مثال 3 طرح query

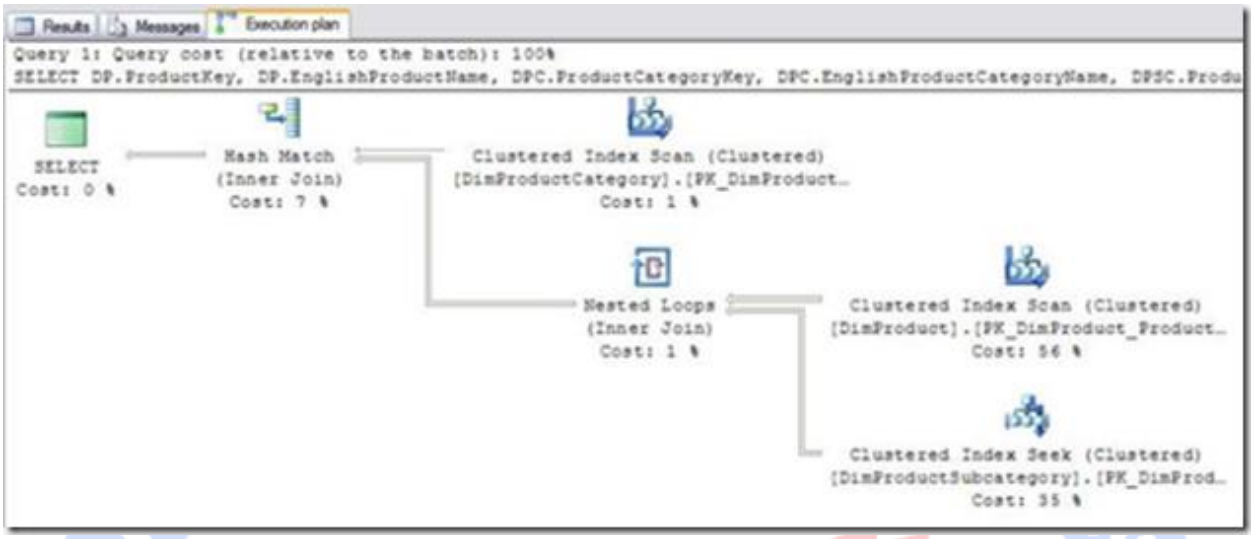
در این مثال یک query پیچیده تر ایجاد خواهیم کرد.

توضیحات

سعی می کنیم برخی داده ها را در مورد تولیدات و مولفه های تولید ردیف کنیم. می توانیم از ProductKey (کلید تولید) و Product Subcategory Key(کلید زیرمجموعه ی تولید) برای اتصال داده ها از جدول های مختلف استفاده کنیم. بنابراین query مانند زیر به نظر خواهد رسید.

```
SELECT DP.ProductKey,
 DP.EnglishProductName,
 DPC.ProductCategoryKey,
 DPC.EnglishProductCategoryName,
 DPSC.ProductSubcategoryKey,
 DPSC.EnglishProductSubcategoryName
FROM AdventureWorksDW..DimProduct DP
 INNER JOIN AdventureWorksDW..DimProductSubcategory DPSC
 ON DP.ProductSubcategoryKey=DPSC.ProductSubcategoryKey
 INNER JOIN AdventureWorksDW..DimProductCategory DPC
 ON DPSC.ProductCategoryKey=DPC.ProductCategoryKey
```

یک پنجره ی جدید باز کرده و CTRL+M را فشار دهید تا Actual Execution Plan را وارد کنید وسپس کد بالا را اجرا کنید.
تصویر زیر نشان دهنده ی طرح این query می باشد. لطفا به اتصالات در اجرای درخت دقت داشته باشید.



در تصویر دو اپراتور Clustered Index Scan ، یک Clustered Index Seek، یک Nested Loops، یک Hash Match و یک Result مشاهده کنید.

SQL Server در واقع query را به این روش پردازش می کند:

Clustered Index Scan روی جدول DimProduct (هزینه 56 درصد).

Clustered Index Seekرو جدول DimProductSubcategory (هزینه 35 درصد).

Nested Loops برای اتصال ورودی های بالا.

Clustered Index Scanروی جدول DimProductCategory (هزینه یک درصد)

Hash Match برای اتصال ورودی های بالا (هزینه 7 درصد)

انتخاب برای بازگشت نتیجه ی مجموعه (هزینه 0 درصد)

با مشاهده ی هزینه ی اپراتورها متوجه خواهید شد که Clustered Index Scan روی جدول DimProduct خیلی پر هزینه است و از 56 درصد کل منابع استفاده می کند. اگر به یاد داشته باشید، عملکرد مشابهی نیز در مثال قبلی اجرا شد. ما همچنین یک Clustered Index Seek روی جدول DimProductsSubcategory داریم که دارای هزینه ی بالایی است (35 درصد)، که ToolTip مربوط به آن را در تصویر زیر مشاهده می کنید.

