**MySQL性能优化的最佳经验，随时补充**

**1、为查询优化你的查询**

大多数的MySQL服务器都开启了查询缓存。这是提高性最有效的方法之一，而且这是被MySQL的数据库引擎处理的。当有很多相同的查询被执行了多次的时候，这些查询结果会被放到一个缓存中，这样，后续的相同的查询就不用操作表而直接访问缓存结果了。

这里最主要的问题是，对于程序员来说，这个事情是很容易被忽略的。因为，我们某些查询语句会让MySQL不使用缓存。请看下面的示例：

// 查询缓存不开启

$r = mysql\_query("SELECT username FROM user WHERE signup\_date >= CURDATE()");

// 开启查询缓存

$today = date("Y-m-d");

$r = mysql\_query("SELECT username FROM user WHERE signup\_date >= '$today'");

上面两条SQL语句的差别就是 CURDATE() ，MySQL的查询缓存对这个函数不起作用。所以，像 NOW() 和 RAND() 或是其它的诸如此类的SQL函数都不会开启查询缓存，因为这些函数的返回是会不定的易变的。所以，你所需要的就是用一个变量来代替MySQL的函数，从而开启缓存。

**2、EXPLAIN 你的SELECT查询**

使用EXPLAIN关键字可以让你知道MySQL是如何处理你的SQL语句的。

有表关联的查询，如下列：

select username, group\_name from users u joins groups g on (u.group\_id = g.id)

发现查询缓慢，然后在group\_id字段上增加索引，则会加快查询

**3、当只要一行数据时使用LIMIT 1**

当你查询表的有些时候，你已经知道结果只会有一条结果，但因为你可能需要去fetch游标，或是你也许会去检查返回的记录数。  
在这种情况下，加上LIMIT 1 可以增加性能。这样一样， MySQL数据库引擎会在找到一条数据后停止搜索，而不是继续往后查找下一条符合记录的数据。  
下面的示例，只是为了找一下是否有“中国”的用户，很明显，后面的会比前面的更有效率。（请注意，第一条中是Select \*，第二条是Select 1）

// 没有效率的：

$r = mysql\_query("SELECT \* FROM user WHERE country = 'China'");

if (mysql\_num\_rows($r) > 0) {

// ...

}

// 有效率的：

$r = mysql\_query("SELECT 1 FROM user WHERE country = 'China' LIMIT 1");

if (mysql\_num\_rows($r) > 0) {

// ...

}

**4、为搜索字段建索引**

索引并不一定就是给主键或是唯一的字段。如果在你的表中，有某个字段你总要会经常用来做搜索，那么，请为其建立索引吧。

**5、在Join表的时候使用相当类型的列，并将其索引**

如果你的应用程序有很多JOIN查询，你应该确认两个表中Join的字段是被建过索引的。这样，MySQL内部会启动为你优化Join的SQL语句的机制。

而且，这些被用来Join的字段，应该是相同的类型的。例如：如果你要把DECIMAL字段和一个INT字段JOIN在一起，MYSQL就无法使用他们的索引。对于那些STRING类型，还需要有相同的字符集才行（两个表的字符集有可能不一样）

**6、千万不要ORDER BY RAND()**

**7、避免SELECT \***

从数据库里读出越多的数据，那么查询就会变得越慢。并且，如果你的数据库服务器和WEB服务器是两台独立的服务器的话，这还会增加网络传输的负载。

所以，你应该养成一个需要什么就取什么的好的习惯。

// 不推荐

$r = mysql\_query("SELECT \* FROM user WHERE user\_id = 1");

$d = mysql\_fetch\_assoc($r);

echo "Welcome {$d['username']}";

// 推荐

$r = mysql\_query("SELECT username FROM user WHERE user\_id = 1");

$d = mysql\_fetch\_assoc($r);

echo "Welcome {$d['username']}";

