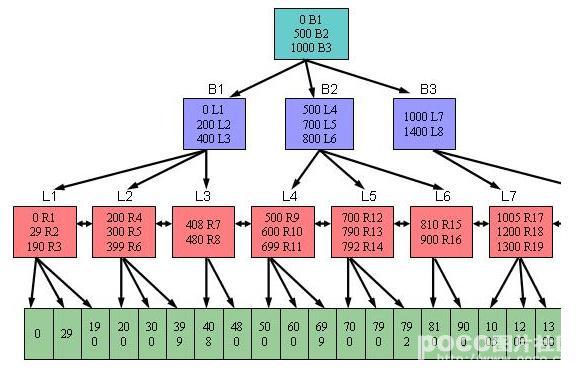
**Oracle中B-TREE索引的深入理解**

**索引概述**

索引与表一样，也属于段（segment）的一种。里面存放了用户的数据，跟表一样需要占用磁盘空间。只不过，在索引里的数据存放形式与表里的数据存放形式非常的不一样。在理解索引时，可以想象一本书，其中书的内容就相当于表里的数据，而书前面的目录就相当于该表的索引。同时，通常情况下，索引所占用的磁盘空间要比表要小的多，其主要作用是为了加快对数据的搜索速度，也可以用来保证数据的唯一性。  
但是，索引作为一种可选的数据结构，你可以选择为某个表里的创建索引，也可以不创建。这是因为一旦创建了索引，就意味着oracle对表进行DML（包括INSERT、UPDATE、DELETE）时，必须处理额外的工作量（也就是对索引结构的维护）以及存储方面的开销。所以创建索引时，需要考虑创建索引所带来的查询性能方面的提高，与引起的额外的开销相比，是否值得。  
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，由于我们的这篇文章主要就是对B树索引所做的探讨，因此下面只要说到索引，都是指B树索引。  
**B树索引内部结构**  
B树索引是一个典型的树结构，其包含的组件主要是：  
1) 叶子节点（Leaf node）：数据行的键值（key value）、键值对应数据行的 ROWID。  
2) 分支节点（Branch node）：最小的键值前缀（minimum key prefix），用于在（本块的）两个键值之间做出分支选择，指向包含所查找键值的子块（child block）的指针（）所有的 键值-ROWID 对（key and ROWID pair）都与其左右的兄弟节点（sibling）向链接（link），并按照（key，ROWID）的顺序排序  
3) 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的分支节点。

可以用下图一来描述B树索引的结构。其中，B表示分支节点，而L表示叶子节点。



对于分支节点块（包括根节点块）来说，其所包含的索引条目都是按照顺序排列的（缺省是升序排列，也可以在创建索引时指定为降序排列）。每个索引条目（也可以叫做每条记录）都具有两个字段。第一个字段表示当前该分支节点块下面所链接的索引块中所包含的最小键值；第二个字段为四个字节，表示所链接的索引块的地址，该地址指向下面一个索引块。在一个分支节点块中所能容纳的记录行数由数据块大小以及索引键值的长度决定。比如从上图一可以看到，对于根节点块来说，包含三条记录，分别为（0 B1）、（500 B2）、（1000 B3），它们指向三个分支节点块。其中的0、500和1000分别表示这三个分支节点块所链接的键值的最小值。而B1、B2和B3则表示所指向的三个分支节点块的地址。  
对于叶子节点块来说，其所包含的索引条目与分支节点一样，都是按照顺序排列的（缺省是升序排列，也可以在创建索引时指定为降序排列）。每个索引条目（也可以叫做每条记录）也具有两个字段。第一个字段表示索引的键值，对于单列索引来说是一个值；而对于多列索引来说则是多个值组合在一起的。第二个字段表示键值所对应的记录行的ROWID，该ROWID是记录行在表里的物理地址。如果索引是创建在非分区表上或者索引是分区表上的本地索引的话，则该ROWID占用6个字节；如果索引是创建在分区表上的全局索引的话，则该ROWID占用10个字节。

知道这些信息以后，我们可以举个例子来说明如何估算每个索引能够包含多少条目，以及对于表来说，所产生的索引大约多大。对于每个索引块来说，缺省的PCTFREE为10％，也就是说最多只能使用其中的90％。同时9i以后，这90％中也不可能用尽，只能使用其中的87％左右。也就是说，8KB的数据块中能够实际用来存放索引数据的空间大约为6488（8192×90％×88％）个字节。  
假设我们有一个非分区表，表名为warecountd，其数据行数为130万行。该表中有一个列，列名为goodid，其类型为char（8），那么也就是说该goodid的长度为固定值：8。同时在该列上创建了一个B树索引。  
在叶子节点中，每个索引条目都会在数据块中占一行空间。每一行用2到3个字节作为行头，行头用来存放标记以及锁定类型等信息。同时，在第一个表示索引的键值的字段中，每一个索引列都有1个字节表示数据长度，后面则是该列具体的值。那么对于本例来说，在叶子节点中的一行所包含的数据大致如下图二所示：



