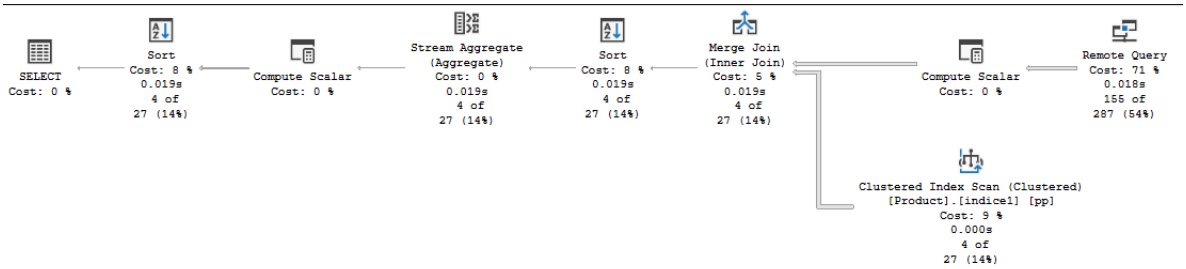
Consulta 1: Productos más vendidos con un precio menor o igual a 100 y de color amarillo

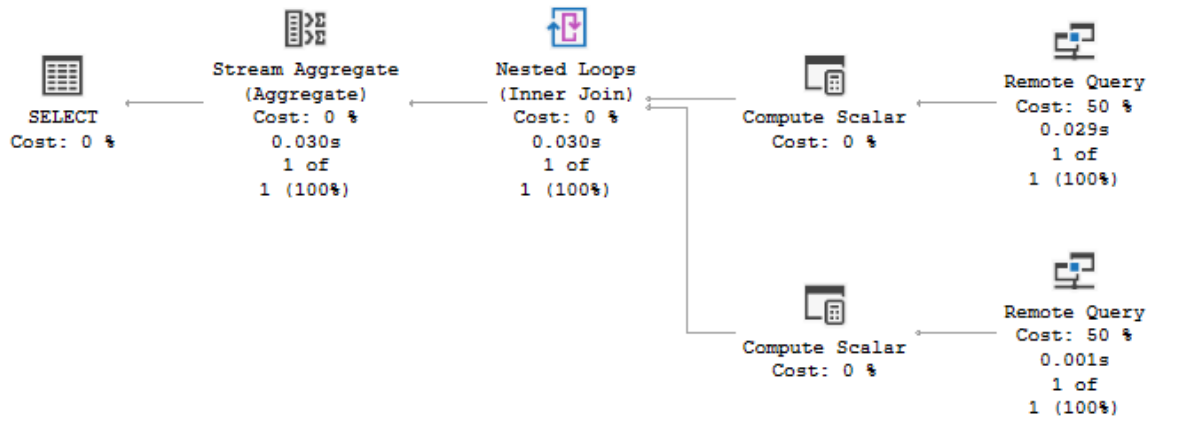
```
select pp.ProductID, pp.Name,pp.ProductNumber, pp.ListPrice,
count(ss.ProductID) as veces_vendido
from Adventure.Production.Product pp
inner join [LINKSERVER2].AdventureSales.Sales.SalesOrderDetail ss
on pp.ProductID = ss.ProductID
where pp.ListPrice <=100 and pp.Color = 'Yellow'
group by ss.ProductID, pp.Name, pp.ProductID,pp.ProductNumber, pp.ListPrice
order by count(ss.ProductID) desc;
```



En la primera consulta creamos un índice agrupado para el id del producto en ambas bases de datos que lo ocupan ya que antes de este índice la consulta no estaba optimizada una vez creado notamos un uso menos de CPU y disco al momento de ejecutar la consulta

