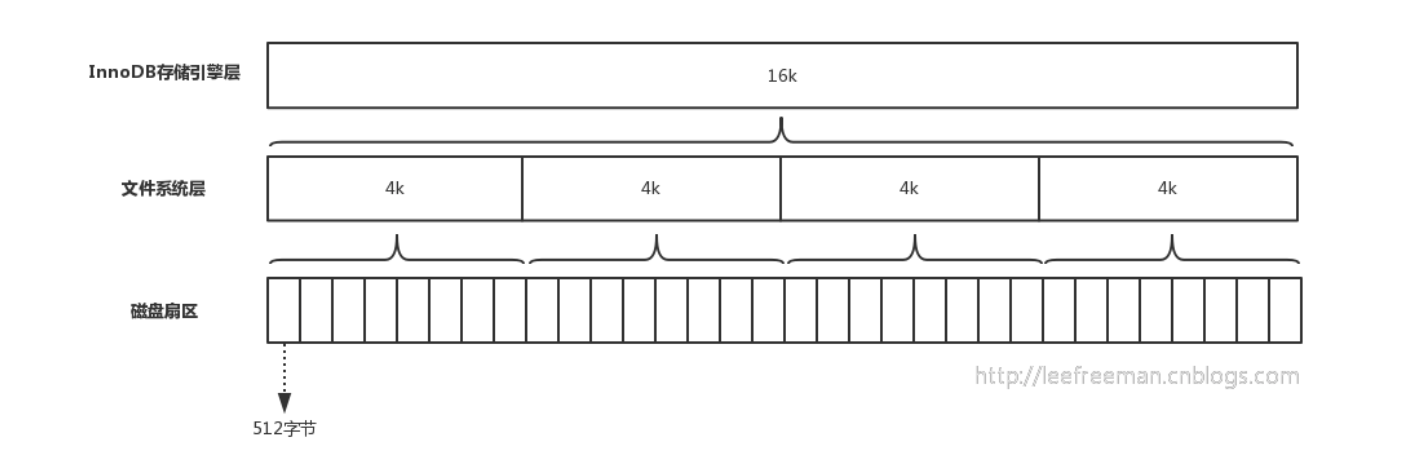
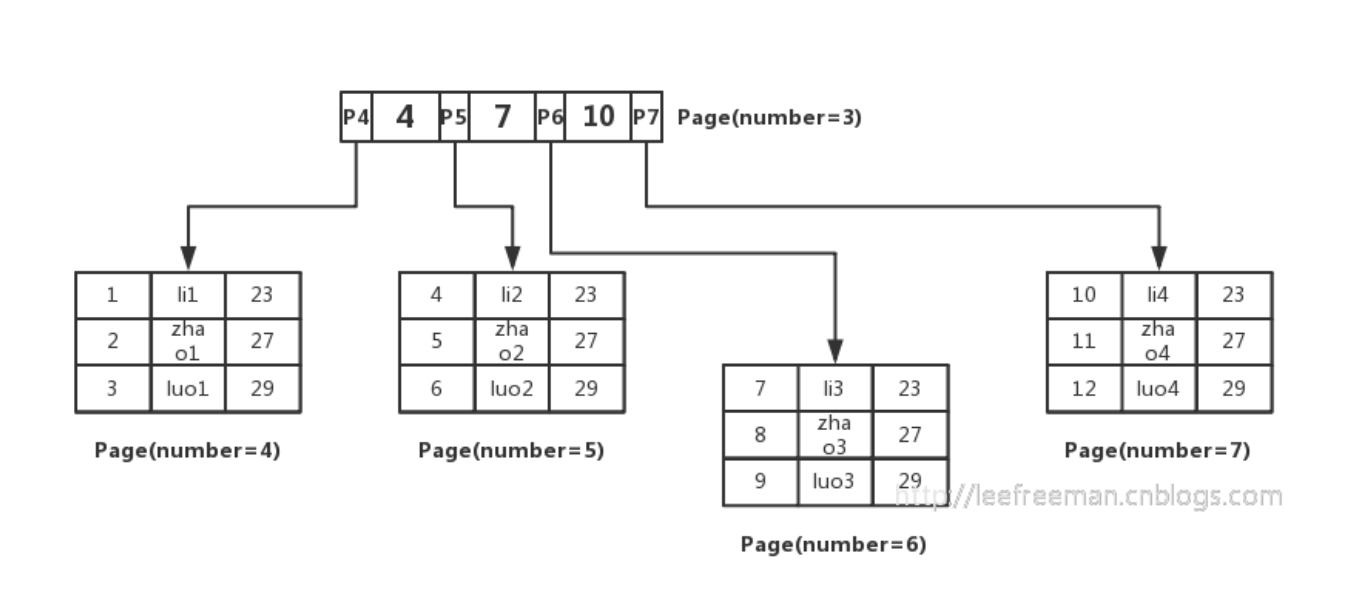
# 索引B+树

