

در این مطلب قصد دارم به نکاتی ساده در T-SQL بپردازم، امیدوارم مفید واقع شود.

1- تفاوت Count(*) و Count(column)

در Count(*)، همه Row ها و مقادیرشان مورد جستجو قرار می‌گیرد، اما در Count(column) فقط مقادیر غیر Null ستون مورد نظر، مورد جستجو قرار می‌گیرد.
 مثال:

```
Create Table Test(ID int,Firstname varchar(20));
Insert Into Test (ID,Firstname) Values(1,'K');
Insert Into Test (ID,Firstname) Values(2,'B');
Insert Into Test (ID) Values(3);
```

با اجرای Query زیر خواهیم داشت:

```
Select COUNT(*) From Test
```

خروجی آن 3 می‌باشد.

اما با اجرای Query زیر خواهیم داشت:

```
Select COUNT(Firstname) From Test
```

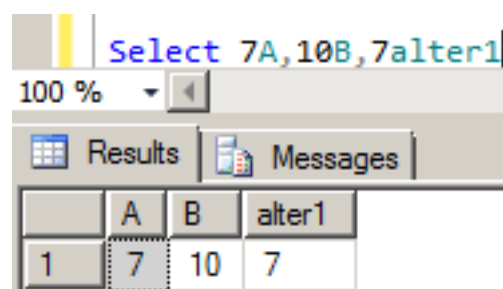
خروجی آن 2 می‌باشد، چون با توجه به سه رکوردی که در جدول Test درج شده بود، رکورد سوم برای فیلد Firstname با مقدار Null پر شده است.

هرگاه در اجرای Count، هدفشان بدست آوردن تعداد ستون خاصی است، از Count(column) استفاده نمایید.

2- بوسیله Script های زیر می‌توانیم عدد صفر تولید نماییم.

```
select count(cast(null as int))
select count(*) where 'a'='b'
select €
select ¥
select £
select $
select count(*)-count(*)
select Ascii('A')-Ascii('A')
select LEN('')
```

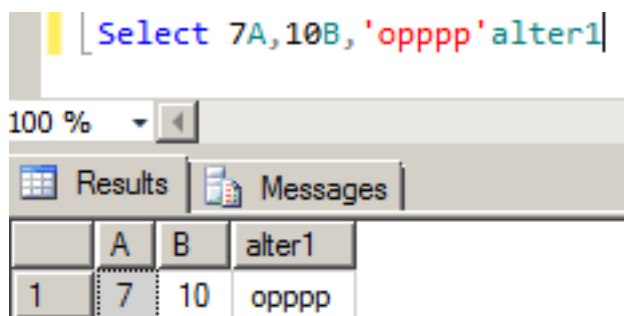
3- با یک Select ساده می‌توان نام ستون و مقدار را کنار هم نوشت و مشاهده نمود.



The screenshot shows a SQL query window with the text: `Select 7A,10B,7alter1`. Below the query window, the 'Results' tab is active, displaying a table with the following data:

	A	B	alter1
1	7	10	7

در مثال بالا مقادیر ستونها، عددی در نظر گرفته شده است، چنانچه تمایل به نمایش حروف داشته باشید، کفایت کاراکترهای حرفی را بین سنگل کوتیشن قرار دهید، همانند شکل زیر:



4- دو روش پیشنهادی برای تشخیص عدد بزرگتر از بین دو عدد
روش اول

ابتدا به فرمول زیر توجه نمایید:

$$(a+b)+ABS(a-b)$$

حاصل جواب فرمول، برابر است با دو برابر عدد بزرگتر، بنابراین اگر حاصل فوق را ضربدر عدد 5/0 نماییم، عدد بزرگتر بدست خواهد آمد. در نتیجه خواهیم داشت:

$$0.5(a+b)+ABS(a-b)$$

با اجرای Script زیر خروجی عدد 90.34 می‌باشد:

```
DECLARE @Value1 DECIMAL(5,2) = 80.22
DECLARE @Value2 DECIMAL(5,2) = 90.34
SELECT (0.5 * ((@Value1 + @Value2) + ABS(@Value1 - @Value2))) AS MaxColumn
```

اشکال در این روش این است که، اگر مقدار یکی از اعداد Null باشد، ماکزیمم بین دو عدد Null نمایش داده خواهد شد.
روش دوم

```
DECLARE @Value1 DECIMAL(5,2) = 9.22
DECLARE @Value2 DECIMAL(5,2) = 8.34
SELECT CASE WHEN @Value1 > @Value2 THEN @Value1 ELSE @Value2 END AS MaxColumn
```

در این روش اگر مقدار یکی از اعداد Null باشد، ماکزیمم بین دو عدد، عدد غیر Null می‌باشد.

5- مشاهده مشخصات کلیه دیتابیس‌های موجود در SQL Server با استفاده از Sys.Databases .

