Top 20+ MySQL最佳实践

Edward Lee, 2014-09-15

本文翻译自《[Top 20+ MySQL Best Practices](http://code.tutsplus.com/tutorials/top-20-mysql-best-practices--net-7855)》——Tuts+ Code Tutorial，同时参考《[MySQL性能优化的最佳20+条经验](http://coolshell.cn/articles/1846.html)》的译文。

今天，数据库操作通常是大多数Web应用的**主要性能瓶颈**。不仅DBA需要担心这些性能问题，作为程序员的我们也要做好自己的那部分（正确地构建表，编写优化的查询和更好的代码）。

优化主要涉及**表结构**和**索引**的合理设计（**垂直拆分字段**）、**存储引擎**的正确选择（InnoDB）、**查询缓存**的充分利用、使用**EXPLAIN**查看SQL的执行计划、按需索取数据

下面是一些针对程序员的MySQL优化技术。

1. 为查询缓存优化你的查询

大多数MySQL服务器已启用**查询缓存(Query Cache)**，这是提高性能的最有效方法之一，其是由数据库引擎处理的。当相同的查询被执行多次时，查询结果是从查询缓存中取出，这是相当快速的。

主要问题是，大多数程序员往往忽略它。

// query cache does NOT work (非确定性函数)

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

// query cache **works**! (占位符)

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

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

查询缓存不工作的原因是CURDATE()函数的使用，这会应用到所有非确定性函数（如NOW()、RAND()等）。因为该函数的返回结果是随时变化的，所以MySQL决定对该查询禁用查询缓存。所以，你所需要的就是用一个变量来代替这些函数，从而开启查询缓存。

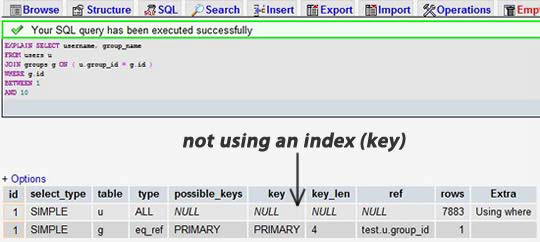
详见“[5.13. MySQL查询高速缓冲](http://dev.mysql.com/doc/refman/5.1/zh/database-administration.html#query-cache)”参考手册

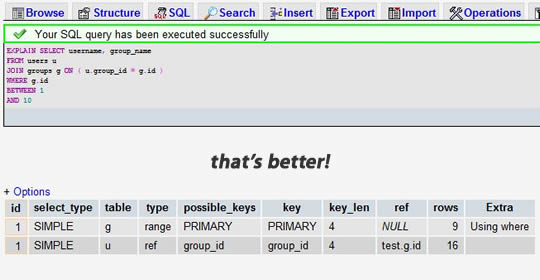
1. EXPLAIN SELECT查询

使用EXPLAIN语句可以获得关于MySQL是如何执行SELECT语句的信息，这可以帮助你分析查询语句或是表结构的性能瓶颈和其它问题。

EXPLAIN的查询结果会告诉你哪些**索引**被使用了、数据表是如何被扫描和排序的 等等信息。

示例：执行一个联接查询(join on)

增加group\_id字段的索引后：

现在不是扫描7883行记录，只会扫描2张表的9和16行记录。经验法则：乘以所有的行数，你的查询性能与这个结果数成正比。

详见“[7.2.1. EXPLAIN语法（获取SELECT相关信息）](http://dev.mysql.com/doc/refman/5.1/zh/optimization.html#explain)”参考手册

1. 当只查询唯一行时加上“LIMIT 1”

在查询数据表之前，就已经知道你要找的只有一行数据。

你可能想抓取一条记录，也可能想检查满足WHERE子句条件的记录是否存在。（应用场景）

对于这样的cases，增加“LIMIT 1”到查询语句可以提高性能。通过这种方式，数据库引擎在找到一条记录后就会停止扫描，而不是扫描整个表或索引。

// do I have any users from Alabama?

// what NOT to do:

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

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

// ...

}

// much better:

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

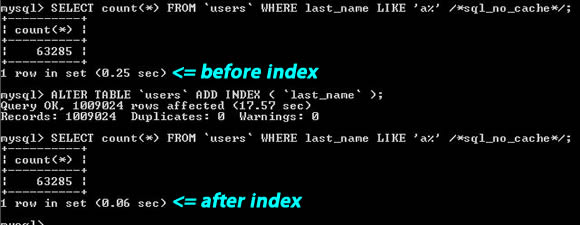
if (mysql\_num\_rows($r) > 0) {

// ...