在计算机中磁盘存储数据最小单元是扇区，一个扇区的大小是512字节，而文件系统（例如XFS/EXT4）他的最小单元是块，一个块的大小是4k，而对于我们的InnoDB存储引擎也有自己的最小储存单元——页（Page），一个页的大小是16K。



数据表中的数据都是存储在页中的，所以一个页中能存储多少行数据呢？假设一行数据的大小是1k，那么一个页可以存放16行这样的数据。

如果数据库只按这样的方式存储，那么如何查找数据就成为一个问题，因为我们不知道要查找的数据存在哪个页中，也不可能把所有的页遍历一遍，那样太慢了。所以人们想了一个办法，用B+树的方式组织这些数据。如图所示：



我们先将数据记录按主键进行排序，分别存放在不同的页中（为了便于理解我们这里一个页中只存放3条记录，实际情况可以存放很多），除了存放数据的页以外，还有存放键值+指针的页，如图中page number=3的页，该页存放键值和指向数据页的指针，这样的页由N个键值+指针组成。当然它也是排好序的。这样的数据组织形式，我们称为索引组织表。现在来看下，要查找一条数据，怎么查？

如select \* from user where id=5;

这里id是主键,我们通过这棵B+树来查找，首先找到根页，你怎么知道user表的根页在哪呢？其实每张表的根页位置在表空间文件中是固定的，即page number=3的页（这点我们下文还会进一步证明），找到根页后通过二分查找法，定位到id=5的数据应该在指针P5指向的页中，那么进一步去page number=5的页中查找，同样通过二分查询法即可找到id=5的记录：



#### 一棵B+树可以存放多少行数据？

这里我们先假设B+树高为2，即存在一个根节点和若干个叶子节点，那么这棵B+树的存放总记录数为：根节点指针数\*单个叶子节点记录行数。

上文我们已经说明单个叶子节点（页）中的记录数=16K/1K=16。（这里假设一行记录的数据大小为1k，实际上现在很多互联网业务数据记录大小通常就是1K左右）。

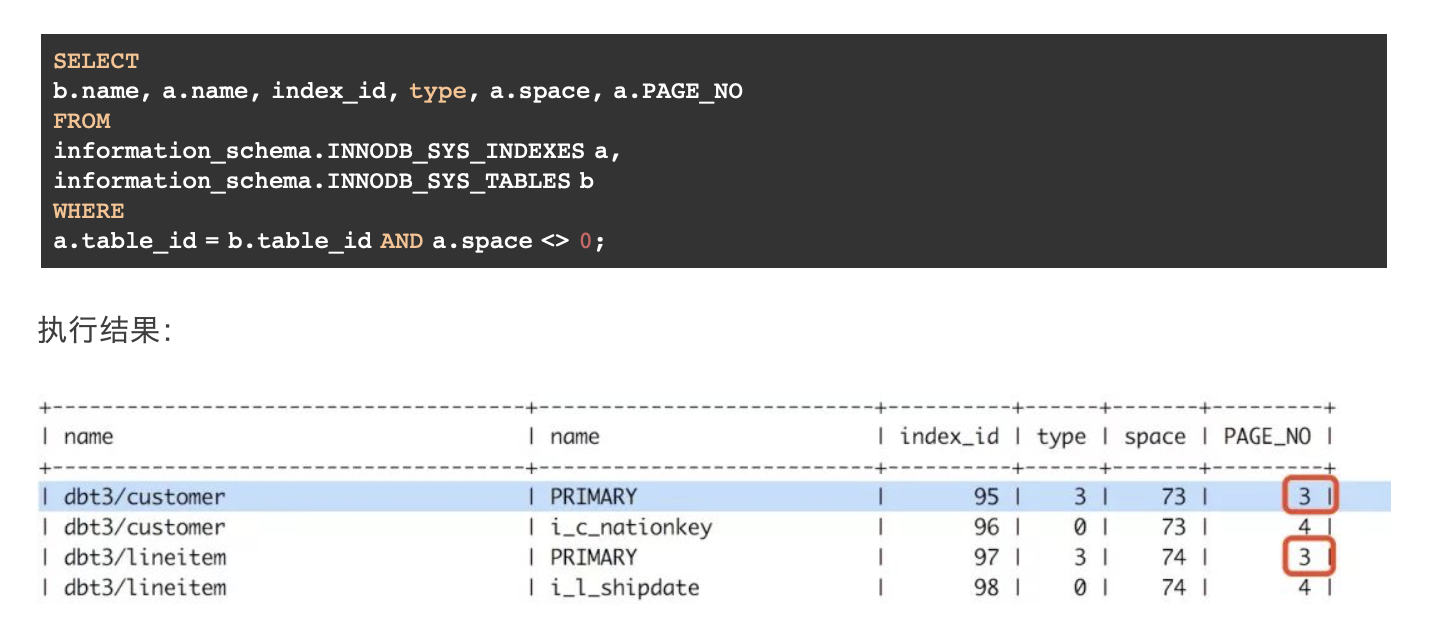
那么现在我们需要计算出非叶子节点能存放多少指针，其实这也很好算，我们假设主键ID为bigint类型，长度为8字节，而指针大小在InnoDB源码中设置为6字节，这样一共14字节，我们一个页中能存放多少这样的单元，其实就代表有多少指针，即16384/14=1170。那么可以算出一棵高度为2的B+树，能存放1170\*16=18720条这样的数据记录。