Select * From sys.databases

خروجی بصورت زیر خواهد بود:

00 %

	name	database_id	source_database_id	owner_sid
1	master	1	NULL	0x01
2	tempdb	2	NULL	0x01
3	model	3	NULL	0x01
4	msdb	4	NULL	0x01
5	ReportServer\$MSSQLSERVER2012	5	NULL	0x010500000000000515000000AD5D
6	ReportServer\$MSSQLSERVER2012TempDB	6	NULL	0x010500000000000515000000AD5D
7	AdventureWorks2012	7	NULL	0x010500000000000515000000AD5D

6- بوسیله دستور OUTPUT می‌توان خروجی Queryهای Delete، Update و Insert را مشاهده نمود:

مثال اول برای Query Delete :

```
Delete Test OUTPUT deleted.* Where Firstname='K'
```

100 %

	ID	Firstname
1	1	K

در شکل، تک رکورد حذف شده را مشاهده می‌نمایید.

مثال دوم برای Query Update بصورت زیر میباشد:

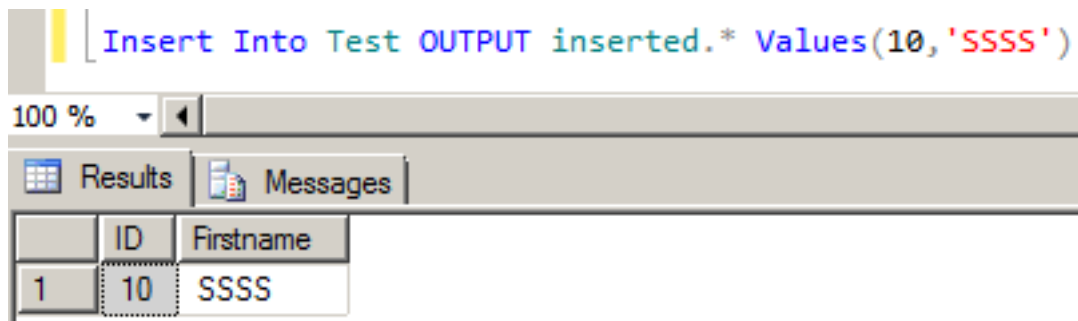
```
Update Test Set Firstname ='A' OUTPUT deleted.*,inserted.* Where Firstname=
```

100 %

	ID	Firstname	ID	Firstname
1	2	B	2	A
2	2	B	2	A

در شکل، مقدار A، مقدار جدیدی است که بروز رسانی شده است و مقدار B مقداری است که مربوط به قبل از بروز رسانی می‌باشد.

مثال سوم برای Query Insert بصورت زیر می‌باشد:



موفق باشید.

نظرات خوانندگان

نویسنده: تازه کار
تاریخ: ۲۳:۲۹ ۱۳۹۱/۰۸/۲۹

مورد 4 و 6 جالب بود . مرسی

نویسنده: مهدی موسوی
تاریخ: ۱۱:۵۵ ۱۳۹۱/۰۸/۳۰

سلام.

بنظرم این فقط یه حقه کثیف هستش که مطلقا منظور از چنین SQL رو همیشه بسادگی متوجه شد. منظورم روش اول در شماره 4 (بدست آوردن ماکزیموم دو عدد) و همونطوری که [اینجا](#) هم بهش اشاره شده، اگر مقدار جمع اون مقدار از Data Type اولیه بیشتر بشه، Arithmetic overflow رخ میده.

روش دوم رو هم میشه با IIF ساده تر نوشت:

```
DECLARE @Value1 DECIMAL(5,2) = 9.22
DECLARE @Value2 DECIMAL(5,2) = 8.34
SELECT IIF(@Value1 > @Value2, @Value1, @Value2) AS MaxColumn
```

یه روش دیگه هم هست، اونم اینکه بدین شکل با استفاده از VALUES مقادیر رو به Rowها بیاریم و با استفاده از MAX عدد بزرگتر رو انتخاب کنیم:

```
SELECT MAX(Col) FROM (VALUES (@Value1),(@Value2)) AS alias(Col)
```

موفق باشید.

نویسنده: فرهاد فرهمندخواه
تاریخ: ۱۲:۵۵ ۱۳۹۱/۰۸/۳۰

سلام

مرسی از نظر شما دوست عزیز، درمورد استفاده از کلمه حقه کثیف با شما موافق نیستم، روشی که در این مقاله ارائه شد، فقط یک روش می باشد نه حقه، که می تواند یک روش مناسبی نباشد، با توجه با توضیحات شما، روش ارائه شده، روش خوبی به نظر نمی رسد، بنده هم از راهنمایی شما استفاده بردم و متشکرم.

