ORACLE

Oracle Index Key Compression and Advanced Index Compression

June, 2024, Version 23ai
Copyright © 2024, Oracle and/or its affiliates
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

This document in any form, software, or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle software license and service agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This document is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described in this document remains at the sole discretion of Oracle. Due to the nature of the product architecture, it may not be possible to safely include all features described in this document without risking significant destabilization of the code.

Copyright © 2024, Oracle and/or its affiliates / Public

ORACLE

Table of contents

Introduction	4
Index Key (Prefix) Compression (Included with Enterprise Edition)	4
Enabling Index Key Compression	5
Uses of Index Key Compression	6
Determining the Optimal Prefix Column Length	6
Index Key Compression Limitations	7
Compressing Index-Organized Tables (IOTs)	7
Advanced Index Compression (Included with Advanced Compression)	9
Advanced Index Compression LOW	9
Enabling Advanced Index Compression LOW	10
Advanced Index Compression HIGH	10
Enabling Advanced Index Compression HIGH	12
Uses of Advanced Index Compression	12
Advanced Index Compression with Partitioned Indexes	12
Advanced Index Compression Limitations	13
Conclusion	13
Free Compression Advisor	13

Copyright © 2024, Oracle and/or its affiliates / Public

Introduction

The amount of data that enterprises store is growing exponentially, whereas the IT budget to manage this data is not growing nearly at the same rate. This exponential growth of data presents daunting challenges for enterprises. IT must support fast growing amounts of data, which could be due to the explosion in online content, government data retention regulations or purely due to growth in enterprises business.

However, as databases grow at accelerating rates, it can be difficult to continue to meet performance requirements while staying within budget. The key is to manage the data growth, without hurting the performance of the system, without incurring additional costs and with minimal administrative intervention.

Even though the cost of storage has been declining dramatically, the cost of enterprise class storage is not declining nearly at the same rate. The enormous growth in the data volume makes storage one of the biggest cost elements of most IT budgets.

Innovations in Oracle compression technologies help organizations reduce the resources and costs of managing large data volumes. Oracle Database has features, and technologies, to help organizations cope with these challenges, including table compression, backup compression, network compression, hybrid columnar data compression, LOB (and file) compression and index compression.

An index is a data structure that improves the performance of data retrieval operations at the cost of additional writes and storage space to maintain the structure itself. It is very common to find multiple indexes being created on a single table to support a multitude of access paths for applications.

Indexes are used extensively in OLTP and mixed workload environments, as they are capable of efficiently supporting a wide variety of access paths to the data stored in relational tables. This can cause indexes to contribute a greater share to the overall storage of a database when compared to the size of the base tables alone. Often, indexes take upward of 50% of the total database space and it is not uncommon to have over 20 indexes on a single table (many more in some cases).

Every additional index that is created on the table, even though it speeds up certain queries, introduces additional overhead for the DML or data change operations, which maintain these indexes. It is highly critical to store and manage these indexes as efficiently as possible, from both storage and efficient access perspectives.

The remainder of this document discusses the various index compression capabilities of Oracle Database.

Index Key (Prefix) Compression (Included with Enterprise Edition)

Index Key Compression, also referred to as Index Prefix Compression, has the potential to substantially reduce the overall size of indexes and helps both multi-column unique indexes and non-unique indexes alike. As a result, it is one of the most critical index optimization features available to DBAs for effectively managing the space used by the indexes.

Index Key Compression allows for compressing portions of the key values in an index segment (or Index-Organized Table (IOT)), by reducing the storage inefficiencies of storing repeating values multiple times. It compresses the data by splitting the index key into two parts:

Copyright © 2024, Oracle and/or its affiliates / Public

Prefix Entries: the leading group of columns, which are potentially shared across multiple key values.

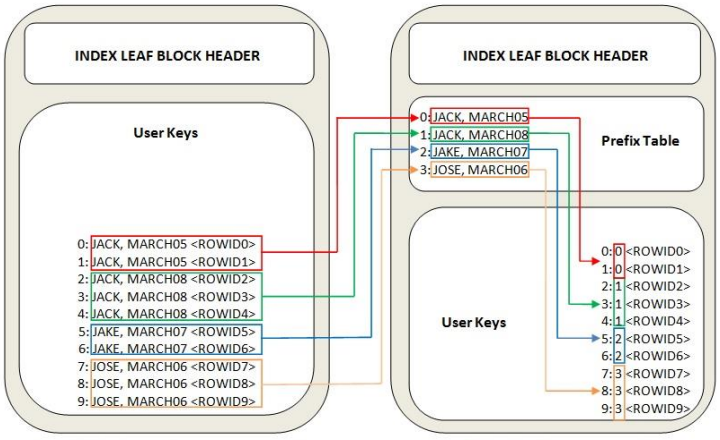
Suffix Entries: the suffix columns, which are unique to every index key.

