What is fragmentation?

As data is modified in a database, the database and its indexes become fragmented. As indexes become fragmented, ordered data retrieval becomes less efficient and reduces database performance.

Understanding the different types of fragmentation

There are several types of fragmentation that can occur and impact SQL Server performance and space usage. Note that logical order and page density issues exist on tables and indexes within SQL Server. These issues cannot be resolved by operating system level defragmentation tools because the fragmentation exists within the files, rather than at the file level itself.

File fragmentation at the operating system level

When deletes and inserts are performed over time, pages become fragmented as the physical sequence of data pages no longer matches their logical order. This fragmentation happens at the file allocation level and can be addressed with system tools. On larger systems, such as a storage area network (SAN), the disk subsystem automatically maintains low fragmentation levels. If you have a small to medium size system and you do not have a SAN, you should run a system defragmentation tool before addressing logical order and page density fragmentation within SQL Server.

Logical order fragmentation

This issue, also known as external fragmentation within SQL Server, is similar to file fragmentation at the operating system level. When data is deleted, inserted, and modified over time, an index can cause pages to be out of order, where the next logical page is not the same as the next physical page.

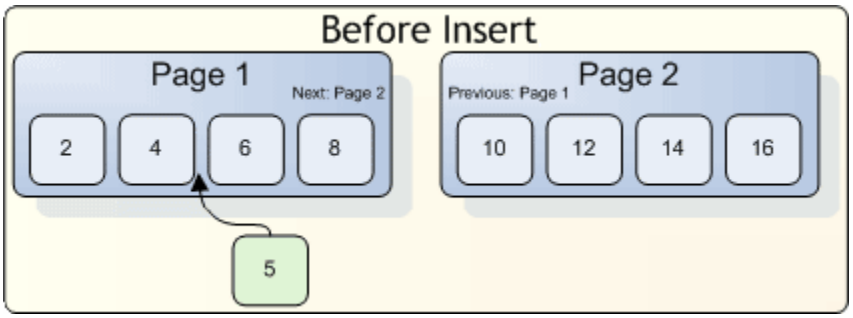
Page density fragmentation

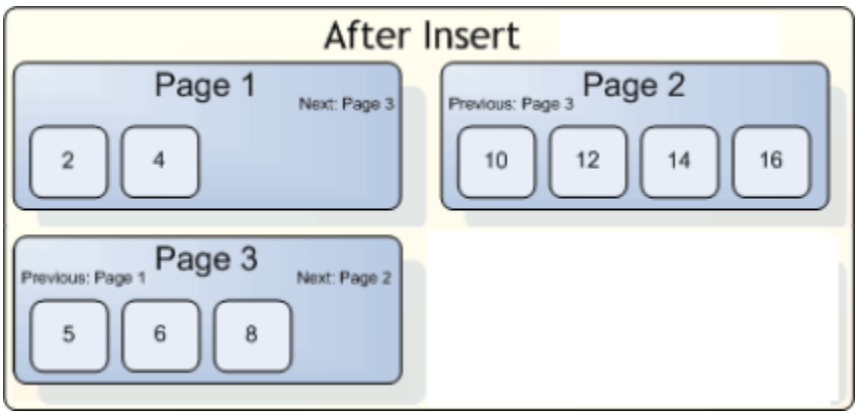
This issue, also known as internal fragmentation, occurs as pages split to make room for information added to a page, there may be excessive free space left on the pages. This extra space can cause SQL Server to read more pages than necessary to perform certain tasks.

SQL Defrag Manager defragments the leaf level of an index so the physical order of the pages matches the left-to-right logical order of the leaf pages. The leaf pages of a clustered index contain the table data. This process improves index scanning performance and all data retrieval activities.

Fragmentation examples

Imagine there are two data pages for a table with a clustered index. The data is ordered and the pages are full as shown in the following figure. A new row with a primary key of "5" needs to be inserted, and since it is a clustered index, the new row is inserted in order. Because the target page is full enough that the new row does not fit, SQL Server splits the page roughly in half and inserts the new data on the new page, as shown in the following figure. Now, the logical order of the index does not match the physical order, and the index has become fragmented.





How does SQL Defrag Manager defragment indexes?

