# 影响mysql场景因素

## 服务器硬件

**CPU：**一般情况下CPU资源不会是性能瓶颈的直接原因；MySQL不支持多cpu对同一SQL并发处理。

**内存：**直接影响MySQL缓冲池的大小及MySQL数据库的整体运行稳定性；如内存资源不足，容易造成MySQL的会话拥堵甚至实例重启。

**存储IO：**直接影响MySQL的处理性能；在大量数据变更的业务场景下，对存储的IO性能要求往往较高。

## 数据库存储引擎

**MyISAM：**不支持事务型查询，在OLTP类型业务场景中不建议使用。

**InnoDB：**支持事务型查询，支持行级锁，对并发业务支持较好。

# Mysql参数

1）连接请求的参数：max\_connections

MySQL的最大连接数，增加该值增加mysqld要求的文件描述符的数量。连接请求量大时，建议调高此值调的越高，内存开销越大。

mysql>show variables like 'max\_connections';

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

|Variable\_name|Value|

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

|max\_connections|512 |

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

mysql>show status like 'max%connections';

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

|Variable\_name|Value|

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

|max\_used\_connections|512 |

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

2）全局缓存参数

key\_buffer\_size指定索引缓冲区的大小，它决定索引处理的速度，尤其是索引读的速度。Key\_reads是内存中没有找到索引直接从硬盘读取索引的数量。

mysql>show variables like' key\_buffer\_size'; 

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

|Variable\_name|Value|

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

|key\_buffer\_size|536870912 |

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

mysql>show status like 'key\_read%';

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

|Variable\_name|Value|

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

|Key\_read\_requests|178306331520 |

|Key\_reads|67 |

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

使用查询缓冲，MySQL将查询结果存放在缓冲区中，今后对于同样的SELECT语句（区分大小写），将直接从缓冲区中读取结果。

mysql>show variables like ' key\_buffer\_size';

mysql>show status like ' key\_read%';

查询缓存碎片率= Qcache\_free\_blocks/ Qcache\_total\_blocks\* 100%

查询缓存利用率= (query\_cache\_size–Qcache\_free\_memory) / query\_cache\_size\* 100%

查询缓存命中率= (Qcache\_hits–Qcache\_inserts) / Qcache\_hits\* 100%

3）每个连接的缓存参数

①Sort\_buffer\_size

每个需要进行排序的线程分配该大小的一个缓冲区。增加这值加速ORDER BY或GROUP BY操作。默认数值是2097144(2M)，可改为16777208 (16M)。

②Join\_buffer\_size

联合查询操作所能使用的缓冲区大小。

record\_buffer\_size，read\_rnd\_buffer\_size，sort\_buffer\_size，join\_buffer\_size为每个线程独占，也就是说，如果有100个线程连接，则占用为16M\*100。

③ table\_open\_cache

表高速缓存的大小。每当MySQL访问一个表时，如果在表缓冲区中还有空间，该表就被打开并放入其中，这样可以更快地访问表内容。

mysql> show global status like 'open%tables%';

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

| Variable\_name| Value |

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

| Open\_tables| 1024 |

| Opened\_tables| 1465 |

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

mysql>showvariableslike'table\_open\_cache';

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

|Variable\_name|Value|

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

|table\_open\_cache|1024|

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

④tmp\_table\_size

临时表大小。通过设置tmp\_table\_size选项来增加一张临时表的大小，例如做高级GROUP BY操作生成的临时表。

mysql>showglobal statuslike' created\_tmp%';

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

|Variable\_name|Value |

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

|Created\_tmp\_disk\_tables|21197|

| Created\_tmp\_files| 58|

| Created\_tmp\_tables| 1771587 |

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

mysql> show variables like 'tmp\_table\_size';

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

| Variable\_name| Value |

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

| tmp\_table\_size| 16777216 |

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

⑤ thread\_cache\_size

可以复用的保存在缓冲区中的线程的数量。当客户端断开之后，服务器处理此客户的线程将会缓存起来以响应下一个客户而不是销毁（前提是缓存数未达上限）。

mysql>show global status like 'Thread%';

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

|Variable\_name|Value|

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

|Threads\_cached|31|

|Threads\_connected|239|

|Threads\_created|2914|

|Threads\_running|4|

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

mysql>show variables like 'thread\_cache\_size';

