

Chapter 2

Row Chaining in Oracle

More has been written about row chaining and row migration than any other subject concerning the Oracle RDBMS. Row chaining occurs when a row can no longer fit into its original block. If the entire row can fit in a new block, the row is moved completely, leaving only a forwarding pointer – this is known as row migration. If the row has grown so large that it may not fit in a single block then the row is split into two or more blocks – row chaining. When Oracle is forced to split a row into pieces, it often splits individual columns into one or more pieces.

Row identifiers (rowids) reference individual contiguous pieces of data on disk, a “row piece”. In the absence of row chaining each row piece is a row. In row chaining, an individual row consists of several row pieces. In our browser, when you see a “row” on disk, you are viewing a row piece. Only by taking a closer look at the row piece can one determine if it is a complete row, or only part of one.

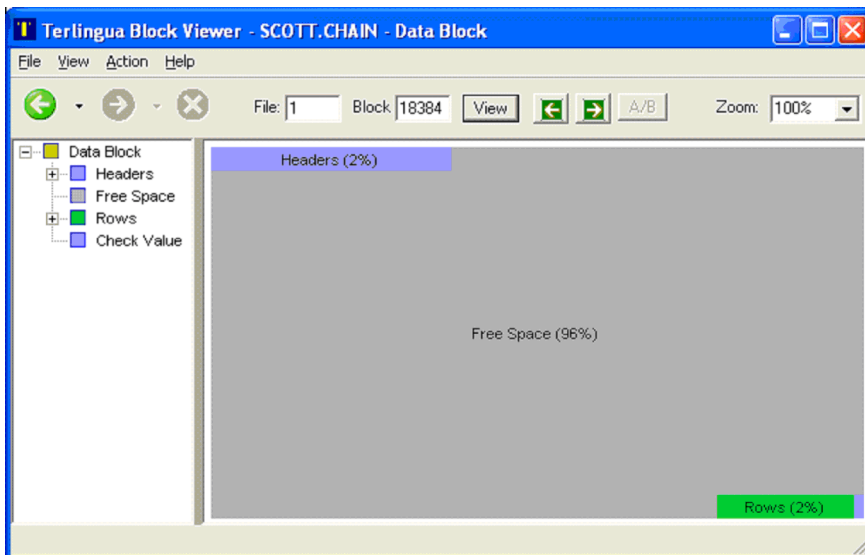
Although technically not relevant to row chaining, we will also cover NULLS, trailing NULLS, and deleted rows. First, consider a typical row chaining example table.

```
SQL> create table chain (
  2     c1 varchar(2000), c2 varchar(2000)
  3     , c3 varchar(2000), c4 varchar(2000)
  4     , c5 varchar(2000), c6 varchar(2000)
  5     , c7 varchar(2000), c8 varchar(2000)
  6     , c9 varchar(2000)
  7   ;
Table created.
SQL> insert into chain values('0',  NULL, NULL, NULL, NULL, '00', NULL, NULL, NULL);
1 row created.
SQL> insert into chain values('1',  NULL, NULL, NULL, NULL, NULL, '01', NULL, NULL);
1 row created.
SQL> insert into chain values('2',  NULL, NULL, NULL, NULL, NULL, NULL, '02', NULL);
1 row created.
SQL> insert into chain values('3',  NULL, NULL, NULL, NULL, NULL, NULL, NULL, '03');
1 row created.
SQL> commit;
Commit complete.
```

Note the first few rows have NULL column values at the end, “trailing NULL columns” (in Oracle parlance). The NCOLS field in the following query, from the meta-view TABLE_ROWS, uniquely provided by our product, confirms that Oracle does not store trailing NULL columns on disk. It simply records a physical column count less than the column count of the table.

