

① B+树的索引



CONTENTS

Never put off what you can
do today until tomorrow

▷ B+树索引结构

对B+树索引的整体结构进行讲解

▷ B+树索引管理机制

对B+树查询和更新操作进行讲解

▷ B+树索引实现方式

对B+树的文件组织进行描述

▷ B+树的比较分析

对B+树进行比较分析



B+树索引结构

对B+树索引的整体结构进行讲解

You cannot improve your past, but you can improve your future.
Once time is wasted, life is wasted.

B+树索引结构

For man is man and
master of his fate.



从物理上说，索引通常可以分为：分区和非分区索引、常规B树索引、位图（bitmap）索引、翻转（reverse）索引等。其中，B树索引属于最常见的索引B树索引是一个典型的树结构，其包含的组件主要是：

叶子节点（Leaf node）：包含条目直接指向表里的数据行。

分支节点（Branch node）：包含的条目指向索引里其他的分支节点或者是叶子节点。

根节点（Root node）：一个B树索引只有一个根节点，它实际就是位于树的最顶端分支节点。

B+树索引结构



B+树的特性

有n棵子树的结点中含有n个关键码

所有的叶子结点中包含了全部关键码的信息，及指向含有这些关键码记录的指针，且叶子结点本身依关键码的大小自小而大的顺序链接。

所有的非终端结点可以看成是索引部分，结点中仅含有其子树根结点中最大（或最小）关键码。

B+树索引结构



B+树的特性

树上的操作（插入，删除）能保持数的平衡

如果能实现删除算法，则除根节点外，每一个节点都将保证最小50%的占有率。但是，删除的实现通常是简单的定位数据项，并移走它，而不为保证50%的占有率而调整它，这主要是因为文件在典型情况下是增长而不是缩小。

搜索记录需要从根开始遍历到合适的叶子。从根到叶子的路径长度称为树的高度，因为是平衡树，所以从根到任意叶子的长度都是相同的。

B+树索引结构

For man is man and
master of his fate.

B+树的结点结构

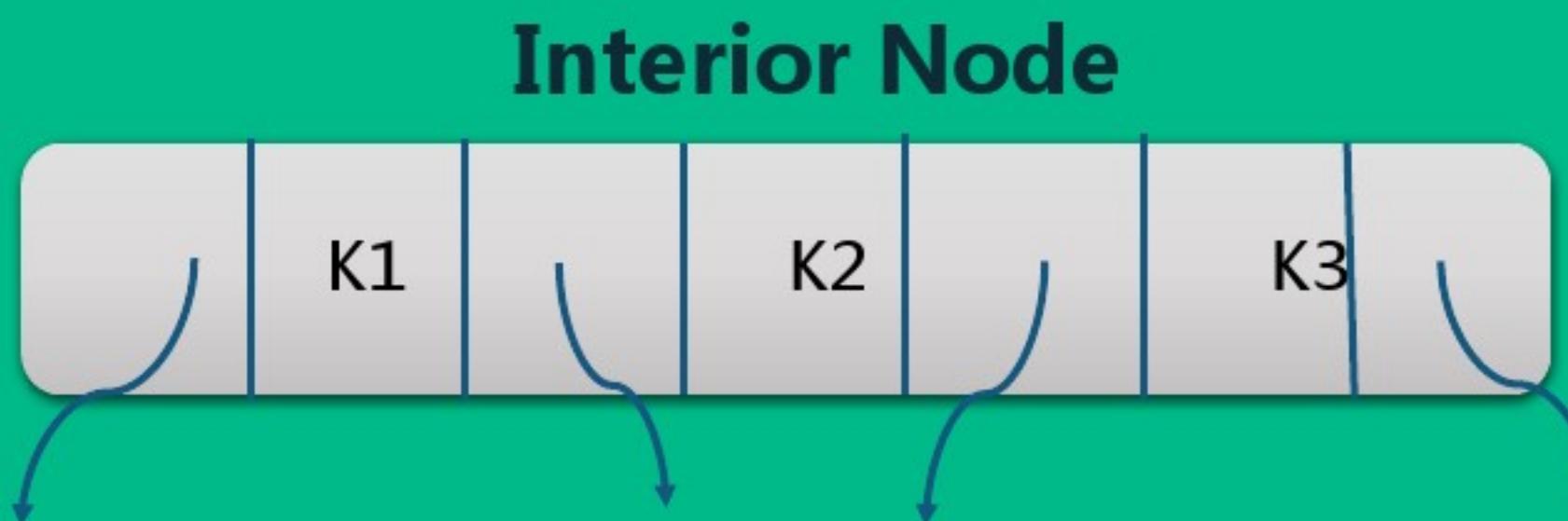


对一 $a=1, 2, \dots, n$ ，指针 P_i 只指向具有搜索码值 K_i 的一个文件记录或指向一个指针桶，桶中的每个指针指向具有搜索码值 K_i 的一个文件记录，指针桶只在搜索码不是主码且文件不按搜索码顺序存放时才使用。因为各叶结点之间按照所含的搜索码值大小有一个线性的顺序，我们就用 P_n 将叶结点按搜索码顺序串在一起。这种排序使我们能够高效地对文件进行顺序处理。这就是指针 P_n 有特殊的作用

B+树索引结构

For man is man and
master of his fate.