Clustered Index Seek (Clustered)

Scanning a particular range of rows from a clustered index.

Physical Operation	Clustered Index Seek
Logical Operation	Clustered Index Seek
Actual Number of Rows	397
Estimated I/O Cost	0,003125
Estimated CPU Cost	0,0001581
Estimated Number of Executions	605,9997
Number of Executions	606
Estimated Operator Cost	0,0989336 (35%)
Estimated Subtree Cost	0,0989336
Estimated Number of Rows	1
Estimated Row Size	69 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	4

Object

[AdventureWorksDW].[dbo].[DimProductSubcategory].[PK_DimProductSubcategory_ProductSubcategoryKey] [DPSC]

Output List

[AdventureWorksDW].[dbo].[DimProductSubcategory].ProductSubcategoryKey; [AdventureWorksDW].[dbo].[DimProductSubcategory].EnglishProductSubcategoryName; [AdventureWorksDW].[dbo].[DimProductSubcategory].ProductCategoryKey

Seek Predicates

Seek Keys[1]: Prefix: [AdventureWorksDW].[dbo].[DimProductSubcategory].ProductSubcategoryKey = Scalar Operator([AdventureWorksDW].[dbo].[DimProduct].[ProductSubcategoryKey] as [DP].[ProductSubcategoryKey])

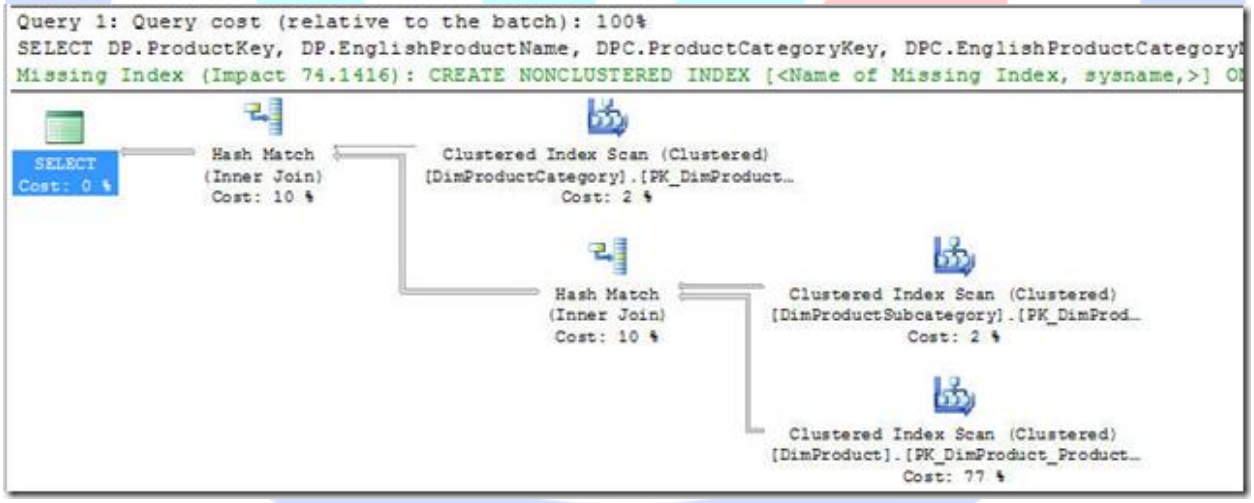
با نگاه دقیق تر به عملکرد Clustered Index Seek مشخص می شود که این عملکرد 397 ردیف گزارش داده و 606 آیتم را اجرا می کند، به همین دلیل است که این عملکرد چنین درصد بالایی از منابع را استفاده می کند. این امر توسط اتصال Nested Loopsنیز ذکرمی شود. ابتدا query داده را از جدول DimProduct می گیرد و سپس ردیف متناظر را از DimProductSubcategory می آورد، به همین دلیل 606 بار اجرا شد. احتمالا این سوال برای شما مطرح می شود که اگر 606 بار آیتم ها را اجرا کرده، چرا فقط 397 ردف را گزارش داده است؟ به این دلیل که 606 ردیف از جدول DimProductدر عملکرد Clustered Index Scan بازگشته اند، اما فقط 397 عدد از آنها دارای ProductSubcategoryKeyمی باشند که در اتصال داخلی (INNER JOIN) روی جدول DimProductSubcategory استفاده شد.