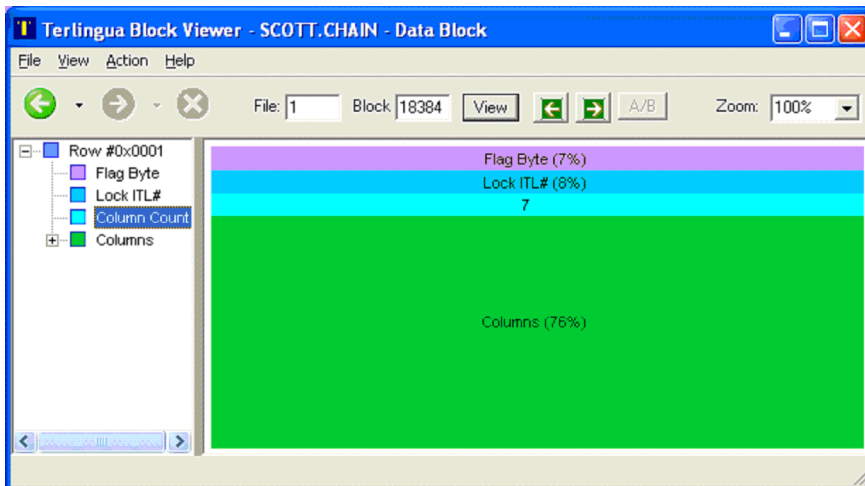
```
SQL> select rpad(myrid, 18)          myrid
  2     , rowflagdecode              rowflag
  3     , rpad(nextrid, 18)         nextrid
  4     , totalsize                 piecesize
  5     , columncount              ncols
  6     , firstcollen              firstl
  7     , lastcollen               lastl
  8   from TABLE_ROWS
  9   where owner = 'SCOTT'
 10   and name  = 'CHAIN'
 11   ;
```

MYRID	ROWFLAG	NEXTRID	PIECESIZE	NCOLS	FIRSTL	LASTL
000047d0.0000.0001	--H-FL--	00000000.0000.0000	12	6	1	2
000047d0.0001.0001	--H-FL--	00000000.0000.0000	13	7	1	2
000047d0.0002.0001	--H-FL--	00000000.0000.0000	14	8	1	2
000047d0.0003.0001	--H-FL--	00000000.0000.0000	15	9	1	2



A graphical view of the one block table follows.

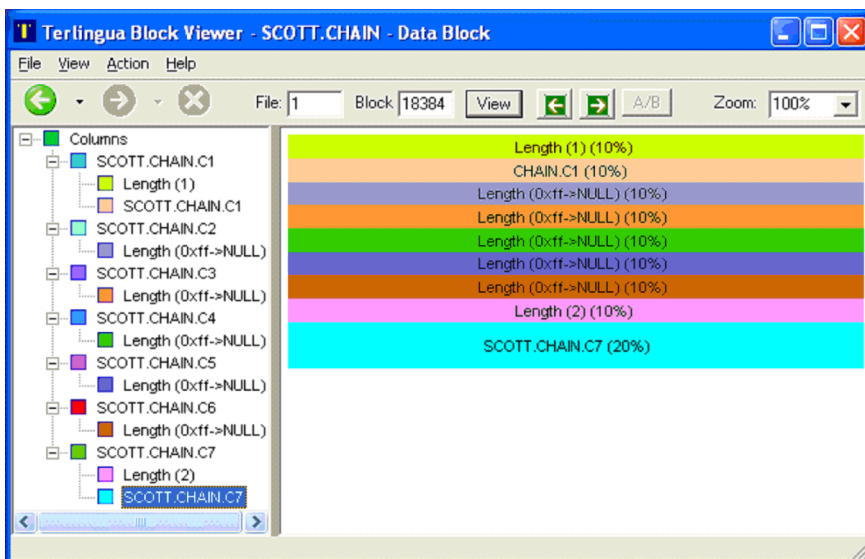
These four small rows only occupy 2% of the block.



Row Format

Let us look at a particular row.

Each row has a three byte header consisting of a Flag Byte, an ITL#, and a Column Count. The flag byte contains 8 bits of on/off information (to be discussed in detail shortly). The ITL# indicates count represents the physical number of columns present in the row piece.



