عنوان مقاله: Tuning در SQL Server با Set Statistics IO
نویسنده مقاله: تیم فنی نیک آموز
تاریخ انتشار: فروردین ۱۴۰۱
منبع: https://nikamooz.com/Query-Tuning-in-SQL-Server-with-Set-Statistics-IO

مسئله

اصول اولیه نحوه استفاده از دستور Set Statistics IO برای Tuning کوئری‌های SQL Server کدام هستند؟ چارچوبی برای فراخوانی عبارت و خلاصه‌ای از خروجی‌های کلیدی آن به چه صورتی است؟ همچنین، چندین مثال از تفسیر خروجی دستورات برای tuning کوئری‌های SQL Server ارائه شود. در نهایت، نشان دادن چگونگی گزارش خروجی هزینه کوئری با نمودارهای طرح اجرا می‌تواند به تایید نتایج tuning بر اساس Set Statistics IO کمک کند.

راه‌حل

این مقاله بر Tuning کوئری‌ها با دستور Set Statistics IO تمرکز دارد. این عبارت بین خواندن‌های فیزیکی و خواندن‌های منطقی و همچنین اسکن داده‌ها تمایز قائل می‌شود. خواندن‌های فیزیکی بر انتقال داده‌ها از فضای ذخیره‌سازی دیسک به حافظه و ذخیره در یک حافظه Cache داده متمرکز می‌شود (گاهی اوقات به آن مخزن بافر داده نیز می‌گویند). خواندن منطقی به خواندن از حافظه Cache داده اشاره دارد. SQL Server تنها پس از انتقال داده‌ها از فضای ذخیره‌سازی دیسک به حافظه Cache، یک برنامه اجرایی برای داده‌ها ایجاد می‌کند. اسکن‌ها به تعداد دفعاتی که داده‌ها پس از انتقال از حافظه دیسک به حافظه برای اسکن یک کوئری خاص انجام می‌شود، اشاره دارد. آمار تولید شده توسط دستور Set Statistics IO مربوط به خواندن فیزیکی، خواندن منطقی و اسکن داده‌ها است.

ممکن است با بررسی execution plan ها به عنوان ابزاری برای بررسی نحوه بالابردن کارایی کوئری آشنا باشید، اما آن مبحث نشان می‌دهد که دسترسی به برنامه‌های اجرایی را چگونه می‌توان برای برخی از DBA ها در برخی از سرورهای پایگاه داده ممنوع کرد. در مورد tuning کوئری‌ها، استفاده از دستور Set Statistics IO می‌تواند اطلاعات مفیدی را ارائه دهد. حتی اگر یک متخصص SQL Server حق دسترسی به Execution Plan را داشته باشد، که به حق دسترسی showplan معروف است، دستور Set Statistics IO همچنان می‌تواند اطلاعات ارزشمندی را برای tuning کوئری‌هایی که در execution plan در دسترس نیستند، ارائه دهد.

چارچوبی برای دستور Set Statistics IO

اسکریپت کوتاه زیر چارچوبی متشکل از شبه کد و کامنت‌ها را برای استخراج Statistics IO برای کوئری‌های T-SQL ارائه می‌دهد که مجموعه‌های نتایج را برمی‌گرداند. اغلب امکان تولید یک مجموعه نتایج با دو یا چند طرح کوئری T-SQL مختلف وجود دارد. با مقایسه Statistics IO از هر طرح کوئری، می‌توانید درک کنید که کدام یک از آنها کمترین بار خواندن و اسکن را روی یک سرور قرار می‌دهند.

- کد با یک عبارت Use Statement برای تعیین یک محتوای پایگاه داده پیش‌فرض شروع می‌شود.
- سپس، دستور DBCC Dropcleanbuffers و دستور checkpoint صفحات بافر داده را ذخیره می‌کند و بافرهای داده را از محتویات ارزیابی‌های قبلی Statistics IO پاک می‌کند. این دستورها به شما این امکان را می‌دهد که یک کوئری را با یک مخزن بافر داده خالی، بدون نیاز به غیرفعال کردن و راه اندازی مجدد سرور، آزمایش کنید. استفاده از dbcc dropcleanbuffers در سرور محیط تولید، می‌تواند باعث تأثیر منفی روی کارایی شود.
- قبل از اجرای دستور sql که می‌خواهید Statistics IO را ارزیابی کنید، دستور Set Statistics IO on را فراخوانی کنید.
- سپس عبارت sql ای را که برای Statistics IO می‌خواهید اجرا کنید.
- پس از اجرای دستور sql query، می‌توانید قابلیت ارزیابی Statistics IO را با دستور Set Statistics IO off غیرفعال کنید.

```
-- a framework for using the Set Statistics IO  statement

use statement that specifies a default database context
go

-- clean and then clear dirty data buffers
dbcc dropcleanbuffers;
checkpoint;

-- turn Set Statistics IO  on
Set Statistics IO  on

-- T-SQL query statement to be tuned
T-SQL query statement with reads and scans to be optimized

-- turn Set Statistics IO  on
Set Statistics IO  off
```