B+树的结点结构

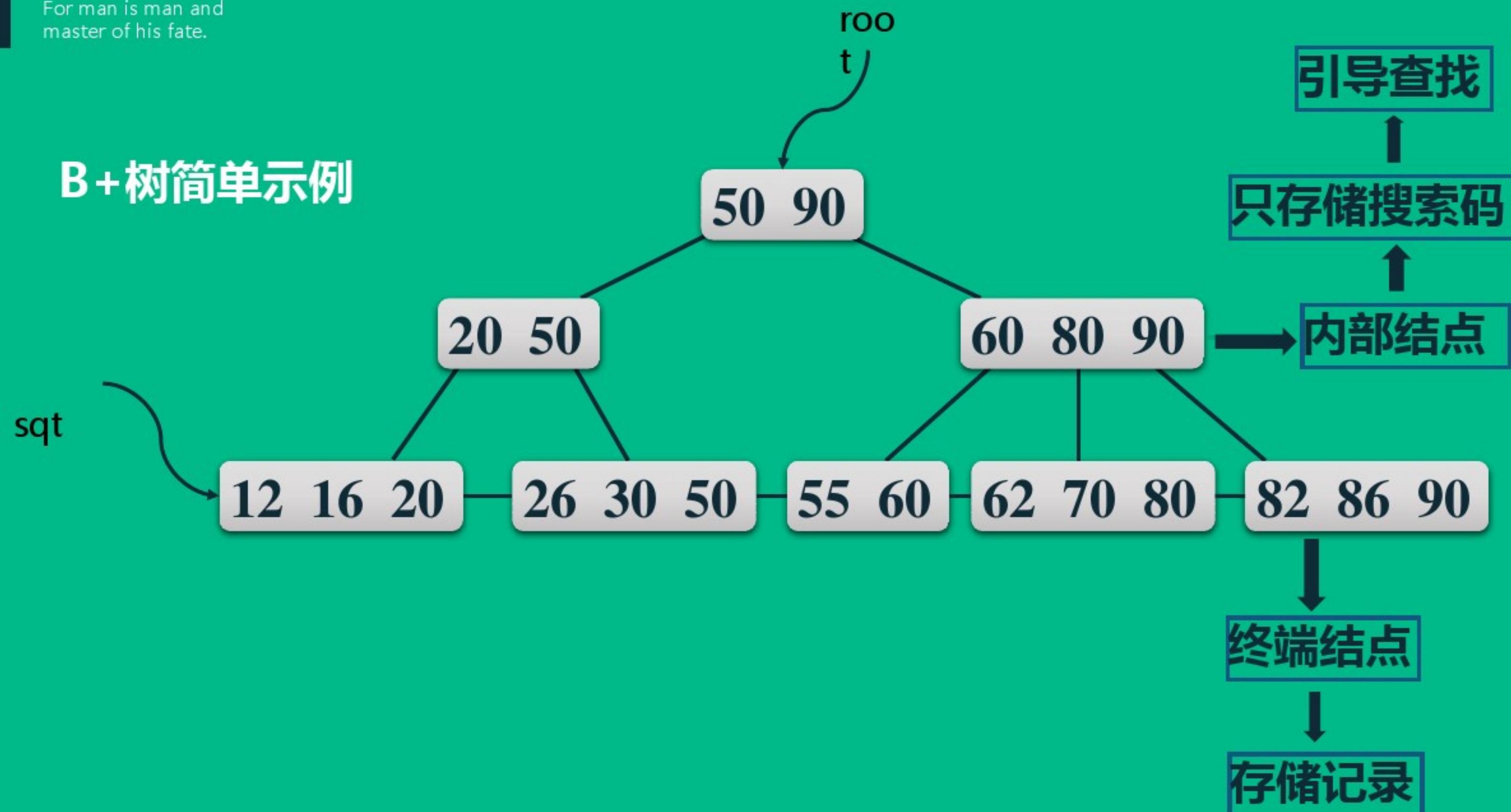


B+树的非叶结点形成叶结点上的一个多层次(稀疏)索引。非叶结点的结构和叶结点相同，只不过非叶结点中的所有指针都指向树中的结点。一个非叶结点至少包含 $[n/2]$ 个指针，最多 n 个。结点的指针数称为该结点的扇出。

B+树索引结构

For man is man and
master of his fate.

B+树简单示例



B+树索引结构

For man is man and
master of his fate.

B+树索引的访问

我们已经知道了B树索引的体系结构，那么当Oracle需要访问索引里的某个索引条目时，Oracle是如何找到该索引条目所在的数据块的呢？

当oracle进程需要访问数据文件里的数据块时，oracle会有两种类型的I/O操作方式：

随机访问

顺序访问

每次读取一个数据块
(通过等待事件“**db
file sequential read**”体现出来)。

每次读取多个数据
块(通过等待事件
“**db file
scattered read**”
体现出来)。

B+树索引结构

For man is man and
master of his fate.

■ B+树索引的访问

第一种方式则是访问索引里的数据块，而第二种方式的I/O操作属于全表扫描。这里顺带有一个问题，为何随机访问会对应到db file sequential read等待事件，而顺序访问则会对应到db file scattered read等待事件呢？这似乎反过来了，随机访问才应该是分散（scattered）的，而顺序访问才应该是顺序（sequential）的。其实，等待事件主要根据实际获取物理I/O块的方式来命名的，而不是根据其在I/O子系统的逻辑方式来命名的。下面对于如何获取索引数据块的方式中会对此进行说明。



B+树索引结构

For man is man and
master of his fate.

■ B+树索引的访问

当oracle需要获得一个索引块时，首先从根节点开始，根据所要查找的键值，从而知道其所在的下一层的分支节点，然后访问下一层的分支节点，再次同样根据键值访问再下一层的分支节点，如此这般，最终访问到最底层的叶子节点。可以看出，其获得物理I/O块时，是一个接着一个，按照顺序，串行进行的。在获得最终物理块的过程中，我们不能同时读取多个块，因为我们在没有获得当前块的时候是不知道接下来应该访问哪个块的。因此，在索引上访问数据块时，会对应到db file sequential read等待事件，其根源在于我们是按照顺序从一个索引块跳到另一个索引块，从而找到最终的索引块的。



B+树索引结构

For man is man and
master of his fate.

■ B+树索引的访问

那么对于全表扫描来说，则不存在访问下一个块之前需要先访问上一个块的情况。全表扫描时，oracle知道要访问所有的数据块，因此唯一的问题就是尽可能高效的访问这些数据块。因此，这时oracle可以采用同步的方式，分几批，同时获取多个数据块。这几批的数据块在物理上可能是分散在表里的，因此其对应到db file scattered read等待事件。



B+树索引结构

For man is man and
master of his fate.

快速的Record查找

快速的Record遍历

不通过overflow page的形式维护排序树结构

B+树功能

A

B

C





B+树索引管理机 制

Action speak louder than words.

You cannot improve your past, but you can improve your future.
Once time is wasted, life is wasted.

B+树索引管理机制

For man is man and
master of his fate.

B+树索引的查找管理

对B+树可以进行两种查找运算：

- 1.从最小关键字起；
- 2.从根结点开始，进行随机查找。

在查找时，若非终端结点上的关键值等于给定值，
并不终止，而是继续向下直到叶子结点。因此，在
B+树中，不管查找成功与否，每次查找都是走了一
条从根到叶子结点的路径。其余同B-树的查找类似。



B+树索引管理机制

For man is man and
master of his fate.

B+树索引的插入管理

一种是在一个已经充满了数据的表上创建索引时，索引是怎么管理的。

当在一个充满了数据的表上创建索引（`create index`命令）时，Oracle会先扫描表里的数据并对其进行排序，然后生成叶子节点。生成所有的叶子节点以后，根据叶子节点的数量生成若干层级的分支节点，最后生成根节点。这个过程是很清晰的。

另一种则是当一行接着一行向表里插入或更新或删除数据时，索引是怎么管理的。

当一开始在一个空的表上创建索引的时候，该索引没有根节点，只有一个叶子节点。随着数据不断被插入表里，该叶子节点中的索引条目也不断增加，当该叶子节点充满了索引条目而不能再放下新的索引条目时，该索引就必须扩张，必须再获取一个可用的叶子节点。这时，索引就包含了两个叶子节点，但是两个叶子节点不可能单独存在的，这时它们必须有一个上级的分支节点，其实这也就是根节点了。

