

برای پردازش یک عبارت در بسیاری از موارد نیاز هست که عبارت به کلمات تشکیل دهنده اش تجزیه شود. روش‌های متنوعی برای انجام این عمل وجود دارد که یکی از شناخته شده‌ترین آنها استفاده از جدول اعداد می‌باشد (البته از بین روش‌های مجموعه گرا/set-based).

روشهایی که قرار هست در ادامه توضیح داده شوند بر اساس کوئری بازگشتی می‌باشند. الگوریتم‌های متنوعی بر اساس recursive CTE برای حل این مساله خلق شده اند. که من تنها به دو روش آن اکتفا می‌کنم.

Recursive CTE در نسخه‌ی 2005 به SQL Server اضافه شده است. توسط این تکنیک مسائل پیچیده و گوناگونی را میتوان بسادگی حل نمود. مخصوصا مسائلی که ماهیت بازگشتی دارند مثل پیمایش یک درخت یا پیمایش یک گراف وزن دار.

روش اول:

یک کوئری بازگشتی دارای دو بخش هست به نام‌های Anchor و recursive. در بخش دوم کوئری باز خودش را فراخوانی می‌کند تا به داده‌هایی که در مرحله قبل تولید شده اند دسترسی پیدا کند در اولین فراخوانی توسط عضو recursive، داده‌های تولید شده در قسمت Anchor قابل دسترسی هستند. در قسمت دوم، کوئری آنقدر خود را فراخوانی می‌کند تا دیگر سطری از مرحله قبل وجود نداشته باشد که به آن مراجعه کند.

توضیح تکنیک:

در گام اول اندیس شروع و پایان کلمه اول را بدست می‌آوریم.

سپس در گام بعدی از اندیس پایان کلمه قبلی به عنوان اندیس شروع کلمه جدید استفاده می‌کنیم.

و اندیس پایان کلمه توسط تابع charindex بدست می‌آید.

کوئری تا زمانی ادامه پیدا میکند که کلمه برای تجزیه کردن در رشته باقی مانده باشد. فقط فراموش نکنید که حتما باید آخر عبارت یک کارکتر space داشته باشید.

```
DECLARE @S VARCHAR(50)='I am a student I go to school ';
WITH CTE AS
(
    SELECT 1 rnk,
           1 start,
           CHARINDEX(' ', @s) - 1 ed

    UNION ALL

    SELECT rnk + 1,
           ed + 2,
           CHARINDEX(' ', @s, ed + 2) - 1
    FROM CTE
    WHERE CHARINDEX(' ', @s, ed + 2) > 0
)
SELECT rnk, SUBSTRING(@s, start, ed - start + 1) AS word
FROM CTE

/* Result
rnk      word
-----
1        I
2        am
3        a
4        student
5        I
6        go
7        to
8        school
*/
```

روش دوم:

در این روش در همان CTE عبارت تجزیه می‌شود و عمل تفکیک به مرحله بعدی واگذار نمی‌شود، در گام اول، اولین کلمه انتخاب می‌شود. و سپس آن کلمه از رشته حذف می‌شود. با این روش همیشه اندیس شروع کلمه برابر با 1 خواهد بود و اندیس پایان کلمه توسط تابع charindex بدست خواهد آمد. در گام بعدی اولین کلمه موجود در رشته ای که قبلاً اولین کلمه از آن جدا شده است بدست می‌آید و باز مثل قبلی کلمه انتخاب شده از رشته جدا شده و رشته برش یافته به مرحله بعد منتقل می‌شود. در این روش مثل روش قبلی آخر عبارتی که قرار هست تجزیه شود باید یک کارکتر خالی وجود داشته باشد.

```
DECLARE @a VARCHAR(50)='I am a student I go to school ';
WITH MyWords(ranking, word, string) AS(
    SELECT 1,
           CAST(SUBSTRING(@a, 1, CHARINDEX(' ', @a) - 1) AS VARCHAR(25)),
           STUFF(@a, 1, CHARINDEX(' ', @a), '')
    UNION ALL
    SELECT ranking + 1,
           CAST(SUBSTRING(string, 1, CHARINDEX(' ', string) - 1) AS VARCHAR(25)),
           STUFF(string, 1, CHARINDEX(' ', string), '')
    FROM MyWords
    WHERE CHARINDEX(' ', string) > 0
)
SELECT ranking, word FROM MyWords;
```

