sqlity.net

1.832.3SQLITY (775489)

B+Trees – How SQL Server Indexes are Stored on Disk

2014-06-18 - General, Series, SQL Server Internals, Storage Wednesday

Introduction

SQL Server organizes indexes in a structure known as B+Tree. Many think, B+Trees are binary trees. However, that is not correct. A binary tree is a hierarchical structure organizing nodes (table rows) in a manner that allows searches to be executed extremely efficiently. On the flipside, the binary tree structure is very volatile when it comes to updates, often requiring the entire structure to be rebuilt when a single data point was changed. (See Wikipedia for more details.) That makes binary tree a very poor choice when it comes to persisting data on disk.

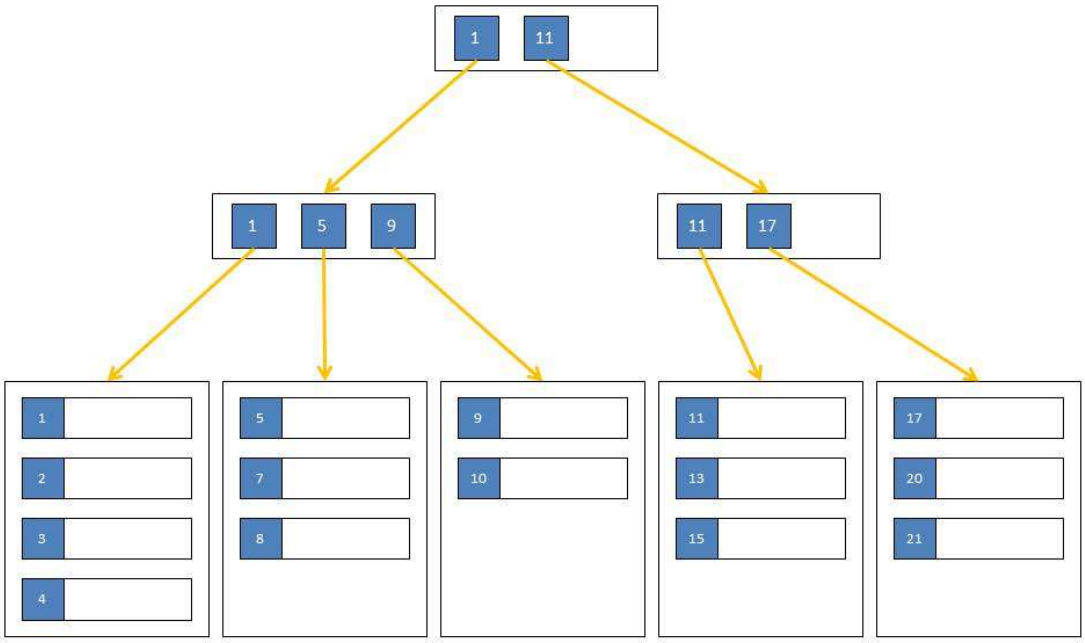
The B-Tree

In 1972, Rudolf Bayer and Ed McCreight, both working at Boeing at the time, were looking for a way to overcome some of the shortfalls of the binary tree. Their solution was the B-Tree. The main difference between a binary tree and a B-Tree is that the latter allows for more than one data point (table row) per node. B-Trees are also balanced, which means that the time it takes to execute a search within this structure is mostly independent of the value to be found.

For a long time it was unclear what the "B" in the name represented. Candidates discussed in public where "Boeing", "Bayer", "Balanced", "Bushy" and others. In 2013, the B-Tree had just turned 40, Ed McCreight revealed in an interview, that they intentionally never published an answer to this question. They were thinking about many of these options themselves at the time and decided to just leave it an open question.

The B+Tree

B-Trees are a lot more efficient than binary trees when it comes to updates, but some operations can still turn out expensive, depending on where the node that will hold the new or updated data lives in the tree. Therefore, another optimization was made to B-Trees to help with this problem. Instead of treating all nodes equal, the new structure has two types of nodes. The lowest level nodes, also called leaf nodes, hold the actual data. All other nodes including the root node only hold the key values and pointers to the next nodes. This type of tree is called a B+Tree and you can see an example below:



There are no limitations on the number of key-pointer-pairs or data rows within a node. The only limitation is that all leaf nodes have the same distance from the root node. That means that the work to seek for a particular data point is always the same, no matter what the key value is. It also keeps updates very localized in the tree. I might have to move a few rows to a new node during an update but then I probably only need to change a single parent node to integrate that new node. It is however possible for a single change to affect every level of the tree, but those changes are rare. You can find more information about B+Trees here.

B+Trees in SQL Server

SQL Server stores its indexes in B+Tree format. There are a few exceptions - for example temporary hash indexes, created during a hash join operation, or column store indexes, which are not really indexes at all. However, all key-based clustered and non-clustered persisted SQL Server indexes are organized and stored as B+Trees.

