SQL ServerCentral.com
Proactively monitor your growing SQL Server environment.
Try free SQL Monitor
a community service from redgate
Try SQL Clone

Daily SQL Articles by email:
your@email.com
Sign up

Write For Us
Find out more

Proactively monitor your growing SQL Server environment.
Try free
SQL Monitor

Improve your SQL Server knowledge daily with more articles by email.
your@email.com
Sign up

Thank this author by sharing: G+ 0
Rate this | Join the discussion | Add to briefcase

Create, Alter, and Drop Indexes: Stairway to SQL Server Indexes Level 12

By David Durant, 2012/02/17

The Series

This article is part of the Stairway Series: Stairway to SQL Server Indexes

Indexes are fundamental to database design, and tell the developer using the database a great deal about the intentions of the designer. Unfortunately indexes are too often added as an afterthought when performance issues appear. Here at last is a simple series of articles that should bring any database professional rapidly 'up to speed' with them

In Level 10 we looked at the internal structure of an index; and in Level 11 we looked at a potential side effect of the balanced tree structure: index fragmentation. With this knowledge of index structure in mind, we can examine what happens when an index is impacted by a Data Definition Language statement or by a Data Manipulation Language statement. In this Level we examine the impact of the three DDL verbs; CREATE, ALTER and DROP. In the next Level, we cover the DML verbs; INSERT, UPDATE and DELETE.

Creating, altering and dropping indexes falls under the subject of *index maintenance*. CREATE, ALTER and DROP are the verbs for index maintenance only because the SQL Server team wanted all objects to use the same DDL verbs. As you will see in this Level, when working with indexes these verbs do more than their names imply; enabling you to create, rebuild, reorganize, disable, and delete indexes; as well as modify their metadata.

When you create or alter an index, you have a choice of options to specify. These option values become part of the index's metadata and are stored system tables that are exposed to you through the *sys.indexes* system view. They are used by SQL Server when querying and updating data; and during index maintenance. Throughout this Level we mention several of these options without going into detailed specifics at that point. Rather, a discussion of all the options appears near the end of this Level.

The larger your tables, the greater the impact of executing DDL statements against indexes; both in the consumption of server resources and in the slowing of other queries that are executing

Related Articles
FORUM Rebuild or Reorganize Indexes
2010/03/15
Rebuild or Reorganize Indexes
FORUM Reorganize Index Task
2012/04/10
Reorganize Index Task
FORUM Rebuild,Reorganize index job failed
2010/03/01
SCRIPT
FORUM
Tags

Fresh articles daily:
Get the SQL Server Central newsletter and get a new SQL Server article each day. Get Database Weekly for a roundup of all the biggest SQL news from around the web.
your@email.com
Sign up
No thanks

Copyright © 2002-2012 Redgate. All Rights Reserved. Privacy Policy. Terms of Use. Report Abuse.

at the same time. By knowing what happens within your indexes during DDL and DML operations, you can:

- Understand the need for regular index maintenance.
- Maximize the performance of your maintenance operations.
- Minimize the impact of your maintenance operations on other queries.
- Reduce the frequency of this maintenance.

We begin with creating a clustered index; creating nonclustered indexes will come later.

What happens when you create a clustered index depends upon the current state of the table and the CREATE INDEX options that you specify.

If:

The table already is a clustered index:

An error is raised. A table cannot have two clustered indexes, for that would cause its rows to be in two different sequences at the same time; and that is not possible.

The table is empty:

SQL Server updates the system tables to reflect the existence of the clustered index. No space is allocated at this time.

The table has rows. The table does not have nonclustered indexes:

SQL Server updates the system tables to reflect the existence of the clustered index.

SQL Server sorts the rows of the table into index key sequence, places those rows into pages in compliance with your FILLFACTOR specification, and generates the non-leaf levels of the index. External fragmentation will be near zero.

The table has rows. The table has nonclustered indexes:

SQL Server frees all space occupied by the nonclustered indexes, but retains their metadata.

SQL Server updates the system tables to reflect the existence of the clustered index.

SQL Server builds the clustered index. (See previous condition.)

The nonclustered indexes are rebuilt in accordance with their current metadata. There is no other choice; the nonclustered indexes must be completely rebuilt. The bookmark portion of each entry was a row id; now it will be the clustered index's key value. Thus, each new index entry will be a different size than the old one.

Therefore, if you are creating several indexes on a table, save time and effort by creating the clustered index first, and then creating the nonclustered indexes.

If:

The table is empty:

SQL Server updates the system tables to reflect the existence of the nonclustered index. No space is allocated at this time.

The table has rows:

SQL Server updates the system tables to reflect the existence of the nonclustered index.

SQL Server scans the table, or some other nonclustered index that contains the needed columns, builds the index entry for each row, sorts the entries into index key sequence, places those rows into pages in compliance with your FILLFACTOR option, and generates the non-leaf levels of the index. External fragmentation will be near zero.

