SQL SERVER Performance Tuning Notes

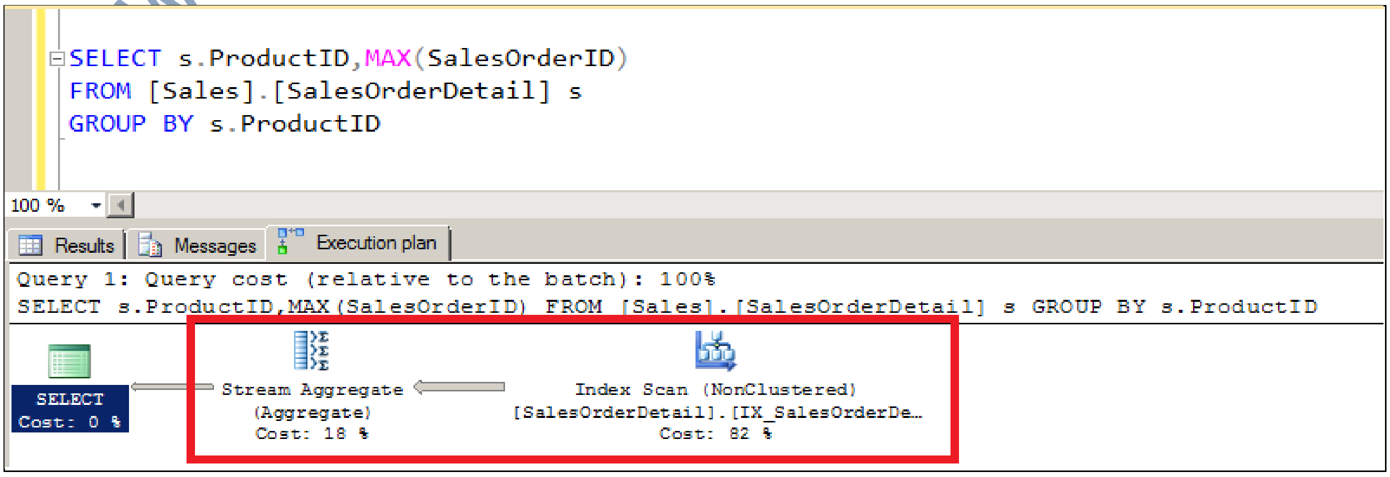
http://msbiskills.com/

How to find latest order Id from Orders table for each product? (Top n orders for each product) Let's write a better query.

Let's say I wanted to find out the maximum sales order id for each project. Connect to Adventure works 2012 database. Now for that let's say we wrote below query.

```
SELECT s.ProductID,MAX(SalesOrderID)
FROM [Sales].[SalesOrderDetail] s
GROUP BY s.ProductID
```

Well the above query works fine and returns 266 records. Now let's examine the execution plan. Here we have a nonclustered index scan and a stream Aggregate. The Stream Aggregate is used to group some rows by one or more columns, and to calculate any aggregation expressions that are specified in the query. The common types of aggregation are: SUM, COUNT, AGV, MIN, and MAX.