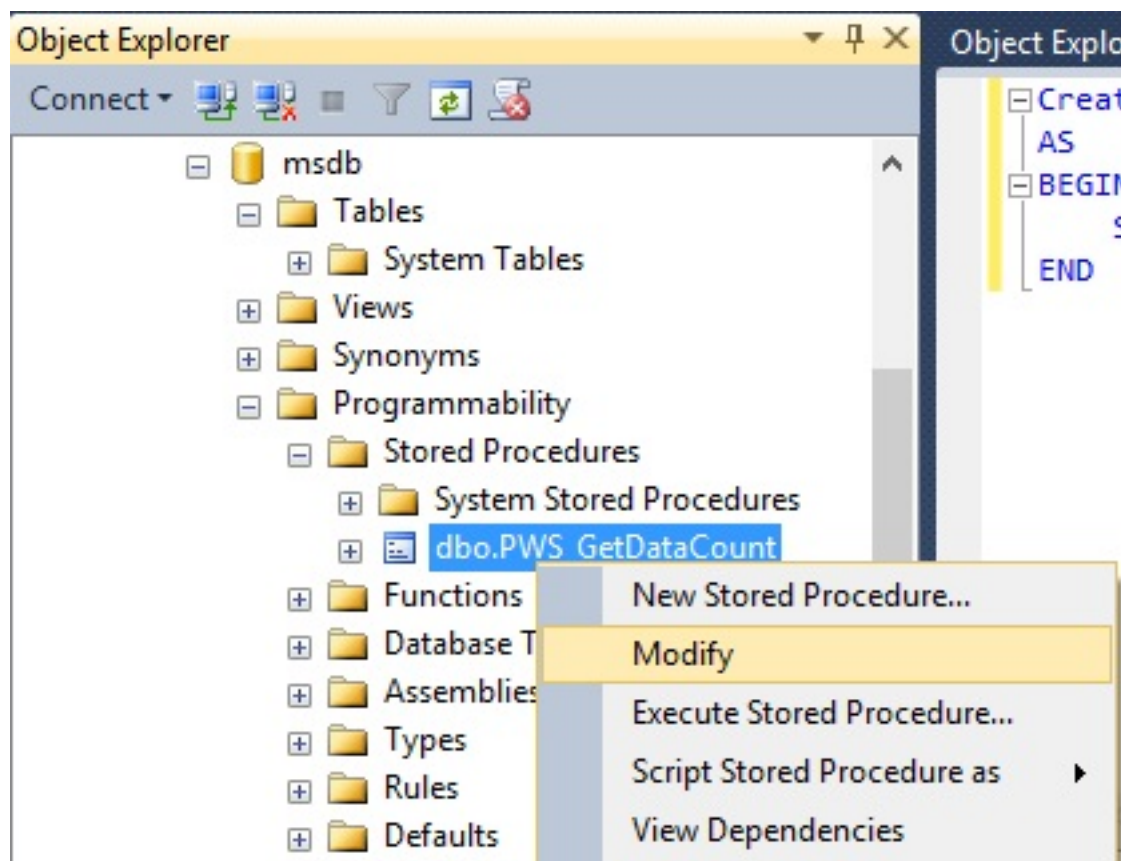
نویسنده: Daniel
تاریخ: ۲۲:۰۹ ۱۳۹۱/۰۸/۳۰

نکات جالبی بودند مرسی
جای Count(column) بهتر از Count(1) استفاده کرد.

در بسیاری مواقع پیش می‌آید که در SQL رویه‌های ذخیره شده ([Stored Procedure](#)) ایجاد می‌کنیم. اما پس از ایجاد در SQL این رویه قابل مشاهده بوده و می‌توان آن را ویرایش کرد. مثلاً می‌خواهیم یک رویه ذخیره شده ساده را ایجاد نماییم. ابتدا در [SQL Management Studio](#) (یا روش‌های دیگر) اقدام به ایجاد رویه می‌نماییم.

```
Create Procedure dbo.PWS_GetDataCount
AS
BEGIN
SELECT COUNT(*) FROM backupfile
END
```

همانگونه که در تصویر زیر مشاهده می‌نمایید این رویه ایجاد شده و قابل ویرایش می‌باشد.

