

Chapter 3

Oracle Index Basics

This article explores the basic architecture of the Oracle B-tree index. An index is an hierarchically organized structure designed to facilitate the rapid lookup of row identifiers (rowids) based on key values. Keys may or may not be unique. Keys may consist of more than one column – a “compound” key.

Branch, leaf and root blocks

Indices are structured as a collection of two types of blocks – leaf blocks and branch blocks. Leaf blocks contain key/rowid pairs which refer to specific rows in the table. Above the leaf blocks are branch blocks which contain key/dba pairs referring to other index blocks. One particular block, the “root” block, is the beginning or root of the index structure. If the index contains few enough keys, the root block may be a leaf block. As the index grows, the root block becomes a branch block.

Our Example

To facilitate the discussion, our example is an index created on a table of patients.

```
SVRMGR> create table pat (ssn varchar2(11), name varchar2(800));
Statement processed.
SVRMGR> create index pati on pat(ssn, name);
Statement processed.
```

The patient table, 'PAT', consists of a social security number of the form '555-55-5555', and a name. An index, 'PATI', is created as a compound key with a large key value (name) to readily view the index structure with a relatively small number of keys.

To populate the table (and index) with 40 random social security numbers we use the Oracle supplied random number package and a helper view.

```
SVRMGR> -- Create view of random social security numbers - format is '123-45-6789'
SVRMGR> create or replace view randssn as
  2>   select to_char(mod(abs(sys.dbms_random.random), 1000), 'FM000' ) || '-' ||
  3>          to_char(mod(abs(sys.dbms_random.random), 100), 'FM00' ) || '-' ||
  4>          to_char(mod(abs(sys.dbms_random.random), 10000), 'FM0000') ssn
  5>   from all_objects
  6> ;
Statement processed.
SVRMGR> -- Initialize the generator
SVRMGR> execute sys.dbms_random.initialize(10);
Statement processed.
SVRMGR> -- Select a few samples
SVRMGR> select ssn from randssn where rownum < 10;
SSN
-----
692-67-3549
564-94-9324
601-12-4079
476-77-8825
874-48-2039
254-28-9002
130-34-6459
112-09-2994
353-71-2580
9 rows selected.
```

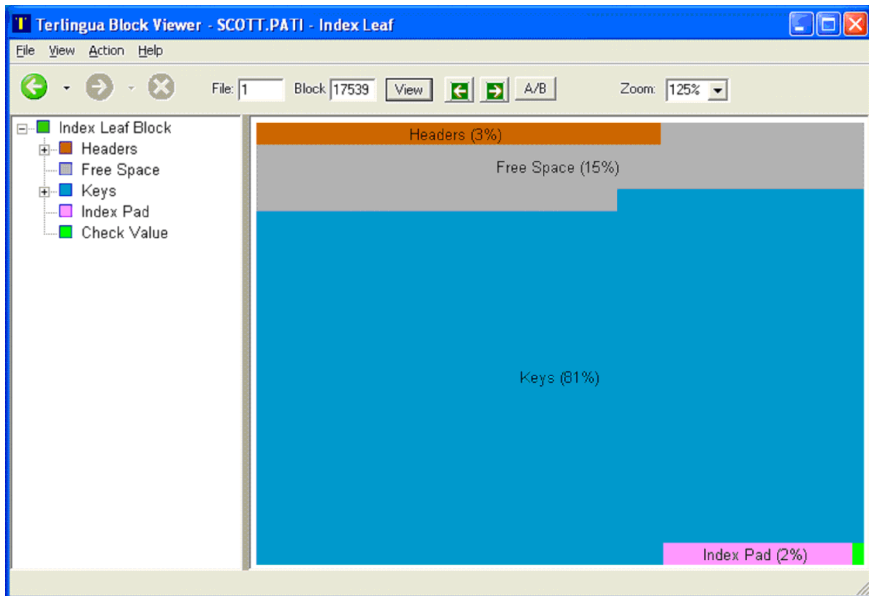
Executing the following query from a meta view, that comes with our product, reveals the overall view of the index structure.

```
SQL> select min(indexlevel)      indexlevel
       , count(*)                nkeys
       , mydbadot                dba
       , rpad(min(char01), 11)  minkey
       , rpad(max(char01), 11)  maxkey
  from index_keys
  where owner                    = 'SCOTT'
     and name                    = 'PATI'
  group by mydbadot
  order by 1 desc, 4
 11 ;
```

