**THE TIPPING POINT IN SQL SERVER**

**SEEK vs SCAN**

Toadere Andreas-Robert

*Computer Science,*

*Babeș-Bolyai University, Cluj-Napoca, Romania*

[taie2634@scs.ubbcluj.ro](mailto:taie2634@scs.ubbcluj.ro)

**Abstract**

This paper presents the concept of the “Tipping Point” in Microsoft SQL Server. This concept of the “Tipping Point” can be defined as a behavior specific to the server in deciding when to use an index seek and when to use an index scan in order to optimize a query in a specific situation. This specific situation involves a decision that the system has to take between performing an index seek on a non-covering non-clustered index or a scan - of course, if there is a covering non-clustered index the system will surely chose to perform an index seek on that respective index. The “Tipping Point’’ occurs because the system has to estimate the necessary resources for the ways that a query can be performed and then the most performant one is chosen. The significant difference in performance between a seek and a scan is represented by both the number of I/O (input/output) operations performed by the query that are necessary for the Lookup. Although the “Tipping Point” concept is not found in the documentation, it is widely known between users and can easily be found on forums etc. This paper is trying to find the context in which the system changes it decision from performing an index seek and what are the factors which influents that decision.

**Keywords**

*SQL-Server, Tipping Point, Index seek, Index scan*

**1. Introduction**

As the data amount worldwide is doubling every two years, the optimization of manipulating data is more important than ever. Handling huge amounts of data in as little time as possible has become a must nowdays , with hardware inovation incapable to cope with software development and the increasing amount of data. Therefore, database systems have to make the most performant decisions on a regular basis in order to satisfy the users’ needs in timewise.

As for this paper, it will study the SQL-Server decision making when it comes to access and retrieve data. Moreover, the goal of this paper is to present a more clear answer to the question “When does an index scan become more efficient than an index seek?”. Many people have found themselves proud of discovering this behavior in SQL Server and then calling it a bug – they could not be more wrong.

First, it must be understood why performing an index seek and then a key lookup can become more expensive than a scan. The answer to that question is that the key lookup is the expensive part timewise when retrieving data via an index seek. In fact, the behavior we are studying is when a key lookup produces so much I/O operation that is essentially costs more time ( or so does the system estimates ) than performing an index scan, which, in this case supposedly reduces the CPU consumption and produces less I/O operation.

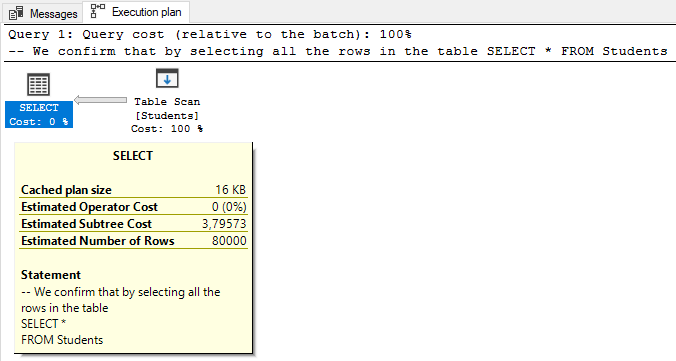
Secondly, it must be understood which are the factors on which the system decides when to switch its *execution plan.* This, in fact, represents the Tipping Point – a multitude of factors that determine the system to choose the scan in the detriment of the index seek.

**2. Methods**

The means of the research include the usage of some SQL-Server provided instruments and the using the method of “trial and error” on a few examples in order to determine when the Tipping Point occurs. As is general knowledge, in SQL-Server *a page* size is 8kb and it can not be changed. Later, we will see that the number of pages that will be retrieved in order to perform a query is the factor that constitutes the system decision in whether to perform a scan or a seek.

In the following examples, we will use the *Execution Plan* in SQL-Server, which is available by hovering part of the program with the cursor and then pressing ***CTRL+L.*** The effect will be that the Execution Plan window will now appear presenting information about each query that was hovered by the cursor: how much each query will cost in proportion to the total cost of all queries and so on. For *SELECT*, it also provides information about how many rows is the query expected to return. This information can be visualized by going with the cursor over the “SELECT” in the execution plan.