Column Format

The individual columns appear immediately after the column count. Each column is a length followed by data with the length 0xFF reserved to indicate NULL.

With this methodology Oracle can store the entire 9 column row created by:

```
SQL> insert into chain values('1', NULL, NULL, NULL, NULL, NULL, '01', NULL, NULL);
```

with 13 bytes – 3 for the row header, 7 one byte lengths, and 3 bytes for the data. Two additional bytes are needed in the block header to hold the offset of this row in the block bringing the total cost to 15 bytes. Oracle reserves the one byte length of 0xfe to indicate that the column length is actually stored in a following two byte value. This allows Oracle to represent columns larger than 253 bytes. For example, the length of 1024 would be represented as "0xfe0400" - the 0xfe indicating a two byte length and the 0x0400 indicating 1024.

Row Migration

With a basic understanding of the row and column format, we can now move onto row chaining and migration. First, let us increase the size of all the rows to occupy a larger percentage of the block.

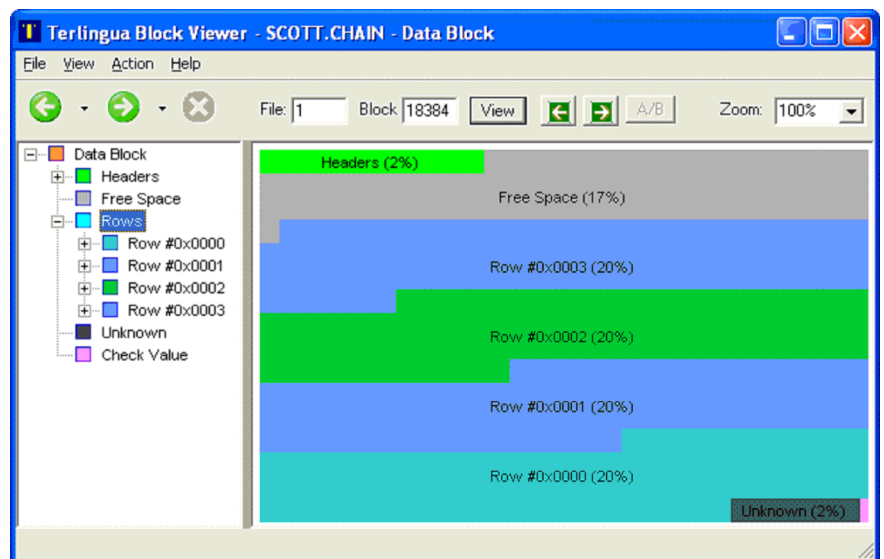
```
SQL> update chain set c3 = rpad(' ', 800);
4 rows updated.
SQL> commit;
Commit complete.
```

```
SQL> select rpad(myrid, 18)          myrid
       2      , rowflagdecode        rowflag
       3      , rpad(nextrid, 18)    nextrid
       4      , totalsize            piecesize
       5      , columncount          ncols
       6      , firstcollen          firstl
       7      , lastcollen           lastl
       8      from TABLE_ROWS
       9      where owner = 'SCOTT'
      10      and name = 'CHAIN'
      11 ;
```

MYRID	ROWFLAG	NEXTRID	PIECESIZE	NCOLS	FIRSTL	LASTL
000047d0.0000.0001	--H-FL--	00000000.0000.0000	814	6	1	2
000047d0.0001.0001	--H-FL--	00000000.0000.0000	815	7	1	2
000047d0.0002.0001	--H-FL--	00000000.0000.0000	816	8	1	2
000047d0.0003.0001	--H-FL--	00000000.0000.0000	817	9	1	2

This is graphically displayed as.