INDEXLEVEL	NKEYS	DBA	MINKEY	MAXKEY
1	15	1.16496	1	9
0	2	1.16497	081-08-7388	084-73-5622
0	4	1.17539	110-96-8149	184-45-1682
0	1	7.4938	193-72-7052	193-72-7052
0	4	1.17538	219-69-5713	233-02-2989
0	1	7.4942	258-72-1800	258-72-1800
0	1	7.4940	278-87-0378	278-87-0378
0	2	1.17540	281-77-2553	289-39-5390
0	4	1.17536	357-59-5655	363-27-3772
0	4	7.4103	436-68-8596	479-19-4054
0	1	7.4941	482-73-1638	482-73-1638
0	3	7.4102	549-38-2766	580-86-3383
0	3	7.4937	629-66-0887	699-68-6369
0	4	7.4101	723-91-9173	784-73-5471
0	1	7.4943	798-80-2422	798-80-2422
0	3	1.17537	819-45-0329	876-76-8588
0	2	7.4939	949-46-3452	959-96-9589

17 rows selected.

Querying the index blocks directly reveals the number of keys, and the minimum and maximum key value for each of the leaf blocks. The root block contains 15 keys, which point to 15 of the 16 leaf blocks. The first leaf block (1.16497) is referred to as the 'left most child' and is treated separately in the root index block.



On the left is a graphical look at one of the leaf blocks (1.17539) and reveals the headers growing from top down and the keys growing from bottom up consuming the common free space.

Let us drill down into the headers (screen shot on top of page 23).

The Block Header, Data Block Header, and Interesting Transaction Table are all discussed to some degree in "Row Locking in Oracle" and will not be discussed further

here. The Offsets section contains offsets into the index block where each of the keys reside. The index header, common to both index leaf and index branch blocks, is shown on page 23.

The field titled "Level" with a value of 0 indicates that this is a leaf block and the "Number of keys" field confirms this block contains four keys. Popping up a couple of view levels and drilling down reveals an individual key (bottom of page 23).

