<https://docs.microsoft.com/en-us/sql/t-sql/statements/create-columnstore-index-transact-sql>

# CREATE COLUMNSTORE INDEX (Transact-SQL)

2016-12-16 25 min to read Contributors

* [Barbara Kess](https://github.com/barbkess)

* [Gene Milener](https://github.com/MightyPen)

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

**THIS TOPIC APPLIES TO:**yesSQL Server (starting with 2012)yesAzure SQL DatabaseyesAzure SQL Data Warehouse yesParallel Data Warehouse

Convert a rowstore table to a clustered columnstore index or create a nonclustered columnstore index. Use a columnstore index to efficiently run real-time operational analytics on an OLTP workload or to improve data compression and query performance for data warehousing workloads.

Note

Starting with SQL Server 2016, you can create the table as a clustered columnstore index. It is no longer necessary to first create a rowstore table and then convert it to a clustered columnstore index.

Skip to examples:

* [Examples for converting a rowstore table to columnstore](https://docs.microsoft.com/en-us/sql/t-sql/statements/create-columnstore-index-transact-sql#convert)
* [Examples for nonclustered columnstore indexes](https://docs.microsoft.com/en-us/sql/t-sql/statements/create-columnstore-index-transact-sql#nonclustered)

Go to scenarios:

* [Columnstore indexes for real-time operational analytics](https://msdn.microsoft.com/library/dn817827.aspx)
* [Columnstore indexes for data warehousing](https://msdn.microsoft.com/library/dn913734.aspx)

Learn more:

* [Columnstore indexes guide](https://msdn.microsoft.com/library/gg492088.aspx)
* [Columnstore indexes feature summary](https://msdn.microsoft.com/library/dn934994.aspx)

Topic link icon [Transact-SQL Syntax Conventions](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/transact-sql-syntax-conventions-transact-sql)

## Syntax

Copy

-- Syntax for SQL Server and Azure SQL Database

-- Create a clustered columnstore index on disk-based table.

CREATE CLUSTERED COLUMNSTORE INDEX index\_name

ON [database\_name. [schema\_name ] . | schema\_name . ] table\_name

[ WITH ( < with\_option> [ ,...n ] ) ]

[ ON <on\_option> ]

[ ; ]

--Create a non-clustered columnstore index on a disk-based table.

CREATE [NONCLUSTERED] COLUMNSTORE INDEX index\_name

ON [database\_name. [schema\_name ] . | schema\_name . ] table\_name

( column [ ,...n ] )

[ WITH ( < with\_option> [ ,...n ] ) ]

[ ON <on\_option> ]

[ WHERE <filter\_expression> [ AND <filter\_expression> ] ]

[ ; ]

<with\_option> ::=

DROP\_EXISTING = { ON | OFF } -- default is OFF

| MAXDOP = max\_degree\_of\_parallelism

| ONLINE = { ON | OFF }

| COMPRESSION\_DELAY = { 0 | delay [ Minutes ] }

| DATA\_COMPRESSION = { COLUMNSTORE | COLUMNSTORE\_ARCHIVE }

[ ON PARTITIONS ( { partition\_number\_expression | range } [ ,...n ] ) ]

<on\_option>::=

partition\_scheme\_name ( column\_name )

| filegroup\_name

| "default"

<filter\_expression> ::=

column\_name IN ( constant [ ,...n ]

| column\_name { IS | IS NOT | = | <> | != | > | >= | !> | < | <= | !< } constant

Copy

-- Syntax for Azure SQL Data Warehouse and Parallel Data Warehouse

CREATE CLUSTERED COLUMNSTORE INDEX index\_name

ON [ database\_name . [ schema\_name ] . | schema\_name . ] table\_name

[ WITH ( DROP\_EXISTING = { ON | OFF } ) ] --default is OFF

[;]

## Arguments

CREATE CLUSTERED COLUMNSTORE INDEX  
Create a clustered columnstore index in which all of the data is compressed and stored by column. The index includes all of the columns in the table, and stores the entire table. If the existing table is a heap or clustered index, the table will be converted to a clustered columnstore index. If the table is already stored as a clustered columnstore index, the existing index will be dropped and rebuilt.

index\_name  
Specifies the name for the new index.

If the table already has a clustered columnstore index, you can specify the same name as the existing index, or you can use the DROP EXISTING option to specify a new name.

ON [database\_name. [schema\_name ] . | schema\_name . ] table\_name  
Specifies the one-, two-, or three-part name of the table to be stored as a clustered columnstore index. If the table is a heap or clustered index the table will be converted from rowstore to a columnstore. If the table is already a columnstore, this statement will rebuild the clustered columnstore index.

