SQL SERVER Performance Tuning Notes

http://msbiskills.com/

What is Selectivity? How we can use this in creating our indexes?

Hi Guys Today I am going to talk about Selectivity in performance tuning and optimization. Selectivity refers to number of unique values in a column. Now if the column has high number of unique values then we should use that column as the first member while creating indexes. We will see this in a short span of time. If the column has high number of unique values then it means it has high selectivity. In layman terms we can use that column in where clause more as compared to the remaining columns.

Now most people say that if the column is highly selective then we should use it in the index as a first column in a multi column index. This is because histogram builds on the first column and the Meta data builds on the first column. It also helps when the B+ tree is internally created based on the unique values.

But there are scenarios where this is not the case. Let's go through an example where this rule does not work.

First let's create a table and insert some data.

```
CREATE TABLE TestIndexes
(
      ID INT
     ,NAME VARCHAR(5)
     ,GENDER VARCHAR(1)
)
GO

INSERT INTO TestIndexes
SELECT       Number ID
           , LEFT(NEWID(),5) NAME
```

```
           , CASE WHEN LEFT(RAND() * 10 + 1 ,1) * 5 % 2 = 0 THEN 'M' ELSE 'F'
END GENDER
FROM master..spt_values WHERE Number > 0 AND Number < 3000
UNION ALL
SELECT       Number + 1 ID
           , LEFT(NEWID(),5) NAME
           , CASE WHEN LEFT(RAND() * 10 + 1 ,1) * 5 % 2 = 0 THEN 'F' ELSE 'M'
END GENDER
FROM master..spt_values WHERE Number > 0 AND Number < 3000
```

Ok so the above query will create a new table for us and will insert 4496 records for us. Now let's say we write a query which will give us all males and ID is between 50 and 10000.

```
SELECT Gender,ID FROM TestIndexes
WHERE GENDER = 'M' AND ID > 50 AND ID < 10000
```

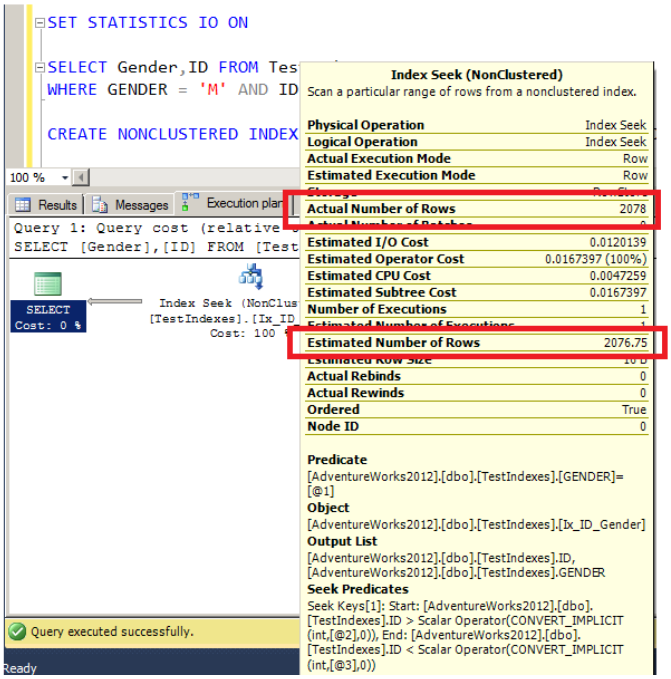
Now when you execute the above query you will get desired data with a table scan since we don't have any index.

Ok now let's create an index. For that we need to consider the selectivity. In this case since Id column has high selectivity as it has more unique values then the gender column which has only 2 values 'M' and 'F'.

Let's go ahead and create that index.

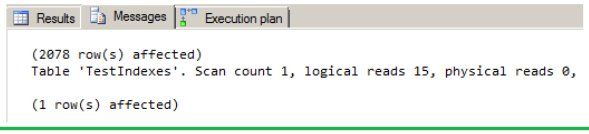
```
CREATE NONCLUSTERED INDEX Ix_ID_Gender ON TestIndexes(ID,GENDER)
```

Note – Please note that the above index is a multi-column NonClustered index and we are using ID as a first column. Now execute the query again and check the execution plan and Statistics. [Add actual execution plan and Statistics IO ON]



Here we are doing an index seek which is good and our stats are also good as actual and estimated number rows are almost same. And we are reading 15 pages from the disk.

