06 | 数据库检索：如何使用B+树对海量磁盘数据建立索引?

2020-04-08 陈东

检索技术核心20讲

进入课程 >

讲述：陈东

时长 19:01　大小 17.43M

你好，我是陈东。

在基础篇中，我们学习了许多和检索相关的数据结构和技术。但是在大规模的数据环境下，这些技术的应用往往会遇到一些问题，比如说，无法将数据全部加载进内存。再比如说，无法支持索引的高效实时更新。而且，对于复杂的系统和业务场景，我们往往需要对基础的检索技术进行组合和升级。这就需要我们对实际的业务问题和解决方案十分了解。

所以，从这一讲开始，我会和你一起探讨实际工作中的系统和业务问题，分享给你一些业界中常见的解决方案，帮助你积累对应的行业经验，让你能够解决工作中的检索难题。

在工业界中，我们经常会遇到的一个问题，许多系统要处理的数据量非常庞大，数据无法全部存储在内存中，需要借助磁盘完成存储和检索。我们熟悉的关系型数据库，比如 MySQL 和 Oracle，就是这样的典型系统。

数据库中支持多种索引方式，比如，哈希索引、全文索引和 B+ 树索引，其中 B+ 树索引是使用最频繁的类型。因此，今天我们就一起来聊一聊磁盘上的数据检索有什么特点，以及为什么 B+ 树能对磁盘上的大规模数据进行高效索引。

磁盘和内存中数据的读写效率有什么不同?

首先，我们来探讨一下，存储在内存中和磁盘中的数据，在检索效率方面有什么不同。

内存是半导体元件。对于内存而言，只要给出了内存地址，我们就可以直接访问该地址取出数据。这个过程具有高效的随机访问特性，因此内存也叫**随机访问存储器**（Random Access Memory，即 RAM）。内存的访问速度很快，但是价格相对较昂贵，因此一般的计算机内存空间都相对较小。

而磁盘是机械器件。磁盘访问数据时，需要等磁盘盘片旋转到磁头下，才能读取相应的数据。尽管磁盘的旋转速度很快，但是和内存的随机访问相比，性能差距非常大。到底有多大呢？一般来说，如果是随机读写，会有 10 万到 100 万倍左右的差距。但如果是顺序访问大批量数据的话，磁盘的性能和内存就是一个数量级的。为什么会这样呢？这和磁盘的读写原理有关。那具体是怎么回事呢？

磁盘的最小读写单位是扇区，较早期的磁盘一个扇区是 512 字节。随着磁盘技术的发展，目前常见的磁盘扇区是 4K 个字节。操作系统一次会读写多个扇区，所以操作系统的最小读写单位是**块**（Block），也叫作**簇**（Cluster）。当我们要从磁盘中读取一个数据时，操作系统会一次性将整个块都读出来。因此，对于大批量的顺序读写来说，磁盘的效率会比随机读写高许多。

现在你已经了解磁盘的特点了，那我们就可以来看一下，如果使用之前学过的检索技术来检索磁盘中的数据，检索效率会是怎样的呢？

假设有一个有序数组存储在硬盘中，如果它足够大，那么它会存储在多个块中。当我们要对这个数组使用二分查找时，需要先找到中间元素所在的块，将这个块从磁盘中读到内存里，然后在内存中进行二分查找。如果下一步要读的元素在其他块中，则需要再将相应块从磁盘