و خروجی:

ranking	word
1	I
2	am
3	a
4	student
5	I
6	go
7	to
8	school

نظرات خوانندگان

نویسنده: محسن
تاریخ: ۲۱:۴۴ ۱۳۹۱/۱۰/۲۹

از مقاله شما دوست عزیز کمال تشکر را دارم.

نویسنده: محمد سلم آبادی
تاریخ: ۲۲:۲۳ ۱۳۹۱/۱۰/۲۹

ممنونم دوست گرامی

مقدمه و شرح مساله

توسط ویژگی‌های جدیدی که در نسخه 2012 به بحث window افزوده شد می‌توانیم مسالهای running total و running average را به شکل بهینه ای حل کنیم.

ابتدا این دو مساله را بدون بکارگیری ویژگی‌های جدید، حل نموده و سپس سراغ توابع جدید خواهیم رفت.

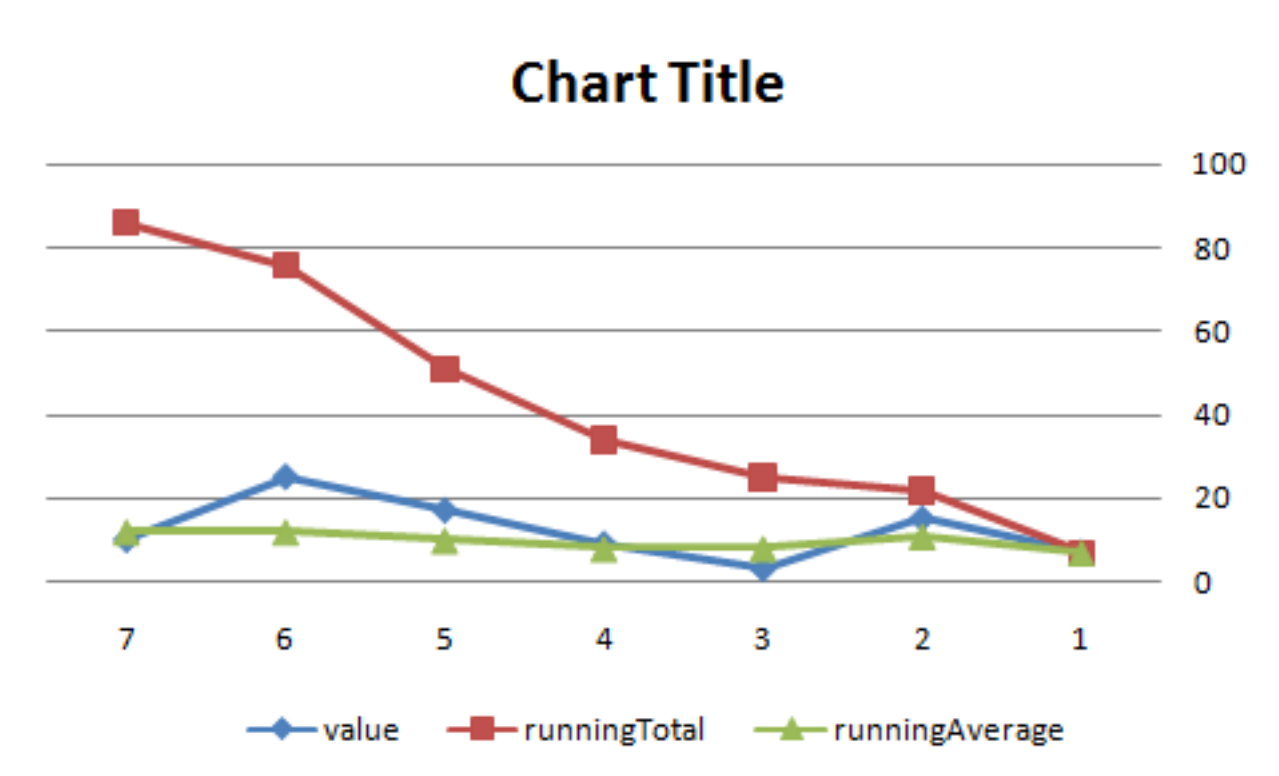
قبل از هر چیزی لازم است جدول زیر ساخته شود و داده‌های نمونه در آن درج شود:

```
create table testTable
(
    day_nbr integer not null primary key clustered,
    value integer not null check (value > 0)
);
insert into testTable
values (10, 7), (20, 15), (30, 3), (40, 9), (50, 17), (60, 25), (70, 10);
```