WITH  
DROP\_EXISTING = [OFF] | ON  
DROP\_EXISTING = ON specifies to drop the existing clustered columnstore index, and create a new columnstore index.

The default, DROP\_EXISTING = OFF expects the index name is the same as the existing name. An error will occur is the specified index name already exists.

MAXDOP = max\_degree\_of\_parallelism  
Overrides the existing maximum degree of parallelism server configuration for the duration of the index operation. Use MAXDOP to limit the number of processors used in a parallel plan execution. The maximum is 64 processors.

max\_degree\_of\_parallelism values can be:

* 1 - Suppress parallel plan generation.
* >1 - Restrict the maximum number of processors used in a parallel index operation to the specified number or fewer based on the current system workload. For example, when MAXDOP = 4, the number of processors used will be 4 or less.
* 0 (default) - Use the actual number of processors or fewer based on the current system workload.

For more information, see [Configure the max degree of parallelism Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/configure-the-max-degree-of-parallelism-server-configuration-option), and [Configure Parallel Index Operations](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/configure-parallel-index-operations).

COMPRESSION\_DELAY = **0** | delay [ Minutes ]  
This feature will be available after SQL Server 2016 CTP 3.3

For a disk-based table, delay specifies the minimum number of minutes a delta rowgroup in the CLOSED state must remain in the delta rowgroup before SQL Server can compress it into the compressed rowgroup. Since disk-based tables don't track insert and update times on individual rows, SQL Server applies the delay to delta rowgroups in the CLOSED state.

The default is 0 minutes.

For recommendations on when to use COMPRESSION\_DELAY, see [Get started with Columnstore for real time operational analytics](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/get-started-with-columnstore-for-real-time-operational-analytics).

DATA\_COMPRESSION = **COLUMNSTORE** | COLUMNSTORE\_ARCHIVE  
||  
|-|  
|**Applies to**: SQL Server 2016 through SQL Server 2017.|

Specifies the data compression option for the specified table, partition number, or range of partitions. The options are as follows:

COLUMNSTORE  
COLUMNSTORE is the default and specifies to compress with the most performant columnstore compression. This is the typical choice.

COLUMNSTORE\_ARCHIVE  
COLUMNSTORE\_ARCHIVE will further compress the table or partition to a smaller size. Use this option for situations such as archival that require a smaller storage size and can afford more time for storage and retrieval.

For more information about compression, see [Data Compression](https://docs.microsoft.com/en-us/sql/relational-databases/data-compression/data-compression).

ON  
With the ON options you can specify options for data storage, such as a partition scheme, a specific filegroup, or the default filegroup. If the ON option is not specified, the index will use the settings partition or filegroup settings of the existing table.

partition\_scheme\_name **(** column\_name **)**  
Specifies the partition scheme for the table. The partition scheme must already exist in the database. To create the partition scheme, see [CREATE PARTITION SCHEME](https://docs.microsoft.com/en-us/sql/t-sql/statements/create-partition-scheme-transact-sql).

column\_name specifies the column against which a partitioned index will be partitioned. This column must match the data type, length, and precision of the argument of the partition function that partition\_scheme\_name is using.

filegroup\_name  
Specifies the filegroup for storing the clustered columnstore index. If no location is specified and the table is not partitioned, the index uses the same filegroup as the underlying table or view. The filegroup must already exist.

**"**default**"**  
To create the index on the default filegoup, use "default" or [ default ].

If "default" is specified, the QUOTED\_IDENTIFIER option must be ON for the current session. QUOTED\_IDENTIFIER is ON by default. For more information, see [SET QUOTED\_IDENTIFIER (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/statements/set-quoted-identifier-transact-sql).

CREATE [**NONCLUSTERED** ] COLUMNSTORE INDEX  
Create an in-memory nonclustered columnstore index on a rowstore table stored as a heap or clustered index. The index can have a filtered condition and does not need to include all of the columns of the underlying table. The columnstore index requires enough space to store a copy of the data. It is updateable and will be updated as the underlying table is changed. The nonclustered columnstore index on a clustered index enables real-time analytics.

index\_name  
Specifies the name of the index. index\_name must be unique within the table, but does not have to be unique within the database. Index names must follow the rules of [identifiers](https://docs.microsoft.com/en-us/sql/relational-databases/databases/database-identifiers).

**(** column [ **,**...n ] **)**  
Specifies the columns to store. A nonclustered columnstore index is limited to 1024 columns.

Each column must be of a supported data type for columnstore indexes. See [Limitations and Restrictions](https://docs.microsoft.com/en-us/sql/t-sql/statements/create-columnstore-index-transact-sql#LimitRest) for a list of the supported data types.

ON [database\_name. [schema\_name ] . | schema\_name . ] table\_name  
Specifies the one-, two-, or three-part name of the table that will contain the index.

