ORACLE

Oracle Database 23ai New Compression and SecureFiles Storage Features Overview

July 2025, 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

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.

ORACLE

Table of Contents

About this Document

This document discusses new Oracle Database 23ai compression related features focused on reducing both data and index storage requirements, as well as a new feature focused on helping administrators better manage SecureFiles LOB segment space.

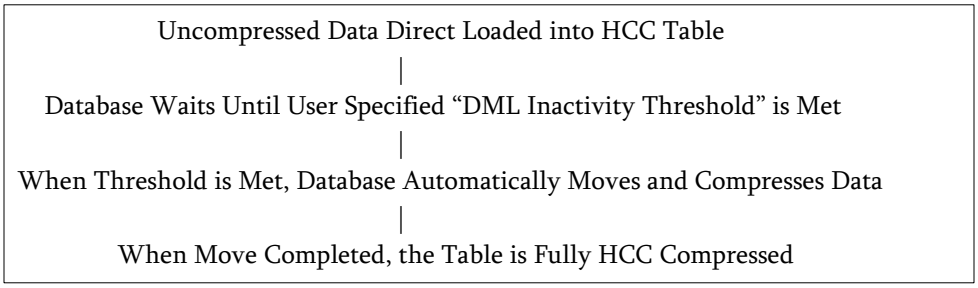
The new features include Automatic Storage Compression, Automatic SecureFiles Shrink and Advanced Index Compression LOW for IOTs.

Overview of Automatic Storage Optimization

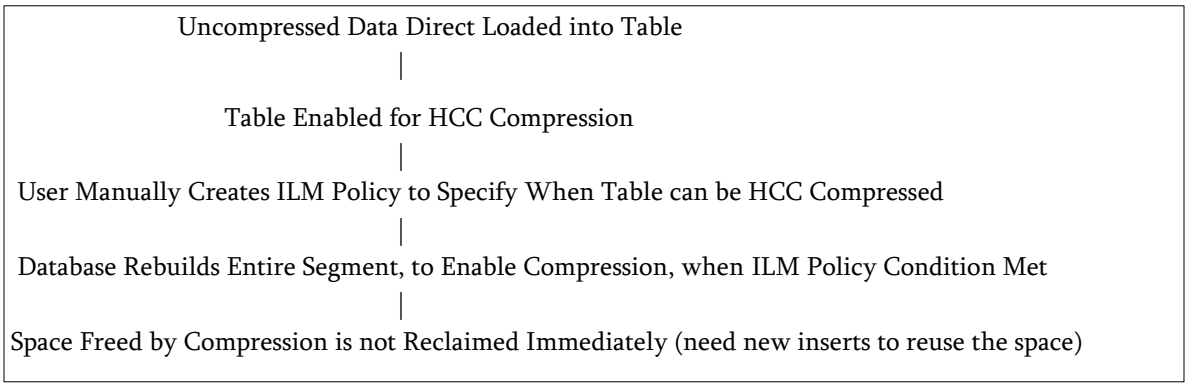
Organizations use Oracle's Hybrid Columnar Compression (HCC) for space savings and fast analytics performance. However, the compression and decompression overhead of Hybrid Columnar Compression can affect direct load performance. To improve direct load performance, Automatic Storage Compression enables Oracle Database to direct load data into an uncompressed format initially, and then gradually move rows into Hybrid Columnar Compression format in the background.

When Automatic Storage Compression is enabled, direct loads into a Hybrid Columnar Compression object would use the uncompressed format to achieve faster loads. The database will then wait until there are no modifications, to the newly loaded data for the duration of the user specified DML inactivity threshold. At that point, the data from the uncompressed direct load will be gradually HCC compressed in the background using AutoTask.

At a high level, the process would appear as follows:



Compare this to the existing manual ILM process:



Usage Prerequisites

The table must be in a tablespace with these properties:

- In the PDB, set `HEAT_MAP=ON`
- Table(s) need to be specified HCC and reside on a tablespace that uses SEGMENT SPACE MANAGEMENT AUTO and AUTOALLOCATE

Usage Example

- Enable Automatic Storage Compression at the PDB level

```
exec dbms_ilm_admin.enable_auto_optimize;
```

- Create a table without HCC compression

  For this example, a table named “MYTAB” will be used as an example. The table was created without any compression.

