<https://docs.microsoft.com/en-us/sql/relational-databases/data-compression/data-compression>

# Data Compression

2016-7-1 12 min to read Contributors

* [BYHAM](https://github.com/BYHAM)

* [Craig Guyer](https://github.com/craigg-msft)

**THIS TOPIC APPLIES TO:** yesSQL Server (starting with 2016)yesAzure SQL DatabasenoAzure SQL Data Warehouse noParallel Data Warehouse

SQL Server 2017 and Azure SQL Database support row and page compression for rowstore tables and indexes, and supports columnstore and columnstore archival compression for columnstore tables and indexes.

For rowstore tables and indexes, use the data compression feature to help reduce the size of the database. In addition to saving space, data compression can help improve performance of I/O intensive workloads because the data is stored in fewer pages and queries need to read fewer pages from disk. However, extra CPU resources are required on the database server to compress and decompress the data, while data is exchanged with the application. You can configure row and page compression on the following database objects:

* A whole table that is stored as a heap.
* A whole table that is stored as a clustered index.
* A whole nonclustered index.
* A whole indexed view.
* For partitioned tables and indexes, you can configure the compression option for each partition, and the various partitions of an object do not have to have the same compression setting.

For columnstore tables and indexes, all columnstore tables and indexes always use columnstore compression and this is not user configurable. Use columnstore archival compression to further reduce the data size for situations when you can afford extra time and CPU resources to store and retrieve the data. You can configure columnstore archival compression on the following database objects:

* A whole columnstore table or a whole clustered columnstore index. Since a columnstore table is stored as a clustered columnstore index, both approaches have the same results.
* A whole nonclustered columnstore index.
* For partitioned columnstore tables and columnstore indexes, you can configure the archival compression option for each partition, and the various partitions do not have to have the same archival compression setting.

Note

Data can also be compressed using the GZIP algorithm format. This is an additional step and is most suitable for compressing portions of the data when archiving old data for long term storage. Data compressed using the COMPRESS function cannot be indexed. For more information, see [COMPRESS (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/functions/compress-transact-sql).

## Considerations for When You Use Row and Page Compression

When you use row and page compression, be aware the following considerations:

* The details of data compression are subject to change without notice in service packs or subsequent releases.
* Compression is available in Azure SQL Database
* Compression is not available in every edition of SQL Server. For more information, see [Features Supported by the Editions of SQL Server 2016](https://docs.microsoft.com/en-us/sql/sql-server/editions-and-supported-features-for-sql-server-2016).
* Compression is not available for system tables.
* Compression can allow more rows to be stored on a page, but does not change the maximum row size of a table or index.
* A table cannot be enabled for compression when the maximum row size plus the compression overhead exceeds the maximum row size of 8060 bytes. For example, a table that has the columns c1**char(8000)** and c2**char(53)** cannot be compressed because of the additional compression overhead. When the vardecimal storage format is used, the row-size check is performed when the format is enabled. For row and page compression, the row-size check is performed when the object is initially compressed, and then checked as each row is inserted or modified. Compression enforces the following two rules:
  + An update to a fixed-length type must always succeed.
  + Disabling data compression must always succeed. Even if the compressed row fits on the page, which means that it is less than 8060 bytes; SQL Server prevents updates that would not fit on the row when it is uncompressed.
* When a list of partitions is specified, the compression type can be set to ROW, PAGE, or NONE on individual partitions. If the list of partitions is not specified, all partitions are set with the data compression property that is specified in the statement. When a table or index is created, data compression is set to NONE unless otherwise specified. When a table is modified, the existing compression is preserved unless otherwise specified.
* If you specify a list of partitions or a partition that is out of range, an error will be generated.
* Nonclustered indexes do not inherit the compression property of the table. To compress indexes, you must explicitly set the compression property of the indexes. By default, the compression setting for indexes will set to NONE when the index is created.
* When a clustered index is created on a heap, the clustered index inherits the compression state of the heap unless an alternative compression state is specified.
* When a heap is configured for page-level compression, pages receive page-level compression only in the following ways:
  + Data is bulk imported with bulk optimizations enabled.
  + Data is inserted using INSERT INTO ... WITH (TABLOCK) syntax and the table does not have a nonclustered index.
  + A table is rebuilt by executing the ALTER TABLE ... REBUILD statement with the PAGE compression option.
* New pages allocated in a heap as part of DML operations will not use PAGE compression until the heap is rebuilt. Rebuild the heap by removing and reapplying compression, or by creating and removing a clustered index.
* Changing the compression setting of a heap requires all nonclustered indexes on the table to be rebuilt so that they have pointers to the new row locations in the heap.
* You can enable or disable ROW or PAGE compression online or offline. Enabling compression on a heap is single threaded for an online operation.
* The disk space requirements for enabling or disabling row or page compression are the same as for creating or rebuilding an index. For partitioned data, you can reduce the space that is required by enabling or disabling compression for one partition at a time.
* To determine the compression state of partitions in a partitioned table, query the data\_compression column of the sys.partitions catalog view.
* When you are compressing indexes, leaf-level pages can be compressed with both row and page compression. Non–leaf-level pages do not receive page compression.
* Because of their size, large-value data types are sometimes stored separately from the normal row data on special purpose pages. Data compression is not available for the data that is stored separately.
* Tables which implemented the vardecimal storage format in SQL Server 2005 will retain that setting when upgraded. You can apply row compression to a table that has the vardecimal storage format. However, because row compression is a superset of the vardecimal storage format, there is no reason to retain the vardecimal storage format. Decimal values gain no additional compression when you combine the vardecimal storage format with row compression. You can apply page compression to a table that has the vardecimal storage format; however, the vardecimal storage format columns probably will not achieve additional compression.

