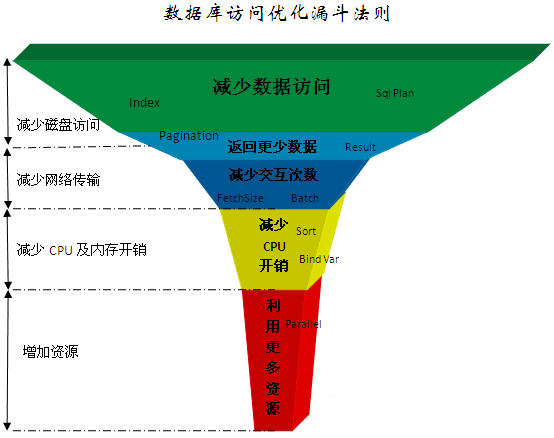
**MySQL数据库性能优化**

1. **简介**

今天，数据库的操作越来越成为整个应用的性能瓶颈了，这点对于Web应用尤其明显。关于数据库的性能，当我们去设计数据库表结构，对操作数据库时，尤其是查表时的SQL语句，我们都需要注意数据操作的性能。这里我们主要从SQL优化、数据库对象优化两个方面介绍一下MySQL的基本优化思路。要正确的优化SQL，我们需要快速定位能性的瓶颈点，也就是说快速找到我们SQL主要的开销在哪里？下图为网上大家公认的数据库基本优化图：



从上图中的信息，我们可以归纳出来优化数据库的五个方向：

1、减少数据访问（减少磁盘访问）

2、返回更少数据（减少网络传输或磁盘访问）

3、减少交互次数（减少网络传输）

4、减少服务器CPU开销（减少CPU及内存开销）

5、利用更多资源（增加硬件资源）

上面五个原则，按类别分，主要是从两个角度对整个数据库进行优化：前四个方法都是从程序设计的角度，最后一个是通过简单的增加资源来优化。其每一个优化法则所带来的性能提升比例不一样，并且对于不同的优化方向，其可优化手段也有多有少，优化成本也高有底。从经验上来看，上面五个原则从上到下，其可供DBA使用的优化手段是从多到少的，优化的资源开销是从低到高的。所以，我们任何一个数据库的性能优化都应该按这个规则由上到下来诊断问题并提出解决方案，而不应该首先想到的是增加资源解决问题。

下面主要从SQL语句、数据库索引、数据库对象、存储引擎四个方面分析一下如何优化数据库。这几个方面的优化，都是从不同的角度，来达到上面所归纳的五点原则的前四点原则的效果。

1. **设计和SQL优化**

**2.1 如何确定低效率的SQL**

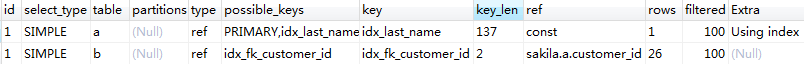
**2.1.1 explain分析低效的执行计划**

MySQL为数据库用户提供了EXPLAIN和DESC命令来查看查询语句的执行信息，包括查询语句在执行过程中表如何连接和连接的顺序等信息，通过这些信息，可以帮助数据库开发人员选择更好的索引和写出更优化的查询语句。下面是一个简单的查询语句通过explain分析后得到的结果：

SQL执行语句：

C:\Users\20160301301\Desktop\捕获.PNG捕获

查询计划：



其各项信息如下：

Id：查询序列号，类似于主键

select\_type：选择的类型，常见的取值有SIMPLE(简单表)、PRIMARY(主查询)、UNION(UNION中的后面的查询语句)

Table：输出结果集的表

Partitions：匹配的分区

Type：连接类型，表示MySQL找到所需行的方式，有如下重要的类型

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ALL | index | range | index\_subquery | unique\_subquery | index\_merge |
| 全表扫描 | 索引全扫描 | 索引范围扫描 | 非唯一 | 唯一索引子查询 | 索引合并优化 |
| ref\_or\_null | fulltext | ref | eq\_ref | const | system |
| 同ref，有null | 全文索引 | 匹配单独值 | 唯一索引 | 单表匹配一行 | 同const |