中读入内存。直到查询结束，这个过程可能会多次访问磁盘。我们可以看到，这样的检索性能非常低。

由于磁盘相对于内存而言访问速度实在太慢，因此，对于磁盘上数据的高效检索，我们有一个极其重要的原则：**对磁盘的访问次数要尽可能的少**！

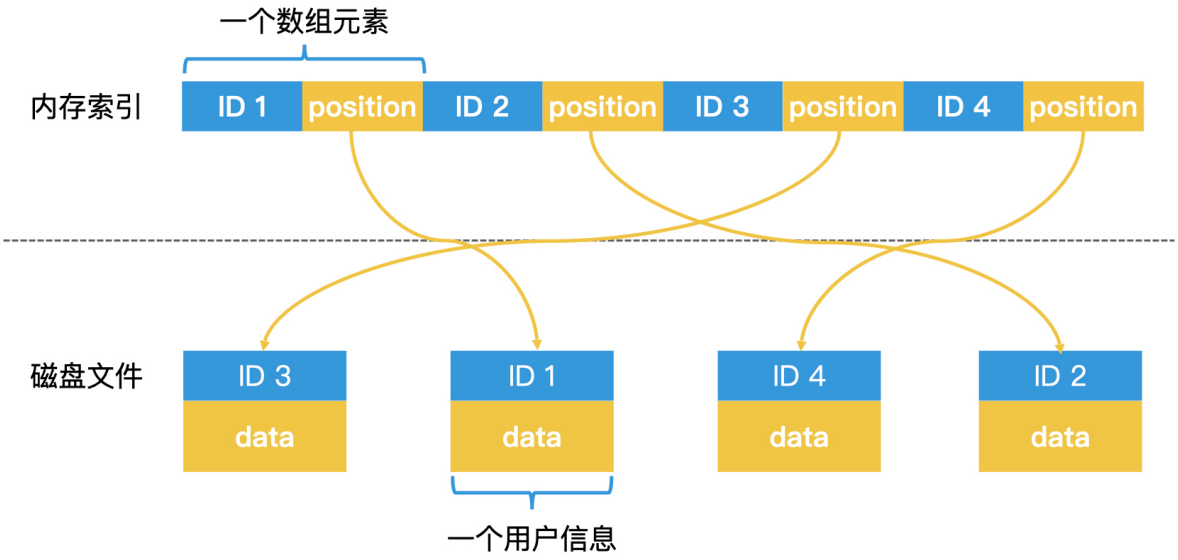
那问题来了，我们应该如何减少磁盘的访问次数呢？将索引和数据分离就是一种常见的设计思路。

如何将索引和数据分离？

我们以查询用户信息为例。我们知道，一个系统中的用户信息非常多，除了有唯一标识的 ID 以外，还有名字、邮箱、手机、兴趣爱好以及文章列表等各种信息。一个保存了所有用户信息的数组往往非常大，无法全部放在内存中，因此我们会将它存储在磁盘中。

常规数组存储

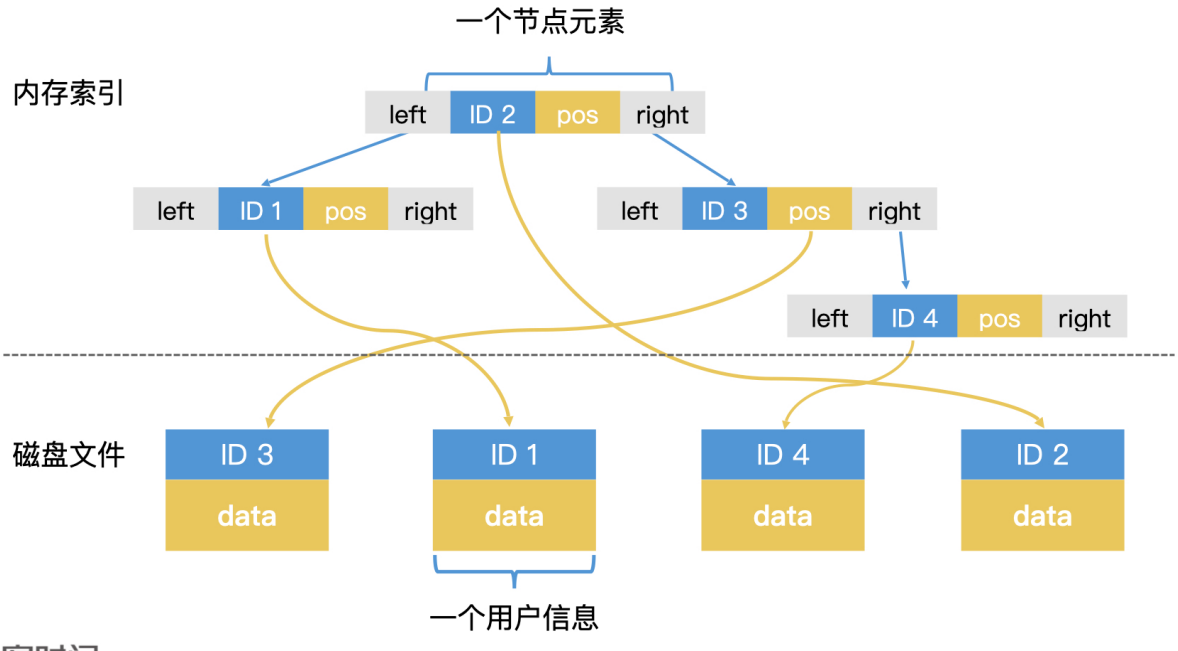
当我们以用户的 ID 进行检索时，这个检索过程其实并不需要读取存储用户的具体信息。因此，我们可以生成一个只用于检索的有序索引数组。数组中的每个元素存两个值，一个是用户 ID，另一个是这个用户信息在磁盘上的位置，那么这个数组的空间就会很小，也就可以放入内存中了。这种用有序数组做索引的方法，叫作**线性索引**（Linear Index）。