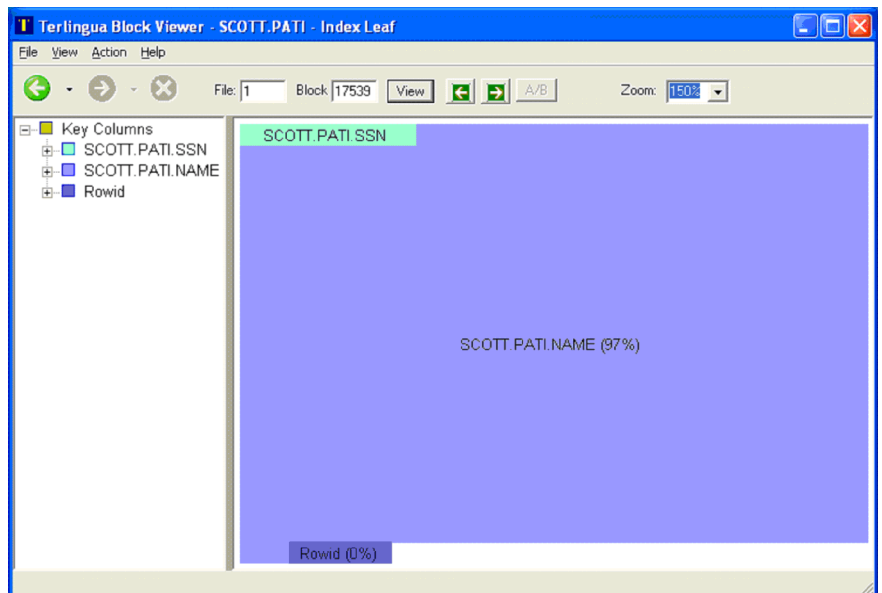
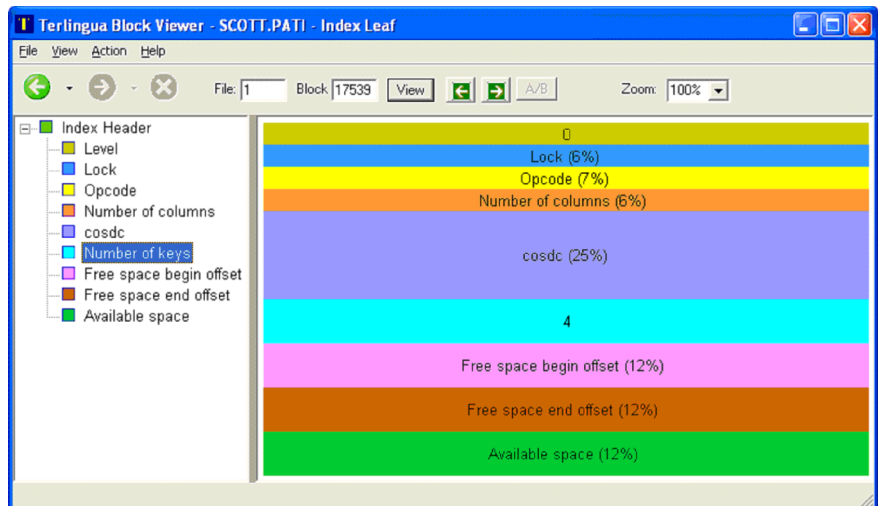
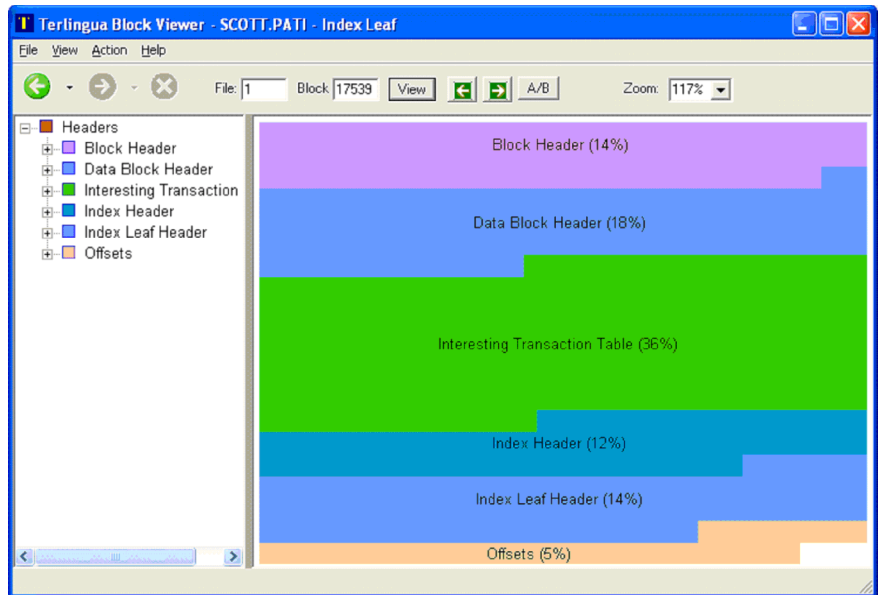
Note how Oracle appends the rowid of the row containing the key to the end of our compound key SSN.NAME. It does this to non-unique keys to insure unique keys in the index.

Leaf Block Splitting

With a solid understanding of the basic layout of index leaf blocks, we can discuss what database administrators try to avoid during normal operation – leaf block splitting. Leaf block splitting occurs when an inserted key cannot fit into the designated index leaf block. At this time, the leaf block is split in two with some of the keys remaining in the current block and some going into a new block. In order to accomplish this task, many data blocks will be modified:

1. First a new block is needed – At best this implies modification to the segment header to remove a block from the free list. At worst we may need to bump the high water mark (6 blocks touched) and perhaps allocate a new extent (many blocks touched).
2. Populate our new block with some of the old keys plus our new one.
3. Remove those same keys from the old block.
4. Modify the previous block field of the next block of the index to point back to our new block.
5. And finally, update the branch block to inform it of the new division key and block.

A minimum of 5 blocks are modified. Each of these modifications are recorded in a rollback segment as well. On top of all this, Oracle sets a lock in the leaf block to be split before starting the split. This prevents some other transaction from concurrently splitting the same block. This is a costly operation to be avoided during operational hours.



