alter index supplier\_pk rebuild;

alter index supplier\_pk coalesce;

REBUILD statement is used to reorganize or compact an existing index or to change its storage characteristics. The REBUILD statement uses the existing index as the source for the new index. As a result, the index will consume twice as much space during the rebuild process

COALESCE statement instructs Oracle to merge the contents of index blocks where possible to free blocks for reuse. The index structure itself is not affected.

Rebuild takes exclusive table locks, whereas a coalesce is a purely online operation.  
 The rebuild causes a completely new index segment to be constructed from scratch.  
 Coalesce simply shuffles index entries around within the existing index segment.  
 The practical significance of that difference is that a rebuild requires potentially two lots of space for it to work.  
 Coalesce requires little or no extra space.  
 Coalesce leaves the overall size of your index unchanged.  
 It will shuffle index entries around between nodes so that everything gets compacted, but the index structure will not change in overall size: therefore, you will end up with empty blocks included in the finished structure.

They can be reused by new insertions and updates, of course, but still if your aim is to actually reduce the size of your index, the coalesce won't achieve it.

==========================================

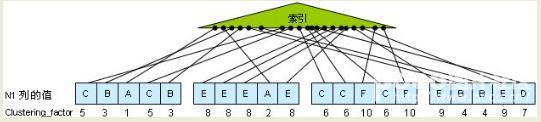
[**深入理解重建索引(原创)**](http://czmmiao.iteye.com/blog/1481957)

* **博客分类：**
* [Oracle体系架构](http://czmmiao.iteye.com/category/143940)

**什么时候需要重建索引**

索引在普遍意义上能够给数据库带来带来提升，但索引的额外开销也是不容小视的，而索引的重建也是维护索引的重要工作之一。 经过维护的索引可带来以下好处：  
 1、CBO对于索引的使用可能会产生一个较小的成本值，从而在执行计划中选择使用索引。  
 2、使用索引扫描的查询扫描的物理索引块会减少，从而提高效率。  
 3、于需要缓存的索引块减少了，从而让出了内存以供其他组件使用。  
重建索引的原因主要包括：  
 1、 删除的空间没有重用，导致 索引出现碎片  
 2、 删除大量的表数据后，空间没有重用，导致 索引"虚高"  
 3、索引的 clustering\_facto 和表不一致  
 也有人认为当索引树高度超过4的时候需要进行重建，但是如果表数量级较大，自然就不会有较高的树，而且重建不会改变索引树高度，除非是由于大量引起的索引树“虚高”，重建才会改善性能，当然这又回到了索引碎片的问题上了。  
索引出现碎片

由于索引中只有删除和插入操作，且索引中更新完全不同于表达更新。如果索引中的 记录关键字需要更新，就需要将旧记录的位置标记为删除，并在相应的叶子节点插入新的索引纪录。这种删除标记并非真正的删除索引块中的记录，索引块中被标记 为删除的记录只有在相同索引条目插入到相同块的相同位置时才能重用。由于即使相同的索引记录也不一定插入到被删除的空间中，故如果对索引频繁进行 update和delete操作很容易导致索引出现碎片。较高的PCTFREE也容易出现索引碎片。索引的碎片也就导致了，访问索引数据时需要访问更多的 索引块  
索引虚高  
 上面的说的是频繁update和delete导致索引块中有碎片，那如果进行大量的delete操作把整个索引块的数据都删了呢？索引中的索引条目仍然只 被标记为删除而没有被真正清空。设想下，如果这时候的索引关键字是一个不断增大的id，那么被标记为删除的索引条目就永远不会被重用，那树就不会不断增 长，也就出现了，表的数据空间减少了，而索引的数据空间却在不断增大的情况。由于索引的高度不断增加，访问索引数据时需要访问更多的索引块。  
 **clustering\_factor对** **B树索引** **的影响**   
对于clustering\_factor来说，它是用来比较索引的顺序程度与表的杂乱排序程度的一个度量。Oracle在计算某个 clustering\_factor时，会对每个索引键值查找对应到表的数据，在查找的过程中，会跟踪从一个表的数据块跳转到另外一个数据块的次数（当 然，它不可能真的这么做，源代码里只是简单的扫描索引，从而获得ROWID，然后从这些ROWID获得表的数据块的地址）。每一次跳转时，有个计数器就会 增加，最终该计数器的值就是clustering\_factor。下图描述了这个原理。



