**SQL SERVER – Index Seek vs. Index Scan – Diffefence and Usage – A Simple Note**

In this article we shall examine the two modes of data search and retrieval using indexes- index seeks and index scans, and the differences between the two.

Firstly, let us revisit indexes briefly. An index in a SQL Server database is analogous to the index at the start of a book. That is, its function is to allow you to quickly find the data you are searching for inside the book; in the case of a database, the “book” is a table.

An index scan means that SQL Server reads all rows in a table, and then returns only those rows that satisfy the search criteria. When an index scan is performed, all the rows in the leaf level of the index are scanned. This essentially means that all of the rows of the index are examined instead of the table directly. This is sometimes contrasted to a table scan, in which all the table data is read directly. However, there is usually little difference between an index scan and a table scan.

You may wonder why the Query Optimizer may choose to do an index or table scan. Surely it is much faster to first look up data using an index than to go through all the rows in a table? In fact, for small tables data retrieval via an index or table scan is faster than using the index itself for selection. This is because the added overhead of first reading the index, then reading the pages containing the rows returned by the index, does not offer any performance improvement for a table with only a few rows.

Other reasons to use an index scan would be when an index is not selective enough, and when a query will return a large percentage (greater than 50%) of rows from the table. In such cases the additional overhead of first using the index may result in a small degradation of performance.

An index seek, on the other hand, means that the Query Optimizer relies entirely on the index leaf data to locate rows satisfying the query condition. An index seek will be most beneficial in cases where a small percentage (less than 10 or 15%) of rows will be returned. An index seek will only affect the rows that satisfy a query condition and the pages that contain these qualifying rows; this is highly beneficial, in performance terms, when a table has a very large number of rows.

It is also worth noting that it is usually not worthwhile to create indexes on low-cardinality columns as they would rarely be used by the Query Optimizer. A low-cardinality column is one that contains a very small range of distinct values, for example a ‘Gender’ column would have only two distinct values- Male or Female. An example of a high-cardinality column is of course the primary key column, in which each value is distinct.

In summary, the Query Optimizer generally tries to perform an index seek. If this is not possible or beneficial (for example when the total number of rows is very small) then an index scan is used instead.

Scans and seeks are the iterators that SQL Server uses to read data from tables and indexes.  These iterators are among the most fundamental ones that we support.  They appear in nearly every query plan.

**What is the difference between a scan and a seek?**

A scan returns the entire table or index.  A seek efficiently returns rows from one or more ranges of an index based on a predicate.  For example, consider the following query:

select OrderDate from Orders where OrderKey = 2

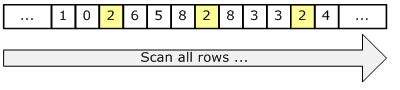
**Scan**

With a scan, we read each row in the orders table, evaluate the predicate “where OrderKey = 2” and, if the predicate is true (i.e., if the row qualifies), return the row.  In this case, we refer to the predicate as a “residual” predicate.  To maximize performance, whenever possible we evaluate the residual predicate in the scan.  However, if the predicate is too expensive, we may evaluate it in a separate filter iterator.  The residual predicate appears in text showplan with the WHERE keyword or in XML showplan with the <Predicate> tag.

Here is the text showplan (slightly edited for brevity) for this query using a scan:

  |--Table Scan(OBJECT:([ORDERS]), WHERE:([ORDERKEY]=(2)))

The following figure illustrates the scan:



Since a scan touches every row in the table whether or not it qualifies, the cost is proportional to the total number of rows in the table.  Thus, a scan is an efficient strategy if the table is small or if most of the rows qualify for the predicate.  However, if the table is large and if most of the rows do not qualify, we touch many more pages and rows and perform many more I/Os than is necessary.

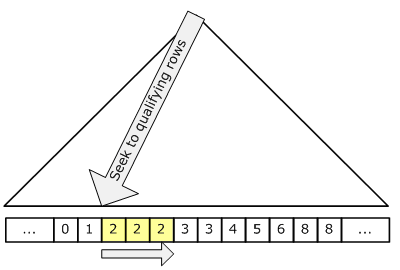
**Seek**

Going back to the example, if we have an index on OrderKey, a seek may be a better plan.  With a seek, we use the index to navigate directly to those rows that satisfy the predicate.  In this case, we refer to the predicate as a “seek” predicate.  In most cases, we do not need to re-evaluate the seek predicate as a residual predicate; the index ensures that the seek only returns rows that qualify.  The seek predicate appears in the text showplan with the SEEK keyword or in XML showplan with the <SeekPredicates> tag.

Here is the text showplan for the same query using a seek:

  |--Index Seek(OBJECT:([ORDERS].[OKEY\_IDX]), SEEK:([ORDERKEY]=(2)) ORDERED FORWARD)

The following figure illustrates the seek:



Since a seek only touches rows that qualify and pages that contain these qualifying rows, the cost is proportional to the number of qualifying rows and pages rather than to the total number of rows in the table.  Thus, a seek is generally a more efficient strategy if we have a highly selective seek predicate; that is, if we have a seek predicate that eliminates a large fraction of the table.

**A note about showplan**

In showplan, we distinguish between scans and seeks as well as between scans on heaps (an object with no index), clustered indexes, and non-clustered indexes.  The following table shows all of the valid combinations:

|  |  |  |
| --- | --- | --- |
|  | **Scan** | **Seek** |
| **Heap** | Table Scan |  |
| **Clustered Index** | Clustered Index Scan | Clustered Index Seek |
| **Non-clustered Index** | Index Scan | Index Seek |

**To be continued …**

There is much more to write about scans and seeks.  In my next post, I will continue by discussing bookmark lookup and how bookmark lookup relates to scans and seeks.

s