Oracle Advanced Compression Proof-of-Concept Guidelines, Insights and Best Practices

July, 2024, Version 23ai

Public

Purpose statement

This document provides an overview of features and enhancements included in release 23ai. It is intended solely to help you assess the business benefits of upgrading to 23ai and planning for the implementation and upgrade of the product features described.

Disclaimer

Table of contents

About This Document

This document is not a step-by-step guide to performing a compression proof-of-concept. Instead, this document provides compression guidelines/best practices learned from users testing, as well as other insights to you help plan your compression proof-of-concept, as well as help you understand the results of your proof-of-concept.

Introduction

The massive growth in data volumes experienced by enterprises introduces significant challenges. Companies must quickly adapt to the changing business landscape without influencing the bottom line. IT managers need to efficiently manage their existing infrastructure to control costs yet continue to deliver application query 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 on-premise 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, time and storage savings in backups and improved query performance.

FEATURES TYPICALLY EVALUATED

- **Advanced Row Compression**
  Enables table data to be compressed during all types of data manipulation operations
- **Advanced Index Compression**
  Reduces the size of all supported unique and non-unique indexes
- **RMAN Backup Compression**
  Compresses backup data when using Oracle Recovery Manager (RMAN)
- **Advanced LOB Compression**
  Compresses SecureFiles LOBs

Advanced Row Compression

Advanced Row Compression maintains compression during all types of data manipulation operations, including conventional DML such as INSERT and UPDATE. In addition, Advanced Row Compression minimizes the overhead of write operations on compressed data, making it suitable for transactional / OLTP environments as well as data warehouses, extending the benefits of compression to all application workloads.

Advanced Row Compression uses a unique compression algorithm specifically designed to work with OLTP/DW 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.

Through this innovative design, compressed data is self-contained within the database block, as the metadata used to translate compressed data into its original state is stored in the block header. When compared with competing compression algorithms that maintain a global database symbol table, Oracle's approach offers significant benefits by not introducing additional I/O (needed with a global symbol table) when accessing compressed data.

The compression ratio achieved in each environment depends on the data being compressed, specifically the cardinality of the data. In general, organizations can expect to reduce their storage space consumption by a factor of 2x to 4x by using Advanced Row Compression and/or Advanced Index Compression. That is, the amount of space consumed by uncompressed data/indexes will be two to four times larger than that of the compressed data.

But the benefits of Advanced Row Compression go beyond just on-disk storage savings. A key query performance advantage is Oracle's ability to read compressed blocks (data and indexes) directly, in memory, without uncompressing the blocks. This can help improve query performance due to the reduction in I/O, and the reduction in system calls related to the I/O operations. Further, the buffer cache becomes more efficient by storing more data without having to add memory.

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.

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.

Online Redefinition (DBMS_REDEFINITION)

This approach will enable Advanced Row Compression for future DML and will 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 rebuilt after the online redefinition.

ALTER TABLE … **MOVE ROW STORE COMPRESS ADVANCED**

This approach will enable Advanced Row Compression for future DML and will 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 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 will compress existing data. ALTER TABLE ... MOVE TABLE, PARTITION or SUBPARTITION … **ONLINE** allows DML operations to continue to run uninterrupted on the table, partition or subpartition being moved. Indexes are maintained during the move operation, so a manual index rebuild is not required.

Important Documentation Note

Please see the current Oracle Database documentation for additional details, usage examples and restrictions regarding the operations discussed above.

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 proof-of-concept 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 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 can 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 can 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.

Direct-Path Versus Conventional-Path Bulk Loads

Performing bulk-load operations and choosing either direct-path or conventional-path methods can have a significant influence regarding load performance.

Users performing bulk-load insert operations may see slower insert performance, particularly if they are inserting many rows using a conventional-path load.

The reason why conventional-path loads may be slower, for many 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 is 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 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.

AWR and Direct-Path/Conventional-Path Bulk Loads

If during proof-of-concept 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.

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 (conventional-path 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 (direct-path load 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.

Advanced Index Compression

Advanced Index Compression, a feature of Advanced Compression, 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).

Advanced Index Compression is an enabling technology for multiple compression levels – LOW and HIGH. Average compression ratios can range from 2x to 5x depending on which compression level is implemented. With substantial storage savings from Advanced Index Compression, IT managers can often reduce the need to purchase new storage.

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/HIGH**;

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 does not 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
- Functional Indexes are not supported 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.

Note that Index-Organized Tables (IOT's) are essentially indexes, so they cannot be compressed with Advanced Row Compression or Basic Compression. However, IOT's can be compressed with Advanced Index Compression LOW.

RMAN Backup Compression

Due to RMAN's tight integration with Oracle Database, already compressed data/index blocks remain compressed during RMAN backups and do not need to be uncompressed before recovery – providing a reduction in storage costs and a potentially large reduction in backup and restore times.

Regarding compressing the backup, itself, 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 backup 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.

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/Os 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).

Advanced LOB Compression (for SecureFiles LOB segments)

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 in-line.

Advanced Row Compression depends upon deduplication to reduce the size of a block. With unstructured data stored in-line, 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, will not 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 LOB segments that cannot be compressed when stored in-line.

