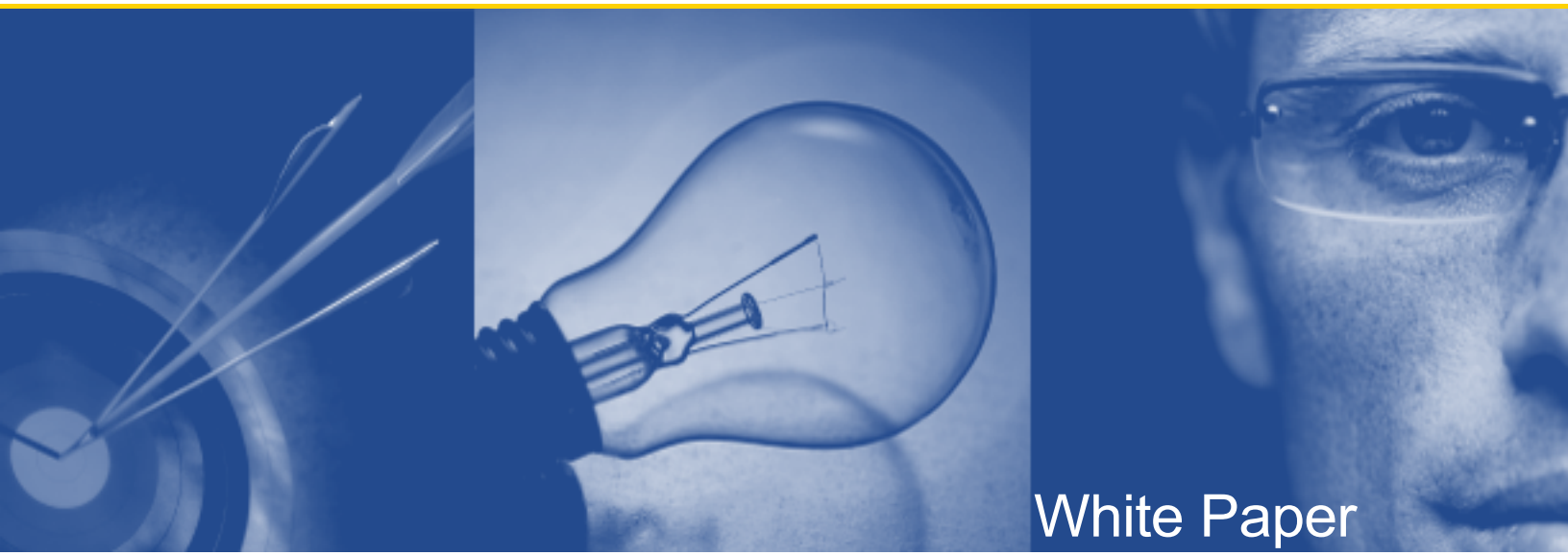
QUEST SOFTWARE®

Why and How You Should Find and Fix Index Fragmentation

Written by
Michelle Ufford,
SQL Developer and DBA for GoDaddy.com



WARRANTY

The information contained in this document is subject to change without notice. Quest Software makes no warranty of any kind with respect to this information. QUEST SOFTWARE SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTY OF THE MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Quest Software shall not be liable for any direct, indirect, incidental, consequential, or other damage alleged in connection with the furnishing or use of this information.

TRADEMARKS

All trademarks and registered trademarks used in this guide are property of their respective owners.

World Headquarters
5 Polaris Way
Aliso Viejo, CA 92656
www.quest.com
e-mail: info@quest.com

Please refer to our Web site (www.quest.com) for regional and international office information.

Updated—July 2009

CONTENTS

INTRODUCTION

Database maintenance is a complex activity involving many components and a variety of tasks. This paper covers just one challenging task: index maintenance.

It may help to think of a database in the context of an automobile and index maintenance as an oil change. Cars require regular maintenance in order to run their best; although failing to perform maintenance will not cause your car to break down immediately, it can lead to engine trouble and expensive repairs over a long period of time. The same is true with databases.

High-performance Formula One race cars require more care and more frequent maintenance than their family-sedan counterparts, and so do high-volume databases. The more data your database stores and the more transactions it performs, the more critical regular maintenance is and the more frequently you need to perform it.