上面的表格中从左到右从上到校，其type类型对应的效率越来越好，一般来说只要某个索引扫描的连接类型可以达到第二行中任何一个，就是很理想的查询了。

possible\_keys：可以供选择的索引

key ：查询中实际使用到的索引

key\_len：使用到的索引的长度

Ref：显示使用哪个列或常数与key一起从表中选择行

Rows：EMYSQL执行查询的行数，简单且重要，数值越大越不好，说明没有用好索引

Filtered：返回结果的行占需要读到的行的百分比，仅供参考使用

Extra：MySQL解决查询的详细信息

**2.1.2 show status 了解SQL的执行频率**

MySQL 客户端连接成功后，通过show [session|global]status 命令可以提供服务器状态信息，也可以在操作系统上使用mysqladmin extended-status 命令获得这些消息。show

[session|global] status 可以根据需要加上参数“session”或者“global”来显示session 级（当前连接）的统计结果和global 级（自数据库上次启动至今）的统计结果。如果不写，默认使用参数是“session”。

这其中显示的统计参数比较多，比较有参考意义的是如下几组参数：

Com\_select：执行SELECT操作的次数，一次查询只累加

Com\_insert：执行INSERT 操作的次数，对于批量插入的INSERT 操作，只累加一次

Com\_update：执行UPDATE 操作的次数

Com\_delete：执行DELETE 操作的次数

另外也有类似Innodb\_rows\_read的参数可以单独查看INNODB的增删改查的情况，通过以上几个参数，可以很容易地了解当前数据库的应用是以插入更新为主还是以查询操作为主，以及各种类型的SQL 大致的执行比例是多少。

对于事务型的应用，通过Com\_commit 和Com\_rollback 可以了解事务提交和回滚的情况，对于回滚操作非常频繁的数据库，可能意味着应用编写存在问题。此外，以下几个参数便于用户了解数据库的基本情况。

Connections：试图连接MySQL 服务器的次数

Uptime：服务器工作时间

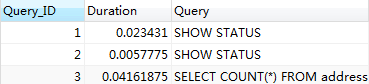
Slow\_queries：慢查询的次数

**2.1.3 show profile分析SQL**

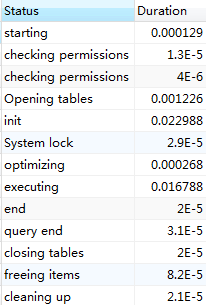
MySQL从5.0.37版本以后开始提供show profile功能，通过SELECT @@profiling可以查看该功能是否开启，默认该功能是关闭的。

通过profile功能，数据库使用者可以更清晰的了解到SQL执行的过程。例如下面一个简单的查询语句。

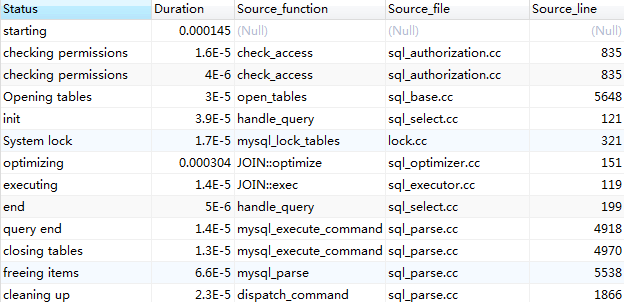
先执行：SELECT COUNT(\*) FROM address;

在执行：SHOW PROFILES;

可以看到当前的查询ID为3：SHOW PROFILE;可以看到执行过程中，线程的每个状态的消耗的时间。



在获取到最消耗时间的线程状态后，进一步可以查看其在ALL、CPU、BLOCK IO、CONTEXT SWITCH等明细类型上的消耗时间，从而为下一步的优化提供依据。例如如果想查看每个步骤对应的源码的文件和解析过程，可以使用命令：SHOW PROFILE SOURCE FOR QUERY 3;就可以看到上面的各个子状态在CPU上面消耗的时间。