As the prefixes are potentially shared across multiple keys in a block, these can be stored more optimally (only once per block) and 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 make up 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 it is possible to construct the complete key without incurring any additional block IO. Re-constructing the key is a very inexpensive memory only operation.

The illustration below shows the logical representation of a non-unique index leaf block with nine keys in it. The block on the left is the uncompressed representation, where every row stores all the key columns along with the ROWID of the corresponding table row. As apparent from the data, there are many repeats in the leading columns (the Prefix Columns), and these can be represented in the block more efficiently.

The block on the right is the compressed representation of the same index leaf block, where the prefix columns are stored only once, and each user row stores the reference to the corresponding prefix, which results in the index data being compressed.



Enabling Index Key Compression

For new indexes and index partitions, enabling Index Key Compression is easy - simply **CREATE** the index or index partition and specify the index compression clause. See the example below:

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

An existing index or index partition can be **REBUILT** compressed using the syntax shown below:

ALTER INDEX idxname REBUILD 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, it is possible to specify the prefix length as part of the index compression clause, which is the number of columns in the prefix entries:

CREATE INDEX idxname ON tabname(col1, col2, col3) COMPRESS 2;

Copyright © 2024, Oracle and/or its affiliates / Public

The number, prefix column length, after the **COMPRESS** keyword denotes how many columns to compress. The maximum prefix length for a non-unique index is the number of columns in the index key, and for a unique index is the number of key columns minus one.

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 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.

In many cases the overhead to construct the complete user row is offset by more efficient representation of the block, ability to fit many more user rows in a block, reduction in IO required to read the index rows and better buffer cache efficiency, such that the applications see improvement in overall performance.

Index Key Compression achieves a more optimal representation of an index and ensures that it stays permanently compressed without any subsequent overhead on the maintenance operations. As a result, it has a positive impact on the storage and space savings, but also achieves secondary benefits such as better cache efficiency, fewer leaf blocks and less deep tree resulting in potentially fewer logical IOs and cheaper execution plans.

Uses of Index Key Compression

Index Key compression can be extremely useful in many different scenarios, a few of which are listed below:

- Index Key Compression can be used with 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 only once as a prefix entry in the index block without the ROWID. The remaining rows become suffix entries consisting of only the ROWID.

- Index Key Compression can be used with a 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).

- Index Key Compression can also be used with Index Organized Tables. The same considerations as unique multicolumn indexes apply.

Determining the Optimal Prefix Column Length

The key to getting good index compression is identifying which indexes will benefit from it and correctly specifying the prefix column length for those indexes. This requires a deep understanding of the data to choose the most optimal prefix column count.

If you want to estimate the ideal compression ratio and the percentage of leaf blocks that can be saved, you need to look at **INDEX_STATS** view after **ANALYZING** the index:

```
ANALYZE INDEX index name VALIDATE structure;
SELECT name, height, blocks, opt_cmpr_count, opt_cmpr_pctsave
FROM   index_stats
WHERE  name = index name;
```

“OPT_CMPR_COUNT” indicates the number of columns to compress in the index to get maximum space savings in the leaf blocks (prefix column length).

Copyright © 2024, Oracle and/or its affiliates / Public

"OPT_CMPR_PCTSAVE" indicates the percentage reduction in leaf block space used if index is compressed using this prefix length.

Index Key Compression Limitations

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 prefix compression may not be the best solution.

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 are not shared (if at all) by other index rows. Thus, it is possible that compression in these cases is not beneficial and could end up increasing the index size due to the overhead of storing all the prefix entries.

For index compression to be beneficial, ensure that low cardinality columns are the leading columns in a concatenated index. Otherwise, there is a risk of getting negative compression such that leaf blocks can no longer store as many keys as their non-compressed counterparts.

Additionally, there is 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 the overhead associated with having prefix entries for every index row.

The key to getting good index compression is identifying which indexes will benefit from it and correctly specifying the prefix column length. The discussion above on how to figure out the optimal Prefix Column Length can help, but this approach has the following down sides:

- Requires a deep understanding of the data to choose the most optimal prefix column count.
- Specified prefix column count may not be optimal to produce the best compression ratio for every block in the index.
- Requires running **ANALYZE INDEX** to obtain an optimal prefix column count, which produces the optimal count for the index. This is not at the granularity of a block, so it may not yield the best compression ratio. Additionally, running **ANALYZE INDEX** takes an exclusive lock on the table, effectively making the table "offline" for this period.
- Possible to get negative compression, as pointed out earlier, such as in the case where the specified prefix columns are unique in a block.

