Three Types of Table Compression

Tim Gorman
24-June 2014
Agenda

• The story behind the story
• Overview of Data Compression
• Review of Oracle storage concepts
  o Internal block and row formats
  o Table clusters and row-chaining
• Table compression in Oracle
  o De-duplication compression (basic and OLTP)
  o Hybrid Columnar Compression (HCC)
  o Trailing NULL columns
The Story Behind The Story

• This isn’t a presentation about table compression
  o It ended up that way, however

• Instead, this began as a story about a solution to a specific problem
  o It was a lot of fun
  o I wanted to share it
  o But I had to fill in a lot of background before getting to the punch line
  o Which seems to make this a presentation about compression
  o Please bear with me for the next 59 minutes?
Data Compression

• White paper: *Introduction to Data Compression*
  o Guy E Blelloch, Carnegie-Mellon University, 25-Sep 2010

• Lempel Ziv (LZ) lossless compression methods
  o Simplified generic LZ algorithm
    • Divides source into fixed-length (i.e. 10- or 12-bit) *patterns*
    • Stores distinct *patterns* in lookup table
    • Replaces *patterns* in output stream with lookup hash value
  o Variations on LZ methods
    • DEFLATE: focuses on speed (zip, gzip, LZO, ZLIB, etc)
    • Layered compression: focuses on compression ratio, relatively slow, uses several layers of compression techniques (BZIP2)
Compression in Oracle

- Index compression
- Table compression
  - Basic
  - OLTP*
- RMAN backup compression*
- SecureFile (LOB) compression*
- Data Pump export compression*
- Data Guard redo transport compression*
- Hybrid Columnar compression*

* Advanced Compression Option

* Exadata / ZFS / Pillar storage only

Oracle8i
Oracle9i
Oracle10g
Oracle11gR1
Oracle11gR2
Compression in Oracle

• Index compression

• Table compression
  o Basic
  o OLTP*

• RMAN backup compression*
• SecureFile (LOB) compression*
• Data Pump export compression* * Exadata / ZFS / Pillar storage only
• Data Guard redo transport compression*

• Hybrid Columnar compression*

* Advanced Compression Option

Oracle8i
Oracle9i
Oracle10g
Oracle11gR1
Oracle11gR2
Table Compression

CREATE TABLE ...

COMPRESS [ FOR DIRECT_LOAD OPERATIONS | BASIC ]
COMPRESS FOR ALL OPERATIONS | COMPRESS FOR OLTP
COMPRESS FOR QUERY [ LOW | HIGH ]
COMPRESS FOR ARCHIVE [ LOW | HIGH ]

Key
• Oracle9i +
• Oracle11gR1
• Oracle11gR2 +
COMPRESS [ BASIC ]

- Similar in concept to LZ algorithm
  - Distinct column values stored in symbol table within block
  - Column values replaced by offset value into symbol table

- Initial Oracle table compression implementation
  - No extra cost with Enterprise Edition, not available in Standard Edition
  - Enabled with COMPRESS in 9i and 10g, COMPRESS [ FOR DIRECT_LOAD OPERATIONS ] in 11gR1, COMPRESS [ BASIC ] from 11gR2 onward
  - Available only during direct-path bulk-loading operations

- Restrictions and limitations
  - Not supported for:
    - tables with more than 255 columns
    - index-organized tables (IOTs)
    - table clusters
  - ALTER TABLE .. DROP COLUMN not supported
    - Can only SET UNUSED
COMPRESS FOR OLTP

• Advanced compression option
  o Additional licensing required in addition to Enterprise Edition
  o Enabled with COMPRESS FOR ALL OPERATIONS added in 11gR1
    • Later renamed to COMPRESS FOR OLTP in 11gR2
  o Allows all types of conventional and direct-path DML
    • Compression triggered when block FULL encountered