根据同样的原理我们可以算出一个高度为3的B+树可以存放：1170\*1170\*16=21902400条这样的记录。所以在InnoDB中B+树高度一般为1-3层，它就能满足千万级的数据存储。在查找数据时一次页的查找代表一次IO，所以通过主键索引查询通常只需要1-3次IO操作即可查找到数据。

### 怎么得到InnoDB主键索引B+树的高度？

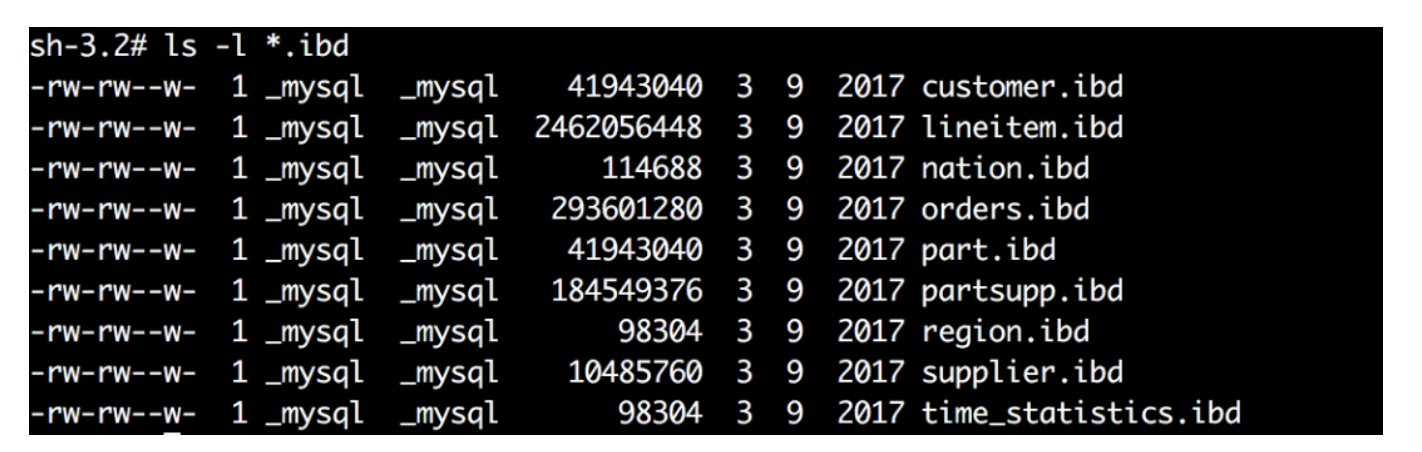
上面我们通过推断得出B+树的高度通常是1-3，下面我们从另外一个侧面证明这个结论。在InnoDB的表空间文件中，约定page number为3的代表主键索引的根页，而在根页偏移量为64的地方存放了该B+树的page level。如果page level为1，树高为2，page level为2，则树高为3。即B+树的高度=page level+1；下面我们将从实际环境中尝试找到这个page level。