从上图可以看到，在本例的叶子节点中，一个索引条目占18个字节。同时我们知道8KB的数据块中真正可以用来存放索引条目的空间为6488字节，那么在本例中，一个数据块中大约可以放360（6488/18）个索引条目。而对于我们表中的130万条记录来说，则需要大约3611（1300000/360）个叶子节点块。

而对于分支节点里的一个条目（一行）来说，由于它只需保存所链接的其他索引块的地址即可，而不需要保存具体的数据行在哪里，因此它所占用的空间要比叶子节点要少。分支节点的一行中所存放的所链接的最小键值所需空间与上面所描述的叶子节点相同；而存放的索引块的地址只需要4个字节，比叶子节点中所存放的ROWID少了2个字节，少的这2个字节也就是ROWID中用来描述在数据块中的行号所需的空间。因此，本例中在分支节点中的一行所包含的数据大致如下图三所示：



从上图可以看到，在本例的分支节点中，一个索引条目占16个字节。根据上面叶子节点相同的方式，我们可以知道一个分支索引块可以存放大约405（6488/16）个索引条目。而对于我们所需要的3611个叶子节点来说，则总共需要大约9个分支索引块。

这样，我们就知道了我们的这个索引有2层，第一层为1个根节点，第二层为9个分支节点，而叶子节点数为3611个，所指向的表的行数为1300000行。但是要注意，在oracle的索引中，层级号是倒过来的，也就是说假设某个索引有N层，则根节点的层级号为N，而根节点下一层的分支节点的层级号为N-1，依此类推。对本例来说，9个分支节点所在的层级号为1，而根节点所在的层级号为2。  
注意：在Oracle中null被定义为无限大，且null不等于null，故在索引不会存有与null值对应的条目。如果不加其他限制条件的对表进行is null扫描，将会是全表扫描，如果是is not null扫描将会是全索引扫描

这里仅仅是作为研究来讨论如何估算，学习一样东西，我们当然要知其然，也知其所以然，实际环境中可以利用explain plan for 查看创建索引的执行计划，从而对索引大小，创建时间进行预判，具体可参见

http://czmmiao.iteye.com/blog/1471756

**B树索引的访问**

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：  
1） 随机访问，每次读取一个数据块（通过等待事件“db file sequential read”体现出来）。  
2） 顺序访问，每次读取多个数据块（通过等待事件“db file scattered read”体现出来）。  
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。  
事实上在B树索引虽然为一个树状的立体结构，但其对应到数据文件里的排列当然还是一个平面的形式，也就是像下面这样。  
/根/分支/分支/叶子/…/叶子/分支/叶子/叶子/…/叶子/分支/叶子/叶子/…/叶子/分支/.....  
因此，当oracle需要访问某个索引块的时候，势必会在这个结构上跳跃的移动。  
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。  
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。