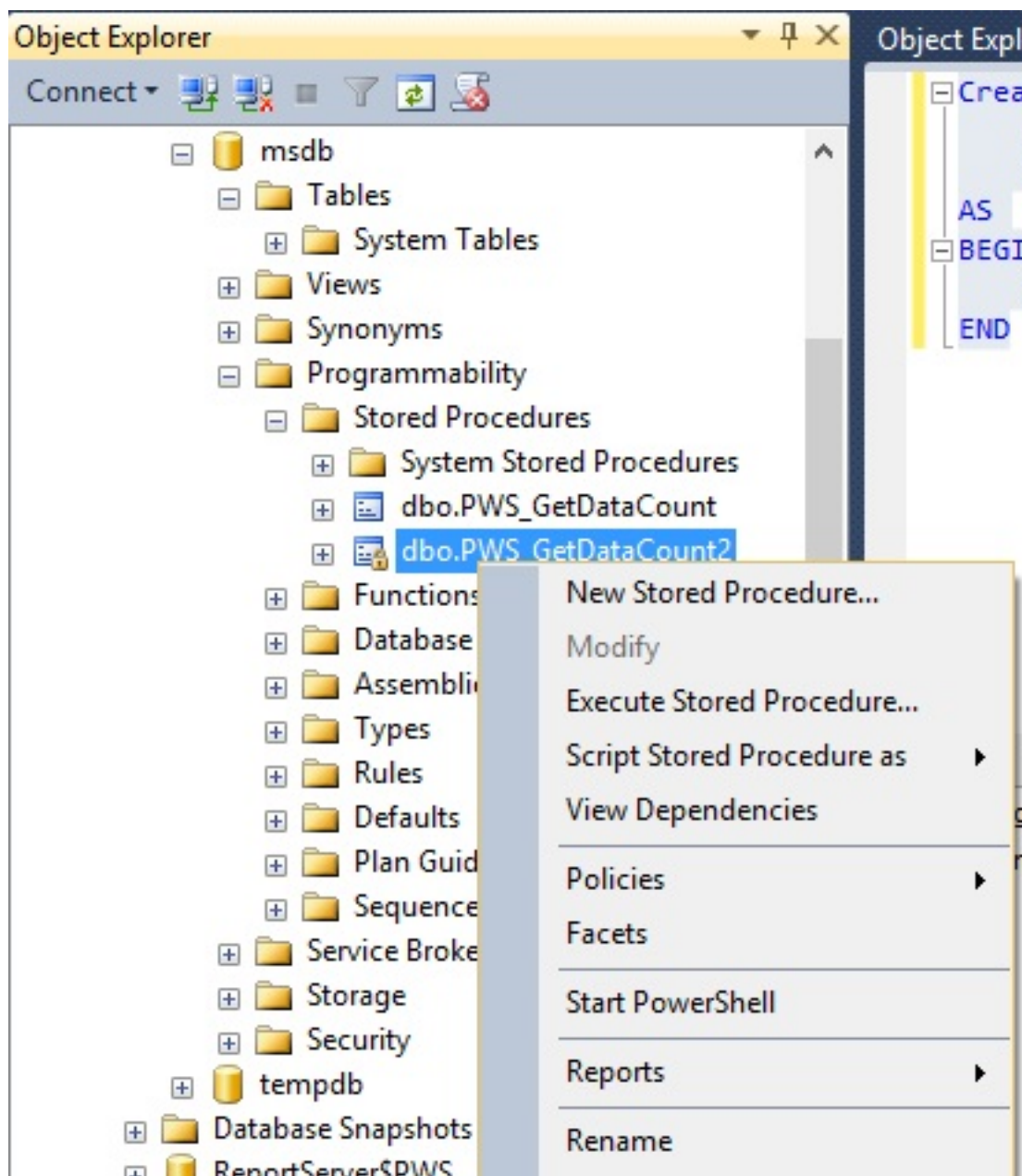


خوب تا اینجای کار هیچگونه مشکلی نیست، اما مواقعی پیش می‌آید که نمی‌خواهیم کسی به سورس این رویه‌ها دسترسی داشته باشد، یا اینکه آنها را تغییر دهد، مثلاً مواقعی که نرم افزار شما روی سایت مشتری اجرا شده و مشتری دسترسی به دیتابیس و رویه‌های آن دارد. در این مواقع می‌توان در صورت لزوم این رویه‌ها را فقط خواندنی کرد، یعنی حتی خود شما هم نمی‌توانید رویه را ویرایش نمایید (پس دقت کنید). روال کار بدین گونه است:

```
Create Procedure dbo.PWS_GetDataCount2
--Parameters
With Encryption
AS
BEGIN
```

```
SELECT COUNT(*) FROM backupfile
END
```

در واقع با نوشتن With Encryption قبل از AS می‌توان رویه ذخیره شده را رمزگذاری کرد. پس از ایجاد این رویه همانگونه که در تصویر زیر مشاهده می‌نمایید این رویه شکل یک کلید بروی آن ظاهر شده و دیگر قابل ویرایش نیست (بهتر است رویه‌های خود را در زمان انتشار روی سیستم مقصد رمزگذاری نمایید).



دقت نمایید که استفاده از این عمل تأثیری در سرعت اجرای رویه ذخیره شده ندارد. البته روش‌هایی هم برای عکس این عمل وجود دارد که می‌توانید از طریق این [لینک](#) به اطلاعات بیشتری دست پیدا کنید.