Note

SQL Server 2017 supports the vardecimal storage format; however, because row-level compression achieves the same goals, the vardecimal storage format is deprecated. This feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature.

## Using Columnstore and Columnstore Archive Compression

||  
|-|  
|**Applies to**: SQL Server ( SQL Server 2014 through [current version](http://go.microsoft.com/fwlink/p/?LinkId=299658)), Azure SQL Database.|

### Basics

Columnstore tables and indexes are always stored with columnstore compression. You can further reduce the size of columnstore data by configuring an additional compression called archival compression. To perform archival compression, SQL Server runs the Microsoft XPRESS compression algorithm on the data. Add or remove archival compression by using the following data compression types:

* Use **COLUMNSTORE\_ARCHIVE** data compression to compress columnstore data with archival compression.
* Use **COLUMNSTORE** data compression to decompress archival compression. This resulting data will continue to be compressed with columnstore compression.

To add archival compression, use [ALTER TABLE (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-table-transact-sql) or [ALTER INDEX (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-index-transact-sql) with the REBUILD option and DATA COMPRESSION = COLUMNSTORE.

Examples:

Copy

ALTER TABLE ColumnstoreTable1

REBUILD PARTITION = 1 WITH (DATA\_COMPRESSION = COLUMNSTORE\_ARCHIVE) ;

ALTER TABLE ColumnstoreTable1

REBUILD PARTITION = ALL WITH (DATA\_COMPRESSION = COLUMNSTORE\_ARCHIVE) ;

ALTER TABLE ColumnstoreTable1

REBUILD PARTITION = ALL WITH (DATA\_COMPRESSION = COLUMNSTORE\_ARCHIVE ON PARTITIONS (2,4)) ;

To remove archival compression and restore the data to columnstore compression, use [ALTER TABLE (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-table-transact-sql) or [ALTER INDEX (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-index-transact-sql) with the REBUILD option and DATA COMPRESSION = COLUMNSTORE.

Examples:

Copy

ALTER TABLE ColumnstoreTable1

REBUILD PARTITION = 1 WITH (DATA\_COMPRESSION = COLUMNSTORE) ;

ALTER TABLE ColumnstoreTable1

REBUILD PARTITION = ALL WITH (DATA\_COMPRESSION = COLUMNSTORE) ;

ALTER TABLE ColumnstoreTable1

REBUILD PARTITION = ALL WITH (DATA\_COMPRESSION = COLUMNSTORE ON PARTITIONS (2,4) ) ;

This next example sets the data compression to columnstore on some partitions, and to columnstore archival on other partitions.

Copy

ALTER TABLE ColumnstoreTable1

REBUILD PARTITION = ALL WITH (

DATA\_COMPRESSION = COLUMNSTORE ON PARTITIONS (4,5),

DATA COMPRESSION = COLUMNSTORE\_ARCHIVE ON PARTITIONS (1,2,3)

) ;

### Performance

Compressing columnstore indexes with archival compression will cause the index to perform slower than columnstore indexes that do not have the archival compression. Use archival compression only when you can afford to use extra time and CPU resources to compress and retrieve the data.

The benefit of slower performance is reduced storage which is useful for data that is not frequently accessed. For example, if you have a partition for each month of data, and most of your activity is for the most recent months, you could archive older months to reduce the storage requirements.

### Metadata

The following system views contain information about data compression for clustered indexes:

* [sys.indexes (Transact-SQL)](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-indexes-transact-sql) - The **type** and **type\_desc** columns include CLUSTERED COLUMNSTORE and NONCLUSTERED COLUMNSTORE.
* [sys.partitions (Transact-SQL)](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-partitions-transact-sql) – The **data\_compression** and **data\_compression\_desc** columns include COLUMNSTORE and COLUMNSTORE\_ARCHIVE.

The procedure [sp\_estimate\_data\_compression\_savings (Transact-SQL)](https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-estimate-data-compression-savings-transact-sql) does not apply to columnstore indexes.

