INDEXES

Knowledge sharing on Indexes in SQL Server  
 Huu Phat LE

SQL Server is the engine that allows you to store, modify, and retrieve data. Working with databases is a very different prospect from working with application programs, which tend to perform one (or a few) very small tasks sequentially. Requests to a database server often incur massive retrieval and sort operations under the covers, even when the outcome is a very small result set.

In addition to good database design and effective query writing, one of the most effective and essential elements to an efficient database system is indexing.

# What is Indexes in SQL Server?

Indexes play a key role in SQL Server performance.

A database index is very much like the index in a book: the book index has an alphabetized list of topics with page numbers to the location of the data. A database index has an ordered list of values (made up of one or more table columns), with pointers to the row in which the value and its corresponding data reside.

Without indexes, any query or data modification causes the SQL engine to search the referenced tables from the top down. This is akin to searching for a piece of information in a book by reading it from page 1. A single well-placed index can shorten your query time from dozens of minutes to under a second.

# Types of Indexes in SQL Server

Basically, the index is classified into two categories: Clustered Index and Non-Clustered Index

**Clustered Index**

* 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. There can be only one clustered index per table, because the data rows themselves can be stored in only one order.
* The only time the data rows in a table are stored in sorted order is when the table contains a clustered index. When a table has a clustered index, the table is called a clustered table.

**Non-Clustered Index**

* Non-clustered indexes have a structure separate from the data rows. A non-clustered index contains the non-clustered index key values and each key value entry has a pointer to the data row that contains the key value.
* The pointer from an index row in a non-clustered index to a data row is called a row locator. The structure of the row locator depends on whether the data pages are stored in a heap or a clustered table. For a heap, a row locator is a pointer to the row. For a clustered table, the row locator is the clustered index key.

To add to the two basic types of Index, Clustered Index and Non-Clustered Index, we can expand the Index type in the following ways:

**Unique Index**

* The type of Index used to ensure uniqueness in the columns that are created Index. If the index is created based on multiple columns, the uniqueness of the value is calculated on all those columns, not just the individual columns.

For example, if you created an Index on the FirstName and LastName columns in a table, the values ​​of these two columns must be unique, but the values ​​may still be the same.

* A Unique Index is automatically generated when you define a primary key or a unique constraint (Unique Constraint)

**Index with included columns**

* An index on a column that is derived from the value of one or more other columns, or certain deterministic inputs.

**Covering Index**

* An index type that includes all the columns needed to process a particular query. For example, your query might retrieve the FirstName and LastName columns from a table, based on a value in the ContactID column. From there, to speed up query processing, you can create an index that includes all three of these columns.

# Indexes Design

Because Index can take up a lot of hard drive space, it is not advisable to deploy too many Indexes if they are not really needed. In addition, the Index will be automatically updated when the data stream itself is updated, so that may result in additional costs and affect the performance of the data processing process. Therefore, designing the Index in SQL Server requires some considerations before implementing it.

For tables that are accessed more and more frequently, use as few columns as possible in an Index and do not use the Index sporadically on tables of data.

* If a table has a large volume of data but a low frequency of data updates, you should use many indexes needed to improve query performance.
* For Clustered Index, try to keep the length of the indexed columns as short as possible. Ideally, create a Clustered Index on a column with a Unique attribute and do not allow Null values. This is why primary keys are often used for the Clustered Index of the table
* The uniqueness of the values ​​in a column affects the performance of the Index. In general, the more duplicates are, the worse the performance of the Index is. In other words, the higher the uniqueness of the value in a column, the higher the efficiency of the Index. Therefore, if the values ​​of a certain column in a table are unique, then you should create a unique index on that column.

**Few hard and fast rules for indexing**

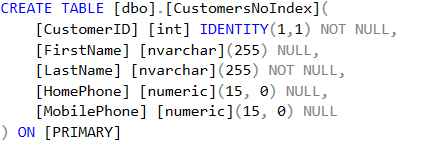
* Each table should have a clustered index that is (ideally) small, selective, ever increasing, and static. (Note that a table without a clustered index is called a heap.)
* Implement non-clustered indexes on foreign key relationships – in other words, on columns that are likely to be used in JOINs.
* Implement non-clustered indexes on columns that are frequently used in WHERE clauses.
* Do not implement single-column indexes on every column in a table. This will take up space unnecessarily and cause high overhead on INSERTs and UPDATEs.
* In multi-column indexes, list the most selective (nearest to unique) first in the column list. For example, when indexing an employee table for a query on social security number (SSN) and last name (LastName), your index declaration should be

*CREATE NONCLUSTERED INDEX ix\_Employee\_SSN  
ON dbo.Employee (SSN, LastName);*

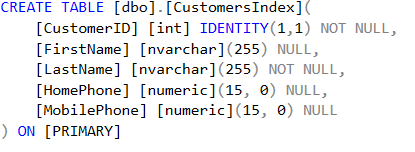
# How Indexes improve performance?

Index is a very powerful means to increase the performance of the statement. Example below will show you how SQL Server uses indexes to increase performance.

First, let’s create two new table, [dbo].[CustomersNoIndex]



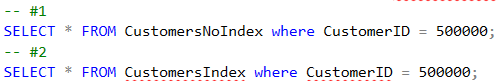
and [dbo].[CustomersIndex],



And create an index on the CustomerID field for the [dbo].[CustomersUsingIndex] table:

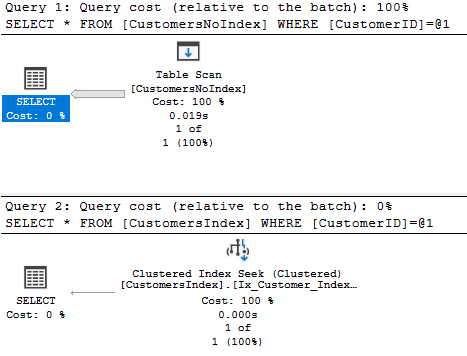


Execute two SELECT statements to query two tables:



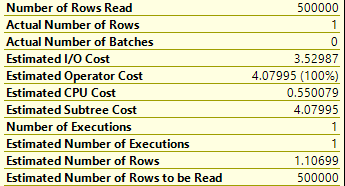
These two statements will give the same result, the only difference is that the second statement queries the CustomersIndex table that has the index on the field to search (CustomerID).

When we view the estimated execution plan, we can see:

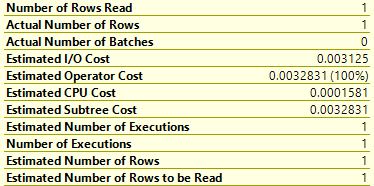


If we hover over the big right-to-left arrow in the estimated plan, we see this:

* On the first statement



* On the second statement



We see that the first statement accounted for 100% of the total cost, while the second statement only accounted for 0%. In other words, the index on the CustomerID field has helped the command perform up to 19 times faster. Because, the main thing SQL Server will do on the first statement is a table scan. That mean it will read all the rows of this table until it finds one with a Customer Id of 500000. And in the second statement, SQL Server only needs to read only one row of this table.

In addition, the index of resources consumed when using the index is also improved.

Thus, Index helped the amount of data the system needs to process to find the result to a minimum and improve query performance.

**Reference for more information**

<https://docs.microsoft.com/sql/relational-databases/indexes>