• Restrictions and limitations
  o Not supported for:
    • tables with more than 255 columns
    • index-organized tables (IOTs)
    • table clusters
  o Migrated chained rows will be compressed
    • But rows chained due to row-length exceeding block size will not
  o Required List of Critical Patches
    • Support note #1061366.1
Block Format

- Database block layout illustration
Block Format

- **Header**
  - Fixed header (110 bytes)
    - KCBH: Type, hdr, RDBA, SCN Base/Wrap, Seq, Flag, Chksum,(20 bytes)
    - KTBBH: Transaction Fixed Header (72 bytes)
    - KDBH: Data Header Structure (14 bytes)
    - KDBT: Table Directory Entry (4 bytes)
  - Interested Transaction List or ITL
    - XID, UBA, flag, lock, SCN Base/Wrap(23 bytes)
    - INITRANS <= number of entries <= MAXTRANS

- **Free space**
  - Header grows outward from beginning, row data grows inward from tail

- **Tail**
  - Check(4 bytes, fixed)

- **Row entries**
Row Format

- **Row-header**
  - Flag :: Lock :: column-count [ :: cluster-key-ID [ :: chained-ROWID ] ]
    - Flag, Lock, column-count = 1 byte each
    - cluster-key-ID
    - chained-ROWID (6-8 bytes)

- **Column-piece**
  - Length :: data
    - Length <= 254 bytes then 1-byte
    - Else length > 254 bytes, then 3-bytes
    - Data
      - DATE = 7 bytes
      - NUMBER = 1 byte exponent plus variable-length mantissa
      - VARCHAR2, CHAR = text
      - NULL data values
        - Non-trailing placeholder = 0xFF
        - Trailing NULLs are not stored

- **Online References**
  - Oracle11g Concepts, [http://docs.oracle.com/cd/E14072_01/server.112/e10713/logical.htm#i4894](http://docs.oracle.com/cd/E14072_01/server.112/e10713/logical.htm#i4894)
Row Format

Flag | Lock | Col Cnt | Len #1 | Payload #1 | Len #2 | Payload #2 | Len #3 | Payload #3 | Len #4 | Payload #4 | Len #5 | Payload #5 | Len #6 | Payload #6 | Len #7 | Payload #7

Cluster Tables

- Tables which share one or more columns
  - Known as cluster key columns
- Rows from clustered tables reside within the same database block
  - Physically pre-joined relational tables
Cluster Tables

```
# Example of cluster tables

tab 0, row 0, @0x3f87
  tl: 25  fb: K-H-FL--  lb: 0x0  cc: 1
  curc: 6  comc: 6  pk: 0x0040db0d.0  nk: 0x0040db0d.0
  col 0: [ 5] c4 04 04 50 24

# Example of another cluster table

tab 0, row 1, @0x3f6e
  tl: 25  fb: K-H-FL--  lb: 0x0  cc: 1
  curc: 18  comc: 18  pk: 0x0040db0d.1  nk: 0x0040db0d.1
  col 0: [ 5] c4 04 04 50 25

...several hundred lines edited out for brevity...

```

Flag byte showing "cluster key"

```
# Example of another cluster table

tab 1, row 0, @0x3a1b
  tl: 65  fb: -CH-FL--  lb: 0x0  cc: 20  cki: 0
  col 0: [ 4] c3 05 45 2c
  col 1: [ 2] c1 02
  col 2: [ 2] c1 08

# Reference back to cluster key

...several hundred lines edited out for brevity...