}

详见“[7.2.14. MySQL如何优化LIMIT](http://dev.mysql.com/doc/refman/5.1/zh/optimization.html#limit-optimization)”参考手册

1. 索引被搜索的字段

**索引(Indexes)**并不一定就是主键(primary key)或唯一键(unique key)。如果在你的表中，有某个字段会经常用来做搜索，那么请为其建立索引吧。

从上图看，这条规则也适用于部分字符串搜索（如last\_name LIKE '**a**%'）。当从字符串的开头搜索时，MySQL是能利用该列的索引。

你还应该了解哪些种类的搜索不能使用常规索引，如WHERE post\_content LIKE '%apple%'。

详见“[7.4.5. MySQL如何使用索引](http://dev.mysql.com/doc/refman/5.1/zh/optimization.html#mysql-indexes)”参考手册

1. 索引并使用相同列类型的联接(Join)

你需要确保，联接的列必须在所有表中都被索引了。这会影响MySQL内部如何优化联接操作

例如，如果你将一个DECIMAL列联接到另一张表的INT列，MySQL是不会使用任何索引的。

// looking for companies in my state

$r = mysql\_query("SELECT company\_name FROM users

LEFT JOIN companies ON (users.state = companies.state)

WHERE users.id = $user\_id");

// **both** state columns should be **indexed**

// and they both should be **the same type and character encoding**

// or MySQL might do full table scans

1. 不用使用“ORDER BY RAND()”

问题在于，MySQL在排序之前会为表中的每一行执行RAND()操作，并只返回给你一行记录。

如果你真的需要查询结果的随机几行记录，最好通过应用代码实现。

详见“[7.2.12. MySQL如何优化ORDER BY](http://dev.mysql.com/doc/refman/5.1/zh/optimization.html#order-by-optimization)”参考手册

1. 避免使用“SELECT \*”

从表中读取的数据越多，查询就会变得越慢。同时，它增加了磁盘操作的时间损耗。还有，当数据库服务器与web服务器分开时，还有增加网络传输的负载，因为数据需要在服务器之间传输。

所以，你应该养成一个需要什么就取什么的好习惯。

// not preferred

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

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

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

// better: 推荐

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

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

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

// the differences are more significant with bigger result sets

1. 永远为每张表设置一个id字段

我们应该为数据库里的每张表都设置一个**id列**作为其自增主键（推荐使用UNSIGNED INT）

有一个例外情况，那就是关联表的外键。

1. 使用ENUM而不是VARCHAR

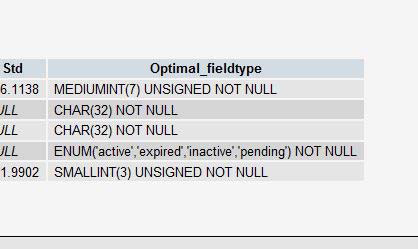
ENUM类型列是非常快和紧凑的。在内部被存储为TINYINT，但其外表上显示为字符串。这样一来，用这个类型来做一些选项列表变得相当的完美。

如何去重新组织你的表结构？使用PROCEDURE ANALYSE()，你可以得到相关的建议。（见第十条）

详见“[7.4.2. 使你的数据尽可能小](http://dev.mysql.com/doc/refman/5.1/zh/optimization.html#data-size)”参考手册

1. 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。

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。一定要记住，你才是最终做决定的人。

1. 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.”

详见“[7.2.7. MySQL如何优化IS NULL](http://dev.mysql.com/doc/refman/5.1/zh/optimization.html#is-null-optimization)”参考手册

1. Prepared Statements

Prepared Statements很像存储过程，是一种运行在后台的**SQL语句集合**。我们可以从使用中获得很多好处，无论是性能问题还是安全问题。

Prepared Statements可以检查一些你绑定好的变量，这样可以保护你的程序不会受到“SQL注入式”攻击。当然，你也可以手动地检查你的这些变量。然而，手动地检查容易出问题，而且经常会被程序员忘了。当我们使用一些框架或是ORM的时候，这样的问题会好一些。

在性能方面，当一个相同的查询被使用多次时，这会为你带来可观的性能优势。你可以给这些Prepared Statements定义一些参数，而MySQL只会解析一次。

同时，最新版本的MySQL在传输Prepared Statements是使用二进制形式，所以这会使得网络传输非常有效。

1. 无缓冲的查询

mysql\_unbuffered\_query()发送一个SQL语句到MySQL而并不像mysql\_query()一样去自动fetch和缓存结果。这会节约相当可观的内存，尤其是那些会产生大量结果的查询语句。并且，你不需要等到所有的结果都返回，只需要第一行数据返回的时候，你就可以开始处理查询结果了。

然而，这会有一些限制。因为你要么把所有行都读走，或是你要在进行下一次的查询前调用mysql\_free\_result()清除结果。而且，mysql\_num\_rows()或mysql\_data\_seek()将无法使用。所以，是否使用无缓冲的查询你需要仔细考虑。