第一种情况

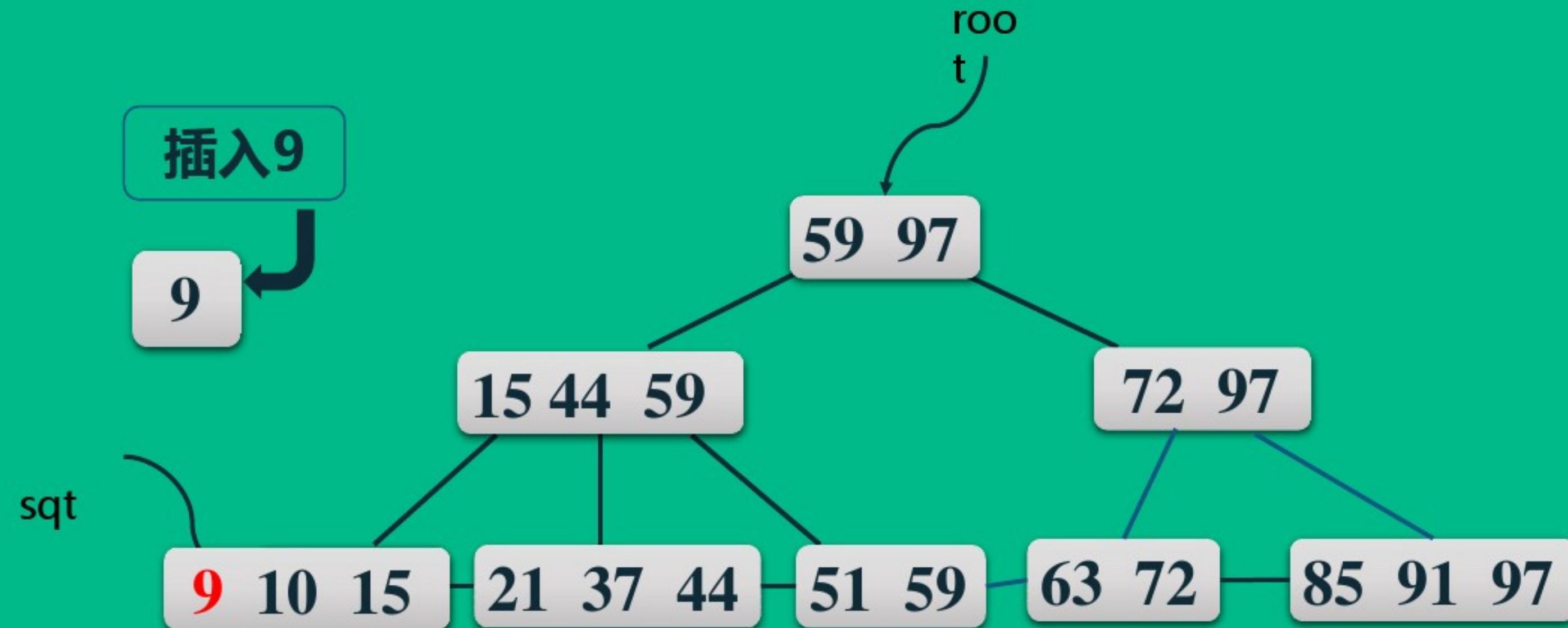
第二种情况



B+树索引管理机制

For man is man and
master of his fate.

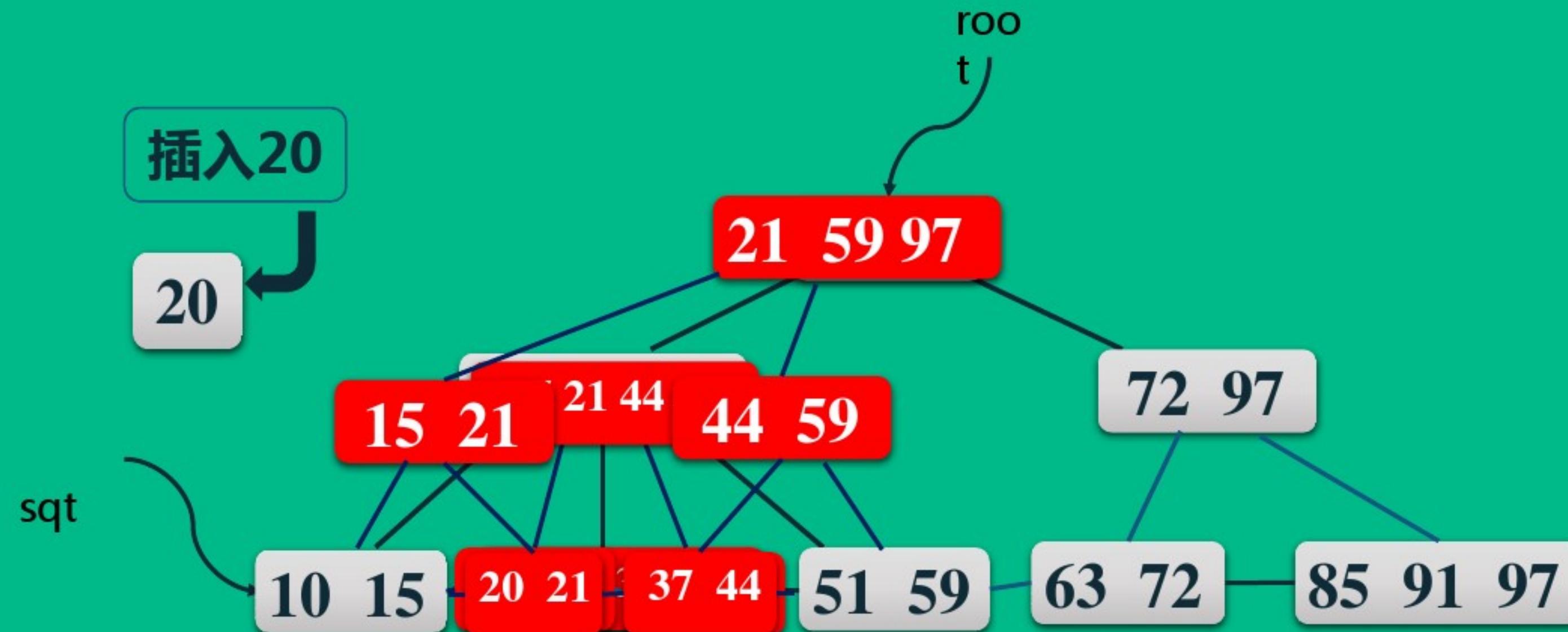
3阶B+树插入简单示例



B+树索引管理机制

For man is man and
master of his fate.

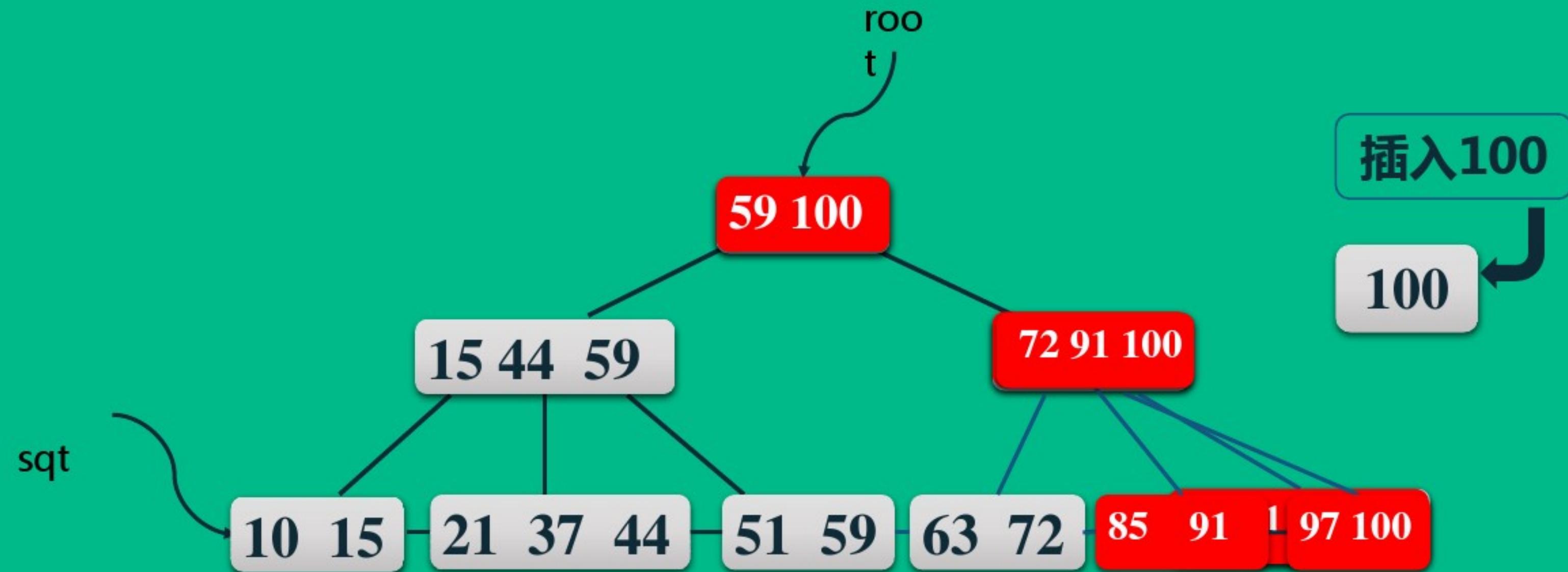
3阶B+树插入简单示例



B+树索引管理机制

For man is man and
master of his fate.