极客时间

索引与数据分离：线性索引

在数据频繁变化的场景中，有序数组并不是一个最好的选择，二叉检索树或者哈希表往往更有普适性。但是，哈希表由于缺乏范围检索的能力，在一些场合也不适用。因此，二叉检索树这种树形结构是许多常见检索系统的实施方案。那么，上图中的线性索引结构，就变成下图这个样子。



极客时间

索引与数据分离：树形索引

尽管二叉检索树可以解决数据动态修改的问题，但在索引数据很大的情况下，依然会有数据无法完全加载到内存中。这种情况我们应该怎么办呢？

一个很自然的思路，就是将索引数据也存在磁盘中。那如果是树形索引，我们应该将哪些节点存入磁盘，又要如何从磁盘中读出这些数据进行检索呢？你可以先想一想，然后我们一起来看看业界常用的解决方案 B+ 树是怎么做的。

如何理解 B+ 树的数据结构？

B+ 树是检索技术中非常重要的一个部分。这是为什么呢？因为 **B+ 树给出了将树形索引的所有节点都存在磁盘上的高效检索方案**，使得索引技术摆脱了内存空间的限制，得到了广泛的应用。

前面我们讲了，操作系统对磁盘数据的访问是以块为单位的。因此，如果我们想将树型索引的一个节点从磁盘中读出，即使该节点的数据量很小（比如说只有几个字节），但磁盘依然会将整个块的数据全部读出来，而不是只读这一小部分数据，这会让有效读取效率很低。**B+ 树的一个关键设计，就是让一个节点的大小等于一个块的大小。节点内存储的数据，不是一个元素，而是一个可以装 m 个元素的有序数组**。这样一来，我们就可以将磁盘一次读取的数据全部利用起来，使得读取效率最大化。