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

|Variable\_name|Value|

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

|thread\_cache\_size|32|

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

4）配置InnoDB的参数

① Innodb\_buffer\_pool\_size

InnoDB使用该参数指定大小的内存来缓冲数据和索引，其对InnoDB的重要性等于key\_buffer\_size对MyISAM的重要性。

② Innodb\_log\_buffer\_size

Innodb\_log缓存大小，一般为1-8M，默认为1M，对于较大的事务，可以增大缓存大小。可设置为4M或8M。

5）慢查询参数：log\_slow\_queries

# 数据库表设计

**表体量过大：**字段过多或者记录数过多的“大表”，在查询中会消耗大量资源，且执行效率低；**建议根据业务类型拆分大表（分区表）。**

**使用外键：**无论是MySQL还是Oracle，都不建议采用外键进行表关联。

**缺少主键：**无论对于主从同步还是查询性能，主键发挥的作用都非常重要；建议所有业务表都添加主键。

# SQL语句

**多表关联：**多表关联容易造成关联数据过大，影响查询效率；建议查询中的关联表数量不超过2个。

**全表扫描：**触发全表扫描容易造成大量IO读写，严重降低查询效率；建议在查询条件中加入带索引的过滤条件。

根据现网环境优化执行的难易度，在优化顺序可以按照：**SQL语句->数据库表设计->数据库参数配置->数据库存储引擎->服务器硬件。**

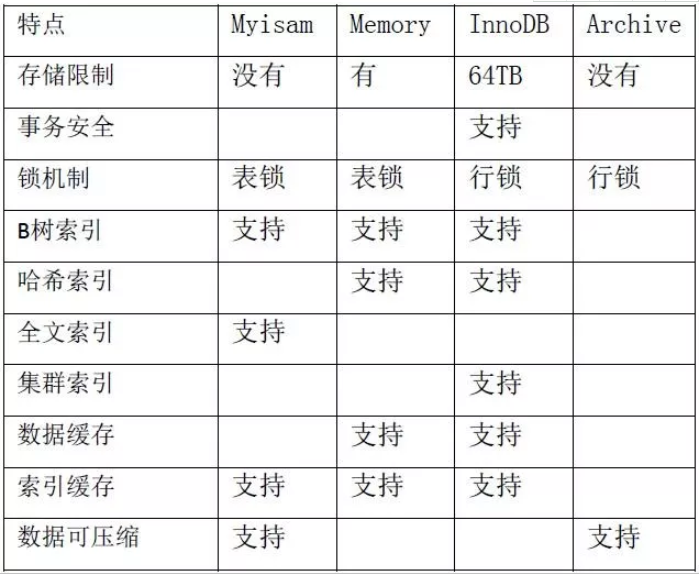
下文我们重点论述上面第四、第五点，通过编写高效的SQL语句，并以合适的方式创建表和索引，使系统始终保持良好的性能。

# 表设计建议

以合适的方式建立表，可以提高数据库运行效率，有效降低历史数据清理时的维护工作难度。

## 选定存储引擎

MySQL支持多种存储引擎，在处理不同类型的应用时，可以通过选择使用不同的存储引擎提高应用的效率，或者提供灵活的存储。MySQL的存储引擎包括：MyISAM、 InnoDB、BDB、MEMORY、MERGE、EXAMPLE、NDB Cluster、ARCHIVE、CSV、BLACKHOLE、FEDERATED等。下面是几种常用的存储引擎的对比和推荐使用方式。



其中，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。其设计目的主要面向在线事务处理（OLTP）及应用。但是对比 Myisam的存储引擎，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。从MySQL5.5版本开始，InnoDB存储引擎是默认的存储引擎。Myisam存储引擎不支持事务，表锁设计，支持全文索引，主要面向一些OLAP数据库应用及Web应用。每个MyISAM在磁盘上存储成三个文件。文件名都和表名相同，扩展名分别是.frm（存储表定义）、.MYD (MYData，存储数据)、.MYI (MYIndex，存储索引)。数据文件和索引文件可以放置在不同的目录，平均分布IO，获得更快的速度。在移动云生产环境中我们建议所有业务表必须是innodb表。

## 表命名规范

1）命名大小写规范：在 MySQL 中，数据库对应数据目录中的目录。数据库中的每个表至少对应数据库目录中的一个文件(也可能是多个，取决于存储引擎)。因此，所使用操作系统的大小写敏感性决定了数据库名和表名的大小写敏感性。这说明在大多数 Unix 中数据库名和表名对大小写敏感，而在 Windows 中对大小写不敏感。MySQL有配置参数lower\_case\_table\_names，不可动态更改，linux系统默认为 0，即库表名以实际情况存储，大小写敏感。如果是1，以小写存储，大小写不敏感。如果是2，以实际情况存储，但以小写比较。MySQL5.6默认为0。若大小写混合使用，易导致使用及管理混乱，且字段名显式区分大小写，但实际使用不区分，即不可以建立两个名字一样但大小写不一样的字段。因此，建议为了统一规范, 库名、表名、字段名使用小写字母，连接统一用下划线‘\_’。

2）命名字符长度规范：库名、表名、字段名支持最多64个字符，但为了统一规范、易于辨识以及减少传输量，禁止超过32个字符。

3）避免使用MySQL保留字：当库名、表名、字段名等属性含有保留字时，SQL语句必须用反引号引用属性名称，这将使得SQL语句书写、SHELL脚本中变量的转义等变得非常复杂。

