使用索引能加快查询速度，编写 SQL 时就需要注意与索引相关的规则：

* 1. 字段类型转换导致不用索引，如字符串类型的不用引号，数字类型的用引号等，这有可能会用不到索引导致全表扫描；
  2. mysql 不支持函数转换，所以字段前面不能加函数，否则这将用不到索引；
  3. 不要在字段前面加减运算；
  4. 字符串比较长的可以考虑索引一部份减少索引文件大小，提高写入效率；
  5. like % 在前面用不到索引；
  6. 根据联合索引的第二个及以后的字段单独查询用不到索引；
  7. 不要使用 select \*；
  8. 排序请尽量使用升序 ;
  9. or 的查询尽量用 union 代替 （Innodb）；
  10. 复合索引高选择性的字段排在前面；
  11. order by / group by 字段包括在索引当中减少排序，效率会更高

除了上述索引使用规则外，SQL 编写时还需要特别注意一下几点：

* 1. 尽量规避大事务的 SQL，大事务的 SQL 会影响数据库的并发性能及主从同步；
  2. 分页语句 limit 的问题；
  3. 删除表所有记录请用 truncate，不要用 delete；
  4. 不让 mysql 干多余的事情，如计算；
  5. 输写 SQL 带字段，以防止后面表变更带来的问题，性能也是比较优的 ( 涉及到数据字典解析，请自行查询资料)；
  6. 在 Innodb上用 select count(\*)，因为 Innodb 会存储统计信息；
  7. 慎用 Oder by rand()。

2．组合索引

如果where语句中有多个字段，那么可以考虑创建组合索引。

组合索引中字段的顺序是非常重要的，越是唯一的字段越是要靠前。

另外，无论是组合索引还是单个列的索引，尽量不要选择那些唯一性很低的字段。

比如说，在只有两个值0和1的字段上建立索引没有多大意义。

<https://www.cnblogs.com/shanshanlaichi/p/6568097.html>

聚集索引和非聚集索引

类似于在汉语字典中按拼音查字和按偏旁查字。

每个表只能有一个聚集索引，因为目录只能按照一种方法进行排序。

下面的表总结了何时使用聚集索引或非聚集索引（很重要）：

|  |  |  |
| --- | --- | --- |
| 动作描述 | 使用聚集索引 | 使用非聚集索引 |
| 列经常被分组排序 | 应 | 应 |
| 返回某范围内的数据 | 应 | 不应 |
| 一个或极少不同值 | 不应 | 不应 |
| 小数目的不同值 | 应 | 不应 |
| 大数目的不同值 | 不应 | 应 |
| 频繁更新的列 | 不应 | 应 |
| 外键列 | 应 | 应 |
| 主键列 | 应 | 应 |
| 频繁修改索引列 | 不应 | 应 |

一个表应该建立多少索引合适

如果表的80%以上的语句都是读操作，那么索引可以多些，但是不要太多。特别是不要对那些更新频繁的表建立很多的索引。很少表有超过5个以上的索引。过多的索引不但增加其占用的磁盘空间，也增加数据库维护索引的开销。

可考虑将聚集索引用于：

包含大量非重复值的列

使用下列运算符返回一个范围值的查询：between、>、>=、<和<=。

被连续访问的列

返回大型结果集的查询。

经常被使用联接或group by子句的查询访问的列：一般来说，这些都是外键列

聚集索引不适用于：

频繁更改的列

我们可以看到如果仅用聚集索引的起始列作为查询条件和同时用到复合聚集索引的全部列的查询速度是几乎一样的，甚至比用上全部的复合索引列还要略快（在查询结果集数目一样的情况下）；而如果仅用复合聚集索引的非起始列作为查询条件的话，这个索引是不起任何作用的。当然，语句1、2的查询速度一样是因为查询的条目数一样，如果复合索引的所有列都用上，而且查询结果少的话，这样就会形成“索引覆盖”，因而性能可以达到最优。同时，请记住：无论您是否经常使用聚合索引的其他列，但其前导列一定要是使用最频繁的列。

https://blog.csdn.net/wang1127248268/article/details/53413655

Sql语句优化

常见的简化规则如下：

* + 1. 不要有超过5个以上的表连接（join）
    2. 考虑使用临时表或表变量存放中间结果
    3. 少用子查询
    4. 视图嵌套不要过深，一般视图嵌套不要超过2个为宜