WITH

DROP\_EXISTING = [OFF] | ON  
DROP\_EXISTING = ON The existing index is dropped and rebuilt. The index name specified must be the same as a currently existing index; however, the index definition can be modified. For example, you can specify different columns, or index options.

DROP\_EXISTING = OFF An error is displayed if the specified index name already exists. The index type cannot be changed by using DROP\_EXISTING. In backward compatible syntax, WITH DROP\_EXISTING is equivalent to WITH DROP\_EXISTING = ON.

MAXDOP = max\_degree\_of\_parallelism  
Overrides the [Configure the max degree of parallelism Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/configure-the-max-degree-of-parallelism-server-configuration-option) configuration option for the duration of the index operation. Use MAXDOP to limit the number of processors used in a parallel plan execution. The maximum is 64 processors.

max\_degree\_of\_parallelism values can be:

* 1 - Suppress parallel plan generation.
* >1 - Restrict the maximum number of processors used in a parallel index operation to the specified number or fewer based on the current system workload. For example, when MAXDOP = 4, the number of processors used will be 4 or less.
* 0 (default) - Use the actual number of processors or fewer based on the current system workload.

For more information, see [Configure Parallel Index Operations](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/configure-parallel-index-operations).

Note

Parallel index operations are not available in every edition of Microsoft SQL Server. For a list of features that are supported by the editions of SQL Server, see [Editions and Supported Features for SQL Server 2016](https://docs.microsoft.com/en-us/sql/sql-server/editions-and-supported-features-for-sql-server-2016).

ONLINE = [ON | OFF]  
||  
|-|  
|**Applies to**: SQL Server 2017, in nonclustered columnstore indexes only.|  
ON specifies that the nonclustered columnstore index will remain online and available while the new copy of the index is being built.

OFF specifies that the index will not be available for use while the new copy is being built. As this is a nonclustered index only, the base table will remain available, only the nonclustered columnstore index will not be used to satisfy queries until the new index is complete.

COMPRESSION\_DELAY = **0** | <delay>[Minutes]  
This feature will be available after SQL Server 2016 CTP 3.3

Specifies a lower bound on how long a row should stay in delta rowgroup before it is eligible for migration to compressed rowgroup. For example, a customer can say that if a row is unchanged for 120 minutes, make it eligible for compressing into columnar storage format. For columnstore index on disk-based tables, we don’t track the time when a row was inserted or updated, we use the delta rowgroup closed time as a proxy for the row instead. The default duration is 0 minutes. A row is migrated to columnar storage once 1 million rows have been accumulated in delta rowgroup and it has been marked closed.

DATA\_COMPRESSION  
Specifies the data compression option for the specified table, partition number, or range of partitions. The options are as follows:

COLUMNSTORE  
||  
|-|  
|**Applies to**: SQL Server 2016 through SQL Server 2017.|

Applies only to columnstore indexes, including both nonclustered columnstore and clustered columnstore indexes. COLUMNSTORE is the default and specifies to compress with the most performant columnstore compression. This is the typical choice.

COLUMNSTORE\_ARCHIVE  
||  
|-|  
|**Applies to**: SQL Server 2016 through SQL Server 2017.|

Applies only to columnstore indexes, including both nonclustered columnstore and clustered columnstore indexes. COLUMNSTORE\_ARCHIVE will further compress the table or partition to a smaller size. This can be used for archival, or for other situations that require a smaller storage size and can afford more time for storage and retrieval.

For more information about compression, see [Data Compression](https://docs.microsoft.com/en-us/sql/relational-databases/data-compression/data-compression).

WHERE [ AND ]  
||  
|-|  
|**Applies to**: SQL Server 2016 through SQL Server 2017.|

Called a filter predicate, this specifies which rows to include in the index. SQL Server creates filtered statistics on the data rows in the filtered index.

The filter predicate uses simple comparison logic. Comparisons using NULL literals are not allowed with the comparison operators. Use the IS NULL and IS NOT NULL operators instead.

Here are some examples of filter predicates for the Production.BillOfMaterials table:

WHERE StartDate > '20000101' AND EndDate <= '20000630'

WHERE ComponentID IN (533, 324, 753)

WHERE StartDate IN ('20000404', '20000905') AND EndDate IS NOT NULL

For guidance on filtered indexes, see [Create Filtered Indexes](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/create-filtered-indexes).

ON  
These options specify the filegroups on which the index will be created.

partition\_scheme\_name **(** column\_name **)**  
Specifies the partition scheme that defines the filegroups onto which the partitions of a partitioned index will be mapped. The partition scheme must exist within the database by executing [CREATE PARTITION SCHEME](https://docs.microsoft.com/en-us/sql/t-sql/statements/create-partition-scheme-transact-sql). column\_name specifies the column against which a partitioned index will be partitioned. This column must match the data type, length, and precision of the argument of the partition function that partition\_scheme\_name is using. column\_name is not restricted to the columns in the index definition. When partitioning a columnstore index, Database Engine adds the partitioning column as a column of the index, if it is not already specified.