## 建立常规表

MySQL常规表对应到文件系统上单个数据文件。在MySQL5.6中建表时，不指定任何参数，默认会建立存储引擎为innodb的常规表。常规表使用与大部分应用场景。默认情况下，由于部分操作系统对文件大小的限制，表大小限制为2G。

## 建立分区表

MySQL从5.1版本开始支持分区表，从5.6开始MySQL表分区以单个数据文件形式存储于文件系统中，根据所使用的不同分区规则可以分成几大类型：

**RANGE 分区：**基于属于一个给定连续区间的列值，把多行分配给分区。比较常用如按照时间字段划分分区，2019年1月的数据放到201901分区，2019年2月的数据放到201902分区以此类推。范围分区方式适用于应用中频繁对分区键值进行范围查询的场合。另外针对部监控表随时间不断累积数据，大量的历史数据积压，一方面会降低应用程序的效率，另一方面亦浪费大量的存储空间。因此需要对历史表进行定期清理，以基本保持当前总数据量。基于这个原则，建议对所有历史表按清理时间键值进行范围分区，时间范围建议按月进行。表分区的命名采用以下的规范：<表名>\_pYYYYMMDD，其中YYYY为分区数据的年份，MM为分区数据的月份，DD为分区数据的日期。

**LIST 分区：**类似于按RANGE分区，区别在于LIST分区是基于列值匹配一个离散值集合中的某个值来进行选择。列值分区与范围分区有类似之处，该分区与范围分区类似的是需要指定列的值，但是其分区值必须明确指定。

**HASH分区：**基于用户定义的表达式的返回值来进行选择的分区，该表达式使用将要插入到表中的这些行的列值进行计算。这个函数可以包含MySQL中有效的、产生非负整数值的任何表达式。此种分区方式最适用于查询条件中，对分区字段进行单值查询的情况(如，col=1)。但是hash分区，并不适用于对索引字段使用范围查询，如对字段使用大于>，小于<，操作的查询语句中。

**KEY分区：**类似于按HASH分区，区别在于KEY分区只支持计算一列或多列，且MySQL服务器提供其自身的哈希函数。必须有一列或多列包含整数值。

**复合分区：** 基于RANGE/LIST 类型的分区表中每个分区的再次分割。子分区可以是 HASH/KEY 等类型。

## 表字段规范

* 尽量使用TINYINT、SMALLINT、MEDIUM\_INT作为整数类型而非INT，如果非负则加上UNSIGNED；
* VARCHAR的长度只分配真正需要的空间；
* 使用枚举或整数代替字符串类型；
* 尽量使用TIMESTAMP而非DATETIME；
* 单表不要有太多字段，建议在20以内；
* 避免使用NULL字段，很难查询优化且占用额外索引空间；
* 用整型来存IP。

## 统一字符集

系统、服务端、客户端、库、表、开发程序端需统一字符集，通常中英文环境用utf8。

# 表使用建议

根据MySQL的表建立规范，以及在实际维护中的表使用经验相结合，对表使用作出如下的建议。

## 选择合适的数据类型

InnoDB 存储引擎和数据列。建议使用 varchar类型：对于InnoDB数据表，内部的行存储格式没有区分固定长度和可变长度列（所有数据行都使用指向数据列值的头指针），因此在本质上，使用固定长度的 char列不一定比使用可变长度varchar列简单。因而，主要的性能因素是数据行使用的存储总量。由于CHAR平均占用的空间多于varchar，因此使用varchar来最小化需要处理的数据行的存储总量和磁盘I/O是比较好的。

## text和blob

在使用text和blob字段类型时要注意以下几点,以便更好的发挥数据库的性能：

**1）text和blob值在执行了大量的删除或更新操作的时候容易影响效率。**

删除该类型值会在数据表中留下很大的"空洞"，以后填入这些"空洞"的记录可能长度不同,为了提高性能,建议定期使用 OPTIMIZE TABLE 功能对这类表进行碎片整理。

**2）使用合成的（synthetic）索引。**

合成的索引列在某些时候是有用的。一种办法是根据其它的列的内容建立一个散列值，并把这个值存储在单独的数据列中。之后可以通过检索散列值找到数据。但是，这种索引只能用于精确匹配的查询（散列值对于类似<或>=等范围搜索操作符 是没有用处的）。可以使用MD5()函数生成散列值，也可以使用SHA1()或CRC32()，或者使用自己的应用程序逻辑来计算散列值。需注意数值型散列值可以很高效率地存储。同样，如果散列算法生成的字符串带有尾部空格，此时不要把它们存储在char与varchar列中，它们会受到尾部空格去除的影响。合成的散列索引对于那些text和blob数据列特别有用。用散列标识符值查找的速度比搜索blob列本身的速度快很多。