نظرات خوانندگان

نویسنده: فرهاد فرهمندخواه
تاریخ: ۱۳۹۱/۰۹/۰۶ ۸:۲۰

سلام

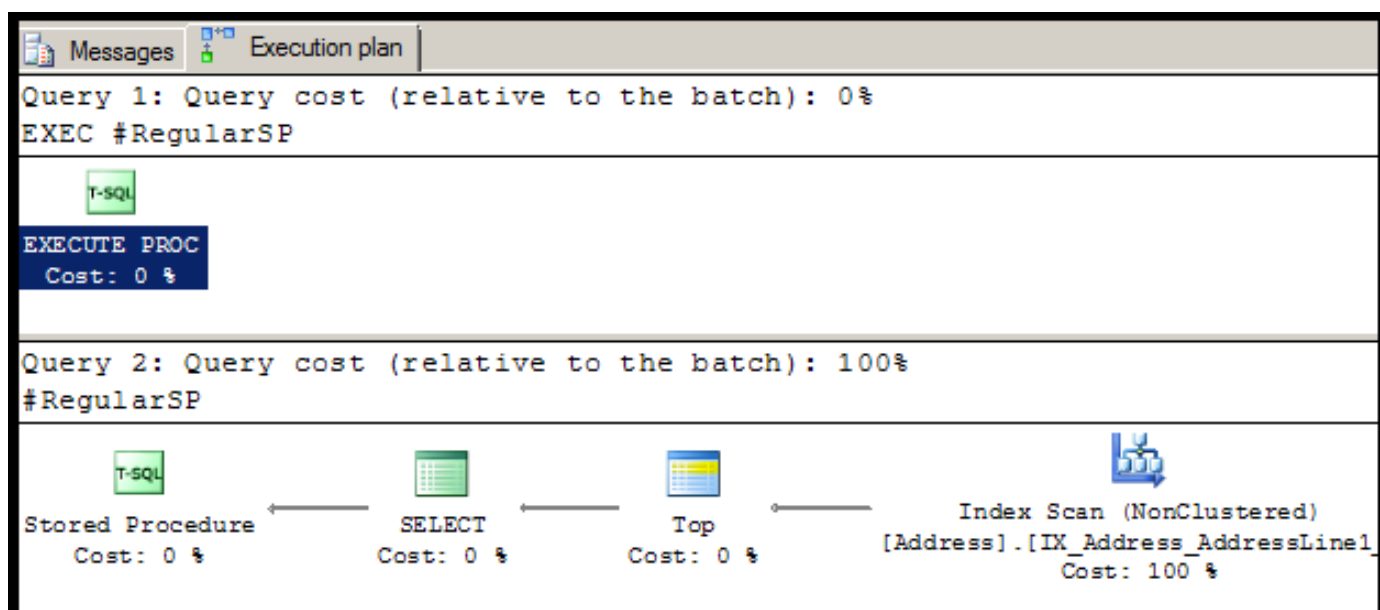
مرسی از مقاله شما، فقط نکته ای در مورد Stored Procedure های Encrypt شده وجود دارد، و آن این است که، این نوع Procedure ها را نمی‌توان بطور قابل فهم در Execution Plan مشاهده نمود.
مثال:

```
CREATE PROCEDURE #RegularSP
AS
SELECT TOP 10 City
FROM Person.Address
GO
```

اجرا با Execution Plan

```
EXEC #RegularSP
```

خروجی بصورت زیر خواهد بود:



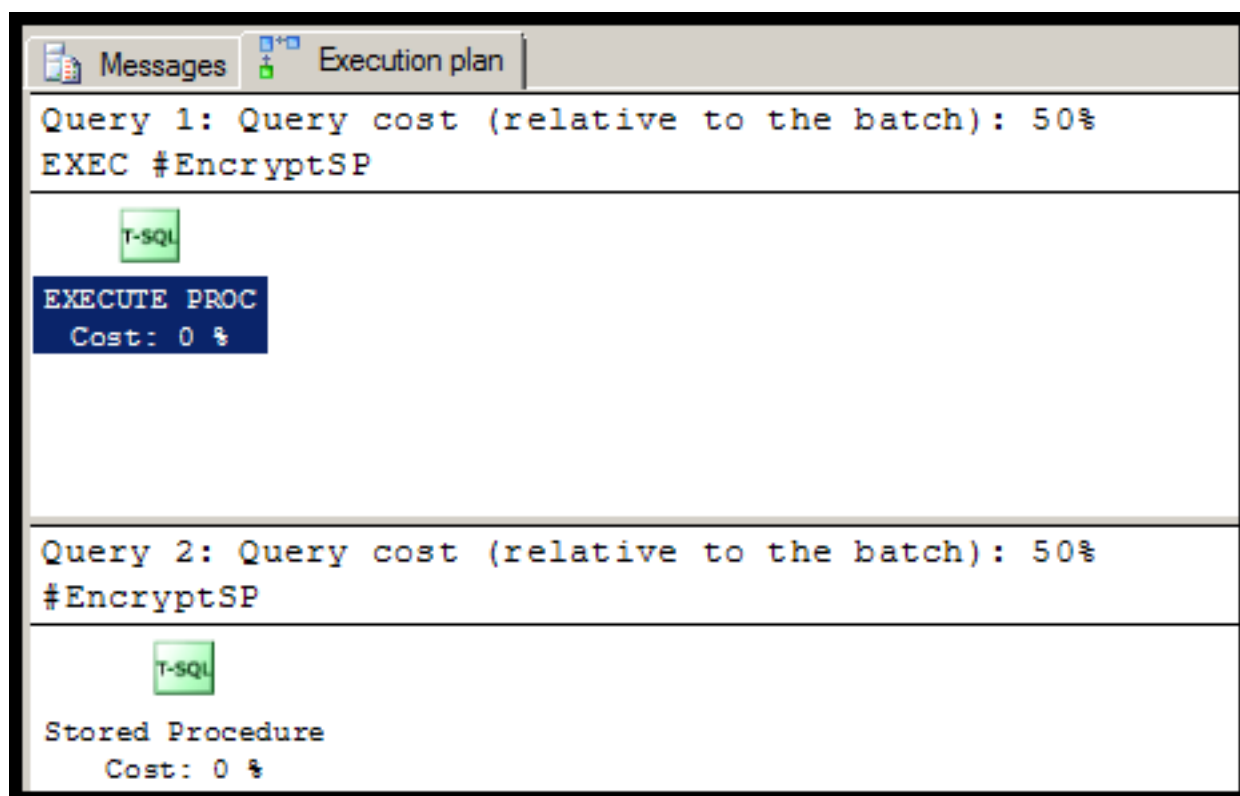
ایجاد Stored Procedure بصورت Encrypt شده:

```
CREATE PROCEDURE #EncryptSP
WITH ENCRYPTION
AS
SELECT TOP 10 City
FROM Person.Address
```

