SQL SERVER Performance Tuning Notes

http://msbiskills.com/

Why Order By is bad in Queries? / Execution/Query Plan Operator – The Sort Operator

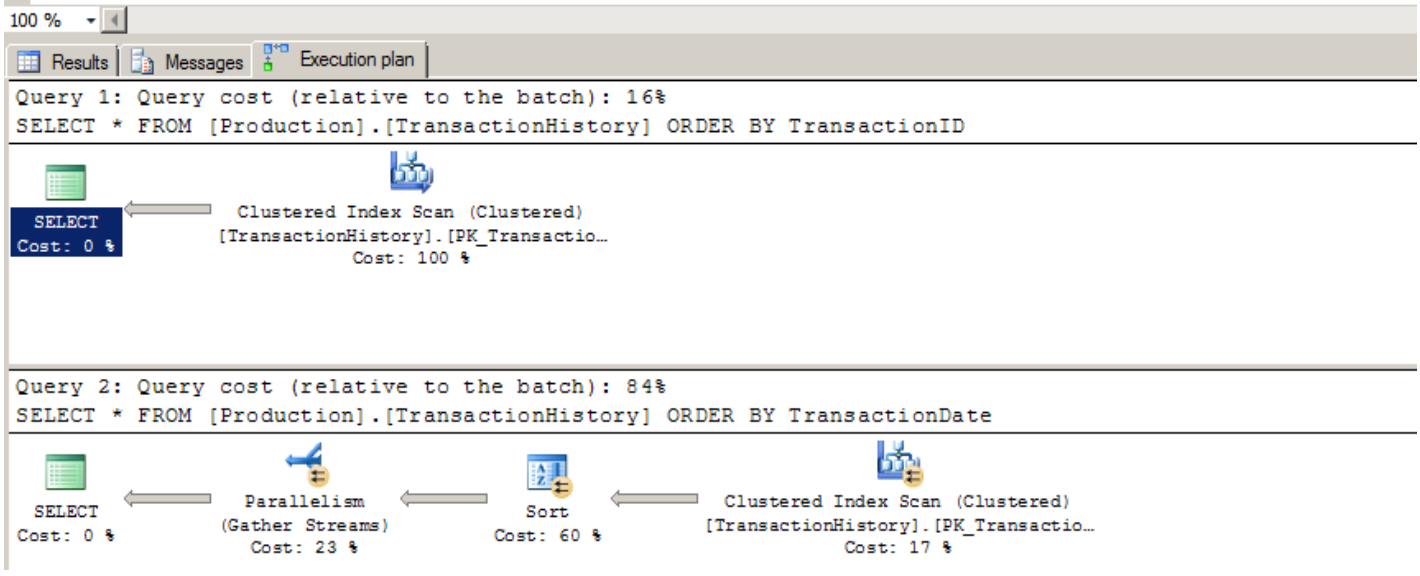
Order By is not always a bad thing in the query/execution plans.

Internally Order by is implemented by Sort operator or by an index.

For example, In the below examples, in first query we are having an clustered index scan and in the second one we are using a sort operator as we don't have any index on TransactionDate column.


```
SELECT * FROM [Production].[TransactionHistory]
ORDER BY TransactionID

SELECT * FROM [Production].[TransactionHistory]
ORDER BY TransactionDate
```



The SORT operator sorts all rows received by the operator into order. Well in some cases, the SORT operation is performed in TempDB. This is because we don't have proper indexes in place or we didn't pass proper information to the query optimizer.

Also note that TempDB is used for all databases within the SQL Server Instance, this can lead to TempDB becoming a bottle-neck and thereby affecting performance.

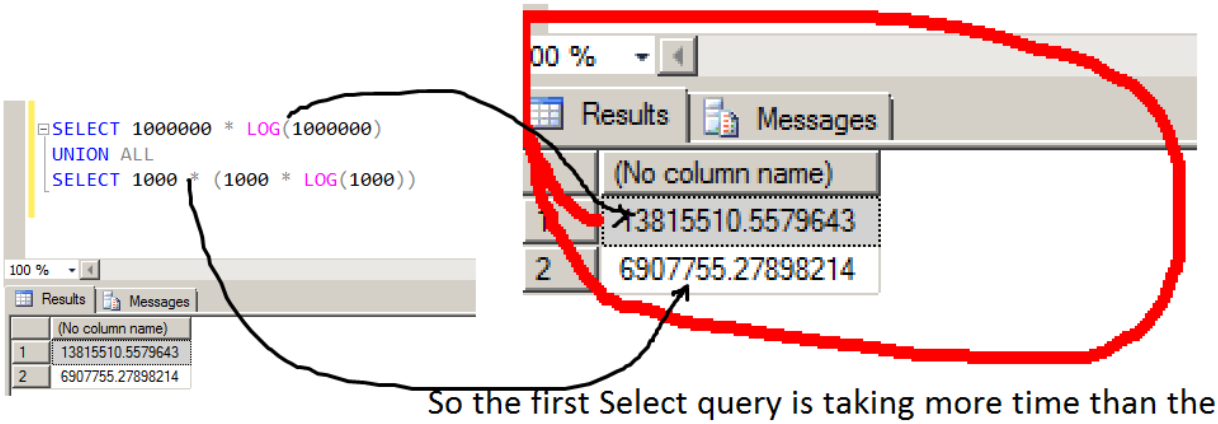
Basically sort is required by many operators like Order By, Merge Join, Stream Aggregate and Window functions. Let's say you wanted the data in a particular order so its need to be sorted. This sort operator is not scaled well. It scales extra linearly. So Sorting 10000 rows is more than 10 time slower than sorting 1000 rows. Sort is very important and on the other hand it could degrade your queries also if you lots and lots of sort operator in your queries execution plan.

Always check you requirement whether you really need order by in your query or not. Lot of developer just put order by in their queries just to see the ordered output. Remember this is not free. There is cost involved for sort operator. This can really degrade your queries performance because sorting data at SQL Server is lot more expensive than at some other places like app tier.

Also **note that doing smaller sorts is always better than a very large sort.** Very large here refers to huge volume of data.

Algorithmic cost of sort is **O (N*Log (N))** - N here is the number of input rows.

So mathematically we can prove that doing multiple mini sorts is much better than performing a sort on a large dataset. Example below-



So the first Select query is taking more time than the second query. So mathematically we can say that the sorting small sets of data rather than one big sort.

Example of Sort on smaller set of data

```
SELECT p.ProductID,tr.SalesOrderID
FROM [Production].[Product] p
CROSS APPLY
    (
      SELECT TOP 1 SalesOrderID
      FROM [Sales].[SalesOrderDetail] s
      WHERE s.ProductID = p.ProductID
       ORDER BY SalesOrderID DESC
    ) tr
```

What we are doing in the above query is we are reading all product ids from products master table and then for each product id we are hitting orders table to find out the latest salesorderid which works perfectly in our case since we need only 1 salesorderid per product that too we are getting with the help of nonclustered index seek. Also check the cost each query took.

The quickest and easiest way to avoid a SORT operator is by creating an Index. As we know, indexes are ordered by the columns so that, if you create an index covering your query, the Query Optimizer identifies this index and uses it to avoid a SORT operation.

Summary

This is the small information about Sort Operator. I will write another port about sort operator where we will dig more into the topic like how internally sort functions, about methods like Quick sort and merge sort, how sort spills data into tempDB and disks, etc.

That's all folks; I hope you've enjoyed learning about Sort Operator, and I'll see you soon with more "Performance Tuning" articles.

Thanks!

Pawan Kumar Khowal

MSBISKills.com