3）**把text或blob列分离到单独的表中。**

通过把这些数据列移动到单独的数据表中，可以让你把原数据表中的数据列转换为固定长度的数据行格式。这会减少主表中的碎片，使你得到固定长度数据行的性能优势。此时能避免在主数据表上运行 SELECT \*查询的时候通过网络传输大量的text或blob值。

## 拆分大字段、访问频率低的字段

将大字段、访问频率低的字段拆分到单独的表中存储,分离冷热数据。有利于有效利用缓存，防止读入无用的冷数据，较少磁盘IO，同时保证热数据常驻内存提高缓存命中率。

## 数据文件磁盘分离

MySQL表以数据文件形式存储于文件系统，针对不同的表的读写会打开不同的数据文件。建议对不同的热表进行存储的磁盘分离。通过将不同的热表建立在不同的lun上，分散I/O，这样就能进一步减少I/O消耗的瓶颈。

# 索引建立规范

建立合适的索引，是提高数据库运行效率的一个很好的工具，这种效果是立竿见影的，但这里也不并不是说表上的索引越多越好，过之而不及。在数据库设计过程中，需要为表选择一些合适的索引。在数据库中索引的维护代价是表的3倍，宁缺勿滥，这是建立索引时的一个遵循标准。

# 索引使用规范

根据MySQL的索引使用经验相结合，对索引使用做出如下的建议。

## 根据表数据量评估索引

详细评估和分析建立索引所在表的实际数据量，数据量达到GB级别、记录数达到百万级别、访问频繁的表，需要建立合适的索引。相反，在数据量较少且访问频率不高的情况下，如只有一百行记录以下的表不需要建立索引。因为在数据量少的情况下，使用全表扫描效果比走索引更好。

## 选择适当的索引字段

索引字段的选择需要结合业务需求，评估出应用中作为查询条件出现比较频繁的字段，在此字段上建立单独或者复合索引。选择建立索引的字段，应该遵循以下的原则：

1）高选择性，选择性是指通过索引字段查询返回结果集占表总数据量的百分比，结果集占表总数据量的百分比越小选择性越高，反之越低。选择性越高，通过索引查询返回的结果集越少，索引更为高效。在OLTP应用系统中，选择性应高于1，也就是结果集占表总数据量的百分比应<1%。

2）空值少，避免在空值(Null)很多的字段上建立B-tree索引，大量空值会降低索引效率，索引字段中的空值占总数据量的百分比应少于10%。

3）数据分布均匀，索引字段中，个别数据值占总数据量的百分率明显比其它数据值占总数据量的百分率高，表明该字段数据值分布不均，容易引起数据库选择错误索引，生成错误的查询执行计划。应该避免在数据值分布不均的字段上建立索引。

## 避免过度索引

每个额外的索引都要占用额外的磁盘空间，并降低写操作的性能，这一点我们前面已经介绍过。在修改表的内容时，索引必须进行更新，有时可能需要重构，因此，索引越多，所花的时间越长。如果有一个索引很少利用或从不使用，那么会不必要地减缓表的修改速度。此外，MySQL在生成一个执行计划时，要考虑各个索引，这也要费时间。创建多余的索引给查询优化带来了更多的工作。索引太多，也可能会使 MySQL 选择不到所要使用的最好索引。只保持所需的索引有利于查询优化。如果想给已索引的表增加索引，应该考虑所要增加的索引是否是现有多列索引的最左索引。如果是，则就不要费力去增加这个索引了，因为已经有了。

## 使用唯一索引

考虑某列中值的分布。对于唯一值的列，索引的效果最好，而具有多个重复值的列，其索引效果最差。例如，存放年龄的列具有不同值，很容易区分各行。而用来记录性别的列，只含有“ 男”和“女”，则对此列进行索引没有多大用处（不管搜索哪个值，都会得出大约一半的行）。

## 使用短索引

如果对串列进行索引，应该指定一个前缀长度，只要有可能就应该这样做。例如，如果有一个 CHAR(200) 列，如果在前10个或20个字符内，多数值是惟一的，那么就不要对整个列进行索引。对前10个或20个字符进行索引能够节省大量索引空间，也可能会使查询更快。较小的索引涉及的磁盘 I/O 较少，较短的值比较起来更快。更为重要的是，对于较短的键值，索引高速缓存中的块能容纳更多的键值，因此，MySQL也可以在内存中容纳更多的值。这增加了找到行而不用读取索引中较多块的可能性。

## 利用符合索引前置列

在创建一个 n 列的索引时，实际是创建了 MySQL可利用的 n 个索引。多列索引可起几个索引的作用，因为可利用索引中最左边的列集来匹配行。这样的列集称为最左前缀。（这与索引一个列的前缀不同，索引一个列的前缀是利用该的前 n 个字符作为索引值。） 例如：(a,b,c)、(a,b),后者为冗余索引。当SQL的where条件包含a，b时，能正确的走前一索引，后者作为冗余没有建立的必要。关键在于找到适合的前置列，可以避免建冗余的索引。