If partition\_scheme\_name or filegroup is not specified and the table is partitioned, the index is placed in the same partition scheme, using the same partitioning column, as the underlying table.

A columnstore index on a partitioned table must be partition aligned.

For more information about partitioning indexes, see [Partitioned Tables and Indexes](https://docs.microsoft.com/en-us/sql/relational-databases/partitions/partitioned-tables-and-indexes).

filegroup\_name  
Specifies a filegroup name on which to create the index. If filegroup\_name is not specified and the table is not partitioned, the index uses the same filegroup as the underlying table. The filegroup must already exist.

**"**default**"**  
Creates the specified index on the default filegroup.

The term default, in this context, is not a keyword. It is an identifier for the default filegroup and must be delimited, as in ON **"**default**"** or ON **[**default**]**. If "default" is specified, the QUOTED\_IDENTIFIER option must be ON for the current session. This is the default setting. For more information, see [SET QUOTED\_IDENTIFIER (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/statements/set-quoted-identifier-transact-sql).

## Permissions

Requires ALTER permission on the table.

## General Remarks

A columnstore index can be created on a temporary table. When the table is dropped or the session ends, the index is also dropped.

## Filtered Indexes

A filtered index is an optimized nonclustered index, suited for queries that select a small percentage of rows from a table. It uses a filter predicate to index a portion of the data in the table. A well-designed filtered index can improve query performance, reduce storage costs, and reduce maintenance costs.

### Required SET Options for Filtered Indexes

The SET options in the Required Value column are required whenever any of the following conditions occur:

* Create a filtered index.
* INSERT, UPDATE, DELETE, or MERGE operation modifies the data in a filtered index.
* The filtered index is used by the query optimizer to produce the query plan.

| SET options | Required value | Default server value | Default  OLE DB and ODBC value | Default  DB-Library value |
| --- | --- | --- | --- | --- |
| ANSI\_NULLS | ON | ON | ON | OFF |
| ANSI\_PADDING | ON | ON | ON | OFF |
| ANSI\_WARNINGS\* | ON | ON | ON | OFF |
| ARITHABORT | ON | ON | OFF | OFF |
| CONCAT\_NULL\_YIELDS\_NULL | ON | ON | ON | OFF |
| NUMERIC\_ROUNDABORT | OFF | OFF | OFF | OFF |
| QUOTED\_IDENTIFIER | ON | ON | ON | OFF |

* \*Setting ANSI\_WARNINGS to ON implicitly sets ARITHABORT to ON when the database compatibility level is set to 90 or higher. If the database compatibility level is set to 80 or earlier, the ARITHABORT option must explicitly be set to ON.
* If the SET options are incorrect, the following conditions can occur:
* The filtered index is not created.
* The Database Engine generates an error and rolls back INSERT, UPDATE, DELETE, or MERGE statements that change data in the index.
* Query optimizer does not consider the index in the execution plan for any Transact-SQL statements.

For more information about Filtered Indexes, see [Create Filtered Indexes](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/create-filtered-indexes).

## Limitations and Restrictions

If the underlying table has a column of a data type that is not supported for columnstore indexes, you must omit that column from the nonclustered columnstore index.

Nonclustered columnstore indexes:

* Cannot have more than 1024 columns.
* A table with a nonclustered columnstore index can have unique constraints, primary key constraints, or foreign key constraints, but the constraints cannot be included in the nonclustered columnstore index.
* Cannot be created on a view or indexed view.
* Cannot include a sparse column.
* Cannot be changed by using the **ALTER INDEX** statement. To change the nonclustered index, you must drop and re-create the columnstore index instead. You can use **ALTER INDEX** to disable and rebuild a columnstore index.
* Cannot be created by using the **INCLUDE** keyword.
* Cannot include the **ASC** or **DESC** keywords for sorting the index. Columnstore indexes are ordered according to the compression algorithms. Sorting would eliminate many of the performance benefits.
* Cannot include large object (LOB) columns of type nvarchar(max), varchar(max), and varbinary(max) in nonclustered column store indexes. Only clustered columnstore indexes support LOB types, beginning in SQL Server 2017 version. Note, prior versions do not support LOB types in clustered and nonclustered columnstore indexes.

