IDERA

What is Index Fragmentation

Learn About Index Fragmentation And Defragmentation

An Idera Whitepaper

Introduction

When changing data in a database, the database and its indexes become fragmented. As indexes become fragmented, ordered data retrieval becomes less efficient. Inefficient data retrieval reduces the performance of the database.

Understanding Different Types of Fragmentation

There are several types of fragmentation that can occur. These types of fragmentation impact the performance of the database and the usage of disk space. Logical order and page density issues exist on tables and indexes within databases. Tools for defragmentation at the level of the operating system cannot resolve these issues. The reason is that the fragmentation exists within the files, rather than at the file level itself.

File fragmentation at the operating system level

When performing deletes and inserts 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. System tools can address this fragmentation. On larger systems, such as a storage area network (SAN), the disk subsystem automatically maintains low fragmentation levels. For small to medium size systems without a SAN, run a tool for system defragmentation before addressing logical order and page density fragmentation within databases.

Logical order fragmentation

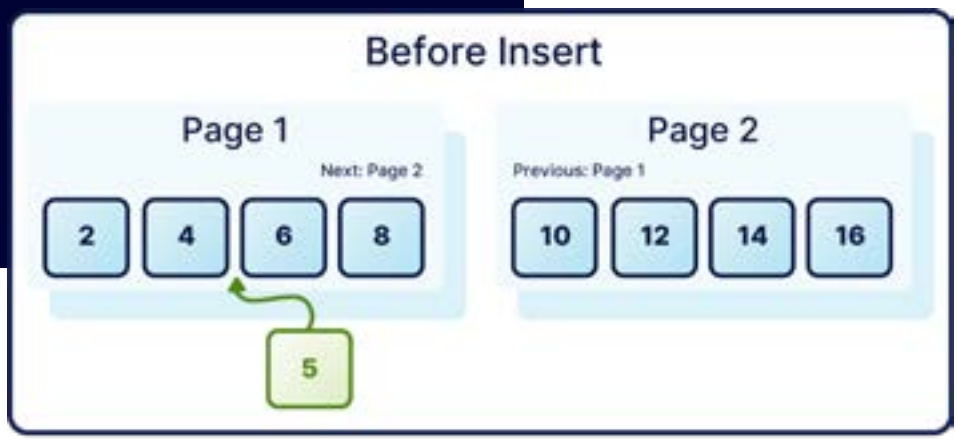
Logical order fragmentation, also known as external fragmentation within databases, is like file fragmentation at the operating system level. When deleting, inserting, and changing data over time, an index can cause pages to be out of order. In that case, the next logical page is not the same as the next physical page.

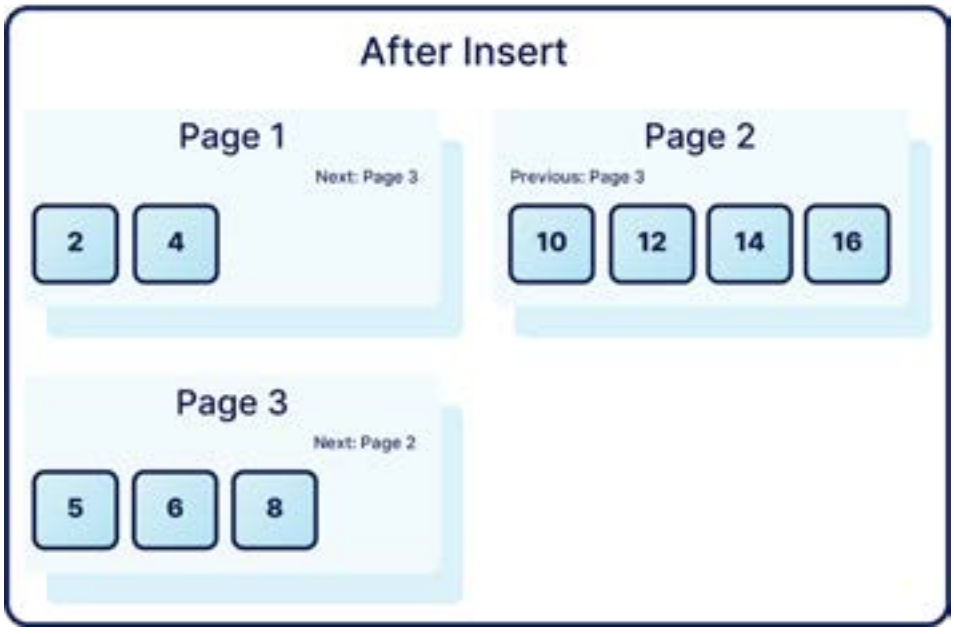
Page density fragmentation

Page density fragmentation, also known as internal fragmentation, occurs as pages split to make room for information added to a page. In that case, there may be excessive free space left on the pages. This extra space can cause database instances to read more pages than necessary to perform certain tasks. It is necessary to defragment the leaf level of an index so that 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 the performance of index scanning and of all data retrieval activities.