3阶B+树插入简单示例



B+树索引管理机制

For man is man and
master of his fate.

B+树索引的删除管理

当删除表里的一条记录时，其对应于索引里的索引条目并不会被物理的删除，只是做了一个删除标记。

当一个新的索引条目进入一个索引叶子节点的时候，Oracle会检查该叶子节点里是否存在被标记为删除的索引条目，如果存在，则会将所有具有删除标记的索引条目从该叶子节点里物理的删除。

当一个新的索引条目进入索引时，oracle会将当前所有被清空的叶子节点（该叶子节点中所有的索引条目都被设置为删除标记）收回，从而再次成为可用索引块。

B+树索引管理机制

For man is man and
master of his fate.

尽管被删除的索引条目所占用的空间大部分情况下都能够被重用，但仍然存在一些情况可能导致索引空间被浪费，并造成索引数据块很多但是索引条目很少的后果，这时该索引可以认为出现碎片。而导致索引出现碎片的情况主要包括：

The First



不合理的、较高的 PCTFREE。很明显，这将导致索引块的可用空间减少。

The Second



索引键值持续增加（比如采 sequence 生成序列号的键值），同时对索引键值按照顺序连续删除，这时可能导致索引碎片的发生。

The Last

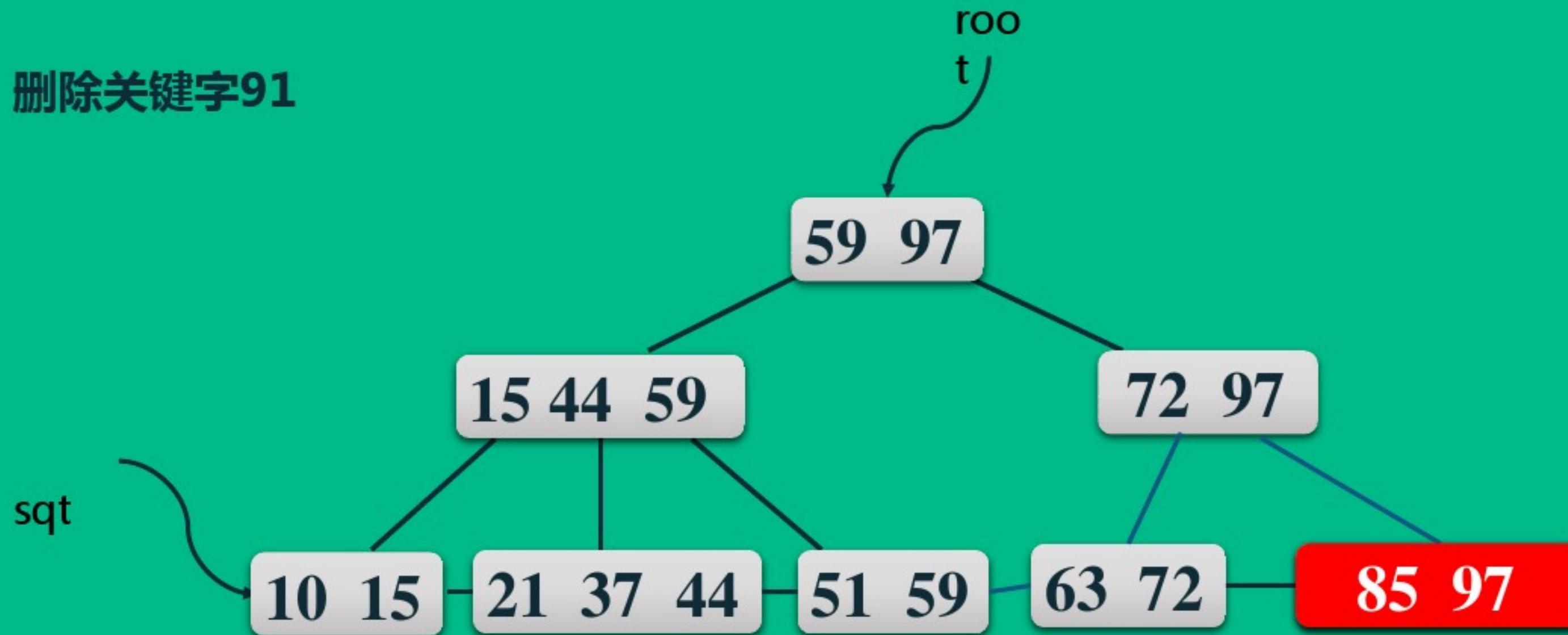
经常被删除或更新的键值，以后几乎不再会被插入时，这种情况与上面的情况类似。

B+树索引管理机制

For man is man and
master of his fate.

B+树索引删除的简单示例

删除关键字91

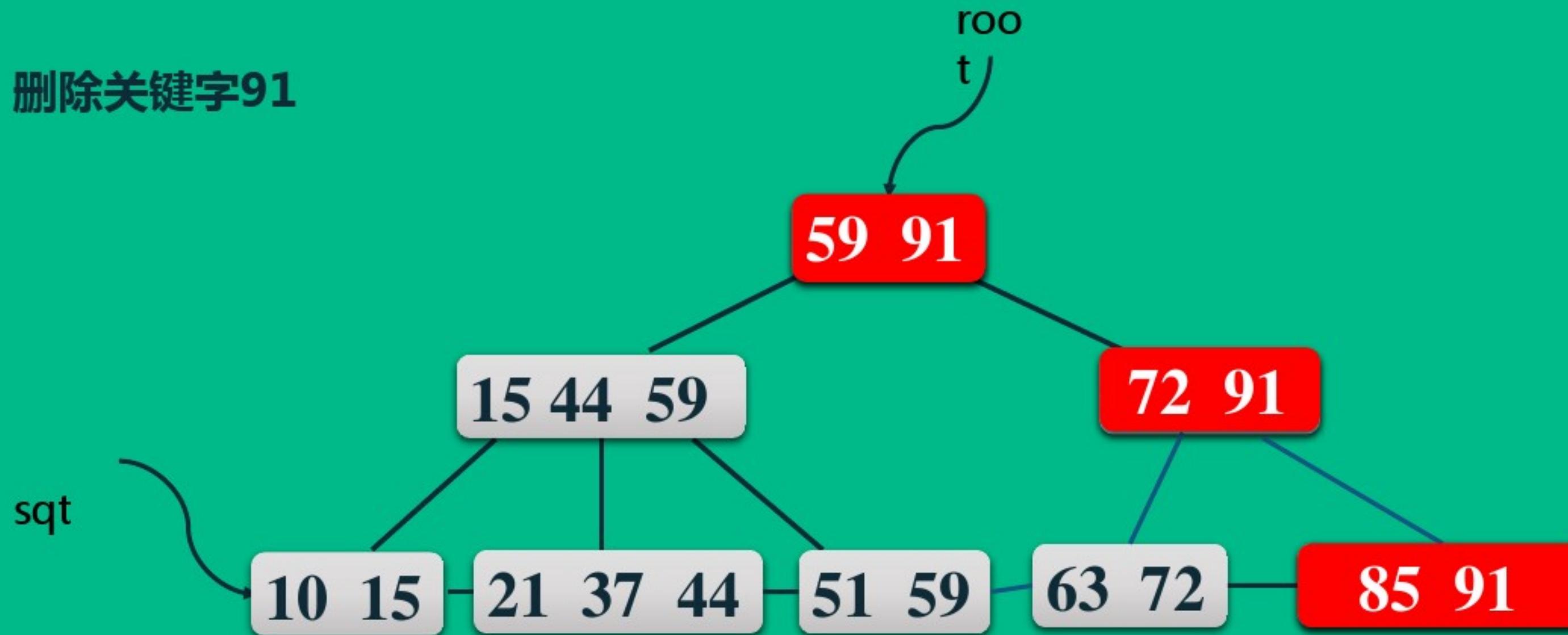


B+树索引管理机制

For man is man and
master of his fate.