**Each column in a columnstore index must be of one of the following common business data types:**

* datetimeoffset [ ( n ) ]
* datetime2 [ ( n ) ]
* datetime
* smalldatetime
* date
* time [ ( n ) ]
* float [ ( n ) ]
* real [ ( n ) ]
* decimal [ ( precision [ , scale ] **)** ]
* numeric [ ( precision [ , scale ] **)** ]
* money
* smallmoney
* bigint
* int
* smallint
* tinyint
* bit
* nvarchar [ ( n ) ]
* nvarchar(max) (Applies to SQL Server 2017, in clustered columnstore indexes only)
* nchar [ ( n ) ]
* varchar [ ( n ) ]
* varchar(max) (Applies to SQL Server 2017, in clustered columnstore indexes only)
* char [ ( n ) ]
* varbinary [ ( n ) ]
* varbinary (max) (Applies to SQL Server 2017, in clustered columnstore indexes only)
* binary [ ( n ) ]
* uniqueidentifier (Applies to SQL Server 2014 and later)

**Columns that use any of the following data types cannot be included in a columnstore index:**

* ntext, text, and image
* nvarchar(max), varchar(max), and varbinary(max) (Applies to SQL Server 2016 and prior versions, and nonclustered columnstore indexes)
* rowversion (and timestamp)
* sql\_variant
* CLR types (hierarchyid and spatial types)
* xml
* uniqueidentifier (Applies to SQL Server 2012)

**Columnstore indexes cannot be combined with the following features:**

* Computed columns
* Page and row compression, and **vardecimal** storage format (A columnstore index is already compressed in a different format.)
* Replication
* Filestream

You cannot use cursors or triggers on a table with a clustered columnstore index. This restriction does not apply to nonclustered columnstore indexes; you can use cursors and triggers on a table with a nonclustered columnstore index.

**SQL Server 2014 limitations**

These limitations apply only to SQL Server 2014. In this release we introduced updateable clustered columnstore indexes. Nonclustered columnstore indexes were still read-only.

* Change tracking. You cannot use change tracking with nonclustered columnstore indexes (NCCI) because they are read-only. It does work for clustered columnstore indexes (CCI).
* Change data capture. You cannot use change data capture for nonclustered columnstore index (NCCI) because they are read-only. It does work for clustered columnstore indexes (CCI).
* Readable secondary. You cannot access a clustered clustered columnstore index (CCI) from a readable secondary of an Always OnReadable availability group. You can access a nonclustered columnstore index (NCCI) from a readable secondary.
* Multiple Active Result Sets (MARS). SQL Server 2014 uses MARS for read-only connections to tables with a columnstore index. However, SQL Server 2014 does not support MARS for concurrent data manipulation language (DML) operations on a table with a columnstore index. When this occurs, SQL Server will terminate the connections and abort the transactions.

For information about the performance benefits and limitations of columnstore indexes, see [Columnstore Indexes Overview](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview).

## Metadata

All of the columns in a columnstore index are stored in the metadata as included columns. The columnstore index does not have key columns. These system views provide information about columnstore indexes.

* [sys.indexes (Transact-SQL)](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-indexes-transact-sql)
* [sys.index\_columns (Transact-SQL)](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-index-columns-transact-sql)
* [sys.partitions (Transact-SQL)](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-partitions-transact-sql)
* [sys.column\_store\_segments (Transact-SQL)](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-column-store-segments-transact-sql)
* [sys.column\_store\_dictionaries (Transact-SQL)](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-column-store-dictionaries-transact-sql)
* [sys.column\_store\_row\_groups (Transact-SQL)](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-column-store-row-groups-transact-sql)

## Examples for converting a rowstore table to columnstore

### A. Convert a heap to a clustered columnstore index

This example creates a table as a heap and then converts it to a clustered columnstore index named cci\_Simple. This changes the storage for the entire table from rowstore to columnstore.

Copy

CREATE TABLE SimpleTable(

ProductKey [int] NOT NULL,

OrderDateKey [int] NOT NULL,

DueDateKey [int] NOT NULL,

ShipDateKey [int] NOT NULL);

GO

CREATE CLUSTERED COLUMNSTORE INDEX cci\_Simple ON SimpleTable;

GO

### B. Convert a clustered index to a clustered columnstore index with the same name.

This example creates a table with clustered index, and then demonstrates the syntax of converting the clustered index to a clustered columnstore index. This changes the storage for the entire table from rowstore to columnstore.

Copy

CREATE TABLE SimpleTable (

ProductKey [int] NOT NULL,

OrderDateKey [int] NOT NULL,

DueDateKey [int] NOT NULL,

ShipDateKey [int] NOT NULL);

GO

CREATE CLUSTERED INDEX cl\_simple ON SimpleTable (ProductKey);

GO

CREATE CLUSTERED COLUMNSTORE INDEX cl\_simple ON SimpleTable

WITH (DROP\_EXISTING = ON);

GO

### C. Handle nonclustered indexes when converting a rowstore table to a columnstore index.

This example shows how to handle nonclustered indexes when converting a rowstore table to a columnstore index. Actually, beginning with SQL Server 2016 no special action is required; SQL Server will automatically define and rebuild the nonclustered indexes on the new clustered columnstore index.

