عنوان مقاله: بیایید این کوئری را Tune (بهینه) کنیم!

نویسنده مقاله: تورج عزیزی

تاریخ انتشار: دی ماه ۱۳۹۷

منبع: https://nikamooz.com/lets-query-that-tune

مقدمه

دوستان در این مقاله یک کوئری را به عنوان نمونه انتخاب کرده ام و قصد دارم قدم به قدم آن را tune کرده و تاثیر tune را پس اعمال آن بررسی کنم.

کوئری که مشکل دارد

اجازه دید به کوئری زیر نگاهی بیندازیم:

```
DECLARE @i INT = ۹۹۹
SELECT
SalesOrderID,
SalesOrderDetailID,
CarrierTrackingNumber,
OrderQty,
LineTotal
FROM Sales.SalesOrderDetail
WHERE ProductID < @i
ORDERBY CarrierTrackingNumber
GO
```

همانطوری که در اینجا می بینید من از یک متغیر محلی به همراه یک گزاره شرطی نامساوی استفاده کرده ام تا تعدادی ردیف را از جدول Sales.SalesOrderDetail برگردانم. وقتی کوئری را اجرا می کنید و به پلن اجرای کوئری نگاه می کنید با چند مشکل جدی در پلن مواجه می شوید.

SQL Server باید Clustered Index روی جدول Sales.SalesOrderDetail را به طور کامل Scan کند، چون هیچ ایندکس Nonclustered ای که کوئری را پوشش دهد (Covering Index) وجود ندارد. این کوئری ۱۳۸۲ logical read دارد و زمان سپری شده اش حدود ۸۰۰ ms است.

• Query Optimizer یک عملگر فیلتر Explicit در پلن کوئری منتشر کرد، که یک مقایسه ردیف- به –ردیف برای یافتن ردیف های واجد شرایط (ProductID < @i) انجام می دهد.

• به دلیل وجود عبارت ORDER BY CarrierTrackingNumber ، یک عملگر SORT Explicit در پلن کوئری مشاهده می شود.

• عملگر SORT روی TempDb انجام می شود (Spill)، چون در تخمین Cardinality اشتباه رخ داده است.شرط نامساوی و متغیر محلی با هم باعث شده اند که SQL Server از مقدار Hard code شده ۳۰% از کل ردیف های جدول استفاده کند. در کوئری ما این مقدار ۳۶۳۹۵ ردیف (۱۲۱۳۱۷ * ۳۰%) است. در واقعیت کوئری ۱۲۰۶۲۱ ردیف را بر می گرداند، که به این معنی است که عملگر Sortباید مرتب سازی را در دیتابیس TempDb انجام دهد چون میزان حافظه درخواستی اش خیلی کوچکتر از حد نیاز بوده است.

و حالا من از شما می پرسم – چطور می توانید این کوئری را بهینه کنید؟ پیشنهاد های شما چیست؟ کمی فکر کنید، چطور می شود این کوئری را بدون بازنویسی بهینه کرد؟

بیایید کوئری را tune کنیم!

البته که باید روی استراتژی ایندکس گذاری مان برای بهینه کردن کار کنیم. بدون وجود یک Non-clustered index پشتیبان ، پلن بالا تنها پلنی خواهد بود که Query Optimizer می تواند به کار گیرد. اما یک Non-clustered index خوب برای این کوئری بخصوص چیست؟ به طور نرمال، من در مورد ایجاد Non-clustered index ابتدا به شرط جستجو نگاه می کنم:

```
WHERE ProductID < @i
```

ما به دنبال ردیف هایی هستیم که فیلتر روی ProductID آنها اعمال می شود. بنابراین اقدام به ایجاد یک Non-clustered indexروی این ستون می کنیم. پس اجازه دهید ایندکس را ایجاد کنیم:

```
CREATE NONCLUSTERED INDEX idx_Test ON Sales.SalesOrderDetail(ProductID)
GO
```

