ORACLE ADVANCED COMPRESSION BEST PRACTICES

Oracle Advanced Compression Proof-of-Concept (POC) Insights and Best Practices

FEATURES TYPICALLY EVALUATED

- **Advanced Row Compression**
  Enables table data to be compressed during all types of data manipulation operations
- **Index Key Compression and/or Advanced Index Compression**
  Reduces the size of all supported unique and non-unique indexes
- **Backup Data Compression**
  Compression for backup data when using Oracle Recovery Manager (RMAN) or Oracle DataPump
- **Advanced LOB Compression and Deduplication**
  Provides compression and deduplication for LOBS managed by SecureFiles

Oracle Advanced Compression

The massive growth in data volumes being experienced by enterprises introduces significant challenges. Companies must quickly adapt to the changing business landscape without impacting the bottom line. IT managers need to efficiently manage their existing infrastructure to control costs, yet continue to deliver extraordinary application performance.

Oracle Advanced Compression, and Oracle Database, together provide a robust set of compression, performance and data storage optimization capabilities that enable IT managers to succeed in this complex environment.

Whether it is a cloud or an on-premises Oracle database deployment, Oracle Advanced Compression can deliver robust compression across different environments with no changes in applications. Benefits from Oracle Advanced Compression include smaller database storage footprint, savings in backups and improved system performance.

What is Useful to Know for Your Compression POC

This document isn't meant to be a step-by-step guide to performing a compression POC. Instead, this document is intended to provide some best practices learned from customer POC's and to provide insights to help plan your compression POC, as well as help you understand the results of your POC.

Enabling Advanced Row Compression

For new tables and partitions, enabling Advanced Row Compression is easy: simply CREATE the table or partition and specify "ROW STORE COMPRESS ADVANCED". See the example below:

```
CREATE TABLE emp (emp_id NUMBER, first_name VARCHAR2(128),
last_name VARCHAR2(128)) ROW STORE COMPRESS ADVANCED;
```

There are numerous ways to enable Advanced Row Compression for existing tables. While a complete discussion of each method is beyond the scope of this document, this document does provide an overview of the methods typically used by customers. For more information on these methods, please see the Oracle Database documentation.

ALTER TABLE … ROW STORE COMPRESS ADVANCED

This approach will enable Advanced Row Compression for all future DML -- however, the existing data in the table will remain uncompressed.

ORACLE

18c ORACLE Database

More Information…

Advanced Compression OTN page: http://www.oracle.com/technetwork/database/options/compression/overview/index.html

Oracle.com Storage Optimization page: https://www.oracle.com/database/storage-management/index.html

Storage Optimization Blog: https://blogs.oracle.com/DBStorage/

Online Redefinition (DBMS_REDEFINITION)

This approach will enable Advanced Row Compression for future DML and also compress existing data. Using DBMS_REDEFINITION keeps the table online for both read/write activity during the migration. Run DBMS_REDEFINITION in parallel for best performance.

Online redefinition will clone the indexes to the interim table during the operation. All the cloned indexes are incrementally maintained during the sync (refresh) operation so there is no interruption in the use of the indexes during, or after, the online redefinition.

The only exception is when online redefinition is used for redefining a partition -- any global indexes are invalidated and need to be rebuilt after the online redefinition.

ALTER TABLE … MOVE ROW STORE COMPRESS ADVANCED

This approach will enable Advanced Row Compression for future DML and also compress existing data. While the table is being moved it is online for read activity but has an exclusive (X) lock – so all DML will be blocked until the move command completes. Run ALTER TABLE…MOVE in parallel for best performance.

ALTER TABLE… MOVE will invalidate any indexes on the partition or table; those indexes will need to be rebuilt after the ALTER TABLE… MOVE. For partition moves, the use of ALTER TABLE… MOVE PARTITION with the UPDATE INDEXES clause will maintain indexes (it places an exclusive (X) lock so all DML will be blocked until the move command completes) – not available for non-partitioned tables.

