，默认会被设置的和 read\_buffer\_size 一样大小，为 128KB，经验值是每1M内存分配1KB，比如3G内存的服务器可以分配3M。

1. sort\_buffer\_size：

这个值非常有助于提高执行 myisamchk 时的速度，所以在配置文件中关于 myisamchk 的配置中这个的值设置的比较大。这个值对于执行大量的 sort 排序操作时也非常有帮助，在my-huge.cnf配置文件中设置为2M，但在我的3G内存的服务器上设置的是9M，排序的查询比较少。

1. tmp\_table\_size：

用于存储临时表的内存空间大小

5．thread\_cache\_size ：

缓存mysql的线程个数

6. **table\_cache：**

允许 MySQL 打开的表的最大个数，并且这些都cache在内存中

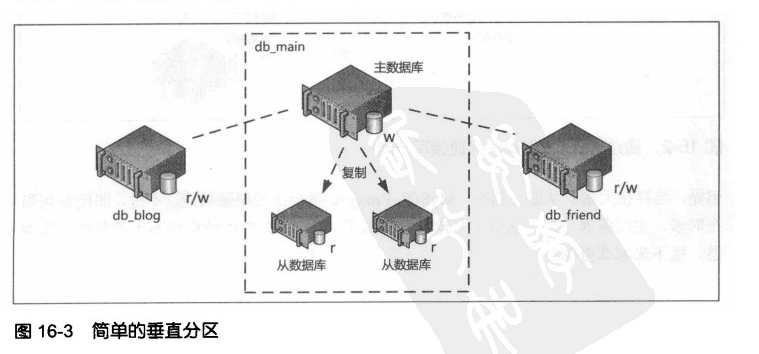
1. tmp\_table\_size：

这个值决定内存中临时表的大小，如果表太大，那么MyISAM表将会创建在磁盘上。尽量优化查询避免使用临时表，如果不行，确保临时表总是创建在内存中。查询 processlist 如果查询使用临时表使用的时间比较长，说明有必要增加tmp\_table\_size 的大小。也需要注意这个值是for 每个线程的。一个例子是我在一个服务器上把这个值从32M增加到64M后立即就有效果。

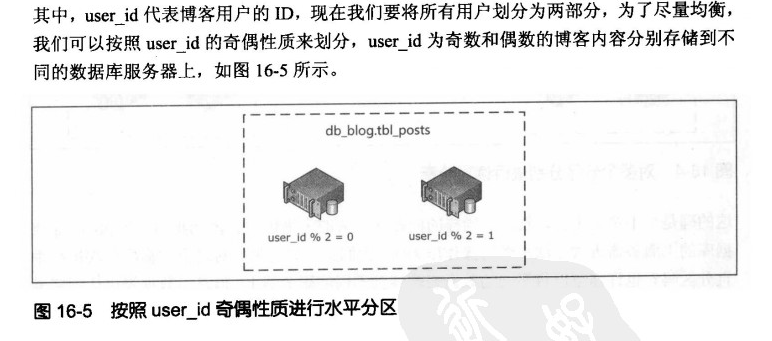
## MySQL服务器扩展

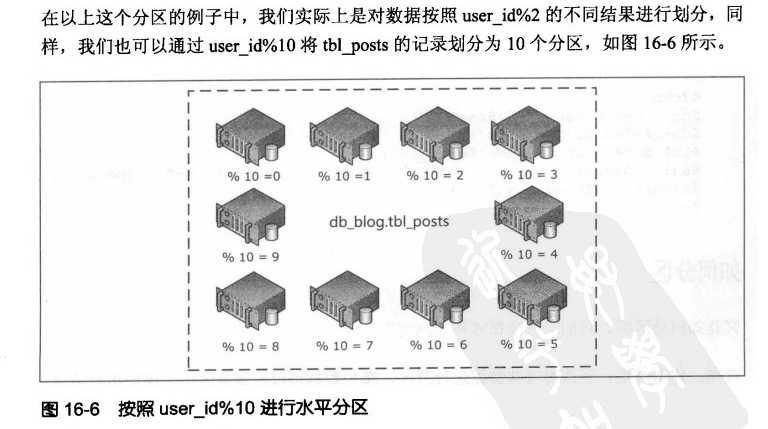
* + 1. 主从配置(主服务器日志重放)

1. 开启主服务器上的二进制日志。
2. 在主服务器和从服务器上分别进行简单的配置和授权。
3. 读写分离 主服务器写入 从服务器读取。
4. 使用数据库反向代理
5. 垂直分区。对于数据库写操作频繁的站点 实行垂直分区。最简单的方法是讲不通的数据库分不到不同的服务器上，因为很多数据库之间并不存在关系。这种方式就是垂直分区



1. 水平分区。意味着我们可以将统一数据表中的记录通过特定的算法进行分离，分别保存在不同的数据表中。





**优化规则:**

1. 优化更需要优化的Query；

2. 定位优化对象的性能瓶颈；

3. 明确的优化目标；

4. 从Explain 入手；

5. 多使用profile

6. 永远用小结果集驱动大的结果集；

7. 尽可能在索引中完成排序；

8. 只取出自己需要的Columns；

9. 仅仅使用最有效的过滤条件；

10. 尽可能避免复杂的Join 和子查询；

SQL\_CALC\_FOUND\_ROWS