Application developers and DBAs need to be very selective on which indexes to compress and correctly set the prefix column count for these indexes. Oracle protects you under certain obvious conditions, but it is your responsibility to compress the indexes in the right manner.

Compressing Index-Organized Tables (IOTs)

Before any reorganization, it is recommended that you determine if prefix compression would be useful, and if so, how may prefix columns should be specified.

This step is required for Index Key Compression and requires some knowledge of the data. If there are repeated leading columns, then typically prefix compression is beneficial.

Copyright © 2024, Oracle and/or its affiliates / Public

For example, if the keys look like:

A B C D
A C D B
A D B C

Since there is a repeated first column, and that can be stored once as a prefix, compression would reduce the suffix rows to the following:

B C D
C D B
D B C

It is a more difficult choice if there are rows like:

A B C D
A B D C
A C E F
A G H I

Here, it is difficult to know whether to choose 1 or 2 prefix columns to compress – choosing 2 would mean we store the following prefix rows:

A B
A C
A G

Choosing one would mean only storing "A". The "optimal" choice is the one that saves most space in prefix+suffix rows together.

As indicated earlier, **ANALYZE INDEX** determines the optimal count for how many columns to compress. For example, with DDL like this:

```
CREATE TABLE tiot (c1 number, c2 number, c3 number, constraint tiot_pk primary key (c1, c2, c3))
ORGANIZATION INDEX;
```

ANALYZE INDEX can be used as follows:

```
ANALYZE INDEX TIOT_PK VALIDATE STRUCTURE;
SELECT OPT_CMPR_COUNT, OPT_CMPR_PCTSAVE FROM index_stats;


OPT_CMPR_COUNT OPT_CMPR_PCTSAVE
---------------------  ------------------------
1                      20
```

ANALYZE INDEX indicates that a prefix count of 1 would give an estimated 20% saving in space.

The IOT segment can be rebuilt, enabling compression, using the following **ALTER TABLE MOVE** command online and indicates 1 column is to be used for compression.

```
ALTER TABLE tiot MOVE COMPRESS 1 ONLINE;
```

Below are some SQL commands/examples on how to achieve index key compression on an existing non-compressed IOT (as well as examples of SQL commands that are invalid with IOTs)

Copyright © 2024, Oracle and/or its affiliates / Public

ALTER TABLE <table-name> MOVE COMPRESS [number of columns] [ONLINE]

Examples:

- alter table tiot move compress online;
- alter table tiot move compress 1 online;
- alter table tiot move compress 2 online;
- alter table tiot move compress 1

ALTER TABLE <table-name> COMPRESS

This is an invalid command for an IOT.

ALTER INDEX REBUILD COMPRESS

This is an invalid command for an IOT. Attempts to rebuild the Primary Key of the IOT will result in:

ORA-28650: Primary index on an IOT cannot be rebuilt

DBMS_REDEFINITION can be used to compress an existing non-compressed IOT. Please see the dbms_redefinition documentation for more details.

Using partition exchange is not possible as a means of enabling compression when using prefix compression of indexes. The reason is that a partitioned IOT cannot be moved as a whole, nor can a partition be moved to be compressed if the table (IOT) is not compressed.

Further, can a partition be exchanged unless both the partition and the table have the same compression attribute (i.e., they are both (non) compressed already).

Advanced Index Compression (Included with Advanced Compression)

Index entries, with many duplicate keys, can be compressed making it possible to reduce both the storage overhead and the access overhead for large index range scans or fast full scans. Prefix 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 prefix compression may not be the best solutions.

Advanced Index Compression, a feature of Advanced Compression, simplifies index compression and at the same time achieves much higher compression ratios for indexes. Advanced Index Compression enables the highest levels of data compression and provides enterprises with tremendous cost-savings and performance improvements due to reduced I/O.

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

This document will discuss each of the compression levels, in detail, in the next sections.

Advanced Index Compression LOW

Unlike the previously discussed Index Key Compression, Advanced Index Compression LOW computes the prefix column count for compressed indexes. Rather than using a static prefix count for all index leaf block, it aims towards computing an optimal prefix count for every index leaf block in the index.