The ALTER TABLE... MOVE statement allows you to relocate data of a non-partitioned table, or of a partition of a partitioned table, into a new segment, and optionally into a different tablespace. ALTER TABLE…MOVE ROW STORE COMPRESS ADVANCED compresses the data by creating new extents for the compressed data in the tablespace being moved to -- it is important to note that the positioning of the new segment can be anywhere within the data file, not necessarily at the tail of the file or head of the file. When the original segment is released, depending on the location of the extents, it may or may not be possible to shrink the data file.

ALTER TABLE … MOVE TABLE/PARTITION/SUBPARTITION … ONLINE

This approach will enable Advanced Row Compression for future DML and also compress existing data. ALTER TABLE ... MOVE TABLE/PARTITION/SUBPARTITION … **ONLINE** allows DML operations to continue to run uninterrupted on the table, partition or subpartition that is being moved.

Indexes are maintained during the move operation, so a manual index rebuild is not required. Starting with Oracle Database 12.2 on Oracle Cloud, move tables online as well as partitions and subpartitions.

When Advanced Row Compression Occurs

Advanced Row Compression uses a unique compression algorithm specifically designed to work with OLTP and Data Warehouse applications. The algorithm works by eliminating duplicate values within a database block, even across multiple columns. Compressed blocks contain a structure called a symbol table that maintains compression metadata.

When a block is compressed, duplicate values are eliminated by first adding a single copy of the duplicate value to the symbol table. Each duplicate value is then replaced by a short reference to the appropriate entry in the symbol table.

While the compression benefits and compression techniques used are similar, within the database different compression types can be invoked when using Advanced Row Compression. Below are some examples of the different compression types, understanding when these different compression types are used will help when analyzing POC results.

Insert Direct Load Compression

Performed when data is inserted using the direct-path load mechanisms, such as insert with an append hint or using SQL*Loader. In this case, the data is being inserted above the segment high water mark (a virtual last used block marker for a segment) and can be written out to data blocks very efficiently. The compression engine has a large volume of rows to work with and is able to buffer, compress and write out compressed rows to the data block(s). As a result, the space savings are immediate. Rows are never written in an uncompressed format for Insert Direct Load compression.

Recursive Compression

Invoked on conventional DML operations such as single row or array inserts and updates. This compression type writes out the rows in uncompressed format, and when the data block reaches an internal block fullness threshold compression is invoked. Under such a scenario, Oracle is able to compress the data block in a recursive transaction which is committed immediately after compression.

The space saved due to compression is immediately released and can be used by any additional transactions. Compression is triggered by the user DML operation (user transaction), but actual compression of data happens in a recursive transaction, the fate of compression is therefore not tied to the fate of the user's transaction.

Important MOS Notes for Oracle Compression:

Master note:
DOC ID: 1223705.1

Critical patches:
DOC ID: 1061366.1

See if rows are compressed:
DOC ID: 1477918.1

RMAN compression:
DOC ID: 563427.1

Index Key Compression:
DOC ID: 601690.1

Data Guard Redo Transport Compression:
DOC ID: 729551.1

Direct-Path versus Conventional-Path Inserts

Performing bulk-load operations, and choosing either direct-path or conventional-path methods (see here), can have a significant influence in regards to load performance.

Users performing bulk-load insert operations may see slower insert performance, particularly if they are inserting a large number of rows using a conventional-path load. The reason why conventional-path loads may be slower, for a large number of rows, is that as the new rows are inserted into existing compressed blocks the inserts are performed uncompressed, then as additional inserts are performed on the same block, and the block begins to fill up, the internal threshold will be met and the block will be compressed (see the recursive compression discussion above for more details). If additional space is freed up after the compression then inserts will again be performed on the block, thus leading to compression again, possibly multiple more times for the same block, during the same conventional-path load operation.

This means that when using conventional-path inserts it is possible that the same block will be compressed multiple times during the same operation – consuming CPU resources and time. If the workload is dominated by conventional-path inserts, then it's likely there will be more I/O: when a block is recompressed repeatedly as part of the Advanced Row Compression algorithm (compared to direct-path loads).