1. 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而不是整型的IP。如果你用整型来存放，只需要4个字节，并且你可以有定长的字段。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。

我们必须要使用UNSIGNED INT，因为IP地址会使用整个32位的无符号整型。而你的查询，可以使用INET\_ATON()来把一个字符串IP转成一个整型，并使用INET\_NTOA()把一个整型转成一个字符串IP。

"UPDATE users SET ip = INET\_ATON($ip) WHERE user\_id = $user\_id"

1. 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，则这个表被认为是“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类型的字段，那么这个表就不是“固定长度的表”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。因为记录是固定长度的，可以很容易计算某一行数据的偏移量，所以读取的自然也会很快（减少随机读写）。而如果字段不是定长的，那么，每一次要找下一条的话，需要先找到主键索引。

同时，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，都是要分配那么多的空间。

使用“垂直拆分(Vertical Partitioning)”技术（见下一条），可以将可变长度的列分割到另一个单独的表。

详见“[7.4.2. 使你的数据尽可能小](http://dev.mysql.com/doc/refman/5.1/zh/optimization.html#data-size)”参考手册

扩展阅读“[Memcached的内存存储结构实现](http://blog.charlee.li/memcached-002/)”

1. 垂直拆分(Vertical Partitioning)

“垂直拆分”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。

**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。相比较，你在操作个人信息时，并不需要经常读取或是修改这个字段。那么，为什么不把它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。大多数时候，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提升很多性能。

另外，你需要注意的是，这些被拆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，而且会是指数级地下降。

1. 拆分大的DELETE或INSERT语句

如果你需要在一个在线的网站上执行一个大的DELETE或INSERT语句，你需要非常的小心，要避免你的操作让整个网站停止相应。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运行着很多的子进程/线程。所以，其工作起来相当有效率，而我们的服务器并不希望有太多的子进程和数据库链接，这是极大地占用服务器资源的事情，尤其是内存。

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

所以，如果你需要删除大量的行，使用**LIMIT**子句做到较小的批量操作是一个很好的方法。

while (1) {

mysql\_query("DELETE FROM logs WHERE log\_date <= '2009-10-01' LIMIT 10000");

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

// done deleting

break;

}

// you can even pause a bit

usleep(50000);

}

详见“[7.2.16. INSERT语句的速度](http://dev.mysql.com/doc/refman/5.1/zh/optimization.html#insert-speed)”、“[7.2.17. UPDATE语句的速度](http://dev.mysql.com/doc/refman/5.1/zh/optimization.html#update-speed)”、“[7.2.18. DELETE语句的速度](http://dev.mysql.com/doc/refman/5.1/zh/optimization.html#delete-speed)”参考手册

1. 越小的列会越快

对于大多数的数据库引擎来说，**硬盘操作**可能是最大的性能瓶颈。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“[第11章：列类型](http://dev.mysql.com/doc/refman/5.1/zh/column-types.html)”来查看所有的数据类型

如果一个表只会有几行记录（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（使用BIGINT更好）。

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](http://beta.slashdot.org/story/75009)（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。

详见“[7.4.2. 使你的数据尽可能小](http://dev.mysql.com/doc/refman/5.1/zh/optimization.html#data-size)”参考手册

1. 选择正确的存储引擎

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？](http://coolshell.cn/articles/652.html)》讨论过这个事情。

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update操作完成。另外，MyISAM对于SELECT COUNT(\*)这类的计算是超级无比的快。

**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。但是它支持“**行锁**”，于是在**写操作比较多**的时候，会更优秀。并且，它还支持更多的高级应用，比如**事务**。

更多存储引擎介绍请详见“[第15章：存储引擎和表类型](http://dev.mysql.com/doc/refman/5.1/zh/storage-engines.html)”参考手册

1. 使用一个对象关系映射器(Object Relational Mapper)

使用ORM(Object Relational Mapper)，你能够获得可靠的性能收益。一个ORM可以做的所有事情，也能被手动的编写出来。但是，这需要一个高级专家。

ORM是伟大的“**Lazy Loading(延迟加载)**”，也就是说，只有在需要取值的时候才会真正地去做。但你也需要小心这种机制的副作用，因为这很有可能会因为要去创建很多很多小的查询反而会降低性能。

ORM还可以把你的SQL语句打包成一个事务，这会比单独执行他们快得多得多。

1. 小心“持久链接(Persistent Connections)”

“持久链接”的目的是用来减少重新创建MySQL链接的次数。当一个链接被创建时，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。

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，如内存问题、文件句柄数等等，因为你只有有限的链接数。

而且，Apache运行在极端并行（高于700并发）的环境中，会创建很多很多的子进程。这就是为什么这种“持久链接”的机制工作地不好的原因。在你决定要使用“持久链接”之前，你需要好好地考虑一下你的整个系统的架构。