**2.1.4 慢查询日志**

慢查询可以帮助我们定位到特定的SQL语句进行SQL语句层面的优化，慢查询日志会记录那些执行时间超过设定值的SQL语句，从而便于我们定位到问题的所在。

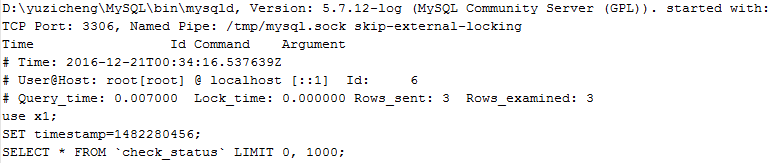
通过以下命令可以查看慢查询的配置：SHOW VARIABLES LIKE '%slow%';

slow\_query\_log：开启或关闭

slow\_query\_log\_file：慢查询日志的存放位置

long\_query\_time：查询时间的最大值，超过这个时间就会被慢查询日志记录

设置好上述几个值以后，当我们在对数据库进行增删改查时，当某条语句的执行时间超过了long\_query\_time的值，慢查询日志文件中就会记录，我们只需要查看慢查询日志，就可以定位到执行效率低的SQL，从而进行优化。



**2.2 常用SQL的优化方法**

1. **大批量数据插入**

采用顺序主键策略，例如自增主键，或者修改业务逻辑，让插入的记录尽可能顺序主键，这是因为数据是按照主键的顺序保存的；采用多值表（10条）插入方式最为合适；将进程/线程数控制在2倍CPU数目相对合适。

对于MyISAM的表，可以以以下方式关闭唯一索引快速导入大量数据：

ALTER table table\_name DISABLE KEYS；

Loading the data；

ALTER table table\_name ENABLE KEYS；

导入非空表时，可以使用如上方式，导入到空表，由于空表是先插入数据在建立唯一索引的，所以不用这样设置。

对于innodb的表，除最开始的三条结论，还可以在插入数据时关闭唯一检验和自动提交，从而提高插入速度。

1. **优化insert语句**

如果同一用户插入多行数据，应使用多个值表的insert语句，可以大大减少数据库连接和关闭的时间消耗

如果不同的用户插入多行，可以使用到insert delayed语句，让插入的数据先保存在内存中，这样多个用户对表的操作可以打包放在内存中一同插入，大大提高速度。

1. **优化排序语句**

MySQL有两种排序方式，一种是通过有序索引直接扫描返回有序数据，这种方式在通过分析执行计划时，显示为USING INDEX；一种是将取得的数据在内存中进行排序然后再将数据返回给客户端，这种排序在执行计划中显示为USING Filsort，在 MySQL 中 Filsort的实现算法实际上是有两种的，一种是首先根据相应的条件取出相应的排序字段和可以直接定位行数据的行指针信息，然后在 sort buffer 中进行排序。另外一种是一次性取出满足条件行的所有字段，然后在 sort buffer 中进行排序。

根据以上的内容，我们可以得出，对于排序操作的优化目标：尽量减少额外的排序、尽量通过索引直接返回有序数据。尽量让where条件和order by使用相同的索引，并且order by 排序和索引顺序相同，且order by字段都是升序或者都是降序。这样就可以避免出现Filsort。

1. **优化嵌套查询**

不考虑特殊的情况，联表查询要比嵌套查询更有效。

多个子查询能够合并成一个子查询，这样可以把多次表扫描、多次连接减少为单次表扫描和单次连接；exists比in一般更有效率，当子查询速度慢时，可用JOIN来改写一下该查询来进行优化。

1. **优化OR条件**

对于含有OR的条件的查询，如果要使用到索引，则OR的每一个条件都必须使用到索引，可以每个字段分别使用不同的索引，否则优化器不会使用索引查询，MySQL在处理含有OR子句的查询时，实际上是对OR的每个子句进行查询后再将结果进行UNION操作的来的。因此对于符合索引的多个字段，拆分开用OR分隔，则复合索引并不会起作用。