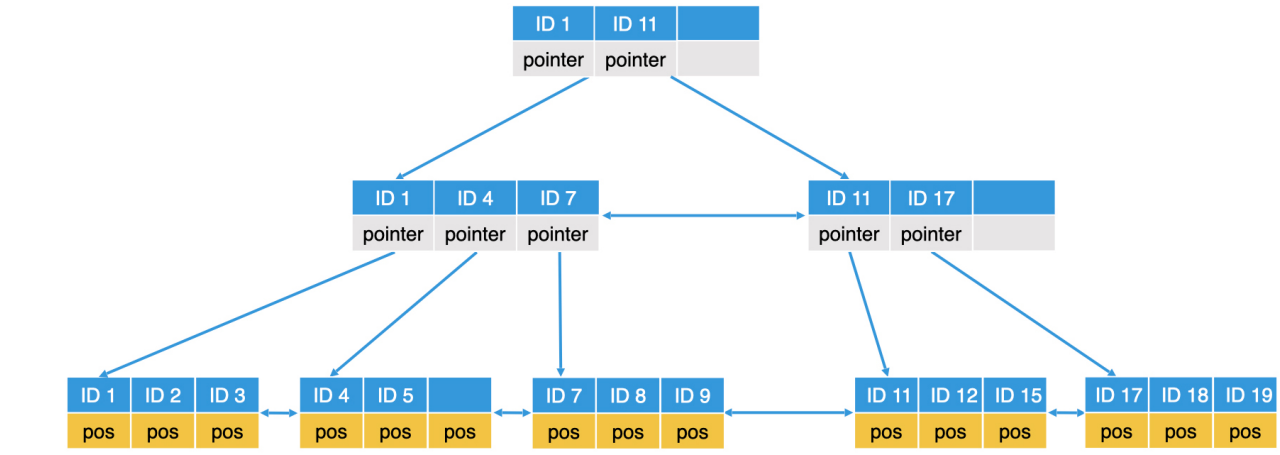
B+ 树还有另一个设计，就是将所有的节点分为内部节点和叶子节点。尽管内部节点和叶子节点的数据结构是一样的，但存储的内容是不同的。

内部节点仅存储 key 和维持树形结构的指针，并不存储 key 对应的数据（无论是具体数据还是文件位置信息）。这样内部节点就能存储更多的索引数据，我们也就可以使用最少的内部节点，将所有数据组织起来了。而叶子节点仅存储 key 和对应数据，不存储维持树形结构的指针。通过这样的设计，B+ 树就能做到节点的空间利用率最大化。

B+ 树的内部节点和叶子节点

此外，B+ 树还将同一层的所有节点串成了有序的双向链表，这样一来，B+ 树就同时具备了良好的范围查询能力和灵活调整的能力了。

因此，B+ 树是一棵完全平衡的 m 阶多叉树。所谓的 m 阶，指的是每个节点**最多**有 m 个子节点，并且每个节点里都存了一个紧凑的可包含 m 个元素的数组。



B+ 树的整体结构

B+ 树是如何检索的?

这样的结构，使得 B+ 树可以作为一个完整的文件全部存储在磁盘中。当从根节点开始查询时，通过一次磁盘访问，我们就能将文件中的根节点这个数据块读出，然后在根节点的有序数组中进行二分查找。

具体的查找过程是这样的：我们先确认要寻找的查询值，位于数组中哪两个相邻元素中间，然后我们将第一个元素对应的指针读出，获得下一个 block 的位置。读出下一个 block 的节点数据后，我们再对它进行同样处理。这样，B+ 树会逐层访问内部节点，直到读出叶子节点。对于叶子节点中的数组，直接使用二分查找算法，我们就可以判断查找的元素是否存在。如果存在，我们就可以得到该查询值对应的存储数据。如果这个数据是详细信息的位置指针，那我们还需要再访问磁盘一次，将详细信息读出。

我们前面说了，B+ 树是一棵完全平衡的 m 阶多叉树。所以，B+ 树的一个节点就能存储一个包含 m 个元素的数组，这样的话，一个只有 2 到 4 层的 B+ 树，就能索引数量级非常大的数据了，因此 B+ 树的层数往往很矮。比如说，一个 4K 的节点的内部可以存储 400 个元素，那么一个 4 层的 B+ 树最多能存储 400^4，也就是 256 亿个元素。

不过，因为 B+ 树只有 4 层，这就意味着我们最多只需要读取 4 次磁盘就能到达叶子节点。并且，我们还可以通过将上面几层的内部节点全部读入内存的方式，来降低磁盘读取的次数。

