INDEX FRAGMENT

When indexes are first built, little or no fragmentation should exist. Over time, as data is inserted, updated, and deleted, fragmentation levels on the underlying indexes may begin to rise. So let's see how it happens.

When a page of data fills to 100 percent and more data must be added to it, a page split occurs. To make room for the new incoming data, SQL Server moves half of the data from the full page to a new page. The new page that is created is created after all the pages in the database. Therefore, instead of going right from one page to the next when looking for data, SQL Server has to go from one page to another page somewhere else in the database looking for the next page it needs. This is called index fragmentation.

There are basically two types of fragmentation:

* **External fragmentation** - External, a.k.a logical, fragmentation occurs when an index leaf page is not in logical order, in other words it occurs when the logical ordering of the index does not match the physical ordering of the index. This causes SQL Server to perform extra work to return ordered results. For the most part, external fragmentation isn't too big of a deal for specific searches that return very few records or queries that return result sets that do not need to be ordered.
* **Internal fragmentation** - Internal fragmentation occurs when there is too much free space in the index pages. Typically, some free space is desirable, especially when the index is created or rebuilt. You can specify the Fill Factor setting when the index is created or rebuilt to indicate a percentage of how full the index pages are when created. If the index pages are too fragmented, it will cause queries to take longer (because of the extra reads required to find the dataset) and cause your indexes to grow larger than necessary. If no space is available in the index data pages, data changes (primarily inserts) will cause page splits as discussed above, which also require additional system resources to perform.

The sys.dm\_db\_index\_physical\_stats function returns tabular data regarding one particular table or index.

|  |  |
| --- | --- |
| **Input Parameter** | **Description** |
| database\_id | The default is 0 (NULL, 0, and DEFAULT are equivalent values in this context) which specify to return information for all databases in the instance of SQL Server else specify the databaseID from sys.databases if you want information about a specific database. If you specify NULL for database\_id, you must also specify NULL for object\_id, index\_id, and partition\_number. |
| object\_id | The default is 0 (NULL, 0, and DEFAULT are equivalent values in this context) which specify to return information for all tables and views in the specified database or else you can specify object\_id for a particular object. If you specify NULL for object\_id, you must also specify NULL for index\_id and partition\_number. |
| index\_id | The default is -1 (NULL, -1, and DEFAULT are equivalent values in this context) which specify to return information for all indexes for a base table or view. If you specify NULL for index\_id, you must also specify NULL for partition\_number. |
| partition\_number | The default is 0 (NULL, 0, and DEFAULT are equivalent values in this context) which specify to return information for all partitions of the owning object. partition\_number is 1-based. A nonpartitioned index or heap has partition\_number set to 1. |
| mode | mode specifies the scan level that is used to obtain statistics. Valid inputs are DEFAULT, NULL, LIMITED, SAMPLED, or DETAILED. The default (NULL) is LIMITED.  LIMITED - It is the fastest mode and scans the smallest number of pages. For an index, only the parent-level pages of the B-tree (that is, the pages above the leaf level) are scanned. In SQL Server 2008, only the associated PFS and IAM pages of a heap are examined; the data pages of the heap are not scanned. In SQL Server 2005, all pages of a heap are scanned in LIMITED mode.   SAMPLED - It returns statistics based on a 1 percent sample of all the pages in the index or heap. If the index or heap has fewer than 10,000 pages, DETAILED mode is used instead of SAMPLED.   DETAILED - It scans all pages and returns all statistics. |

**Example**

Let's see an example. The first script provided below gives the fragmentation level of a given database including all tables and views in the database and all indexes on these objects. The second script gives the fragmentation level of a particular object in the given database. The details about the columns and its meaning returned by the **sys.dm\_db\_index\_physical\_stats** are given in the below table.