The four rows now occupy the majority of the block. The "Unknown" piece at the end is uncompressed space Oracle has yet to reclaim for free space. We can migrate row #2 by making it too large to fit in this block, but small enough to fit in another. The results of the migration are revealed via the query on TABLE_ROWS.



```
SQL> update chain set c2 = rpad('0', 1500, '0')
       2      , c3 = rpad('0', 1500, '0')
       3      where c1 = '2';
1 row updated.
SQL> commit;
Commit complete.
SQL> select rpad(myrid, 18)          myrid
       2      , rowflagdecode        rowflag
       3      , rpad(nextrid, 18)    nextrid
```

```

4      , totalsize          piecesize
5      , columncount       ncols
6      , firstcollen       firstl
7      , lastcollen        lastl
8      from TABLE_ROWS
9      where owner = 'SCOTT'
10     and name = 'CHAIN'
11 ;

```

MYRID	ROWFLAG	NEXTRID	PIECESIZE	NCOLS	FIRSTL	LASTL
000047d0.0000.0001	--H-FL--	00000000.0000.0000	814	6	1	2
000047d0.0001.0001	--H-FL--	00000000.0000.0000	815	7	1	2
000047d0.0002.0001	--H-----	000047d1.0000.0001	9	0	0	0
000047d0.0003.0001	--H-FL--	00000000.0000.0000	817	9	1	2
000047d1.0000.0001	----FL--	00000000.0000.0000	3024	8	1	2

The row '000047d0.0002.0001' has migrated to '000047d1.0000.0001'.

3 of the 8 flag word bits

Three of the eight bits in the flag word are exemplified by '--H-FL--'. The 'H' bit indicates the row piece is the head row piece. The 'F' bit indicates the first column of this row piece is the first column of the row. The 'L' bit indicates the last column of this row piece is the last column of the row. In a normal one piece row, the flag is '--H-FL--'. In our totally migrated row, the original row piece only contains '--H-----' and the migrated row piece contains '--FL--'.

Row Chaining

We create a truly chained row by updating row #1 to be too large to fit within one block. We can then observe the resulting row pieces by using a helper table and a CONNECT BY clause.

```

SQL> update chain set c2 = rpad('0', 1900, '0')
2      , c3 = rpad('0', 1900, '0')
3      , c4 = rpad('0', 1900, '0')
4      , c5 = rpad('0', 1900, '0')
5      , c6 = rpad('0', 1900, '0')
6      , c7 = rpad('0', 1900, '0')
7      , c8 = rpad('0', 1900, '0')
8      , c9 = rpad('0', 1900, '0')
9  where c1 = '1';
1 row updated.
SQL> commit;
Commit complete.
SQL> drop table helper;
Table dropped.
SQL> create table helper as
2  select rpad(myrid, 18)          myrid
3      , rowflagdecode           rowflag
4      , rpad(nexttrid, 18)      nexttrid
5      , totalsize              piecesize
6      , columncount            ncols
7      , firstcollen            firstl
8      , lastcollen             lastl
9  from TABLE_ROWS
10 where owner = 'SCOTT'
11 and name = 'CHAIN'
12 ;
Table created.
SQL> select myrid
2      , rowflag
3      , nexttrid
4      , piecesize
5      , ncols
6      , firstl
7      , lastl
8  from helper

```

```

9  start with myrid = '000047d0.0001.0001'
10 connect by prior nextrid = myrid
11 ;

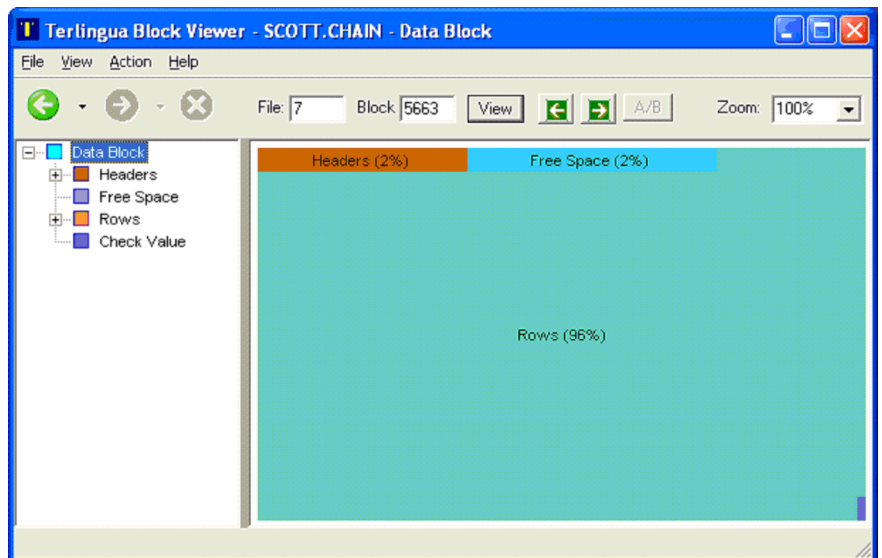
```

