Clustered, Non Clustered indexes and Heaps - General guidelines

Last updated by | Vitor Tomaz | Feb 24, 2023 at 3:32 AM PST

Contents

Issue

Sometimes when engaging with customers we might come across with questions about indexes, especially when we are analyzing execution plans or troubleshoot performance issues in general.

It`s a good idea that we have some basics covered, actions and some misconceptions.

Investigation/Analysis

First covering some of the basics:

Clustered index According with books online: "Clustered indexes sort and store the data rows in the table or view based on their key values. These are the columns included in the index definition". Since the clustered index contains the data rows, you can only have per table. Clustered indexes have an overhead on writes when compared with Heap tables (since the data needs to be sorted)

Nonclustered indexes It is stored separately from the data rows, so you can have more than one. The nonclustered key value has a row locator that points to a row or a clustered key value depending if is a Heap or a clustered table. Contrary to Clustered indexes, it can contain non-key values (INCLUDED columns).

Heaps It`s a table without a clustered index. Data will not be ordered, so INSERTs are faster. When reading from a Heap table more random IO will be observed since pages are not located close to each other.

How index fragmentation happens: Lets go through an example. Imagine that you have 2 pages, page 1 and 2. On page 1 you have the records 1,2, 4 and 5 and on page 2 you have 6, 7, 8 and 9. Both pages are already filled up.

- a transaction INSERT the record 3, that will go to the first page.
- since there is no space a split will happen - page 3 will be allocated and record 3, 4 and 5 will be on this new page.
- Now the next page for page 1 will be Page 3 (that is not contiguous). Page 2 will be the next page of Page 3.
- Page 1 only contains records 1 and 2 (50% of the page).

Now the next page for page 1 will be Page 3 (that is not contiguous). Page 2 will be the next page of Page 3

This also introduces the concept of page density: with the event of page splits, it will mean that some pages will not be full. In the example above, Page 1 density went from 100% to 50%.

On events of low density the impact can be seen in:

- More space used (more pages are being used)
- More I/O and CPU
- Execution plan compile (I/O cost estimates)

How to check index fragmentation:

The query below will check fragmentation on all indexes, for all databases on the Managed Instance.

Note: might take sometime to execute. You might want to go with a more cautious approach, checking for a specific database (the database that you want to troubeshoot)

```
Declare @indexes table (DBName sysname, index_name sysname, index_id int, index_type nvarchar(60), fragmentati

insert into @indexes
exec SP_MSforeachdb @command1 =' use [?];
SELECT OBJECT_NAME(ips.OBJECT_ID)
 ,i.NAME
 ,ips.index_id
 ,index_type_desc
 ,avg_fragmentation_in_percent
 ,avg_page_space_used_in_percent
 ,page_count
FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, ''SAMPLED'') ips
INNER JOIN sys.indexes i ON (ips.object_id = i.object_id)
 AND (ips.index_id = i.index_id)
 where avg_fragmentation_in_percent > 0
ORDER BY avg_fragmentation_in_percent DESC'

select * from @indexes
```

For just one database

```
SELECT OBJECT_NAME(ips.OBJECT_ID)
 ,i.NAME
 ,ips.index_id
 ,index_type_desc
 ,avg_fragmentation_in_percent
 ,avg_page_space_used_in_percent
 ,page_count
FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'SAMPLED') ips
INNER JOIN sys.indexes i ON (ips.object_id = i.object_id)
 AND (ips.index_id = i.index_id)
 where avg_fragmentation_in_percent > 0
ORDER BY avg_fragmentation_in_percent DESC
```

Just like Clustered and Non clustered indexes, Heaps can also be fragmented and suffer from low page density. But on heaps take special attention to a new concept - forwarded records https://www.sentryone.com/blog/tracking-and-alerting-on-forwarded-records-with-sql-sentry

In short, when a page split occurs on a Heap, when a record is moved to another page, there will be a pointer on the old page referencing the new page. This will have an impact on performance since more requests are done to read the data.

```
Select Database_id,
Index_id,
Object_name([object_id]) as TableName,
Case when SI.name is null then 'HEAP' else SI.name End as IndexName,
Index_type_desc,
Avg_fragmentation_in_percent,
       forwarded_record_count
From sys.dm_db_index_physical_stats(db_id(),object_id('<table_name>'),null,null,'detailed') AS SDDIPS
Inner join sys.sysindexes AS SI on SDDIPS.[object_id] = SI.id
AND SDDIPS.index_id = SI.indid
Where index_level = 0
```

Mitigation

Solutions:

- Rebuild and / or reorganize indexes
- Set the fillfactor to a value other than 0 or 100 - be careful with this. The fillfactor must be set carefully and for indexes where it makes sense. Fillfactor impacts page density and can lead to higher IO. https://learn.microsoft.com/en-us/sql/relational-databases/indexes/specify-fill-factor-for-an-index?view=sql-server-ver16
- Table design and choosing data types correctly can help, but this will be out of scope of support.
- on the case of HEAP TABLES, you can run ALTER TABLE [table_name] REBUILD to rebuild the HEAP.

Some myths around indexes:

- "*Index fragmentation doesnt matter if you have SSDs*" - this is not true. SSD`s can in fact speed up reads, but they will not prevent your buffer cache to have more pages than you are supposed to have - page density.
- "*On you maintenance task, just Rebuild all your indexes*" - this is just a waste of time and resources. When rebuilding use fragmentation thresholds.
- "*Change the fillfactor of all indexes to XX*" - This is not the correct approach. Focus on indexes where you can see frequent page splits happening. Remember that setting a fillfactor has an impact on the index page density
- "*Create an index on each column*" - this is not the correct indexing strategy. The index design should reflect the query predicates and workload. Indexes will always introduce overhead on write operations.

Some points to have in mind when engaging with customers:

- Rebuilding or creating indexes offline will create locks. Make sure that the customer knows it
- Indexes introduce overhead on writes. The customer should test the index and validate if it creates problems on the workload.

Root Cause Classification

How good have you found this content?