خلاصه‌ای از Statistics IO موجود از Set Statistics IO

جدول زیر Statistics IO گزارش شده توسط دستور Set Statistics IO در SQL Server ۲۰۱۹ را نشان می‌دهد. شما می‌توانید با بررسی آمار گزارش شده برای دو اجرای متوالی Set Statistics IO ، دو عبارت کوئری T-SQL را با هم مقایسه کنید. هنگامی که یک کوئری به دو یا چند جدول ارجاع می‌شود، ممکن است استخراج Statistics IO به طور جداگانه برای هر جدول ارجاع شده و همچنین مجموعه ترکیبی همه جداول در منبع، برای یک کوئری مفید باشد.

اگر دو عبارت کوئری دارید که مجموعه نتایج یکسانی را تولید می‌کنند، کوئری با کمترین تعداد خواندن، بهینه‌تر است. همچنین باید روی هماهنگ کردن عبارات کوئری تمرکز کنید تا خواندن فیزیکی را نسبت به خواندن‌های منطقی به حداقل ممکن برسانید. به این دلیل که خواندن فیزیکی، صفحات داده را از یک دستگاه ذخیره‌سازی جمع‌آوری می‌کند در حالی که خواندن‌های منطقی صفحات را برای یک کوئری از حافظه Cache داده جمع‌آوری می‌کند. جمع‌آوری صفحات از حافظه سریعتر از دیسک است.

نام statistic io	شرح statistic io
Scan count	تعداد جستجوها یا اسکن‌ها برای بازیابی همه مقادیر برای ساخت مجموعه داده نهایی برای خروجی
logical reads	خواندن یک صفحه داده از حافظه.
physical reads	خواندن صفحه داده از دیسک، زمانی که آن صفحه در حافظه موجود نیست.
page server reads	به انتقال یک صفحه از دیسک به بافر داده در حافظه اشاره دارد. سرور تعداد صفحات خوانده شده در هر ثانیه را در تمام پایگاه‌های داده منعکس می‌کند.
read-ahead reads	هِد خواندن، یک صفحه داده را قبل از درخواستی خاص از دیسک، به حافظه منتقل می‌کند.
page server read-ahead reads	به انتقال یک صفحه از دیسک به بافر داده در حافظه قبل از درخواستی خاص اشاره دارد.
lob logical reads	Logical reads for text, ntext, image, varchar(max), nvarchar(max), varbinary(max), or columnstore index pages.
lob physical reads	Physical reads for text, ntext, image, varchar(max), nvarchar(max), varbinary(max), or columnstore index pages.
lob page server reads	Refers to the transfer of a text, ntext, image, varchar(max), nvarchar(max), varbinary(max), or columnstore index pages from disk to the data buffer in memory across all databases.
lob read-ahead reads	Refers to read-ahead reads for text, ntext, image, varchar(max), nvarchar(max), varbinary(max), or columnstore index pages.
lob page server read-ahead reads	Refers to the transfer of a text, ntext, image, varchar(max), nvarchar(max), varbinary(max), or columnstore index pages from disk to the data buffer in memory before it is specifically requested across all databases.

تاثیر اندازه جدول و خواندن از بافر بر روی Statistics IO

کوئری‌ها زمانی که صفحات کمتری را ارجاع می‌دهند در SQL Server کارآمدتر اجرا می‌شوند. این مورد به این دلیل است که خواندن تعداد زیاد صفحات به منابع بیشتری نسبت به مجموعه‌ای از صفحات کوچکتر نیاز است. همچنین، کوئری‌ها زمانی که داده‌ها را از حافظه می‌خوانند به جای ذخیره‌سازی دیسک، کارآمدتر اجرا می‌شوند. این مورد نیز به این دلیل است که بازیابی اطلاعات از حافظه سریعتر از ذخیره‌سازی دیسک است.

اسکریپت T-SQL زیر این دستورالعمل‌های طراحی را با Statistics IO تایید می‌کند.