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

The result of that last query contains everything you would ever want to know about row chaining. The original rowid is still in place at '000047d0.0001.0001' demarked with the '-H-----' with no columns. The first migrated row piece resides in '00004591.0000.0001', as indicated by the 'F' bit. The last row piece resides in '0000161d.0000.0007', as indicated by the 'L' bit.

The 'P' in the flag indicates the row piece's first column is a partial or continuation column from the previous piece. The 'N' in the flag indicates the row piece's last column is a partial column to be continued in the next piece. In this row, the third column of the table has been fragmented between row piece two and three.

This query also includes the length of the first and last columns of every row piece. For the middle row pieces, the last column length of one row piece, plus the first column length of the next row piece, equals the length of a column (1900 bytes). Every 'N' in one row piece must have a 'P' in the next row piece. The careful reader can also see that over the 5 row pieces we do have 9 logical columns fractured over 12 physical columns. Here is a screen shot of one of the middle pieces of the row.



During chaining, Oracle has ignored the PCTFREE setting of 10 for this table and only set aside 100 or so bytes for expansion. This is the desired behavior since this row has already chained and now the goal is to reduce the number of blocks needed to hold it. PCTFREE is there to prevent chaining, but once chaining has occurred, it makes more sense to use as few blocks as possible to store the row.

Row Deletion and Fragmentation

Let us examine how Oracle deletes a row by deleting row#0.

```

SQL> delete from chain where c1 = '0';
1 row deleted.
SQL> commit;
Commit complete.
SQL> select rpad(myrid, 18)          myrid
2          , rowflagdecode         rowflag
3          , rpad(nextrid, 18)     nextrid
4          , totalsize             piecesize

```

```

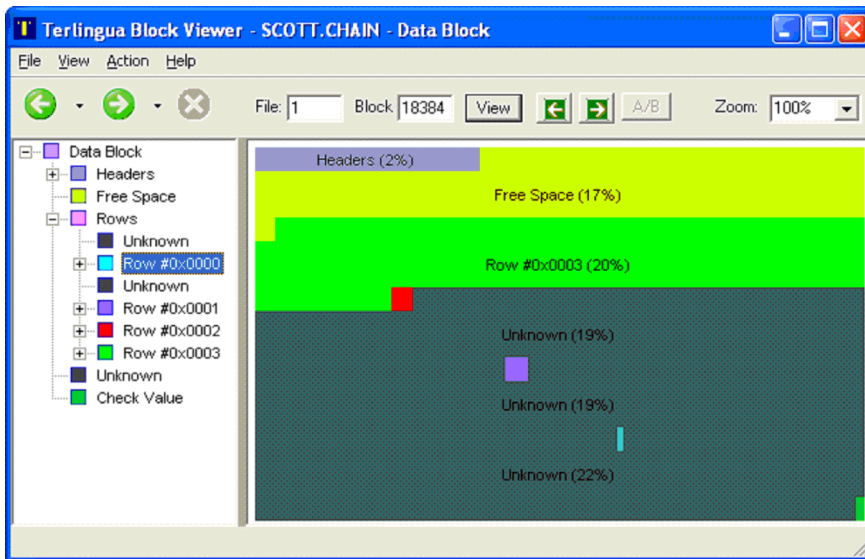
5      , columncount          ncols
6      , firstcollen         firstl
7      , lastcollen          lastl
8  from TABLE_ROWS
9  where owner = 'SCOTT'
10     and name = 'CHAIN'
11 ;