The correct and most optimal numbers of prefix columns are computed automatically on a block-by-block basis, and thus produce the best compression ratio possible. It is now possible to have different index leaf

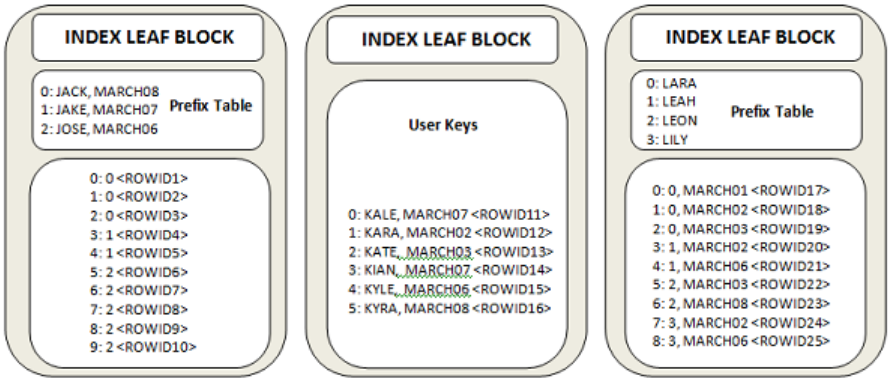
Copyright © 2024, Oracle and/or its affiliates / Public

blocks compressed with different prefix column count or not be compressed at all if there are no repeating prefixes.

The illustration below shows logical structure of three consecutive index leaf blocks, each compressed differently.

- For the block to the left, the optimal prefix column count is 2, and the block is compressed with the first 2 columns from the index key in the prefix.
- For the block in the center, since there are no repeats in the leading columns, the block is left uncompressed.
- For the block on the right, the optimal prefix column count is 1 column, and the block is compressed with only 1 leading prefix column.

The dynamic algorithm to compute prefix column count automatically on block-by-block basis guarantees maximizing the compression benefits for the index and makes sure that the compressed index segment is never bigger in size than its non-compressed counterpart.



Logical Structure of a compressed index leaf block with 2 columns in prefix entry