连接的表越多，其编译的时间和连接的开销也越大，性能越不好控制。最好是把连接拆开成较小的几个部位逐个顺序执行。优先执行那些能够大量减少结果的连接。

连接是outer join非常不好。因为outer join意味着必须对左表或右表查询所有行。如果表很大而没有相应的where语句，那么outer join很容易导致table scan或index scan。要尽量使用inner join避免scan整个表。

Like语句会因为前置百分号而无法使用索引

限制结果集：要尽量减少返回的结果行，包括行数和字段列数

合理的表设计：未必需要非常范式化。有一定的字段冗余可以增加sql语句的效率，减少join的数目，提高语句的执行速度。

考虑使用存储过程封装那些复杂的sql语句或商业逻辑

索引优化

1. 单字段索引：对出现在where子句中的字段加索引
2. 组合索引：如果where语句中有多个字段，那么可以考虑创建组合索引。组合索引中字段的顺序很重要，越是唯一的字段越是要靠前。另外，无论是组合索引还是单个列的索引，尽量不要选择那些唯一性很低的字段。
3. 覆盖索引：覆盖索引能够使得语句不需要访问表仅仅访问索引就能够得到所有需要的数据。

实际上索引的创建原则是比较复杂的。有时候你无法在索引中包含了where子句中所有的字段。在考虑索引是否应该包含一个字段时，应考虑该字段在语句中的作用。比如说如果经常以某个字段作为where条件作精确匹配返回很少的行，那么就绝对值得为这个字段建立索引。对那些非常唯一的字段如主键和外键，经常出现在group by，order by中的字段等等都值得创建索引。

处理百万级以上的数据提高查询速度的方法：

1. 应尽量避免在where子句中使用!=或<>操作符，否则引擎放弃使用索引而进行全表扫描。
2. 对查询进行优化，应尽量避免全表扫描，首先应考虑在where及order by涉及的列上建立索引。
3. 应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：select id from t where num is null。可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：select id from t where num=0
4. 应尽量避免在where子句中使用or来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：select id from t where num = 10 or num =20

可以这样查询：select id from t where num =10 union all select id from t where num=20

5. 下面的查询也将导致全表扫描：(不能前置百分号) select id from t where name like ‘%abc%’

若要提高效率，可以考虑全文检索。

1. in 和not in也要慎用，否则会导致全表扫描，如：select id from t where num in(1,2,3). 对于连续的数值，能用between就不要用in了: select id from t where num between 1 and 3

select xx,phone FROM send  a JOIN (  
 select '13891030091' phone  union select '13992085916' …………  UNION  SELECT '13619100234' ) b   
  on  a.Phone=b.phone  
--替代下面  很多数据隔开的时候  
in('13891030091','13992085916','13619100234'…………)

1. 如果在where子句中使用参数，也会导致全表扫描。因为sql只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时，它必须在编译时进行选择。然而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。全表扫描：select id from t where num=@num 可以改为强制查询使用索引：select id from t with(index(索引名)) where num=@num
2. .应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：  
        select id from t where num/**2**=**100**  
        应改为:  
        select id from t where num=**100**\***2**
3. 应尽量避免在where子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：  
        select id from t where substring(name,**1**,**3**)=’abc’–name以abc开头的id  
        select id from t where datediff(day,createdate,’**2005**-**11**-**30**′)=**0**–’**2005**-**11**-**30**′生成的id  
        应改为:  
        select id from t where name like ‘abc%’  
        select id from t where createdate>=’**2005**-**11**-**30**′ and createdate<’**2005**-**12**-**1**′
4. 不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。
5. 在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。
6. 很多时候用exists代替in是一个好的选择： select num from a where num in(select num from b) 用下面的语句替换： selelct num from a where exists(select 1 from b where num=a.num)
7. .索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有 必要。

