<https://www.mssqltips.com/sqlservertip/1206/understanding-sql-server-indexing/>

**Solution**

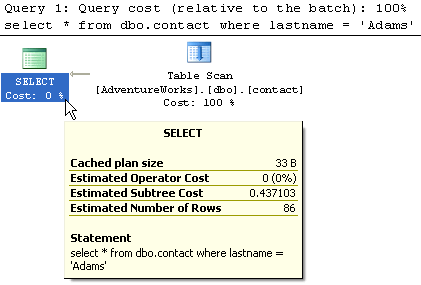
SQL Server offers two types of indexes **clustered** and **non-clustered**. In its simplest definition a clustered index is an index that stores the actual data and a non-clustered index is just a pointer to the data. A table can only have one Clustered index and up to 999 Non-Clustered Indexes (depending on SQL version). If a table does not have a clustered index it is referred to as a Heap.

The primary reason indexes are built is to provide faster data access to the specific data your query is trying to retrieve. This could be either a clustered or non-clustered index. Without having an index SQL Server would need to read through all of the data in order to find the rows that satisfy the query. If you have ever looked at a query plan the difference would be an Index Seek vs a Table Scan as well as some other operations depending on the data selected.

Here are some examples of queries that were run. These were run against table dbo.contact that has about 20,000 rows of data. Each of these queries was run with no index as well as with a clustered and non-clustered indexes. To show the impact a graphical query plan has been provided. This can be created by highlighting the query and pressing Control-L (Ctrl-L) in the query window.

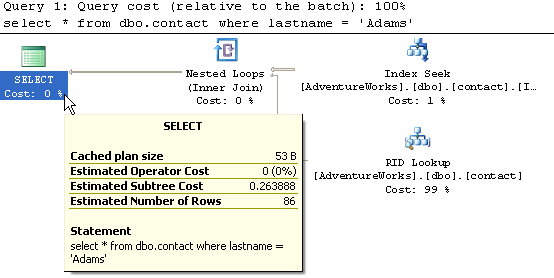
1 - Table with no indexes

When the query runs, since there are no indexes, SQL Server does a Table Scan against the table to look through every row to determine if any of the records have a lastname of "Adams". This query has an Estimated Subtree Cost of 0.437103. This is the cost to SQL Server to execute the query. The lower the number the less resource intensive for SQL Server.



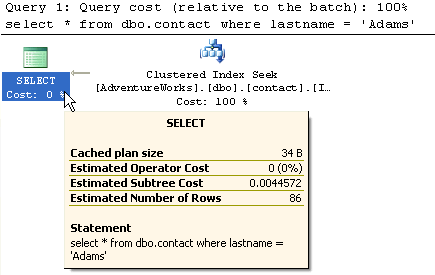
2- Table with non-clustered index on lastname column

When this query runs, SQL Server uses the index to do an Index Seek and then it needs to do a RID Lookup to get the actual data. You can see from the Estimated Subtree Cost of 0.263888 that this is faster then the above query.



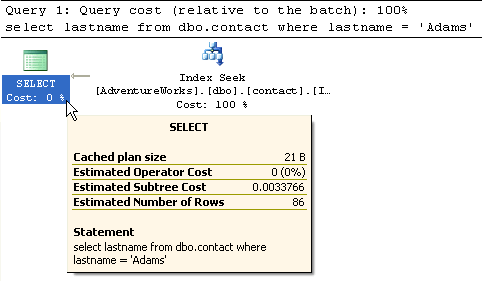
3- Table with clustered index on lastname column

When this query runs, SQL Server does an Index Seek and since the index points to the actual data pages, the Estimated Subtree Cost is only 0.0044572. This is by far the fastest access method for this type of query.

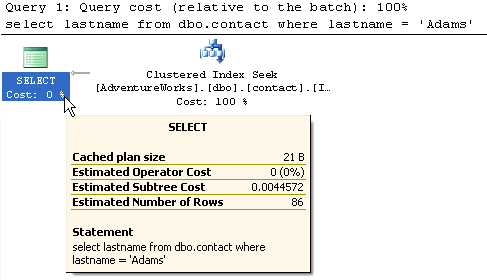


4- Table with non-clustered index on lastname column

In this query we are only requesting column lastname. Since this query can be handled by just the non-clustered index (covering query), SQL Server does not need to access the actual data pages. Based on this query the Estimated Subtree Cost is only 0.0033766. As you can see this even better then example #3.

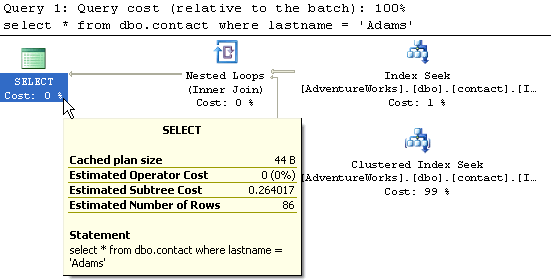


To take this a step further, the below output is based on having a clustered index on lastname and no non-clustered index. You can see that the subtree cost is still the same as returning all of the columns even though we are only selecting one column. So the non-clustered index performs better.



5- Table with clustered index on contactId and non-clustered on lastname column

For this query we now have two indexes. A clustered and non-clustered. The query that is run in the same as example 2. From this output you can see that the RID Lookup has been replaced with a Clustered Index Seek. Overall it is the same type of operations, except using the Clustered Index. The subtree cost is 0.264017. This is almost the same as example 2.



So based on these examples you can see the benefits of using indexes. This example table only had 20,000 rows of data, so this is quite small compared to most database tables. You can probably imagine the impact this would have on very large tables.

The first idea that would come to mind is to use all clustered indexes, but because this is where the actual data is stored a table can only have one clustered index. The second thought may be to index every column. Although this maybe helpful when querying the data, there is also the overhead of maintaining all of these indexes every time you do an INSERT, UPDATE or DELETE.

Another thing you can see from these examples is ability to use non-clustered covering indexes where the index satisfies the entire result set. This is also faster then having to go to the data pages of the Heap or Clustered Index.

To really understand what indexes your tables need you need to monitor the access using a [trace](https://www.mssqltips.com/sqlservertip/1035/sql-server-performance-statistics-using-a-server-side-trace/) and then analyze the data manually or by running the [Index Tuning Wizard](http://www.microsoft.com/technet/prodtechnol/sql/2000/maintain/tunesql.mspx) (SQL 2000) or the [Database Engine Tuning Advisor](http://msdn2.microsoft.com/en-us/library/ms173494.aspx). From here you can tell whether your tables are over indexed or under indexed.

**Next Steps**

* Take a look at these other [tips on indexing](https://www.mssqltips.com/sql-server-tip-category/38/indexing/)
* [Improve SQL Server Performance with Covering Index Enhancements](https://www.mssqltips.com/sqlservertip/1078/improve-sql-server-performance-with-covering-index-enhancements/)
* Read through books online to get a better understanding on indexing