```
DUMP traces

• ALTER SYSTEM DUMP command
  DATAFILE [ file# | ‘file-name’ ]
  BLOCK [ block# | MIN block# BLOCK MAX block# ]

• Examples in SQL*Plus...
  SHOW PARAMETER USER_DUMP_DEST
  ALTER SESSION SET TRACEFILE_IDENTIFIER = DUMP_DBF;
  ALTER SYSTEM DUMP DATAFILE 11 BLOCK 2378;
  ALTER SYSTEM DUMP DATAFILE 741 BLOCK MIN 62078 BLOCK MAX 62085;

• Finding file# and block# for an object...
  o View DBA_EXTENTS columns FILE_ID, BLOCK_ID, and (BLOCKS-1)
    select ‘ALTER SYSTEM DUMP DATAFILE ‘ || file_id || ‘
          BLOCK MIN ‘ || block_id || ‘ BLOCK MAX ‘ || (block_id-1) || ‘;’ || ‘txt
    from dba_extents
    where segment_name = ‘T1_PK’ and segment_type = ‘INDEX’
    order by file_id, block_id;
Row Chaining

• Rows are chained for three reasons
  o Row migration
    • An UPDATE increases the length of the row so it can no longer fit
    • Only the row header is left behind, and chain-ROWID points to the location of the row in a different block
  o Row chaining across blocks
    • Row takes more space than database blocks can provide
    • Row is broken into pieces to fit, and chained across blocks
      o Chain-ROWID points to the location of the next chunk
  o Row chaining within blocks
    • Row has more than 255 columns
    • Row is broken into 255-column pieces, and chained within blocks
      o No Chain-ROWID used, row pieces are adjacent within block
Row Chaining

- Dump of example table with 300 numeric columns

---

```
tab 0, row 0, @0x3c8a
  tl: 766 fb: -----L--  lb: 0x1  cc: 255
  col 0: [ 2]  c1 10
  col 1: [ 2]  c1 11
  col 2: [ 2]  c1 12

...several hundred lines edited out for brevity...
```

```
col 253: [ 2]  c1 13
col 254: [ 2]  c1 14
```

```
tab 0, row 1, @0x3bfb
  tl: 143 fb: --H-F---  lb: 0x1  cc: 45
  nrid: 0x06c1472e.0
  col 0: [ 1]  80
  col 1: [ 2]  c1 02
  col 2: [ 2]  c1 03

...several dozen lines edited out for brevity...
```

```
col 43: [ 2]  c1 0e
col 44: [ 2]  c1 0f
```
COMPRESS

• Symbol table is implemented as a 2\textsuperscript{nd} table in the block
  o Just like a clustered tables

• Each entry in symbol table contains repetitive data values
  o One or more columns per entry
    • If two or more rows contain the same data values in the one or more contiguous columns, then this will be represented and replaced by an entry in the symbol table
COMPRESS

• Database block layout illustration
  o Distinct data values stored once in symbol table
  o Basic compression only occurs on direct-path INSERT
    • Conventional INSERT, UPDATE leave NOCOMPRESS rows
<table>
<thead>
<tr>
<th>Col</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><em>NULL</em></td>
</tr>
<tr>
<td>1</td>
<td>0x56 0x41 0x4c 0x49 0x44</td>
</tr>
<tr>
<td>2</td>
<td>0x4e</td>
</tr>
<tr>
<td>3</td>
<td>0x4e</td>
</tr>
<tr>
<td>4</td>
<td>0x4e</td>
</tr>
<tr>
<td>5</td>
<td>0x53 0x59 0x53</td>
</tr>
<tr>
<td>6</td>
<td>0x50 0x41 0x43 0x4b 0x41 0x47 0x45</td>
</tr>
<tr>
<td>7</td>
<td>0x78 0x6b 0x0b 0x02 0x16 0x01 0x1f</td>
</tr>
<tr>
<td>8</td>
<td><em>NULL</em></td>
</tr>
<tr>
<td>9</td>
<td>0x78 0x70 0x09 0x0f 0x04 0x21 0x39</td>
</tr>
<tr>
<td>10</td>
<td>0x32 0x30 0x30 0x37 0x2d 0x31 0x31 0x30 0x3a 0x33 0x30</td>
</tr>
<tr>
<td>11</td>
<td>0x44 0x42 0x4d 0x53 0x5f 0x37 0x41 0x52 0x4e 0x49 0x4e 0x47</td>
</tr>
<tr>
<td>12</td>
<td>0xc2 0x29 0x4a</td>
</tr>
</tbody>
</table>

