

Welcome!

The following five articles demonstrate how the “Terlingua Block Viewer for Oracle” reveals detailed information about your Oracle database that is not provided by any other vendor (including Oracle). This information is vital to the better understanding of row locking, row chaining, clusters, hashed clusters, B*tree indices, index organized tables, bitmap indices – practically every aspect of the physical layout of data on disk. The Block Viewer delivers this information via either a graphical block level browser, or very powerful meta-views built directly above the raw Oracle data. The Block Viewer may read blocks directly from the Oracle buffer cache, or from the on disk data files.

While solving real world problems, this “Welcome” section highlights several of the Block Viewer’s unique features. The subsequent articles are a mixture of solutions to real world problems and a primer on the inner workings of Oracle. I hope you enjoy them.

Row Chaining and Row Migration

Our TABLE_ROWS view reveals more about the chained rows in a table than the ANALYZE TABLE command in Oracle. For example, this SQL fragment displays numerous fields for each row piece that is part of a single row chain.

```
SQL> select myrid
2      , rowflag
3      , nextrid
4      , piecesize
5      , ncols
6      , firstl
7      , lastl
8      from TABLE_ROWS
9      where owner = 'SCOTT'
10     and name = 'CHAIN'
11     start with myrid = '000047d0.0001.0001'
12     connect by prior nextrid = myrid
13 ;
```

MYRID	ROWFLAG	NEXTRID	PIECESIZE	NCOLS	FIRSTL	LASTL
000047d0.0001.0001	--H-----	00004591.0000.0001	9	0	0	0
00004591.0000.0001	----F--N	0000161f.0000.0007	3574	3	1	1651
0000161f.0000.0007	-----PN	0000161e.0000.0007	3899	3	249	1734
0000161e.0000.0007	-----PN	0000161d.0000.0007	3899	3	166	1817
0000161d.0000.0007	-----LP-	00000000.0000.0000	3893	3	83	1900

5 rows selected

Administrators can determine which rows are small enough to benefit from being reinserted and which rows are simply too long to fit in a single block. An administrator can also determine the level of row chaining that would persist even if the block size were enlarged.

Space Utilization Within Data Blocks

Our TABLE_BLOCKS view, built directly on the Oracle database blocks, allows administrators to view space utilization within data blocks. The following query produces a histogram of database blocks based on available space (useful in determining if a table rebuild is in order to reduce future chaining).

```

select round(availablespace, -1) availablespace
       , count(*)                nblocks
       , min(mydbadot)           sample
from TABLE_BLOCKS
where owner = 'SCOTT'
      and name = 'ACCOUNT'
group by round(availablespace, -1);
AVAILABLESPACE      NBLOCKS SAMPLE
-----
                0                1 1.26710
                260             5991 1.33803
                270                9 1.33802
                280             659 1.26858
                290                7 1.26711
                830                1 1.40262
6 rows selected.

```

From this query we can see that the majority of our table blocks have between 260 and 280 bytes available for new or growing rows.

Health of a B*Tree Index

By directly querying the index blocks themselves, our INDEX_KEYS meta-view is uniquely able to determine the overall health of an index. For example, the following query reveals the number of keys in each index leaf block and the key ranges each leaf block covers.

```

SQL> select count(*)                nkeys
       , mydbadot                  dba
       , rpad(min(char01), 11) minkey
       , rpad(max(char01), 11) maxkey
from index_keys
where owner = 'SCOTT'
      and name = 'PATI'
      and indexlevel = 0
group by mydbadot
order by 3;
NKEYS DBA                MINKEY                MAXKEY
-----
    36 7.11159           002-28-1787          098-75-1374
    27 7.11163           101-01-6109          149-85-3519
    47 7.11162           151-78-3706          251-48-0713
    33 1.17106           261-34-5178          329-27-4958
    25 7.11166           333-92-2527          399-38-9904
    33 1.17104           405-09-8951          498-93-4833
    33 7.11164           500-92-5529          585-90-2419
    48 7.11161           592-20-7952          696-66-0358
    32 1.17105           700-18-8878          769-25-2130
    25 1.17326           771-69-7071          831-73-6413
    30 7.11160           840-50-0123          895-01-7211
    31 7.11165           901-64-7586          997-34-4012
12 rows selected.

```

This query reveals that the index leafs contain anywhere from 25 to 48 keys per block. Although this may be a normal, fairly balanced index, it could benefit from a rebuild.

```

SQL> alter index pati rebuild pctfree 40;
Index altered.
SQL> select count(*)                nkeys
       , mydbadot                  dba
       , rpad(min(char01), 11) minkey
       , rpad(max(char01), 11) maxkey
from index_keys
where owner = 'SCOTT'
      and name = 'PATI'
      and indexlevel = 0
group by mydbadot
order by 3
11 ;

```

NKEYS	DBA	MINKEY	MAXKEY
30	7.20381	002-28-1787	091-51-9300
30	1.17336	093-85-7414	145-77-0464
30	1.17337	147-56-0537	221-79-6525
30	1.17338	223-27-0853	282-38-6809
30	7.20382	284-64-0990	353-11-3387
30	7.20383	353-77-3089	437-35-4859
30	7.20384	438-03-8007	522-95-8335
30	7.20385	524-17-2630	613-03-6284
30	7.20386	613-14-2831	672-39-5374
30	1.16769	675-75-9523	739-77-6249
30	1.16770	741-65-8515	802-85-7708
30	1.16771	804-53-5694	880-27-6106
30	1.16772	881-60-9619	962-23-5683
10	1.16773	967-83-0133	997-34-4012

14 rows selected.

