# MySQL优化

优化是个非常庞大而复杂的话题，不同的人员所处的位置不同，例如：业务开发人员可以专注于优化SQL语句，DBA不仅优化SQL语句，还要对表的设计，库的设计、存储引擎的选择，服务器配置，索引的建立等等各方面进行优化，运维人员可能需要对操作系统进行优化。等等。总而言之，系统的优化涉及到系统的方方面面和每一个环节，任何一个环节都可能称为瓶颈，单纯的优化系统的某一部分是不行的，很多情况下，都是去解决性能问题，我认为好的DBA应该提前对数据库进行监控，及时发现问题和系统的性能瓶颈，而不是问题发生之后去救火。本章内容主要翻译自MySQL 5.7手册第8章。

## 1、优化概述

数据库性能取决于很多因素，包括应用程序、架构是否合理、数据库层的设计是否合理、操作系统配置是否合理，硬件使用是否合理，在数据库层面，通常有以下几个因素：表的设计是否合理，索引建立的是否合理，SQL语句写的是否合理，数据库服务器的配置是否合理，针对于MySQL还包括存储引擎的选择是否合理。这些都会影响数据库的性能。通常情况下这些会对CPU以及IO有很大的影响。当你考虑数据库性能时，首先应该学习软件方面的高级规则和指导原则，（我也这么认为，软件层面的优化以及合理的配置往往效果会更加明显，比如没建索引的时候建立合适的索引通常会提升很大的性能，但是这时候去考虑硬件层面的优化往往不和适宜）。当你成为专家之后，再去了解更深层次的内部机理，再去了解CPU和IO这些更细的粒度。通常情况下，用户的目标是从现有的软件和硬件配置中获得最佳的数据库性能。少量的高级用户会寻求改进MySQL软件本身的方式来满足更高的性能要求，或开发自己的存储引擎和硬件设备来扩展MySQL生态系统。接下来简要介绍数据库层面的优化和硬件层面的优化。

### 1.1 数据库级优化

数据库应用程序最重要的影响因素是数据库相关的基本设计：

* 表结构是否正确？特别是，列是否具有正确的数据类型，每个表都有适合该表上工作类型的列？例如，被应用程序频繁更新的场景，通常应该是多个表少量的列，而分析大量数据的应用程序操作的表通常具有少量的表，每个表多个列。
* 是否有正确的 [索引](https://dev.mysql.com/doc/refman/5.7/en/optimization-indexes.html)来提高查询效率？
* 是否为每个表使用适当的存储引擎，并合理利用使用的存储引擎的优势和功能？特别地，诸如[InnoDB](https://dev.mysql.com/doc/refman/5.7/en/optimizing-innodb.html) 或非事务性存储引擎 [MyISAM](https://dev.mysql.com/doc/refman/5.7/en/optimizing-myisam.html) 的选择对于性能和可扩展性可能是非常重要的。

**注意：**InnoDB是新表的默认存储引擎。

* 每个表是否使用适当的行格式？这个选择还取决于表的存储引擎。特别地，压缩表使用较少的磁盘空间，因此需要较少的磁盘I/O来读取和写入数据，但是会耗CPU，这就要依据业务特点来进行选择了。InnoDB在各种情况下都支持压缩，MyISAM对于只读的表支持压缩。
* 应用程序是否使用适当的 [锁定策略](https://dev.mysql.com/doc/refman/5.7/en/locking-issues.html)？例如，在可能的情况下允许共享访问，以提高数据库操作的并发度，并在适当时请求独占访问，以使关键的操作成为最高的优先级。另外，存储引擎的选择是重要的。InnoDB存储引擎会处理大部分的锁定问题，而不需要人为的参与，通常会使数据库有较好的并发，同时对应用程序代码的影响会更小。
* 所有[用于缓存的内存区域](https://dev.mysql.com/doc/refman/5.7/en/buffering-caching.html)设置是否正确？也就是说，内存缓冲区要足够大以容纳经常访问的数据，但不能太大，以致它们超过物理内存并导致coredump。要配置的主要内存区域是InnoDB缓冲池（buffer pool），MyISAM的键缓存（key cache）和MySQL查询缓存。