```

MYRID	ROWFLAG	NEXTRID	PIECESIZE	NCOLS	FIRSTL	LASTL
000047d0.0000.0001	--HDFL--	00000000.0000.0000	2	0	0	0
000047d0.0001.0001	--H----	00004591.0000.0001	9	0	0	0
000047d0.0002.0001	--H----	000047d1.0000.0001	9	0	0	0
000047d0.0003.0001	--H-FL--	00000000.0000.0000	817	9	1	2
000047d1.0000.0001	----FL--	00000000.0000.0000	3024	8	1	2
0000161d.0000.0007	----LP-	00000000.0000.0000	3893	3	83	1900
0000161e.0000.0007	-----PN	0000161d.0000.0007	3899	3	166	1817
0000161f.0000.0007	-----PN	0000161e.0000.0007	3899	3	249	1734
00004591.0000.0001	----F--N	0000161f.0000.0007	3574	3	1	1651

9 rows selected.

Observe the row flag value of '--HDFL--' in rowid '000047d0.0000.0001'. The 'D' in the row flag indicates that the row is deleted. If more space is needed in this block, the rows may be recompressed and stored back at the end of the block. Until such time, our block is looking a bit fragmented.



be recompressed and stored back at the end of the block. Until such time, our block is looking a bit fragmented.

A "CREATE TABLE AS SELECT" should clear up most of the block level fragmentation.

```

SQL> create table chain2 as select * from chain;
Table created.
SQL> select rpad(myrid, 18)          myrid
2      , rowflagdecode              rowflag
3      , rpad(nextrid, 18)          nextrid
4      , totalsize                  piecesize
5      , columncount                ncols
6      , firstcollen                firstl
7      , lastcollen                 lastl
8  from TABLE_ROWS
9  where owner = 'SCOTT'
10     and name = 'CHAIN2'
11 ;

```

MYRID	ROWFLAG	NEXTRID	PIECESIZE	NCOLS	FIRSTL	LASTL
00004ae1.0000.0001	--H-FL--	00000000.0000.0000	817	9	1	2
00004ae2.0000.0001	--H-FL--	00000000.0000.0000	3018	8	1	2
00004ae2.0001.0001	--H-F--N	00004ae3.0000.0001	583	2	1	569
00004ae3.0000.0001	-----PN	00004ae4.0000.0001	3595	3	1331	346
00004ae4.0000.0001	-----PN	00004ae5.0000.0001	3593	3	1554	123
00004ae5.0000.0001	-----PN	00004ae6.0000.0001	3595	2	1777	1803
00004ae6.0000.0001	-----PN	00001620.0000.0007	3595	3	97	1582
00001620.0000.0007	----LP-	00000000.0000.0000	324	1	318	318

8 rows selected.

The recreation has removed the deleted row and more tightly packed the remaining 3 rows into 8 blocks. Our large row is still chained, of course.

A PCTFREE Surprise

Our relatively long row that was chained over 5 pieces in "CHAIN" is now chained over 6 pieces in "CHAIN2"!

```
SQL> drop table helper;
Table dropped.
SQL> create table helper as
  2  select rpad(myrid, 18)          myrid
  3          , rowflagdecode        rowflag
  4          , rpad(nextrid, 18)    nextrid
  5          , totalsize            piecesize
  6          , columncount         ncols
  7          , firstcollen         firstl
  8          , lastcollen          lastl
  9  from TABLE_ROWS
 10  where owner = 'SCOTT'
 11  and name = 'CHAIN2'
 12  ;
Table created.
SQL> select myrid
  2          , rowflag
  3          , nextrid
  4          , piecesize
  5          , ncols
  6          , firstl
  7          , lastl
  8  from helper
  9  start with myrid = '00004ae2.0001.0001'
 10  connect by prior nextrid = myrid
 11  ;
```