- یک عبارت use WideWorldImporters را به عنوان پایگاه داده پیش‌فرض قرار می‌دهد. این پایگاه داده برای دانلود از سایت مایکروسافت در دسترس است.
- سه مرتبه اجرای دستور Set Statistics IO on و Set Statistics IO off عمل برگرداندن Statistics IO را انجام می‌دهند.
  - اولین کوئری تعداد ردیف‌های جدول شهرها را در شِمای برنامه شمارش می‌کند. قبل از این کوئری، dbcc Dropcleanbuffer و عبارت Checkpoint وجود دارد، بنابراین هیچ صفحه کثیفی در حافظه cache داده در حافظه وجود ندارد. صفحه کثیف، صفحه‌ای در حافظه با به روز رسانی‌هایی است که روی دیسک commit نشده‌اند. در جدول شهرها، ده ها هزار شهر وجود دارد.
  - کوئری دوم همانند کوئری اول است. با این حال، از آنجایی که Statistics IO برای این کوئری پس از کوئری اول بدون اجرای عبارات میانی برای clean و clear کردن حافظه cache داده‌ها اجرا می‌شود، داده‌هایی که کوئری برای انجام شمارش نیاز دارد، از قبل در حافظه هستند. به عبارت دیگر، قبل از شمارش ردیف‌های جدول شهرها، نیازی به انتقال داده‌ها از حافظه دیسک به حافظه نیست.
  - سومین کوئری نیز همانند طرح کوئری اول است، با این تفاوت که ردیف‌های جدول StateProvinces را در طرح برنامه شمارش می‌کند. این جدول دارای ۵۳ ردیف است.

```
-- designate WideWorldImporters as the default database
use WideWorldImporters
go

-- count of cities from clean buffers
dbcc dropcleanbuffers;
checkpoint;

Set Statistics IO  on

select count(*) number_of_cities
from Application.Cities
```

```
Set Statistics IO  off

-- count of cities from dirty buffers
--dbcc dropcleanbuffers;
--checkpoint;

Set Statistics IO  on

select count(*) number_of_cities
from Application.Cities

Set Statistics IO  off

-- count of stateprovinces from clean buffers
dbcc dropcleanbuffers;
checkpoint;

Set Statistics IO  on

select count(*) number_of_stateprovinces
from Application.StateProvinces

Set Statistics IO  off
```

در اینجا یک مقایسه از مجموعه نتایج حاصل از عبارات کوئری اول و دوم است. نتایج حاصل از اولین کوئری در سمت چپ و نتایج حاصل از کوئری دوم در سمت راست قرار دارد. جای تعجب نیست که هر دو عبارت کوئری تعداد سطرهای یکسانی را برای جدول شهرها برمی‌گردانند.

From first query statement	From second query statement
number_of_cities 1 37940	number_of_cities 1 37940

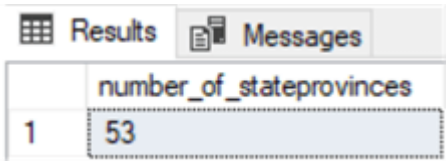
تصویر صفحه زیر قسمتی از پنجره پیام‌ها را برای عبارات کوئری اول و دوم ارائه می‌دهد. بین خطوط پیام برای هر کوئری تفاوت‌هایی وجود دارد که به راحتی قابل شناسایی هستند.

- پیام‌های اولین کوئری روی جدول Cities در اولین خط نشان داده شده است.
  - به یاد داشته باشید که dbcc dropcleanbuffers و دستور checkpoint قبل از فراخوانی کوئری، cache داده را clean و clear می‌کند.
  - مقادیر غیر صفر برای خواندن فیزیکی و خواندن read-ahead نشان می‌دهد که SQL Server باید قبل از اجرای طرح کوئری برای دستورات کوئری، صفحات داده را از محل ذخیره‌سازی دیسک به حافظه cache داده منتقل کند.
  - مقدار غیر صفر برای خواندن‌های منطقی تعداد صفحات داده نشان می‌دهد که در حین اجرای طرح کوئری برای دستورات کوئری به cache داده‌ها دسترسی پیداکرده‌اند.

- پیام‌های دومین کوئری روی جدول Cities در دومین خط نشان داده شده است.
  - توجه داشته باشید که مقادیر برای خواندن فیزیکی و خواندن read-ahead هر دو صفر هستند. به این دلیل که داده‌های منبع برای کوئری قبلا از دیسک به حافظه cache داده برای اجرای اولین کوئری منتقل شده بود.
  - حافظه cache داده‌ها قبل از اجرای طرح کوئری دوم clean و clear نشده است.

- از آنجایی که کوئری دوم نیازی به جمع‌آوری داده‌های منبع از فضای ذخیره‌سازی مبتنی بر دیسک ندارد (۰ خواندن فیزیکی و ۰ خواندن پیشخوان)، کارآمدتر از کوئری اول است.

در تصویر زیر خروجی کوئری سوم نشان داده شده است. فقط یک مقدار ۵۳ را برمی‌گرداند که تعداد ردیف‌ها در جدول StateProvinces است.



تاثیرات فیلتر کردن با Having در مقابل Where در Statistics IO

دو کوئری بعدی دو رویکرد متفاوت را با هم مقایسه می‌کند. موضوع کوئری برگرداندن تعداد شهرها بر اساس ایالت‌هایی که با حرف A در پایگاه داده WideWorldImporters شروع می‌شوند. در ایالات متحده چهار ایالت وجود دارد که با حرف A شروع می‌شود. اینها آلاباما، آلاسکا، آریزونا و آرکانزاس هستند. در این بخش دو روش برای شمارش تعداد شهرها بر اساس ایالت برای ایالت‌هایی که با حرف A شروع می‌شوند نشان داده می‌شود.