## 考虑在列上进行的比较类型

索引可用于“ <”、“ < = ”、“ = ”、“ > =”、“ >”和 BETWEEN 运算。在模式具有一个直接量前缀时，索引也用于 LIKE 运算。如果只将某个列用于其他类型的运算时（如 STRCMP( )），对其进行索引没有价值。

## 高效SQL编写规范建议

1. 大批量插入数据

如果同时执行大量的插入，建议使用多个值的INSERT语句(方法二)。这比使用分开INSERT语句快（方法一），一般情况下批量插入效率有几倍的差别。

方法一：

insert into tablename values(1,2);

insert into tablename values(1,3);

insert into tablename values(1,4);

方法二：

Insert into tablename values(1,2),(1,3),(1,4);

选择后一种方法的原因有二。

* 减少SQL语句解析的操作， MySQL没有类似Oracle的share pool，采用方法二，只需要解析一次就能进行数据的插入操作；
* SQL语句较短，可以减少网络传输的IO。

此外，还有以下建议提高插入性能：

* 通过使用 INSERT DELAYED 语句得到更高的速度。Delayed 的含义是让 insert 语句马上执行，其实数据都被放在内存的队列中，并没有真正写入磁盘；
* 这比每条语句分别插入要快的多，但需要注意，DELAYED关键字只用于MyISAM，MEMORY这类只支持表锁的存储引擎；
* 将索引文件和数据文件分在不同的磁盘上存放（利用建表中的选项）

## ****查询优先还是更新（insert、update、delete）优先****

MySQL 还允许改变语句调度的优先级，它可以使来自多个客户端的查询更好地协作，这样单个客户端就不会由于锁定而等待很长时间。改变优先级还可以确保特定类型的查询被处理得更快。我们首先应该确定应用的类型，判断应用是以查询为主还是以更新为主的，是确保查询效率还是确保更新的效率，决定是查询优先还是更新优先。下面我们提到的改变调度策略的方法主要是针对只存在表锁的存储引擎，比如 MyISAM 、MEMROY、MERGE，对于Innodb 存储引擎，语句的执行是由获得行锁的顺序决定的。MySQL 的默认的调度策略可用总结如下：

1）写入操作优先于读取操作。

2）对某张数据表的写入操作某一时刻只能发生一次，写入请求按照它们到达的次序来处理。

3）对某张数据表的多个读取操作可以同时地进行。MySQL 提供了几个语句调节符，允许你修改它的调度策略：

* LOW\_PRIORITY关键字应用于DELETE、INSERT、LOAD DATA、REPLACE和UPDATE；
* HIGH\_PRIORITY关键字应用于SELECT和INSERT语句；
* DELAYED关键字应用于INSERT和REPLACE语句。

如果写入操作是一个 LOW\_PRIORITY（低优先级）请求，那么系统就不会认为它的优先级高于读取操作。在这种情况下，如果写入者在等待的时候，第二个读取者到达了，那么就允许第二个读取者插到写入者之前。只有在没有其它的读取者的时候，才允许写入者开始操作。这种调度修改可能存在 LOW\_PRIORITY写入操作永远被阻塞的情况。SELECT 查询的HIGH\_PRIORITY（高优先级）关键字也类似。它允许SELECT 插入正在等待的写入操作之前，即使在正常情况下写入操作的优先级更高。另外一种影响是，高优先级的 SELECT 在正常的 SELECT 语句之前执行，因为这些语句会被写入操作阻塞。如果希望所有支持LOW\_PRIORITY 选项的语句都默认地按照低优先级来处理，那么 请使用--low-priority-updates 选项来启动服务器。通过使用 INSERTHIGH\_PRIORITY 来把 INSERT 语句提高到正常的写入优先级，可以消除该选项对单个INSERT语句的影响。

## 避免出现select \*

select \* 操作在任何类型数据库中都不是一个好的SQL开发习惯。使用select \* 取出全部列，会让优化器无法完成索引覆盖扫描这类优化，会影响优化器对执行计划的选择，也会增加网络带宽消耗，更会带来额外的I/O,内存和CPU消耗。建议评估业务实际需要的列数，指定列名以取代select \*。

* 规范：Select col1,col2,col3… from t1;
* 不规范：Select \* from t1。

## ****避免使用insert..selec****t****..语句****

当使用insert...select...进行记录的插入时，如果select的表是innodb类型的，不论insert的表是什么类型的表，都会对select的表的纪录进行锁定。对于那些从Oracle迁移过来的应用，需要特别的注意，因为Oracle并不存在类似的问题，所以在Oracle的应用中insert...select...操作非常常见。例如：有时候会对比较多的纪录进行统计分析，然后将统计的中间结果插入到另外一个表，这样的操作因为进行的非常少，所以可能并没有设置相应的索引。