If you want to drop the nonclustered indexes, use the DROP INDEX statement prior to creating the columnstore index. The DROP EXISTING option only drops the clustered index that is being converted. It does not drop the nonclustered indexes.

In SQL Server 2012 and SQL Server 2014, you could not create a nonclustered index on a columnstore index. This example shows how in previous releases you need to drop the nonclustered indexes before creating the columnstore index.

Copy

--Create the table for use with this example.

CREATE TABLE SimpleTable (

ProductKey [int] NOT NULL,

OrderDateKey [int] NOT NULL,

DueDateKey [int] NOT NULL,

ShipDateKey [int] NOT NULL);

GO

--Create two nonclustered indexes for use with this example

CREATE INDEX nc1\_simple ON SimpleTable (OrderDateKey);

CREATE INDEX nc2\_simple ON SimpleTable (DueDateKey);

GO

--SQL Server 2012 and SQL Server 2014: you need to drop the nonclustered indexes

--in order to create the columnstore index.

DROP INDEX SimpleTable.nc1\_simple;

DROP INDEX SimpleTable.nc2\_simple;

--Convert the rowstore table to a columnstore index.

CREATE CLUSTERED COLUMNSTORE INDEX cci\_simple ON SimpleTable;

GO

### D. Convert a large fact table from rowstore to columnstore

This example explains how to convert a large fact table from a rowstore table to a columnstore table.

To convert a rowstore table to a columnstore table.

1. First, create a small table to use in this example.

Copy

--Create a rowstore table with a clustered index and a non-clustered index.

CREATE TABLE MyFactTable (

ProductKey [int] NOT NULL,

OrderDateKey [int] NOT NULL,

DueDateKey [int] NOT NULL,

ShipDateKey [int] NOT NULL )

)

WITH (

CLUSTERED INDEX ( ProductKey )

);

--Add a non-clustered index.

CREATE INDEX my\_index ON MyFactTable ( ProductKey, OrderDateKey );

1. Drop all non-clustered indexes from the rowstore table.

Copy

--Drop all non-clustered indexes

DROP INDEX my\_index ON MyFactTable;

1. Drop the clustered index.
   * Do this only if you want to specify a new name for the index when it is converted to a clustered columnstore index. If you do not drop the clustered index, the new clustered columnstore index will have the same name.

Note

The name of the index might be easier to remember if you use your own name. All rowstore clustered indexes use the default name which is 'ClusteredIndex\_<GUID>'.

1. Copy
2. --Process for dropping a clustered index.
3. --First, look up the name of the clustered rowstore index.
4. --Clustered rowstore indexes always use the DEFAULT name ‘ClusteredIndex\_<GUID>’.
5. SELECT i.name
6. FROM sys.indexes i
7. JOIN sys.tables t
8. ON ( i.type\_desc = 'CLUSTERED' ) WHERE t.name = 'MyFactTable';
9. --Drop the clustered rowstore index.
10. DROP INDEX ClusteredIndex\_d473567f7ea04d7aafcac5364c241e09 ON MyDimTable;
11. Convert the rowstore table to a columnstore table with a clustered columnstore index.

Copy

--Option 1: Convert to columnstore and name the new clustered columnstore index MyCCI.

CREATE CLUSTERED COLUMNSTORE INDEX MyCCI ON MyFactTable;

--Option 2: Convert to columnstore and use the rowstore clustered

--index name for the columnstore clustered index name.

--First, look up the name of the clustered rowstore index.

SELECT i.name

FROM sys.indexes i

JOIN sys.tables t

ON ( i.type\_desc = 'CLUSTERED' )

WHERE t.name = 'MyFactTable';

--Second, create the clustered columnstore index and

--Replace ClusteredIndex\_d473567f7ea04d7aafcac5364c241e09

--with the name of your clustered index.

CREATE CLUSTERED COLUMNSTORE INDEX

ClusteredIndex\_d473567f7ea04d7aafcac5364c241e09

ON MyFactTable

WITH DROP\_EXISTING = ON;

### E. Convert a columnstore table to a rowstore table with a clustered index

To convert a columnstore table to a rowstore table with a clustered index, use the CREATE INDEX statement with the DROP\_EXISTING option.

Copy

CREATE CLUSTERED INDEX ci\_MyTable

ON MyFactTable

WITH ( DROP EXISTING = ON );

### F. Convert a columnstore table to a rowstore heap

To convert a columnstore table to a rowstore heap, simply drop the clustered columnstore index.