MYRID	ROWFLAG	NEXTRID	PIECESIZE	NCOLS	FIRSTL	LASTL
00004ae2.0001.0001	--H-F--N	00004ae3.0000.0001	583	2	1	569
00004ae3.0000.0001	-----PN	00004ae4.0000.0001	3595	3	1331	346
00004ae4.0000.0001	-----PN	00004ae5.0000.0001	3593	3	1554	123
00004ae5.0000.0001	-----PN	00004ae6.0000.0001	3595	2	1777	1803
00004ae6.0000.0001	-----PN	00001620.0000.0007	3595	3	97	1582
00001620.0000.0007	-----LP-	00000000.0000.0000	324	1	318	318

6 rows selected.

When the row initially was chained 'in place', Oracle responded optimally by placing as much as the row as possible (only saving a hundred bytes or so for minor growth) into each block ignoring PCT_FREE. Now that the row has been reinserted from scratch, Oracle still honors PCT_FREE. The entire point of PCT_FREE is to prevent row chaining but this row is already chained. The 10% free should only be for those blocks containing rows not yet chained. This gets worse if the database administrator is aware of potential chaining in this table. Say every week it is rebuilt with a PCT_FREE value of 50. This row may grow to occupy 10 blocks! The more you know about Oracle, the more questions arise.

Conclusion

This article has spanned many topics including trailing nulls, NULL column representation, row migration, row chaining, column fracturing, deletion of rows, and repacking of blocks. It was more than I intended to cover upon starting this article, but one topic either depended upon, or flowed into the next.

As always, I hope this article has provided some important insight into the functioning of Oracle, as well as given the reader some appreciation of 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

The following could not be used in the article above and stay within the desired scope. For those wishing more – read on.

Trailing NULLs Left Dangling

What happens if we set some of the middle columns of a large chained row back to NULL - “unchaining”?

```
SQL> update chain set c3=NULL, c4=NULL, c5=NULL, c6=NULL, c7=NULL, c8=NULL
 2  where c1 = '1';
1 row updated.
SQL> commit;
Commit complete.
SQL> drop table helper;
Table dropped.
SQL> create table helper as
 2  select rpad(myrid, 18)          myrid
 3          , rowflagdecode       rowflag
 4          , rpad(nextrid, 18)   nextrid
 5          , totalsize           piecesize
 6          , columncount        ncols
 7          , firstcollen        firstl
 8          , lastcollen         lastl
 9  from TABLE_ROWS
10  where owner = 'SCOTT'
11  and name = 'CHAIN'
12  ;
Table created.
SQL> select myrid
 2          , rowflag
 3          , nextrid
 4          , piecesize
 5          , ncols
 6          , firstl
 7          , lastl
 8  from helper
 9  start with myrid = '000047d0.0001.0001'
10 connect by prior nextrid = myrid
11  ;
```

MYRID	ROWFLAG	NEXTRID	PIECESIZE	NCOLS	FIRSTL	LASTL
000047d0.0001.0001	--H----	00004591.0000.0001	9	0	0	0
00004591.0000.0001	----F---	0000161f.0000.0007	1921	3	1	0
0000161f.0000.0007	-----	0000161e.0000.0007	11	2	0	0
0000161e.0000.0007	-----	0000161d.0000.0007	11	2	0	0
0000161d.0000.0007	-----L--	00000000.0000.0000	1907	2	0	1900

Oracle did not do much here. It could jump from '00004591.0000.0001' to '000161d.0000.0007' and place the four NULL columns of '0000161f.0000.0007' and '0000161e.0000.0007' in the last piece. It could also mark '0000161f.0000.0007' and '0000161e.0000.0007' as deleted. However, Oracle did none of this, it simply performed the least work possible at the time.