在上图中，我们有一个表，该表有4个数据块，以及20条记录。在列N1上有一个索引，上图中的每个小黑点就表示一个索引条目。列N1的值如图所示。而N1 的索引的叶子节点包含的值为：A、B、C、D、E、F。如果oracle开始扫描索引的底部，叶子节点包含的第一个N1值为A，那么根据该值可以知道对应 的ROWID位于第一个数据块的第三行里，所以我们的计数器增加1。同时，A值还对应第二个数据块的第四行，由于跳转到了不同的数据块上，所以计数器再加 1。同样的，在处理B时，可以知道对应第一个数据块的第二行，由于我们从第二个数据块跳转到了第一个数据块，所以计数器再加1。同时，B值还对应了第一个 数据块的第五行，由于我们这里没有发生跳转，所以计数器不用加1。  
 在上面的图里，在表的每一行的下面都放了一个数字，它用来显示计数器跳转到该行时对应的值。当我们处理完索引的最后一个值时，我们在数据块上一共跳转了十次，所以该索引的clustering\_factor为10。  
 注意第二个数据块，clustering\_factor为8出现了4次。因为在索引里N1为E所对应的4个索引条目都指向了同一个数据块。从而使得 clustering\_factor不再增长。同样的现象出现在第三个数据块中，它包含三条记录，它们的值都是C，对应的 clustering\_factor都是6。  
 从clustering\_factor的计算方法上可以看出，我们可以知道它的最小值就等于表所含有的数据块的数量；而最大值就是表所含有的记录的总行 数。很明显，clustering\_factor越小越好，越小说明通过索引查找表里的数据行时需要访问的表的数据块越少。  
 所以我们可以得出结论，如果仅仅是为了降低索引的clustering\_factor而重建索引没有任何意义。降低clustering\_factor的关键在于重建表里的数据。事实上，生产环境下，我们甚至没有必要考虑 clustering\_factor对索引访问的影响，这个是表数据分布决定的，如果想考虑，就得先创建索引，然后分析 clustering\_factor，最后对表进行排序，再重新创建索引，可行性非常低。因此，这里只是作为研究讨论，实际环境下还是要结合具体情况进行分析。针对索引碎片和索引的"虚高"，如果查询范围主要是通过unique index访问数据，可以不用理会 索引碎片和索引的"虚高"，如果数据范围，主要是通过range scan的方式则需要重建索引，至于原理，相信读了笔者下面的文章后肯定会明白  
 <http://czmmiao.iteye.com/blog/1481227> 。关于索引是否需要重建，Oracle有这么一句话  
 Generally speaking, the need to rebuild b-tree indexes is very rare, basically because a b-tree index is largely self-managed or self-balanced.

**如何查找出需要重建的索引** 我们通过下面实验来具体看下如何查找需要重建的索引  
 准备实验环境如下   
SQL> create table ind (id int,name varchar2(100));  
 Table created.

SQL> begin  
   2  for i in 1..10000 loop  
   3  insert into ind values(i,to\_char(i)||'aaa');  
   4  end loop;  
   5  commit;  
   6  end;  
   7  /  
 PL/SQL procedure successfully completed.  
 SQL> create index ind\_id\_idx on ind(id);  
 Index created.

SQL> analyze index ind\_id\_idx validate structure;  
 Index analyzed.