**Actual row len**

**Row hdr**

**Non-repeated value**

**Row hdr**

**Non-repeated value**

---

*www.Delphix.com*
BASIC lifecycle

- Data lifecycle with basic compression
  - Normal DML operations as well as direct-path supported

- Time A: Empty
- Time B: Compressed on direct-path APPEND INSERT
- Time C: Rows deleted
- Time D: Rows updated
- Time E: Rows inserted
FOR OLTP lifecycle

- Data lifecycle with advanced compression
  - Normal DML operations as well as direct-path supported

Time A: Empty
Time B: Newly inserted uncompressed data
Time C: Compressed
Time D: Compressed with newly uncompressed data inserted or updated
Time E: Compressed with newly uncompressed data inserted or updated
Time F: Compressed with newly uncompressed data inserted or updated
Time G: Re-compressed
HCC

• Built in to the base database 11gR2 and above
  o But only available on Oracle storage (i.e. Exadata, ZFS, and Pillar)
• Columnar storage pivots the idea of row storage
  o Each entry represents the values of a column across many rows
  o Rather than each entry representing values in a row across many columns
• Hybrid (not true) columnar storage
  o Each set of column values does contain values not all the rows in the table
    • Covers a limited set of rows only
• Advantages:
  o Achieve greater compression ratio
    • Compressing similar types of data, rather than different types of data
    • Less metadata, more payload
  o SELECT and UPDATE operations in SQL are column oriented
• Disadvantages:
  o Relational databases manage generally transactions by row
    • Row locks exist, but column locks do not exist
  o INSERT and DELETE operations are row oriented
HCC

- Compression unit (CU) is a logical data structure
  - Header
    - Offsets and lengths of column entries
  - Bitmap
    - Identifies deleted or updated (migrated) rows
  - Column entries
    - Data values for N rows of an individual column
    - Each column entry compressed separately using specified compression algorithm (LZ0, LZIP, or BZIP2)
HCC

- Entire CU is stored as a single chained row entry
  - CU can be broken into chunks at any point, then chained across rows

Online references:

Peter Brink [http://www.slideshare.net/Enkitec/hybrid-columnar-compression-in-a-nonexadata-system](http://www.slideshare.net/Enkitec/hybrid-columnar-compression-in-a-nonexadata-system)
HCC

- When DML is performed on compressed data
  - INSERT
    - Inserted as a new row entry using OLTP compression
  - UPDATE
    - Marked as deleted in the bitmap entry of CU, then inserted as a new row entry using OLTP compression
  - DELETE
    - Marked as deleted in the bitmap entry of CU

- Deleted data is not removed, simply marked “deleted”
HCC

• More block dump output...

... data_block_dump, data header at 0x2b8bbc16e67c
===============
tsiz: 0x1f80
hsiz: 0x1c
pbl: 0x2b8bbc16e67c
    76543210
flag=-0------
ntab=1
nrow=1
frre=-1
fsbo=0x1c
fseo=0x1f
avsp=0x3
tosp=0x3
...

#tables = 1, #rows = 1

Free space begin offset and end offset only 2 bytes apart
mec_kdbh9ir2=0x0
    76543210
shcf_kdbh9ir2=----------
    76543210
flag_9ir2=--R----- Archive compression: Y
    fcls_9ir2[0]={ }
0x16:pti[0] nrow=1 offs=0
0x1a:pri[0] offs=0x1f
block_row_dump:
tab 0, row 0, @0x1f
tl: 8033 fb: ------PN lb: 0x0 cc: 1
nrid: 0x04001491.0
col 0: [8021]
Compression level: 00 (Out of range)
    Length of CU row: 8021
kdzhhrh: -----------START_CU:
        00 00 1f 55 00 4c c7 01 f3 9b 62 b5 3f 7d bc 88 88 86 83 e1 c6 4e 91 01 72 ...

