SQLShack

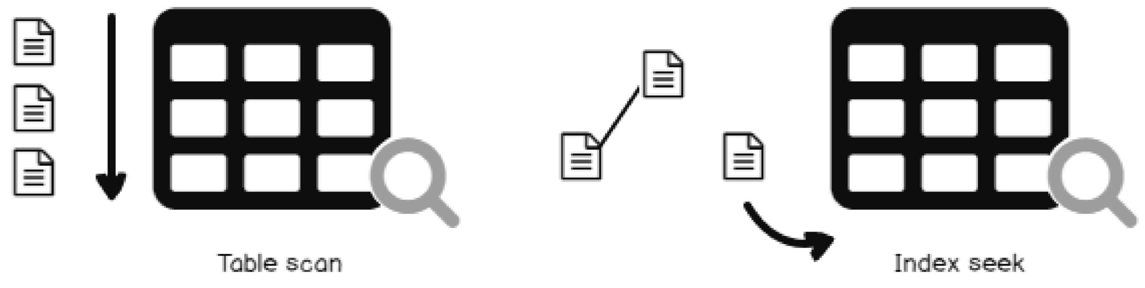
SQL Server index structure and concepts

March 19, 2018 by Ahmad Yaseen

In my previous article, SQL Server Table Structure Overview, we described, in detail, the difference between Heap table structure, in which the data pages are not sorted in any ordering criteria and the pages itself are not sorted or linked between each other, and Clustered tables, in which the data is sorted within the data pages and the pages will be also linked in a double linked list, based on the index key. In this article, we will go through the structure of the SQL Server index, itself.

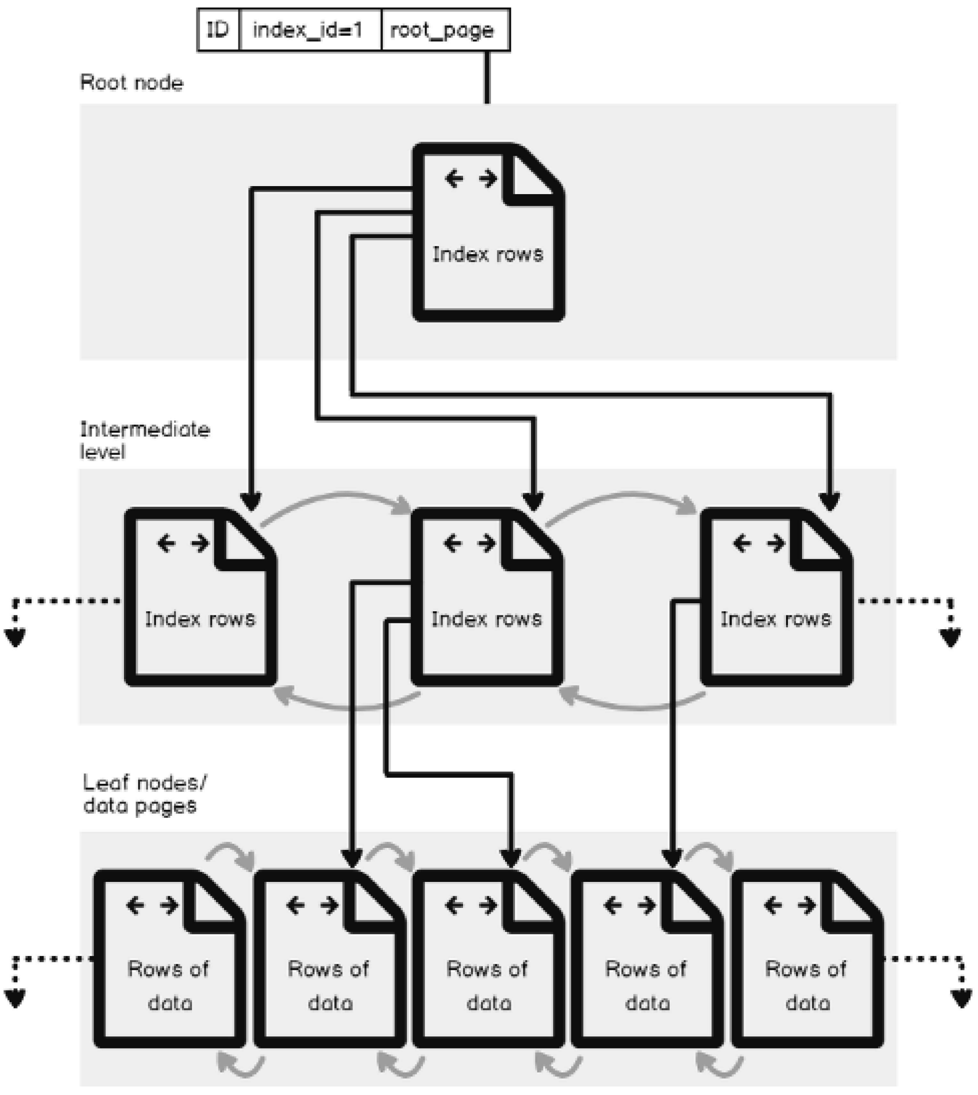
SQL Server index is considered as one of the most important factors of the performance tuning process, that is created to speed up the data retrieval and the query processing operations from a database table or view, by providing swift access to the database table rows, without the need to scan all the table's data, in order to retrieve the requested data. You can imagine the table index as a book's index that allows you to find the requested information very fast within your book, rather than reading all the book pages in order to find a specific subject.