Here we are reading 15 pages to satisfy this query, One logical read is 8KB page.



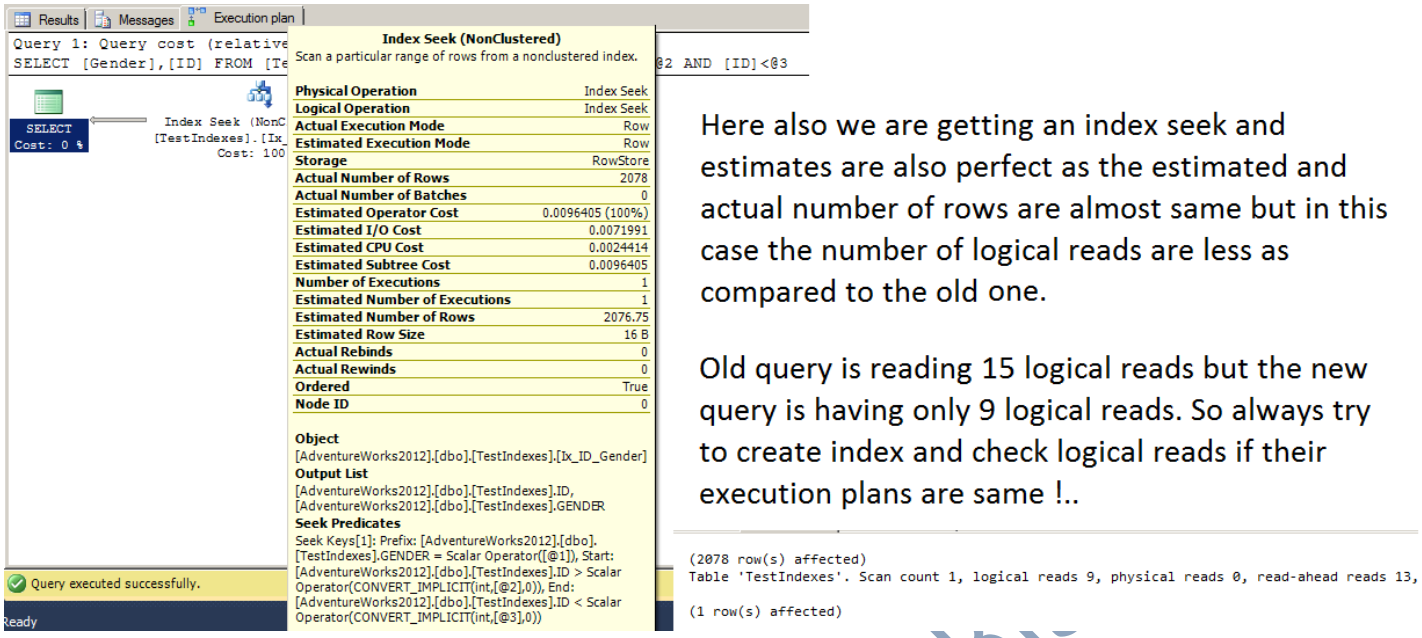
So we are doing an index seek which is good and our stats are also good as estimated and actual number of rows is also same. In this case we are reading 15 logical pages.

Now let's drop this newly created index and create the same index with Gender as the first column and ID as the second column.

```
DROP INDEX Ix_ID_Gender ON TestIndexes
GO

CREATE NONCLUSTERED INDEX Ix_ID_Gender ON TestIndexes(GENDER,ID)
GO
```

Okay so our new index is in place let's executing our query again and check the execution plan and statistics.



Here also we are getting an index seek and estimates are also perfect as the estimated and actual number of rows are almost same but in this case the number of logical reads are less as compared to the old one.

Old query is reading 15 logical reads but the new query is having only 9 logical reads. So always try to create index and check logical reads if their execution plans are same !..

So it's clearly visible that the new query and the old query's execution plans are same. But their statistics are different. Query with the old index is having 15 logical reads and the new query is having only 9 logical reads.

Summary

So all in all we can say that if the column is highly selective then we should use it in the index as a first column but this is not the case always. So always create the index and try out different options against the number of queries and then finalize the index.

That's all folks; I hope you've enjoyed learning about selectivity and how we can effectively use it while designing our indexes, and I'll see you soon with more "Performance Tuning" articles.

Thanks!

Pawan Kumar Khowal

MSBISKills.com