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. Advanced LOB Compression LOW is optimized for high performance. Advanced LOB Compression LOW maintains about 80% of the compression achieved through MEDIUM, while utilizing less CPU. Finally, Advanced LOB Compression HIGH achieves the highest storage savings but incurs the most CPU overhead.

Oracle Database detects if SecureFiles 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.

Considerations Before Testing Starts

As part of the proof-of-concept pre-test planning, make note and act (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 proof-of-concept (data, index and backup storage reduction, query/insert/update performance, bulk load operations performance, application performance etc.....).
- If the proof-of-concept is being performed using Oracle E-Business Suite, see Oracle MOS Note 2458161.1 for additional information.
- 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* and above). See Oracle MOS Note 1612095.1 for additional information.
- 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 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 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
- Advanced Row Compression works well with TDE tablespace-level encryption. With tablespace-level encryption, compression is done before encryption, so the compression ratio is not affected by the encryption. With TDE column-level encryption, the encryption is done before compression, which will negatively impact the compression ratio

About Compression Overhead

Before performing a proof-of-concept users sometime speculate that the overhead of decompression could influence 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 are not 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.

But to be truly sure of any potential overhead associated with compression, it is recommended to test using your organizations data, applications and test environment that simulates how compression will be used in production.

Improving Compression Ratios

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. If the data is unique, then it is very possible that the table/partition will not compress well or at all.

There are some 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 is 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, which could produce better compression ratios.

However, presorting will require additional preparation of the data before loading - you will need to weigh that additional time versus any compression ratio gain . Test with your own data to determine if data sorting will have an impact on your compression ratio.

Larger Block Size

It is possible a larger block size will have a better compression ratio if the larger block has more duplicate data on the block.

However, larger blocks do not always ensure higher Advanced Row Compression ratios. Test with your own data to determine if larger block sizes will have an impact on your compression ratio.

What Does a Typical Proof-of-Concept Look Like?

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 features that are tested during a compression proof-of-concept includes:

- Advanced Row Compression
- Advanced Index Compression
- RMAN Backup Compression
- Advanced LOB Compression
- Data Guard Redo Transport Compression

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

In terms of the actual proof-of-concept, customers often indicate the following:

- Before proof-of-concept testing, estimate compression ratios (storage reduction) for structured data, indexes, and unstructured data. Compression Advisor (see below) can be used to estimate Advanced Row Compression, Advanced Index Compression and Advanced LOB Compression ratios.
- Use testing to identify performance improvements and any possible performance impact from compression. Determine this by running your applications, using your data on test platforms (similar to your production hardware), and profiling the performance before and after compression. Ideally, the application testing includes application queries, bulk load operations using both conventional-path and direct-path loads, single row DML (i.e., conventional insert, update and delete operations) and RMAN backups.
- While the general suggestion is to compress all tables, some organizations instead choose only to compress the largest tables that account for approximately 80%+ of their data storage requirements
- MOS note Doc ID 729551.1 is useful for information about estimating compression savings when using Data Guard Redo Transport Compression.

- If available, Oracle's Real Application Testing (RAT) product can be a useful tool for a compression Proof-of-concept

Free Compression Advisor

An easy way to get started, with Advanced Compression, is by using compression advisor. The "DBMS_COMPRESSION" PL/SQL package (commonly called compression advisor) gathers compression-related information within a database environment.

This includes estimating the compressibility of both uncompressed partitioned, and non-partitioned tables, and gathering row-level compression information on previously compressed tables/partitions. Compression advisor provides organizations with the storage reduction information needed to make compression-related usage decisions.

The output of running compression advisor is an estimation of the compression ratio for the specific table or partition that was the target of compression advisor. The output indicates the "COMPRESSION RATIO" presented as a number such as 2.1. This number indicates that, for this specific table or partition, the estimated compression ratio is 2.1x, which represents about a 50% reduction in the footprint of the table or partition should compression be enabled.

DBMS_COMPRESSION is included with Oracle Database Enterprise Edition.

See **Appendix A** below, for a simple example of what a compression proof-of-concept may look like as illustrated in a multi-step process.

Appendix A

Example of Compression proof-of-concept as a Multi-Step Process

Apply all Relevant Patches (optionally upgrade to latest release)

- Upgrade to latest release if applicable
- Apply patches

Define Success Criteria

- Database performance
- Database size
- Backup space reduction
- Backup time/restore time
- Application performance
- Data Guard (if applicable)
- Data Pump Compression (if applicable)

Compression Advisor -- DBMS_COMPRESSION

- Obtain compression ratio estimates for Data/Indexes
- Determine the overall list of tables/indexes to be compressed

Baseline Before Compression in Test Environment: Production Workload/Data

- Gather database performance data (including bulk load operations, queries, inserts/updates etc.…)
- Gather backup/restore times

- Gather Data Guard performance data (if applicable)
- Gather database size for tables/indexes
- Gather backup size
- Gather Data Pump file size
- Gather AWR reports

Implement Compression in Test Environment

- Compress all candidate tables/indexes identified using preferred method (online/offline)
- Perform bulk load operations and compare against baseline
- Run SQL statements (query/insert/update) and compare against baseline
- Perform SQL tuning adjustments for non-performing queries (if any)
- Run production workload and verify performance
- Gather AWR Reports and compare to baseline

Prepare for Production Cutover

- Lessons Learned
- Document all the benefits and issues/resolutions encountered during Proof-of-concept
- Define cutover plan