如果迁移到MySQL数据库后不进行相应的调整，那么在进行这个操作期间，对需要select的表实际上是进行的全表扫描导致的所有记录的锁定，将会对应用的其他操作造成非常严重的影响。

究其主要原因，是因为MySQL在实现复制的机制时和Oracle是不同的，如果不进行select表的锁定，则可能造成从数据库在恢复期间插入结果集的不同，造成主从数据的不一致。如果不采用主从复制，关闭binlog并不能避免对select纪录的锁定。如果使用这个binlog进行从数据库的恢复，或者进行主数据库的灾难恢复，都将可能和主数据库的执行效果不同。

因此，我们并不推荐通过设置这个参数来避免insert...select...导致的锁，如果需要进行可能会扫描大量数据的insert...select操作，我们推荐使用select...into outfile和load data infile的组合来实现，这样是不会对纪录进行锁定的。

例子：  
INSERT INTO SMAP\_HISTORY.SMAP2\_SESSION (SESSION\_ID,SESSION\_TICKET\_ID) SELECT S.SESSION\_ID,S.SESSION\_TICKET\_ID FROM SMAP.SMAP2\_SESSION S WHERE SESSION\_SID = #sessionId#;

以上语句会对表SMAP2\_SESSION施加表锁，而由于业务上该表存在大量insert语句，业务压力大的时候极易造成严重的阻塞。

## 适当使用commit

适当使用commit可以释放事务占用的资源而减少消耗，commit后能释放的资源如下：

* 事务占用的undo数据块；
* 事务在redo log中记录的数据块；
* 释放事务施加的，减少锁争用影响性能。特别是在需要使用delete删除大量数据的时候，必须分解删除量并定期commit。

## ****减少表的锁冲突****

对 Innodb 类型的表：

1）首先要确认，在对表获取行锁的时候，要尽量的使用索引检索纪录，如果没有使用索引访问，那么即便你只是要更新其中的一行纪录，也是全表锁定的。要确保 sql 是使用索引来访问纪录的，必要的时候，请使用 explain 检查 sql 的执行计划，判断是否按照预期使用了索引。

2）由于 MySQL 的行锁是针对索引加的锁，不是针对纪录加的锁，所以虽然是访问不同行的纪录，但是如果是相同的索引键，是会被加锁的。应用设计的时候也要注意，这里和 Oracle 有比较大的不同。

3）当表有多个索引的时候，不同的事务可以使用不同的索引锁定不同的行，当表有主键或者唯一索引的时候，不是必须使用主键或者唯一索引锁定纪录，其他普通索引同样可以用来检索纪录，并只锁定符合条件的行。

4）如果要使用锁定读，（SELECT ... FOR UPDATE 或 ... LOCK IN SHARE MODE），尝试用更低的隔离级别，比如 READ COMMITTED。

## ****使用SQL\_BUFFER\_RESULT减少锁定时间****

将强制 MySQL 生成一个临时结果集。只要所有临时结果集生成后，所有表上的锁定均被释放。这能在遇到表锁定问题时或要花很长时间将结果传给客户端时有所帮助。当处理一个会让客户端耗费点时间才能处理的大结果集时，可以考虑使用SQL\_BUFFER\_RESULT 提示字。这样可以告诉MySQL将结果集保存在一个临时表中，这样可以尽早的释放各种锁。需注意，该参数不能用于子查询中以及union之后 语法：SELECT SQL\_BUFFER\_RESULT …

## ****正确使用hint优化语句****

MySQL中可以使用hint指定优化器在执行时选择或忽略特定的索引。一般而言，处于版本变更带来的表结构索引变化，更建议避免使用hint，而是通过Analyze table多收集统计信息。但在特定场合下，指定hint可以排除其他索引干扰而指定更优的执行计划。

1）USE INDEX 在你查询语句中表名的后面，添加 USE INDEX 来提供希望 MySQL 去参考的索引列表，就可以让 MySQL 不再考虑其他可用的索引。例子: SELECT col1 FROM table USE INDEX (mod\_time, name)...

2）IGNORE INDEX 如果只是单纯的想让 MySQL 忽略一个或者多个索引，可以使用 IGNORE INDEX 作为 Hint。例子: SELECT col1 FROM table IGNORE INDEX (priority) ...

3）FORCE INDEX 为强制 MySQL 使用一个特定的索引，可在查询中使用FORCE INDEX 作为Hint。例子: SELECT col1 FROM table FORCE INDEX (mod\_time) ...