## How Compression Affects Partitioned Tables and Indexes

When you use data compression with partitioned tables and indexes, be aware of the following considerations:

* When partitions are split by using the ALTER PARTITION statement, both partitions inherit the data compression attribute of the original partition.
* When two partitions are merged, the resultant partition inherits the data compression attribute of the destination partition.
* To switch a partition, the data compression property of the partition must match the compression property of the table.
* There are two syntax variations that you can use to modify the compression of a partitioned table or index:
  + The following syntax rebuilds only the referenced partition:

Copy

ALTER TABLE <table\_name>

REBUILD PARTITION = 1 WITH (DATA\_COMPRESSION = <option>)

* + The following syntax rebuilds the whole table by using the existing compression setting for any partitions that are not referenced:

Copy

ALTER TABLE <table\_name>

REBUILD PARTITION = ALL

WITH (DATA\_COMPRESSION = PAGE ON PARTITIONS(<range>),

... )

Partitioned indexes follow the same principle using ALTER INDEX.

* When a clustered index is dropped, the corresponding heap partitions retain their data compression setting unless the partitioning scheme is modified. If the partitioning scheme is changed, all partitions are rebuilt to an uncompressed state. To drop a clustered index and change the partitioning scheme requires the following steps:
  + Drop the clustered index.
  + Modify the table by using the ALTER TABLE ... REBUILD ... option that specifies the compression option.

To drop a clustered index OFFLINE is a very fast operation, because only the upper levels of clustered indexes are removed. When a clustered index is dropped ONLINE, SQL Server must rebuild the heap two times, once for step 1 and once for step 2.

## How Compression Affects Replication

||  
|-|  
|**Applies to**: SQL Server ( SQL Server 2014 through [current version](http://go.microsoft.com/fwlink/p/?LinkId=299658)).|  
When you are using data compression with replication, be aware of the following considerations:

* When the Snapshot Agent generates the initial schema script, the new schema will use the same compression settings for both the table and its indexes. Compression cannot be enabled on just the table and not the index.
* For transactional replication the article schema option determines what dependent objects and properties have to be scripted. For more information, see [sp\_addarticle](https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-addarticle-transact-sql).

The Distribution Agent does not check for down-level Subscribers when it applies scripts. If the replication of compression is selected, creating the table on down-level Subscribers will fail. In the case of a mixed topology, do not enable the replication of compression.

* For merge replication, publication compatibility level overrides the schema options and determines the schema objects that will be scripted.

In the case of a mixed topology, if it is not required to support the new compression options, the publication compatibility level should be set to the down-level Subscriber version. If it is required, compress tables on the Subscriber after they have been created.

The following table shows replication settings that control compression during replication.

| User intent | Replicate partition scheme for a table or index | Replicate compression settings | Scripting behavior |
| --- | --- | --- | --- |
| To replicate the partition scheme and enable compression on the Subscriber on the partition. | True | True | Scripts both the partition scheme and the compression settings. |
| To replicate the partition scheme but not compress the data on the Subscriber. | True | False | Scripts out the partition scheme but not the compression settings for the partition. |
| To not replicate the partition scheme and not compress the data on the Subscriber. | False | False | Does not script partition or compression settings. |
| To compress the table on the Subscriber if all the partitions are compressed on the Publisher, but not replicate the partition scheme. | False | True | Checks if all the partitions are enabled for compression.  Scripts out compression at the table level. |

## How Compression Affects Other SQL Server Components

||  
|-|  
|**Applies to**: SQL Server ( SQL Server 2014 through [current version](http://go.microsoft.com/fwlink/p/?LinkId=299658)).|

Compression occurs in the storage engine and the data is presented to most of the other components of SQL Server in an uncompressed state. This limits the effects of compression on the other components to the following:

* Bulk import and export operations

When data is exported, even in native format, the data is output in the uncompressed row format. This can cause the size of exported data file to be significantly larger than the source data.

When data is imported, if the target table has been enabled for compression, the data is converted by the storage engine into compressed row format. This can cause increased CPU usage compared to when data is imported into an uncompressed table.

When data is bulk imported into a heap with page compression, the bulk import operation will try to compress the data with page compression when the data is inserted.

* Compression does not affect backup and restore.
* Compression does not affect log shipping.
* Data compression is incompatible with sparse columns. Therefore, tables containing sparse columns cannot be compressed nor can sparse columns be added to a compressed table.
* Enabling compression can cause query plans to change because the data is stored using a different number of pages and number of rows per page.

## See Also

[Row Compression Implementation](https://docs.microsoft.com/en-us/sql/relational-databases/data-compression/row-compression-implementation)  
[Page Compression Implementation](https://docs.microsoft.com/en-us/sql/relational-databases/data-compression/page-compression-implementation)  
[Unicode Compression Implementation](https://docs.microsoft.com/en-us/sql/relational-databases/data-compression/unicode-compression-implementation)