13. 尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会 逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了

14. 尽可能的使用 varchar/nvarchar 代替 char/nchar ，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

15. 任何地方都不要使用 select \* from t ，用具体的字段列表代替“\*”，不要返回用不到的任何字段。

16. 尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限（只有主键索引）。

17. 在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，然后insert。

18. 尽量避免向客户端返回[**大数据**](http://lib.csdn.net/base/spark)量，若数据量过大，应该考虑相应需求是否合理

19. 尽量避免大事务操作，提高系统并发能力。

20. 在IN后面值的列表中，将出现最频繁的值放在最前面，出现得最少的放在最后面，减少判断的次数

创建索引遵循的原则

建议只有当要索引的数据在表中所占的数据量，少于总数据量的15%时，使用索引才会提高查询的速度，否则对表的全表扫描的速度还比使用索引的速度快

 不要将具有很少不同值的列作为索引列。这样的列具有很差的选择性，并且不能优化性能，除非频繁选择的值比其他列值来更少出现。

不要将频繁修改的列作为索引列。因为修改索引列的UPDATE语句和修改索引表的INSERT和DELETE语句将比没有索引要用更多的时间。这样的语句必须修改索引中的数据，还要修改表中的数据。

不要将只出现在带函数或操作符的WHERE子句中的列作为索引列

1.什么情况下应该使用组合索引而非单独索引呢？

假设有条件语句A=a AND B=b，如果A和B是两个单独的索引，在AND条件下只有一个索引起作用，对于B则要逐个判断，而如果使用组合索引(A, B)，只要遍历一棵树就可以了，大大增加了效率。但是对于A=a OR B=b，由于是或的关系，因而组合索引是不起作用的，因而可以使用单独索引，这个时候，两个索引可以同时起作用。

2.如果有组合索引，还需要单列索引吗？

如果组合索引是(A, B)，则对于条件A=a，是可以用上这个组合索引的，因为组合索引是先按照第一列进行排序的，所以没必要对于A单独建立一个索引，但是对于B=b就用不上了，因为只有在第一列相同的情况下，才比较第二列，因而第二列相同的，可以分布在不同的节点上，没办法快速定位。

3.为什么尽量不使用NULL

NULL在B+树里面比较难以处理，往往需要特殊的逻辑进行处理，反而降低了效率。

4.为什么不要在更新频繁的字段上建立索引

更新一个字段意味着相应的索引也要更新，更新往往意味着删除然后再插入，索引本来是一种事先在写的阶段形成一定的数据结构，从而使得在读的阶段效率较高的方式，但是如果一个字段是写多读少，则不建议使用索引。

5.为什么在查询条件里面不要使用函数

例如ID+1=10这种条件，索引是事先写入的时候生成好的，ID+1这种操作在查询阶段，索引无能为例，没办法把所有的索引都先做一个计算，然后再比较吧，代价太大了，因而应该使用ID=10-1。

6.为什么不要使用NOT等负向查询条件

你可以想象一下，对于一棵B+树，跟节点是40，如果你的条件是等于20，就去左面查，你的条件等于50，就去右面查，但是你的条件是不等于66，索引应该咋办？还不是遍历一遍才知道。

7.为什么模糊查询不要以通配符开头

对于一棵B+树来讲，如果根是字符def，如果通配符在后面，例如abc%，则应该搜索左面，例如efg%，则应该搜索右面，如果通配符在前面%abc，则不知道应该走哪一面，还是都扫描一遍吧。