### 1.2 硬件级优化

随着数据库变得越来越繁忙，任何数据库应用程序最终都会受到硬件限制。DBA和系统管理员必须评估是否可以调整应用程序或重新配置服务器以避免这些瓶颈，或者决定是否需要更多的硬件资源。系统瓶颈通常来源于这些因素：

* 磁盘寻道。磁盘需要一定的时间才能找到一个数据。对于现代磁盘，平均时间通常低于10ms，所以理论上在一秒钟可以做约100次寻道。这个时候换用新的磁盘往往性能提升不明显，并且很难优化一个表。优化搜索时间的方式是将数据分发到多个磁盘上，这就可以对多个磁盘同时进行IO操作。但是分发算法将非常关键。
* 磁盘读写。当磁盘处于正确的位置时，我们需要读取或写入数据。使用现代磁盘，一个磁盘提供至少10-20MB / s的吞吐量。这比寻道更容易优化，因为你可以从多个磁盘并行读取。
* CPU周期。当数据在主内存中时，我们必须处理它以获得我们期望的结果。与内存相比较大的表是最常见的瓶颈。
* 内存带宽。当CPU需要的数据比CPU缓存中的数据更多时，主存带宽成为瓶颈。对于大多数系统来说，这是一个罕见的瓶颈，但某些时候也需要注意。

## 2、优化SQL语句

数据库内核的核心逻辑是通过SQL来驱动执行的。因此，对SQL的优化常常具有很大效果。

### 2.1 SELECT 优化

在应用程序以及数据库设计初期，最先考虑的应该是数据库的schema的设计，设计合理的数据库以及表结构，依据业务制定合理的分库分表原则。这些设计好之后，就是在使用过程中对SQL语句的优化了。本节专注于SQL语句中SELECT的优化。优化查询主要考虑如下因素：

* 为了使SELECT ... WHERE查询速度更慢，首先要检查的是是否可以添加 [索引](https://dev.mysql.com/doc/refman/5.7/en/glossary.html#glos_index)。在WHERE子句中使用的列上设置索引，以加快评估，筛选和结果的最终检索。为了避免浪费磁盘空间，请构建一小组索引，以加快应用程序中使用的许多相关查询。索引对于使用诸如[联接](https://dev.mysql.com/doc/refman/5.7/en/glossary.html#glos_join)和 [外键](https://dev.mysql.com/doc/refman/5.7/en/glossary.html#glos_foreign_key)等功能引用不同表的查询特别重要 。您可以使用该[EXPLAIN](https://dev.mysql.com/doc/refman/5.7/en/explain.html)语句来确定哪些索引用于a [SELECT](https://dev.mysql.com/doc/refman/5.7/en/select.html)。请参见 [第8.3.1节“MySQL如何使用索引”](https://dev.mysql.com/doc/refman/5.7/en/mysql-indexes.html)和 [第8.8.1节“使用EXPLAIN优化查询”](https://dev.mysql.com/doc/refman/5.7/en/using-explain.html)。
* 隔离和调整查询的任何部分，例如函数调用，这需要花费很多时间。根据查询的结构方式，可以为结果集中的每一行调用一次函数，或者甚至对表中的每一行调用一次函数，从而大大放大了任何低效率。
* 最小化 查询中的[全表扫描](https://dev.mysql.com/doc/refman/5.7/en/glossary.html#glos_full_table_scan)数 ，特别是对于大表。
* 通过[ANALYZE TABLE](https://dev.mysql.com/doc/refman/5.7/en/analyze-table.html)定期使用该语句来使表统计信息保持最新状态 ，因此优化器具有构建高效执行计划所需的信息。
* 了解每个表的存储引擎特有的调优技术，索引技术和配置参数。双方InnoDB并 MyISAM有两套准则的实现和维持查询高性能。有关详细信息，请参见[第8.5.6节“优化InnoDB查询”](https://dev.mysql.com/doc/refman/5.7/en/optimizing-innodb-queries.html)和 [第8.6.1节“优化MyISAM查询”](https://dev.mysql.com/doc/refman/5.7/en/optimizing-queries-myisam.html)。
* 您可以InnoDB使用[第8.5.3节“优化InnoDB只读事务”中](https://dev.mysql.com/doc/refman/5.7/en/innodb-performance-ro-txn.html)的技术来优化表的 单查询事务 。
* 避免以难以理解的方式转换查询，特别是如果优化程序自动执行某些相同的转换。
* 如果性能问题不能通过其中一个基本准则解决，请通过阅读[EXPLAIN](https://dev.mysql.com/doc/refman/5.7/en/explain.html)计划并调整索引，WHERE子句，连接条款等来调查特定查询的内部详细信息 。（当达到一定的专业水平时，阅读 [EXPLAIN](https://dev.mysql.com/doc/refman/5.7/en/explain.html)计划可能是您的每个查询的第一步。）
* 调整MySQL用于缓存的内存区域的大小和属性。通过有效利用 InnoDB [缓冲池](https://dev.mysql.com/doc/refman/5.7/en/glossary.html#glos_buffer_pool)， MyISAM密钥缓存和MySQL查询缓存，重复的查询运行速度更快，因为在第二次和随后的时间内从内存中检索结果。
* 即使对于使用高速缓存内存区域快速运行的查询，您仍然可以进一步优化，以便减少缓存内存，从而使应用程序更具可扩展性。可扩展性意味着您的应用程序可以处理更多同步的用户，更大的请求等，而不会大大降低性能。
* 处理锁定问题，您的查询速度可能会受到其他会话同时访问表的影响。