注意:index\_stats只能在同一个 session里先执行完analyze index indexname validate structure后才能查到数据，在其他的session里查index\_stats是查不到数据的，即使那个初始的session已经执行过 analyze index indexname validate structure。顺带提一句， analyze index indexname validate structure会对整张表加排他锁，阻止表上的所有DML语句。 我们也可以使用online关键字，analyze index indexname validate structure online，这样就可以不对表加锁，但不会填充index\_stats视图。  
index\_stats的主要相关字段如下

--LF\_ROWS Number of values currently in the index  
 --LF\_ROWS\_LEN Sum in bytes of the length of all values  
 --DEL\_LF\_ROWS Number of values deleted from the index  
 --DEL\_LF\_ROWS\_LEN Length of all deleted values  
 col name         heading 'Index Name'          format a30  
 col del\_lf\_rows  heading 'Deleted|Leaf Rows'   format 99999999  
 col lf\_rows\_used heading 'Used|Leaf Rows'      format 99999999  
 col ibadness     heading '% Deleted|Leaf Rows' format 999.99999  
 SQL> SELECT name,  
 2       del\_lf\_rows,  
 3      lf\_rows - del\_lf\_rows lf\_rows\_used,  
 4      to\_char(del\_lf\_rows / (lf\_rows)\*100,'999.99999') ibadness  
 5    FROM index\_stats  
 6       where name = upper('&&index\_name');  
                                  Deleted      Used % Deleted  
 Index Name                     Leaf Rows Leaf Rows Leaf Rows  
 ------------------------------ --------- --------- ------------------------------  
 IND\_ID\_IDX                             0     10000     .00000  
可以看到没有删除的索引  
 更新1000条记录  
 SQL> update ind set id=id+1 where id> 9000;  
 1000 rows updated.  
 SQL> commit;  
 Commit complete.  
 SQL> analyze index ind\_id\_idx validate structure;  
 Index analyzed.  
 SQL> col name         heading 'Index Name'          format a30  
     col del\_lf\_rows  heading 'Deleted|Leaf Rows'   format 99999999  
     col lf\_rows\_used heading 'Used|Leaf Rows'      format 99999999  
     col ibadness     heading '% Deleted|Leaf Rows' format 999.99999  
     SELECT name,  
        del\_lf\_rows,  
        lf\_rows - del\_lf\_rows lf\_rows\_used,  
        to\_char(del\_lf\_rows / (lf\_rows)\*100,'999.99999') ibadness  
     FROM index\_stats  
        where name = upper('&&index\_name');SQL> SQL> SQL> SQL> SQL>   2    3    4    5    6   
 old   6:        where name = upper('&&index\_name')  
 new   6:        where name = upper('ind\_id\_idx')  
                                  Deleted      Used % Deleted  
 Index Name                     Leaf Rows Leaf Rows Leaf Rows  
 ------------------------------ --------- --------- ------------------------------  
 IND\_ID\_IDX                          1000     10000    9.09091  
删除的索引占了9.09%，如果删除的索引条目占了10~15%，则可以考虑重建索引或者coalesce  
如何重建索引  
 重建索引有3种方法，具体如下:   
1、删除重新建索引可以采用 PARALLEL， NOLOGGING和 COMPUTE STATISTICS 进行处理，该方法是最慢的，最耗时的。一般不建议。  
 2、实验alter index .........rebuild命令重建  
 它使用原索引的叶子节点作为新索引的数据来源。我们知道，原索引的叶子节点的数据块通常都要比表里的数据块要少很多，因此进行的I/O就会减少；同时，由 于原索引的叶子节点里的索引条目已经排序了，因此在重建索引的过程中，所做的排序工作也要少的多。从oracle 8.1.6以后，ALTER INDEX … REBUILD命令可以添加ONLINE关键字。这使得在重建索引的过程中，用户可以继续对原来的索引进行修改，也就是说可以继续对表进行DML操作和删 除，但在11g之前，在开始和结束创建索引的时刻仍然会锁表。 由于新旧索引在建立时同时存在，因此，使用这种技巧则需要有额外的磁盘空间可临时使用，当索引建完后把老索引删除，如果没有成功，也不会影响原来的索引。利用这种办法，指定了tablespace关键字后 Alter index indexname rebuild  tablespace tablespacename 还可以用来将一个索引以到新的表空间。和 重建索引一样， alter index indexname rebuild 也可以采用 PARALLEL， NOLOGGING和 COMPUTE STATISTICS 进行处理 ，使用 COMPUTE STATISTICS处理的好处在于 可以在重建索引的过程中，就生成CBO所需要的统计信息，这样就避免了索引创建完毕以后再次运行analyze或dbms\_stats来收集统计信息。   
这个命令的执行步骤如下：  
 首先，逐一读取现有索引，以获取索引的关键字。  
 其次，按新的结构填写临时数据段。  
 最后，一旦操作成功，删除原有索引树，降临时数据段重命名为新的索引。  
 需要注意的是alter index indexname rebuild 命令中必须使用tablespace字句，以保证重建工作是在现有索引相同的表空间进行。