**8、永远为两张表设置一个ID**

我们应该为数据库里的每张表都设置一个ID作为其主键，而最好的是一个INT型（推荐使用UNSIGNED），并设置上自动增长的AUTO INCREMENT标志。

就算是你 users 表有一个主键叫 “email”的字段，你也别让它成为主键。使用 VARCHAR 类型来当主键会使用得性能下降。另外，在你的程序中，你应该使用表的ID来构造你的数据结构。

而且，在MySQL数据引擎下，还有一些操作需要使用主键，在这些情况下，主键的性能和设置变得非常重要，比如，集群，分区……

**9、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。

**10、从 PROCEDURE ANALYSE() 取得建议 ？**

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。

**11、尽可能的使用 NOT NULL**

除非你有一个很特别的原因去使用 NULL 值，你应该总是让你的字段保持 NOT NULL。这看起来好像有点争议，请往下看。

首先，问问你自己“Empty”和“NULL”有多大的区别（如果是INT，那就是0和NULL）？如果你觉得它们之间没有什么区别，那么你就不要使用NULL。（你知道吗？在 Oracle 里，NULL 和 Empty 的字符串是一样的！)

不要以为 NULL 不需要空间，其需要额外的空间，并且，在你进行比较的时候，你的程序会更复杂。 当然，这里并不是说你就不能使用NULL了，现实情况是很复杂的，依然会有些情况下，你需要使用NULL值。

下面摘自MySQL自己的文档

“NULL columns require additional space in the row to record whether their values are NULL. For MyISAM tables, each NULL column takes one bit extra, rounded up to the nearest byte.”

**12、把IP地址存成 UNSIGNED INT**

很多程序员都会创建一个 VARCHAR(15) 字段来存放字符串形式的IP而不是整形的IP。如果你用整形来存放，只需要4个字节，并且你可以有定长的字段。而且，这会为你带来查询上的优势，尤其是当你需要使用这样的WHERE条件：IP between ip1 and ip2。

我们必需要使用UNSIGNED INT，因为 IP地址会使用整个32位的无符号整形

**13、固定长度的表会更快**

如果表中的所有字段都是“固定长度”的，整个表会被认为是 “static” 或 “fixed-length”。 例如，表中没有如下类型的字段： VARCHAR，TEXT，BLOB。只要你包括了其中一个这些字段，那么这个表就不是“固定长度静态表”了，这样，MySQL 引擎会用另一种方法来处理。

固定长度的表会提高性能，因为MySQL搜寻得会更快一些，因为这些固定的长度是很容易计算下一个数据的偏移量的，所以读取的自然也会很快。而如果字段不是定长的，那么，每一次要找下一条的话，需要程序找到主键。

并且，固定长度的表也更容易被缓存和重建。不过，唯一的副作用是，固定长度的字段会浪费一些空间，因为定长的字段无论你用不用，他都是要分配那么多的空间。

**14、垂直分割**

“垂直分割”是一种把数据库中的表按列变成几张表的方法，这样可以降低表的复杂度和字段的数目，从而达到优化的目的。（以前，在银行做过项目，见过一张表有100多个字段，很恐怖）

示例一：在Users表中有一个字段是家庭地址，这个字段是可选字段，相比起，而且你在数据库操作的时候除了个人信息外，你并不需要经常读取或是改写这个字段。那么，为什么不把他放到另外一张表中呢？ 这样会让你的表有更好的性能，大家想想是不是，大量的时候，我对于用户表来说，只有用户ID，用户名，口令，用户角色等会被经常使用。小一点的表总是会有好的性能。

示例二： 你有一个叫 “last\_login” 的字段，它会在每次用户登录时被更新。但是，每次更新时会导致该表的查询缓存被清空。所以，你可以把这个字段放到另一个表中，这样就不会影响你对用户ID，用户名，用户角色的不停地读取了，因为查询缓存会帮你增加很多性能。

另外，你需要注意的是，这些被分出去的字段所形成的表，你不会经常性地去Join他们，不然的话，这样的性能会比不分割时还要差，而且，会是极数级的下降。

**15、拆分大的 DELETE 或 INSERT 语句**

如果你需要在一个在线的网站上去执行一个大的 DELETE 或 INSERT 查询，你需要非常小心，要避免你的操作让你的整个网站停止相应。因为这两个操作是会锁表的，表一锁住了，别的操作都进不来了。