For example:

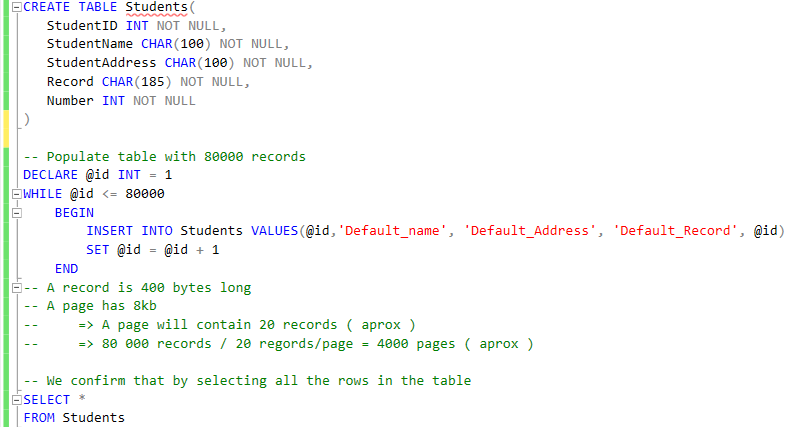


We will also use the “SET STATISTICS” command in order to retrieve data from the last executed query. For the following examples the “SET STATISTICS IO” and “SET STATISTICS TIME” will be used for the research. In addition to that, sometimes the syntax “WITH (FORCESEEK)” will appear among the code, having the effect that the respective query will be completed via an index seek. This actually “overrides” the system and completes the query with an index seek no matter what.

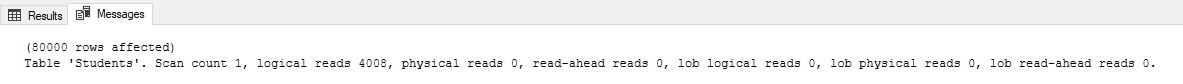
**3. Results**

Example 1:

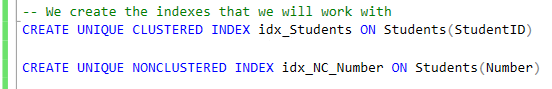
We create a new database with a table called Students and the we populate it with 80000 records:



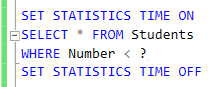
When we perform the SELECT with “SET STATISTICS IO ON” we will get:



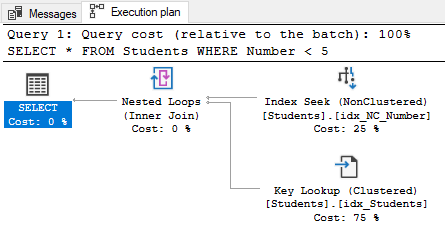
This means that when we perform a scan, the system performs 4008 logical reads. Next step is to create two indexes the following way:



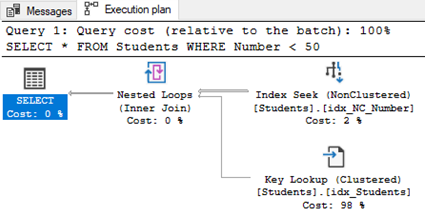
Now we will try to determine when the system will change its *execution plan* from choosing to perform an index seek to choosing to perform a scan. In order to do that we will have a query of the form:



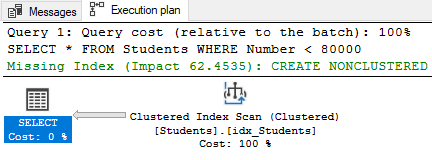
We can see in the following example that when “?” is equal with 5 the *execution plan* is as follows: the system will perform an index seek on non-clustered index “idx\_NC\_Number”. We can also see that the Key Lookup will take 75% of the required time: that means that after the records are find in the non-clustered index via their Number, the system has to retrieve the rest of the information for each record by performing logical reads and I/O which are very costly time-wise.



As we can see below, when “?” becomes 50, the Key Lookup will take no less than 98% of the total required time, proving how much time the key lookup takes.