B+树索引删除的简单示例

删除关键字91

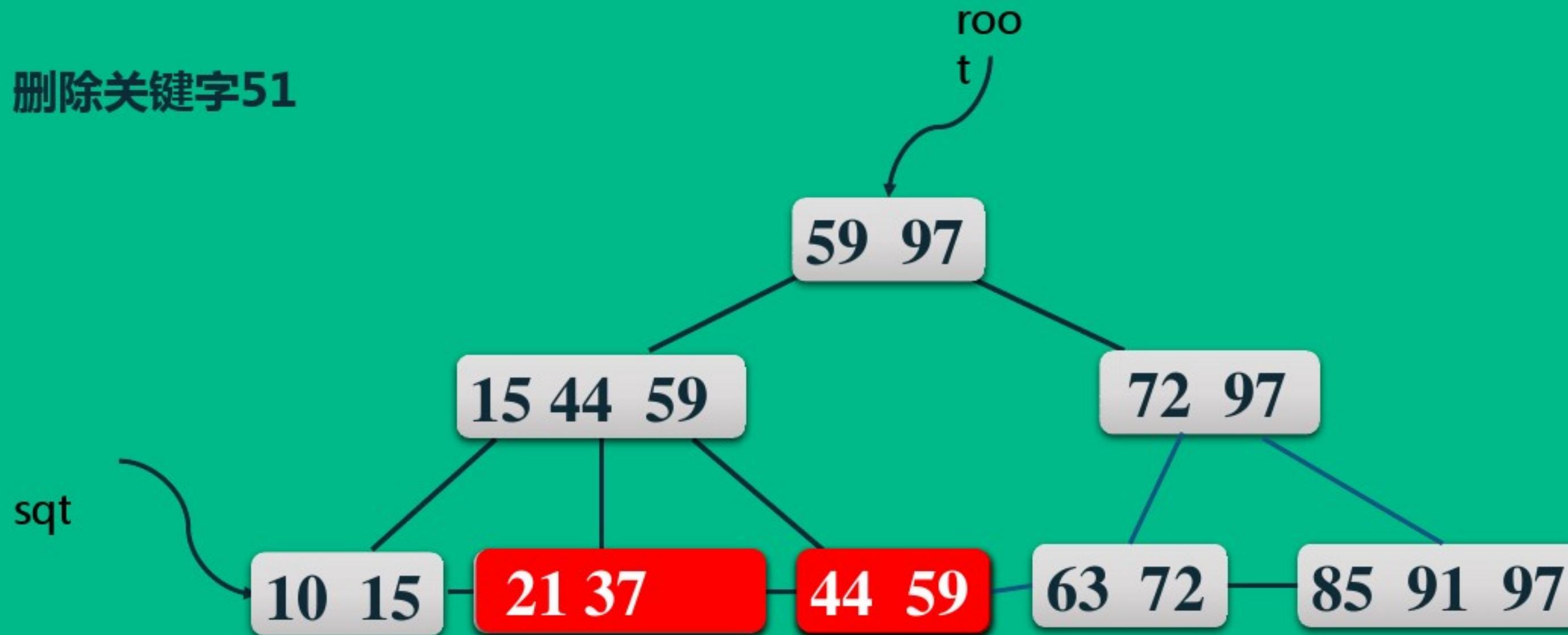


B+树索引管理机制

For man is man and
master of his fate.

B+树索引删除的简单示例

删除关键字51

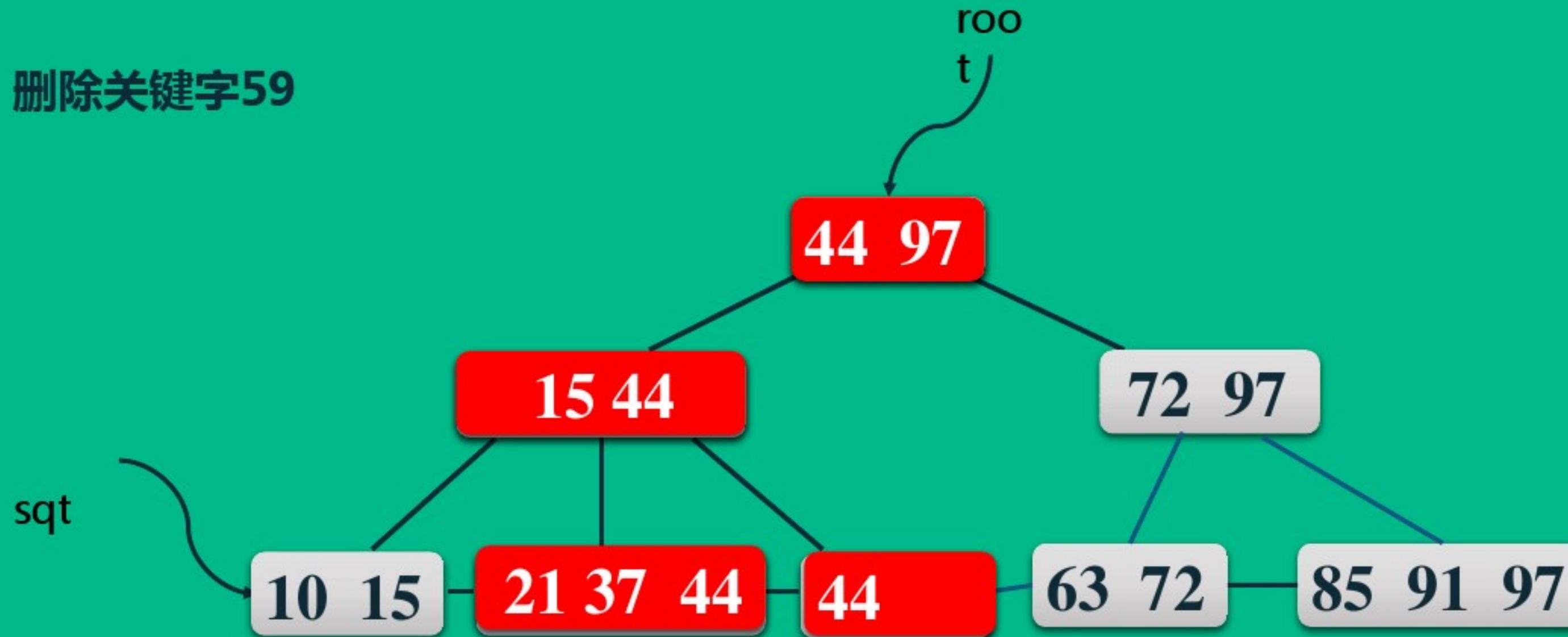


B+树索引管理机制

For man is man and
master of his fate.