Apache 会有很多的子进程或线程。所以，其工作起来相当有效率，而我们的服务器也不希望有太多的子进程，线程和数据库链接，这是极大的占服务器资源的事情，尤其是内存。

如果你把你的表锁上一段时间，比如30秒钟，那么对于一个有很高访问量的站点来说，这30秒所积累的访问进程/线程，数据库链接，打开的文件数，可能不仅仅会让你泊WEB服务Crash，还可能会让你的整台服务器马上掛了。

所以，如果你有一个大的处理，你定你一定把其拆分，使用 LIMIT 条件是一个好的方法。下面是一个示例：

while (1) {

//每次只做1000条

mysql\_query("DELETE FROM logs WHERE log\_date <= '2009-11-01' LIMIT 1000");

if (mysql\_affected\_rows() == 0) {

// 没得可删了，退出！

break;

}

// 每次都要休息一会儿

usleep(50000);

}

**16、 越小的列会越快**

对于大多数的数据库引擎来说，硬盘操作可能是最重大的瓶颈。所以，把你的数据变得紧凑会对这种情况非常有帮助，因为这减少了对硬盘的访问。

参看 MySQL 的文档 Storage Requirements 查看所有的数据类型。

如果一个表只会有几列罢了（比如说字典表，配置表），那么，我们就没有理由使用 INT 来做主键，使用 MEDIUMINT, SMALLINT 或是更小的 TINYINT 会更经济一些。如果你不需要记录时间，使用 DATE 要比 DATETIME 好得多。

当然，你也需要留够足够的扩展空间，不然，你日后来干这个事，你会死的很难看，参看Slashdot的例子（2009年11月06日），一个简单的ALTER TABLE语句花了3个多小时，因为里面有一千六百万条数据。

**17、选择一个正确的存储引擎**

在 MySQL 中有两个存储引擎 MyISAM 和 InnoDB，每个引擎都有利有弊。酷壳以前文章《MySQL: InnoDB 还是 MyISAM?》讨论和这个事情。

MyISAM 适合于一些需要大量查询的应用，但其对于有大量写操作并不是很好。甚至你只是需要update一个字段，整个表都会被锁起来，而别的进程，就算是读进程都无法操作直到读操作完成。另外，MyISAM 对于 SELECT COUNT(\*) 这类的计算是超快无比的。

InnoDB 的趋势会是一个非常复杂的存储引擎，对于一些小的应用，它会比 MyISAM 还慢。他是它支持“行锁” ，于是在写操作比较多的时候，会更优秀。并且，他还支持更多的高级应用，比如：事务。

**18、小心“永久链接”**

“永久链接”的目的是用来减少重新创建MySQL链接的次数。当一个链接被创建了，它会永远处在连接的状态，就算是数据库操作已经结束了。而且，自从我们的Apache开始重用它的子进程后——也就是说，下一次的HTTP请求会重用Apache的子进程，并重用相同的 MySQL 链接。

PHP手册：mysql\_pconnect()  
在理论上来说，这听起来非常的不错。但是从个人经验（也是大多数人的）上来说，这个功能制造出来的麻烦事更多。因为，你只有有限的链接数，内存问题，文件句柄数，等等。

而且，Apache 运行在极端并行的环境中，会创建很多很多的了进程。这就是为什么这种“永久链接”的机制工作地不好的原因。在你决定要使用“永久链接”之前，你需要好好地考虑一下你的整个系统的架构。

参考

**19、当查询较慢的时候，可用Join来改写一下该查询来进行优化**

mysql> select sql\_no\_cache \* from guang\_deal\_outs where deal\_id in (select id from guang\_deals where id = 100017151) ;

Empty set (18.87 sec)

mysql> select sql\_no\_cache a.\* from guang\_deal\_outs a inner join guang\_deals b on a.deal\_id = b.id where b.id = 100017151;