چگونه می توانیم اجرا را بهبود ببخشیم؟

به روشی که query در حال حاضر نوشته شده است، کار زیادی نمی توان انجام داد، زیرا همه ی داده ها از جدول DimProduct گزارش شده اند. تنها کاری که می توانیم انجام دهیم، فیلتر کردن رکوردهای NULL SubcategoryKey می باشد، مانند زیر:

```
SELECT DP.ProductKey,
 DP.EnglishProductName,
 DPC.ProductCategoryKey,
 DPC.EnglishProductCategoryName,
 DPSC.ProductSubcategoryKey,
 DPSC.EnglishProductSubcategoryName
FROM AdventureWorksDW..DimProduct DP
 INNER JOIN AdventureWorksDW..DimProductSubcategory DPSC
ON DP.ProductSubcategoryKey=DPSC.ProductSubcategoryKey
 INNER JOIN AdventureWorksDW..DimProductCategory DPC
ON DPSC.ProductCategoryKey=DPC.ProductCategoryKey
WHERE
 DP.ProductSubcategoryKey IS NOT NULL
```

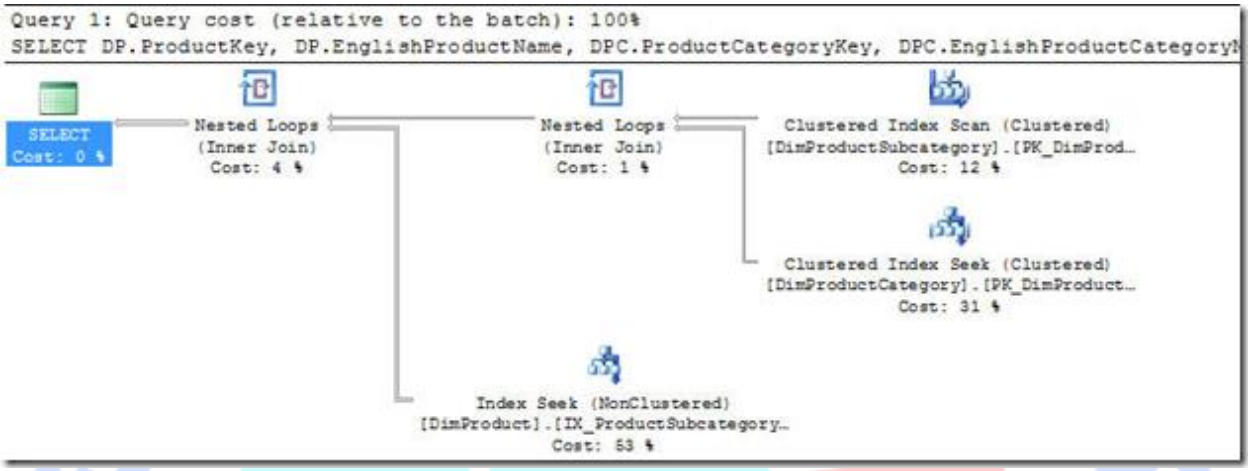
اگر این کد را اجرا کنیم و به طرح query دقت کنیم، مشاهده می کنیم که یک طرح متفاوت داریم. همچنین متوجه خواهید شد که یک نشان ایندکس گم شده هم داریم.



این نشان گم شده مانند زیر ایجاد شده است:

```
CREATE NONCLUSTERED INDEX [IX_DimProduct_ProductSubcategoryKey]
ON [dbo].[DimProduct] ([ProductSubcategoryKey])
INCLUDE ([ProductKey],[EnglishProductName])
```

سپس اگر query را مجددا اجرا کرده و به طرح query دقت کنیم، تغییرات مجدد را مشاهده خواهیم کرد:



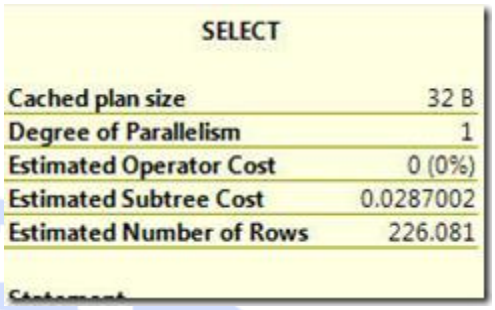
آیا برنامه بهتر می شود یا نه؟

یک راه برای چک کردن این امر، مشاهده ی هزینه ی کل اجرای query می باشد. اگر برای اکثر عملکردها و اجرای هر query دارای TollTip باشیم، می توانیم هزینه ی کل query را به دست آوریم.

در زیر اجرای سه query را در ترتیب اجرای درخت نمایش داده شدهاست. مشاهده می کنید که در زمان بهبود بخشیدن به Query Estimated Subtree Cost (هزینه ی برآورد شده ی زیرمجموعه) به این ترتیب می باشد: 0.02 -> 0.20 -> 0.28. بنابراین این تغییرات باعث بهبود query می شود.

SELECT	
Cached plan size	48 B
Degree of Parallelism	1
Memory Grant	1024
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	0.282053
Estimated Number of Rows	226.081

SELECT	
Cached plan size	64 B
Degree of Parallelism	1
Memory Grant	1536
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	0.205318
Estimated Number of Rows	226.081



آدرس آموزشگاه : تهران - خیابان شریعتی - بالا تر از خیابان ملک - جنب بانک صادرات - پلاک 651 طبقه دوم - واحد7

88146323 - 88446780 - 88146330

http://www.tahlildadeh.com/