**DML对B树索引的影响**

**INSERT**

在每个INSERT操作过程中，关键字必须被插入在正确叶节点的位置。如果叶节点已满，不能容纳更多的关键字，就必须将叶节点拆分。拆分的方法有两种：

1)如果新关键字值在所有旧叶节点块的所有关键字中是最大的，那么所有的关键字将按照99:1的比例进行拆分，使得在新的叶节点块中只存放有新关键字，而其他的所有关键字（包括所有删除的关键字）仍然保存在旧叶节点块中。  
2)如果新关键字值不是最大的，那么所有的关键字将按照50:50的比例进行拆分，这时每个叶节点块（旧与新）中将各包含原始叶节点中的一半关键字。  
这个拆分必须通过一个指向新叶节点的新入口向上传送到父节点。如果父节点已满，那么这个父节点也必须进行拆分，并且需要将这种拆分向上传送到父节点的父节点。这时，如果这个父节点也已满，将继续进行这个过程。这样，某个拆分可能最终被一直传送到根节点。如果根节点满了，根结点也将进行分裂。根结点在进行分裂的时候，就是树的高度增加的时候。根节点进行分裂的方式跟其他的的节点分裂的方式相比较，在物理位置上的处理也是不同的。根节点分裂时，将原来的根结点分裂为分支节点或叶节点，保存到新的块中，而将新的根节点信息保存到原来的根结点块中，这样做的是为因为避免修改数据字典所带来的相对较大的开销。  
注意：现在Oracle都是采用了平衡算法，正常情况下即使索引关键字不断增大，也不会产生不平衡树。当索引关键字不断增大，导致树级别单方向增长时，Oracle会自动进行索引翻转以维持索引的平衡，当然这种操作非常消耗资源  
在索引的每一个层次之间，每一个层最左边的节点的block头部都有一个指向下层最左边的块的指针，这样有利于fast full [SCAN[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://bfc99.blog.51cto.com/)](http://bfc99.blog.51cto.com/) 的快速定位最左边的叶子节点。  
每个拆分过程都是要花费一定的开销的，特别是要进行物理硬盘I/O动作。此外，在进行拆分之前，Oracle必须查找到一个空块，用来保存这个拆分。可以用以下步骤来进行查找空块的动作：  
1) 在索引的自由列表(free-list, 又称为空闲列表) 中查到一个空闲块，可以通过CREATE/ALTER INDEX命令为一个索引定义多个空闲列表。索引空闲列表并不能帮助Oracle查找一个可用来存放将要被插入的新关键字的块。这是因为关键字值不能随机地存放在索引中可用的第一个“空闲”叶节点块中，这个值必须经过适当的排序之后，放置在某个特定的叶节点块中。只有在块拆分过程中才需要使用索引的空闲列表，每个空闲列表都包含有一个关于“空”块的链接列表。当为某个索引定义了多个空闲列表时，首先将从分配给进程的空间列表中扫描一个空闲块。如果没有找到所需要的空闲块，将从主空闲列表中进行扫描空闲块的动作。  
2) 如果没有找到任何空闲块，Oracle将试图分配另一个扩展段。如果在表空间中没有更多的自由空间，Oracle将产生错误ORA-01654。  
3) 如果通过上述步骤，找到了所需的空闲块，那么这个索引的高水位标(HWM)将加大。  
4) 所找到的空闲块将用来执行拆分动作。  
在创建B\*树索引时，一个需要注意的问题就是要避免在运行时进行拆分，或者，要在索引创建过程中进行拆分（“预拆分”），从而使得在进行拆分时能够快速命中，以便避免运行时插入动作。当然，这些拆分也不仅仅局限于插入动作，在进行更新的过程中也有可能会发生拆分动作。  
**UPDATE**  
索引更新完全不同于表更新，在表更新中，数据是在数据块内部改变的（假设数据块中有足够的空间来允许进行这种改变）；但在索引更新中，如果有关键字发生改变，那么它在树中的位置也需要发生改变。请记住，一个关键字在B\*树中有且只有一个位置。因此，当某个关键字发生改变时，关键字的旧表项必须被删除，并且需要在一个新的叶节点上创建一个新的关键字。旧的表项有可能永远不会被重新使用，这是因为只有在非常特殊的情况下， Oracle才会重用关键字表项槽，例如，新插入的关键字正好是被删除的那个关键字（包括数据类型、长度等等）。（这里重用的是块，但完全插入相同的值的时候，也不一定插入在原来的被删除的位置，只是插入在原来的块中，可能是该块中的一个新位置。也正因为如此，在索引块中保存的的记录可能并不是根据关键字顺序排列的，随着update等的操作，会发生变化。）那么，这种情况发生的可能性有多大呢？许多应用程序使用一个数列来产生NUMBER关键字（特别是主关键字）。除非它们使用了RECYCLE选项，否则这个数列将不会两次产生完全相同的数。这样，索引中被删除的空间一直没有被使用。这就是在大规模删除与更新过程中，表大小不断减小或至少保持不变但索引不断加大的原因。

**DELETE**

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。  
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：  
1、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。  
2、索引键值持续增加（比如采用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。因为前面我们知道，某个索引块中删除了部分的索引条目，只有当有键值进入该索引块时才能将空间收回。而持续增加的索引键值永远只会向插入排在前面的索引块中，因此这种索引里的空间几乎不能收回，而只有其所含的索引条目全部删除时，该索引块才能被重新利用。  
3、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。

**总结**  
通过上面对B树的分析，可以得出以下的应用准则：  
1、避免对那些可能会产生很高的更新动作的列进行索引。  
2、避免对那些经常会被删除的表中的多个列进行索引。若有可能，只对那些在这样的表上会进行删除的主关键字与/或列进行索引。如果对多个列进行索引是不可避免的，那么就应该考虑根据这些列对表进行划分，然后在每个这样的划分上执行TRUNCATE动作（而不是DELETE动作）。TRUNCATE在与DROP STORAGE短语一同使用时，通过重新设置高水位标来模拟删除表与索引以及重新创建表与索引的过程。  
3、避免为那些唯一度不高的列创建B\*树索引。这样的低选择性将会导致树节点块的稠密性，从而导致由于索引“平铺( flat)”而出现的大规模索引扫描。唯一性的程度越高，性能就越好，因为这样能够减少范围扫描，甚至可能用唯一扫描来取代范围扫描。  
4）空值不存储在单列索引中。对于复合索引的方式，只有当某个列不空时，才需要进行值的存储。在为DML语句创建IS NULL或IS NOT NULL短语时，应该切记这个问题。  
5）IS NULL不会导致索引扫描，而一个没有带任何限制的IS NOT NULL则可能会导致完全索引扫描。