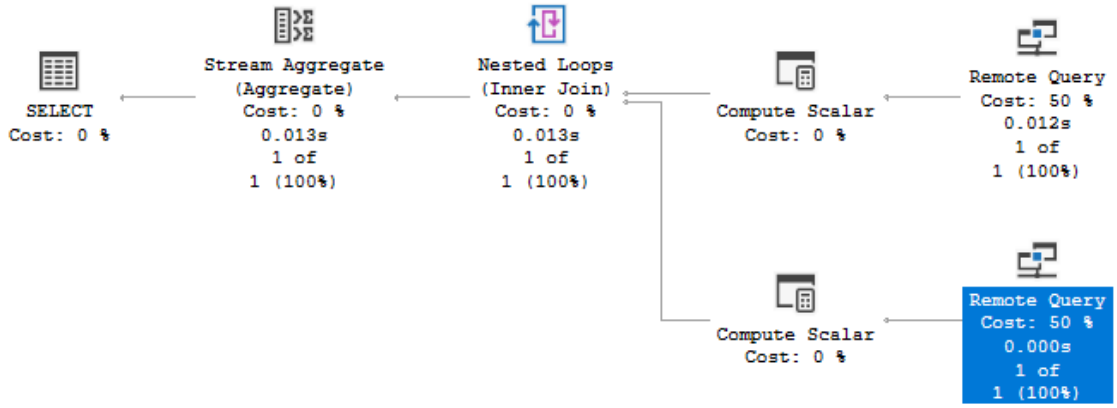
Consulta 2: Determinar el número de ventas en un territorio especifico

```
select st.Name, sum(so.OrderQty * so.UnitPrice) as total_vendido
from [LINKSERVER2].AdventureSales.Sales.SalesOrderHeader sh
inner join [LINKSERVER2].AdventureSales.Sales.SalesOrderDetail so
on sh.SalesOrderID = so.SalesOrderID
inner join [LINKSERVER2].AdventureSales.Sales.SalesTerritory st
on st.TerritoryID = sh.TerritoryID
where st.TerritoryID = 3
group by st.TerritoryID, st.Name;
```



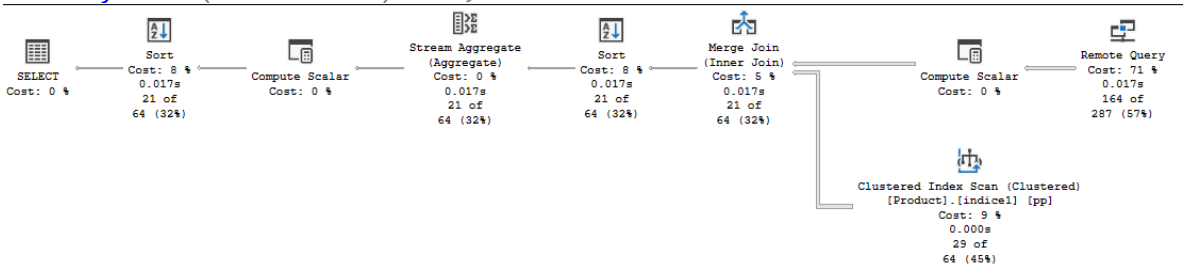
Esta consulta ocupa la misma sentencia de comparar los ID de cada producto, por lo que esta optimizada al crear los índices para la consulta anterior, sin embargo, necesita los índices para

referirse al territorio, al momento de crearlos la consulta queda optimizada y se refleja en los tiempos de ejecución



Consulta 3: Listar los productos más vendidos que tengan un precio mayor o igual a 600 y un nivel de cuidado del stock igual a 500

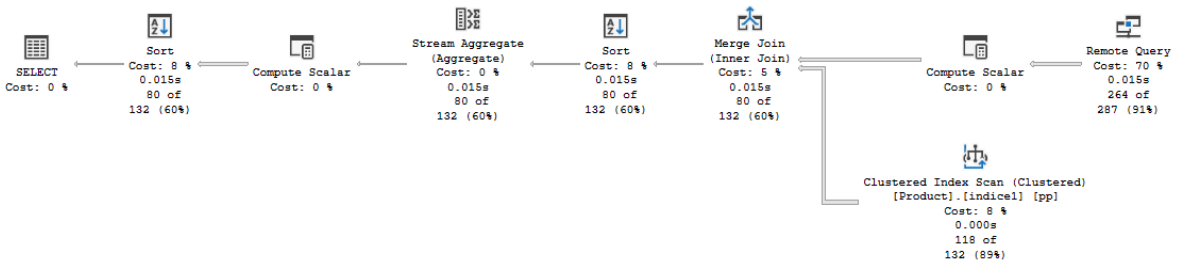
```
select pp.ProductID, pp.Name, count(ss.ProductID) as veces_vendido
from Production.Product pp
inner join [LINKSERVER2].AdventureSales.Sales.SalesOrderDetail ss
on pp.ProductID = ss.ProductID
where pp.ListPrice >= 600 and pp.SafetyStockLevel = 500
group by ss.ProductID, pp.Name, pp.ProductID
order by count(ss.ProductID) desc;
```