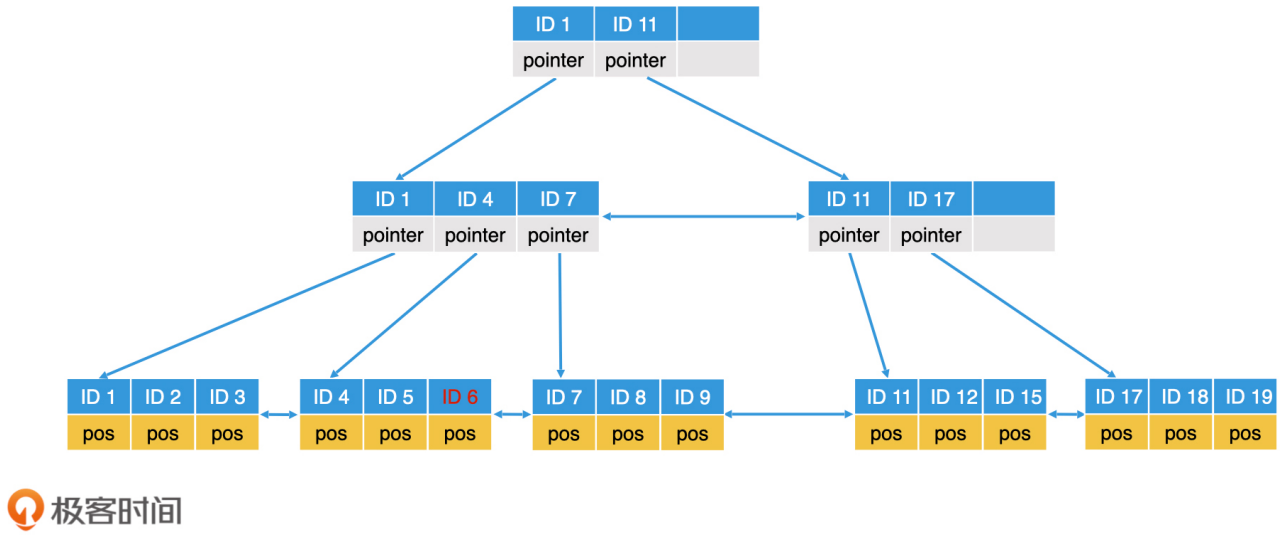
比如说，对于一个 4 层的 B+ 树，每个节点大小为 4K，那么第一层根节点就是 4K，第二层最多有 400 个节点，一共就是 1.6M；第三层最多有 400^2，也就是 160000 个节点，一共就是 640M。对于现在常见的计算机来说，前三层的内部节点其实都可以存储在内存中，只有第四层的叶子节点才需要存储在磁盘中。这样一来，我们就只需要读取一次磁盘即可。这也是为什么，B+ 树要将内部节点和叶子节点区分开的原因。通过这种只让内部节点存储索引数据的设计，我们就能更容易地把内部节点全部加载到内存中了。

B+ 树是如何动态调整的?

现在，你已经知道 B+ 树的结构和原理了。那 B+ 树在“新增节点”和“删除节点”这样的动态变化场景中，又是怎么操作的呢？接下来，让我们一起来看一下。

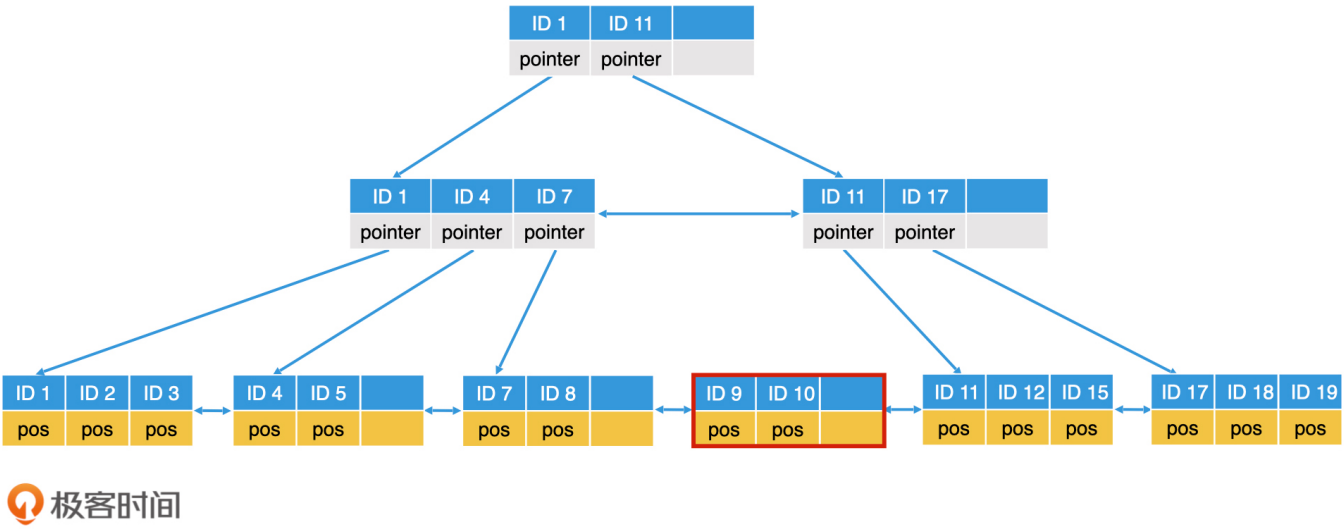
首先，我们来看插入数据。由于具体的数据都是存储在叶子节点上的，因此，数据的插入也是从叶子节点开始的。以一个节点有 3 个元素的 B+ 树为例，如果我们要插入一个 ID=6