Copy

DROP INDEX MyCCI

ON MyFactTable;

### G. Defragment by rebuilding the entire clustered columnstore index

Applies to: SQL Server 2014

There are two ways to rebuild the full clustered columnstore index. You can use CREATE CLUSTERED COLUMNSTORE INDEX, or [ALTER INDEX (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-index-transact-sql) and the REBUILD option. Both methods achieve the same results.

Note

Beginning with SQL Server 2016, use ALTER INDEX REORGANIZE instead of rebuilding with the methods described in this example.

Copy

--Determine the Clustered Columnstore Index name of MyDimTable.

SELECT i.object\_id, i.name, t.object\_id, t.name

FROM sys.indexes i

JOIN sys.tables t

ON (i.type\_desc = 'CLUSTERED COLUMNSTORE')

WHERE t.name = 'RowstoreDimTable';

--Rebuild the entire index by using CREATE CLUSTERED INDEX.

CREATE CLUSTERED COLUMNSTORE INDEX my\_CCI

ON MyFactTable

WITH ( DROP\_EXISTING = ON );

--Rebuild the entire index by using ALTER INDEX and the REBUILD option.

ALTER INDEX my\_CCI

ON MyFactTable

REBUILD PARTITION = ALL

WITH ( DROP\_EXISTING = ON );

## Examples for nonclustered columnstore indexes

### A. Create a columnstore index as a secondary index on a rowstore table

This example creates a nonclustered columnstore index on a rowstore table. Only one columnstore index can be created in this situation. The columnstore index requires extra storage since it contains a copy of the data in the rowstore table. This example creates a simple table and a clustered index, and then demonstrates the syntax of creating a nonclustered columnstore index.

Copy

CREATE TABLE SimpleTable

(ProductKey [int] NOT NULL,

OrderDateKey [int] NOT NULL,

DueDateKey [int] NOT NULL,

ShipDateKey [int] NOT NULL);

GO

CREATE CLUSTERED INDEX cl\_simple ON SimpleTable (ProductKey);

GO

CREATE NONCLUSTERED COLUMNSTORE INDEX csindx\_simple

ON SimpleTable

(OrderDateKey, DueDateKey, ShipDateKey);

GO

### B. Create a simple nonclustered columnstore index using all options

The following example demonstrates the syntax of creating a nonclustered columnstore index by using all options.

Copy

CREATE NONCLUSTERED COLUMNSTORE INDEX csindx\_simple

ON SimpleTable

(OrderDateKey, DueDateKey, ShipDateKey)

WITH (DROP\_EXISTING = ON,

MAXDOP = 2)

ON "default"

GO

For a more complex example using partitioned tables, see [Columnstore Indexes Overview](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview).

### C. Create a nonclustered columnstore index with a filtered predicate

The following example creates a filtered nonclustered columnstore index on the Production.BillOfMaterials table in the AdventureWorks2012 database. The filter predicate can include columns that are not key columns in the filtered index. The predicate in this example selects only the rows where EndDate is non-NULL.

Copy

IF EXISTS (SELECT name FROM sys.indexes

WHERE name = N'FIBillOfMaterialsWithEndDate'

AND object\_id = OBJECT\_ID(N'Production.BillOfMaterials'))

DROP INDEX FIBillOfMaterialsWithEndDate

ON Production.BillOfMaterials;

GO

CREATE NONCLUSTERED COLUMNSTORE INDEX "FIBillOfMaterialsWithEndDate"

ON Production.BillOfMaterials (ComponentID, StartDate)

WHERE EndDate IS NOT NULL;

### D. Change the data in a nonclustered columnstore index

||  
|-|  
|**Applies to**: SQL Server 2012 through SQL Server 2014.|

Once you create a nonclustered columnstore index on a table, you cannot directly modify the data in that table. A query with INSERT, UPDATE, DELETE, or MERGE will fail and return an error message. To add or modify the data in the table, you can do one of the following:

* Disable or drop the columnstore index. You can then update the data in the table. If you disable the columnstore index, you can rebuild the columnstore index when you finish updating the data. For example,

Copy

ALTER INDEX mycolumnstoreindex ON mytable DISABLE;

-- update mytable --

ALTER INDEX mycolumnstoreindex on mytable REBUILD

* Load data into a staging table that does not have a columnstore index. Build a columnstore index on the staging table. Switch the staging table into an empty partition of the main table.
* Switch a partition from the table with the columnstore index into an empty staging table. If there is a columnstore index on the staging table, disable the columnstore index. Perform any updates. Build (or rebuild) the columnstore index. Switch the staging table back into the (now empty) partition of the main table.