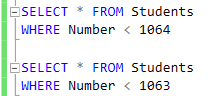


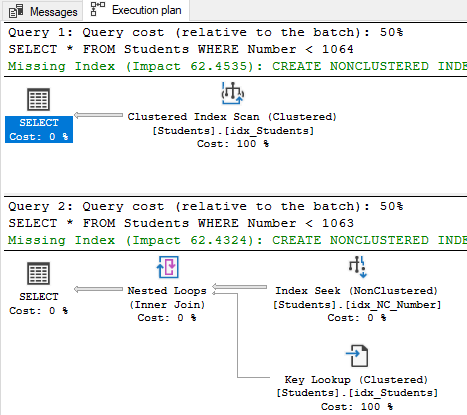
As expected, when “?” is a very large number ( like 80000 in this example ) the system will choose to perform a scan in order to complete the query:



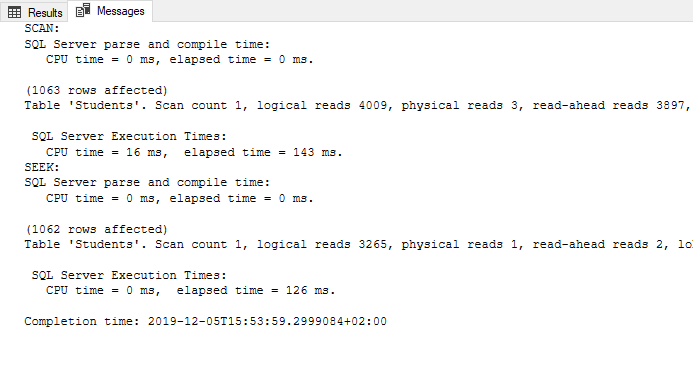
The goal of this paper is to determine the actual number when the change in the execution plan occurs and what is the reasoning behind the change. After a few attempts, the guessing work succeeded and I was able to find that the change to the index scan occurs when the system estimates more than 1062 rows.

Subsequently, for the following two almost identically queries the execution plan is very different:





This is in fact, the “Tipping Point”. After running the queries with the command “SET STATISTICS TIME ON”, we can also see why:

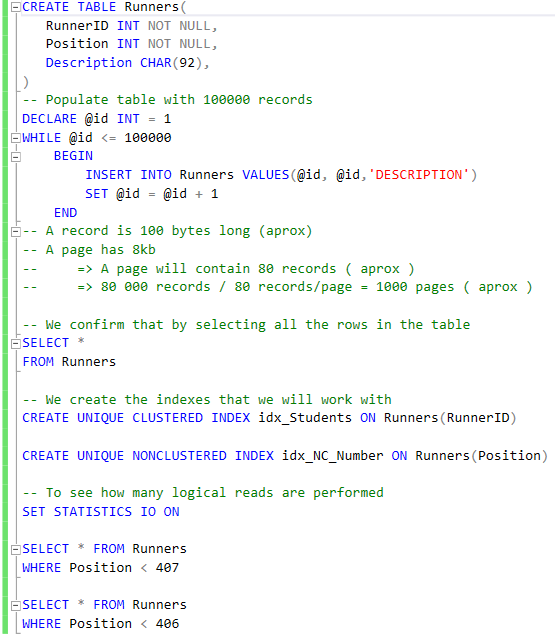


If we look at the elapsed time and the number of logical reads we will see that the queries are very similar performance-wise. This proves that a non-covering index is not so efficient anymore due to the I/O that is required.

The result is that the system starts to perform a scan when the estimated number of rows is 1063 or higher, from the total of 80000. With a quick calculus we find that in this case the Tipping Point occurred when at least 1.32875% rows are estimated to match the query. Also we can say that the switch happens when the index seek and key lookup perform 81.44176% logical reads of the number of logical reads that are performed when the system chooses to scan the records.

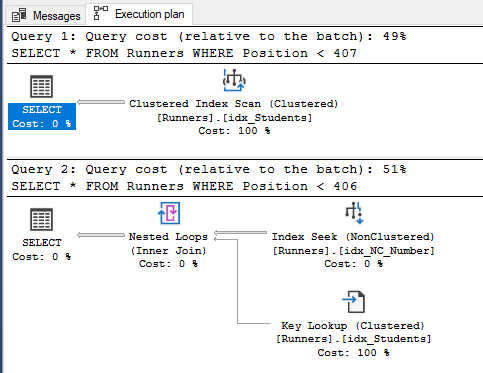
Example 2:

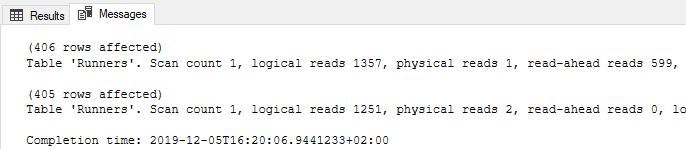
In the second example we do the same thing except now we will insert 100000 record in the table Runners, in which each record is 100 bytes.