مساله running total بسیار ساده است، یعنی جمع مقدار سطر جاری با مقادیر سطرهای قبلی (بر اساس یک ترتیب معین) running average هم مشابه به running total هست با این تفاوت که میانگین مقادیر سطر جاری و سطرهای قبلی محاسبه می‌شود.

	day_nbr	value	runningTotal	runningAverage
1	10	7	7	7
2	20	15	22	11
3	30	3	25	8
4	40	9	34	8
5	50	17	51	10
6	60	25	76	12
7	70	10	86	12

و نتیجه به صورت نمودار:

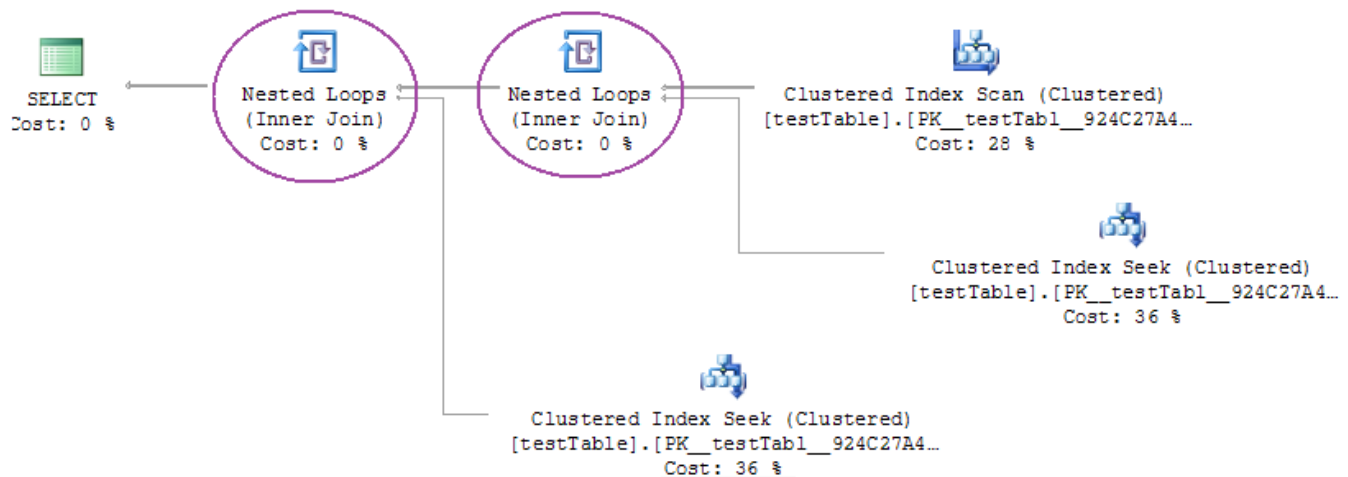


راه حل در SQL Server 2000

توسط دو correlated scalar subquery می‌توانیم مقادیر دو ستون مورد نظر را محاسبه کنیم:

```
select *,
    runningTotal = (select sum(value)
                    from testTable
                    where day_nbr <= t.day_nbr),
    runningAverage = (select avg(value)
                     from testTable
                     where day_nbr <= t.day_nbr)
from testTable t;
```