在实际操作之前，你可以通过InnoDB元数据表确认主键索引根页的page number为3，你也可以从《InnoDB存储引擎》这本书中得到确认。



可以看出数据库dbt3下的customer表、lineitem表主键索引根页的page number均为3，而其他的二级索引page number为4。关于二级索引与主键索引的区别请参考MySQL相关书籍，本文不在此介绍。

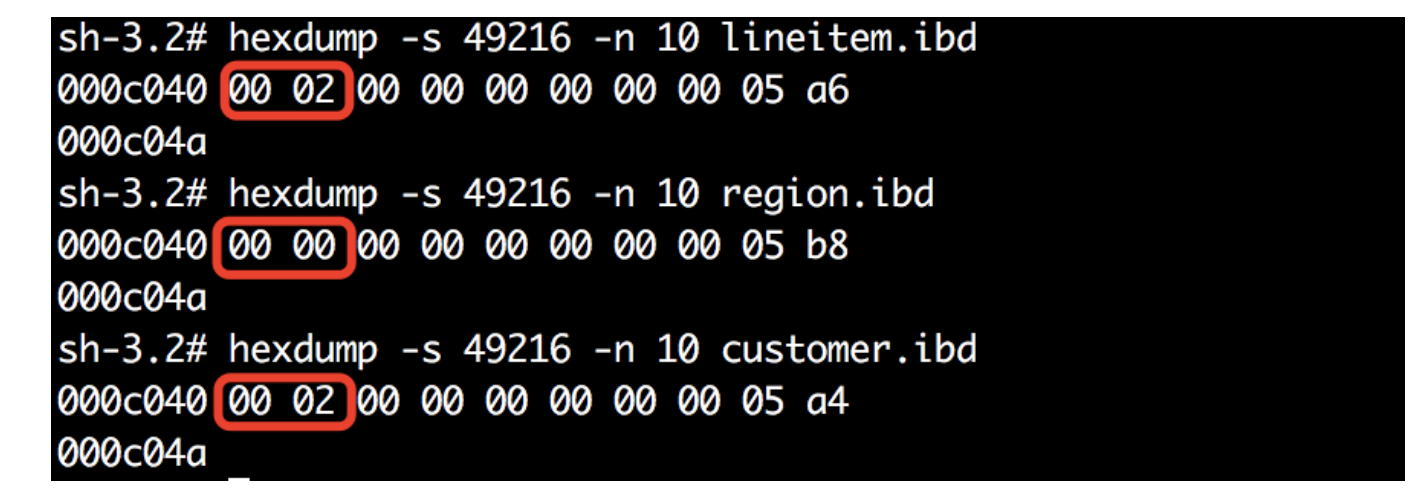
下面我们对数据库表空间文件做想相关的解析：



因为主键索引B+树的根页在整个表空间文件中的第3个页开始，所以可以算出它在文件中的偏移量：16384\*3=49152（16384为页大小）。