Assume that you have a query that retrieves list of employees' information from the Employees table based on the EmployeeID column. Without having an index on the EmployeeID column, SQL Server will scan all the table rows to retrieve the requested data. If you create an index on the EmployeeID column on the Employees table, and perform a search based on the EmployeeID value, the SQL Server Engine will seek for the requested EmployeeID values in the index and use that index to locate the rest of the employees' information from the related rows in the source table, providing a significant performance enhancement and reducing the effort required to locate the requested data, as shown in the figure below:



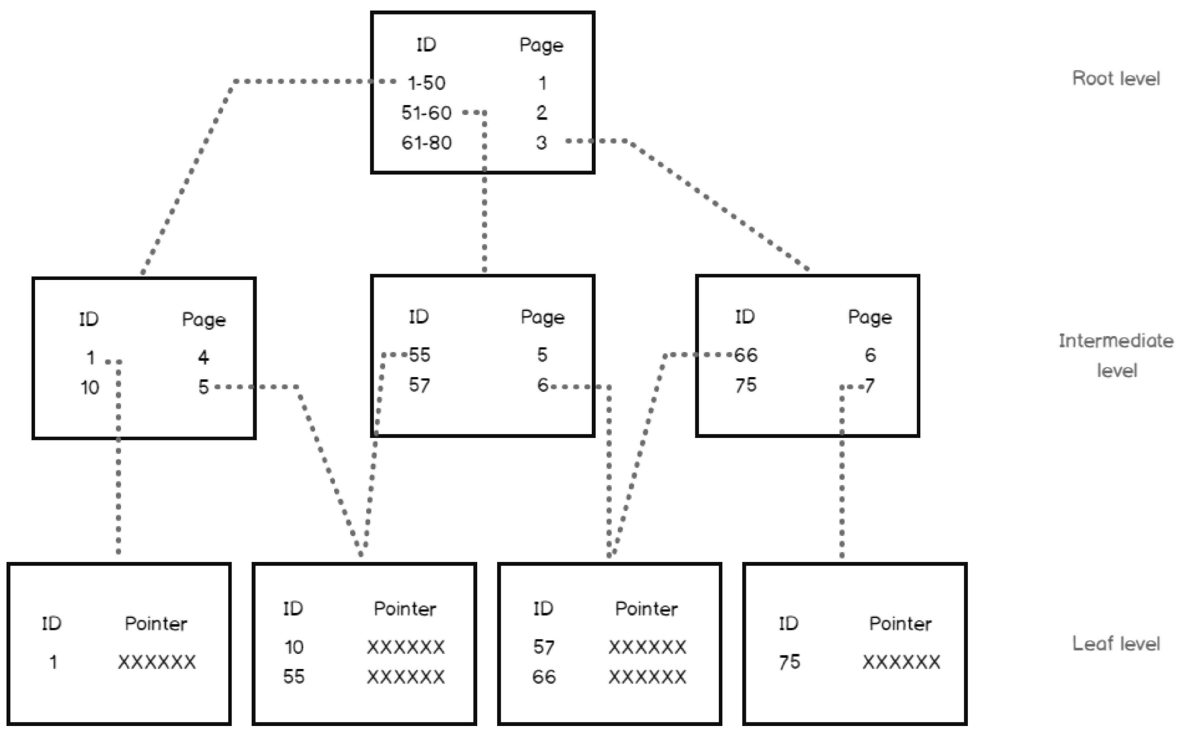
The rapid search capabilities provided by the index is achieved due to the fact that, the SQL Server index is created using the shape of B-Tree structure, that made up of 8K pages, with each page in that structure is called an index node. The B-Tree structure provides the SQL Server Engine with a fast way to move through the table rows based on index key, that decides to navigate left or right, to retrieve the requested values directly, without scanning all the underlying table rows. You can imagine the performance degradation that may occur due to scanning large database table.