Now, by the same reasoning as before we find that the tipping point in this case is 406.

This is the *Execution plan* and statistical output for the selects:



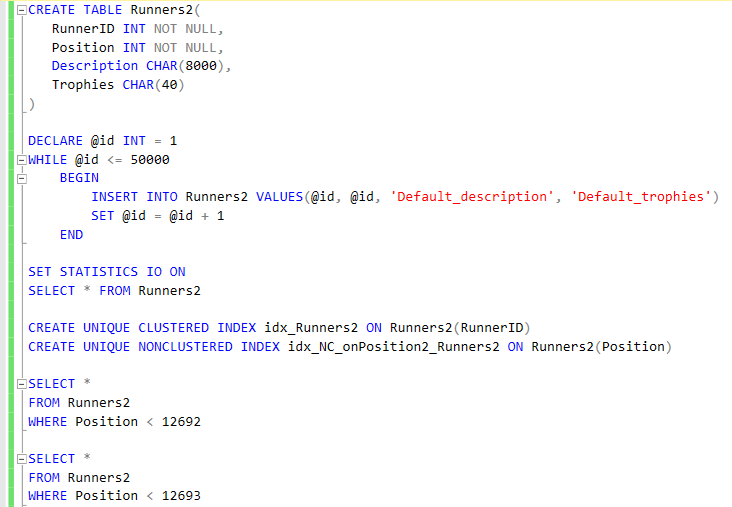


As before, the result is that the system starts to perform a scan when the estimated number of rows is 406 or higher, from the total of 100000. With a quick calculus we find that in this case the Tipping Point occurred when at least 0.406% rows are estimated to match the query. Also we can say that the switch happens when the index seek and key lookup perform 92.18865% logical reads of the number of logical reads that are performed when the system chooses to scan the records.

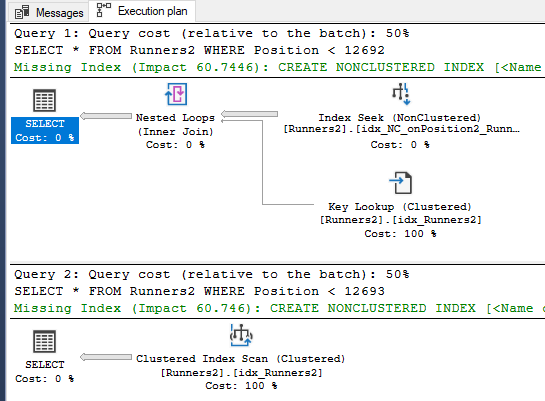
I found the two results from the two examples not to be conclusive enough so I decided to try again the process, but this time to create the table in such a way that a data record requires more than 8kb o that the respective record does not fit on a page. As a mention, I found that the maximum size for any data type for a column is 8000. Then I tried to design the table such that a record exceeds the size of 8kb without having an attribute over 8kb, failing in doing so and getting the following error message: “Creating or altering table 'Runners2' failed because the minimum row size would be 12015, including 7 bytes of internal overhead. This exceeds the maximum allowable table row size of 8060 bytes”. Then I chose to design the table in such a way that a data record is close of the maximum size limit, so, following the model of the last two examples the code for the third example looks like this:

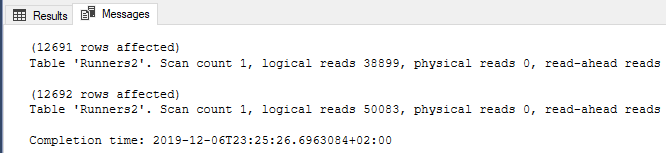
Example 3:

In the third example we do the same thing except now we will insert 50000 record in the table Runners2, in which each record is 8048 bytes.



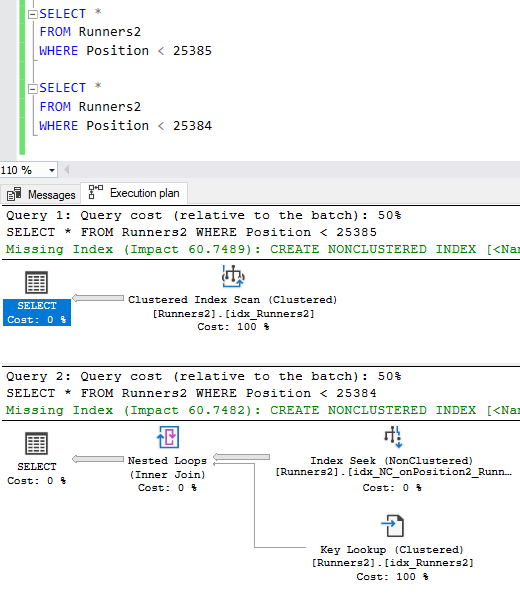
As it can be seen, the Tipping Point occurs when the estimated number of rows is 12693. Below, the execution plan can be seen along with the message returned by the I/O statistics after the selects where executed.