Direct-path load operations are preferred when operating on larger numbers of rows since, unlike conventional-path loads, direct-path loads are done above the high-water mark, so blocks are completely filled and compressed only once, and then written to disk. This

streamlines the bulk inserts and avoids the multiple compressions of the same block which is possible when performing bulk inserts using conventional-path loads.

Later in this document we'll discuss how to use Automatic Workload Repository reports (AWR) to determine whether bulk inserts are using direct-path or conventional-path methods.

Index Key Compression

If used correctly, Index Key Compression (see here) has the potential to substantially reduce the overall size of indexes. It helps both multi-column unique indexes and non-unique indexes alike and is also one of the most critical index optimization options available.

Index Key Compression can be used to compress portions of the key values in an index segment (or Index Organized Table (IOT)), by reducing the storage inefficiencies of storing repeating values. It compresses the data by splitting the index key into two parts; the leading group of columns, called the prefix entry (which are potentially shared across multiple key values), and the suffix columns (which is unique to every index key). As the prefixes are potentially shared across multiple keys in a block, these can be stored once and then shared across multiple suffix entries, resulting in the index data being compressed.

Index Key compression is done in the leaf blocks of a B-Tree index. The keys are compressed locally within an index leaf block (both the prefix and suffix entries are stored within same block). Suffix entries form the compressed representation of the index key. Each one of these compressed rows refers to the corresponding prefix, which is stored in the same block. By storing the prefixes and suffixes locally in the same block, each index block is self-contained and no additional block IO is required to construct the complete key

Enabling Index Key Compression:

This can be achieved easily by specifying the COMPRESS option for the index. New indexes can be automatically created as compressed, or the existing indexes can be rebuilt compressed.

```
CREATE INDEX idxname ON tabname(col1, col2, col3) COMPRESS;
```

By default the prefix consists of all indexed columns for non-unique indexes, and all indexed columns excluding the last one for unique indexes. Alternatively you can also specify the prefix length, which is the number of columns in the prefix.

```
CREATE INDEX idxname ON tabname(col1, col2, col3) COMPRESS
[<prefix_col_length>]
```

The <prefix_col_length> after the COMPRESS keyword denotes how many columns to compress. The default (and the maximum) prefix length for a non-unique index is the number of key columns, and the maximum prefix length for a unique index is the number of key columns minus one. If you specify a prefix length of 1 in the above example, then the prefix is only (col1) and the suffix is (col2, col3).

Prefix entries are written to the index block only if the index block does not already contain that prefix. They are available for sharing across multiple suffix entries immediately after being written and remain available until the last referencing suffix entry is deleted and cleaned out from the block. Although key compression reduces the storage requirements of an index by sharing parts of keys across multiple entries, there is a small CPU overhead to reconstruct the key column values during index lookup or scans (which is minimized by keeping the prefixes locally in the block).

Index Key compression can be useful in many different scenarios, such as:

- In case of a non-unique index where ROWID is appended to make the key unique. If such an index is compressed using key compression, the duplicate key is stored as a prefix entry in the index block without the ROWID. The remaining rows become suffix entries consisting of only the ROWID
- In case of unique multicolumn index (key compression is not possible for unique single column index because there is a unique piece but there are no prefix grouping pieces to share)
- In case of Index Organized Tables, the same considerations as unique multicolumn indexes apply

Index Key Compression can be very beneficial when the prefix columns of an index are repeated many times within a leaf block. However, if the leading columns are very selective or if there are not many repeated values for the prefix columns, then Index Key Compression won't be beneficial. In these scenarios, Oracle still creates prefix entries storing all unique combinations of compressed column values within a leaf block. The index rows will refer to the prefix entry, which is hardly shared (if at all) by other index rows. So, it's possible that compression in these cases is not beneficial, and could end up increasing the index size due to the overhead of storing all of the prefix entries.

For index compression to be beneficial, you should ensure low cardinality columns are the leading columns in a concatenated index. Otherwise you run the risk of getting negative compression (e.g. leaf blocks can no longer store as many keys as their non-compressed counterparts). There is also no point in compressing a single column unique index or compressing every column in a concatenated, multi-column unique index. In these cases compression will result in an index structure that increases in size rather than decreasing (negative compression) due to all of the overheads associated with having prefix entries for each and every index row.