- می‌توانید جدول Cities را با جدول StateProvinces جوین کنید و سپس ایالت‌ها را بر اساس نام ایالت که با حرف A شروع می‌شود گروه‌بندی کنید. سپس، فقط تعداد ردیف‌ها را بر اساس نام ایالت بشمارید.
- همچنین، می‌توانید از عبارت Where برای فیلتر کردن جوین یا ردیف‌های جداول شهرها و ایالت‌ها استفاده کنید. سپس، می‌توانید تعداد ردیف‌ها را بر اساس نام ایالت بشمارید.

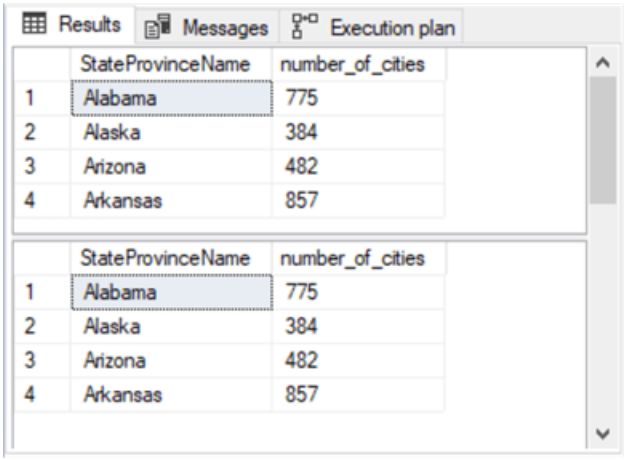
در اینجا اسکریپتی برای شمارش شهرها بر اساس ایالت‌هایی که با حرف A شروع می‌شوند نشان داده شده است که هر دو رویکرد را در بر گرفته است. هر رویکرد در یک Set Statistics IO تعبیه شده است. این طراحی تضاد کارایی هر رویکرد را به راحتی نشان می‌دهد.

- اسکریپت با دستور use شروع می‌شود تا محتوای پیش‌فرض پایگاه داده را مشخص کند.
- اولین بلوک کد بعد از دستور use برای فیلتر کردن از عبارت having که بعد از group by آورده شده، استفاده می‌کند.
- دومین بلوک کد بعد از دستور use برای فیلتر کردن با عبارت Where قبل از group by آورده شده است.
- هر یک از سه بلوک کد با خطوط تیره از هم جدا شده‌اند.

```
-- designate WideWorldImporters as the default database
use WideWorldImporters
go
-----------------------------------------------------------------------
-- count of Cities by StateProvince with having clause
dbcc dropcleanbuffers;
checkpoint;

Set Statistics IO  on

-- count of Cities within StateProvinces
select StateProvinces.StateProvinceName, count(*) number_of_cities
from Application.Cities
inner join Application.StateProvinces
on Cities.StateProvinceID = StateProvinces.StateProvinceID
group by StateProvinces.StateProvinceName
having left(StateProvinceName,1) = 'A'
-----------------------------------------------------------------------
Set Statistics IO  off

-- count of Cities by StateProvince with where clause
dbcc dropcleanbuffers;
checkpoint;

Set Statistics IO  on
```

```
-- count of Cities by StateProvince
select StateProvinces.StateProvinceName, count(*) number_of_cities
from Application.Cities
inner join Application.StateProvinces
on Cities.StateProvinceID = StateProvinces.StateProvinceID
where left(StateProvinceName,1) = 'A'
group by StateProvinces.StateProvinceName

Set Statistics IO  off
```

در اینجا مجموعه نتایج حاصل از کوئری با عبارت having در مقایسه با مجموعه نتایج حاصل از کوئری با عبارت Where نشان داده شده است. مجموعه نتایج بالایی برای کوئری با عبارت having است. مجموعه نتایج پایینی برای کوئری با عبارت Where است. همان طور که به وضوح از این مقایسه می‌بینید، از منظر نتایج مهم نیست که تعداد شهرها را به صورت ایالت با یک عبارت having یا با یک عبارت Where محاسبه کنید. با این حال، آیا دو رویکرد مختلف، هزینه‌های کوئری یکسانی دارند؟