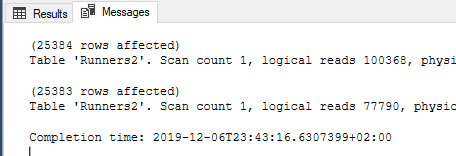




As before, the result is that the system starts to perform a scan when the estimated number of rows is 12693 or higher, from the total of 50000. With a quick calculus we find that in this case the Tipping Point occurred when at least 25.386% rows are estimated to match the query. Also we can say that the switch happens when the index seek and key lookup perform 77.6690% logical reads of the number of logical reads that are performed when the system chooses to scan the records.

After seeing this result I though that relevant information would surface if the table had more records, so, I deleted all the records from Runners2 and inserted 100000 this time. As the current number of record is double than the previous number it was not hard to find the new Tipping Point as it was around the double of the previous one:





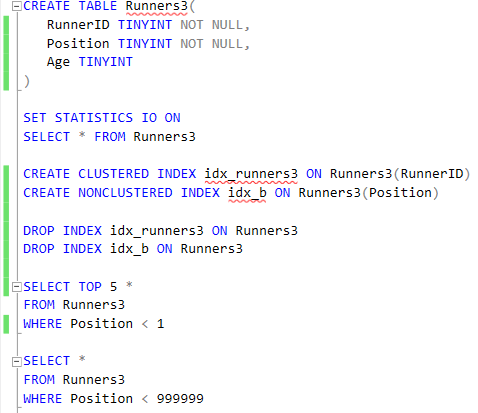
The results for the second part of this example are as it follows:

As before, the result is that the system starts to perform a scan when the estimated number of rows is 25383 or higher, from the total of 100000. With a quick calculus we find that in this case the Tipping Point occurred when at least 25.386% rows are estimated to match the query. Also we can say that the switch happens when the index seek and key lookup perform 77.5047% logical reads of the number of logical reads that are performed when the system chooses to scan the records.

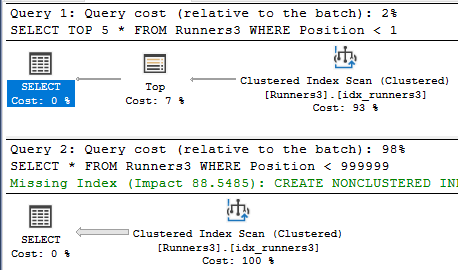
For the final example I decided to design a table that would be the opposite of the last one: a table with records of 3 bytes each.

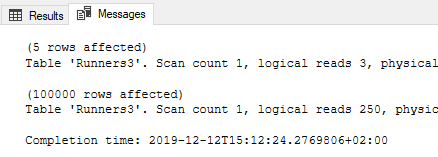
Example 4:

In the next example, table Runners3 has 100000 rows of 3 bytes each.



The results of this example are, as follows:





An unexpected conclusion from this is example is that in this case, the system uses an index scan no matter what the estimated number of rows is.

The table of results is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Example | Data record size (b) | Nr. of rows | TP occurred around logical reads | TP occurred around nr. of rows |
| 1 | 400 | 80000 | 81.44176% | 1.32875% |
| 2 | 100 | 100000 | 92.18865% | 0.406% |
| 3a | 8050 | 50000 | 77.6690% | 25.386% |
| 3b | 8050 | 100000 | 77.6690% | 25.386% |
| 4 | 3 | 100000 | - | - |

**4. Conclusion**

In conclusion, the occurrence of ‘’The Tipping Point’’ can not be predicted as it is based on more factors than just the estimated number of rows. In fact, from the table presented above we can see the percentage of the estimated number of rows when the ‘’Tipping Point’’ occurs fluctuates almost 25%, while the percentage of the logical reads in the same context fluctuates only 15%. On this grounds, the “Tipping Point” appears to corelate with the number of the logical reads needed to perform the query and retrieve de data than with the estimated number of rows.