的节点，首先要查询到对应的叶子节点。如果叶子节点的数组未满，那么直接将该元素插入数组即可。具体过程如下图所示：



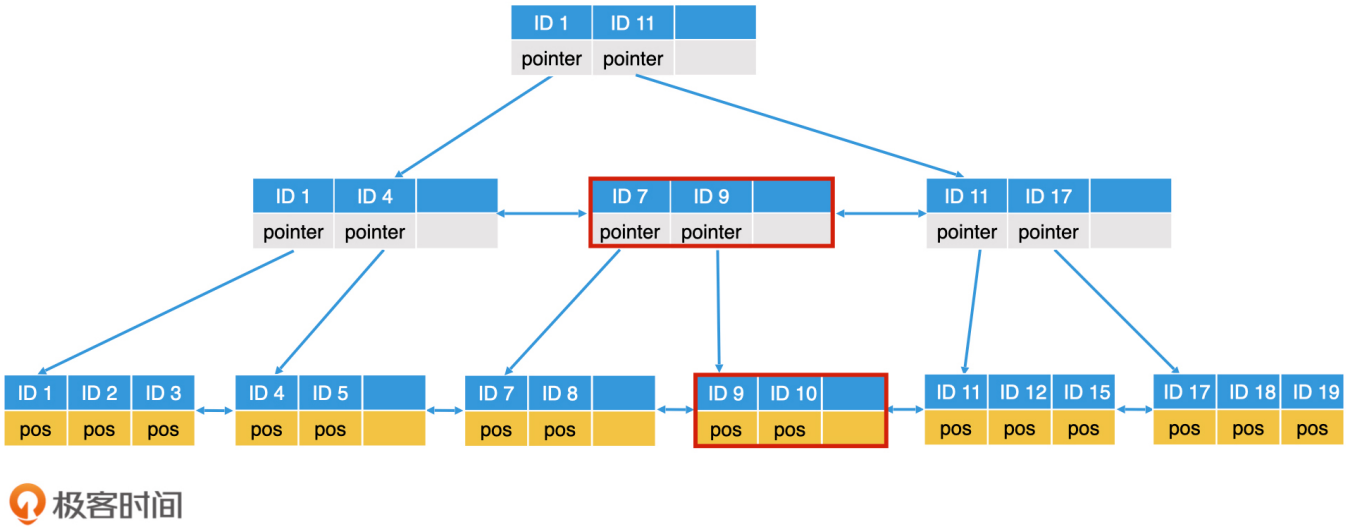
插入 ID 6

如果我们插入的是 ID=10 的节点呢？按之前的逻辑，我们应该插入到 ID 9 后面，但是 ID 9 所在的这个节点已经存满了 3 个节点，无法继续存入了。因此，我们需要将该叶子节点**分裂**。分裂的逻辑就是生成一个新节点，并将数据在两个节点中平分。