Length of row = 8033
Flag “PN”: cont’d from Previous, cont’ing to Next
Length of CU chunk = 8021
HCC

- COMPRESS FOR QUERY [ LOW | HIGH ]
  - Faster decompression for more frequent query usage
  - Lower compression ratio

- COMPRESS FOR ARCHIVE [ LOW | HIGH ]
  - Slower decompression for less frequent query usage
  - Higher compression ratio

<table>
<thead>
<tr>
<th>Level</th>
<th>Algorithm</th>
<th>Expected compression ratio</th>
<th>Load Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASIC</td>
<td>Dedup</td>
<td>2:3 → 1:4 (60% → 25%)</td>
<td>Direct-path</td>
</tr>
<tr>
<td>OLTP</td>
<td>Dedup</td>
<td>2:3 → 1:4 (60% → 25%)</td>
<td>Any</td>
</tr>
<tr>
<td>QUERY LOW</td>
<td>LZ0</td>
<td>1:5 → 1:10 (20% → 10%)</td>
<td>Direct-path</td>
</tr>
<tr>
<td>QUERY HIGH</td>
<td>ZLIB</td>
<td>1:10 → 1:18 (10% → 6%)</td>
<td>Direct-path</td>
</tr>
<tr>
<td>ARCHIVE LOW</td>
<td>ZLIB</td>
<td>1:12 → 1:20 (8% → 5%)</td>
<td>Direct-path</td>
</tr>
<tr>
<td>ARCHIVE HIGH</td>
<td>BZIP2</td>
<td>1:15 → 1:30 (6% → 3%)</td>
<td>Direct-path</td>
</tr>
</tbody>
</table>
HCC

- Hybrid columnar storage has usage implications
  - **Query:**
    - "select col2" will perform two (2) LIOS
    - "select *" will perform six (6) LIOS
  - **Upshot:**
    - let developers and ad-hoc query-writers know that columnar storage implies no wildcards for columns
**DBMS_COMPRESSION**

- **Procedure GET_COMPRESSION_RATIO**
  - Assists in determining if compression is worthwhile
  - Creates a temporary table with the specified type of compression, populated with a specified number of rows, returns actual compression statistics

- **Function GET_COMPRESSION_TYPE**
  - Determines how the specified row is compressed (or not)
DECLARE
    v_blkcnt_cmp number;
    v_blkcnt_uncmp number;
    v_row_cmp number;
    v_row_uncmp number;
    v_cmp_ratio number;
    v_comptype_str varchar2(4000);
BEGIN
    DBMS_COMPRESSION.GET_COMPRESSION_RATIO(
        scratchtbsname => 'TOOLS',
        ownname => 'PROD_OWNER',
        tabname => 'ORDER_ACTIVITY',
        partname => NULL,
        comptype => DBMS_COMPRESSION.COMP_FOR_OLTP, /* QUERY_LOW|HIGH, ARCHIVE_LOW|HIGH */
        blkcnt_cmp => v_blkcnt_cmp,
        blkcnt_uncmp => v_blkcnt_uncmp,
        row_cmp => v_row_cmp,
        row_uncmp => v_row_uncmp,
        cmp_ratio => v_cmp_ratio,
        comptype_str => v_comptype_str);
    dbms_output.put_line('Blocks compressed: ' || v_blkcnt_cmp);
    dbms_output.put_line('Blocks uncompressed: ' || v_blkcnt_uncmp);
    dbms_output.put_line('Rows per block compressed: ' || v_row_cmp);
    dbms_output.put_line('Rows per block uncompressed: ' || v_row_uncmp);
    dbms_output.put_line('Compression Ratio: ' || v_cmp_ratio);
    dbms_output.put_line('Comment: ' || v_comptype_str);
END;
/