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### E. Change a clustered index to a clustered columnstore index

By using the CREATE CLUSTERED COLUMNSTORE INDEX statement with DROP\_EXISTING = ON, you can:

* Change a clustered index into a clustered columnstore index.
* Rebuild a clustered columnstore index.

This example creates the xDimProduct table as a rowstore table with a clustered index, and then uses CREATE CLUSTERED COLUMNSTORE INDEX to change the table from a rowstore table to a columnstore table.

Copy

-- Uses AdventureWorks

IF EXISTS (SELECT name FROM sys.tables

WHERE name = N'xDimProduct'

AND object\_id = OBJECT\_ID (N'xDimProduct'))

DROP TABLE xDimProduct;

--Create a distributed table with a clustered index.

CREATE TABLE xDimProduct (ProductKey, ProductAlternateKey, ProductSubcategoryKey)

WITH ( DISTRIBUTION = HASH(ProductKey),

CLUSTERED INDEX (ProductKey) )

AS SELECT ProductKey, ProductAlternateKey, ProductSubcategoryKey FROM DimProduct;

--Change the existing clustered index

--to a clustered columnstore index with the same name.

--Look up the name of the index before running this statement.

CREATE CLUSTERED COLUMNSTORE INDEX <index\_name>

ON xdimProduct

WITH ( DROP\_EXISTING = ON );

### F. Rebuild a clustered columnstore index

Building on the previous example, this example uses CREATE CLUSTERED COLUMNSTORE INDEX to rebuild the existing clustered columnstore index called cci\_xDimProduct.

Copy

--Rebuild the existing clustered columnstore index.

CREATE CLUSTERED COLUMNSTORE INDEX cci\_xDimProduct

ON xdimProduct

WITH ( DROP\_EXISTING = ON );

### G. Change the name of a clustered columnstore index

To change the name of a clustered columnstore index, drop the existing clustered columnstore index, and then recreate the index with a new name.

We recommend only doing this operation with a small table or an empty table. It will take a long time to drop a large clustered columnstore index and rebuild with a different name.

Using the cci\_xDimProduct clustered columnstore index from the previous example, this example drops the cci\_xDimProduct clustered columnstore index and then recreates the clustered columnstore index with the name mycci\_xDimProduct.

Copy

--For illustration purposes, drop the clustered columnstore index.

--The table continues to be distributed, but changes to a heap.

DROP INDEX cci\_xdimProduct ON xDimProduct;

--Create a clustered index with a new name, mycci\_xDimProduct.

CREATE CLUSTERED COLUMNSTORE INDEX mycci\_xDimProduct

ON xdimProduct

WITH ( DROP\_EXISTING = OFF );

### H. Convert a columnstore table to a rowstore table with a clustered index

There might be a situation for which you want to drop a clustered columnstore index and create a clustered index. This stores the table in rowstore format. This example converts a columnstore table to a rowstore table with a clustered index with the same name. None of the data is lost. All data goes to the rowstore table and the columns listed become the key columns in the clustered index.

Copy

--Drop the clustered columnstore index and create a clustered rowstore index.

--All of the columns will be stored in the rowstore clustered index.

--The columns listed will be the included columns in the index.

CREATE CLUSTERED INDEX cci\_xDimProduct

ON xdimProduct (ProductKey, ProductAlternateKey, ProductSubcategoryKey, WeightUnitMeasureCode)

WITH ( DROP\_EXISTING = ON);

### I. Convert a columnstore table back to a rowstore heap

Use [DROP INDEX (SQL Server PDW)](http://msdn.microsoft.com/en-us/f59cab43-9f40-41b4-bfdb-d90e80e9bf32) to drop the clustered columnstore index and convert the table to a rowstore heap. This example converts the cci\_xDimProduct table to a rowstore heap. The table continues to be distributed, but is stored as a heap.

Copy

--Drop the clustered columnstore index. The table continues to be distributed, but changes to a heap.

DROP INDEX cci\_xdimProduct ON xdimProduct;

**1 Comment**

Sign in

**33 people listening**

+ Follow

Post comment as...

[stefan.pascal](https://docs.microsoft.com/en-us/sql/t-sql/statements/create-columnstore-index-transact-sql)

Stefan.Pascal May 9, 2017

The syntax shown above is incorrect!

In order to apply a filter expression to a NONCLUSTERD COLUMNSTORE index you have to use the syntax shown below (WHERE <filter\_expression> has to be specified immediately after the column list):

CREATE [NONCLUSTERED] COLUMNSTORE INDEX index\_name

  ON [database\_name. [schema\_name ] . | schema\_name . ] table\_name

        ( column [ ,...n ] )

    [ WHERE <filter\_expression> [ AND <filter\_expression> ] ]

    [ WITH ( < with\_option> [ ,...n ] ) ]

    [ ON <on\_option> ]

[ ; ]