Neglecting index maintenance can hurt a database in many ways. In particular, index fragmentation can lead to wasted storage space, inefficient I/O, and poor query performance. In this paper, we'll discuss what index fragmentation is, how it is caused, how to find it, and how to fix it.

UNDERSTANDING FRAGMENTATION AND HOW SQL SERVER STORES DATA

Although there are many kinds of fragmentation, including disk and file fragmentation, we'll be talking about index fragmentation only. Understanding index fragmentation requires a basic understanding of how SQL Server stores data. SQL Server organizes the pages allocated to an index in a B-tree structure. The top level of the B-tree is called the root node; the bottom level contains the leaf nodes. In a clustered index, the leaf node is where the actual data resides. The intermediate level encompasses the layers between the root and the leaf nodes. The number of intermediate levels depends on the size of the index; a small index might not even have an intermediate level.

Microsoft's Books Online provides an illustration to help us visualize the basic B-tree structure of a clustered index:

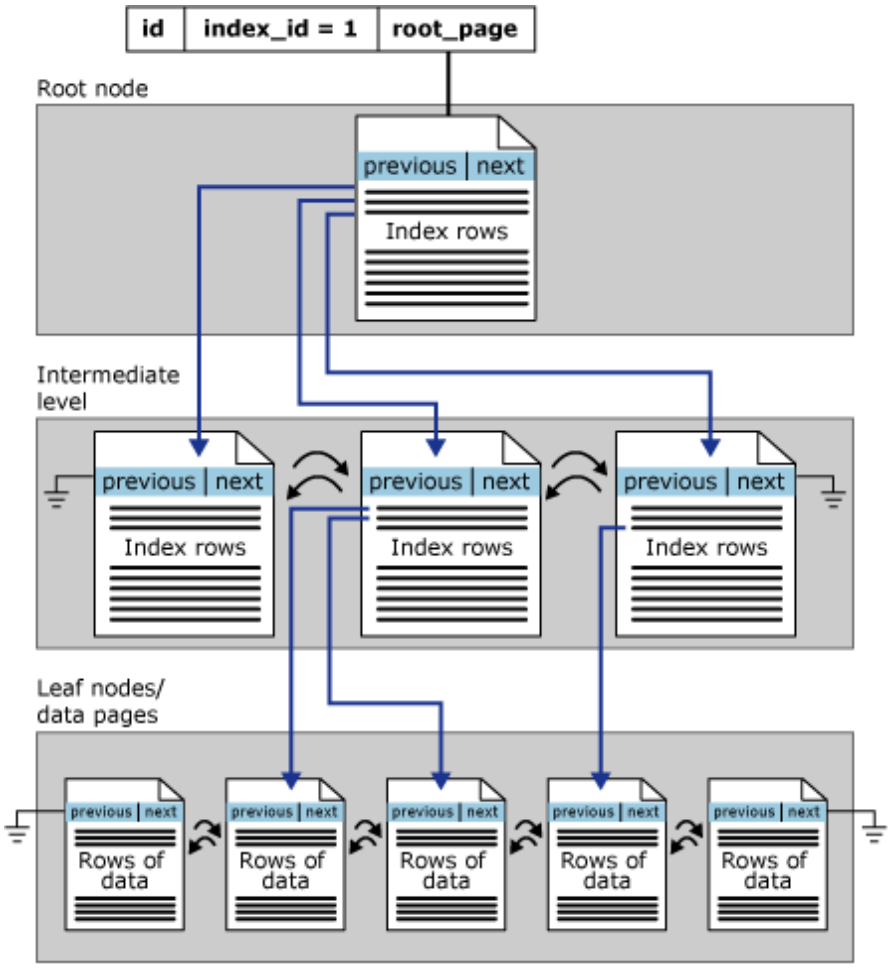


Figure 1. The basic B-tree structure of a clustered index

Pages are doubly linked, as illustrated by the "previous | next" section on each page. When searching for a single record, SQL Server navigates through the B-tree using the index key. For a range of values, SQL Server will first use the B-tree to navigate to the starting key value in the range, and then it'll utilize the pointers to scan subsequent pages.

Types of Index Fragmentation

There are two different types of index fragmentation: internal and external.