The health of the index is confirmed by querying the index blocks directly to reveal the number of keys they contain.

```
SQL> select min(indexlevel)      indexlevel
2      , count(*)                nkeys
3      , mydbadot                dba
4      , rpad(min(char01), 11)  minkey
5      , rpad(max(char01), 11)  maxkey
6  from index_keys
7  where owner                   = 'SCOTT'
8  and name                      = 'PATI'
9  group by mydbadot
10 order by 1 desc, 4
11 ;
```

INDEXLEVEL	NKEYS	DBA	MINKEY	MAXKEY
1	15	1.16496	1	9
0	2	1.16497	081-08-7388	084-73-5622
0	4	1.17539	110-96-8149	184-45-1682
0	1	7.4938	193-72-7052	193-72-7052
0	4	1.17538	219-69-5713	233-02-2989
0	1	7.4942	258-72-1800	258-72-1800
0	1	7.4940	278-87-0378	278-87-0378
0	2	1.17540	281-77-2553	289-39-5390
0	4	1.17536	357-59-5655	363-27-3772
0	4	7.4103	436-68-8596	479-19-4054
0	1	7.4941	482-73-1638	482-73-1638
0	3	7.4102	549-38-2766	580-86-3383
0	3	7.4937	629-66-0887	699-68-6369
0	4	7.4101	723-91-9173	784-73-5471
0	1	7.4943	798-80-2422	798-80-2422
0	3	1.17537	819-45-0329	876-76-8588
0	2	7.4939	949-46-3452	959-96-9589

17 rows selected.

Note the number of leaf blocks containing four keys. These blocks indicate the key ranges that cannot receive another key of similar size without causing an index split. The following helper view and query reveals the extent of the problem.

```
SQL> create or replace view helper as
2  select min(indexlevel)      indexlevel
3  , count(*)                  nkeys
4  , mydbadot                  dba
5  , rpad(min(char01), 11)    minkey
6  , rpad(max(char01), 11)    maxkey
7  from index_keys
8  where owner                 = 'SCOTT'
9  and name                    = 'PATI'
10 group by mydbadot
11 ;
```

View created.

```
SQL> select nkeys
2      , count(*)
3      , rpad(min(minkey), 11) samplemin
4      , rpad(min(maxkey), 11) samplemax
5  from helper
6  group by nkeys
7  order by 1
8  ;
```

NKEYS	COUNT(*)	SAMPLEMIN	SAMPLEMAX
1	5	193-72-7052	193-72-7052
2	3	081-08-7388	084-73-5622
3	3	549-38-2766	580-86-3383
4	5	110-96-8149	184-45-1682
15	1	1	9

Five of our index leaf blocks, or approximately 1/3 of the total index leaf blocks, already record the maximum number of 4 keys and cannot receive another key of similar size. A novice administrator might try to drop and recreate the index. This can only help, right?

```
SQL> drop index pati;
Index dropped.
SQL> create index pati on pat(ssn, name);
Index created.
SQL> select nkeys
2      , count(*)
3      , rpad(min(minkey), 11) samplemin
4      , rpad(min(maxkey), 11) samplemax
5 from helper
6 group by nkeys
7 order by 1
8 ;
```

NKEYS	COUNT(*)	SAMPLEMIN	SAMPLEMAX
4	10	081-08-7388	144-15-7241
9	1	15	83

We have made the problem worse! By creating the index anew, we have shrunk the index from 17 blocks to 11, but have guaranteed our index will cause a leaf block split no matter what key is inserted. The solution is to use the storage parameter `pct_free` during the recreation of the index.