اجرا با Execution Plan

```
EXEC #EncryptSP
```


خروجی بصورت زیر خواهد بود:



حذف Stored Procedure های ایجاد شده:

```
DROP PROCEDURE #RegularSP
DROP PROCEDURE #EncryptSP
```

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۰۹/۰۶ ۱۰:۵۶

درسته دوست من، بدلیل اینکه شما با عمل Encrypt فقط یک جعبه سیاه می‌بینید که در صورت وجود ورودی هایی میگیرد (پارامتر) و خروجی هایی پس می‌دهد ما داخل اون مشاهده نمی‌کنیم، و دلیل اصلی ما برای Encrypt همین است که کسی محتوی SP را نتواند ببیند حتی خود ما.

نویسنده: ali
تاریخ: ۱۳۹۱/۰۹/۰۷ ۹:۰۶

مطلب بسیار خوبی بود و باعث یادآوری بسیاری نکات برای بنده شد و البته رمزگشایی SP ها هم کار چندان سختی نیست و نرم افزارهای زیادی جهت این امر می‌توان در اینترنت پیدا نمود.

[سایت سازنده نرم افزار](#)
[sql-decryptor](#)

نویسنده: صابر فتح الهی
تاریخ: ۱۱:۵۹ ۱۳۹۱/۰۹/۰۷

بله اخر پستم اشاره کردم به این موضوع

نویسنده: علیرضا زهانی
تاریخ: ۰:۵۲ ۱۳۹۱/۱۰/۲۶

مرسی

بسیار مفید و کاربردی

عنوان: آشنایی با تابع EOMONTH در SQL Server

نویسنده: فرهاد فرهمندخواه

تاریخ: ۱۳:۴۸ ۱۳۹۱/۰۹/۰۶

آدرس: www.dotnettips.info

برچسب‌ها: T-SQL

گاهی اوقات لازم است، تاریخ آخرین روز ماه جاری یا دو ماه بعدتر یا یک ماه قبل‌تر و غیره... را نیاز داشته باشیم. SQL Server در نسخه 2008 خود تابعی ارائه داده است، که تاریخ آخرین روز ماه را برمی گرداند. و Syntax آن به شرح ذیل می‌باشد:

```
EOMONTH ( start_date [, month_to_add ] )
```

این تابع دو پارامتر دریافت می‌کند، اولین پارامتر یک فرمت تاریخ می‌پذیرد، دومین پارامتر، اختیاری است و یک عدد می‌پذیرد و بیانگر تعداد ماه بعد از تاریخ یا تعداد ماه قبل از تاریخ، پارامتر اول می‌باشد. با چند مثال نحوه استفاده از تابع EOMONTH را می‌آموزیم:

مثال اول:

```
SELECT EOMONTH('20110201') as 'آخرین روز ماه فوریه در سال 2011';  
SELECT EOMONTH('20120201') as 'آخرین روز ماه فوریه در سال 2012';  
SELECT EOMONTH('20130201') as 'آخرین روز ماه فوریه در سال 2013';
```

خروجی بصورت زیر می‌باشد:

Results		Messages	
	آخرین روز ماه فوریه در سال 2011		
1	2011-02-28		
	آخرین روز ماه فوریه در سال 2012		
1	2012-02-29		
	آخرین روز ماه فوریه در سال 2013		
1	2013-02-28		

مثال دوم:

یافتن آخرین روز دو ماه بعدتر از تاریخ جاری

```
SELECT EOMONTH(GETDATE(),2) as 'آخرین روز ماه ژانویه در سال 2013';
```

خروجی بصورت زیر خواهد بود:

100 %	
Results Messages	
	آخرین روز ماه ژانویه در سال 2013
1	2013-01-31

مثال سوم:

یافتن آخرین روز دو ماه قبل تر از تاریخ جاری:

```
SELECT EOMONTH(GETDATE(),-2) as 'آخرین روز ماه سپتامبر'
```

خروجی :