GET_COMPRESSION_RATIO

www.Delphix.com  KScope 2014  24-June 2014  33
Trailing NULLCOLs

• A form of compression that exists in all current versions of Oracle...
  o Takes advantage of how columns are stored within rows
    • Row
      o Row-header :: column-piece [ :: column-piece ... ]
      o Column-piece
        • Non-NULL data values
          o Length :: data
        • NULL data values
          o Non-trailing placeholder = 0xFF
          o Trailing NULL values are not stored
Trailing NULLCOLs

• A form of compression that exists in all current versions of Oracle...
  • Takes advantage of how columns are stored within rows
    • Row
      • Row-header :: column-piece [ :: column-piece ... ]
      • Column-piece
        • Non-NULL data values
          • Length :: data
        • NULL data values
          • Non-trailing placeholder = 0xFF
            • Trailing NULL values are not stored
Trailing NULLCOLs

• **Case study**
  o Oracle’s Demantra product
    • Application for demand management, sales and operations planning, projections, and what-if analysis
  o Central fact table is named SALES_DATA
    • Frequently customized with additional columns
  o SALES_DATA had over 750 columns and 250m rows
    • All analytic queries performed FULL table scans on SALES_DATA, over and over and over and over...
  o It turned out that the SALES_DATA table had only 40-50 out of 750 columns populated on average
    • DBMS_STATS showed average row length of 766 bytes
Trailing NULLCOLs

• Compress?
  o Table in question had more than 750 columns
    • Rules out BASIC compression
  o Database was 10gR2
    • Rules out OLTP and HCC compression

• Solution?
  o Rebuild SALES_DATA with columns ordered by NUM_NULLS descending
    • Then load all rows into the new table
    • Average row length dropped from 766 to 102 bytes
      o 7:1 compression ratio
    • Total table size dropped from 190 Gb to about 26 Gb
Trailing NULLCOLs

• OK, but we need a way to determine if a table would benefit from such a rebuild
  o Without having to test it

• Procedure CARL (Calculate Average Row Length)
  1. Queries rows and calculates current average row length
  2. Sorts columns by NUM_NULLS DESC from DBA_TAB_COLUMNS view
  3. Recalculate average row length

• Download from [http://EvDBT.com/scripts/](http://EvDBT.com/scripts/)
  o Script “carl.sql”
  o Prerequisites...
    • CARL relies on good column statistics
    • Uses DBMS_OUTPUT package to output results
      o Enable SERVEROUTPUT ON in SQL*Plus
Points to ponder...

• What happens when you attempt to access HCC data on non-Oracle (a.k.a. non-HCC-enabled) storage?
  o ORA-64307 “hybrid columnar compression is not supported for tablespaces on this storage type”

• Luis Moreno Campos’ blog
  o [link](http://ocpdba.wordpress.com/2011/05/06/recover-hcc-compressed-tables-to-non-exadata-storage/)
  o Testing with RMAN, moving HCC data from Exadata to non-Oracle storage
  o RMAN backup and restore operations are successful. Why?
  o INSERTs are successful. Why?
  o SELECTs, UPDATEs, and DELETEs fail. Why?
  o ALTER TABLE ... MOVE is successful. Why?
Summary

• Multiple ways to compress table data
  o Two ways are provided and supported by Oracle
    • in certain versions, some need patching, each has prerequisites
  o One way is possible by understanding how Oracle stores data
    • Not useful for all situations, but ya never know...

• Compression will *usually* improve query performance
  o Understand each and every type of compression and how they work
Contact and References

• Tim Gorman
  o Email: Tim.Gorman@Delphix.com
  o Twitter: TimothyJGorman
  o Blog: EvDBT.com

• References
  o Jonathan Lewis -
    http://jonathanlewis.wordpress.com/2012/07/20/compression_units/
    http://jonathanlewis.wordpress.com/2011/10/04/hcc/
  o Peter Brink -
    http://www.slideshare.net/Enkitec/hybrid-columnar-compression-in-a-nonexadata-system
  o Graham Thornton -
  o Uwe Hesse
http://www.KScope14.com

@ODTUG  #KScope14

Conference for Hyperion EPM, Oracle developers, and DBAs