The key to getting good index compression is identifying which indexes will benefit from it and how to specify <prefix_col_length> correctly. For help, please see MOS note: "*How to find the optimal Index Key COMPRESS level for Indexes* (**Doc ID 601690.1**)"

Oracle does try and protect you under certain cases. For example, you are not able to create a compressed index on a single column unique index. Nor are you allowed to specify a prefix length equal to or greater than the number of columns for a unique index, (remember the default prefix length for a unique concatenated index is the number of indexed columns minus one) etc.

Advanced Index Compression

Advanced Index Compression, part of Oracle Advanced Compression with Oracle Database 18c, helps automate index compression so that a DBA is no longer required to

specify the number of prefix columns to consider for compression (as is required with Index Key Compression).

The correct and most optimal numbers of prefix columns will be computed automatically on block-by-block basis, and thus produce the best compression ratio possible. Using Advanced Index Compression it is possible to have different index leaf blocks compressed differently (with different prefix column count) or not be compressed at all, if there are no repeating prefixes.

Enabling Advanced Index Compression:

Advanced Index Compression can be enabled by specifying the COMPRESS ADVANCED sub-clause of the CREATE/ALTER INDEX clause. New indexes can be automatically created as compressed, or existing indexes can be rebuilt compressed.

CREATE INDEX idxname ON tabname(col1, col2, col3) **COMPRESS ADVANCED LOW**;

Note that there is no need to provide the number of columns in the prefix entries with Advanced Index Compression as this will be computed automatically for every leaf block.

Advanced Index Compression works well on all supported indexes, including the ones that were not good candidates for prefix key compression. Creating an index using Advanced Index Compression reduces the size of all unique and non-unique indexes (or at least doesn't increase the size due to negative compression) and at the same time improves the compression ratio significantly while still providing efficient access to the indexes.

Advanced Index Compression has following limitations:

- Advanced Index Compression is not supported on Bitmap Indexes
- You cannot compress your Index Organized Tables (IOTs) with Advanced Index Compression (use Index Key Compression)
- You cannot compress your Functional Indexes with Advanced Index Compression

With Advanced Index Compression you can simply enable compression for your B-Tree indexes and Oracle will automatically compress every index leaf block when beneficial, automatically taking care of computing the optimal prefix column length for every block. This makes index compression truly local at a block level, where both the compression prefix table, as well as the decision on how to compress the leaf block, is made locally for every block with the aim to achieve the most optimal compression ratio for the entire index segment.

What to Consider for Before POC Testing Begins

As part of the Proof-of-Concept pre-test planning, make note and take action (as needed) on the following Oracle compression suggested best practices:

- Upgrade to the latest release (or apply any critical patches to the current release) (See MOS note: ***List of Critical Patches Required For Oracle 11g Table Compression (Doc ID 1061366.1))***
- Define Success Criteria for the POC (data, index and backup storage reduction, query/insert/update performance, bulk load operations performance, application performance and etc….)
- If the POC is being performed using Oracle E-Business Suite or SAP, see these Oracle Advanced Compression White Papers for additional information. Oracle E-Business Suite/SAP
- Ensure compressed columns have no "long" data types – this data type isn't supported by Advanced Row compression
- Ensure compressed tables/partitions have less than 255 columns (this limit was removed in Oracle Database 12*c*)
- Although CPU overhead is typically minimal, implementing Advanced Row Compression and Index Compression is ideal on systems with available CPU cycles, as compression will have additional, although minor overhead for some DML operations
- The best test environment for each compression capability is where you can most closely duplicate the production environment– this will provide the most realistic (pre- and post- compression) performance and functionality comparisons
- The general recommendation is to compress all of the application related tables in the database with one exception: if the table is used as a queue. That is, if the rows are inserted into the table, then later most or all of the rows are deleted, then more rows are inserted and then again deleted. This type of activity is not a good use case for compression due to the overhead to constantly compress rows that are transient in nature
- Index Key Compression can be very beneficial when the prefix columns of an index are repeated many times within a leaf block. However, if the leading columns are very selective, or if there are not many repeated values for the prefix columns, then Index Key Compression would not be beneficial
- Advanced Row Compression works well with tablespace-level encryption. Tables are compressed before encryption, so the compression ratio isn't affected by the encryption. With column-level encryption, the encryption is done before compression, which will negatively impact the compression ratio

For more information see the Index Compression and Advanced Compression Technical White Papers for Oracle Database 18c.