1. **优化分页查询**

LIMIT 子句可以被用于强制 SELECT 语句返回指定的记录数。日常使用中，其格式一般为：limit 当前页码\*页面容量-1 , 页面容量，默认情况下，数据库对于limit操作的处理是，扫描前N页，返回最后的几条需要的记录，很明显查询和排序的代价都很高。

针对此问题，一般的优化思路是，根据索引或者其他查询，先定位目标行，再读取目标数据。所以主要的优化方向在于如何快速定位目标行。

基于索引再排序：在索引上完成排序分页的操作，然后根据主键关联返回原表查询所需要的其他列的内容。也就是说根据索引先定位目标行的主键，然后再根据主键查询其他数据

MySQL自带的BETWEEN…AND… 语句查询优化

将LIMIT转换为某个位置的查询，例如LIMIT 500，10可以转换为，id>500 and id<510

1. **合理使用正则表达式**

正则表达式通常用来检索或者替换哪些符合某个模式的文本内容，它能表达比一般表达式更加丰富的内容，从而减少sql的书写量

**2.3 数据库范式和数据类型**

数据库设计需要满足的规范就是范式，满足这些规范的数据库是简洁的、结构明晰的，同时，不会发生插入、删除和更新操作异常。反之不仅给数据库的编程人员制造麻烦，而且可能存储了大量不需要的冗余信息。但是并不是数据库范式的级别越高越好，范式的级别越高，意味着查询操作所关联的表就会越多，这样也会消耗很多时间，所以数据库设计时要保留适当的冗余，这就是典型的反范式设计。而且数据库设计时，尽量使用到NOT NULL，NULL不仅会占用存储空间，而且还不利于优化。

另外一点比较重要的是，对于不同的数据，要合理选择合适的数据类型。MySQL支持的数据类型如下：

数字类型：尽量不要使用DOUBLE，其存储长度和精确性都不好。同样，固定精度的小数，建议乘以固定倍数转换成整数存储，可以大大节省存储空间，且不会带来任何附加维护成本。对于整数的存储，建议区分开 TINYINT / INT / BIGINT 的选择，三者所占用的存储空间也有很大的差别，能确定不会使用负数的字段，建议添加unsigned定义。

字符类型：尽量不要使用 TEXT 数据类型，其处理方式决定了他的性能要低于char或者是varchar类型的处理。定长字段，建议使用 CHAR 类型，不定长字段尽量使用 VARCHAR，且设定适当的最大长度，而不是非常随意的给一个很大的最大长度限定，因为不同的长度范围，MySQL也会有不一样的存储处理。

时间类型：尽量使用TIMESTAMP类型，其存储空间是 DATETIME 类型的一半。只需要精确到某一天的数据类型，建议使用DATE类型，其存储空间只比TIMESTAMP还少。不要用INT类型类存储一个unix timestamp 的值，不仅不直观，而且会给维护带来不必要的麻烦。

ENUM & SET：对于状态字段，可以尝试使用 ENUM 和SET来存放，可以极大的降低存储空间，而且即使需要增加新的类型，只要增加于末尾，修改结构也不需要重建表数据。

LOB类型：不要在数据库中存放 LOB 类型数据

**2.4 合理使用存储过程和函数**

SQL语句在执行的时候需要先编译，然后执行，而存储过程和函数是一组为了完成特定功能的SQL语句集，经编译后存储在数据库中，省去了编译所需要的时间成本，故存储过程能实现较快的执行速度，另外，存储过程可以减少客户端和服务器之间数据的传输量，从而减少网络流量，且通过限制存储过程的调用权限，也能在一定程度上提高安全性。