An Unfragmented Index

First consider what an unfragmented index looks like. A simplified unfragmented index is shown in Figure 2, which is adapted from an illustration in Books Online. Notice that, while the page numbers are not contiguous, they are ever-increasing.

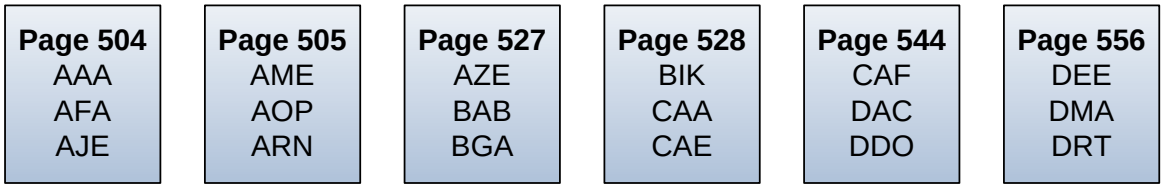


Figure 2. A simplified unfragmented index

External Fragmentation

External fragmentation occurs when the physical sequence of the index page chain does not match the logical ordering. That is, any page that does not follow a sequential order is fragmented. In Figure 3, the pages in green are fragmented because the physical and logical ordering of these pages is out of sync.

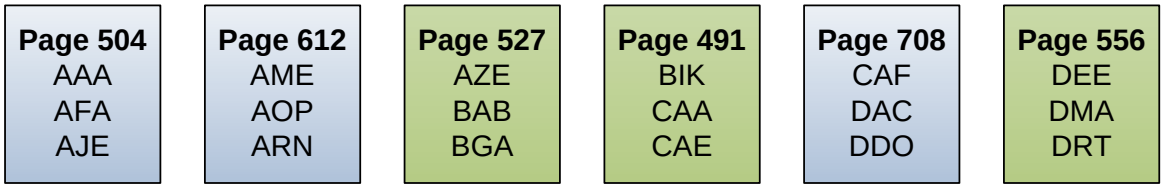


Figure 3. External fragmentation

External fragmentation can increase the amount of IO necessary to perform ordered index scans, such as querying on a range of values, because the storage engine reads index pages serially in key order.

Internal Fragmentation

Internal fragmentation occurs when the amount of free space on a page is not optimal; that is, the index is consuming more space than necessary. Figure 4 demonstrates internal fragmentation: the pages highlighted in green contain fewer rows than other pages.

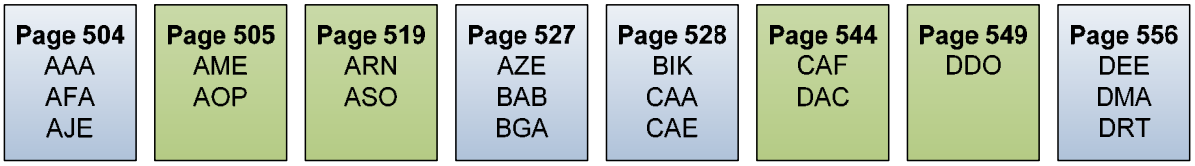


Figure 4. Internal fragmentation

Free space on a page is not necessarily a problem, especially if more records will be written to the page. However, a page that is only partially full and has little to no chance to be written to is internally fragmented and should be addressed. For example, consider an index that has so much internal fragmentation that it is only 50% full on average. That index will consume twice as much disk space as necessary, which will lead to other issues. More space will be required for backups. A single IO will return only half as many records, so SQL Server will need to perform twice as much IO to return the same amount of data, which means queries will take longer to execute. Therefore, correcting internal fragmentation is important to both database performance and cost control.

How Does Fragmentation Occur?

Fragmentation can occur any time a record is inserted, updated, or deleted. Therefore, the higher the volume of your database, the greater the potential for fragmentation and the more frequently you'll need to maintain it.