2.1.1 优化where条件

本节讨论where子句的优化，示例使用SELECT，同样适用于DELETE和UPDATE.

注意：因为MySQL的优化器一直在开发进步，因此MySQL优化的技术不一定全都包含在这里。

为了使算术运算更快，你可能会尝试重写你的查询，但同时会牺牲可读性。MySQL通常会对这些方面做优化，因此，通常你不需要牺牲可读性和可维护性来重写查询。例如：

* 删除不必要的括号：

((a AND b) AND c OR (((a AND b) AND (c AND d))))

-> (a AND b AND c) OR (a AND b AND c AND d)

* 常数折叠：

(a<b AND b=c) AND a=5 -> b>5 AND b=c AND a=5

* 恒定条件去除：

(B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6) -> B=5 OR B=6

* 索引使用的常量表达式仅计算一次。
* 在没有WHERE条件的情况下，对一个表执行COUNT(\*)的时候，对于MyISAM 和MEMORY表MySQL直接从表信息中检索。
* 提早判断无效的常数表达式，从而确定一些不会返回任何行的SELECT，这时MySQL将直接返回。
* 在没有使用GROUP BY或者聚合函数（COUNT(), MIN()等）的情况下，HAVING和WHERE将合并处理。
* 对于JOIN中的每个表，构建更加简单的WHERE以便更快速的跳过行。
* 查询中，MySQL在任何其他表之前读取所有常量表。常量表是以下任何一种：
  + 一张空表或者包含一行的表。
  + 与PRIMARY KEY或UNIQUE索引上的WHERE子句一起使用的表，其中所有索引部分与常量表达式进行比较，并定义为NOT NULL。

以下所有表都用作常量表：

SELECT \* FROM t WHERE *primary\_key*=1;

SELECT \* FROM t1,t2 WHERE t1.*primary\_key*=1 AND t2.*primary\_key*=t1.id;

这个优化是非常明显的，例如对于上边的第二个查询：如果先读取第二个表中满足条件的数据，则需要对t2中每一行数据做判断，而且此时t1中也可以有多行，这就导致过滤where条件的时间为count(t2) \* count(t1)，然后再过滤t1表中的数据。如果先读取常数表，则马上判断t1中的一行数据，然后第二个条件也成了一个主键上的等值查询了，这会非常快的执行。将时间复杂度从count(t2) \* count(t1)下降到了lg（count（t1））。

