SQL ServerCentral.com
SQL Prompt Code analysis as you type SQLQuery1.sql Try it now redgate a community service from redgate Try SQL Clone

Daily SQL Articles by email:
your@email.com
Sign up

Write For Us
Find out more

SQL Prompt
Code analysis as you type
SQLQuery1.sql
Try it now
redgate

Improve your SQL Server knowledge daily with more articles by email.
your@email.com
Sign up

Thank this author by sharing: G+ 0
Rate this | Join the discussion | Add to briefcase

Pages and Extents: Stairway to SQL Server Indexes Level 4

By David Durant, 2011/02/01

The Series

This article is part of the Stairway Series: Stairway to SQL Server Indexes

Indexes are fundamental to database design, and tell the developer using the database a great deal about the intentions of the designer. Unfortunately indexes are too often added as an afterthought when performance issues appear. Here at last is a simple series of articles that should bring any database professional rapidly 'up to speed' with them

In the preceding levels, we performed identical queries against indexed and un-indexed tables, comparing the work required for each. Our primary metric was the “logical read”. We were always comparing the reads required to query the indexed table versus the un-indexed table. Now it is time to explain why logical reads are an excellent metric and also explain what is actually being read.

When you submit a request for information to SQL Server, it knows that it can always satisfy that request by scanning the entire table(s). SQL Server understands that an index benefits the query only if using that index is less work than scanning the entire table. And if you were to ask SQL Server “What is work?”, its primary answer would be “Disk I/O.”. The IO required by a query is a good indicator that query’s cost; mainly because IO consumes two critical resources, time and memory.

The I/O required to scan an entire table is an often misunderstood metric because SQL Server does not read rows; it reads pages. And reading a page is a much different unit of work than reading a row.

This level is shorter than most because it focuses solely on how SQL Server performs IO. And an understanding of SQL Server IO is necessary for understanding why some indexes benefit a query and others do not, or why some data modifications execute faster than others, or why some index maintenance tasks require less time than others. In short, a basic knowledge of SQL Server IO is necessary for understanding all subsequent levels in this stairway.

Related Articles

BLOG SQL Server Quickie #2 - Extents
2012/08/04
Yesterday I have uploaded my 2nd SQL Server Quickie to YouTube. In this episode I'm talking about E...

BLOG SQL Server Quickie #2 – Extents
2012/08/04
Yesterday I have uploaded my 2nd SQL Server Quickie to YouTube. In this episode I’m talking about Ex...

BLOG Querying Microsoft SQL Server :
E
2
Q
Ir
S

BLOG E
2
Q
Ir
S

ARTICLE C
S
2
N
indexing, and taken a deeper dive into Nonclustered Indexes, this ...

Stay up to date:
Daily newsletters with brand new articles, scripts, editorials and a Question of the Day help you keep on top of SQL Server.
your@email.com
Sign up
No thanks

Tags
indexing
stairway series

Write For Us
Find out more

Copyright © 2002-2018 Redgate. All Rights Reserved. Privacy Policy. Terms of Use. Report Abuse.

When you create a database, you specify the files in which your database will be located. SQL Server regards each file as one long string of bytes. It divides the file, logically but not physically, into blocks of 8K bytes. These 8K blocks are called pages. Thus the first 8K bytes of the file is page #0, the next 8K is page #1, and so on. A page is the smallest unit of I/O. SQL Server always reads or writes at least one page per I/O. If multiple contiguous pages need to be read or written, SQL Server may choose to do them all in a single I/O.

A page is not only a unit of I/O; it is also a unit of ownership. That is, if a page contains a row from *TableA*, it will contain nothing but rows from *TableA*; if it contains an entry from nonclustered *IndexB*, it will contain nothing but entries from *IndexB*. Along with its data, each page contains some header information and a set of offset pointers that help SQL Server locate individual rows or entries on the page.

In previous Levels, we have presented several figures to illustrate the sequencing of entries in an index, be it a clustered or nonclustered index. Below, we expand one of those figures, showing the clustered index SalesOrderDetail table, to reflect the concept of pages:

```
SalesOrderID SalesOrderDetailID ProductID
OrderQty UnitPrice

Page n-1:
43668        106                722
3          178.58
43668        107                708
1           20.19

Page n:
43668        108                733
3          356.90
43668        109                763
3          419.46
43669        110                747
1          714.70
43670        111                710
1            5.70
43670        112                709
2            5.70
43670        113                773
2        2,039.99
43670        114                776
1        2,024.99
43671        115                753
1        2,146.96
43671        116                714
2           28.84
43671        117                756
1          874.79

Page n+1:
43671        118                768
2          419.46
43671        119                732
2          356.90
43671        120                763
2          419.46
43671        121                755
2          874.79
```

```
43671        122                764
2          419.46
43671        123                716
1           28.84
43671        124                711
1           20.19
43671        125                708
1           20.19
43672        126                709
6            5.70
43672        127                776
2        2,024.99

Page n+2:
43672        128                774
1        2,039.99
43673        129                754
1          874.79
43673        130                715
3           28.84
43673        131                729
1          183.94
```

There is no requirement that the logical sequence and physical sequence of pages be the same. The sample data shown above is spread across pages n, n+1, n+2, and n+3; but it could have been spread over pages n, n+9, n-5 and n+2. This deviation between the logical and physical sequence is called external fragmentation. Correspondingly, the percentage of empty space within a page referred to as the internal fragmentation of that page. In an upcoming Level, we look in greater detail at the causes, implications, and cures of both types of fragmentation.