Esta consulta esta optimizada ya que ocupa los índices creados anteriormente en el ID del producto

Consulta 4: Listar los productos más vendidos que tengan un precio estándar menor o igual a 250 y el día de manufactura sea igual a 1

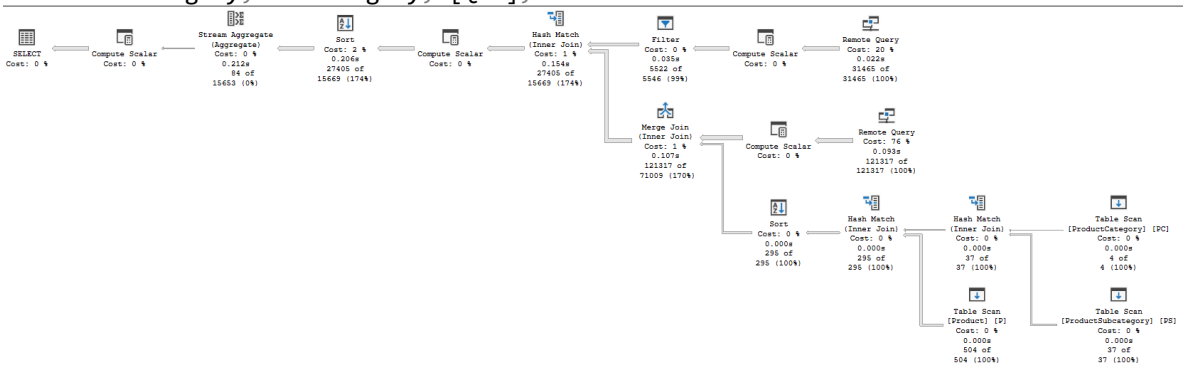
```
select pp.ProductID, pp.Name, count(ss.ProductID) as veces_vendido
from Production.Product pp
inner join [LINKSERVER2].AdventureSales.Sales.SalesOrderDetail ss
on pp.ProductID = ss.ProductID
where pp.StandardCost <= 250 and pp.DaysToManufacture = 1
group by ss.ProductID, pp.Name, pp.ProductID
order by count(ss.ProductID) asc;
```



Al igual que las anteriores consultas esta optimizada gracias a los índices creados.

Consulta 5: listar por categoría los productos y su cantidad de ventas en el año de 2011 a 2012

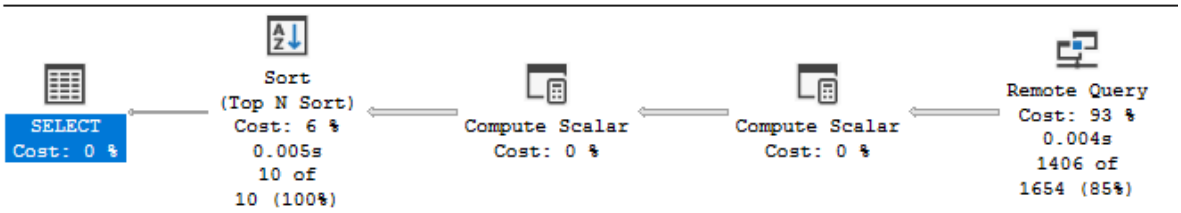
```
SELECT PC.Name AS Category, PS.Name AS Subcategory,
DATEPART(yy, SOH.OrderDate) AS [Year]
, 'Q' + DATENAME(qq, SOH.OrderDate) AS [Qtr]
, STR(SUM(DET.UnitPrice * DET.OrderQty)) AS [$ Sales]
FROM Production.ProductSubcategory AS PS
INNER JOIN [LINKSERVER2].AdventureSales.Sales.SalesOrderHeader AS SOH
INNER JOIN [LINKSERVER2].AdventureSales.Sales.SalesOrderDetail DET ON
SOH.SalesOrderID = DET.SalesOrderID
INNER JOIN Production.Product P ON DET.ProductID = P.ProductID
ON PS.ProductSubcategoryID = P.ProductSubcategoryID
INNER JOIN Production.ProductCategory PC ON PS.ProductCategoryID =
PC.ProductCategoryID
WHERE YEAR(SOH.OrderDate) = '2012' or YEAR(SOH.OrderDate) ='2011'
GROUP BY DATEPART(yy, SOH.OrderDate), PC.Name, PS.Name, 'Q'
+ DATENAME(qq, SOH.OrderDate), PS.ProductSubcategoryID
ORDER BY Category, SubCategory, [Qtr];
```