Causes of External Fragmentation

Consider the following very simplified illustration of how external fragmentation could occur:

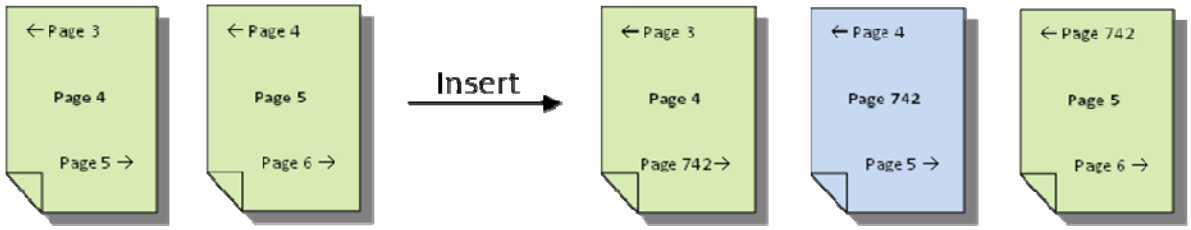


Figure 5. A page split can cause external index fragmentation

Page 4 needs to be written to (to maintain ordering of the index key), but it is full. Therefore, SQL Server has to perform a page split: a new page is allocated to the index and approximately half the data is moved from the full page to the new page; the forward and backward pointers of the surrounding pages are also updated. Because the logical ordering of data no longer matches the physical ordering, we now have external fragmentation in the index. The new page is not fragmented; the page it points to (in this case, page 5) is fragmented, because it is the one out of sequential order.

Causes of Internal Fragmentation

Page splits can also cause internal fragmentation, often at the same time that external fragmentation is caused. Consider the result of the page split just described:

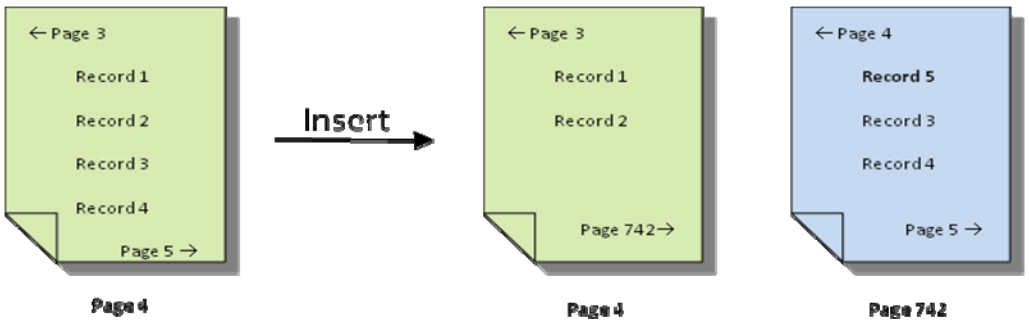


Figure 6. A page split can cause internal index fragmentation

During the page split, SQL Server moved half of the records from Page 4 to Page 742. The reasoning behind this behavior is that if one record needs to be written to Page 4, additional records may be on the way. This design is generally a good thing and improves overall write performance, especially in OLTP systems. However, if the page being updated and split contains old data that may never be written to again, you're left with an excess amount of free space on the page: internal fragmentation.

Minimizing Fragmentation

Since a single page split often results in both internal and external fragmentation, if you have one type of fragmentation, you probably have both. Certain designs, like clustering on a static and sequential value, can minimize the amount of fragmentation that can occur, and therefore it is recommended that most tables utilize a static, sequential column for a clustering key, such as an INT IDENTITY. But while attempting to minimize fragmentation is certainly a good idea, it is probably more important to accept that fragmentation will occur and to have a plan in place for managing it.

HOW TO FIND FRAGMENTATION

The sys.dm_db_index_physical_stats DMF

In SQL Server 2000, you may have used **DBCC ShowContig** to find fragmentation. SQL Server 2005 and 2008 provides the **sys.dm_db_index_physical_stats** dynamic management function (DMF), which gives more detailed information and is easier to manipulate than its DBCC predecessor. It's also worth mentioning that the fragmentation algorithm has been updated in SQL Server 2008, so that fragmentation levels may appear to be higher in newer versions. In reality, the newer algorithm is more refined and is merely better at recognizing fragmentation than its predecessor.