Results Messages	
	آخرین روز ماه سپتامبر
1	2012-09-30

موفق باشید

عنوان: مشکل ارتباط با SQL Server در لوکال

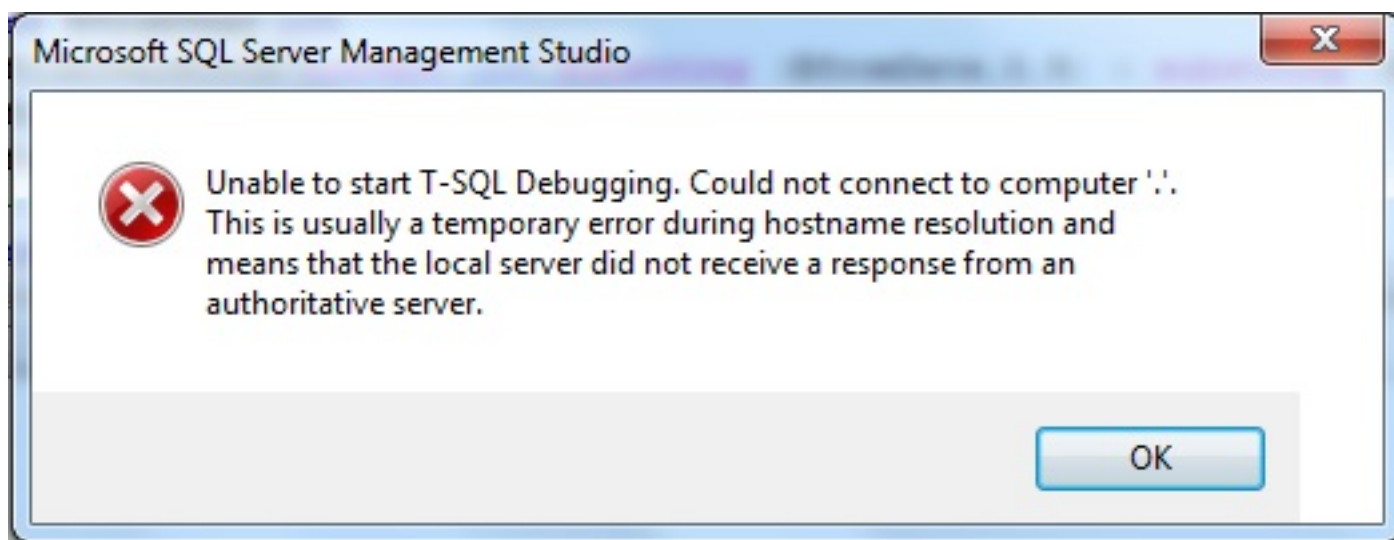
نویسنده: مهدی پایروند

تاریخ: ۱۱:۴۷ ۱۳۹۱/۰۹/۰۸

آدرس: www.dotnettips.info

برچسب‌ها: Debugging, SQL Server, T-SQL

در حین کار با SQL Server نیاز به دیباگ اسکریپتی طولانی و اورژانسی T-SQL بود که در محیط Management Studio با مشکل زیر برخورد کردم:



در این مورد نظرات و پیشنهادات زیادی از جمله ریستارت سرویس SQL Server و تعویض کلمه عبور لاگین و یا پاک کردن کلمات عبور کش شده در سیستم و حتی ریستارت کامپیوتر (: از دوستان همکار در فروم‌های موجود در اینترنت گذاشته شده بود و در گوشه ای هم اشاره به '.' شده بود که طبق عادت همیشگی برای لاگین به بانک استفاده میکردم و خواسته شده بود که برای لاگین لوکال به SQL Server از نام کامپیوتر بجای '.' استفاده شود که مفید فایده بود.

مقدمه

مقدار null به معنی پوچ و هیچ می‌باشد اما زمانی که در مقدار دهی جداول از آن استفاده می‌نمایم با توجه به نوع آن ستون فضای متفاوتی اشغال می‌نماید. شاید در پایگاه داده‌های کوچک زیاد مطرح نباشد اما زمانی که حداقل چند گیگ حجم آن باشد و فرضاً 20 تا 30 درصد آن از مقادیر null پر شده باشد فضای زیادی از پوچ گرفته شده است این در حالی است که خیلی از توسعه دهندگان اصلاً به اهمیت استفاده از null توجهی نمی‌کنند و از مقادیری غیر معتبری مثل 0 یا 1- در آن ستون به جای null استفاده می‌کنند.

SQL Server Sparce Columns

sparse column یا ستون‌های تنک قابلیتی از که از SQL Server 2008 اضافه شده و به ستون‌های عادی امکان استفاده بهینه از فضای ذخیره شده برای مقادیر null را می‌دهد. در واقع sparse column فضای مورد نیاز برای مقادیر null نسبت به مقادیر غیر null را کاهش می‌دهد. با استفاده از sparse column فضای ذخیره شده حداقل 20 تا 40 درصد کمتر خواهد شد.

ویژگی‌های Sparse Columns

SQL Server Database Engine از کلمه کلیدی SPARSE برای تعریف یک ستون که مقادیر آن می‌بایست بهینه شود استفاده می‌نماید.

نمای Catalog جداول با ستون sparse شبیه جداول معمولی می‌باشد.

مقدار برگشتی از تابع COLUMNS_UPDATED با ستون sparse متفاوت از ستون معمولی است.

