# 一、索引的基本原理

索引技术的初衷是为了快速从一个大数据集中找出某个字段等于确定值（比如按身份证号找出某个人）的记录。一个规模（行数）为N的数据集，用遍历查找则需要比较N次，而如果数据是按该字段值（在索引中称为键值）有序的，那么就可以建立二叉树用二分法查找，只要比较logN（以2为底）次，比如10亿行数据只要比较30次（10亿约是230），这显然能大大提高性能。有时可能还会有键值有重复的情况（按出生日期找人）或按键值区间的查找需求（按出生日期区间找人），比较次数就会比logN大一些，但基本仍是这个数量级的。

索引的本质就是排序。当然，我们一般不会把原始数据集排序，而是把每条记录的键值和这条记录在数据集中的位置，按键值次序做成一个规模较小的数据集，这也就是索引表了。如果还有其它字段也要用于键值查找，则可以再建立别的索引。原始数据集只有一份，索引可以有多个，如果每个索引都把原始数据集排序，则会使数据集被复制很多遍，占用空间过大。

另外，数据库在建立索引时还要考虑数据会插入删除，简单排序的索引会导致插入删除的成本非常高，这时一般会使用B树以方便快速更新。B树相当于把二叉树扩展成n叉树，本质上仍然是键值有序。

还有一种引申出来的方法是HASH索引，计算记录键值的某种HASH值，散列到1…k的自然数范围。这样查找时连二分比较也不必做，直接用HASH值定位了。HASH方法只用来做键值的精确查找，不能用来实现区间查找，因为HASH函数并不单调，已经失去原来键值的大小信息了，不过这在许多场景下也够用（按身份证号找人）。HASH索引本质上也是排序，只是用了键值的HASH值来排序。我们下面的讨论还是以普通键值排序为例，结论也适用于HASH索引。

从原理上看，显然索引不会提高全量数据遍历的运算性能。

# 二、单索引

理解了上述原理后，我们就能知道什么时候索引会有效，以及书写语法时的注意事项。

## 1、只针对键值本身提条件的，很有效。

如：身份证号等于某值的、出生日期在某个区间内的，这些都很有效。

## 2、针对键值的函数提条件的，大部分无效，小部分取决于数据库优化

如：出生日期是星期几的，索引键是出生日期。索引就没法用，因为星期几对索引无序，这时要把索引直接建在键值函数上，大部分数据库都支持这种索引。

再如：年龄在某个区间的，索引键是出生日期。索引不能直接用，但年龄和出生日期之间是个单调函数，如果数据库优化做得好是可能利用的。但大概率是不行的。

书写查询条件时要尽量写成针对原始索引键值本身，不要使用函数或表达式。

## 3、一般性条件中包含键值条件的，键值条件作为一个最外层的and条件时有效

如：出生日期在某天且姓名中有某字的。数据库会用索引找出出生日期在某天的、然后再在其中遍历查找出姓名中有某字的。现代商用数据库都能够智能地分析条件表达式而找到可以使用索引提速的部分。

再如：出生日期在某天或姓名中有某字的。这时候索引就没法用了，后半部条件反正也只能遍历，那就直接遍历了，索引就忽略了。

书写多个组合查询条件时就要注意尽量把索引键有关的条件放在最外层和其它条件and起来，索引键不能用于缩小查询范围时不会提高性能。

# 三、多索引

如果我们为数据集查询条件中涉及的多个字段都建立索引，是否会进一步提高性能？

从上面的原理分析后结论比较悲催，大部分场景是只能用上一个。

比如在字段A和B上都建有索引，查询条件是 A=1 and B=2。先用索引A过滤出来的A=1的记录，对B并没有序，这时B=2的条件只能硬遍历；反过来也一样，先用B=2过滤的结果集对A无序，也只能遍历了。商用数据库一般会预估成本，选择A和B中的过滤后结果集较小的那个索引来用。