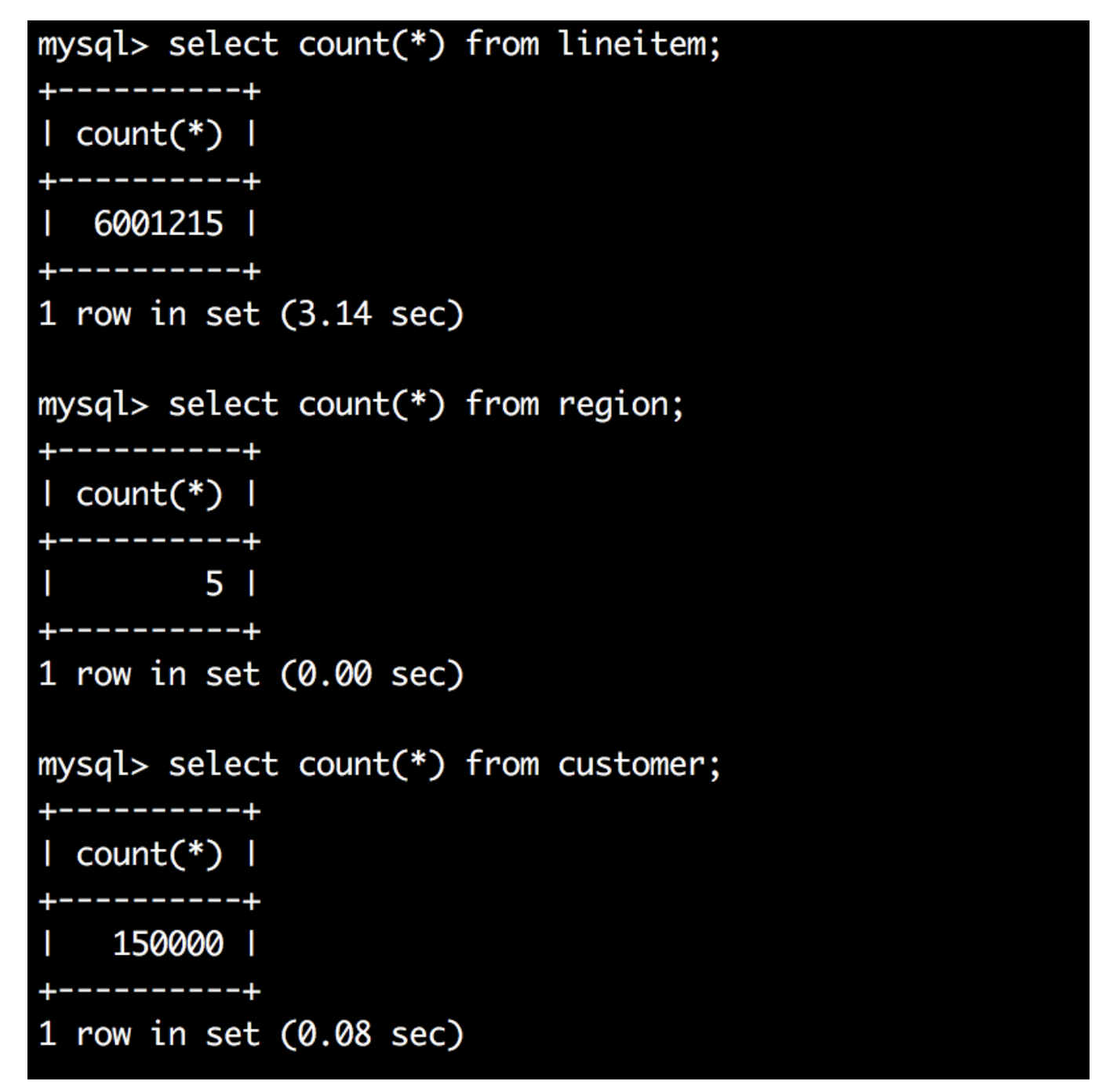
另外根据《InnoDB存储引擎》中描述在根页的64偏移量位置前2个字节，保存了page level的值，因此我们想要的page level的值在整个文件中的偏移量为：16384\*3+64=49152+64=49216，前2个字节中。

接下来我们用hexdump工具，查看表空间文件指定偏移量上的数据：



* linetem表的page level为2，B+树高度为page level+1=3；
* region表的page level为0，B+树高度为page level+1=1；
* customer表的page level为2，B+树高度为page level+1=3；

