<http://searchsqlserver.techtarget.com/feature/SQL-Server-2014-columnstore-index-the-good-the-bad-and-the-clustered>

**SQL Server 2014 columnstore index: the good, the bad and the clustered**

With the release of SQL Server 2012, Microsoft introduced the columnstore index, a memory-optimized column-oriented index geared toward data warehouse workloads. Although columnstore indexes offered significant performance gains over their traditional counterparts, they were not without their limitations. Most notably, in SQL Server 2012, there was no way to create clustered columnstore indexes, and the nonclustered columnstore indexes could not be updated. To update the data, you had to drop and rebuild the index or swap out partitions.

With the release of SQL Server 2012, Microsoft introduced the columnstore index, a memory-optimized column-oriented index geared toward data warehouse workloads. Although columnstore indexes offered significant performance gains over their traditional counterparts, they were not without their limitations. Most notably, in SQL Server 2012, there was no way to create clustered columnstore indexes, and the nonclustered columnstore indexes could not be updated. To update the data, you had to drop and rebuild the index or swap out partitions.

for updateable clustered columnstore indexes. With SQL Server 2014, the clustered columnstore index can be used in place of a traditional rowstore clustered index. The clustered columnstore index permits data modifications and bulk load operations. Although it has limitations, the clustered columnstore index still represents a significant step forward in providing an indexing strategy better suited to the enterprise data warehouse.

### The SQL Server 2014 columnstore index

SQL Server 2014 continues to support [nonclustered columnstore indexes](http://searchsqlserver.techtarget.com/tip/xVelocity-Columnstore-Indexes-in-SQL-Server-2012) as well as the clustered option. Much of the same technology drives the two index types. In both cases, data is stored in a columnar format, with each column of data separated into its own segment. The segments are organized into a row group, which can contain over one million rows. If a table exceeds that amount, SQL Server creates additional row groups and breaks the column segments across the row groups.

When saving the index to disk, SQL Server compresses the data at the segment level where data is the most homogenous. As a result, columnstore indexes can achieve higher compression rates than traditional indexes. Higher compression rates translate into better performance, since compressed data means a smaller in-memory footprint. This allows more data to be held in memory and consequently reduces disk I/O.

Queries that target a columnstore index can also use batch-mode processing, a vector-based query execution mechanism tightly integrated with the columnstore index. Batch-mode processing is optimized for columnstore indexes to take full advantage of their structure and in-memory capabilities.

Columnstore indexes are best for data warehouse workloads consisting primarily of read-only queries that analyze large sets of aggregated data. These types of queries usually require full table or index scans to retrieve the necessary information. However, columnstore indexes provide little advantage to queries that rely on seek operations to locate specific information. Frequent updates can also diminish the advantages of a columnstore index. Nonclustered indexes need to be dropped and re-created and clustered index updates may result in poorer query performance. Tables primarily limited to bulk load operations are usually better candidates.

### The clustered columnstore index

Once you've decided a table is a good fit for a columnstore index, you need to determine which type of index to use. A clustered columnstore index is updateable, which can be a big advantage. Also, like any clustered index, the index stores all of the table's data. This makes the table and the index essentially the same, eliminating the need to leave the index to retrieve the table's data.

Since the clustered index includes every column from the table, queries benefit greatly from having all of the data highly compressed. This leads to improved memory utilization and gains in performance. Clustered columnstore indexes are especially well-suited to large fact tables in a data warehouse.

However, clustered columnstore indexes have restrictions. Creating a clustered columnstore index means that you can't define any other index on that table. Nonclustered indexes will need to be dropped. For clustered indexes, you have the option of either dropping them or converting them to a columnstore index.

In addition, when using clustered columnstore indexes, the table cannot be configured with any unique, primary key or foreign key constraints. You also cannot create the index on a table that includes computed columns, sparse columns or non-supported data types, such as text, XML or varchar(max). If the table contains these elements, you have to use a nonclustered index that excludes those columns. You also cannot sort the index or combine it with features like replication, filestreaming or change data capture. (Refer to SQL Server Books Online for complete details about the many restrictions.)

### Updating the clustered columnstore index

SQL Server lets you use any standard method to modify or insert data into a clustered columnstore index. Since the index is updateable, you can insert, delete, update or bulk-load data, without having to drop the index. To facilitate data modifications, a clustered columnstore index uses one or more deltastores*,* temporary rowstore tables that hold data until changes can be committed to the columnstore index.

As with any index, there will be times when you'll want to rebuild your clustered columnstore index, such as after heavy data modifications or load operations. If your table is partitioned, consider rebuilding the index one partition at a time rather the entire table at once.

When you rebuild an index, SQL Server defragments the segments and physically deletes the rows that have been logically deleted. Data from the deltastores is then merged into the index and the segments are recompressed. SQL Server also lets you reorganize a clustered columnstore index when necessary.