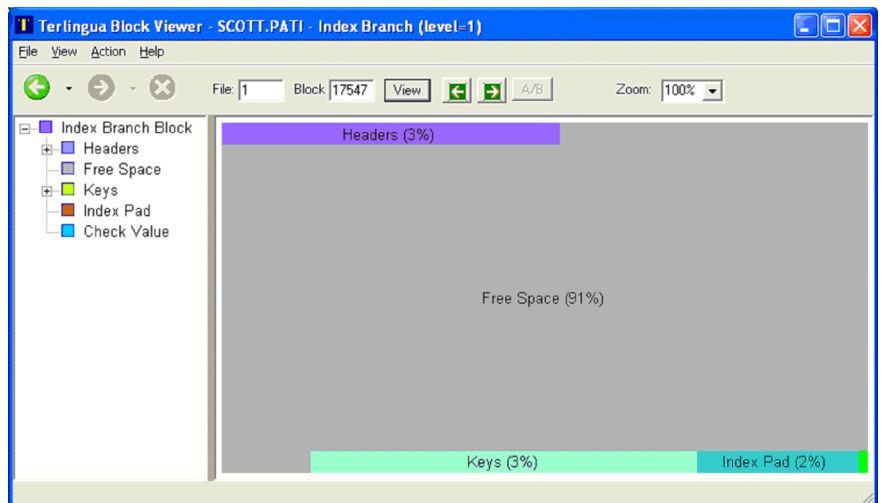
```
SQL> drop index pati;
Index dropped.
SQL> create index pati on pat(ssn, name) pctfree 40;
Index created.
SQL> spool off;
SQL> select nkeys
2      , count(*)
3      , rpad(min(minkey), 11) samplemin
4      , rpad(min(maxkey), 11) samplemax
5 from helper
6 group by nkeys
7 order by 1
8 ;
```

NKEYS	COUNT(*)	SAMPLEMIN	SAMPLEMAX
2	20	081-08-7388	084-73-5622
19	1	1	9

By setting `pct_free` to 40, all 20 index blocks have room to receive two more similarly sized keys. Admittedly, the new index takes up nearly twice the disk space and therefore nearly twice the buffer cache memory to hold. However, with any level of DML load, this index will run significantly faster than the "highly compressed" 11 block index.

The Root Block

One of the concerns about using a sparse index approach is that you do not want to create more levels in an index than is absolutely necessary. The risk of creating another index level can be ascertained by viewing the root block.



Our root block is not in danger of splitting, as it has over 90% free space. How can the root block hold 20 keys and still have 90% free, while an individual leaf block can only hold 4 keys? The following query should clear this up.

```
SQL> select rpad(char01, 11) lowkey
2      , nextdbadot      block
3  from index_keys
4  where owner = 'SCOTT'
5        and name = 'PATI'
6        and indexlevel = 1
7  order by 1
8  ;
```

LOWKEY	BLOCK
1	7.4101
15	7.4102
19	7.4103
22	1.17536
23	1.17537
27	1.17538
289	1.17539
357-6	1.17540
36	7.4937
437	7.4938
47	7.4939
5	7.4940
58	7.4941
66	7.4942
7	7.4943
78	7.4944
79	7.4945
83	7.4946
9	1.20466

19 rows selected.

In the branch index blocks, Oracle only stores enough of the key value to distinguish the highest key value of the previous leaf block from the lowest key value of the next leaf block. Look at the key values in blocks 1.17538 and 1.17539. According to the root block, the separating key between these two blocks is "289".

```
SQL> select mydbadot      dba
2      , rpad(char01, 11) ssn
3  from index_keys
4  where owner      = 'SCOTT'
5        and name    = 'PATI'
6        and ( (file# = 1 and block# = 17538)
7              or (file# = 1 and block# = 17539))
8  order by 2
9  ;
```

DBA	SSN
1.17538	278-87-0378
1.17538	281-77-2553
1.17539	289-39-5390
1.17539	357-59-5655

One can see that the demarcation of "289" is a minimal string to divide those keys between blocks 1.17538 and 1.17539. The string "28" would not be precise enough since "281-77-2553" and "289-39-5390", although in different blocks, share that prefix. Note also the strings "282", "283", "284", "285", "286", "287", and "288" would also work.

Conclusion

Although this article has touched on many topics, perhaps the most important lesson is that smaller or compressed indices are often worse than larger or sparse indices.

As always, I hope this article has provided some important insight into the functioning of Oracle as well as given the reader some appreciation as to the value our tools provide to both the novice and veteran database administrator. If you have found the information herein to be valuable, please mention it in your e-mails or postings to other Oracle web sites.

Extra Credit

Any new discovery raises new questions. For those wishing a bit more - read on.

Would Oracle Unnecessarily Split a Leaf Block?