بعد از ایجاد Non-clustered index باید تغییرات داده شده را آزمایش کنیم، بنابراین کوئری را دوباره اجرا می کنیم. و حدس می زنید چه اتفاقی می افتد؟ Query Optimizer از Non-clustered index استفاده نمی کند! ما میتوانیم Query Optimizer را مجبور کنیم از Non-clustered indexاستفاده کند تا بهتر بفهمیم *چرا Query Optimizer *ایندکس ما را انتخاب نمی کند:

```
DECLARE @i INT = ۹۹۹
SELECT
SalesOrderID,
SalesOrderDetailID,
CarrierTrackingNumber,
OrderQty,
LineTotal
FROM Sales.SalesOrderDetail WITH (INDEX(idx_Test))
WHERE ProductID < @i
ORDER BY CarrierTrackingNumber
GO
```

وقتی به پلن اجرا نگاه می کنید، یک جانور می بینیم – یک پلن موازی!

در این حالت کوئری ۳۷۰۲۴۴ logical read دارد! و زمان سپری شده هم حدود همان ۸۰۰ ms است. این دیگه چه جهنمیه؟ وقتی با جزئیات بیشتر به پلن اجرا نگاه می کنید، می بینید که Query Optimizer یک Bookmark Lookup تولید کرده، چون Non-clustered index ایجاد شده قبلی یک ایندکس پوشش دهنده کوئری محسوب نمی شود. این کوئری از Tipping Point عبور کرده، چون ما تقریباً همه ردیف را با شرط جستجوی فعلی مان برمیگردانیم. بنابراین عاقلانه نیست که بخواهیم Non-clustered index را به همراه عمل گران قیمتی مثل Boomark Lookup استفاده کنیم.

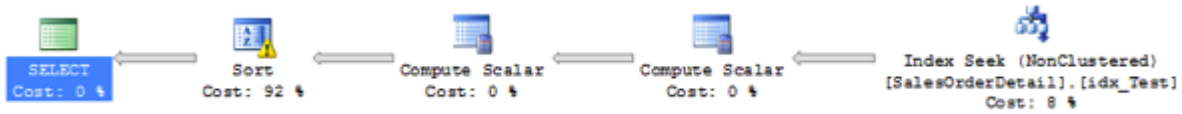
برای حل این مشکل باید ستون های اضافه ای که در لیست SELECT آمده اند را باید در سطح برگ Non-clustered index اضافه یا INCLUDE کنیم:

- CarrierTrackingNumber
- OrderQty
- UnitPrice
- UnitDiscountPrice

اجازه دهید Non-clustered index را دوباره ایجاد کنیم:

```
CREATE NONCLUSTERED INDEX idx_Test ON Sales.SalesOrderDetail(ProductID)
INCLUDE (CarrierTrackingNumber, OrderQty, UnitPrice, UnitPriceDiscount)
WITH
(
DROP_EXISTING = ON
)
GO
```

ما یک تغییر دیگر اعمال کرده ایم، بنابراین باید تغییرمان را دوباره آزمایش کنیم. اما این بار بدون query hint ، چون Query Optimizerباید Non-clustered indexرا به طور خودکار استفاده کند. و حدس بزنید چه اتفاقی می افتد؟ حالا ایندکس انتخاب شده و در پلن اجرا قابل مشاهده است:

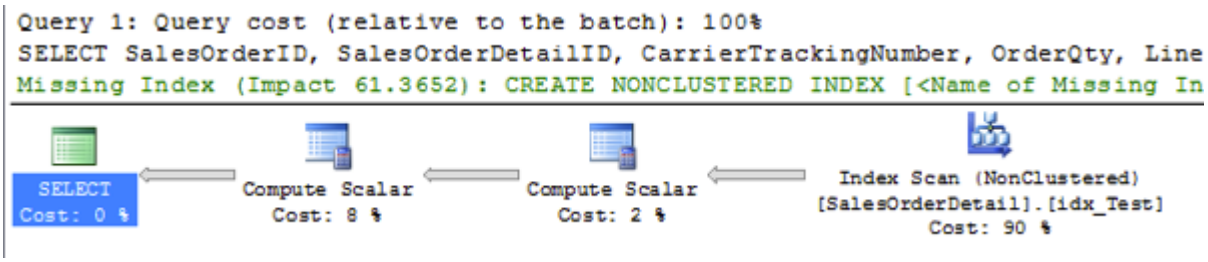


SQL Server حالا یک عمل Seek روی Non-clustered index اجرا کرده، اما ما هنوز یک عملگر Sort علنی در پلن اجرا داریم. و همچنین به دلیل مقدار Hard-code شده ۳۰% در تخمین Cardinality مرتب سازی روی TempDb انجام می شود. Logical read های ما به ۷۵۷ تا کاهش پیدا کرده اما زمان سپری شده هنوز چیزی حدود ۸۰۰ ms است. الان چه کاری می توانید انجام دهید؟

الان میتوانیم ستونCarrierTrackingNumber را در ساختار navigation در Non-clustered index اضافه کنیم. این ستونی است که SQL Server بر اساس ان مرتب سازی می کند. وقتی ما این ستون را در ستون اول Non-clustered index داشته باشیم، ما دیتای آن ستون را از قبل مرتب شده داریم، بنابراین SORT باید از پلن اجرا حذف شود. و به عنوان یک عارضه جانبی مثبت، چیزی روی دیتابیس TempDb برای مرتب سازی وجود ندارد و هیچ عملگری مشکل تخمین Cardinality نخواهد داشت. پس اجازه دهید با این فرض Non-clustered index را دوباره ایجاد کنیم.

```
CREATE NONCLUSTERED INDEX idx_Test ON Sales.SalesOrderDetail(CarrierTrackingNumber, ProductID)
INCLUDE (OrderQty, UnitPrice, UnitPriceDiscount)
WITH
(
DROP_EXISTING = ON
)
GO
```

می توانید در تعریف ایندکس ببینید که ما حالا دیتا را بر حسب ستون های ProductID و CarrierTrackingNumber از پیش مرتب کرده داریم. وقتی کوئری را دوباره اجرا کنید و به پلن اجرا نگاه کنید می توانید ببینید که عملگر Sort حذف شده و SQL Sevrer سطح برگ Non-clustered index را با یک شرط باقیمانده (residual predicate) به طور کامل اسکن می کند:



این پلن آن قدر ها هم بد نیست! ما فقط به ۷۶۴ logical read احتیاج داریم و زمان سپری شده کوئری به ۶۰۰ ms کاهش پیدا کرده. ۲۵% بهینه تر از قبل! اما Query Optimizer یک Non-clustered index با استفاده از قابلیتی *عالی* به نام Missing Index Recommendations پیشنهاد می دهد! چون ما چشم بسته به Query Optimizer اعتماد داریم، Non-clustered index توصیه شده را ایجاد می کنیم:

```
CREATE NONCLUSTERED INDEX [SQL Server doesn't care about names, why I should care about names?]
ON [Sales].[SalesOrderDetail] ([ProductID])
INCLUDE ([SalesOrderID],[SalesOrderDetailID],[CarrierTrackingNumber],[OrderQty],[LineTotal])
GO
```

وقتی کوئری اولیه را دوباره اجرا می کنید چیز های شگفت انگیزی می بینید Query Optimizer :از Non-clustered index قبلی *ما* استفاده کرده و Missing Index Recommendations را کنار گذاشته! شما ایندکسی ایجاد کرده اید هیچوقت توسط SQL Server استفاده نمی شود مگر برای عملیات INSERT, UPDATE یا DELETE که در آن بایدNon-clustered indexهم تاثیر بگیرد.

شما فقط یک سربار ***خالص*** روی دیتابیس تحمیل کرده اید.اما از طرفی Query Optimizer را هم با حذف مرحله Missing Index Recommendationsارضا کرده اید. اما این هدف نهایی ***نیست***: هدف این است که این ایندکس ***مورد استفاده*** هم قرار بگیرد.
نتیجه گیری: هیچوقت به Query Optimizer اعتماد نکنید!