B+树索引删除的简单示例

删除关键字59

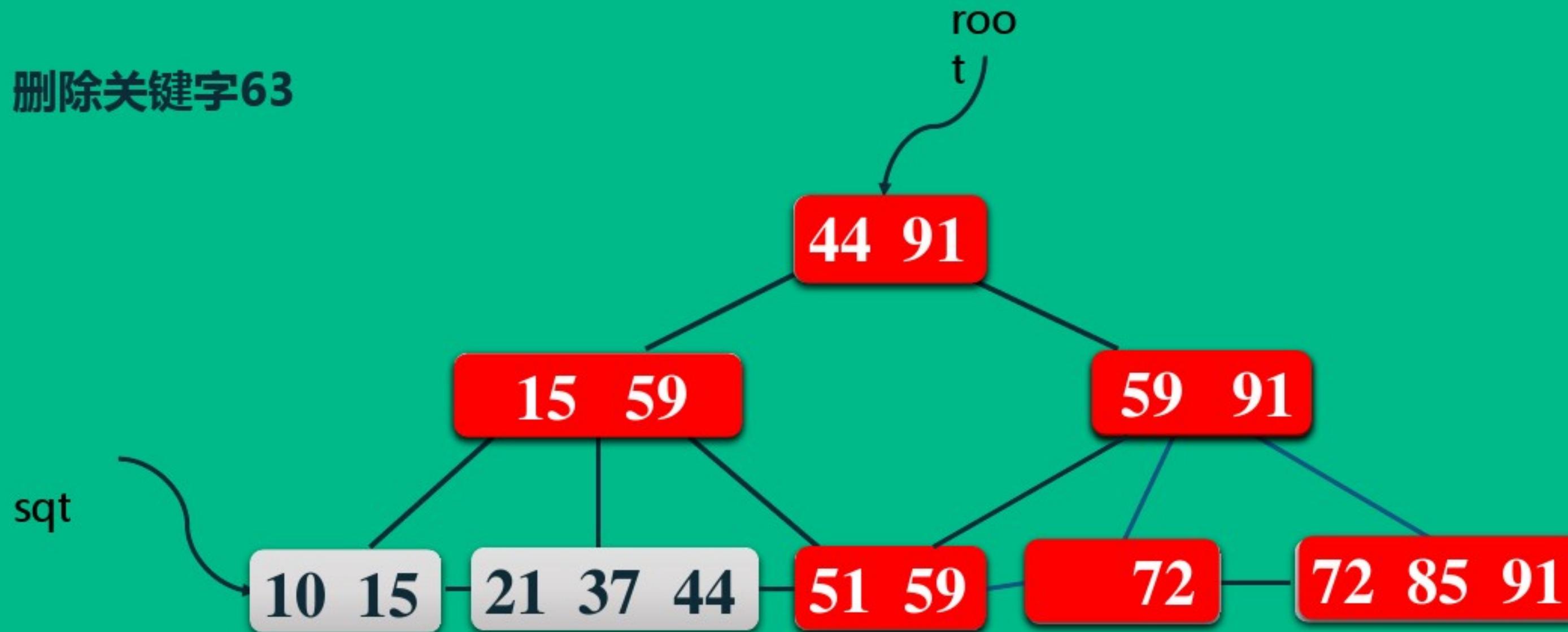


B+树索引管理机制

For man is man and
master of his fate.

B+树索引删除的简单示例

删除关键字63





B+树索引实现方式

Action speak louder than words.

You cannot improve your past, but you can improve your future.
Once time is wasted, life is wasted.

B+树索引实现方式

For man is man and
master of his fate.

下面主要讨论**MyISAM**和**InnoDB**两个存储引擎的索引实现方式：

MyISAM

MyISAM是默认存储引擎。它基于更老的ISAM代码，但有很多有用的扩展。（注意MySQL 5.1不支持ISAM）。每个MyISAM在磁盘上存储成三个文件。每一个文件的名字均以表的名字开始，扩展名指出文件类型。.frm文件存储表定义。数据文件的扩展名为·MYD (MYData)。

InnoDB

InnoDB，是MySQL的数据库引擎之一，为MySQL AB发布binary的标准之一。InnoDB由Innodb Oy公司所开发，2006年五月时由甲骨文并购。与传统的ISAM与MyISAM相比，InnoDB的最大特色就是支持了ACID兼容的事物（Transaction）功能，类似于PostgreSQL。

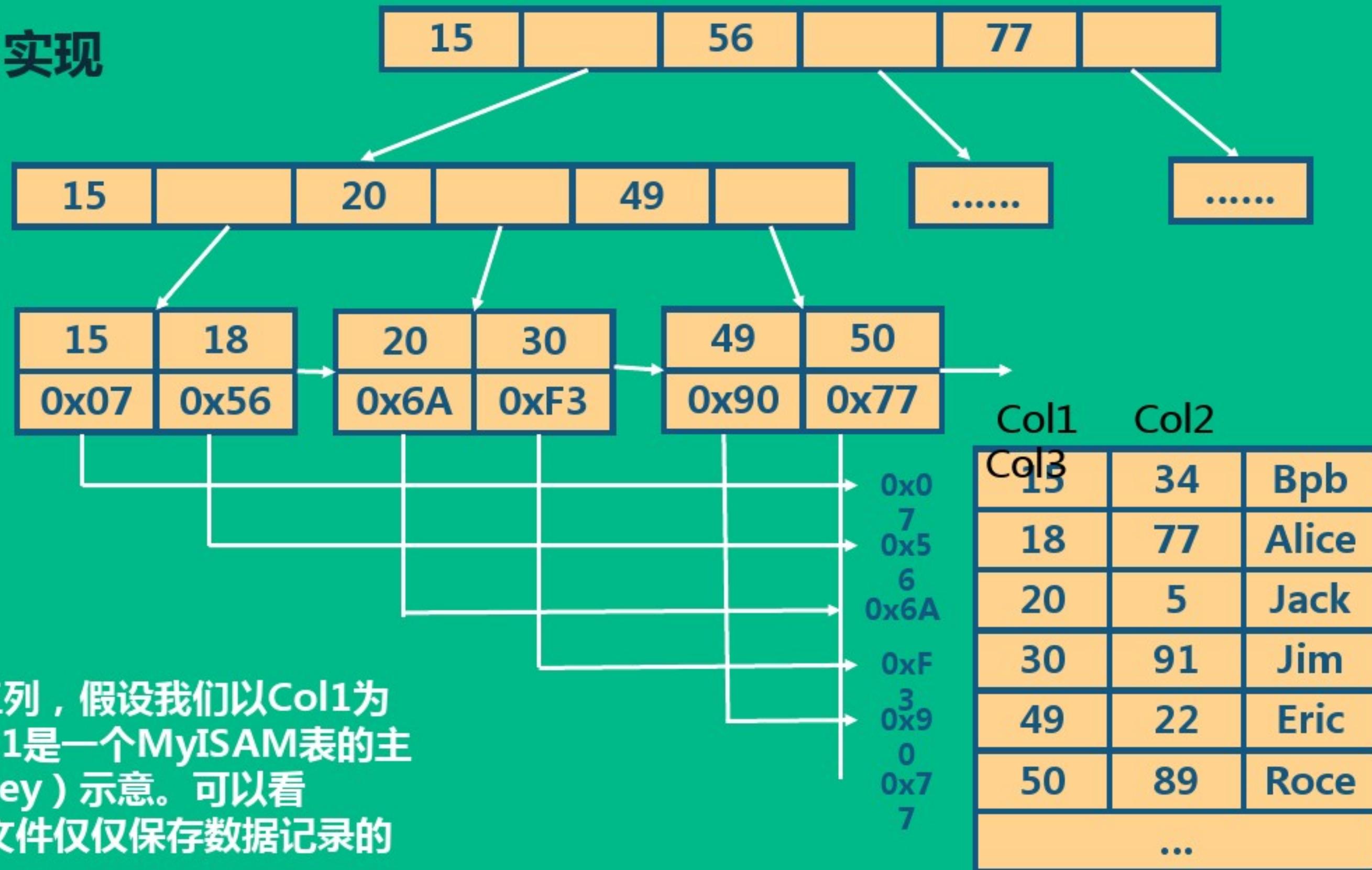
B+树索引实现方式

For man is man and
master of his fate.