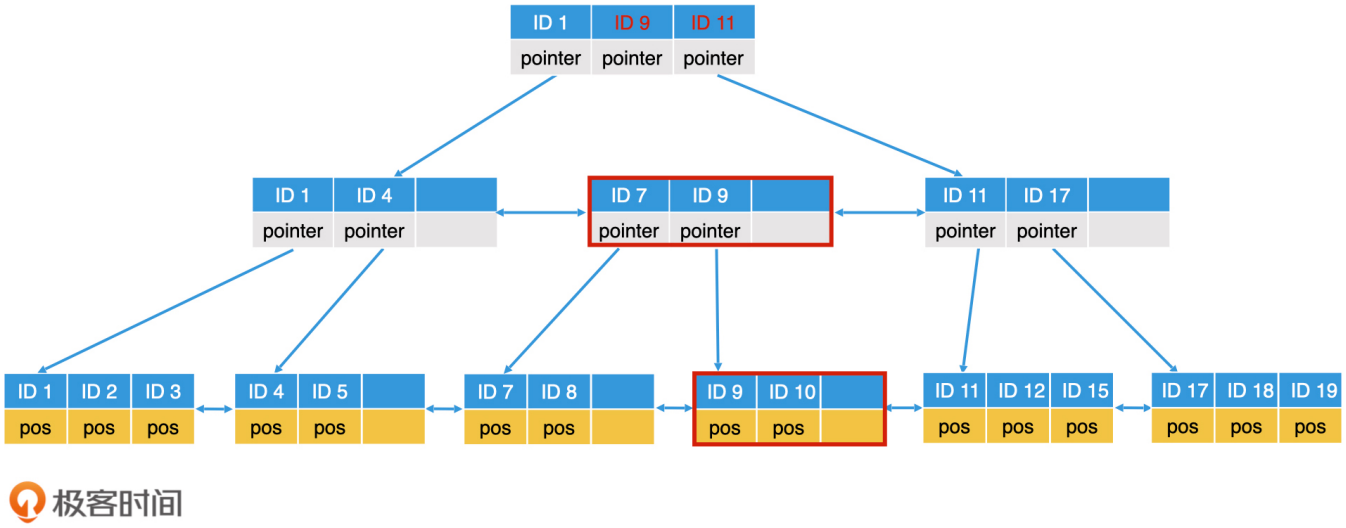
插入 ID 10，叶子节点分裂

叶子节点分裂完成以后，上一层的内部节点也需要修改。但如果上一层的父节点也是满的，那么上一层的父节点也需要分裂。



插入 ID 10，内部节点修改

内部节点调整好了，下一步我们就要调整根节点了。由于根节点未满，因此我们不需要分裂，直接修改即可。



插入 ID 10，根节点修改

删除数据也类似，如果节点数组较满，直接删除；如果删除后数组有一半以上的空间为空，那为了提高节点的空间利用率，该节点需要将左右两边兄弟节点的元素转移过来。可以成功转移的条件是，元素转移后该节点及其兄弟节点的空间必须都能维持在半满以上。如果无法满足这个条件，就说明兄弟节点其实也足够空闲，那我们直接将该节点的元素并入兄弟节点，然后删除该节点即可。

重点回顾

好了，今天的内容就先讲到这里。你会发现，即使是复杂的 B+ 树，我们将它拆解开来，其实也是由简单的数组、链表和树组成的，而且 B+ 树的检索过程其实也是二分查找。因此，如果 B+ 树完全加载在内存中的话，它的检索效率其实并不会比有序数组或者二叉检索树更高，也还是二分查找的 log(n) 的效率。并且，它还比数组和二叉检索树更加复杂，还会带来额外的开销。

但是，B+ 树最大的优点在于，它提供了将索引数据存在磁盘中，以及高效检索的方案。这让检索技术摆脱了内存的限制，得到了更广泛地使用。

另外，这一节还有一个很重要的设计思想需要你掌握，那就是将索引和数据分离。通过这样的方式，我们能将索引的数组大小保持在一个较小的范围内，让它能加载在内存中。在许多大规模系统中，都是使用这个设计思想来精简索引的。而且，B+ 树的内部节点和叶子节点的区分，其实也是索引和数据分离的一次实践。