Results Messages Execution plan

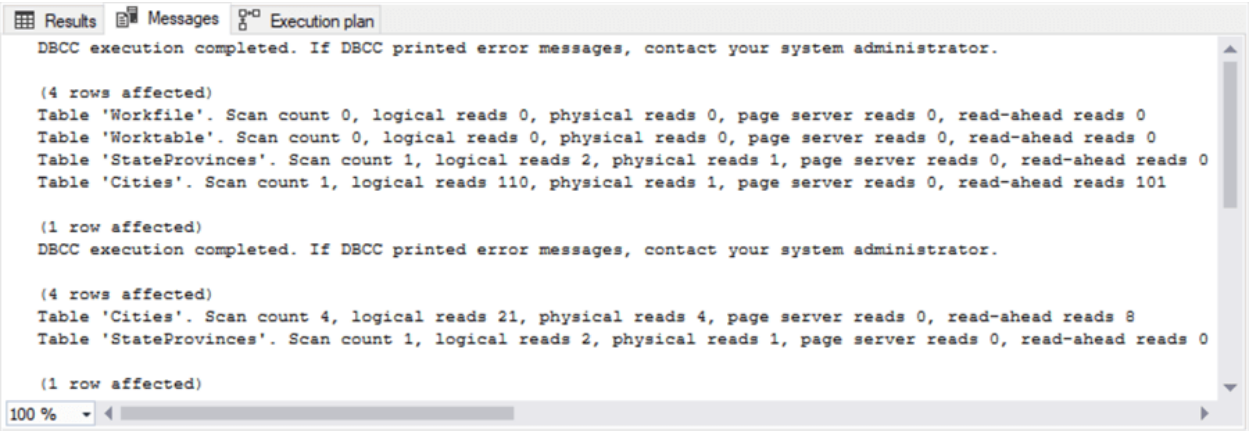
	StateProvinceName	number_of_cities
1	Alabama	775
2	Alaska	384
3	Arizona	482
4	Arkansas	857

	StateProvinceName	number_of_cities
1	Alabama	775
2	Alaska	384
3	Arizona	482
4	Arkansas	857

تصویر صفحه زیر قسمتی از خروجی پنجره پیام‌ها را برای دو کوئری نشان می‌دهد. این خروجی توسط جفت دستورهای Set Statistics IO on و Set Statistics IO off تنظیم شده است.

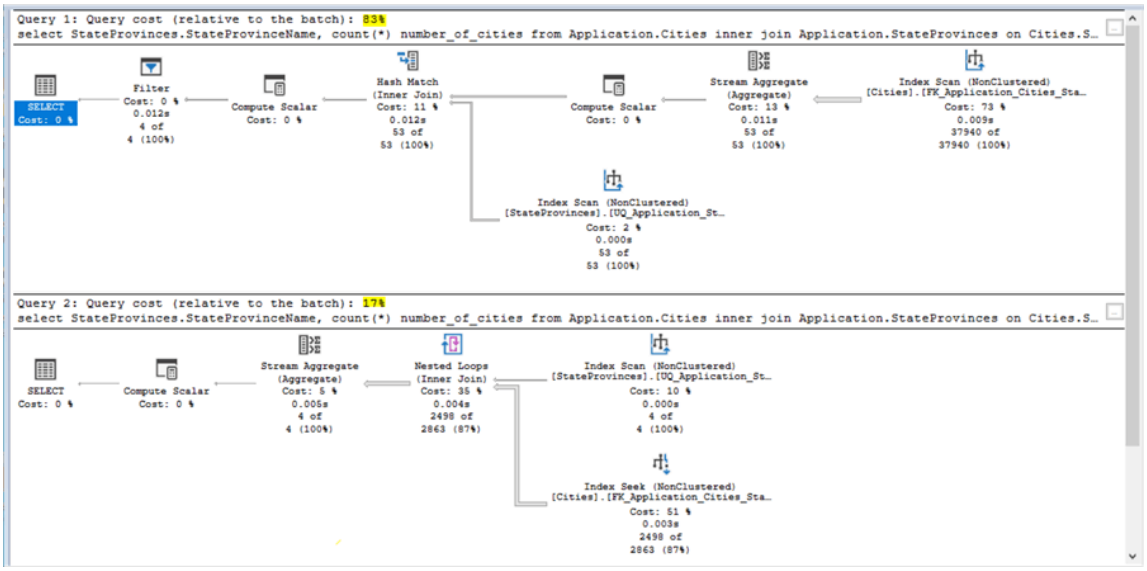
- مجموعه چهار ردیف بالایی که با Table ‘Workfile’ شروع می‌شود تا خطی که با Table ‘Cities’ شروع می‌شود، مربوط به کوئری با عبارت having است، در ادامه خطی که با DBCC executed completion شروع می‌شود مربوط به کوئری با عبارت where است.
- کوئری با عبارت having پیچیده‌تر است و شامل خواندن‌های مبتنی بر دیسک بیشتر از کوئری با عبارت Where است.

- به عنوان مثال، در مجموع ۱۰۲ صفحه داده از طریق خواندن دیسک (خواندن فیزیکی و خواندن read-ahead) در آمار جدول شهرها با عبارت having وجود دارد. در مقابل، فقط ۱۲ صفحه داده از طریق خواندن دیسک برای جدول شهرها با عبارت where مشاهده می‌شود.
- همچنین اجرای کوئری با عبارت having به دو منبع موقت Workfile و Worktable نیاز دارد که برای اجرای کوئری با عبارت Where این موارد لازم نیست.