--To Find out fragmentation level of a given database

--This query will give DETAILED information

--CAUTION : It may take very long time, depending on the number of tables in the DB

USE AdventureWorks

GO

SELECT object\_name(IPS.object\_id) AS [TableName],

SI.name AS [IndexName],

IPS.Index\_type\_desc,

IPS.avg\_fragmentation\_in\_percent,

IPS.avg\_fragment\_size\_in\_pages,

IPS.avg\_page\_space\_used\_in\_percent,

IPS.record\_count,

IPS.ghost\_record\_count,

IPS.fragment\_count,

IPS.avg\_fragment\_size\_in\_pages

FROM sys.dm\_db\_index\_physical\_stats(db\_id(N'AdventureWorks'), NULL, NULL, NULL , 'DETAILED') IPS

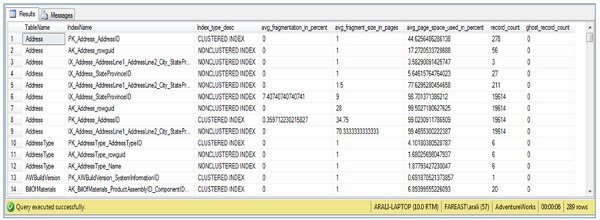
JOIN sys.tables ST WITH (nolock) ON IPS.object\_id = ST.object\_id

JOIN sys.indexes SI WITH (nolock) ON IPS.object\_id = SI.object\_id AND IPS.index\_id = SI.index\_id

WHERE ST.is\_ms\_shipped = 0

ORDER BY 1,5

GO



--To Find out fragmentation level of a given database and table

--This query will give DETAILED information

DECLARE @db\_id SMALLINT;

DECLARE @object\_id INT;

SET @db\_id = DB\_ID(N'AdventureWorks');

SET @object\_id = OBJECT\_ID(N'Production.BillOfMaterials');

IF @object\_id IS NULL

BEGIN

PRINT N'Invalid object';

END

ELSE

BEGIN

SELECT IPS.Index\_type\_desc,

IPS.avg\_fragmentation\_in\_percent,

IPS.avg\_fragment\_size\_in\_pages,

IPS.avg\_page\_space\_used\_in\_percent,

IPS.record\_count,

IPS.ghost\_record\_count,

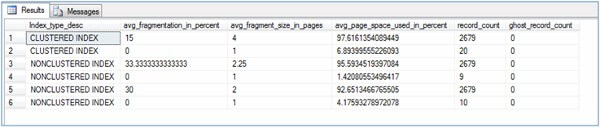
IPS.fragment\_count,

IPS.avg\_fragment\_size\_in\_pages

FROM sys.dm\_db\_index\_physical\_stats(@db\_id, @object\_id, NULL, NULL , 'DETAILED') AS IPS;

END

GO



|  |  |
| --- | --- |
| **Returned Column** | **Description** |
| avg\_fragmentation\_in\_percent | It indicates the amount of external fragmentation you have for the given objects.   The lower the number the better - as this number approaches 100% the more pages you have in the given index that are not properly ordered.  For heaps, this value is actually the percentage of extent fragmentation and not external fragmentation. |
| avg\_page\_space\_used\_in\_percent | It indicates how dense the pages in your index are, i.e. on average how full each page in the index is (internal fragmentation).   The higher the number the better speaking in terms of fragmentation and read-performance. To achieve optimal disk space use, this value should be close to 100% for an index that will not have many random inserts. However, an index that has many random inserts and has very full pages will have an increased number of page splits. This causes more fragmentation. Therefore, in order to reduce page splits, the value should be less than 100 percent. |
| fragment\_count | A fragment is made up of physically consecutive leaf pages in the same file for an allocation unit. An index has at least one fragment. The maximum fragments an index can have are equal to the number of pages in the leaf level of the index. So the less fragments the more data is stored consecutively. |
| avg\_fragment\_size\_in\_pages | Larger fragments mean that less disk I/O is required to read the same number of pages. Therefore, the larger the avg\_fragment\_size\_in\_pages value, the better the range scan performance. |