**一、设计最优的库表结构**

**二、建立最好的索引**

索引是一种数据结构，包含索引键值和指向数据记录物理地址的指针。

**二叉查找树**

二叉查找树的特点：

是一棵二叉树，每个节点都包含一个可以比较的键。

每个节点的键都大于左子节点任意节点的键，且大于右子节点任意节点键。

每个节点都有连个链接，左连接和右连接。链接也可以为null

查找实现伪代码：

1.如果二叉查找树为null，查找失败，返回null

2.如果根节点的键等于要查找的键，返回根节点的值

3.否则，继续在相应的子节点中查找。如果要查找的键小于根节点的键，在左子树中进行查找；如果要查找的键大于根节点的键，在右子树中进行查找。

4.重复以上步骤，直至满足结束条件。

**平衡二叉树**

空树或者任一结点左、右子树高度差的绝对值不超过1

红黑树

B树

B+ 树的结构

各种索引的应用场景，什么样的场景是用什么样的索引比较合适？

**B树索引：**

**MySQL大多数存储引擎都支持B数索引。**

B-tree索引适用于全键值匹配、键值范围或键前缀（最左前缀）查找

全值匹配：

和索引中所有列进行匹配

匹配最左前缀：

只使用索引的第一列

匹配列前缀：

只匹配某一列的值的开头部分

匹配范围值：

索引某一列范围查找

精确匹配某一列，范围匹配另外一列：

只访问索引的查询：

可以用作order by和group by的操作

**B-Tree索引的限制：**

1.如果不是按照最左列开始查找，不能使用索引。

查询的时候不使用索引第一列，而是直接通过第二列或者第三列进行匹配查找是不行的

2.不能跳过索引中的列

查询指定了索引的第一列，后直接使用第三列。MySQL只能使用索引的第一列

3.查询中某个列的范围查询，则右边所有列都无法使用索引的优化查询。

总结：这些限制都和索引列的顺序有关。

**哈希索引**

哈希索引是基于哈希表实现，只有精确匹配索隐列的查询才有效。MySQL中只有**Memory存储引擎显示支持哈希索引。**

哈希索引的限制：

1.

2.

3.

4.

5.

6.

自定义hash索引。触发器语句和delimiter

如何避免哈希冲突？

MySQL中的哈希函数有哪些？

CRC32();SHA1();MD5();

**空间数据索引（R-Tree）**

和B-Tree索引的区别：无须前缀查询，会从所有维度来索引数据。

**全文索引**

**高性能索引策略**

“独立列”：索引列不能是表达式的一部分，也不能是函数的参数

“索引的选择性”：不重复的索引值和数据表记录数的比值。这个数越大越好

“基数经验法则”

前缀索引：

Alter table sakila.city\_demo add key (city(7));

多列索引：

索引列顺序：

当不需要考虑排序和分组时候，将选择性更高的列放在最前面通常是很好的。这个时候索引的作用只是用于优化where条件查找。

聚簇索引

创建索引的语法

**三、合理的设计查询**

MySQL查询中的子任务

1.首先考虑在where和group by的字段上建立索引

2.避免在where字段进行null值的判断，索引不会对null建立索引。

会进行全表的扫描

3.where字句中尽量不要使用!=和<>，否则引擎会放弃使用索引进行全表扫描

在where字句中避免使用or，应该使用union代替

4.尽量不要使用In和not in，否则会导致全表的扫描

可以的话使用between代替

使用exits 代替in

5.like '%acd%'会使用全表扫描

like 'acd%'会使用索引

6.where 子句中使用参数，也会导致全表扫描

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

7.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。

select id from t where num/2 = 100;

应改为:

select id from t where num = 100\*2;

9.避免在where子句中对字段进行函数操作

select id from t where substring(name,1,3) = ’abc’

select id from t where datediff(day,createdate,’2005-11-30′) = 0

改成

select id from t where name like 'abc%'

select id from t where createdate >= '2005-11-30' and createdate < '2005-12-1'

10.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。

11.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。

13.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。

14.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[[1]](#footnote-1)，否则逻辑读会很高，性能很差。

15.select count(\*) from table；这样不带任何条件的count会引起全表扫描，并且没有任何业务意义，是一定要杜绝的

16. 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。

17.应尽可能的避免更新 clustered 索引数据列

18.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。

19.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。

21.尽量使用表变量[[2]](#footnote-2)来代替临时表。

25.尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。

26.使用基于游标的方法或临时表方法之前，应先寻找基于集[[3]](#footnote-3)的解决方案来解决问题，基于集的方法通常更有效。

29.尽量避免大事务操作，提高系统并发能力。

**以上这些问题从另一个角度进行再次总结如下：**

1.什么情况下设置了索引但无法使用，索引无效

2.避免在WHERE子句中使用in，not in，or 或者having。

3.查询的模糊匹配

1. 如何实现？ [↑](#footnote-ref-1)
2. 表变量和临时表区别？ [↑](#footnote-ref-2)
3. 集和游标的区别？ [↑](#footnote-ref-3)