ALTER INDEX allows you to do four things:

Disable the index.

Rebuild the index.

Reorganize the index.

Change the index option settings.

Note: One thing ALTER INDEX does not allow you to do is change the set of columns that comprises the index. This can only be done by dropping the index and then creating a new one of the same name containing the desired columns, or by executing CREATE INDEX with the DROP_EXISTING option specified.

To disable an index, simply use the DISABLE keyword; as in

```
ALTER INDEX PK_FragTest_PKCol
        ON FragTest
        DISABLE;
GO
```

Disabling an index does not remove the definition of the index from the system tables. All disabled indexes can be rebuilt or dropped at a later time.

Disabling a nonclustered index frees the disk space occupied by the index. While a nonclustered index is disabled, all queries behave as if the index did not exist.

Disabling a clustered index frees the disk space occupied by the non-leaf levels of the index, but leaves the rows of the table untouched. While a clustered index is disabled, the table cannot be queried or updated.

Because disabling an index is done by releasing disk space, the process requires minimal processor time and log file IO.

The primary reason for disabling an index is to save disk space when rebuilding the index. If the index has not been disabled, the rebuild process maintains the original version of the index until the new version has been created; resulting in both versions consuming disk space at the same time. By deleting the index first; an entire index's worth of disk space is saved during the rebuild process. Rebuilding an index that has been deleted typically requires about one fifth the disk space that rebuilding an undeleted index requires.

Rebuilding an index recreates the index while allowing you change the option settings, as in:

```
ALTER INDEX PK_FragTest_PKCol
         ON FragTest
         REBUILD
      WITH ( FILLFACTOR = 75
           , SORT_IN_TEMPDB = ON
           , MAXDOP = 3 );
```

Only those options specified in the ALTER INDEX statement will be updated in the index's metadata; all others will remain unchanged.

Thus, after the index has been rebuilt, the external fragmentation will be near zero, and the page fullness will be at or near the FILLFACTOR value you specified in the ALTER INDEX statement. If no FILLFACTOR value was specified in the ALTER INDEX statement, the FILLFACTOR value from the index's metadata will be used.

Reorganizing an index has one target: fragmentation. (See Level 11 – Index Fragmentation.) It is used to reduce external fragmentation, and possibly bring page fullness, closer to the FILLFACTOR value specified in the index's metadata. As you will see, reorganizing an index provides fewer options than rebuilding the index, but reorganizing also requires fewer resources and has less impact on other users.

Four things to keep in mind about reorganizing an index:

Reorganizing and index never increases the size of the index; no new extents are assigned to the index. Reorganizing and index may decrease the size of the index, freeing some no longer needed pages.

Other processes can use the index while it is being reorganized.

The only option that can be specified when reorganizing and index is LOB_COMPACTION. Thus, a new FILLFACTOR value cannot be specified when reorganizing an index.

Reorganizing and index requires that page locks be allowed; which is the default value. Because other processes can use the index while it is being reorganized, it is essential that SQL Server lock individual pages of the index, rather than the entire index, during the reorganization. If the index's ALLOW_PAGE_LOCKS option has be set to OFF, it will not reorganize.

Thus, a typical reorganize statement would look like

```
ALTER INDEX PK_FragTest_PKCol
         ON FragTest
         REORGANIZE;
```

Or

```
ALTER INDEX PK_FragTest_PKCol
         ON FragTest
         REORGANIZE
      WITH ( LOB_COMPACTION = OFF );
```

SQL Server reorganizes an index in two phases.

Phase 1 addresses internal fragmentation.

Phase 1 is limited in what it can do because the reorganization processes is prohibited from increasing the size of the index; it can free pages, but it cannot add pages. Thus, an index whose average page fullness is less than the FILLFACTOR specified in the index's metadata can shrink in size when being reorganized, but an index whose average page fullness is greater than the FILLFACTOR specified in its metadata cannot grow in size as the result of reorganization.

Phase 1 processes the index in logical sequence, eight pages at a time, starting with pages 1 through 8, then 2 through 9, then 3 through 10, until the entire index has been examined. For each set of eight pages, SQL Server checks to see if the entries on those eight pages could be squeezed into seven pages without exceeding the index's FILLFACTOR. If so, it will compress the entries into seven pages and free the eighth page.

Phase 2 addresses external fragmentation.

Phase 2 aligns the index's physical sequence with its logical sequence, one page at a time. It reads the index's logically first page and its physically first page and, if they are not the same page, swaps their contents. It repeats this process for each succeeding page of the two sequences. When it completes the last page swap, the two sequences are identical and external fragmentation has been minimized.

The end result is an index with minimal external fragmentation and a possible reduction in internal fragmentation.