1. **数据库索引优化**

**3.1 MySQL索引分类解析**

MySQL中所有的列都可以被索引，对相关列建立索引以后，能够提高查询操作的效率，大大减少了服务器需要扫描的数据量；可以帮助服务器避免排序或减少使用临时表排序；索引可以减少表扫描的IO次数。但是过多的索引也会降低更新表的速度；可能占用较多的磁盘空间。

MySQL中的索引有：主键索引(PRIMARY)、唯一索引(UNIQUE)、普通索引(NORMAL)、全文索引(FULLTEXT)、组合索引，空间索引，前缀索引等。

按索引结构的不同，又可以分为HASH索引、BTREE索引、RTREE索引、FULLTEXT索引。

按索引的顺序和数据的物理存储顺序的关系，又可以分为聚簇索引和非聚簇索引。聚簇索引的顺序就是数据的物理存储顺序，而非聚簇索引的索引顺序与数据物理排列顺序无关。MySQL的存储引擎到目前为止都不支持用户主动创建聚簇索引，唯一的聚簇索引就是InnoDB存储引擎的主键。

不同的索引，性能也各不一样，根据HASH值建立索引，索引效率非常高，索引的检索可以一次定位；BTREE索引就是一种将索引值按一定的算法，存入一个树形的数据结构中；RTREE索引表示空间索引。

另外需要注意的是，在MySQL中，主键和外键约束也是通过索引来实现的，数据库会自动给主键和外键建立索引。

**3.2 如何建立合适的索引**

1. 在经常用作过滤器的字段上建立索引，即出现在where和连接条件中的列
2. 在SQL语句中经常进行GROUP BY、ORDER BY的字段上建立索引
3. 在数据的基数较少的字段上不必要建立索引，如性别字段；尽量在基数较大的字段上建立索引，而且尽量建立唯一索引
4. 对于经常存取的列避免建立索引，索引会降低删除和更新的速度，因为删除和更新数据时，也需要维护索引的结构
5. 复合索引的建立顺序要按照使用的频度来确定，因为复合索引在使用时是按照最左前缀来匹配的
6. 尽量使用数据量少的索引，索引的数据量越多，维护就会越耗时，而较小的索引需要的磁盘IO也会越少，比较起来也会越快；如果目标字段数据量太大，也可以考虑建立前缀索引

**3.2 MySQL索引使用的限制**

**哪些情况下数据库会使用索引：**

1. 对一个有索引的字段使用>、 >=、 =、<、<=、IS NULL、IN、BETWEEN、LIKE(不以通配符开始)、GROUP BY、ORDER BY
2. 复合索引的最左前缀
3. 列前缀可以使用索引
4. where子句中，部分字段精确匹配，其他字段范围匹配

**哪些情况下存在索引但是数据库无法使用：**

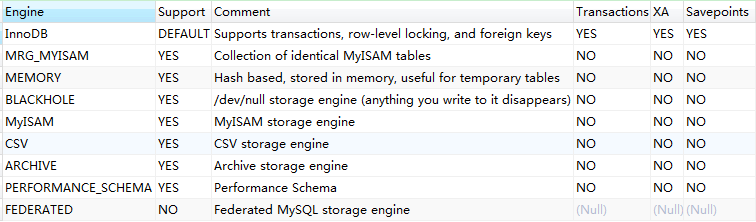
1. 以%开头的LIKE查询不能利用到BTREE索引
2. 数据类型出现隐式转换不会使用索引
3. 复合索引下，查询条件不满足最左原则
4. 数据列中的数据区分度小，查询分析器分析使用索引比全表扫描更慢
5. OR条件中，任意一个条件没有索引
6. Where和ORDER BY的字段使用了不同的索引
7. 请求表上的数据行超出表总记录数30%，变成全表扫描
8. 索引列条件上使用了<>，NOT IN操作符
9. 索引列条件使用函数
10. where子句中，范围匹配后面的其他字段无法使用索引
11. 内存表（HEAP表）使用HASH索引时，使用范围检索或者ORDER BY

......