## ****优化group by语句****

默认情况下，MySQL 排序所有 “GROUP BY col1，col2，....;” 查询的方法如同在查询中指定 “ORDER BY col1，col2，...;” 如果显式包括一个包含相同的列的 ORDER BY子句，MySQL 可以毫不减速地对它进行优化，尽管仍然进行排序。

如果查询包括 GROUP BY 但你想要避免排序结果的消耗，你可以指定 ORDER BY NULL禁止排序。例如：

SELECT a, COUNT(1) FROM table GROUP BY a ORDER BY NULL ;

## ****优化order by语句****

在某些情况中，MySQL 可以使用一个索引来满足 ORDER BY 子句，而不需要额外的排序。**where 条件和 order by 使用相同的索引，并且 order by 的顺序和索引顺序相同 ，并且 order by 的字段都是升序或者都是降序。**

例如：下列 SQL 可以使用索引。

SELECT col1 FROM t1 ORDER BY key\_part1,key\_part2,... ;

SELECT col1 FROM t1 WHERE key\_part1=1 ORDER BY key\_part1 DESC, key\_part2 DESC; SELECT col1 FROM t1 ORDER BY key\_part1 DESC, key\_part2 DESC;

以上复合索引包含字段key\_part1，key\_part2...

但是以下情况不使用索引（由于order by 的字段混合 ASC和 DESC 。）：

SELECT col1 FROM t1 ORDER BY key\_part1 DESC, key\_part2 ASC；

## ****优化join语句****

MySQL中可以通过子查询来使用 SELECT 语句来创建一个单列的查询结果，然后把这个结果作为过滤条件用在另一个查询中。使用子查询可以一次性的完成很多逻辑上需要多个步骤才能完成的 SQL 操作，同时也可以避免事务或者表锁死，并且写起来也很容易。但是，有些情况下，子查询可以被更有效率的连接(JOIN)..替代。

例子：假设要将所有没有订单记录的用户取出来，可以用下面这个查询完成：

SELECT col1 FROM customerinfo WHERE CustomerID NOT in (SELECT CustomerID

FROM salesinfo )

如果使用连接(JOIN).. 来完成这个查询工作，速度将会有所提升。尤其是当 salesinfo表中对 CustomerID 建有索引的话，性能将会更好，查询如下：

SELECT col1 FROM customerinfo

LEFT JOIN salesinfoON customerinfo.CustomerID=salesinfo.CustomerID

WHERE salesinfo.CustomerID IS NULL

连接(JOIN).. 之所以更有效率一些，是因为 MySQL 不需要在内存中创建临时表来完成这个逻辑上的需要两个步骤的查询工作。

## **优化or条件**

对于 or 子句，如果要利用索引，则or 之间的每个条件列都必须用到索引；如果没有索引，则应该考虑增加索引。

## ****优化union查询****

MySQL通过创建并填充临时表的方式来执行union查询。除非确实要消除重复的行，否则建议使用union all。原因在于如果没有all这个关键词，MySQL会给临时表加上distinct选项，这会导致对整个临时表的数据做唯一性校验，这样做的消耗相当高。

高效：

SELECT COL1, COL2, COL3

FROM TABLE

WHERE COL1 = 10

UNION ALL

SELECT COL1, COL2, COL3 FROM TABLE WHERE COL3= 'TEST';

低效：

SELECT COL1, COL2, COL3

FROM TABLE WHERE COL1 = 10

UNION

SELECT COL1, COL2, COL3 FROM TABLE WHERE COL3= 'TEST';

## ****拆分复杂SQL为多个小SQL，避免大事务****

* 简单的SQL容易使用到MySQL的QUERY CACHE；
* 减少锁表时间特别是使用MyISAM存储引擎的表；
* 可以使用多核CPU。

## ****使用truncate代替delete****

当删除全表中记录时，使用delete语句的操作会被记录到undo块中，删除记录也记录binlog，当确认需要删除全表时，会产生很大量的binlog并占用大量的undo数据块，此时既没有很好的效率也占用了大量的资源。使用truncate替代，不会记录可恢复的信息，数据不能被恢复。也因此使用truncate操作有其极少的资源占用与极快的时间。另外，使用truncate可以回收表的水位。

## ****使用合理的分页方式以提高分页效率****

使用合理的分页方式以提高分页效率 针对展现等分页需求，合适的分页方式能够提高分页的效率。

**案例1：**

select \* from t

where thread\_id = 10000

and deleted = 0

order by gmt\_create asc limit 0, 15;

上述例子通过一次性根据过滤条件取出所有字段进行排序返回。数据访问开销=索引IO+索引全部记录结果对应的表数据IO。因此，该种写法越翻到后面执行效率越差，时间越长，尤其表数据量很大的时候。

适用场景：当中间结果集很小（10000行以下）或者查询条件复杂（指涉及多个不同查询字段或者多表连接）时适用。

**案例2：**

select t.\* from (

select id from t

where thread\_id = 10000 and deleted = 0 order by gmt\_create asc limit 0, 15) a, t

where a.id = t.id;

