Article: Reducing Index Fragmentation
Date: 08/03/2012
Posted by: HeelpBook Staff
Source: Link
Permalink: Link

SQL Server – Reducing Index Fragmentation

General Concepts

When you perform any data modification operations (***INSERT, UPDATE, or DELETE statements***) table fragmentation can occur. When changes are made to the data that affect the index, index fragmentation can occur and the information in the index can get scattered in the database.

Fragmented data can cause **SQL Server** to perform unnecessary data reads, so a ***queries performance against a heavy fragmented table*** can be very poor. If you want to determine the level of fragmentation, you can use the **DBCC SHOWCONTIG** statement. The **DBCC SHOWCONTIG** statement displays fragmentation information for the data and indexes of the specified table or view.

The **DBCC SHOWCONTIG** statement cannot automatically show fragmentation of all the indexes on all the tables in a database it can only work on one table at a time. You can write your own script to show fragmentation of all the tables in a database or you can use the script below (this script shows fragmentation of all the tables in the pubs database):

```
USE pubs
DECLARE @TableName sysname
DECLARE cur_showfragmentation CURSOR FOR
SELECT table_name FROM information_schema.tables WHERE table_type = 'base table'
OPEN cur_showfragmentation
FETCH NEXT FROM cur_showfragmentation INTO @TableName
WHILE @@FETCH_STATUS = 0
BEGIN
  SELECT 'Show fragmentation for the ' + @TableName + ' table'
  DBCC SHOWCONTIG (@TableName)
  FETCH NEXT FROM cur_showfragmentation INTO @TableName
END
CLOSE cur_showfragmentation
DEALLOCATE cur_showfragmentation
```

When you need to perform the same actions for all the tables in a database (***when you need to show the fragmentation of all the tables in a database, as in the script above***), you can create cursor for this purpose, or you can use the **sp_MSforeachtable** undocumented system stored procedure to accomplish the same goal with less work. The following script shows fragmentation of all the tables in the pubs database:

```
USE pubs
GO
EXEC sp_MSforeachtable @command1="print '?' DBCC SHOWCONTIG('?')"
```

```
GO
```

Keep in mind that the undocumented stored procedures could not be supported in the future **SQL Server** versions. So, you can use the **sp_MSforeachtable** undocumented system stored procedure at your own risk.

You can reduce fragmentation and improve read-ahead performance by using one of the following:

- Dropping and re-creating an index.
- Rebuilding an index by using the **DBCC DBREINDEX** statement.
- Defragmenting an index by using the **DBCC INDEXDEFRAG** statement.

Dropping and Re-creating an Index

This is a slowest way to reduce fragmentation, but dropping and re-creation of an index can provide best performance. Because the leaf node of a nonclustered index contains a clustered index key if the table has a clustered index, when a clustered index is deleted on a table that has nonclustered indexes, the nonclustered indexes are all rebuilt as part of the **DROP** operation. So, when you create a clustered index on a table with several secondary indexes all of the secondary indexes must be rebuilt so that the leaf nodes contain the clustering key value instead of the row identifier. This can take significant time on a large table.

So, if you need to drop and re-create both clustered and nonclustered indexes drop the ***nonclustered indexes*** first and the clustered index last, and then create clustered index first and the nonclustered indexes last.

The main disadvantages of this method to reduce fragmentation is that during dropping and recreating a clustered index:

- An **exclusive table lock** is put on the table, preventing any table access by your users.
- A **shared table lock** is put on the table, preventing all but **SELECT** operations to be performed on it.

When you create a clustered index, the table will be copied, the data in the table will be sorted, and then the original table will be deleted. So, you should have enough empty space to hold a copy of the data.

Rebuilding an Index

Rebuilding an index is a more efficient way to reduce fragmentation in comparison with dropping and re-creating an index, this is because rebuilding an index is done by one statement which is easier than coding multiple **DROP INDEX** and **CREATE INDEX**statements.

To rebuild indexes, you can use the **DBCC DBREINDEX** statement.

The **DBCC DBREINDEX** is automatically atomic (***automatically atomic means that the work is done by one statement and you don't need to do anything with this statement to be atomic***). Because the work is done by one statement, it can take advantage of more optimizations with **DBCC DBREINDEX** than it can with individual **DROP INDEX** and **CREATE INDEX** statements.