1. **存储引擎优化**

**4.1 MySQL存储引擎解析**

对于MySQL来说，它提供了很多种类型的存储引擎，我们可以根据对数据处理的需求，选择不同的存储引擎，从而最大限度的利用MySQL强大的功能。经常用到的存储引擎主要有以下几个，下面是对于各种不同的存储引擎的分析。



**4.1.1 MyISAM**

默认的MySQL插件式存储引擎，它是在Web、和其他应用环境下最常使用的存储引擎之一.支持全文索引、压缩、空间函数(GIS)等，但MyISAM不支持事务和行级锁，有一个明显的缺陷就是崩溃后无法安全恢复。

MyISAM表是独立于操作系统的，可以轻松地将其在不同系统平台上移动；每当我们建立一个MyISAM引擎的表时，就会在本地磁盘上建立三个文件，文件名就是表名。

* .frm，存储表定义
* .MYD，存储数据
* .MYI，存储索引

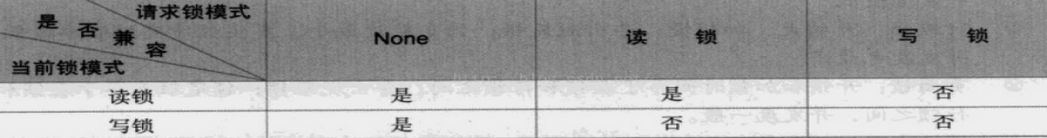
MyISAM表支持三种不同的存储格式：

* 静态表：字段都是非变长字段。每个记录都是固定长度的。从而使存储迅速、容易缓存、出现故障易恢复，但静态表存储数据时会补足空格，所以会占用更多空间。
* 动态表：包含变长字段，记录长度不固定，存储空间占用小，但频繁的更新和删除数据会出现碎片，需定期OPTIMIZE TABLE或myisamchk -r命令来改善性能，并且出现故障恢复相对比较难。
* 压缩表：由myisampack工具创建，磁盘占用很小，每个记录单独压缩，只有非常小的访问开支。

MyISAM表无法处理事务，这就意味着有事务处理需求的表，不能使用MyISAM存储引擎。MyISAM存储引擎特别适合在以下几种情况下使用：

* 选择密集型的表。MyISAM存储引擎在筛选大量数据时非常迅速，这是它最突出的优点。
* 插入密集型、并发相对较低的表。MyISAM的并发插入特性允许同时选择和插入数据。并且有如不需要聚簇索引的维护，插入数据会比较迅速。

MyISAM支持两种表锁机制：表共享读和表独占写

* 共享读锁（S）之间是兼容的，但共享读锁（S）与排他写锁（X）之间，以及排他写锁（X）之间是互斥的，也就是说读和写是串行的。
* 在一定条件下，MyISAM 允许查询和插入并发执行，我们可以利用这一点来解决应用中对同一表查询和插入的锁争用问题。concurrent\_insert，专门用以控制其并发插入的行为，其值分别可以为0、1或2。
* MyISAM 默认的锁调度机制是写优先，用户可以通过设置LOW\_PRIORITY\_UPDATES 参数，或在INSERT、UPDATE、DELETE 语句中指定LOW\_PRIORITY 选项来调节读写锁的争用。
* 由于表锁的锁定粒度大，读写之间又是串行的，因此，如果更新操作较多，MyISAM表可能会出现严重的锁等待。

表级锁更适合于以查询为主，只有少量按索引条件更新数据的应用；

**4.1.2 InnoDB**

MySQL的默认事务型引擎，用于处理大量的短期事务；支持高并发，并且实现了四个标准的隔离级别；基于聚簇索引建立，聚簇索引对主键查询有很高的性能；存储格式是平台独立；支持真正的热备份，MySQL的其他存储引擎不支持热备份；支持行级锁定和外键约束。在InnoDB中，表数据文件本身就是按B+Tree组织的一个索引结构，这棵树的叶节点data域保存了完整的数据记录。这个索引的key是数据表的主键，因此InnoDB表数据文件本身就是主索引。