This query confirms that our new index, created with pctfree set to 40, has 14 leaf blocks with most containing 30 keys – a perfectly balanced index best ready to absorb new inserts without necessitating a leaf block split.

Health of Index Organized Tables

Once rows grow large enough as to no longer fit in the index portion of the IOT, the performance gains sought from IOTs rapidly diminish. The following query from our IOT_KEYS

```
SQL> select count(*)
2     from IOT_KEYS
3     where nextkeyfile# != 0
4         and owner      = 'SCOTT'
5         and name       = 'EMP_IOT'
6     ;
COUNT(*)
-----
429
```

meta-view, confirms the number of rows that have spilled into the overflow segment. Performance will be negatively impacted if the latter columns of a table are frequently queried and if the percentage of rows whose latter columns are stored in the overflow segment increases.

Health of a Bitmap Index

Bitmap indices represent one of the most dramatic ways to improve queries which test for the equality (or inequality) of low cardinality fields. However, “bitmap index” is somewhat of a misnomer since in most situations Oracle does not represent bitmap indexes as a map of bits, more often than not, Oracle uses a series of deltas to go from one rowid to the next. By using the “alter table minimize records_per_block” command, users increase the odds that Oracle may represent more rows with bitmaps than deltas, thereby decreasing the amount of space needed to represent the bitmap index – a potential performance increase. Using our BITMAP_KEYS view, one can directly determine the number of rows represented by bitmaps and the number of rows represented by deltas. With this view, the benefits of using minimize records_per_block can be directly measured by comparing two bitmap indices built above an identical set of rows. The PAT_S_MINIMIZE index was built after the “alter table minimize records_per_block” was executed on the base table.

```

SVRMGR> select decode(btyp, 0, 'DELTA', 1, 'BITMAP') btype
2>      , count(*)
3> from bitmap_keys
4> where owner      = 'SCOTT'
5>    and name       = 'PAT_S_NOTHING'
6>    and indexlevel = 0
7> group by btyp;
BTYP  COUNT(*)
-----
DELTA      7625
BITMAP     2375
2 rows selected.
SVRMGR> select decode(btyp, 0, 'DELTA', 1, 'BITMAP') btype
2>      , count(*)
3> from bitmap_keys
4> where owner      = 'SCOTT'
5>    and name       = 'PAT_S_MINIMIZE'
6>    and indexlevel = 0
7> group by btyp;
BTYP  COUNT(*)
-----
DELTA      4834
BITMAP     5166
2 rows selected.

```

This query reveals that by utilizing the minimize records_per_block feature, Oracle went from representing 23% to 51% of the rows with a bitmap: a potentially dramatic savings.

Which Rows Are Locked by a Given Transaction?

For all the power of row level locking, Oracle provides surprisingly little access to the meta data used to manage it. For example, Oracle provides no support for the simple query of selecting all rows in a table locked by a given transaction. Fortunately, Terlingua does.

```

SQL> select myrid
2   from table_rows
3   where txid = '0001.002.0000027e'
4     and owner = 'SCOTT'
5     and name = 'EMP';
MYRID
-----
000052d1.0001.0001

```

This type of query is invaluable if you know a given session is waiting on a given transaction and you would like to know what specific row is causing the contention. If this sort of contention is common, an application developer can determine which row is driving it.

Data Recovery and Block Patching

Our block viewer may retrieve “by hand” the rows found in a corrupt block or assist in patching corrupt blocks. Since our block viewer need not be as thorough as Oracle is when accessing a data block, there are many circumstances where our viewer will be able to display a corrupted data block and translate the rows and columns found therein. This enables a database administrator to salvage the rows on the corrupt block for reinsertion. As a last resort, our product also has the ability to modify pages directly on disk — a block editor. This, of course, comes with many precautions and should only be used in dire circumstances, ideally in conjunction with Oracle support, but it is far superior to the ill documented block patching tool provided by Oracle.

Conclusion

By providing direct access to your Oracle data blocks, either via the graphical block viewer or through meta-views, our product provides unparalleled understanding of the fundamentals of your data. Any site which places a premium on their database and database applications should not be without the Terlingua Block Viewer for Oracle.

Contents

Chapter 1

Row Level Locking in Oracle	7
Oracle Block Format	7
Unneeded Contention and Possible False Deadlocks	10
Conclusion	13

Chapter 2

Row Chaining in Oracle	15
Row Format	16
Row Chaining	18
Row Deletion and Fragmentation	20
A PCTFREE Surprise	21
Conclusion	22
Extra Credit	22

Chapter 3

Oracle Index Basics	25
Branch, Leaf and Root Blocks	25
Our Example.....	25
Leaf Block Splitting	28
The Root Block	30
Conclusion	31
Extra Credit	31

Chapter 4

Index Organized Tables	35
Our EMP example.....	35
Performance Issues	38
The Neutrals	41
The Downsides	43
Conclusion	46

Chapter 5

Bitmap Indexes	47
The Paper Route	47
Let's Get Started	47
The Paper Route Revisited	50
Conclusion	54
Extra Credit	54
Four Enhancement Requests to Oracle	55