The PATI example above brings up a question. If block 1.17538 were completely filled with keys, and an insertion request came in higher than the highest key in 1.17538 but lower than the lowest key in 1.17539, would Oracle split 1.17538 or simply place the new key in 1.17539 and lower the low key for that block? First, insert two new keys into 1.17538 to fill the block.

```
SQL> insert into pat values('282-55-5555', rpad('Tiger', 800));
1 row created.
SQL> insert into pat values('283-55-5555', rpad('Tiger', 800));
1 row created.
SQL> commit;
Commit complete.
SQL> select mydbadot          dba
       2      , rpad(char01, 11) ssn
       3      from index_keys
       4      where owner      = 'SCOTT'
       5      and name        = 'PATI'
       6      and ( (file# = 1 and block# = 17538)
       7            or (file# = 1 and block# = 17539))
       8      order by 2
       9      ;
DBA          SSN
-----
1.17538      278-87-0378
1.17538      281-77-2553
1.17538      282-55-5555
1.17538      283-55-5555
1.17539      289-39-5390
1.17539      357-59-5655
6 rows selected.
```

Note that block 1.17538 is now full. Now let us insert our test key above "283-55-5555" but below "289" and then query the results.

```
SQL> insert into pat values('284-55-5555', rpad('Tiger', 800));
1 row created.
SQL> commit;
Commit complete.
SQL> select rpad(char01, 11) lowkey
       2      , nextdbadot      block
       3      from index_keys
       4      where owner = 'SCOTT'
       5      and name = 'PATI'
       6      and indexlevel = 1
       7      order by 1
       8      ;
LOWKEY      BLOCK
-----
1           7.4101
15          7.4102
19          7.4103
22          1.17536
23          1.17537
27          1.17538
284        1.20467
```

```

289          1.17539
357-6        1.17540
36           7.4937
437          7.4938
47           7.4939
5            7.4940
58           7.4941
66           7.4942
7            7.4943
78           7.4944
79           7.4945
83           7.4946
9            1.20466
20 rows selected.
SQL> select mydbadot          dba
2          , rpad(char01, 11) ssn
3      from index_keys
4      where owner          = 'SCOTT'
5            and name       = 'PATI'
6            and ( (file#   = 1 and block# = 17538)
7              or (file#   = 1 and block# = 17539)
8              or (file#   = 1 and block# = 20467))
9      order by 2
10     ;
DBA          SSN
-----
1.17538      278-87-0378
1.17538      281-77-2553
1.17538      282-55-5555
1.17538      283-55-5555
1.20467      284-55-5555
1.17539      289-39-5390
1.17539      357-59-5655
7 rows selected.

```

Oracle created a new index leaf block to hold the new key, 1.20467, when in practice it could have placed the new key as the lowest key in 1.17539. The insertion of this new block may look easy, but keep in mind how expensive it is to do. As we saw in the main article, at least 5 data blocks need to be modified to accomplish the goal. By simply inserting the new key into block 1.17539, Oracle could have done it by modifying only 2 blocks:

1. The leaf block itself (1.17539)
2. The root block to lower the lowest key in 1.17539 from "289" to "284".

My last question

A gold star to anyone who can figure out what is in the last 64 bytes, toward the end of every index block? It starts out containing all zeros and only "develops" values in branch blocks during a leaf block split. None of the Oracle block dumps, that I am aware of, parse or dump it. This is an example from the split of one of the leaf blocks in the PATI index example above. Keep in mind, this appears in the branch block, not the leaf block.

```

c1 02 01 80 02 c1 02 02 c1 02 01 80 01 80 7c 02 13 01 ac 01 01 00 32 00 32 00 40 47 6a
00 01 00 40 47 6a 00 01 03 c2 0c 47 bc 01 01 00 00 00 00 00 40 47 6a 00 00 00 40 47 6a
00 00 03 c2 5a 3c

```

"02 c1 02" is the length, plus the Oracle number "1", and "01 80" is the length, plus the Oracle number "0". The "0040476a.0001" and "0040476a.0000", which appear twice, are rowids. "03c20c47" and "03c25a3c" are also length, plus Oracle numbers. I suspect these values are recorded here to prevent some worse case index split behavior. As they say in college texts, "The rest is left as an exercise for the reader."