Parameters

The **sys.dm_db_index_physical_stats** DMF accepts five nullable parameters:

PARAMETER	DESCRIPTION
database_id	The ID of the database. A full list of databases and their IDs can be obtained from the sys.databases catalog view. Optionally, the DB_ID()function can be used (see http://msdn.microsoft.com/en-us/library/ms186274.aspx).
table_id	The ID of the table. A full list of tables and their IDs can be obtained from the sys.tables catalog view. Optionally, the OBJECT_ID()function may be used (see http://msdn.microsoft.com/en-us/library/ms190328.aspx).

index_id	The ID of the index. The following query[1] can be used to find the index_id:
	`SELECT name, index_id, type_desc`
	`FROM sys.indexes`
	`WHERE object_id = OBJECT_ID('Sales.SalesOrderDetail');`

partition_number	The number of the partition. If you are not using partition schemes—and most environments are not—you can set this to either NULL or 1.
mode	There are three modes for this function: LIMITED, SAMPLED, and DETAILED. • **LIMITED** - Scans only the parent-level pages of the index (the parent-level page is the index page directly above the leaf level). Because it scans the fewest pages, LIMITED mode executes faster than the other modes. However, LIMITED mode returns the least amount of information. In particular, it returns external fragmentation information but no internal fragmentation information. Use LIMITED mode if you're looking for general fragmentation information. • **SAMPLED** - Scans 1% of all data pages; if there are fewer than 10,000 pages in the index, DETAILED mode will be used instead. SAMPLED mode returns more information than LIMITED mode, and it returns internal as well as external fragmentation information.

1 All examples utilize the AdventureWorks sample database, which can be found at http://msftdbprodsamples.codeplex.com/.

PARAMETER	DESCRIPTION
	• **DETAILED** - Scans all pages and provides the most complete information. Unlike LIMITED and SAMPLED modes, which only provide estimates, DETAILED mode returns the most accurate statistics. It also returns more rows: LIMITED and SAMPLED modes return only a single row for each leaf node, but DETAILED mode returns a row for every node level of the index. DETAILED mode takes the longest to complete; you should avoid executing this mode on large tables during business hours. LIMITED is the recommended mode because it is the fastest to execute and the least intrusive. Although internal fragmentation statistics are not returned, external fragmentation statistics are typically a sufficient indicator as to whether an index needs to be defragmented.

Columns Returned

The sys.dm_db_index_physical_stats DMF returns quite a few columns. The following table explains the most important ones for index maintenance:

COLUMN	DESCRIPTION
index_level	Contains information about the node of the index. 0 is the leaf level, and the highest index_level is the root node; anything in between is an intermediate level.
avg_fragmentation_in_percent	External fragmentation. The higher the number, the higher the fragmentation level.
fragment_count	The number of physically consecutive leaf pages. The higher the fragment counter, the higher the level of fragmentation.
avg_fragment_size_in_pages	The number of pages in one fragment.
page_count	The number of pages in the index.
avg_page_space_used_in_percent	Internal fragmentation. The lower the number, the higher the level of fragmentation.

Calling the sys.dm_db_index_physical_stats DMF

Example 1: LIMITED Mode

The **sys.dm_db_index_physical_stats** DMF can be called in the following manner:

```
SELECT *
FROM sys.dm_db_index_physical_stats
(
       DB_ID(), -- database ID
       OBJECT_ID('Sales.SalesOrderHeader'), -- table ID
       NULL, -- index ID
       NULL, -- partition ID
       'LIMITED' -- mode
)
ORDER BY index_id, index_level;
```

Raw Results

The query above will return the fragmentation statistics for all indexes on the Sales.SalesOrderHeader table. For ease of viewing, only a part of the results is displayed below:

```
index_id    index_level avg_fragmentation_in_percent fragment_count
----------- ----------- --------------------------- -------------------
1           0           0.142857142857143           17
2           0           98.8593155893536            263
3           0           3                           5
5           0           98.0582524271845            103
6           0           30                          23
avg_fragment_size_in_pages page_count           avg_page_space_used_in_percent
-------------------------- -------------------- ------------------------------
41.1764705882353           700                  NULL
1                          263                  NULL
20                         100                  NULL
1                          103                  NULL
3.04347826086957           70                   NULL
```

Understanding the Fragmentation Data

External fragmentation is indicated by the avg_fragmentation_in_percent column; the higher the value, the worse the fragmentation. In this example, only three of the five indexes need to be defragged (indexes 2, 5, and 6); indexes 1 and 3 do not need to be defragged because they are less than five percent fragmented (we'll talk more about Microsoft's recommendations for defragging in the next section).