Logical Structure of a non-compressed index leaf block

Logical Structure of a compressed index leaf block with 1 column in prefix entry

Enabling Advanced Index Compression LOW

Advanced Index Compression LOW can be enabled easily by specifying the **COMPRESS** option for indexes. 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 ADVANCED LOW;
```

Index-Organized Tables (IOT's) are essentially indexes and can be compressed with Advanced Index Compression (LOW).

Advanced Index Compression HIGH

Advanced Index Compression HIGH is geared towards dramatically improving index compression ratios. It introduces many additional compression techniques, which improves the compression ratios significantly while still providing efficient OLTP access.

With Advanced Index Compression HIGH, every index leaf block can contain compressed and uncompressed rows. The compressed index key entries are stored physically as Compression Units (a concept like Oracle Hybrid Columnar Compression), utilizing more complex compression algorithms on a potentially larger number of index keys to achieve higher levels of compression. While the recently inserted keys and modified keys are stored in the non-compressed region of the leaf block.

Copyright © 2024, Oracle and/or its affiliates / Public

Advanced Index Compression uses an internal threshold to trigger (re) compression of the leaf block. Recently inserted rows are buffered uncompressed in the block, which is then compressed as the block fullness approaches this threshold. This ensures that the cost of compression is amortized over multiple DML operations and that not every operation incurs compression overhead.

With indexes, this internal threshold is geared towards avoiding index block splits and alleviating the need to allocate additional leaf blocks to the index structure.

As stated earlier, Advanced Index Compression utilizes complex sets of compression algorithms to achieve higher compression ratios. Some of the compression techniques used with Advanced Index Compression HIGH include (but are not limited to):

Intra-column Prefix Replacement

Intra-column prefix replacement algorithm exploits the fact that, because of index rows being sorted in key order, there is a high likelihood that a prefix of each key matches the corresponding prefix of the preceding key even at sub key column level.

Replacing the matching prefixes from each row with a reference to the corresponding symbol gives good compression benefits. Additionally, if the cardinality of the symbol table indexes is low, and many index keys have a matching prefix, bit encoding the symbol table references can further improve compression benefits.

Length Byte Compression

It is very common to find many rows in an index with short column lengths. Thus, it is possible to encode these lengths in less than a byte (as with the uncompressed and prefix compressed index) and hence save space.

Additionally, if all key columns in the block have the same length, the block level fixed length can be stored.

Duplicate Key Removal

If the index block has many duplicates, it is possible to realize significant space savings by storing the key exactly once followed by a list of ROWIDs associated with the key in sorted order.

Intra-column prefix compression can then be applied on top of this transformed representation to further compress the now unique set of keys.

ROWID List Compression

ROWID List Compression is an independent transformation that takes the set of ROWIDs for each unique index key and represents them in a compressed form, ensuring that the compressed ROWID representation is logically maintained in the ROWID order to allow for efficient ROWID based lookup.

Row Directory Compression

The general idea behind Row Directory Compression is to layout the compressed rows contiguously in the increasing offset order within each 256-byte region of the index block, which enables maintaining a base offset (once per 256 bytes) and a relative 1 byte offset per compressed row.

Copyright © 2024, Oracle and/or its affiliates / Public

Flag and Lock Byte Compression

Generally, index rows are not locked, and the flags are similar for all the rows in the index block. These lock and flag bytes on disk can be represented more efficiently provided it is possible to access and modify them. Any modification to the flag or lock bytes requires these to be uncompressed.

Note that there is no need to provide the compression technique to use with Advanced Index Compression. Not every compression technique is applicable to every index. The decision on which compression algorithms are applicable to an index is made real-time and can differ from index-to-index and block-to-block.

Enabling Advanced Index Compression HIGH

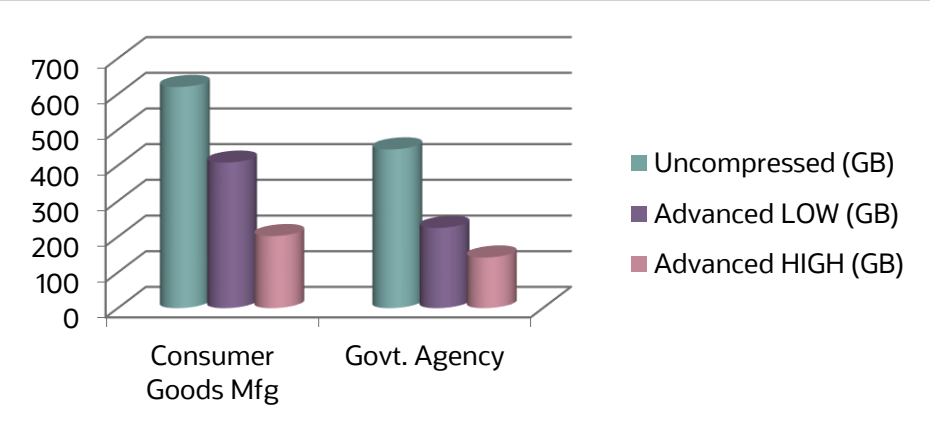
Advanced Index Compression HIGH can be enabled easily by specifying the **COMPRESS** option for indexes. New indexes can be automatically created as compressed, or the existing indexes can be rebuilt compressed.

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

Uses of Advanced Index Compression

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 guarantees that the size does not increase due to negative compression) and at the same time improves the compression ratio significantly while still providing efficient access to the indexes.

The following graph shows sample compression ratios for two customers using Advanced Index Compression in an SAP environment. Along with substantially reducing the storage footprint for the indexes, these workloads also observed significant improvement in the overall system performance.



Advanced Index Compression with Partitioned Indexes

For partitioned indexes, organizations can specify the compression clause for the entire index or on a partition-by-partition basis. Therefore, organizations can choose to have some index partitions compressed, while others are not.

The following example shows a mixture of compression attributes on the partitioned indexes:

```
CREATE INDEX my_test_idx ON test(a, b) COMPRESS ADVANCED HIGH local (PARTITION p1 COMPRESS
ADVANCED LOW, PARTITION p2 COMPRESS, PARTITION p3, PARTITION p4 NOCOMPRESS);
```

The next example below shows Advanced Index Compression support on partitions where the parent index is not compressed:

Copyright © 2024, Oracle and/or its affiliates / Public

```
CREATE INDEX my_test_idx ON test(a, b) NOCOMPRESS local (PARTITION p1 COMPRESS ADVANCED
LOW, PARTITION p2 COMPRESS ADVANCED HIGH, PARTITION p3);
```

Advanced Index Compression Limitations

- Advanced Index Compression is not supported for Bitmap Indexes
- Advanced Index Compression is not supported for compress Functional Indexes

Conclusion

The massive growth in data volume, being experienced by organizations, 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 extraordinary application performance.

With Advanced Index Compression, it is now possible to simply enable compression for all your B-Tree indexes, and Oracle will only compress every index leaf block when beneficial, while 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 and aims towards achieving the most optimal compression ratio for the entire index segment, while still providing efficient access to the indexes.

Using Advanced Index Compression, along with other Oracle Advanced Compression features, organizations can efficiently manage their increasing data requirements with minimal administrative intervention – minimizing database storage costs while continuing to achieve the highest levels of application performance.

Free Compression Advisor

An easy way to get started, with Advanced Index 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.

Connect with us

Call +**1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

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

Copyright © 2024, Oracle and/or its affiliates. This document is provided for information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.