Al optimizar esta consulta, se noto un cambio inmediatamente pues el tiempo de ejecución se redujo considerablemente, al crear los índices en cada relación del join correspondiente.

Consulta 6: listar los consumidores que han hecho ordenes durante el año de 2011

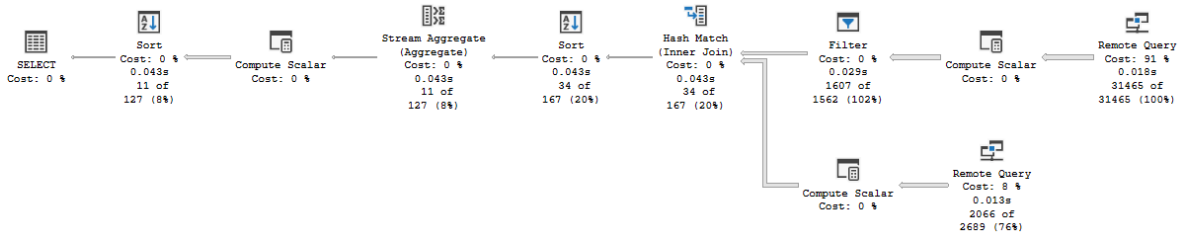
```
SELECT Top 10 CustomerID, STR(SUM(TaxAmt)) AS Totals
FROM [LINKSERVER2].AdventureSales.Sales.SalesOrderHeader
WHERE [OrderDate] BETWEEN '1/1/2011' AND '12/31/2011'
GROUP BY CustomerID
ORDER BY Totals DESC
```



En esta consulta la optimización no fue algo considerable ya que no requeria de mas de una tabla para realizarle, y solo fueron unos pocos segundos de ejecución y de consumo menor de cpu.

Consulta 7: listar el precio promedio de los productos que hayan tenido una cantidad mayor a 10 órdenes en el 2011

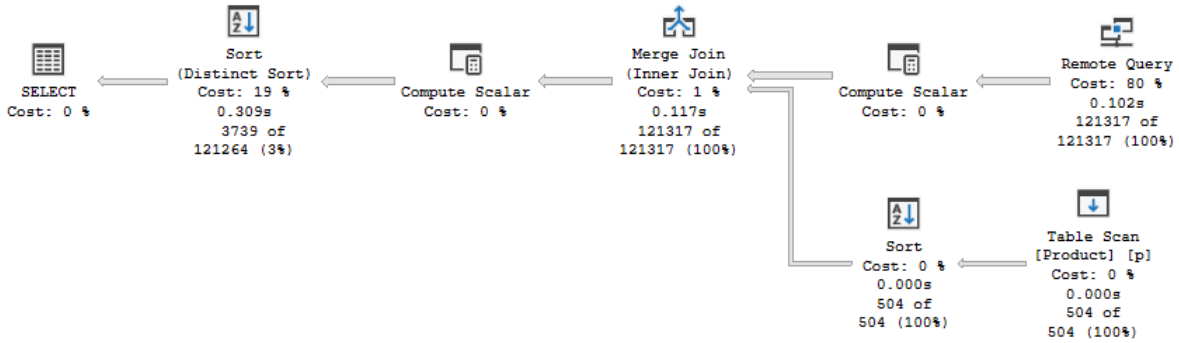
```
SELECT sod.ProductID, AVG(sod.UnitPrice) AS [Average Price]
FROM [LINKSERVER2].AdventureSales.Sales.SalesOrderDetail as sod
INNER JOIN [LINKSERVER2].AdventureSales.Sales.SalesOrderHeader as soh
ON soh.SalesOrderID = sod.SalesOrderID
WHERE OrderQty > 10 AND YEAR(soh.OrderDate) = '2011'
GROUP BY ProductID
ORDER BY [Average Price] DESC
```



Esta consulta no tuvo repercusión al momento de crear los índices por lo que no se pudo optimizar mas.