اگر به نقشه اجرای این query نگاه کنید گره (عملگر) inner join دو بار بکار رفته است (به وجود دو subquery)، که این عدد در روش توابع تجمعی window به صفر کاهش پیدا خواهد کرد

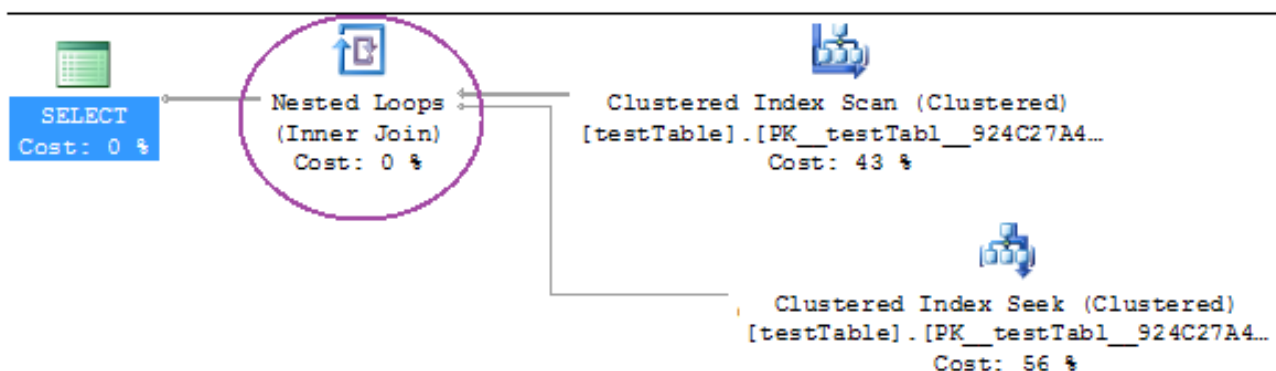


راه حل در SQL Server 2005

توسط cross apply به سادگی می توانیم دو subquery که در روش قبل بود را به یکی کاهش دهیم:

```
select *
from testTable t
cross apply (select sum(value) as runningTotal,
                avg(value) as runningAverage
            from testTable
            where day_nbr <= t.day_nbr)d;
```

این بار تنها یک عملگر inner join در نقشه اجرای query مشاهده می شود:



راه حل در SQL Server 2012

با اضافه شدن برخی از ویژگی های استاندارد به ماده OVER مثل rows و range شاهد بهبودی در عملکرد query هستیم.

یکی از کاربردهای توابع تجمعی window حل مساله running total و running average است.

به تصویر زیر توجه کنید، همانطور که در قبل توضیح دادم ما به سطر جاری و سطرها پیشین نیاز داریم تا اعمال تجمعی (جمع و میانگین) را روی مقادیر بدست آمده انجام دهیم. در تصویر زیر سطر جاری و سطرها قبلی به ازای هر سطر به وضوح قابل مشاهده است، مثلاً هنگامی که سطر جاری برابر با روز 30 است ما خود سطر جاری (current row) و تمام سطرها پیشین و قبلی (unbounded preceding) را نیاز داریم.

Day	value
10	7
20	15
30	3
40	9
50	17
60	25
70	10

Unbounded preceding

Current row

و اکنون query مورد نظر

```
select *, sum(value) over(order by day_nbr rows between unbounded preceding and current row) as
runningTotal,
avg(value) over(order by day_nbr rows between unbounded preceding and current row) as
runningAverage
from testTable
```

در نقشه اجرای این query دیگر خبری از عملگر inner join نخواهد بود که به معنای عملکرد بهتر query است.