- Check that the table is not compressed

  Confirm that the table is not yet compressed.

```
select unique dbms_compression.get_compression_type('SCOTT', 'MYTAB',
rowid) from scott.mytab;
```

DBMS_COMPRESSION.GET_COMPRESSION_TYPE('SCOTT','MYTAB',ROWID)
1

Note: dbms_compression.get_compression_type uses constants that can be used to determine compression type. “1” indicates that the table is currently not compressed.

See the Oracle *PL/SQL Packages and Types Reference* documentation for more information.

- ALTER table to add HCC compression and load data using direct path

  For this example, HCC Query LOW compression as added to the table and insert /*+ append */ used to perform the direct-path load.

  Confirm the uncompressed size of the table before automatic compression begins.

```
select bytes/1024/1024 MB from dba_segments where owner = 'SCOTT' and
segment_name = 'MYTAB';
```

MB
5.625

  5.625 indicates the uncompressed table size (MB).

- Monitor the incremental progress of the automatic compression by checking the “Auto compression data moved” system statistic, which increases over time as data is moved and compressed

  The one-hour default inactivity interval will allow the segment size to be observed before compression starts.

```
select name, value from v$sysstat where name like 'Auto compression data%';
```

NAME	VALUE
Auto compression data movement success	0
Auto compression data movement failure	0
Auto compression data moved	0

Note: v$sysstat will show the sum of values, across all tables that are using automatic compression. If you are

compressing more than one table, then the value of "Auto compression data moved" would include the data moved for those tables as well. Also, due to rounding up, the value may not exactly match the actual size of uncompressed data over time

- As the data movement and compression begins, the value of "Auto compression data moved" increases

```
select name, value from v$sysstat where name like 'Auto compression data%';
```

NAME	VALUE
Auto compression data movement success	1
Auto compression data movement failure	0
Auto compression data moved	6

For this example, the value (MB) of "Auto compression data moved" indicates "6", meaning that approximately 6MB of uncompressed data was moved to compression. Note that when automatic compression started, the uncompressed size of the data was 5.625MB.

As this example demonstrates, the segment size before compression, and the amount of data moved during compression, may not be an exact match. As mentioned earlier, v$sysstat shows the sum of values across all tables using automatic compression, if you are loading more than one table, then the value of "Auto compression data moved" would include the data for those tables as well.

Also, due to rounding up, the value of "Auto compression data moved" may not exactly match the actual size of uncompressed data over time. Although the value is expected to be similar to the uncompressed size of the data.

- Check the compression level of the table and the compressed size

```
select unique dbms_compression.get_compression_type('SCOTT', 'MYTAB',
rowid) from scott.mytab;
```

DBMS_COMPRESSION.GET_COMPRESSION_TYPE('SCOTT','MYTAB',ROWID)
8

Note: dbms_compression.get_compression_type uses constants that can be used to determine compression type. "8" indicates that the table is currently compressed using HCC Query Low compression. See the *Oracle PL/SQL Packages and Types Reference* for more information.

```
select bytes/1024/1024 MB from dba_segments where owner = 'SCOTT' and
segment_name = 'MYTAB';
```

MB
.3125

.3125MB indicates the size of that table after automatic compression to HCC Query Low.

Notes about the output of v$sysstat.

- "Auto compression data movement failure" is the number of unsuccessful data movement (compression) attempts. This value is incremented if 1) the database ran into any errors during compression, or 2) the database ran out of time in the current background task to process further – in which case compression will resume in the next background task.

- "Auto compression data movement success" is the number of successful data movement attempts. Auto Compression may break up the data movement work for a single direct load into multiple batches, so this may not exactly match the number of segments or direct loads.

For more information about the usage of Automatic Storage Compression, please see the "More Information" section at the end of this document.