在 Innodb 存储引擎中，存在两种不同形式的索引，一种是 Cluster 形式的主键索引，另外一种则是和其他存储引擎（如 MyISAM 存储引擎）存放形式基本相同的普通 B-Tree 索引，这种索引在 Innodb 存储引擎中被称为 Secondary Index 。

InnoDB数据库表的本地存储结构如下：

* .frm文件：存储表的框架结构，文件名与表名相同，每个表对应一个frm文件
* .Ibd/ibdata文件：单表表空间文件，每个表使用一个表空间文件（file per table），存放用户数据库表数据和索引。独享表空间存储方式使用.ibd文件，并且每个表一个ibd文件;共享表空间存储方式使用.ibdata文件，所有表共同使用一个ibdata文件

除非有特别的原因，否则应该优先考虑InnoDB引擎，在以下场合下，使用InnoDB是最理想的选择：

* 更新密集的表。InnoDB存储引擎特别适合处理多重并发的更新请求
* 事务。InnoDB存储引擎是支持事务的标准MySQL存储引擎
* 自动灾难恢复。与其它存储引擎不同，InnoDB表能够自动从灾难中恢复
* 外键约束。MySQL支持外键的存储引擎只有InnoDB
* 支持自动增加列AUTO\_INCREMENT属性

InnoDB实现了以下两种类型的行锁，InnoDB行锁是通过给索引项加锁来实现的，即只有通过索引条件检索数据，InnoDB才使用行级锁，否则将使用表锁！

* 共享锁（S）：允许一个事务去读一行，阻止其他事务获得相同数据集的排他锁。
* 排他锁（X)：允许获得排他锁的事务更新数据，阻止其他事务取得相同数据集的共享读锁和排他写锁。

为了允许行锁和表锁共存，实现多粒度锁机制，InnoDB还有两种内部使用的意向锁，这两种意向锁都是表锁。

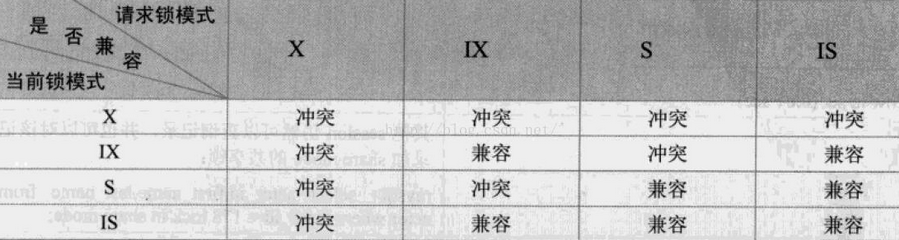
* 意向共享锁（IS）：事务打算给数据行加行共享锁，事务在给一个数据行加共享锁前必须先取得该表的IS锁。
* 意向排他锁（IX）：事务打算给数据行加行排他锁，事务在给一个数据行加排他锁前必须先取得该表的IX锁。

如果一个事务请求的锁模式与当前的锁兼容，InnoDB就将请求的锁授予该事务；反之，如果两者不兼容，该事务就要等待锁释放。

对于UPDATE、DELETE和INSERT语句，InnoDB会自动给涉及数据集加排他锁（X)；对于普通SELECT语句，InnoDB不会加任何锁；

事务可以通过以下语句显示给记录集加共享锁或排他锁。

* 共享锁（S）：SELECT \* FROM table\_name WHERE ... LOCK IN SHARE MODE。
* 排他锁（X)：SELECT \* FROM table\_name WHERE ... FOR UPDATE。