Consulta 8: listar los descuentos que han tenido los productos

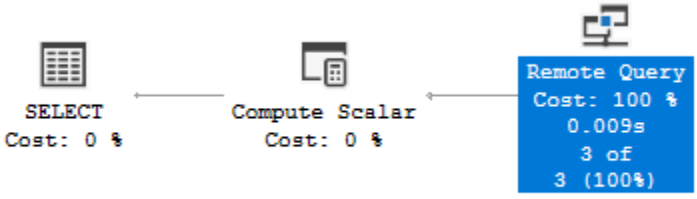
```
SELECT Distinct p.Name AS ProductName,
NonDiscountSales = (OrderQty * UnitPrice),
Discounts = ((OrderQty * UnitPrice) * UnitPriceDiscount)
FROM Production.Product AS p
INNER JOIN [LINKSERVER2].AdventureSales.Sales.SalesOrderDetail AS sod ON p.ProductID
= sod.ProductID
ORDER BY ProductName DESC;
```



Esta consulta no tuvo un cambio aparente con la optimización en los índices creados.

Consulta 9: listar el número de orden, fecha, vendedor y cliente que ha hecho el cliente con el numero 11000

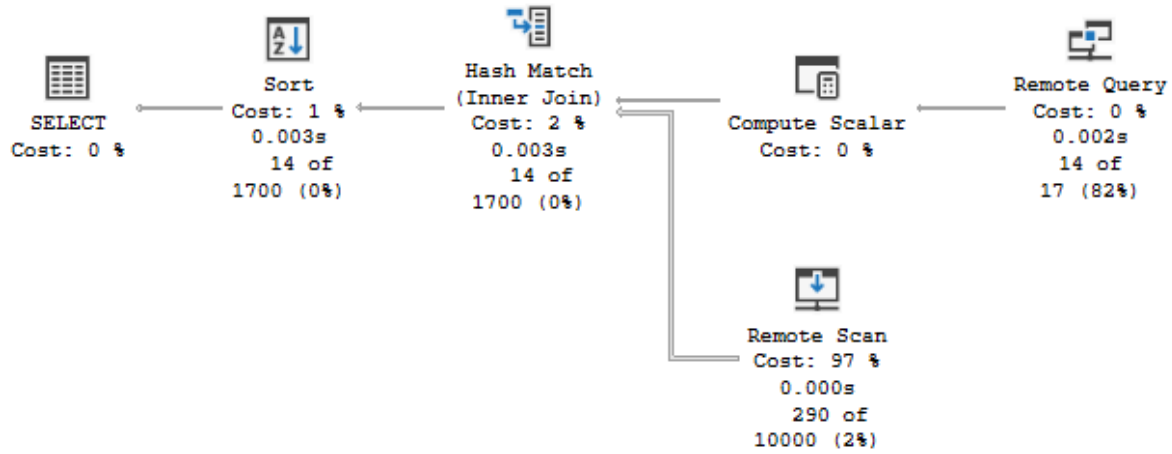
```
SELECT
H.SalesOrderID AS 'N° Orden',
H.OrderDate AS Fecha,
H.CustomerID AS Cliente,
H.SalesPersonID AS Vendedor,
SUM(D.OrderQty+D.UnitPrice) AS [Importe Vendido]
FROM [LINKSERVER2].AdventureSales.Sales.SalesOrderHeader H
INNER JOIN [LINKSERVER2].AdventureSales.Sales.SalesOrderDetail D
ON H.SalesOrderID = D.SalesOrderID
AND CustomerID = 11000
GROUP BY H.SalesOrderID, H.OrderDate, H.CustomerID, H.SalesPersonID
```



Esta consulta si resulto mas eficiente al momento de ejecutarla dando un corto tiempo de ejecución.