Overview of Manual and Automatic SecureFiles Shrink

SecureFiles is the default storage mechanism for LOBs with Oracle Database. Oracle strongly recommends SecureFiles for storing and managing LOBs.

The Oracle Database SecureFiles Shrink feature provides manual, and automatic methods to free the unused space in SecureFiles LOB segments and release the space back to the containing tablespace.

This document provides an overview of both the Manual, and Automatic SecureFiles Shrink features.

Manual SecureFiles Shrink

Use the ALTER TABLE… SHRINK SPACE statement to manually shrink a SecureFiles LOB segment. You can use the Segment Advisor, or a PL/SQL procedure such as DBMS_SPACE.SPACE_USAGE to return information about SecureFiles space usage before deciding on the SecureFiles LOB segments to shrink.

The following points are important regarding the manual shrink method:

- The manual SecureFiles shrink operation is an online DDL with part of the operations being offline, where offline means concurrent DML are blocked until the shrink activity on the critical section ends. The concurrent DML statements do not fail with ORA-00054, but are blocked
- The manual SecureFiles shrink operation disregards any flavor of undo retention and treats it as if the retention is equal to none. Users cannot expect the LOB retention feature to provide the usual guarantees after invoking the shrink operation. A user may see the ORA-1555 snapshot too old message in queries. Run the shrink operation with caution if this is a concern

Manual Shrink can be Invoked Using these Methods:

- This command targets the specified LOB column and all its partitions:

```
ALTER TABLE <table_name> MODIFY LOB <lob_column> SHRINK SPACE
```

- The following command cascades the shrink operation for all LOB columns and its partitions in the specified table:

```
ALTER TABLE <table_name> SHRINK SPACE CASCADE
```

Use manual shrink on SecureFiles LOB segments from Oracle Database 21c and onward.

Automatic SecureFiles Shrink

It may not be feasible for administrators to spend their time checking manually each SecureFiles LOB segment to shrink. Automatic SecureFiles Shrink uses a framework that enables automatic selection of SecureFiles LOB segments to shrink based on a set of criteria (see selection criteria below) and it runs Automatic SecureFiles Shrink in the background.

Automatic SecureFiles Shrink is designed to minimize the functional and performance impact on concurrent workloads. While shrink runs automatically on a SecureFiles LOB segment, all Data Manipulation Language (DML) statements and Data Definition Language (DDL) statements that involve the segment will succeed. Space is gradually freed in the SecureFiles LOB Segment, and the performance impact is minimal.

Automatic SecureFiles Shrink does not have any effect on the BasicFiles LOBs and in-lined LOBs. Automatic SecureFiles Shrink ensures that SecureFiles LOB segments do not consume excessive free space and alleviates administrators from the burden of manually running SecureFiles Shrink.

Automatic SecureFiles Shrink is not enabled by default.

- In on-premises environments, run the following command to enable the Automatic SecureFiles Shrink feature:

  ```
  exec DBMS_SPACE.SECUREFILE_SHRINK_ENABLED():
  ```

- In Autonomous Cloud environments, contact your system administrator to enable Automatic SecureFiles Shrink

SecureFiles LOB Segments Selection Criteria for Automatic Shrink

The Automatic SecureFiles Shrink task excludes the following SecureFiles LOB segments when choosing the SecureFiles LOB segments to shrink:

- The SecureFiles LOB segment is not an idle segment as per LOB Segment Idle Time Limit
- The SecureFiles LOB segment does not contain extra free space greater than the pre-allocation threshold
- The SecureFiles LOB segment has RETENTION MAX, which means the segment keeps as many unexpired blocks as possible
- The SecureFiles LOB segment is currently being shrunk
- The SecureFiles LOB segment does not have enough expired free space that is no longer needed for lob retention requirement. Space that is still needed for lob retention is treated as used space
- The SecureFiles LOB segment has failed a previous shrink task. Previous shrink calls have failed to free space from the SecureFiles LOB segment. Automatic SecureFiles Shrink identifies the LOB segments that it failed to shrink previously and avoids such segments

Automatic SecureFiles Shrink Task

Automatic SecureFiles Shrink performs a series of steps to complete the shrink of SecureFiles LOB segments.