MyISAM索引实现



主键索引



这里设表一共有三列，假设我们以Col1为主键，图myisam1是一个MyISAM表的主索引 (Primary key) 示意。可以看MyISAM的索引文件仅仅保存数据记录的地址。

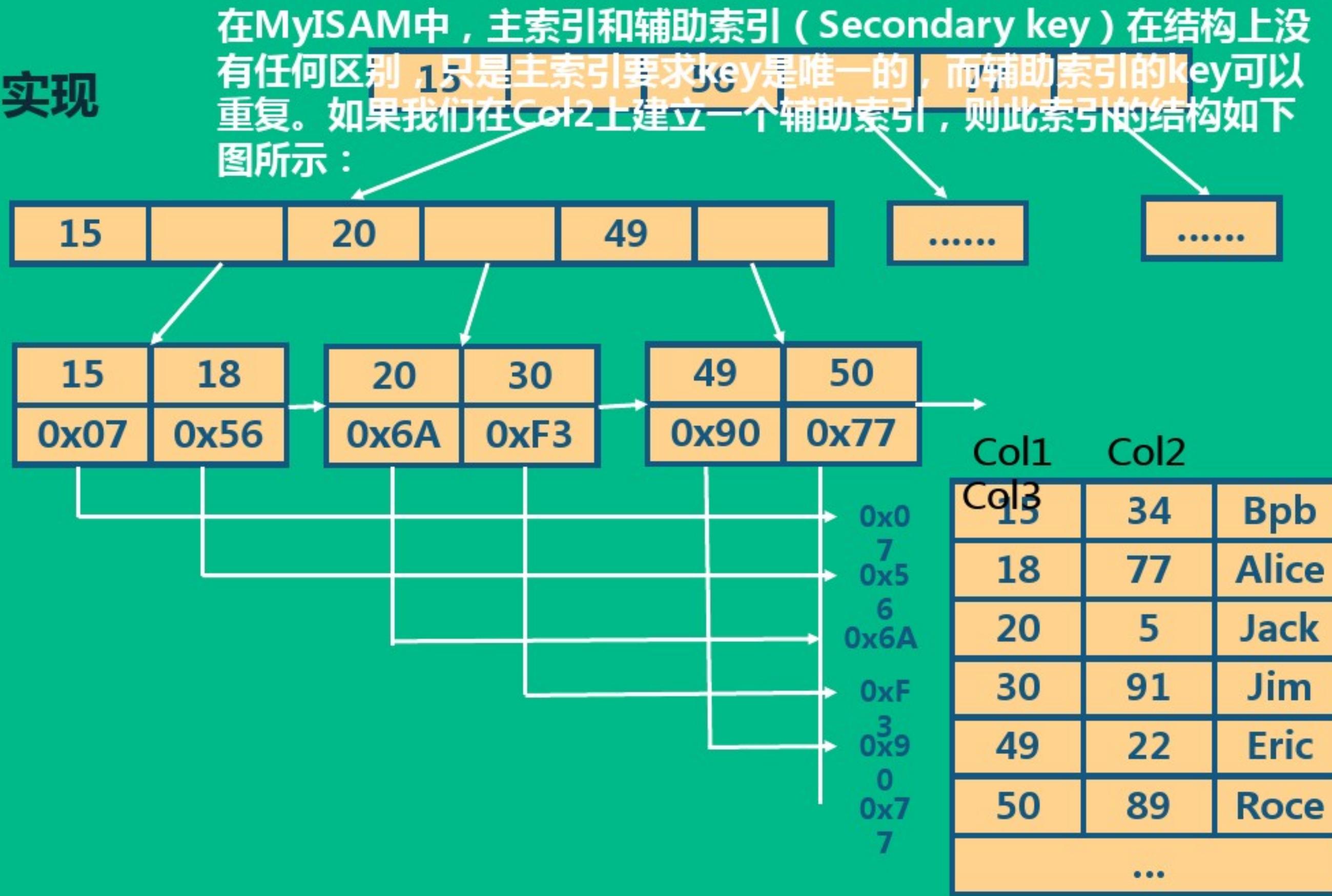
B+树索引实现方式

For man is man and
master of his fate.

MyISAM索引实现



辅助索引



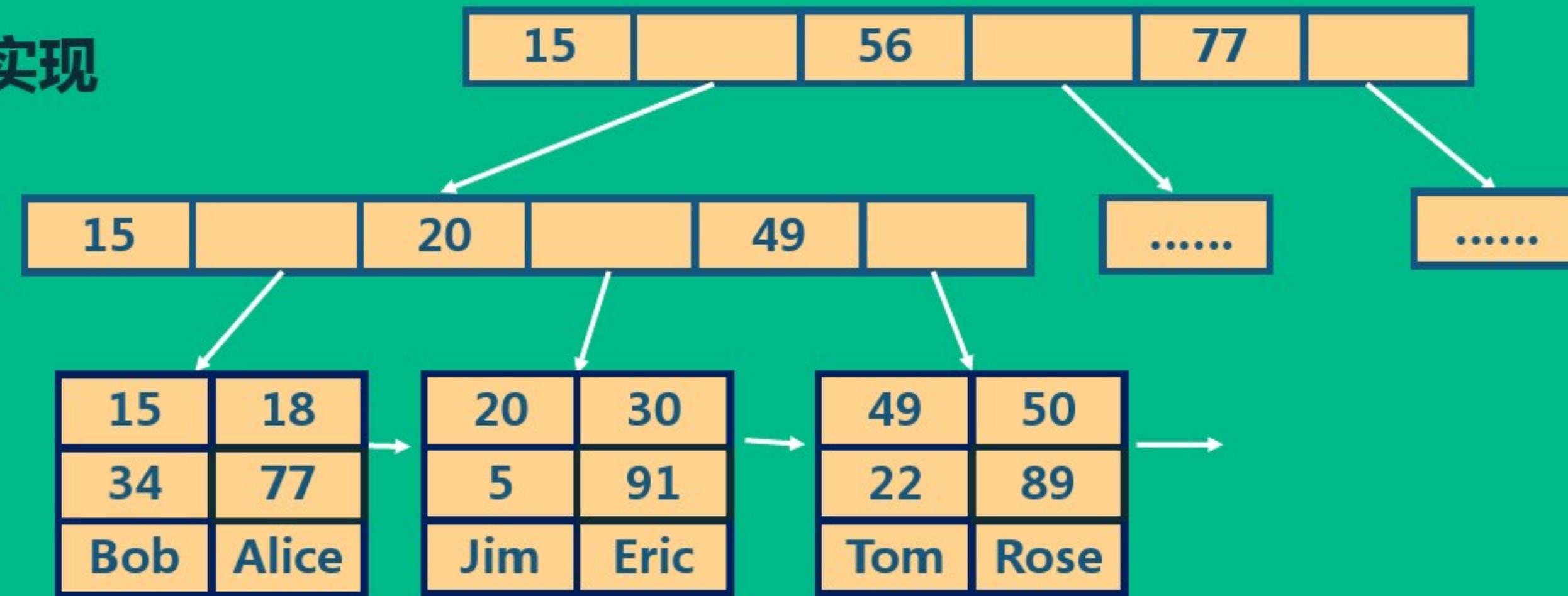
B+树索引实现方式

For man is man and
master of his fate.