You can rebuild all the indexes on all the tables in your database periodically (***for example, one time per week at Sunday***) to reduce fragmentation. The **DBCC DBREINDEX** statement cannot automatically rebuild all of the indexes on all the tables in a database it can only work on one table at a time.

You can write your own script to rebuild all the indexes on all the tables in a database or you can use the script below (the **ind_rebuild** stored procedure rebuilds all indexes with a ***fillfactor of 80*** in every table in the current database):

```
CREATE PROC ind_rebuild
AS
DECLARE @TableName sysname
DECLARE cur_reindex CURSOR FOR
SELECT table_name
  FROM information_schema.tables
  WHERE table_type = 'base table'
OPEN cur_reindex
FETCH NEXT FROM cur_reindex INTO @TableName
WHILE @@FETCH_STATUS = 0
BEGIN
  PRINT 'Reindexing ' + @TableName + ' table'
  DBCC DBREINDEX (@TableName, ' ', 80)
  FETCH NEXT FROM cur_reindex INTO @TableName
END
CLOSE cur_reindex
DEALLOCATE cur_reindex
GO
```

You can use the **sp_MSforeachtable** undocumented system stored procedure to accomplish the same goal with less work. This script rebuilds all indexes with a fillfactor of 80 in every table in the pubs database:

```
USE pubs
GO
EXEC sp_MSforeachtable @command1="print '?' DBCC DBREINDEX ('?', ' ', 80)"
GO
```

During rebuilding a clustered index, an exclusive table lock is put on the table, preventing any table access by your users, and during rebuilding a nonclustered index a shared table lock is put on the table, preventing all but **SELECT** operations to be performed on it, you should schedule **DBCC DBREINDEX** statement during **CPU** idle time and slow production periods.

Defragmenting an Index

SQL Server 2000 introduces a new **DBCC INDEXDEFRAG** statement to defragment clustered and nonclustered indexes on tables and views. This statement defragments the leaf level of the index so that the physical order of the index pages match the left-to-right logical order of the leaf nodes.

The **DBCC INDEXDEFRAG** statement will report to the user an estimated percentage completed every five minutes and can be terminated at any point in the process, so that any completed work is retained.

The main advantage of using **DBCC INDEXDEFRAG** in comparison with **DBCC DBREINDEX** or with dropping and re-creating indexes is that the **DBCC INDEXDEFRAG** is an online operation. This means that the **DBCC INDEXDEFRAG** statement does not hold locks for a long time and thus will not block any running queries or updates. As the time to defragment is related to the amount of fragmentation, you can use the **DBCC INDEXDEFRAG** statement to reduce fragmentation if the index is not very fragmented.

For a very fragmented index, rebuilding (using **DBCC DBREINDEX** statement) can take less time. You can defragment all the indexes on all the tables in your database periodically (***for example, one time per week at Sunday***) to reduce fragmentation.

The **DBCC INDEXDEFRAG** statement cannot automatically defragment all of the indexes on all the tables in a database; it can only work on one table and one index at a time. You can use the script below to defragment all indexes in every table in the pubs database:

```
USE pubs
DECLARE @TableName sysname
DECLARE @indid int
DECLARE cur_tblfetch CURSOR FOR
SELECT table_name FROM information_schema.tables WHERE table_type = 'base table'
OPEN cur_tblfetch
FETCH NEXT FROM cur_tblfetch INTO @TableName
WHILE @@FETCH_STATUS = 0
BEGIN
DECLARE cur_indfetch CURSOR FOR
SELECT indid FROM sysindexes WHERE id = OBJECT_ID (@TableName) and keycnt > 0
OPEN cur_indfetch
FETCH NEXT FROM cur_indfetch INTO @indid
WHILE @@FETCH_STATUS = 0
BEGIN
  SELECT 'Derfagmenting index_id = ' + convert(char(3), @indid) + 'of the '
          + rtrim(@TableName) + ' table'
  IF @indid <> 255 DBCC INDEXDEFRAG (pubs, @TableName, @indid)
  FETCH NEXT FROM cur_indfetch INTO @indid
END
CLOSE cur_indfetch
DEALLOCATE cur_indfetch
  FETCH NEXT FROM cur_tblfetch INTO @TableName
END
CLOSE cur_tblfetch
DEALLOCATE cur_tblfetch
```

Though the **DBCC INDEXDEFRAG**, statement is an online operation, try to schedule it during **CPU** idle time and slow production periods as other maintenance tasks.