When enabled, a shrink task is performed as one instance of the background action performed on AutoTask. The task runs every 30 minutes and performs the following steps:

1. A shrink task has 60 minutes at the start of the task. As the task progresses, it tracks both the time spent so far and the average duration of a shrink call. The latter is used to predict how long the next shrink call would take. If the time left is not enough for another call, the shrink task exits. If a shrink call goes over the 60-minute mark, it is terminated

2. Automatic SecureFiles Shrink fetches the next batch of SecureFiles LOB segments from internal catalog tables (which is ordered by objd). The last objd in the previous shrink task is used as the starting point for the next shrink task
3. Automatic SecureFiles Shrink applies the criteria filters from the Selection Criteria for SecureFiles LOB segment to remove the segments that do not qualify for the shrink task
4. Once the qualified segment is found, the shrink task can start work on the segment
5. Before starting the shrink, the shrink target is computed. The shrink target is based on the Pre-Allocation Threshold and the Automatic SecureFiles Shrink Trickle Threshold
6. Automatic SecureFiles Shrink runs the shrink command. The ALTER TABLE ... SHRINK SPACE command is executed using the OCI interface
7. Automatic SecureFiles Shrink updates the timestamp for the next shrink. This timestamp indicates the earliest time when Automatic SecureFiles Shrink can select this SecureFiles LOB segment again. If space was freed successfully, the timestamp uses the current time. Otherwise, the shrink is assigned a time in the future. If shrink is not successful, a penalty time is assessed to avoid Automatic SecureFiles Shrink from selecting the same LOB segment in future shrink tasks

Key Automatic SecureFiles Shrink Capabilities

- Integrates with Pre-Allocation:

  Automatic SecureFiles Shrink integrates with pre-allocation seamlessly without affecting performance. Automatic SecureFiles Shrink avoids the SecureFiles LOB segments that are recently pre-allocated. Segment pre-allocation is performed in the background for segments that have high demand for free space

- Works with DDL and DML:

  Automatic SecureFiles Shrink targets only idle segments and skips active SecureFiles LOB segments. User driven DDL and DML statements do not fail and face minimal performance impact when Automatic SecureFiles Shrink works in the background. If Automatic SecureFiles Shrink for a SecureFiles LOB segment comes across locked rows, it skips the locked rows because locked rows are indicative of DML activity or waiting on locked rows may cause deadlocks with user transactions.

- Targets Idle SecureFiles LOB Segments:

  To avoid unnecessary block accesses, Automatic SecureFiles Shrink filters SecureFiles LOB segments based on information available in System Global Area (SGA). Automatic SecureFiles Shrink selects only idle SecureFiles LOB segments and skips active SecureFiles LOB segments to minimize performance impact on active SecureFiles LOB segments

- Covers All SecureFiles LOB Segments:

  Automatic SecureFiles Shrink task covers all SecureFiles LOB segments in a PDB over several intervals and this includes user created SecureFiles LOB segments and the SecureFiles LOB segments that are created using features, such as JSON and DBFS

- Performs Shrinks in Iterations:

  Automatic SecureFiles Shrink does not free all the free space in the selected SecureFiles LOB segments at once. Instead, the Automatic SecureFiles shrink task frees a modest amount of space at every shrink call (iteration). The trickle threshold limit defines the amount of space to shrink in every iteration. Over time, the amount of free space in idle SecureFiles LOB segments approaches the minimum that is specified for pre-allocation

- Executes in the Background:

  All steps involved in Automatic SecureFiles Shrink, including the selection of SecureFiles LOB segments to shrink, run in the background. After Automatic SecureFiles Shrink is enabled, it comes into effect with the start of a database instance. No directive regarding how Automatic SecureFiles Shrink should operate is required

- Honors Undo Retention:

  Automatic SecureFiles Shrink respects the undo retention period. It does not allow a query to fail within the undo retention period because an affected SecureFiles LOB segment has been freed, relocated, or reused as a part of an Automatic SecureFiles Shrink task. Unexpired blocks are freed only after the undo retention time

Automatic SecureFiles Shrink simplifies administrator duties. Automatic SecureFiles Shrink automatically selects SecureFiles LOB segments, based on a set of criteria, and executes the shrink operation in the background for the selected SecureFiles LOB segments. With Automatic SecureFiles Shrink, the shrink operation happens transparently in small and gradual steps over time while allowing DDL and DML statements to execute concurrently.

