



链滴

# 索引底层

作者: [fjiang](#)

原文链接: <https://ld246.com/article/1620636854018>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

### 索引底层数据结构

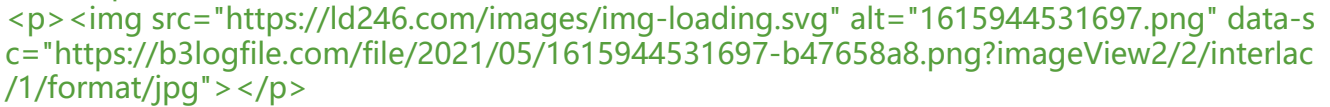
#### 1.为什么底层不用hash查找?

因为 hash 值是无序的,虽然单个查询很快,但是数据库中大部分查询都是范围查询,此时 hash 查找效率很低

#### 2.为什么不用 B 树?

\*\* B-树,这里的 B 表示 balance( 平衡的意思),B-树是一种多路自平衡的搜索树 ( B 树是一颗多路平衡查找树)

\*\*\*\*它类似普通的平衡二叉树,不同的一点是 B-树允许每个节点有更多的子节点。下图是 B-树的简图.



B 树有如下特点:

所有键值分布在整颗树中 (索引值和具体 data 都在每个节点里) ;

任何一个关键字出现且只出现在一个结点中;

搜索有可能在非叶子结点结束 (最好情况  $O(1)$ 就能找到数据) ;

在关键字全集内做一次查找,性能逼近二分查找;

**B 树由来**

\*\* 传统的 AVL 树,红黑树查找效率就已经很高了,但是为什么还要引进 B 树呢? 这因为当数据量很大的时候,无法将所有数据都加载到内存,必然要将数据存放到磁盘中,只要到要用的时候才加载到内存中,一般而言内存访问的时间约为 50 ns,而磁盘在 10 ms 左右。速度相差了近 5 个量级,磁盘读取时间远远超过了数据在内存中比较的时间。这说明程序大部分时间会阻塞在磁盘 IO。如何提高性能?减少 IO 次数,像红黑树,AVL 树从设计上无法迎合磁盘

**B 树如何迎合磁盘**

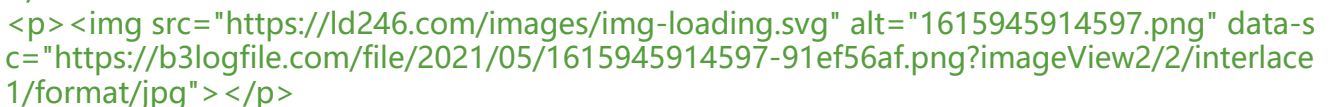
索引的效率依赖与磁盘 IO 的次数,快速索引需要有效的减少磁盘 IO 次数,如何快速引呢? 索引的原理其实是不断的缩小查找范围,就如我们平时用字典查单词一样,先找首字母缩小范围,再第二个字母等等。平衡二叉树是每次将范围分割为两个区间。为了更快, B-树每次将范围分割为多个区间,区间越多,定位数据越快越精确。那么如果节点为区间范围,每个节点就较大了。所以新建点时,直接申请页大小的空间 (磁盘存储单位是按 block 分的,一般为 512 Byte。磁盘 IO 一次读取若干个 block,我们称为一页,具体大小和操作系统有关,一般为 4 k, 8 k 或 16 k), 计算机内存分是按页对齐的,这样就实现了一个节点只需要一次 IO。

#### 3.用 B+ 树有什么好处?

B+ 树是 B 树的变体,也是一种多路搜索树,它与 B 树的不同之处在于:

所有关键字存储在叶子节点出现,内部节点(非叶子节点并不存储真正的 data)

为所有叶子节点增加了一个链指针



因为 B+ 树的内节点不存储数据,数据都在叶子节点上,导致 B+ 树的查询速度固定为  $O(\log n)$ ,而 B 树则不一定,与 key 值所在的位置有关,最好为  $O(1)$

**优点:**

B+ 树叶节点两两相连可以大大增加区间访问性,特别适合范围查询,而 B 树 key 和 value 在一起,无法进行区间查找

**空间局部性原理:**

如果一个存储器的

个位置被访问,那么它附近的位置也会被访问

```
</span> </span> </code> </pre>
```

**B+ 树就很好地利用了空间局部性原理,若我们访问节点 key 为 50,则 key 为 45,55,60 将来也能会被访问,我们可以利用磁盘预读原理提前将这些数据读入内存,减少了磁盘 IO 次数**

**<ol start="2">**

**<li> B+ 树更适合外部存储。由于内节点无 data 值,每个节点能索引的范围更大更准确</li>**

**<li> 由于 B 树节点内部每个 key 都带着 data 域,而 B+ 树节点只存储 key 的副本,真实的 key 和 data 域都在叶子节点存储。前面说过磁盘是分 block 的,一次磁盘 IO 会读取若干个 block,具和操作系统有关,那么由于磁盘 IO 数据大小是固定的,在一次 IO 中,单个元素越小,量就越大。就意味着 B+ 树单次磁盘 IO 的信息量大于 B-树,从这点来看 B+ 树相对 B-树磁盘 IO 次数少。</li>**

**</ol>**

### **索引失效底层原理**

**<ol>**

**<li> 为什么不遵循最左前缀原则索引会失效?</li>**

**</ol>**

**<p>  </p>**

**\*\* 一般索引失效针对联合索引,上图可知,得到的结果是有序的,但实际上联合索引针对左边进行索引排序</p>**

**\*\* a 顺序:1 1 2 2 3 3</p>**

**\*\* b 无序:1 2 1 4 1 2</p>**

**\*\* 如果不遵循最左前缀法则,相当于 a 失效,b 又是无序的,那么就无法通过二分查来获取数据,效率就变得很低</p>**

**<ol start="2">**

**<li>**

**<p> like 失效(只能是单值索引)</p>**

**\*\*比如查询 \*\*</p>**

```
<pre> <code class="highlight-chroma"> <span class="highlight-line"> <span class="highlight-cl">select * from student where sno like "1%"//查询1开头的学号
```

```
</span> </span> <span class="highlight-line"> <span class="highlight-cl">select * from stud nt where sno like "%1"//查询1结尾的学号
```

```
</span> </span> </code> </pre>
```

**<p> 所以进行模糊查询时,% 只能为前缀(% 放在右边),但实在需要中缀,后缀情况时,就要使覆盖索引</p>**

**</li>**

**</ol>**