Direct-Path and Conventional-Path Loads and AWR

If during POC testing you are unsure if bulk loads are using direct-path or conventional-path load methods you can utilize these suggested steps (with AWR) to determine the amount of compression occurring during the SQL operation. See this blog for more

information: https://blogs.oracle.com/DBStorage/entry/critical_statistics_for_advanced_row

Determining Conventional-Path Load Compressions

AWR has an "Instance Activity Stats" section that will list the statistics associated with the total number of positive compressions (**HSC OLTP positive compression**) and the total number of negative compressions (**HSC OLTP negative compression**). Adding these two statistics will give you the total number of attempted compressions (re-compressions or otherwise).

> **HSC OLTP positive compression** + **HSC OLTP negative compression** = Total number of attempted compressions and re-compressions (non-direct load)

Determining Direct-Path Load Compressions

When performing bulk loads using direct-path methods such as "insert append" the data is organized into data blocks and compressed in memory, this means that the bulk load data is compressed only once. The data blocks are filled to the point specified by the tables PCTFREE setting -- the default setting for PCTFREE in Oracle Database is 10% (PCTFREE allows space to be reserved on the data blocks for possible growth during SQL UPDATE operations).

For block compressions above the High Water Mark (HWM), such as in Create Table as Select (CTAS) or insert append cases, there is a statistic called **HSC IDL Compressed Blocks**.

> **HSC IDL Compressed Blocks =** Block compressions above the HWM (such as in CTAS or insert append)

If you only see values for **HSC OLTP positive Compression** and **HSC OLTP negative compression** statistics and no/few values for the **HSC IDL Compressed Blocks** statistic**,** then all the compression occurring is from conventional path operations (in particular, see how many compressions are occurring per second).

If possible and feasible, you should consider modifying bulk inserts so that direct-path loading is performed instead of conventional-path loads for the same operation(s). In doing so you should see a larger value for the **HSC IDL Compressed Blocks** statistic. If there is no statistic labeled "**HSC IDL Compressed Blocks**" this means that there was no block compression above the HWM.

RMAN Compression

Due to RMAN's tight integration with Oracle Database, backup data is compressed before it is written to disk or tape and doesn't need to be uncompressed before recovery –

providing a reduction in storage costs and a potentially large reduction in backup and restore times.

RMAN Basic compression delivers a very good compression ratio, but can sometimes be CPU intensive and CPU availability can be a limiting factor in the performance of backups and restores.

There are three levels of RMAN Compression with Advanced Compression: LOW, MEDIUM, and HIGH. The amount of storage savings increases from LOW to HIGH, while potentially consuming more CPU resources. LOW / MEDIUM / HIGH compression is designed to deliver varying levels of compression while typically using less CPU than RMAN Basic Compression.

Generally speaking, the three levels can be categorized as such:

HIGH - Best suited for backups over slower networks where the limiting factor is network speed

MEDIUM -Recommended for most environments. Good combination of compression ratios and speed

LOW -Least impact on backup throughput and suited for environments where CPU resources are the limiting factor

If you are I/O-limited but have idle CPU, then HIGH could work best, as it uses more CPU, but saves the most space and thus gives the biggest decrease in the number of I/O's required to write the backup files. On the other hand, if you are CPU-limited, then LOW or MEDIUM probably makes more sense - less CPU is used, and about 80% of the space savings will typically be realized (compared to the Basic compression included with RMAN).

More about Compression, Performance and Compression Ratio's

Before performing a POC users sometime speculate that the overhead of decompression could impact query performance. However, in practice, this is typically unlikely.