This benefit, although limited in functionality when compared to a rebuild, is achieved without the need for extra disk space; with a small amount of memory consumption; and while the index is still available for use by other applications.

So the choice is yours whenever index fragmentation has become a problem: rebuild, disable and rebuild, or reorganize. In Level 15 – Best Patterns and Practices, we offer some recommendations.

Some of an index's option settings can be modified without having to rebuild or reorganize an index.

```
ALLOW_ROW_LOCKS
ALLOW_PAGE_LOCKS
IGNORE_DUP_KEY
STATISTICS_NORECOMPUTE.
```

The following SQL statement is example of altering an index to change settings only.

```
ALTER INDEX PK_FragTest_PKCol
         ON FragTest
      SET ( ALLOW_ROW_lOCKS = ON
          , ALLOW_PAGE_lOCKS = ON
          , STATISTICS_NORECOMPUTE = OFF );
GO
```

When you drop an index, its space allocations are released, making the disk space available for use by other objects. And the index's metadata is deleted from the system tables.

As was mentioned in Level 8 – Unique Indexes, you cannot drop an index that is supporting a primary key or unique constraint.

It is important to note that dropping a clustered index does not drop the underlying table. The non-leaf levels' space allocations are released. But the leaf level, which is the rows of the table, remains. The table reverts to being a heap, which means that any nonclustered indexes will be rebuilt automatically.

Therefore, if you are dropping several indexes, one of which is the clustered index, save time and effort by dropping the nonclustered indexes first, then dropping the clustered index.

The CREATE INDEX options can be divided into three categories:

1. Those options that impact index creation, but not subsequent index usage. Most options fall into this category.
2. Those options that impact subsequent index usage, but not index creation, referred to as *post-creation* options in this Stairway. ALLOW_ROW_LOCKS and ALLOW_PAGE_LOCKS fall into this category.
3. Those options that affect both index creation and post-creation activity. DATA_COMPRESSION is an example of this.

The options are listed below. Unless otherwise stated, they impact index creation only.

FILLFACTOR:

Allows you to specify the desired page fullness. Impacts the leaf-level pages only. The default value is zero, which does not mean "leave the page empty". Rather, it specifies "allow enough empty space per page for one row to be inserted".

PAD_INDEX:

Specifies whether the FILLFACTOR will be applied to the non-leaf levels, in addition to the leaf levels.

SORT_IN_TEMPDB:

Specifies whether the sorting required to create the index will be done in the table's database or in tempdb.

IGNORE_DUP_KEY:

Covered in Level 8 – Unique Indexes.

STATISTICS_NORECOMPUTE:

Will be covered in Level 14 – Index Statistics.

DROP_EXISTING:

Note: DROP_EXISTING is only applicable with CREATE INDEX.

DROP_EXISTING = ON:

If an index of the same name and index type (clustered or nonclustered) exists, that index will be dropped and then a new index of the same name with the specified definition will be created. Dropping the old index means that all space occupied by the old index will be freed, and all metadata for the old index will be removed, before creation of the new index begins. If you

want the new index to have the same options, such as FILLFACTOR, as the old index; you must specify those option values in the CREATE INDEX statement.

If an index of the same name but different index type exists, an error will be raised.

If no index of the same name exists, a new index with the specified definition will be created.

DROP_EXISTING = OFF:

If an index of the same name exists, an error will be raised.

If no index of the same name exists, a new index with the specified definition will be created.

ONLINE:

Specifies whether the index's table is accessible from other connections while the index is being created. If a non-clustered index is being created, SELECT statements will able to access the table regardless of the setting. Available only in SQL Server Enterprise, Developer, and Evaluation editions.

ALLOW_ROW_LOCKS and ALLOW_PAGE_LOCKS:

Added in SQL Server 2005 to give database owners some control over *lock escalation*. Influences locking within the index, as shown in Table 11.1.

Settings	Allow Row Locks	Allow Page Locks	Allow Index Lock
Both ON	Yes	Yes	Yes
ROW off – PAGE on	No	Yes	Yes
ROW on – PAGE off	Yes	No	Yes
Both OFF	No	No	Yes

Table 1: Index Locking

These options do not affect the creation of the index; they are post-creation options. And they have limited impact if row versioning is used as the database's concurrency model.

Reorganizing and index requires that page locks be allowed.

Turning off either, or both, option can reduce the amount of lock escalation that occurs during heavy query loads. Indexes that are most likely to benefit from this are ones that are heavily queried but infrequently updated.

Only specify these options if you are an experienced database owner who understands SQL Server's locking mechanism, and who has stress tested the impact of your proposed settings.

MAXDOP:

Used to control the maximum number of processors that can be used in creating the index.

DATA_COMPRESSION:

Specifies which variant of data compression, if any, will be applied to this index. Affects both index creation and post-creation activity. Data compression is beyond the scope of this Stairway.