Empty set (0.01 sec)

原因

mysql> desc select sql\_no\_cache \* from guang\_deal\_outs where deal\_id in (select id from guang\_deals where id = 100017151) ;

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

| id | select\_type | table | type | possible\_keys | key | key\_len | ref | rows | Extra |

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

| 1 | PRIMARY | guang\_deal\_outs | ALL | NULL | NULL | NULL | NULL | 18633779 | Using where |

| 2 | DEPENDENT SUBQUERY | guang\_deals | const | PRIMARY | PRIMARY | 4 | const | 1 | Using index |

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

2 rows in set (0.04 sec)

mysql> desc select sql\_no\_cache a.\* from guang\_deal\_outs a inner join guang\_deals b on a.deal\_id = b.id where b.id = 100017151;

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

| id | select\_type | table | type | possible\_keys | key | key\_len | ref | rows | Extra |

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

| 1 | SIMPLE | b | const | PRIMARY | PRIMARY | 4 | const | 1 | Using index |

| 1 | SIMPLE | a | ref | idx\_guang\_dlout\_dlid | idx\_guang\_dlout\_dlid | 4 | const | 1 | |

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

2 rows in set (0.05 sec)

其实在 guang\_deal\_outs 在deal\_id 上也是有索引的。  
其实我想把子查询设置为

select \* from guang\_deal\_outs where deal\_id in (select id from guang\_deals where id = 100017151);

变成下面的样子

select \* from guang\_deal\_outs where deal\_id in (100017151);

但不幸的是，实际情况正好相反。MySQL试图让它和外面的表产生联系来“帮助”优化查询，它认为下面的exists形式更有效率

select \* from guang\_deal\_outs where exists (select \* from guang\_deals where id = 100017151 and id = guang\_deal\_outs.deal\_id);

这种in子查询的形式，在外部表（比如上面的guang\_deals）数据量比较大的时候效率是很差的（如果对于较小的表，不会造成显著地影响）

批量插入100万级别数据优化

对于一些数据量较大的系统，数据库面临的问题除了查询效率低下，还有就是数据入库时间长。特别像报表系统，每天花费在数据导入上的时间可能会长达几个小时或十几个小时之久。因此，优化数据库插入性能是很有意义的。

经过对MySQL innodb的一些性能测试，发现一些可以提高insert效率的方法，供大家参考参考。

1. 一条SQL语句插入多条数据

常用的插入语句如：

1. **INSERT** **INTO** `insert\_table` (`datetime`, `uid`, `content`, `type`)
2. **VALUES** ('0', 'userid\_0', 'content\_0', 0);
3. **INSERT** **INTO** `insert\_table` (`datetime`, `uid`, `content`, `type`)
4. **VALUES** ('1', 'userid\_1', 'content\_1', 1);

修改成：

1. **INSERT** **INTO** `insert\_table` (`datetime`, `uid`, `content`, `type`)
2. **VALUES** ('0', 'userid\_0', 'content\_0', 0), ('1', 'userid\_1', 'content\_1', 1);

修改后的插入操作能够提高程序的插入效率。这里第二种SQL执行效率高的主要原因是合并后日志量（MySQL的binlog和innodb的事务让 日志）减少了，降低日志刷盘的数据量和频率，从而提高效率。通过合并SQL语句，同时也能减少SQL语句解析的次数，减少网络传输的IO。

这里提供一些测试对比数据，分别是进行单条数据的导入与转化成一条SQL语句进行导入，分别测试1百、1千、1万条数据记录。

[](http://s3.51cto.com/wyfs01/M00/2C/4B/wKioOVI-VlWi-JzaAAAtsOthv_8566.jpg)

2. 在事务中进行插入处理

把插入修改成：

1. START **TRANSACTION**;
2. **INSERT** **INTO** `insert\_table` (`datetime`, `uid`, `content`, `type`)
3. **VALUES** ('0', 'userid\_0', 'content\_0', 0);
4. **INSERT** **INTO** `insert\_table` (`datetime`, `uid`, `content`, `type`)
5. **VALUES** ('1', 'userid\_1', 'content\_1', 1);
6. ...
7. **COMMIT**;