8.为什么OR要改成IN，或者使用Union

OR查询条件的优化往往比较难找到最佳的路径，尤其是OR的条件比较多的时候，尤其如此，对于同一个字段，使用IN就好一些，数据库会对IN里面的条件进行排序，并统一通过二分搜索的方法处理。对于不同的字段，使用Union，则可以让每一个子查询都使用索引。

9.为什么数据类型应该尽量小，常用整型来代替字符型，长字符类型可以考虑使用前缀索引？

因为数据库是按照页存放的，每一页的大小是一样的，如果数据类型比较大，则页数会比较多，每一页放的数据会比较少，树的高度会比较高，因而搜索数据要读取的I/O数目会比较多，插入的时候节点也容易分裂，效率会降低。使用整型来代替字符型多是这个考虑，整型对于索引有更高的效率，例如IP地址等。如果有长字符类型需要使用索引进行查询，为了不要使得索引太大，可以考虑将字段的前缀进行索引，而非整个字段。

六、查询优化的方法论

要找到需要优化的SQL语句，首先要收集有问题的SQL语句。

MySQL 数据库提供了慢SQL日志功能，通过参数slow\_query\_log，获取执行时间超过一定阈值的SQL语录列表。

没有使用索引的SQL语句，可以通过long\_queries\_not\_using\_indexes参数开启。

min\_examined\_row\_limit，扫描记录数大于该值的SQL语句才会被记入慢SQL日志。

找到有问题的语句，接下来就是通过explainSQL，获取SQL的执行计划，是否通过索引扫描记录，可以通过创建索引来优化执行效率。是否扫描记录数过多。是否持锁时间过长，是否存在锁冲突。返回的记录数是否较多。

接下来可以定制化的优化。没有被索引覆盖的过滤条件涉及的字段，在区分度较大的字段上创建索引，如果涉及多个字段，尽量创建联合索引。

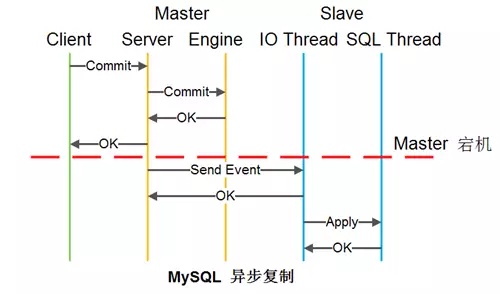
扫描记录数非常多，返回记录数不多，区分度较差，重新评估SQL语句涉及的字段，选择区分度高的多个字段创建索引。

扫描记录数非常多，返回记录数也非常多，过滤条件不强，增加SQL过滤条件schema\_redundant\_indexes查看有哪些冗余索引。

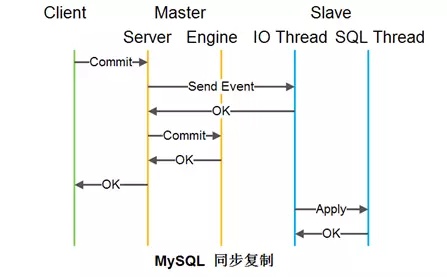
如果多个索引涉及字段顺序一致，则可以组成一个联合索引schema\_unused\_indexes查看哪些索引从没有被使用。

**七、读写分离的原理**

主从复制基于主节点上的服务层的日志实现的，而从节点上有一个IO线程读取这个日志，然后写入本地。另有一个线程从本地日志读取后在从节点重新执行。



如图是主从异步复制的流程图。在主实例写入引擎后就返回成功，然后将事件发给从实例，在从实例上执行。这种同步方式速度较快，但是在主挂了的时候，如果还没有复制，则可能存在数据丢失问题



数据库同步复制也不同，是当从节点落盘后再返回客户端，当然这样会使得性能有所降低，网易数据库团队是通过组提交，并行复制等技术将性能提上来。

有了主从复制，在数据库DAO层可以设置读写分离策略，也有通过数据库中间件做这个事情的。