Depending on the policy settings you select, SQL Defrag Manager defragments tables and indexes in one of the following ways:

Rebuild

To rebuild the indexes on tables, the rebuild defragmentation option uses the `ALTER INDEX REBUILD` command for SQL Server 2005 and newer versions. For SQL Server 2000 it uses `DBCC DBREINDEX`. The rebuild operation creates new, contiguous pages. SQL Server 2005/2008 allows the option to Rebuild Online, which allows access to the tables before the operation is finished. However, choosing to rebuild online requires more resources (disk space, CPU, memory), and may slow performance.

Reorganize

To reorder the leaf pages of the index in-place, the reorganize defragmentation option uses the `ALTER INDEX REORGANIZE` command for SQL Server 2005 and newer versions. For SQL Server 2000 it uses `DBCC INDEXDEFRAG`. This process is similar to a bubble sort. Although the pages are physically reordered, they may not be contiguous within the data file. This issue can cause interleaved indexes, which need to be rebuilt to store them in contiguous pages.

> ⓘ Online index operations are available only in SQL Server Data center, Enterprise, Developer, and Evaluation editions.

Defragmenting an index example

Consider a simplified example of pages after many inserts, updates, and deletes, as shown in the following figure. The page numbering represents the logical sequence of the pages. However, the physical sequence, as shown in the figure from left to right, does not match the logical sequence.

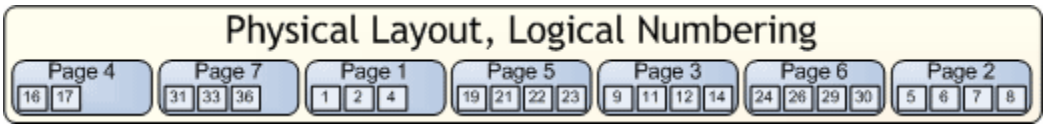
The following figure illustrates multiple passes during the Reorganize defragmentation process, which causes the physical pages to be reordered by having the first logical page swapped with the first physical page, and then the second logical page swapped with the second physical page, and so on.

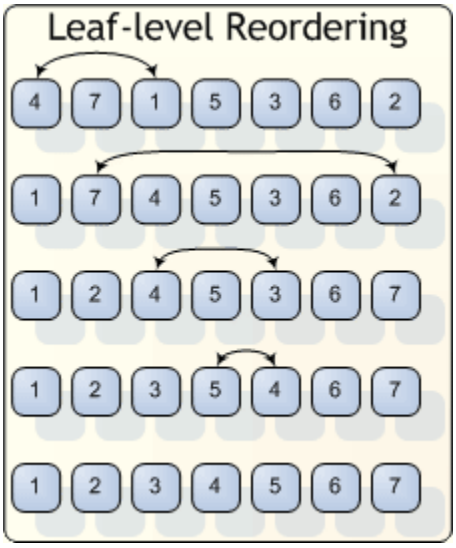
On the first pass, SQL Server finds the first physical page (4) and the first logical page (1). SQL Server then swaps these pages in a discrete transaction.

On the second pass, SQL Server swaps the next physical page (7) with the next logical page (2).

On the third pass, SQL Server swaps the next physical page (4) with the next logical page (3).

On the fourth pass, SQL Server swaps the next physical page (5) with the next logical page (4). Sorting is now complete, as all the remaining physical pages match their logical positions.





How SQL Defrag Manager compacts data

In addition to reordering the leaf pages of the index, SQL Defrag Manager compacts the data in the pages using the original fill factor value specified for the table and then removes any empty pages. Consider the following conditions related to this compaction phase:

- Compaction is completely skipped if the Inhibit Page Locks property is set for the index.

- There are various algorithms built into the compaction phase to stop unnecessary work. For example, if the first page in the index is empty and all the other pages are full, SQL Server does not repeatedly move all the data forward one page.
- SQL Server compacts pages back to the fill factor value defined for the index. Make sure this value is not set too high. For more information, see the SQL Server documentation.
- If a lock cannot be obtained on a page during the compaction phase of DBCC INDEXDEFRAG, SQL Server skips that page.

About interleaved indexes

Interleaving occurs when an index extent, which is a group of eight index pages, is not physically contiguous because an extent for another index is intermingled with it. This condition can happen even when there is no logical fragmentation in the index. Although the pages may be physically and logically ordered, they are not necessarily contiguous. Switching between extents can impact performance as data access is inefficient. To resolve this issue, use SQL Defrag Manager to rebuild the indexes to store them in contiguous pages and reduce the need to switch between extents.