I wonder what happens if we NULL out the last column?

```
SQL> update chain set c9=NULL where c1 = '1';
1 row updated.
SQL> commit;
Commit complete.
SQL> drop table helper;
Table dropped.
SQL> create table helper as
 2  select rpad(myrid, 18)          myrid
 3          , rowflagdecode       rowflag
 4          , rpad(nextrid, 18)   nextrid
 5          , totalsize           piecesize
 6          , columncount        ncols
 7          , firstcollen        firstl
```

```

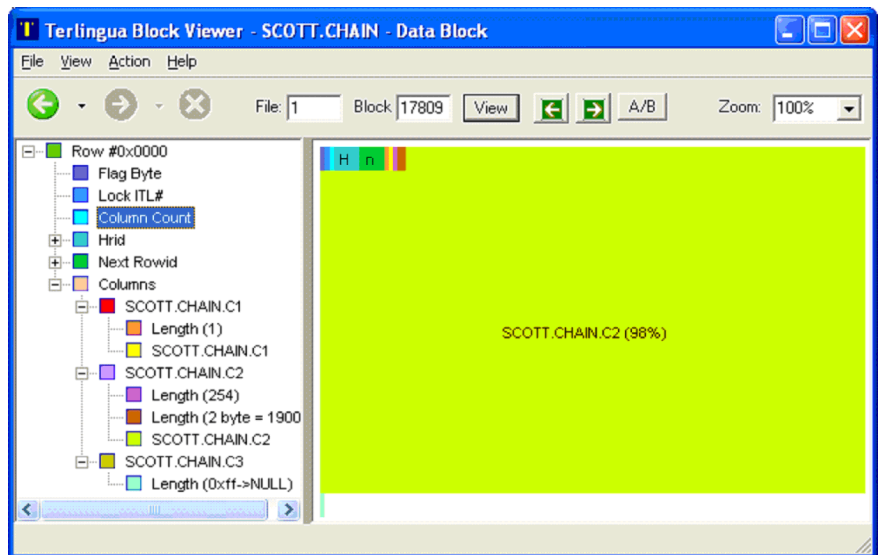
8      , lastcollen          lastl
9  from TABLE_ROWS
10 where owner = 'SCOTT'
11 and name = 'CHAIN'
12 ;
Table created.
SQL> select myrid
2      , rowflag
3      , nextrid
4      , piecesize
5      , ncols
6      , firstl
7      , lastl
8  from helper
9  start with myrid = '000047d0.0001.0001'
10 connect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

```

MYRID	ROWFLAG	NEXTRID	PIECESIZE	NCOLS	FIRSTL	LASTL
000047d0.0001.0001	--H----	00004591.0000.0001	9	0	0	0
00004591.0000.0001	----F---	0000161f.0000.0007	1921	3	1	0
0000161f.0000.0007	-----	0000161e.0000.0007	11	2	0	0
0000161e.0000.0007	-----	0000161d.0000.0007	11	2	0	0
0000161d.0000.0007	----L--	00000000.0000.0000	3	0	0	0

Although Oracle could fit the entire row now into the migrated rowid of '00004591.0000.0001', it again has taken the path of least resistance. Here is a screen shot of the row piece stored in '00004591.0000.0001'

You can see that C1 and C2 are stored in their entirety, as expected. The NULL for C3 is also here. If C3 were not left here, Oracle would mistake the first column of the next piece as C3 instead of C4. Of course, this entire row could be represented by deleting all the pieces after this one, deleting C3 from this row piece, and storing a column count of 2 in the row header. This is the way Oracle would normally represent a 9 column row where the last 7 columns are all NULL. Unfortunately, selecting the value of C9 from this row will cause 5 logical I/Os instead of the 2 needed for a simple migrated row.



The next time someone from Oracle says that Oracle does not store trailing NULL columns on disk, you can now say, "I know an example where Oracle stores 7 trailing NULL columns over 4 blocks!"