Do not be alarmed if your results do not match mine; values for the database_id and object_id columns can vary from installation to installation, and I have deliberately increased fragmentation levels for demonstration purposes.

Notice that the avg_page_space_used_in_percent column, which is the internal fragmentation, is NULL, because we ran the DMF in LIMITED mode; internal fragmentation is provided only for SAMPLED and DETAILED modes.

Example 2: DETAILED Mode

Now let's run the same query but in DETAILED mode:

```
SELECT *
FROM sys.dm_db_index_physical_stats
(
      DB_ID(), -- database ID
      OBJECT_ID('Sales.SalesOrderHeader'), -- table ID
      NULL, -- index ID
      NULL, -- partition ID
      'DETAILED' -- mode
      );
```

Raw Results

```
index_id    index_level avg_fragmentation_in_percent fragment_count
----------- ----------- --------------------------- --------------------
1           0           0.142857142857143           17
1           1           75                          4
1           2           0                           1
2           0           98.8593155893536            263
2           1           0                           1
3           0           3                           5
3           1           0                           1
5           0           98.0582524271845            103
5           1           0                           1
6           0           30                          23
6           1           0                           1
avg_fragment_size_in_pages page_count           avg_page_space_used_in_percent
-------------------------- -------------------- ------------------------------
41.1764705882353           700                  99.0983197430195
1                          4                    28.0825302693353
1                          1                    0.617741536940944
1                          263                  33.9719915987151
1                          1                    81.2083024462565
20                         100                  97.1648134420558
1                          1                    33.3333333333333
1                          103                  41.491722263405
1                          1                    21.6085989621942
3.04347826086957           70                   77.7242401779096
1                          1                    17.2720533728688
```

Understanding the Fragmentation Data

As explained earlier, DETAILED mode returns one row for every level of the index's B-tree. Most people will need to look only at the leaf node, which is indicated by an `index_level` value of 0. In addition, DETAILED mode returns information about internal as well as external fragmentation levels. As opposed to external fragmentation values, the higher the internal fragmentation value in `avg_page_space_used_in_percent` the better.

Guidelines for Choosing a Mode

You typically will not need to use DETAILED mode; a high value for external fragmentation (`avg_fragmentation_in_percent`) often means a low value for internal fragmentation (`avg_page_space_used_in_percent`), as illustrated in the leaf-level results above. If you do decide to run the DMF in DETAILED mode on a production server, you should take great care; I've seen an unchecked query with all NULL parameters in DETAILED mode take down a mission-critical server.

HOW TO DEFRAG YOUR INDEXES

In SQL Server 2005 and 2008, the ALTER INDEX command is used for index defragmentation. While this command can also be used to perform many other tasks, we're going to look only at the two arguments available for defragmentation: REORGANIZE and REBUILD.

Reorganize

The REORGANIZE option does just what its name suggests: it reorganizes data within the pages to reduce fragmentation. Specifically, the process first tries to consolidate data into fewer pages to reduce internal fragmentation. It then attempts to move data so that the physical ordering matches the logical ordering, which reduces external fragmentation.

Index reorganization is an online operation. This means that the table and index is available for querying and updating during the reorganization.

Rebuild

The REBUILD option also does as its name implies: it completely rebuilds the index by creating a new copy of the index and then dropping the old version. This process is more thorough than reorganization but also more expensive.

Rebuilds can be performed online only in the Enterprise edition of SQL Server. In Standard edition, an ALTER INDEX REBUILD will cause the table to become inaccessible for querying and data updates.

Rebuilding an index may not be possible in some situations, such as if your table contains LOBs (large objects, such as XML or VARCHAR (MAX) data types) or if your table is partitioned and you want to only defrag a specific partition. In these cases, you'll need to use REORGANIZE instead.

Best Practices

Some best practices for index defragmentation include:

- For indexes between 5% and 30% fragmented, use ALTER INDEX REORGANIZE.
- For indexes that are more than 30% fragmented, use ALTER INDEX REBUILD WITH (ONLINE = ON). (This option is available in only Enterprise editions of SQL Server.)
- Indexes less than 5% fragmented should not be defragged because the cost of the operation is not worth the benefit.