Fragmentation Examples

Consider the case when there are two data pages for a table with a clustered index. The data is ordered and the pages are full. The following figure shows that. Insert a new row with a primary key of “5”. Since it is a clustered index, insert the new row in order. Because the target page is full enough that the new row does not fit, the database instance splits the page roughly in half and inserts the new data on the new page. The following figure shows that. Now, the logical order of the index does not match the physical order. The index has become fragmented.





How to Defragment Indexes?

It is possible to defragment tables and indexes by rebuilding and reorganizing.

Rebuild

The defragmentation type of rebuild uses the command “DBCC DBREINDEX” to rebuild the indexes on the tables. The rebuild operation creates new, contiguous pages. It may be possible to rebuild online. That allows access to the tables before the operation is finished. However, choosing to rebuild online requires more resources (that is, disk space, CPU, and memory), and may slow performance.

Reorganize

The defragmentation type of reorganize uses the command “DBCC INDEXDEFRAG” to reorder the leaf pages of the index in-place. This process is like 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.

Example: Defragment an Index

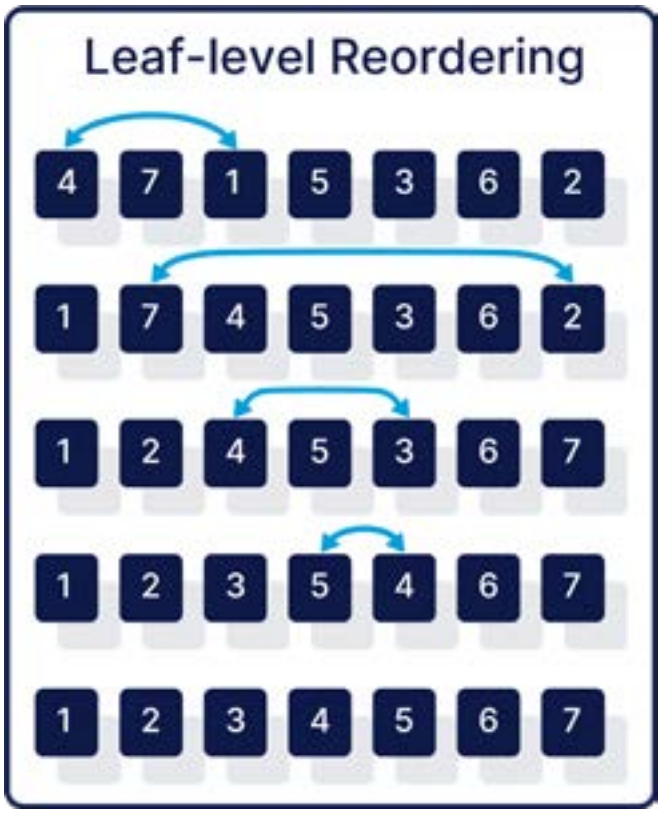
Consider a simplified example of pages after many inserts, updates, and deletes. The following figure shows that. The page numbering represents the logical sequence of the pages. However, the physical sequence, as shown in the figure from left to right, does not match the logical sequence. The following figure illustrates multiple passes during the reorganize defragmentation process. This process causes reordering of the physical pages 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.

1. On the first pass, the database instance finds the first physical page (“4”) and the first logical page (“1”). The database instance then swaps these pages in a discrete transaction.

2. On the second pass, the database instance swaps the next physical page (“7”) with the next logical page (“2”).

3. On the third pass, the database instance swaps the next physical page (“4”) with the next logical page (“3”).

4. On the fourth pass, the database instance 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 to Compact Data

Besides reordering the leaf pages of the index, it is possible to compact the data in the pages using the original fill factor value specified for the table and then remove any empty pages. Consider the following conditions related to this compaction phase:

- Completely skip compaction when inhibiting page locks 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, then the database instance does not repeatedly move all the data forward one page.
- The database instance compacts pages back to the fill factor value defined for the index. Do not set this value too high.
- If a lock cannot be obtained on a page during the compaction phase of the command "DBCC INDEXDEFRAG", then the database instance 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 it intermingles with an extent for another index. 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 affect performance as data access is inefficient. To resolve this issue, rebuild the indexes to store them in contiguous pages and reduce the need to switch between extents.

SQL Defrag Manager

SQL Defrag Manager is a powerful solution capable of detecting SQL Server database fragmentation hot spots and defragmenting them automatically. Its centralized management console enables users to control all defragmentation activities across hundreds of servers. It also makes it easy for users to run reports, to automate alerting, and set defragmentation policies. SQL Defrag Manager performs the defragmentation process through scheduling of jobs to ensure minimum impact to production servers and avoids any potential problems with system resource pre-checks.

Start for FREE

IDERA

www.idera.com