3、使用alter index indexname coalesce命令或alter index indexname shrik space命令重建索引。该命令主要是 用来合并相邻的碎片，相比于rebuild， 有如下优点 ：  
 1、always online，不需要锁索引  
 2、不需要消耗接近两倍的临时空间  
 3、当碎片比率小于25%时比rebuild产生更少的redo日志，当然我们可以在重建索引时将日志关闭。  
 4、 并不重建索引，只对叶子节点进行整合，不改变索引高度和分支节点数量。当我们重建索引 后，索引之间是紧密连接的，高度和分支数量都会改变。如果该索引列上继续进行DML操作，很可能导致树的重新增长、分裂，这是非常消耗资源的操作，同时由 于DML操作的继续，达到一定程度后，该索引很可能重新被认为需要rebuild，如此恶性循环。所以往往我们不建议重建索引，不希望重建索引而引起索引 高度和分支节点的重构，除非在需要迁移索引到另一个表空间时才选择rebuild。  
Insert/update/delete causes the index to evolve over time as the index splits and grows. When the index is rebuild it will become more tightly packed; however as DML operations continue on the table the index splits have to be redone again until the index reaches it's equilibrium. As a result, the redo activity increases and the index splits are now more likely to impact performance directly as we consume more I/O, CPU, etc to serve the index restructuring. After a certain period of time the index may again experience 'issues' and may be re-flagged for a rebuild, causing the vicious cycle to continue. Therefore it is often better to leave the index in its natural equilibrium and/or at least prevent indexes from being rebuilt on a regular basis.   
5、10g以后引入了 alter index indexname shrik space命令，功能上 和alter index indexname coalesce一样，但经测试产生更多的redo日志(以实际测试环境为准)。  
这边给出如下结论，帮助大家整理下 COALESCE、SHRINK和rebuild的区别 ：  
 1、当索引中碎片率<=25%，COALESCE与SHRINK比rebuild的产生的redo日志少，消耗资源更少。两者相比之下SHRINK的成本会更高。  
 2、当 索引中碎片率> 25%的时，REBUILD的成本更小，产生的redo更少  
 对测试过程感兴趣的朋友可以参见链接  
 <http://www.shujukuai.com/?p=102> <http://www.oracledatabase12g.com/archives/alter-index-coalesce-vs-shrink-space.html>

参考至:<https://support.oracle.com/CSP/main/article?cmd=show&type=NOT&id=989093.1>

           <http://space.itpub.net/?uid-9842-action-viewspace-itemid-324587>            <http://ustcer.blog.51cto.com/1135926/258625>            <http://www.itpub.net/thread-181890-1-1.html>            <http://www.shujukuai.com/?p=102>

<http://www.oracledatabase12g.com/archives/alter-index-coalesce-vs-shrink-space.html>