یکی دیگر از راه‌های ایجاد یک کوئری مناسب که کمترین تأثیر را بر SQL Server دارد، مقایسه هزینه کوئری در یک دسته واحد نسبت به آمار دسته‌ای در اجرای یک کوئری با عبارت having در مقابل عبارت Where است:

- آمار هزینه کوئری طرح اجرا برای کوئری با عبارت having در بالای طرح اجرا با عبارت Where در تصویر زیر آمده است.
- هزینه کوئری برای کوئری با عبارت having، ۸۳ درصد در مقابل فقط ۱۷ درصد برای کوئری با عبارت Where است.
- نمودارهای زیر دستورات Set Statistics IO را تأیید می‌کنند و نشان می‌دهند که هزینه برای کوئری با عبارت having به طور چشمگیری بیشتر از کوئری با عبارت Where است.



تاثیر استفاده از توابع Windows Aggregate

ممکن است وسوسه شوید که از یک تابع windows aggregate برای تولید نتایجی مانند نتایج بخش قبل استفاده کنید زیرا توابع Windows Aggregate می‌توانند روی یک عبارت واحد، عمل جمع و پارتیشن‌بندی را انجام دهند. بسته به نیاز شما، اگر به دنبال مجموعه‌ای از نتایج جدول‌بندی شده[1] باشید، توابع Windows Aggregate ممکن است رویکردی بهینه باشد. این بخش نشان می‌دهد که چگونه دستور Set Statistics IO به همراه نمودارهای طرح اجرا می‌تواند مناسب بودن این رویکرد را در مقایسه با رویکرد سنتی‌تر با یک عبارت where، برای دریافت نتایج جدول‌بندی شده مانند موارد قبل روشن کند.

اسکریپت زیر استفاده از تابع count که جز windows aggregate ها است را با دو کوئری برای جمع‌آوری شهرها در مجموعه‌ای از مقادیر StateProvince نشان می‌دهد (برای مقادیر StateProvince که نام آنها با حرف A شروع می‌شود). داده‌های منبع برای نتایج جدول‌بندی شده مانند بخش قبل است، یعنی جداول ایالت‌ها و شهرها در شمای پایگاه داده WideWorldImporters قرار دارد.

اسکریپت زیر دو کوئری برای دریافت نتایج جدول‌بندی شده را نشان می‌دهد. هر دو کوئری از تابع count استفاده می‌کنند. کوئری‌ها یکسان هستند به جز یک کلمه کلیدی (distinct)، که خروجی را تغییر می‌دهد و تأثیر قابل توجهی بر مصرف منابع کوئری دارد.

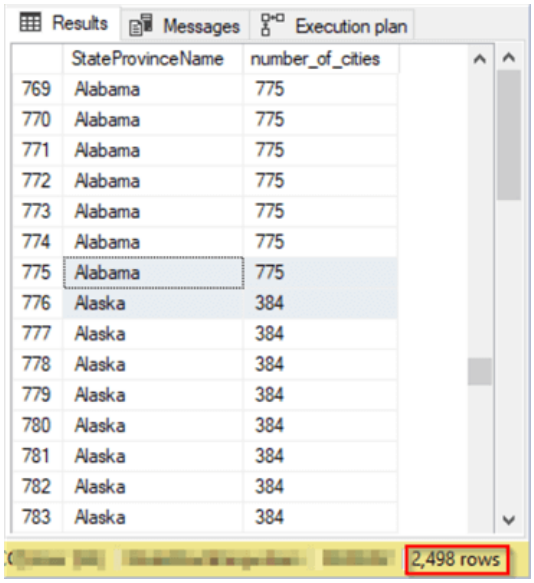
```
-- designate WideWorldImporters as the default database
use WideWorldImporters
go

-- count of cities by StateProvince with where clause
-- and windows aggregate count function

dbcc dropcleanbuffers;
checkpoint;

Set Statistics IO  on

select
StateProvinces.StateProvinceName
,count(Cities.CityID)
over (partition by StateProvinces.StateProvinceName) number_of_cities
from [WideWorldImporters].[Application].[Cities]
inner join Application.StateProvinces
```

[1] Tabulated

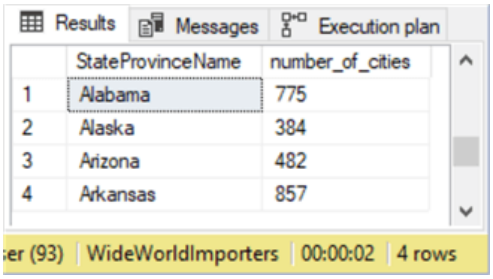
```
on Cities.StateProvinceID = StateProvinces.StateProvinceID
where left(StateProvinceName,۱) = 'A'
order by StateProvinces.StateProvinceName

Set Statistics IO  off

-- count of cities by StateProvince with where clause
-- and windows aggregate count function
-- remove duplicate rows with distinct keyword

dbcc dropcleanbuffers;
checkpoint;

Set Statistics IO  on

select distinct
StateProvinces.StateProvinceName
,count(Cities.CityID)
over (partition by StateProvinces.StateProvinceName) number_of_cities
from [WideWorldImporters].[Application].[Cities]
inner join Application.StateProvinces
on Cities.StateProvinceID = StateProvinces.StateProvinceID
where left(StateProvinceName,۱) = 'A'
order by StateProvinces.StateProvinceName

Set Statistics IO  off
```