**4.1.3 Memory**

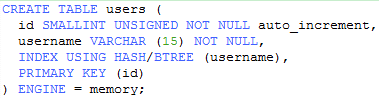
将所有数据保存在RAM中，在需要快速查找引用和其他类似数据的环境下，可提供极快的访问。虽然在内存中存储表数据确实会提供很高的性能，但当mysqld守护进程崩溃时，所有的Memory数据都会丢失。Memory引擎要求存储在Memory数据表里的数据使用的是长度不变的格式，即不能使用BLOB和TEXT这样的长度可变的数据类型，VARCHAR是一种长度可变的类型，但因为它在MySQL内部当做长度固定不变的CHAR类型，所以可以使用。

一般在以下几种情况下使用Memory存储引擎：

* 目标数据较小，而且被非常频繁地访问。在内存中存放数据，所以会造成内存的使用，可以通过参数max\_heap\_table\_size控制Memory表的大小，设置此参数，就可以限制Memory表的最大大小。
* 如果数据是临时的，而且要求必须立即可用，那么就可以存放在内存表中。
* 存储在Memory表中的数据如果突然丢失，不会对应用服务产生实质的负面影响。

Memory同时支持散列索引和B树索引。B树索引优于散列索引的是，可以使用部分查询和通配查询，也可以使用<、>和>=等操作符方便数据挖掘。散列索引进行相等比较非常快，但是对范围比较的速度就慢多了，因此散列索引值适合使用在=和<>的操作符中，不适合在<或>操作符中，也同样不适合用在order by子句中。

可以在表创建时利用USING子句指定要使用的索引的版本，如：



**4.1.4 MERGE**

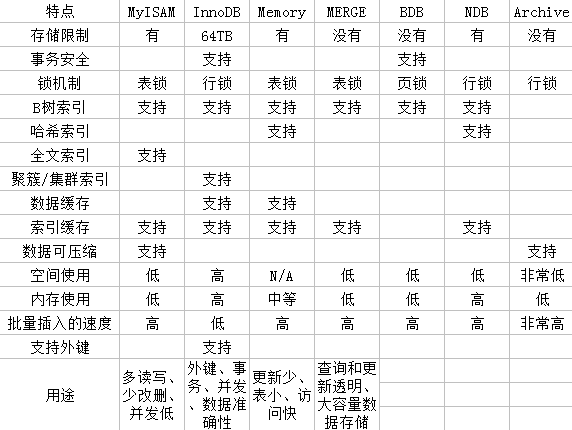
MERGE存储引擎把一组MyISAM数据表当做一个逻辑单元来对待，让我们可以同时对他们进行查询。构成一个MERGE数据表结构的各成员MyISAM数据表必须具有完全一样的结构。每一个成员数据表的数据列必须按照同样的顺序定义同样的名字和类型，索引也必须按照同样的顺序和同样的方式定义。Merge表中并没有数据，对Merge类型的表可以进行查询、更新、删除操作，这些操作实际上是对内部的MyISAM表进行操作。

Merge存储引擎在本地会创建几个文件：.frm文件存储表的定义;.MRG文件包含组合表的信息，包括MERGE表由哪些表组成，插入数据时的依据。

**4.1.5 Archive**

Archive是归档的意思，在归档之后很多的高级功能就不再支持了，仅仅支持最基本的INSERT和SELECT操作，在MySQL5.1之前不支持索引；适合日志和数据采集类应用；支持行级锁和专用的缓存区，所以可以实现高并发的插入，但它不是一个事务型的引擎，而是一个针对高速插入和压缩做了优化的简单引擎。

其它存储引擎用的比较少，下面是所有的存储引擎的对比关系：



**4.2 如何选择存储引擎**

**选择标准：**根据应用特点选择合适的存储引擎，对于复杂的应用系统可以根据实际情况选择多种存储引擎进行组合。选择标准可以分为：

* 是否需要支持事务
* 是否需要使用热备
* 是否需要崩溃恢复：能否接受崩溃
* 是否需要外键支持
* 表是读和插入为主，还是更新为主

经常使用的主要就是InnoDB和MyISAM，MyISAM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，并且文件的大小一般会比InnoDB的小。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。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。

在选择存储引擎时，我们可以根据自己的需求，选择对应的存储引擎即可。