Advanced Row and Index Key/Advanced Index compressed blocks are never "decompressed" at the block level, and for most queries, individual rows aren't decompressed either: most queries can operate directly on the compressed format in the database blocks in memory and most query predicates operate directly on compressed data formats, and only values required for the later stages of the query are decompressed.

There is typically not an increase in overhead for queries on compressed data/indexes, and there is usually a decrease because of the reduction in I/O to query a given amount of user data: if the data is compressed at a 3x ratio, for example, then it takes only 1/3 the amount of I/O to read that data from disk and into the buffer cache when using compression. While it is true that there can be a few "extra" instruction cycles to dereference pointers inside compressed data blocks to extract column values, this is usually more than offset by the reduction in I/O.

The compression ratio of a particular table/partition is primarily related to the amount of duplication that exists, at the block level, for that table or partition. The higher the amount of duplication then the higher the compression ratio, and the more unique the data, then the lower the compression ratio. Generally speaking, if the data is highly unique, then it is very possible that the table/partition may not compress well or at all.

There are a few things you can try to possibly increase the compression ratio for a particular table. As usual, you should test any changes, using your data, applications and systems, to determine the impact any such changes will have in your environment.

Sorting data

It may be possible to improve a table's compression ratio by presorting the data when it's loaded. You will have to decide which column(s) to sort on based on the cardinality of the data in each column: if you can sort on a column that has a small number of distinct values, that could produce better compression ratios. But presorting will require additional preparation of the data before loading - you will need to weigh that additional time versus any compression ratio gain

Use SecureFiles for LOB storage (see here for more information about using SecureFiles)

It is usually possible to improve a table's compression ratio by moving unstructured data to SecureFiles (and using Advanced LOB Compression) instead of storing unstructured data inline. Advanced Row Compression depends upon deduplication to reduce the size of a block. With unstructured data stored inline, it is unlikely that a duplicate of that unstructured data will be in the same block, and this means that the unstructured data in the block, which can often be quite large, won't be compressed. This can lead to lower than expected overall compression ratios for the table. Advanced LOB Compression, however, uses a different compression algorithm and can often compress unstructured data stored in SecureFiles that isn't able to be compressed when stored inline.

Oracle detects if SecureFile data is compressible and will compress using industry standard compression algorithms. If the compression does not yield any savings or if the data is already compressed, SecureFiles will turn off compression for such LOBs.

There are three levels of Advanced LOB Compression: LOW, MEDIUM, and HIGH. By default, Advanced LOB Compression uses the MEDIUM level, which typically provides good compression with a modest CPU overhead of 3-5%. Advanced LOB Compression LOW is optimized for high performance. Advanced LOB Compression LOW maintains about 80% of the compression achieved through MEDIUM, while utilizing 3x less CPU. Finally, Advanced LOB Compression HIGH achieves the highest storage savings but incurs the most CPU overhead.

What does the Typical Compression POC Look Like?

Compression Benefits

- Reduces database storage requirements and associated costs
- Typically improves query performance
- Reduces time and storage required for RMAN backup and recovery
- Data in memory remains in compressed format
- Compresses structured, unstructured, index, backup, network and Data Guard redo log transport data
- Cascades storage savings throughout the data center
- Reduces the size of all supported unique and non-unique indexes -- while still providing efficient access to the indexes
- Can be used with any type of application without application changes
- Reduces the size of the session data unit (SDU) transmitted over a data connection
- Reduces the cost of storing and accessing historical data

As indicated earlier, it is important to note that the best test environment for each compression feature is where you can most closely duplicate the production environment– this will provide the most realistic (pre- and post- compression) performance and functionality comparisons.

Generally, the Advanced Compression option features that are generally tested during a compression POC includes:

- Advanced Row Compression
- Index Key Compression* / Advanced Index Compression
- RMAN Compression
- Advanced LOB Compression/Deduplication
- Data Guard Redo Transport Compression

* Index Key Compression is included with Oracle Database Enterprise Edition and does not require Oracle Advanced Compression

While Advanced Compression does include numerous other features, the above are the features most typically included in a POC. You may choose to include other Advanced Compression features or not include some of these features.