Each node in such a tree is a page in SQL Server terms and you will find two page types in each index. The first type are the data pages (pages of type 1). Each leaf level node in a SQL Server B+Tree index is a single data page. The second type are intermediate index pages (pages of type 2). Each node in an index B+Tree that is not a leaf level node is a single page of type 2. Those pages contain rows just like the data pages. But in addition they contain a pointer for each row that identifies the next child page. That child page can be either of type 1 or of type 2, depending on the location in the B+Tree.

Summary

SQL Server stores key-based persisted indexes in the form of a B+Tree. Each node in such a tree is represented by a single page. Data pages build the leaf level of the tree while all other nodes are made of single pages of type 2.

Categories: General, Series, SQL Server Internals, Storage Wednesday
Tags: b-tree, B+Tree, binary tree, Index, internals, page, SQL Server, storage engine, tree

3 comments

livefyre
- Get Livefyre
- FAQ

Newest | Oldest

Pauzi

I am confused. You are saying the leaf nodes of an index contain the actual data row, be it clustered or non-cluetered index. That's not what I understand from reading other articles on sql server indexes. If what you're saying is true, then let's say we have an employee table with columns empID, firstname and lastname. The clustered index is by the field empID. For this index the leaf nodes will have data rows roughly sorted by the empID. Leaf node 1 has empID 1-100, node 2 has empID 101-200, and so on.

Now if we define another index with the key being lastname we will have the leaf node containing actual data rows, according to you, sorted by the last name of employee. Node 1 will have Ashley - Bernard, node 2 has Connors - Davidsan, node 3 has Eli - Ferguson, an so on. But, in real world there is no guarantee that the sorting will match the sorting by empID, that is Ashley has empID 1, Bernard 100, Connors 101, Davidson 200, and so on. We will probably have Ashley with an empID 456, Bernard 102, Connors 3, Davidson 999.

Now I am just wondering how can the same data rows be sorted in two different ways. One by empID and another by lastname. I admit I am not that bright but if you can show me, graphically if possible, how that is possible I will be very grateful, then I will start my own crusade to point out to many people whose articles are saying the opposite of what you are saying here.

@sqlity moderator

@Pauzi In a B+Tree, only the leaf level contains "Data" while the other levels contain only key values. However, what "Data" means is different for clustered and nonclustered indexes as well as for unique and nonunique indexes. The actual base-table row data you will find only in the leaf level of a clustered B+Tree index. For a nonclustered index the "data" contains included columns (if any) as well as the "pointer" that identifies the position of the base-table row in the clustered index or the heap. (What that pointer looks like is again different for clustered tables and heaps.) The key values themselves are also stored in the leaf level, but some of them you'll also find in the other levels.

Pauzi

@@sqlity @Pauzi Aah ok. Thanks for clearing that up. I was confused by the word 'data'.

Trackbacks

1. *Intermediate Index Pages in SQL Server - sqlity.net* says:
   2014-07-02 at 11:05
   [...] Server organizes all conventional indexes on disk in a structure called B+ Tree. In a B+ Tree two types of nodes exist. The first type is the leaf node containing the actual data. [...]

   Log in to Reply

2. *Data Compression | Simple SQL Server* says:
   2015-12-08 at 08:01
   [...] it's easier to get to the data because the smaller data may have fewer levels in the B+Tree (+). Along the way it has to decompress the root and intermediate level pages (-) which are always [...]

   Log in to Reply

3. *Data Compression - SQL Server - SQL Server - Toad World* says:
   2015-12-08 at 09:47
   [...] it's easier to get to the data because the smaller data may have fewer levels in the B+Tree (+). Along the way it has to decompress the root and intermediate level pages (-) which are always [...]

   Log in to Reply

4. *The 'B' in B-Tree – Indexing in SQL Server - SQL Hammer* says:
   2016-04-04 at 09:06
   [...] B+Trees – How SQL Server Indexes are Stored on Disk (sqlity.net) [...]

   Log in to Reply

5. *What Are B-trees In Sql Server – GoogleTeach* says:
   2016-04-09 at 23:19
   [...] B+Trees – How SQL Server Indexes are ... – Introduction. SQL Server organizes indexes in a structure known as B+Tree. Many think, B+Trees are binary trees. However, that is not correct. A binary tree is a ... [...]

   Log in to Reply

« THE SYS.SYMMETRIC_KEYS CATALOG VIEW

FIVE REASONS WHY YOU NEED TO ENCRYPT YOUR PHI »

We know how to make a database fast. We'll help you do the same.

Free Poster: The 10 Most Common Database Vulnerabilities

Get Yours Now!