这三张表的数据量如下：

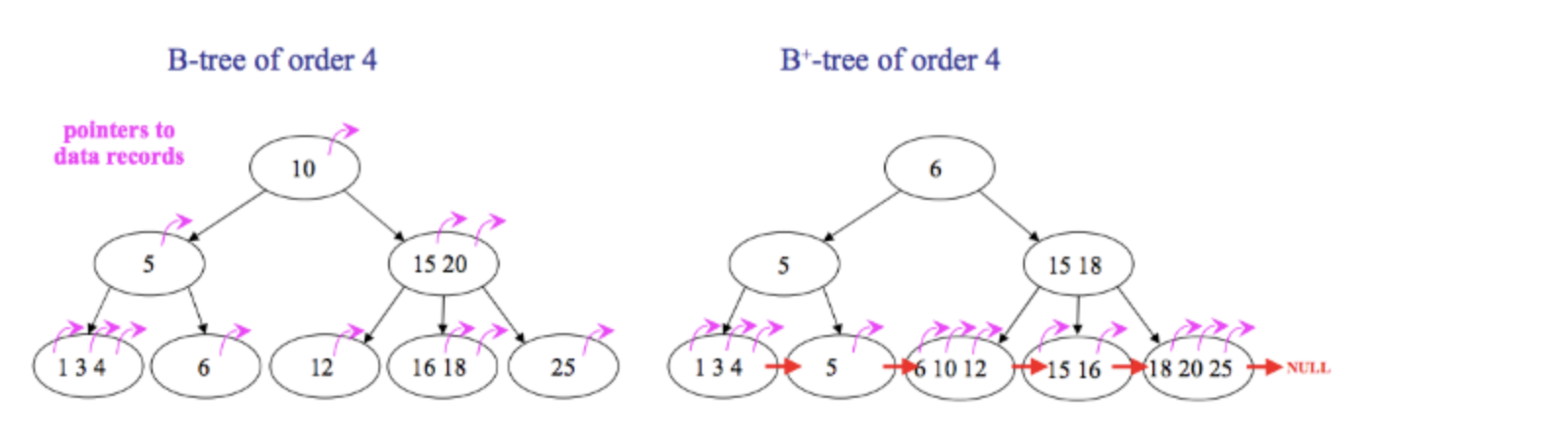


总结：

lineitem表的数据行数为600多万，B+树高度为3，customer表数据行数只有15万，B+树高度也为3。可以看出尽管数据量差异较大，这两个表树的高度都是3，换句话说这两个表通过索引查询效率并没有太大差异，因为都只需要做3次IO。那么如果有一张表行数是一千万，那么他的B+树高度依旧是3，查询效率仍然不会相差太大。

region表只有5行数据，当然他的B+树高度为1。

### B树和B+树的区别



我们假设被索引的列是主键，现在查找主键为5的记录，模拟一下查找的过程：

B树，在倒数第二层的节点中找到5后，可以立刻拿到指针获取行数据，查找停止。

B+树，在倒数第二层的节点中找到5后，由于中间节点不存有指针信息，则继续往下查找，在叶子节点中找到5，拿到指针获取行数据，查找停止。

B+树每个父节点的元素都会出现在子节点中，是子节点的最大（或最小）元素。叶子节点存储了被索引列的所有的数据。

那B+树比起B树有什么优点呢？

1. 由于中间节点不存指针，同样大小的磁盘页可以容纳更多的节点元素，树的高度就小。（数据量相同的情况下，B+树比B树更加“矮胖”），查找起来就更快。  
   2. B+树每次查找都必须到叶子节点才能获取数据，而B树不一定，B树可以在非叶子节点上获取数据。因此B+树查找的时间更稳定。  
   3. B+树的每一个叶子节点都有指向下一个叶子节点的指针，方便范围查询和全表查询：只需要从第一个叶子节点开始顺着指针一直扫描下去即可，而B树则要对树做[中序遍历](https://www.jianshu.com/p/456af5480cee" \t "_blank)。

注：

**myisam**和 **innodb**索引

<https://www.cnblogs.com/yuanrw/p/10225659.html>

1. MySQL默认采用的是MyISAM。
2. MyISAM不支持事务，而InnoDB支持。InnoDB的AUTOCOMMIT默认是打开的，即每条SQL语句会默认被封装成一个事务，自动提交，这样会影响速度，所以最好是把多条SQL语句显示放在begin和commit之间，组成一个事务去提交。
3. InnoDB支持数据行锁定，MyISAM不支持行锁定，只支持锁定整个表。即MyISAM同一个表上的读锁和写锁是互斥的，MyISAM并发读写时如果等待队列中既有读请求又有写请求，默认写请求的优先级高，即使读请求先到，所以MyISAM不适合于有大量查询和修改并存的情况，那样查询进程会长时间阻塞。因为MyISAM是锁表，所以某项读操作比较耗时会使其他写进程饿死。
4. InnoDB支持外键，MyISAM不支持。
5. InnoDB的主键范围更大，最大是MyISAM的2倍。
6. InnoDB不支持全文索引，而MyISAM支持。全文索引是指对char、varchar和text中的每个词（停用词除外）建立倒排序索引。MyISAM的全文索引其实没啥用，因为它不支持中文分词，必须由使用者分词后加入空格再写到数据表里，而且少于4个汉字的词会和停用词一样被忽略掉。
7. MyISAM支持GIS数据，InnoDB不支持。即MyISAM支持以下空间数据对象：Point,Line,Polygon,Surface等。
8. 没有where的count(\*)使用MyISAM要比InnoDB快得多。因为MyISAM内置了一个计数器，count(\*)时它直接从计数器中读，而InnoDB必须扫描全表。所以在InnoDB上执行count(\*)时一般要伴随where，且where中要包含主键以外的索引列。为什么这里特别强调“主键以外”？因为InnoDB中primary index是和raw data存放在一起的，而secondary index则是单独存放，然后有个指针指向primary key。所以只是count(\*)的话使用secondary index扫描更快，而primary key则主要在扫描索引同时要返回raw data时的作用较大。

Mysql 间隙锁

<https://www.cnblogs.com/crazylqy/p/7821481.html>