InnoDB索引实现



主键索引



InnoDB主索引（同时也是数据文件）的示意图，可以看到叶节点包含了完整的数据记录。这种索引叫做聚集索引。因为InnoDB的数据文件本身要按主键聚集，所以InnoDB要求表必须有主键（MyISAM可以没有），如果没有显式指定，则MySQL系统会自动选择一个可以唯一标识数据记录的列作为主键，如果不存在这种列，则MySQL自动为InnoDB表生成一个隐含字段作为主键，这个字段长度为6个字节，类型为长整形。

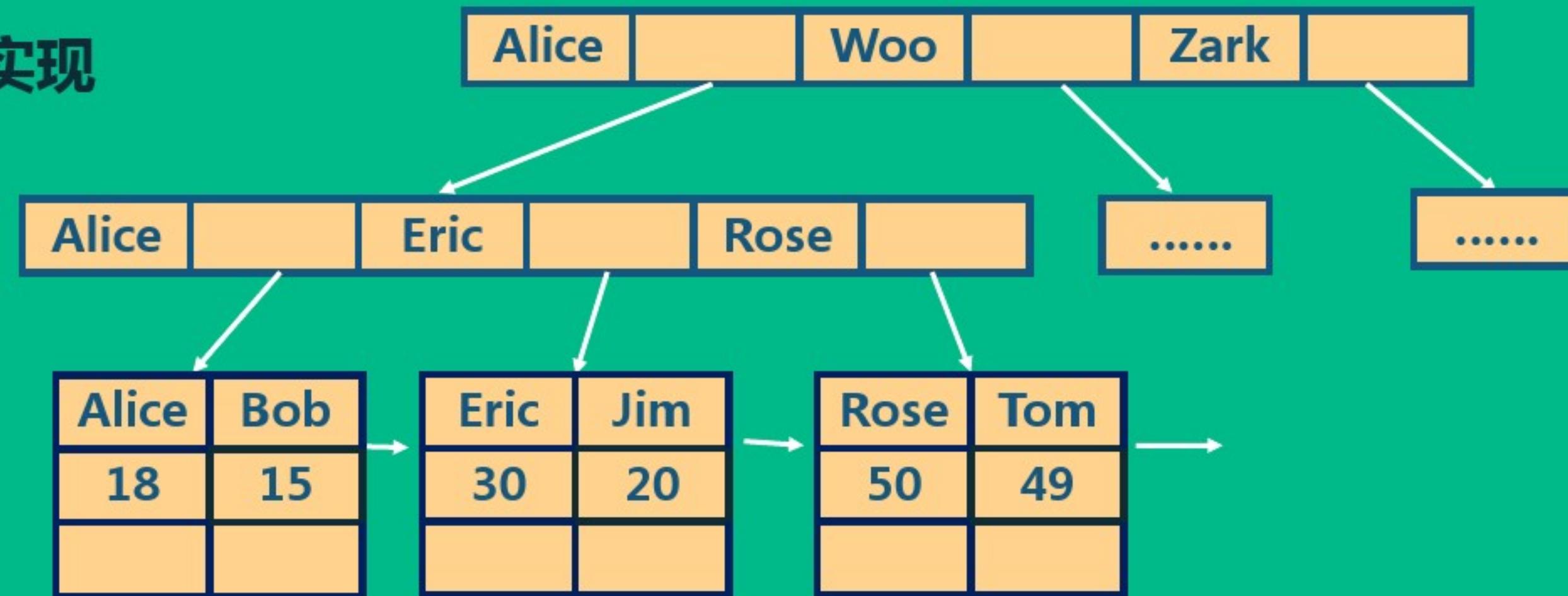
B+树索引实现方式

For man is man and
master of his fate.

InnoDB索引实现



辅助索引



InnoDB的所有辅助索引都引用主键作为data域

B+树索引实现方式

For man is man and
master of his fate.

同样也是一颗B+Tree，data域保存数据记录的地址。因此，MyISAM中索引检索的算法为首先按照B+Tree搜索算法搜索索引，如果指定的Key存在，则取出其data域的值，然后以data域的值为地址，读取相应数据记录。

MyISAM的索引方式也叫做“非聚集”的，之所以这么称呼是为了与InnoDB的聚集索引区分。

InnoDB表是基于聚簇索引建立的。因此InnoDB的索引能提供一种非常快速的主键查找性能。不过，它的辅助索引（Secondary Index，也就是非主键索引）也会包含主键列，所以，如果主键定义的比较大，其他索引也将很大。如果想在表上定义很多索引，则争取尽量把主键定义得小一些。InnoDB不会压缩索引。



B+树索引实现方式

For man is man and
master of his fate.

补充一下

文字符的ASCII码作为比较准则。聚集索引这种实现方式使得按主键的搜索十分高效，但是辅助索引搜索需要检索两遍索引：首先检索辅助索引获得主键，然后用主键到主索引中检索获得记录。

不同存储引擎的索引实现方式对于正确使用和优化索引都非常有帮助，例如知道了InnoDB的索引实现后，就很容易明白为什么不建议使用过长的字段作为主键，因为所有辅助索引都引用主索引，过长的主索引会令辅助索引变得过大。再例如，用非单调的字段作为主键在InnoDB中不是个好主意，因为InnoDB数据文件本身是一颗B+ Tree，非单调的主键会造成在插入新记录时数据文件为了维持B+ Tree的特性而频繁的分裂调整，十分低效，而使用自增字段作为主键则是一个很好的选择。



B+树索引实现方式

For man is man and
master of his fate.



InnoDB索引和MyISAM索引的区别：

一.是主索引的区别，InnoDB的数据文件本身就是索引文件。而MyISAM的索引和数据是分开的。

二.是辅助索引的区别：InnoDB的辅助索引data域存储相应记录主键的值而不是地址。而MyISAM的辅助索引和主索引没有多大区别。





B+树的比较分析

Action speak louder than words.

You cannot improve your past, but you can improve your future.
Once time is wasted, life is wasted.

B+树的比较分析

For man is man and
master of his fate.

为什么选择B+树呢？

B+树中每个非叶节点没有指向某个关键字具体信息的指针，所以每一个节点可以存放更多的关键字数量，即一次性读入内存所需要查找的关键字也就越多，减少了I/O操作。

e.g.假设磁盘中的一个盘块容纳16bytes，而一个关键字2bytes，一个关键字具体信息指针2bytes。一棵9阶B-tree(一个结点最多8个关键字)的内部结点需要2个盘块。而B+树内部结点只需要1个盘块。当需要把内部结点读入内存中的时候，B树就比B+树多一次盘块查找时间(在磁盘中就是盘片旋转的时间)。

第一个原因



B+树的比较分析

For man is man and
master of his fate.

为什么选择B+树呢？

因为B+树的每次查询过程中，都需要遍历从根节点到叶子节点的某条路径。所有关键字的查询路径长度相同，导致每一次查询的效率相当。非叶子节点只存key，大大减少了非叶子节点的大小，那么每个节点就可以存放更多的记录，树更矮了，I/O操作更少了。所以B+Tree拥有更好的性能。

第二个原因



THANKS!