不过，如果是A=1 or B=2反而有可能用上，优化能力较好的数据库会分别用索引过滤出A=1和B=2的记录，再做个并集。

还可以建立多字段索引，如果建立A,B双字段索引，那么用A=1过滤后的结果集就对B有序，就可以继续用该索引过滤B=2的条件。数据库优化较好时会知道A=1 and B=2和B=2 and A=1是一回事，条件书写次序可以不必刻意和索引次序一致，只要注意上一小节所说的情况3就行：尽量把索引涉及条件放在最外层。

但是，A,B双字段索引对单独的B=2这个条件并无效，因为对A,B有序未必对B有序。B=2这种条件普只能再遍历了，这是许多程序员容易犯的错误。完整些说，A,B,C这样的多字段索引，对于A=?，A=? and B=?，A=? and B=? and C=?这类条件都有效，但对于B=?，C=?，B=? and C=?这种条件是无效的，还需要重新建立关于B或C的索引。

出于这个考虑，建立一次A,B,C多字段索引会对A,A/B,A/B/C条件都有效，那我们是否应当尽量把索引字段搞得尽量多？从索引原理上似乎是这样，但这样会导致索引表也大一圈，增加IO成本，所以也不一定，需要适当的权衡。

如果我们按上述原则正确地建立和使用了索引，是否就一定能提高性能呢？

还是不一定！

索引的初衷是用键值取数，大多数情况是从一个巨大的数据集中会取出很少的记录出来。这类场景下，如果按上述原则建立和使用索引，确实是能显著地提高性能。但有时候条件遍历取出的记录非常多，这就很难说是不是能提高性能了，甚至可能反而更差。

原因是这样的：

我们前述说过，建索引时一般不会直接把原始数据集排序，而是另建一个索引表。按索引表的次序取出的数据，对于原始数据集而言并不是连续存放的，数据库优化做得不好时甚至可能是乱序的。硬盘取出大量不连续存放的数据时会同时取出很多无关数据，其耗时不能简单地按取出数据量来计算，这时候使用索引取数的性能提升就不会象希望的那样明显。如果乱序时还强行使用索引则还可能导致重复取，对于机械硬盘再有大量的磁头跳动时间，结果集很大时就极有可能还不如硬遍历的性能好。不过一般商用数据库会预估成本后选择合适的执行计划，发现有可能是这些情况就不再使用索引了，所以看到的表现一般最差也就是和遍历一样了，但如果预估不准，执行计划搞错了就可能出现还不如遍历性能好的现象。

数据库中数据一般是按插入次序存放的，如果这个次序和索引键序基本一致，那么会保证取出数据在物理上存放时是相对连续的，这时候再使用索引过滤，即使取出数据量较大也经常能观察到比较明显的性能提升。

# 四、SQL优化

1、对查询进行优化，要尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

2、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;

最好不要给数据库留null，尽可能的使用 not null填充数据库.

备注、描述、评论之类的可以设置为 null，其他的，最好不要使用null。

不要以为 null 不需要空间，比如：char(100) 型，在字段建立时，空间就固定了， 不管是否插入值（null也包含在内），都是占用 100个字符的空间的，如果是varchar2这样的变长字段， null 不占用空间。

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;

3、应尽量避免在 where 子句中使用 != 或 <> 操作符，否则将引擎放弃使用索引而进行全表扫描。

4、应尽量避免在 where 子句中使用 or 来连接条件，如果一个字段有索引，一个字段没有索引，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 name='admin';

可以这样查询：

select id from t where num=10;

union all

select id from t where name='admin';

5、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;

 6、下面的查询也将导致全表扫描：

select id from t where name like '%abc%';

若要提高效率，可以考虑全文检索。

7、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;

8、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 substr(name,1,3)='abc';

select id from t where to\_char(crttime, 'yyyy-mm-dd')>'2005-11-30';

应改为:

select id from t where name like 'abc%';

select id from t where crttime>to\_date('2005-11-30', 'yyyy-mm-dd');

9、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。

10、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。

11、update 语句，如果只更改1、2个字段，不要update全部字段，否则频繁调用会引起明显的性能消耗，同时带来大量日志。

12、对于多张大数据量（这里几百条就算大了）的表join，要先分页再join，否则逻辑读会很高，性能很差。

13、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。

14、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。

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。