تصویر زیر قسمتی از مجموعه نتایج برای اولین کوئری بدون کلمه کلیدی distinct را نشان می‌دهد. ۲۴۹۸ ردیف در مجموعه نتایج وجود دارد.

- هر ردیف با نام یک مقدار از StateProvice و تعداد شهرها در آن ایالت پر می‌شود. به عنوان مثال، ردیف آخر برای ایالت آلاباما، ۷۷۵ است، زیرا ۷۷۵ شهر در آلاباما وجود دارد.
- شما از خروجی بخش قبل می‌دانید که شمارش‌ها در StateProvince برای چهار ایالت که با حرف A شروع می‌شود عبارتند از: ۷۷۵، ۳۸۴، ۴۸۲، و ۸۵۷. این چهار تا در مجموع به ۲۴۹۸ می‌رسد، که تعداد ردیف‌هایی در نتایج تنظیم شده برای کوئری بدون کلمه کلیدی distinct است.



نتایج تنظیم شده برای کوئری دوم در زیر ظاهر می‌شود. توجه داشته باشید که فقط چهار ردیف در این مجموعه نتایج وجود دارد. این مجموعه نتایج دقیقاً با مقادیر بازگشتی از کوئری‌های بخش قبل مطابقت دارد. بنابراین، با تعیین یک کلمه کلیدی Distinc برای نتایج یک کوئری بر اساس یک تابع windows aggregate، می‌توانید نتایج یک کوئری سنتی را بر اساس فیلتر کردن با عبارت where یا با داشتن عبارت having و یک group by، به همان صورت قبلی تولید کنید.



از اهداف این مقاله، پاسخ به دو سوال زیر است:

- آیا افزودن کلمه کلیدی، منابعی برای انجام کوئری اضافه می‌کند؟
- آیا استفاده از تابع windows aggregate نتایج جدول‌بندی شده را با مصرف منابع کمتری نسبت به کوئری‌های مبتنی بر فیلتر کردن با عبارت where یا با داشتن عبارت having و یک group by، تولید می‌کند؟

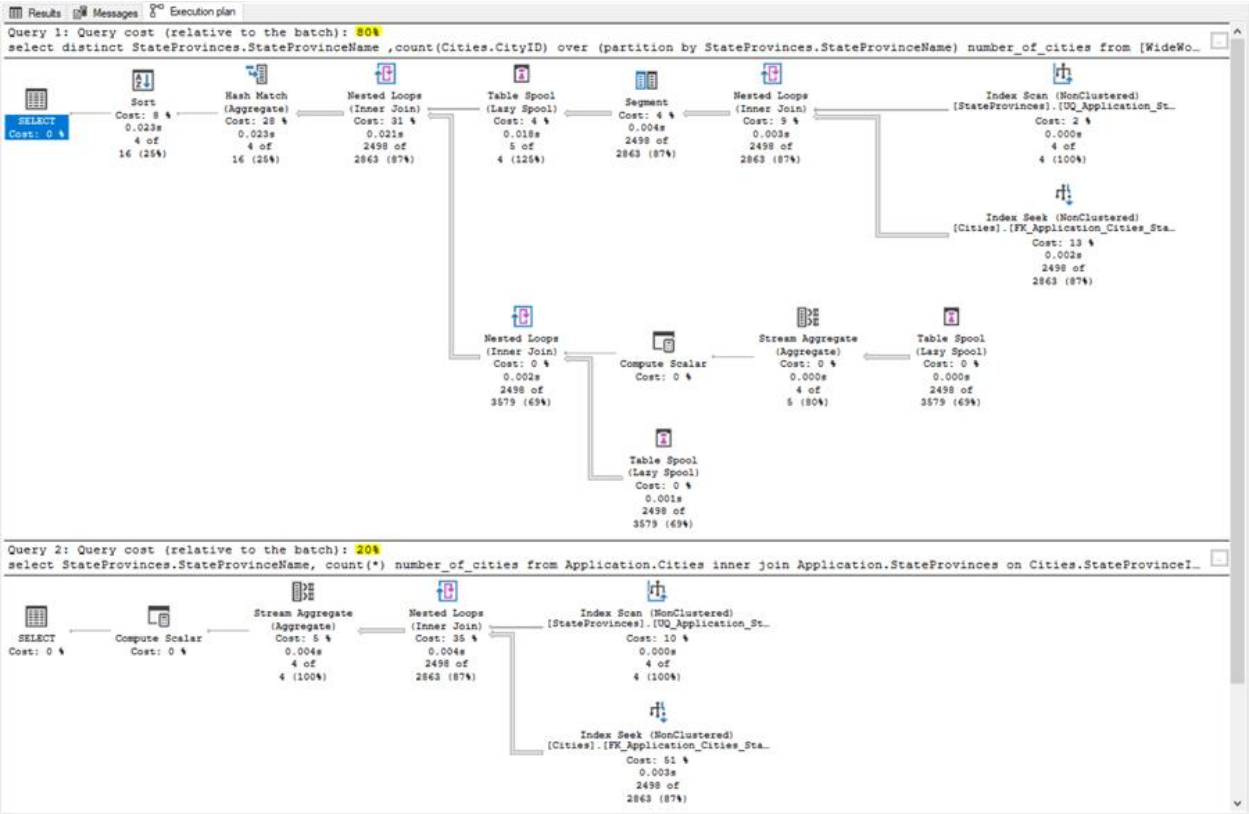
با بررسی سربرگ Messages در SSMS، بینشی در مورد پاسخ سوال اول به دست می‌آورید. جای تعجب نیست که استفاده از کلمه کلیدی distinct منجر به هزینه کوئری بیشتر می‌شود. در اینجا قسمتی از سربرگ Messages آمده است.