上述例子必须满足t表主键是id列，且有覆盖索引secondary key:(thread\_id, deleted, gmt\_create)。通过先根据过滤条件利用覆盖索引取出主键id进行排序，再进行join操作取出其他字段。数据访问开销=索引IO+索引分页后结果（例子中是15行）对应的表数据IO。因此，该写法每次翻页消耗的资源和时间都基本相同，就像翻第一页一样。

适用场景：当查询和排序字段（即where子句和order by子句涉及的字段）有对应覆盖索引时，且中间结果集很大的情况时适用。

## ****避免不走索引的各种场景****

在下面的SQL语句中的WHERE子句不使用索引：

1）条件中有or，且or左右列并非全部由索引 Select col1 from table where key1=1 or no\_key=2

2）like查询以%开头

3）where条件仅包含复合索引非前置列

Select col1 from table where key\_part2=1 and key\_part3=2

索引包含key\_part1，key\_part2，key\_part3三列，但SQL语句没有包含索引前置列。

4）隐式类型转换造成不使用索引

Select col1 from table where key\_varchar=123;

上述语句由于索引对列类型为varchar，但给定的值为数值，涉及隐式类型转换，造成不能正确走索引。

5）避免对索引字段进行计算

避免对索引字段进行任何计算操作，对索引字段的计划操作会让索引的作用失效，令数据库选择其他的较为低效率的访问路径。

6）避免对索引字段进行是否NULL值判断

避免使用索引列值是否可为空的索引，如果索引列值可以是空值，在SQL语句中那些要返回NULL值的操作，将不会用到索引。

7）避免对索引字段不等于符号

使用索引列作为条件进行查询时，需要避免使用<>或者!=等判断条件。如确实业务需要，使用到不等于符号，需要在重新评估索引建立，避免在此字段上建立索引，改由查询条件中其他索引字段代替。

## ****避免重复查询更新的数据****

针对业务中经常出现的更新行同时又希望获得改行信息的需求，MySQL并不支持PostgreSQL那样的UPDATE RETURNING语法，在MySQL中可以通过变量实现。

例如，更新一行记录的时间戳，同时希望查询当前记录中存放的时间戳是什么，简单方法实现：

Update t1 set time=now() where col1=1;

Select time from t1 where id =1;

使用变量，可以重写为以下方式：

Update t1 set time=now () where col1=1 and @now: = now ();

Select @now;

前后二者都需要两次网络来回，但使用变量避免了再次访问数据表，特别是当t1表数据量较大时，后者比前者快很多。

## ****避免出现不确定结果的函数****

特定针对主从复制这类业务场景。由于原理上从库复制的是主库执行的语句，使用如now()、rand()、sysdate()、current\_user()等不确定结果的函数很容易导致主库与从库相应的数据不一致。另外不确定值的函数,产生的SQL语句无法利用QUERY CACHE。

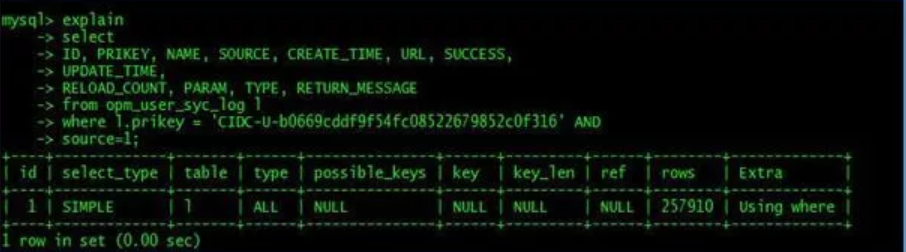
# 使用EXPLAIN分析SQL性能

## 执行计划

执行计划是一条查询语句在数据库中的执行过程或访问路径的描述。

## 怎样查看MySQL执行计划

在需要查看执行计划的SQL前面添加explain并执行，即可获取。



## 读EXPLAIN中的信息

1）table

显示这一行的数据是关于哪张表的。

2）type

这是重要的列，显示连接使用了何种类型。

从最好到最差的连接类型为const、eq\_reg、ref、range、index和ALL。

3）possible\_keys

显示可能应用在这张表中的索引。如果为空，没有可能的索引。

4）key

实际使用的索引。如果为NULL，则没有使用索引。很少的情况下，MYSQL会选择优化不足的索引。

5）key\_len

使用的索引的长度。在不损失精确性的情况下，长度越短越好。

6）ref

显示索引的哪一列被使用了，如果可能的话，是一个常数。

7）rows

MYSQL认为必须检查的用来返回请求数据的行数。

8）Extra

关于MYSQL如何解析查询的额外信息。效率最低的是Using temporary和Using filesort，意味着MYSQL根本不能使用索引，所以检索会很慢。

## 使用EXPLAIN中分析SQL性能



原语句执行的时候是全表扫面，分页查询效率低，改为利用主键来获取后，执行时间由33秒变为1.2秒，效率提升明显。