使用事务可以提高数据的插入效率，这是因为进行一个INSERT操作时，MySQL内部会建立一个事务，在事务内才进行真正插入处理操作。通过使用事务可以减少创建事务的消耗，所有插入都在执行后才进行提交操作。

这里也提供了测试对比，分别是不使用事务与使用事务在记录数为1百、1千、1万的情况。

[](http://s8.51cto.com/wyfs01/M01/2C/4B/wKioOVI-VlWRQHERAAAuZ06k-bc546.jpg)

3. 数据有序插入

数据有序的插入是指插入记录在主键上是有序排列，例如datetime是记录的主键：

1. **INSERT** **INTO** `insert\_table` (`datetime`, `uid`, `content`, `type`)
2. **VALUES** ('1', 'userid\_1', 'content\_1', 1);
3. **INSERT** **INTO** `insert\_table` (`datetime`, `uid`, `content`, `type`)
4. **VALUES** ('0', 'userid\_0', 'content\_0', 0);
5. **INSERT** **INTO** `insert\_table` (`datetime`, `uid`, `content`, `type`)
6. **VALUES** ('2', 'userid\_2', 'content\_2',2);

修改成：

1. **INSERT** **INTO** `insert\_table` (`datetime`, `uid`, `content`, `type`)
2. **VALUES** ('0', 'userid\_0', 'content\_0', 0);
3. **INSERT** **INTO** `insert\_table` (`datetime`, `uid`, `content`, `type`)
4. **VALUES** ('1', 'userid\_1', 'content\_1', 1);
5. **INSERT** **INTO** `insert\_table` (`datetime`, `uid`, `content`, `type`)
6. **VALUES** ('2', 'userid\_2', 'content\_2',2);

由于数据库插入时，需要维护索引数据，无序的记录会增大维护索引的成本。我们可以参照innodb使用的B+tree索引，如果每次插入记录都在索 引的最后面，索引的定位效率很高，并且对索引调整较小；如果插入的记录在索引中间，需要B+tree进行分裂合并等处理，会消耗比较多计算资源，并且插入 记录的索引定位效率会下降，数据量较大时会有频繁的磁盘操作。

下面提供随机数据与顺序数据的性能对比，分别是记录为1百、1千、1万、10万、100万。

[](http://s6.51cto.com/wyfs01/M01/2C/4A/wKioJlI-VlWBA6cvAABH0QHskXA030.jpg)

从测试结果来看，该优化方法的性能有所提高，但是提高并不是很明显。

性能综合测试：

这里提供了同时使用上面三种方法进行INSERT效率优化的测试。

[](http://s6.51cto.com/wyfs01/M01/2C/4A/wKioJlI-VlWAuBWAAAB4F4n0yRk294.jpg)

从测试结果可以看到，合并数据+事务的方法在较小数据量时，性能提高是很明显的，数据量较大时（1千万以上），性能会急剧下降，这是由于此时数据量 超过了innodb\_buffer的容量，每次定位索引涉及较多的磁盘读写操作，性能下降较快。而使用合并数据+事务+有序数据的方式在数据量达到千万级 以上表现依旧是良好，在数据量较大时，有序数据索引定位较为方便，不需要频繁对磁盘进行读写操作，所以可以维持较高的性能。

**注意事项**：

1. SQL语句是有长度限制，在进行数据合并在同一SQL中务必不能超过SQL长度限制，通过max\_allowed\_packet配置可以修改，默认是1M，测试时修改为8M。

2. 事务需要控制大小，事务太大可能会影响执行的效率。MySQL有innodb\_log\_buffer\_size配置项，超过这个值会把innodb的数据刷到磁盘中，这时，效率会有所下降。所以比较好的做法是，在数据达到这个这个值前进行事务提交。