There is also no requirement that each page have the same number of rows. Usually, the normal activity of inserting and deleting rows throughout the indexed table will cause each page to have approximately, but not exactly, the same number of rows / entries. To be more correct, each page will have approximately, but not exactly, the same amount, in bytes, of data. If the rows or index entries contain variable length columns, then the number of rows per page may vary considerably even though the number of bytes per page remains fairly even.

The size of a row is equal to the size of its columns plus the row overhead. The amount of row overhead depends on a variety of factors; but can be summarized as follows:

- Six bytes per row for status information and length information.
- One bit per fixed width column, rounding up to the nearest byte.
- If there are any variable length columns - four bytes for the first, plus two bytes for each additional variable length column
- An additional two bytes per row for the offset pointer located at the end of the page.

Because *SalesOrderDetail* rows have a variable length column, their size is not predetermined; but they average about 95 bytes per row. Since a page is 8K bytes in size, the number of rows per page for the *SalesOrderDetail* table typically will be about 75, considerably more than the ten per page shown in our example. In a future level we discuss the SQL Server management tools that you can use to determine these numbers.

So, although we often speak of SQL Server as "reading rows", it can be misleading to do so. SQL Server does not read rows, it

reads, at a minimum, pages. And we have been misleading in saying that an index allows SQL Server to quickly access a row given its index key value. It is more correct to say that an index allows SQL Server to quickly access a page, not a row, given the index key value. Once SQL Server has brought one or more pages into memory, it examines the page and locates the row(s) that was requested.

SQL Server does another logical grouping on top of pages: it groups eight consecutive pages into a unit called an extent. Normally, an extent, like a page, is a unit of ownership. If one page in an extent is owned by *Table4* or *IndexB*, all eight pages will be owned by that same object. An exception is made for very small tables and indexes, ones that would not fill and entire extent. In this case, more than on table or index might be located in the same extent. But for most objects, the extent is a unit of ownership.

Thus, SQL Server does not look upon a table scan as having to read all the rows of the table; but rather as having to read all the pages or extents of the table. It knows that it will be able to issue I/O requests of 8K bytes, 64k bytes, or even more, possibly in parallel, to read the table. This makes the table scan a much less intimidating option than it would be if each row had to be read individually.

Not only does this reading of pages and extents mean that doing a table scan is less work that we might expect, it also means that, to benefit from a nonclustered index, a query must be more selective than we might expect. Consider the following hypothetical query involving the *SalesOrderDetail* table that requests, for the sake of illustration, approximately 4% of the table's rows:

Query:	SELECT * FROM Sales.SalesOrderDetail WHERE ProductID = 712
Clustered Index:	*SalesOrderID / SalesOrderDetailID*
Average Number of Rows per Page:	75
Nonclustered Index:	ProductID
Percentage of rows being requested:	4%

Since only one row out of every twenty five will be selected, and since their entries will be grouped together in the ProductID nonclustered index, using the ProductID nonclustered index to locate the rows in the table might seem like a good idea. But think again.

Thanks to its clustered index, the table, is in *SalesOrderID / SalesOrderDetailID* sequence, not *ProductID* sequence. Therefore, if the average page holds 75 rows, and the query requests one row out of every 25, the average page will contain 3 of the requested rows, and almost every page will contain at least one of the requested rows. In other words, almost every page will need to be read to satisfy the request. And it would be best to read them with a table scan, one that reads an extent, or more, at a time, bringing an avevage of 24 requested rows (3 * 8) into memory per read.

Newcomers to SQL Server often ask "How selective must a query be for a nonclustered index to be used?". At this Level, you now

know one answer to that question, "More selective than one row per page." Information provided in future Levels will enable you to become ever more accurate in determining which of your indexes are beneficial and which are not.

SQL Server does not read rows. It reads data in units of one page or more. The page, which is the smallest unit of IO, is 8K in size. An extent is 8 consecutive pages in a data file. Normally an extent, and therefore its pages, contains rows or entries of a single object, be it a heap or an index. Because of the efficiency provided by large units of IO, a query must be highly selective to benefit from a nonclustered index.

In Level 5, we look at something you can do to increase the possibility that a nonclustered index will benefit a query, a possibility that becomes a near certainty when the index becomes a covering index for one or more of your queries. In other words, the next Level is all about adding included columns to your indexes.

Downloadable Code

- Pages.SQL

Level 4 - Pages.sql

This article is part of the Stairway to SQL Server Indexes Stairway

Sign up to our RSS feed and get notified as soon as we publish a new level in the Stairway!

Keep up to date with SQL Server - new articles every day.

your@email.com

Sign up

Thank this author by sharing: 0

Rate this | Join the discussion | Add to briefcase

Total article views: 12924 | Views in the last 30 days: 22