课堂讨论

最后，咱们来看一道讨论题。

B+ 树有一个很大的优势，就是适合做范围查询。如果我们要检索值在 x 到 y 之间的所有元素，你会怎么操作呢？

欢迎在留言区畅所欲言，说出你的思考过程和最终答案。如果有收获，也欢迎把这篇文章分享给你的朋友。

上一篇　04 | 状态检索：如何快速判断一个用户是否存在?

下一篇　特别加餐 | 倒排检索加速（一）：工业界如何利用跳表、哈希表、位图进行加速?

精选留言 (7)

写留言

无形
2020-04-08

先找到x所在叶子节点，叶子节点数据二分查找找到x，然后向右遍历直到不在x和y区间内
感谢老师的分享，又学到了好多👏，我之前写的就只是有序数组二分查找，是时候好好改造了😀

作者回复: 答案正确。你可以再对比思考一下，为什么第二讲中的课后问题，我们对于数组的范围查找可以使用两次二分查找，但是在b+树中，却是二分查找（类似）+遍历?
此外，顺便多说一点，这一篇文章我其实讲了三点:
1.磁盘特性
2.索引与数据分离设计
3.b+树
希望对你有帮助

猫和老鼠
2020-04-08

感谢陈东哥，从计算机的组成原理到B+树，娓娓而谈，自己受益良多。
我一个非计算专业的都可以听得很明白。

其实，各种高级的算法和数据结构，都是在不同的应用场景下，对最基础的数据结构和算法进行的组合。所以说，基本功很重要！...
展开

pedro
2020-04-08

问题: 先找到 x 然后遍历找到 y，原因大概是涉及磁盘操作，顺序 io 的速度远大于随机 io，因此如果找 y 也使用二分搜索的话，io 成本高，消耗的时间大于顺利遍历。

牛牛
2020-04-08

B+树
1. 只有叶子节点存储数据
2. 一个节点上存储的是一组数据, 数组存储
3. 同一层的节点之间用双向链表串在一起
这样的话、我觉得范围查询时、可以先找到 min 所在的叶子节点位置、再找到max所在...
展开

范闲
2020-04-08

为什么要有B+ tree
1.数据容量过大的时候，内存无法一次性加载
2.内存的读写速度比磁盘有数量级上的优势，所以不能将数据全部放入磁盘（速度慢），也不能全部放入内存（资源昂贵）
B+:索引和数据分离，三层数结构。索引存入内存提高速度，数据存入磁盘提高存储容量...
展开

作者回复: 你的总结很详细！相信这篇文章已经部分解决了你之前关于“数据量太大如何处理”的问题。
此外，关于课后讨论题，的确我们不需要走step（y - x）步，因为每个叶子节点可以直接判断自

己存的最大一个元素是什么（数组尾部元素），因此，可以通过对比y和这个元素的值，来决定是否需要读取下一个叶子节点。

夜空中最亮的星（华仔...

2020-04-08

今天的课需要听2遍

展开

作者回复: 这是个好的学习习惯！你会发现，这篇写b+树的文章内容可能和你之前学习过的不一样。
我会从磁盘特性出发，结合索引与数据分离的设计思想和你解释b+树的来龙去脉。
学习b+树不是目的，而是一个学习“磁盘上的数据和索引应该如何处理”的过程。毕竟大数据时代，数据的存储和检索往往都要和磁盘打交道，数据库，日志系统，搜索引擎等都要解决这个问题。

Mq

2020-04-08

先找大于等于x的元素，然后遍历元素判断是否大于y如果大于就终止

作者回复: 是的！找到x的过程，是树形结构查找（类似二分查找），找到y的过程，是从x开始遍历。我觉得你可以再深入对比思考一下了，为什么第二讲中，对于数组的范围查找，我们可以使用两次二分查找，但是b+树就是一次二分+遍历了？