Overview of Advanced Index Compression LOW for IOTs

Advanced Index Compression, a feature of Advanced Compression, simplifies index compression. Advanced Index Compression enables the highest levels of data compression and provides enterprises with storage savings and query performance improvements due to reduced I/O. Advanced Index Compression is an enabling technology for multiple compression levels, LOW and HIGH.

This discussion will focus on the LOW level of index compression.

Advanced Index Compression LOW

Advanced Index Compression LOW computes the prefix column count for compressed indexes. Rather than using a static prefix count for all index leaf blocks, it aims to compute an optimal prefix count for every index leaf block.

The correct and most optimal numbers of prefix columns are calculated automatically on a block-by-block basis and thus produce the best compression ratio possible. It is possible to have different index leaf blocks compressed with different prefix column counts or not be compressed at all if there are no repeating prefixes.

Advanced Index Compression LOW for Index-Organized Tables (IOTs).

An index-organized table is a table stored in a variation of a B-tree index structure. In contrast, a heap-organized table inserts rows where they fit.

In an index-organized table, rows are stored in an index defined on the primary key for the table. Each index entry in the B-tree also stores the non-key column values. Thus, the index is the data, and the data is the index. Applications manipulate index-organized tables just like heap-organized tables, using SQL statements.

IOTs are popular because they provide fast random access by primary key without duplicating primary key columns in two structures – a heap table and an index. Index-Organized Tables can now be compressed with Advanced Index Compression LOW.

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

Usage example:

```
create table tiot (c1 number, c2 number, c3 number, c4 number, primary key (c1,
c2)) organization index compress advanced low;
```

In earlier releases, IOTs only supported Oracle's Prefix Key Compression for index compression. Usage of Prefix Compression required user analysis and had the possibility of negative compression (where the overhead of compression outweighed the compression benefits).

Prefix Index Compression is included with Oracle Database Enterprise Edition.

This new feature extends Advanced Index Compression LOW to IOTs, allowing users to enable compression for all IOTs without the possibility of negative compression and without any user analysis required.

So that does this mean for your organization?

Average IOT storage reduction can range from up to 2x to 5x. Using 2x as an example, this means that the amount of space consumed by uncompressed data will be two times larger than that of the compressed data. By reducing their IOT storage requirements, IT managers can reduce, and sometimes eliminate their need to purchase new storage.

The cost of decompressing a block compressed with Advanced Index Compression LOW is compensated by the fact that in most scenarios, the database would be scanning a smaller number of blocks. So, in general, IOT compression typically won't compromise query performance (no significant degradation is typical).

See how well your IOTs will compress with the free Compression Advisor

The "DBMS_COMPRESSION" PL/SQL package (commonly called compression advisor) is included with Oracle Database Enterprise Edition and gathers compression-related information within a database environment.

The output of running compression advisor is an estimation of the compression ratio for the specific table or index that was the target of compression advisor. Compression advisor provides organizations with the storage reduction information needed to make compression-related usage decisions.

For more information about the usage of Advanced Index Compression LOW for IOTs, and Oracle Compression Advisor, please see the "More Information" section below.

More Information

- See the Oracle *VLDB and Partitioning Guide* documentation for more information, and usage examples, for Automatic Storage Compression
- See the Oracle *SecureFiles and Large Objects Developer's Guide* for more information, and usage examples, for SecureFiles Shrink
- See the Oracle *Database Concepts* documentation for more information, and usage examples, for Advanced Index Compression LOW for IOTs
- See the Oracle *PL/SQL Packages and Types Reference* for more information about Compression Advisor (DBMS_COMPRESSION)

ORACLE

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