Consulta 10: listar por territorio la cantidad de ventas, cuotas, comisiones, despidos, estado marital de los empleados que han trabajado

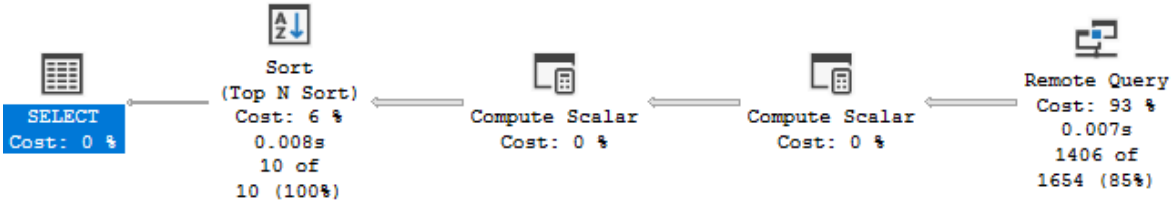
```
SELECT t.TerritoryID,t.Name AS Territory, s.SalesLastYear as [Emp Sales Last Year]
, SalesQuota AS [Emp Sales Quota], s.SalesYTD AS [Emp Sales YTD]
, Bonus AS    [Emp Bonus], CommissionPct as [Emp Commission%]
, HireDate, MaritalStatus, t.SalesLastYear AS [Territory Sales Last Year]
, t.SalesYTD AS [Territory Sales YTD]
from OPENQUERY([LINKSERVER3], 'SELECT * FROM
[AdventurePerson].[HumanResources].Employee') as e
inner join [LINKSERVER2].AdventureSales.Sales.SalesPerson as s on s.BusinessEntityID
= e.BusinessEntityID
inner join [LINKSERVER2].AdventureSales.Sales.SalesTerritory as t on t.[TerritoryID]
= s.TerritoryID
ORDER BY TerritoryID
```

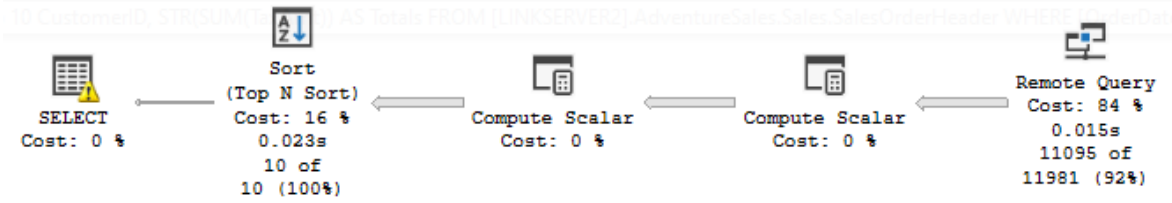


Esta consulta no tuvo un cambio significativo con la optimización.

Consulta 11: listar el top 10 de los clientes con más compras en el año del 2013

```
SELECT Top 10 CustomerID, STR(SUM(TaxAmt)) AS Totals
FROM [LINKSERVER2].AdventureSales.Sales.SalesOrderHeader
WHERE [OrderDate] BETWEEN '1/1/2011' AND '12/31/2011'
GROUP BY CustomerID
ORDER BY Totals DESC
```

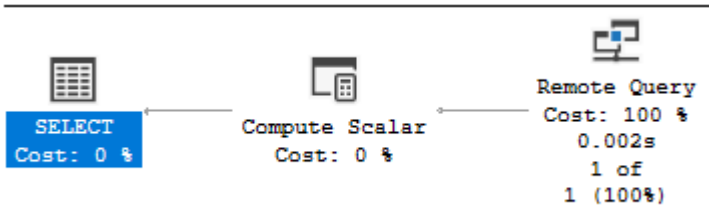




En esta consulta en particular fue que nos encontramos con una advertencia del uso de memoria por parte de la ejecución, en el momento de optimizarla esta advertencia desapareció y el tiempo de ejecución se redujo a la mitad.

Consulta 12: listar la cantidad total de compras del cliente 11000

```
SELECT CustomerID, SUM([TotalDue]) AS Totals
FROM [LINKSERVER2].AdventureSales.Sales.SalesOrderHeader
where CustomerID = 11000
GROUP BY CustomerID
ORDER BY Totals DESC
```



Esta consulta no tuvo un cambio significativo a la optimización por lo corta que es en realidad