2.1.2 范围优化

LOAD DATA INFILE '/home/fubo.xfb/t\_test\_load\_time.csv' INTO TABLE media\_member\_total\_00\_lz4 FIELDS TERMINATED BY ';';

CREATE TABLE `test000` (

`memberid` bigint(20) DEFAULT NULL,

`wangwang` varchar(256) DEFAULT NULL COMMENT 'lz4',

`nick` varchar(128) DEFAULT NULL COMMENT 'lz4',

`active\_site\_count` bigint(20) DEFAULT NULL,

`active\_site\_tanx\_count` bigint(20) DEFAULT NULL,

`active\_adzone\_tanx\_count` bigint(20) DEFAULT NULL,

`active\_site\_taoke\_count` bigint(20) DEFAULT NULL,

`active\_adzone\_taoke\_count` bigint(20) DEFAULT NULL,

`status` varchar(32) DEFAULT NULL COMMENT 'lookup',

`buyer\_star\_id` int(11) DEFAULT NULL,

`buyer\_star` varchar(64) DEFAULT NULL COMMENT 'lookup',

`buyer\_type` int(11) DEFAULT NULL,

`media\_type` int(11) DEFAULT NULL,

`added\_time` varchar(64) DEFAULT NULL,

`cooperate\_time` int(11) DEFAULT NULL,

`baoyang\_media` int(11) DEFAULT NULL,

`vip\_media` int(11) DEFAULT NULL,

`taobao\_registertime` varchar(64) DEFAULT NULL COMMENT 'lz4',

`mama\_registertime` varchar(64) DEFAULT NULL COMMENT 'lz4',

`apv` bigint(20) DEFAULT NULL,

`auv` bigint(20) DEFAULT NULL,

`aclick` bigint(20) DEFAULT NULL,

`ecpm` double DEFAULT NULL,

`ctr` double DEFAULT NULL,

`tanx\_income` double DEFAULT NULL,

`points\_tanx` bigint(20) DEFAULT NULL,

`points\_taoke` bigint(20) DEFAULT NULL,

`violation\_tanx\_count` bigint(20) DEFAULT NULL,

`violation\_taoke\_count` bigint(20) DEFAULT NULL,

`click\_taoke` bigint(20) DEFAULT NULL,

`uv` bigint(20) DEFAULT NULL,

`landing\_pv` bigint(20) DEFAULT NULL,

`taoke\_order\_fee` double DEFAULT NULL,

`finally\_fee` double DEFAULT NULL,

`order\_count` bigint(20) DEFAULT NULL,

`alimm\_fee` double DEFAULT NULL,

`thedate` varchar(20) DEFAULT NULL COMMENT 'lookup',

`finance\_from\_ggz` double DEFAULT NULL,

`pub\_income\_before` double DEFAULT NULL,

`wireless\_mm\_income` double DEFAULT NULL,

`init\_query` bigint(20) DEFAULT NULL,

`show\_pv` bigint(20) DEFAULT NULL,

`wireless\_aclick` bigint(20) DEFAULT NULL,

`ppc` double DEFAULT NULL,

`wireless\_actr` double DEFAULT NULL,

`apv\_ratio` double DEFAULT NULL,

`tk\_grade` varchar(64) DEFAULT NULL COMMENT 'lookup',

`tk\_current\_score` varchar(64) DEFAULT NULL COMMENT 'lookup',

`tk\_current\_uv\_score` bigint(20) DEFAULT NULL,

`tk\_current\_commission\_score` double DEFAULT NULL,

`tk\_current\_shopcount\_score` double DEFAULT NULL,

`tk\_last\_score` varchar(64) DEFAULT NULL COMMENT 'lookup',

`tk\_last\_uv\_score` bigint(20) DEFAULT NULL,

`tk\_last\_commission\_score` double DEFAULT NULL,

`tk\_last\_shopcount\_score` double DEFAULT NULL,

`tk\_grade\_update` varchar(64) DEFAULT NULL COMMENT 'lookup',

`tk\_grade\_potential` varchar(64) DEFAULT NULL COMMENT 'lookup'

) ENGINE=HISTORE DEFAULT CHARSET=utf8;