In terms of the actual POC, customers often indicate the following:

- Before testing estimate compression ratios (storage reduction) for structured data, indexes, backups and unstructured data. With Oracle Database 18c, Compression Advisor (see below) can be used to estimate Advanced Row Compression, Advanced Index Compression and Advanced LOB Compression compression ratios
- Use testing to identify performance improvements and any possible performance impact from compression. This is typically determined by running your applications, using your data on test platforms (similar to production hardware) and profiling the performance before and after compression. Ideally the application testing would include: application queries, bulk load operations using both conventional and direct-path loads, single row DML (i.e. conventional insert, update and delete operations) and RMAN backups *(See the customer POC results White Paper (on the left) as an example)*
- While the general suggestion is to compress all tables (excluding those noted on page 7) some organizations instead choose only to compress the largest tables that account for approximately 80%+ of their data storage requirements
- Using Index Key Compression requires running ANALYZE INDEX to obtain the prefix column count. Advanced Index Compression does not require running Analyze Index
- MOS note Doc ID 729551.1 is useful for information about estimating compression savings when using Data Guard Redo Transport Compression.

Customer POC Testing Results:

Global 500 Customer:
http://www.oracle.com/ocom/groups/public/@otn/documents/webcontent/396808.pdf

- If available, Oracle's Real Application Testing (RAT) product can be a useful tool for a compression POC
- See **Appendix A** below, for a simple example of what a compression POC may look like as illustrated in a multi-step process.

BEST PRACTICE:

Oracle's free Compression Advisor to use a very useful tool for estimating compression ratios, information about advisor is available on OTN at: http://www.oracle.com/technetwork/database/options/compression/downloads/index.html

Compression Advisor

Compression Advisor is a PL/SQL package (DBMS_COMPRESSION) that is used to estimate potential storage savings (before implementing Oracle Advanced Compression) based on analysis of a sample of data, and provides a good estimate of the actual results that may be obtained after implementing Oracle Advanced Compression.

Compression Advisor, which supports Oracle Database 9*i* Release 2 through 11g Release 1, is available for free on the Oracle Technology Network website. Compression Advisor is included with Oracle Database 11g Release 2 and above.

Appendix A – Example of Compression POC as a Multi-Step Process

The Multi-Step POC compression process below is provided strictly as a simple example for what a compression POC may include. Not all POC's will use every step and many POC's will include steps not indicated in the general outline below.

1. **Apply all Relevant Patches (optionally upgrade to latest release)**
    a. Upgrade to latest release if applicable
    b. Apply patches
2. **Define Success Criteria**
    a. Database performance
    b. Database size
    c. Backup space reduction
    d. Backup time/restore time
    e. Application performance
    f. Data Guard (if applicable)
    g. Data Pump Compression (if applicable)
3. **Compression Advisor -- DBMS_COMPRESSION**
    a. Obtain compression ratio estimates for Data/Indexes
    b. Determine the overall list of tables/indexes to be compressed
4. **Baseline Before Compression In Test Environment: Production Workload/Data**
    a. Gather database performance data (including bulk load operations, queries, inserts/updates and etc…)
    b. Gather backup/restore times
    c. Gather Data Guard performance data (if applicable)
    d. Gather database size for tables/indexes
    e. Gather backup size
    f. Gather Data Pump file size
    g. Gather AWR reports
5. **Implement Compression In Test Environment**
    a. Compress all candidate tables/indexes identified using preferred method (online/offline)
    b. Perform bulk load operations and compare against baseline
    c. Run SQL statements (query/insert/update) and compare against baseline
    d. Perform SQL tuning adjustments for non-performing queries (if any)
    e. Run production workload and verify performance
    f. Gather AWR Reports and compare to baseline
6. **Prepare for Production Cutover**
    a. Lessons Learned

b. Document all the benefits and issues/resolutions encountered during POC
c. Define cutover plan

CONTACT US

For more information about Advanced Compression, visit oracle.com or call +1.800.ORACLE1 to speak to an Oracle representative.

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Hardware and Software, Engineered to Work Together