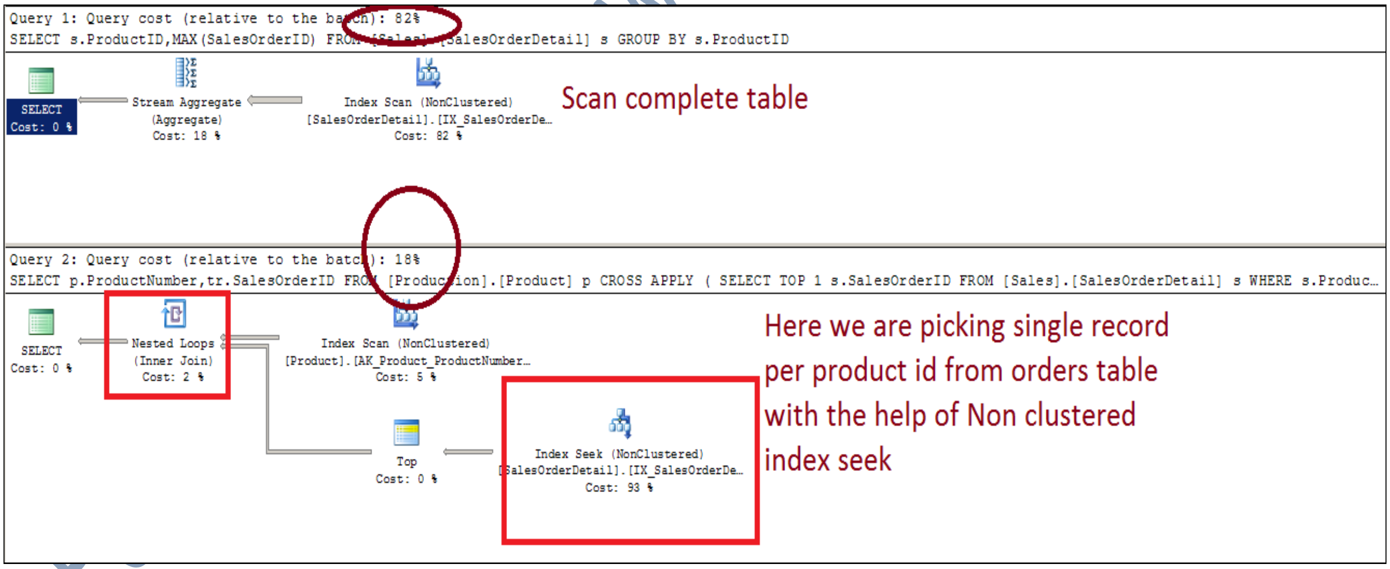
```
SELECT s.ProductID,MAX(SalesOrderID)
FROM [Sales].[SalesOrderDetail] s
GROUP BY s.ProductID
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT s.ProductID,MAX(SalesOrderID) FROM [Sales].[SalesOrderDetail] s GROUP BY s.ProductID
```

Now problem with approach is that we are reading all the pages from leaf level and if we have million or billion row this approach will slow down the performance of the query. A more appropriate and better approach is given below.

```
SELECT p.ProductID,tr.SalesOrderID
FROM [Production].[Product] p
CROSS APPLY
     (
       SELECT MAX(s.SalesOrderID) SalesOrderID
       FROM [Sales].[SalesOrderDetail] s
       WHERE s.ProductID = p.ProductID
     ) tr
```

Now let's put both the queries in the query window and execute. Let's compare the execution plan and the cost taken by each query.



```
Query 1: Query cost (relative to the batch): 82%
SELECT s.ProductID,MAX(SalesOrderID) FROM [Sales].[SalesOrderDetail] s GROUP BY s.ProductID
```

```
Query 2: Query cost (relative to the batch): 18%
SELECT p.ProductNumber,tr.SalesOrderID FROM [Production].[Product] p CROSS APPLY ( SELECT TOP 1 s.SalesOrderID FROM [Sales].[SalesOrderDetail] s WHERE s.Produc…
```

SELECT Cost: 0 %
Stream Aggregate (Aggregate) Cost: 18 %
Index Scan (NonClustered) [SalesOrderDetail].[IX_SalesOrderDe… Cost: 82 %
Scan complete table
SELECT Cost: 0 %
Nested Loops (Inner Join) Cost: 2 %
Index Scan (NonClustered) [Product].[AK_Product_ProductNumber… Cost: 5 %
Top Cost: 0 %
Index Seek (NonClustered) [SalesOrderDetail].[IX_SalesOrderDe… Cost: 93 %
Here we are picking single record per product id from orders table with the help of Non clustered index seek

What we are doing in the second 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 first query is taking 82% of the most and the second query is taking 18% cost only. Thus it is clearly evident that second query excels over first one.

Summary

Always execute your query and check what is there in the execution plan and see if you can tweak the query to get better performance. Apply operator is one of the most efficient tool to improve your queries.

When we are checking things for performance always try to find out these small things. These can help you queries performance greatly. We are reducing I/O by reading less data pages.

That's all folks; I hope you've enjoyed learning about how we can write better that will perform better, and I'll see you soon with more "Performance Tuning" articles.

Thanks!

Pawan Kumar Khowal

MSBISKills.com