It is clear from the Index B-Tree Structure figure below, that the B-Tree structure of the index consists of three main levels: the **Root Level**, the top node that contains a single index page, form which SQL Server starts its data search, the **Leaf Level**, the bottom level of nodes that contains the data pages we are looking for, with the number of leaf pages depends on the amount of data stored in the index, and finally the **Intermediate Level**, one or multiple levels between the root and the leaf levels that holds the index key values and pointers to the next intermediate level pages or the leaf data pages. The number of intermediate levels depends on the amount of data stored in the index.



Assume that we create an index in one of our database tables on the ID column. When you run a query to search for specific rows from that table, based on the ID values of these rows, the SQL Server Engine will start navigating from the root node, to determine which page to reference in the top intermediate level, then continue down through the intermediate nodes to identify the address of the next intermediate node, until it reaches the target leaf node that contains the requested data row or pointer to that row in the main table, depends on the type of the index.

For example, if you issue a query that searches for the row with ID value equal to 57, the SQL Server Engine will start searching in the root node of the index, where it will find that the ID value of 57 exists in the second intermediate node. In the second intermediate node, it will find also that the ID value of 57 is located in the leaf node number 6, where the record with ID value equal to 57, or a pointer to that row will be found on the leaf node, as shown below:



SQL Server indexes can have large number of nodes in each level. This helps in improving the efficiency of the created index by avoiding the need for excessive depth within the index. The **index depth** is the number of levels from the index root node to the leaf nodes. An index that is quite deep will suffer from performance degradation problem. In contrast, an index with large number of nodes in each level can produce a very flat index structure. An index with only 3 to 4 levels is very common.

In addition to the index depth, there are two other important index measurements that control the index effectiveness. The first property is the **index density** which is a measure of the lack of uniqueness of the data in a table. A dense column is one that has a high number of duplicates. The second property is the **index selectivity**, which is a measure of how many rows scanned compared to the total number of rows. An index with high selectivity means a small number of rows scanned when related to the total number of rows.

SQL Server provides us with two main types of indexes, the **Clustered index** that stores the actual data rows of the table at the leaf level of the index, in addition to controlling the sorting criteria of the data within the data pages and the order of the pages itself, based on the clustered index key. This is the reason behind the ability to create only one clustered index on each table. The **Non-clustered index** contains only the values of the index key columns with a pointer to the actual data rows stored in the clustered index or the underlying table, without controlling the order of the data within the pages and the order of the index pages. SQL Server allows you to create up to 999 non-clustered indexes on each table. Recall from the previous article that the table with no clustered index is called **Heap** table, with no criteria that is controlling the data and pages order, and the table that is sorted using a clustered index is called a **Clustered** table. A clustered index will be created automatically when you define a Primary Key constraint in the table, if these is no predefined clustered index on that table.

There are other types of SQL Server indexes, such as the **Unique index** that enforces the column values uniqueness and created automatically when defining a unique constraint, the **Composite** values uniqueness and created automatically when defining a unique constraint, the **Composite index** that contains more than one key column and the **Covering index** that contains all columns requested by a specific query. We will go through all these types in details in the coming articles of this series.

For this point, you should have a full understanding about the table structure, the index structure and the general benefits of adding the indexes. Before going through the index design, usage and improvement, you have to take into consideration that the index is a **double-edged sword**, where a well-designed index will enhance the performance of your system and speed up the data retrieval process. On the other hand, a badly-designed index will cause performance degradation on your system and will cost you extra disk space and delay in the data insertion and modification operations. It is better always to test the performance of the system before and after adding the index on the development environment, before adding it to the production environment.

Table of contents

Ahmad Yaseen

Ahmad Yaseen is a Microsoft Big Data engineer with deep knowledge and experience in SQL BI, SQL Server Database Administration and Development fields.

He is a Microsoft Certified Solution Expert in Data Management and Analytics, Microsoft Certified Solution Associate in SQL Database Administration and Development, Azure Developer Associate and Microsoft Certified Trainer.

Also, he is contributing with his SQL tips in many blogs.

View all posts by Ahmad Yaseen

Related Posts:

1. **Designing effective SQL Server non-clustered indexes**
2. **Top 25 SQL interview questions and answers about indexes**
3. **Designing effective SQL Server clustered indexes**
4. **Top 10 questions and answers about SQL Server Indexes**
5. **SQL Server index operations**

Indexes

29,040 Views