- پیام‌های Set Statistics IO برای اولین کوئری بدون کلمه کلیدی distinct پس از اولین پیام DBCC execution completed ظاهر می‌شود و پیام‌های Set Statistics IO برای دومین کوئری با کلمه کلیدی distinct بعد از دومین پیام DBCC execution completed ظاهر می‌شود.
- پیام‌ها به استثنای یک ردیف برای جدولی با نام "Workfile" یکسان هستند. این ردیف هیچ آماری را در مورد نحوه استفاده از شی اضافی به نام Workfile نشان نمی‌دهد، اما کلمه کلیدی distinct شی Workfile را در فعالیت io برای کوئری دوم معرفی می‌کند.
- من همچنین نمودارهای طرح اجرا را برای دو کوئری بررسی کردم. هزینه کوئری برای یک دسته با هر دو کوئری نشان می‌دهد که کوئری دوم با کلمه کلیدی distinct به منابع بیشتری نیاز دارد. هزینه‌ها برای کوئری بدون کلمه کلیدی distinct، ۳۹ درصد و برای کوئری با کلمه کلیدی distinct، ۶۱ درصد است.
- اگر چه افزودن کلمه کلیدی distinct هزینه‌های نسبی را اضافه می‌کند، اما نتیجه مطلوب را نیز ایجاد می‌کند، که احتمالاً نتیجه مهمتر خواهد بود.

```
DBCC execution completed. If DBCC printed error messages, contact your system administrator.

(2498 rows affected)
Table 'Worktable'. Scan count 3, logical reads 5079, physical reads 0, page server reads 0, read-ahead reads 0
Table 'Cities'. Scan count 4, logical reads 21, physical reads 7, page server reads 0, read-ahead reads 1
Table 'StateProvinces'. Scan count 1, logical reads 2, physical reads 1, page server reads 0, read-ahead reads 0

(1 row affected)

DBCC execution completed. If DBCC printed error messages, contact your system administrator.

(4 rows affected)
Table 'Worktable'. Scan count 3, logical reads 5079, physical reads 0, page server reads 0, read-ahead reads 0
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0
Table 'Cities'. Scan count 4, logical reads 21, physical reads 7, page server reads 0, read-ahead reads 1
Table 'StateProvinces'. Scan count 1, logical reads 2, physical reads 1, page server reads 0, read-ahead reads 0
100 %
```

سوال دوم در مورد اینکه آیا عملکرد windows aggregate با کلمه کلیدی distinct در استفاده از منابع محافظه کارانه‌تر از یک رویکرد سنتی SQL با فیلتر Where و تابع aggregate است؟ می‌توان این سوال را با مقایسه نمودارهای طرح اجرا به راحتی پاسخ داد. هنگامی که دو کوئری با هم در یک دسته اجرا می‌شوند، نتایج زیر در سربرگ Execution plan ظاهر می‌شوند.

- طرح اجرای بالایی برای کوئری با کلمه کلیدی distinct و تابع windows aggregate است. طرح اجرای پایینی برای کوئری با عبارت Where و تابع count است.
- همان طور که می‌بینید، رویکردی که از یک تابع windows aggregate با کلمه کلیدی distinct استفاده می‌کند، ۸۰ درصد از منابع در دسته را مصرف می‌کند، در حالی که کوئری با تابع aggregate اولیه و عبارت where فقط به ۲۰ درصد از منابع در دسته نیاز دارد.



منابع

https://www.mssqltips.com/sqlservertip/۶۴۳۳/query-tuning-in-sql-server-with-set-statistics-io/