For ALTER INDEX commands issued during working hours, consider using the MAXDOP restriction. This option allows you to restrict the number of processors allocated to the ALTER INDEX operation. The MAXDOP restriction is also a feature of Enterprise edition and is unavailable in Standard.

Another option to consider is SORT_IN_TEMPDB. When set to ON, this option will perform all sorting operations in the tempdb database instead of the database the index resides in. This can sometimes improve performance, especially for large index creations and rebuilds.

Small indexes—especially those with less than eight pages—may show high levels of fragmentation even after an ALTER INDEX REBUILD or a REORGANIZE. This is actually very common and is not cause for worry.

Examples

Example 1: Reorganizing an Index

```
ALTER INDEX PK_Employee_EmployeeID
    ON HumanResources.Employee
        REORGANIZE;
```

Example 2: Rebuilding an Index

```
ALTER INDEX PK_SalesOrderHeader_SalesOrderID
    ON Sales.SalesOrderHeader
        REBUILD
            WITH (ONLINE = ON);
```

Example 3: Rebuilding a Large Index to Minimize Server Impact

```
ALTER INDEX PK_SalesOrderHeader_SalesOrderID
    ON Sales.SalesOrderHeader
        REBUILD
            WITH (  ONLINE = ON,
                    MAXDOP = 1,
                    SORT_IN_TEMPDB = ON );
```

For More Information

Index defragmentation is a complex operation, and this paper provides only a rather simple overview. If you're interested in obtaining a better understanding of the process, I highly recommend you pick up a copy of Microsoft SQL Server 2008 Internals.

CONCLUSION

Index maintenance is an important part of database maintenance. Neglecting indexes will eventually lead to high levels of fragmentation, both internal and external, that can in turn cause a myriad of problems, including wasted storage space in your database, SAN, and backups; wasted IO to retrieve multiple rows of data; and slower queries and backups. Fragmentation can have an especially negative impact on business reporting systems, where range scans are more frequently performed. Regular index maintenance in SQL Server can improve the health and performance of your databases, servers, and file storage systems.

ABOUT THE AUTHOR

Michelle Ufford is a SQL developer and DBA for GoDaddy.com, where she works with high-volume, mission-critical databases. She has more than a decade of experience in a variety of technical roles and has worked with SQL Server for the last five years. She enjoys performance tuning and maintains an active SQL Server blog. In addition, Michelle is a regular contributor to SQLServerPedia.com.

Her online presence includes:

- SQLServerPedia Profile: Sqlfool
- Blog: http://sqlfool.com/
- Twitter: http://twitter.com/sqlfool/

ABOUT QUEST SOFTWARE, INC.

Quest Software, Inc., a leading enterprise systems management vendor, delivers innovative products that help organizations get more performance and productivity from their applications, databases, Windows infrastructure and virtual environments. Through a deep expertise in IT operations and a continued focus on what works best, Quest helps more than 100,000 customers worldwide meet higher expectations for enterprise IT. Quest Software helps organizations deliver, manage and control complex database environments through award-winning products for Oracle, SQL Server, IBM DB2, Sybase and MySQL. Quest Software can be found in offices around the globe and at www.quest.com.

Contacting Quest Software

Phone:	949.754.8000 (United States and Canada)
Email:	info@quest.com
Mail:	Quest Software, Inc. World Headquarters 5 Polaris Way Aliso Viejo, CA 92656 USA
Web site:	www.quest.com

Please refer to our Web site for regional and international office information.

Contacting Quest Support

Quest Support is available to customers who have a trial version of a Quest product or who have purchased a commercial version and have a valid maintenance contract. Quest Support provides around the clock coverage with SupportLink, our web self-service. Visit SupportLink at http://support.quest.com

From SupportLink, you can do the following:

- Quickly find thousands of solutions (Knowledgebase articles/documents).
- Download patches and upgrades.
- Seek help from a Support engineer.
- Log and update your case, and check its status.

View the ***Global Support Guide*** for a detailed explanation of support programs, online services, contact information, and policy and procedures. The guide is available at: http://support.quest.com/pdfs/Global Support Guide.pdf