در نوع داده‌های زیر امکان استفاده از sparse columns را ندارند:

text	geography
timestamp	geometry
user-defined data types	image
	ntext

sparse column فضای بیشتری برای ذخیره داده‌های غیر null نسبت به داده‌های نشانه گذاری نشده با SPARSE لازم دارد و این فضا 4 بایت بیشتر از ستون معمولی است. برآورد فضای ذخیره شده براساس نوع داده با طول ثابت در جدول زیر آورده شده است:

نوع داده	بایت بدون sparse	بایت sparse	درصد null
bit	0.125	5	98%
tinyint	1	5	86%
smallint	2	6	76%
int	4	8	64%
bigint	8	12	52%
real	4	8	64%
float	8	12	52%
smallmoney	4	8	64%
money	8	12	52%

نوع داده	بایت بدون sparse	بایت sparse	درصد null
smalldatetime	4	8	64%
datetime	8	12	52%
uniqueidentifier	16	20	43%
date	3	7	69%

نوع داده با دقت - وابسته به طول

نوع داده	بایت بدون sparse	بایت sparse	درصد null
(datetime(2	6	10	57%
(datetime(2	8	12	52%
(time(0	3	7	69%
(time(7	5	9	60%
(datetimetoffset(0	8	12	52%
(datetimetoffset (7	10	14	49%
decimal/numeric(1,s)	5	9	60%
decimal/numeric(38,s)	17	21	42%
vardecimal(p,s)			

نوع داده - داده وابسته به طول

نوع داده	بایت بدون sparse	بایت sparse	درصد null
sql_variant	2*	2*	60%
varchar or char	2*	4*+	60%
nvarchar or nchar	2*	4*	60%
varbinary or binary	2*	4*	60%
xml	2*	4*	60%
hierarchyid	2*	4*	60%

محدودیت‌های استفاده از Sparse columns

- sparse column می‌بایست nullable باشد و نمی‌تواند ROWGUIDCOL یا IDENTITY باشد.
- sparse column مقدار پیش فرض نمی‌تواند داشته باشد
- ستون محاسبه‌ای نمی‌تواند sparse باشد
- sparse column نمی‌تواند بخشی از clustered index یا unique primary key باشد
- sparse column نمی‌تواند بخشی از user-defined table باشد

مثالی از کاربرد Sparse columns

```
CREATE TABLE Employees_sparse (
    EMP_ID INT IDENTITY(5001,1) PRIMARY KEY,
    SSN CHAR(9) NOT NULL,
    TITLE CHAR(10) SPARSE NULL,
```

```

FIRSTNAME VARCHAR(50) NOT NULL,
MIDDLEINIT CHAR(1) SPARSE NULL,
LASTNAME VARCHAR(50) NOT NULL,
EMAIL CHAR(50) SPARSE NULL)
GO

```

```

CREATE TABLE Employees (
EMP_ID INT IDENTITY(5001,1) PRIMARY KEY,
SSN CHAR(9) NOT NULL,
TITLE CHAR(10) NULL,
FIRSTNAME VARCHAR(50) NOT NULL,
MIDDLEINIT CHAR(1) NULL,
LASTNAME VARCHAR(50) NOT NULL,
EMAIL CHAR(50) NULL)
GO

```

در این دو جدول یکی با سه ستون Sparse و دیگری بدون Sparse ایجاد شده و با 50000 ردیف داده پر شده است حال با رویه ذخیره شده sp_spaceused می‌توان فضای ذخیره شده دو جدول را باهم مقایسه نمود.

```

sp_spaceused 'Employees'
GO
sp_spaceused 'Employees_sparse'

```

Results		Messages				
	name	rows	reserved	data	index_size	unused
1	Employees	50000	5064 KB	5008 KB	32 KB	24 KB

	name	rows	reserved	data	index_size	unused
1	Employees_sparse	50000	3080 KB	3064 KB	16 KB	0 KB

البته ذکر این نکته گفتی است که بهتر است از این تکنیک برای جداولی که تعداد زیادی ستون null دارند استفاده شود.

بعضی وقت‌ها به هر علتی لازم است پایگاه داده و فایل هایش را تغییر نام دهیم. اگر در اینترنت جستجو کنیم روش‌های مختلفی برای تغییر نام مثل تغییر با Management Studio یا T-SQL یا روش‌های دیگری یافت می‌شود. اما اکثراً در بین انجام به مشکلی غیر قابل پیش بینی بر می‌خوریم. پایگاه داده در حالت آفلاین یا Pending قرار گرفته و به خطاهای نامفهومی بر می‌خوریم. حالا باید دوباره کلی جستجو کنیم تا مشکل بوجود آمده را حل نمائیم.

بهترین روش تغییر نام پایگاه داده

بهترین روش استفاده از قابلیت Copy Database خود SQL Server است که به راحتی این کار را برای ما انجام می‌دهد. بر روی پایگاه داده مورد نظر راست کلیک کرده و از گزینه Tasks گزینه Copy Database را انتخاب کنید. پس از ظاهر شدن پنجره کپی گزینه Next را انتخاب و در مرحله مبدا و مقصد، سرور جاری را انتخاب کنید و به مرحله بعد بروید. این برای زمانی است که شما می‌خواهید پایگاه داده را در سرور دیگری کپی نماید در پنجره Transfer Method دو روش Detach and Attach و استفاده از SQL Management Object وجود دارد که با همان روش اول به مرحله بعد بروید

در مرحله بعد نام پایگاه داده شما انتخاب شده به مرحله بعد بروید.
مرحله بعد پیکربندی پایگاه داده مقصد می‌باشد که نام و مسیر پایگاه داده جدید را می‌توانید مشخص نمایید.

Copy Database Wizard

Configure Destination Database (1 of 1)
Specify database file names and whether to overwrite existing databases at the destination.

Source database:
Test

Destination database:
Test_new

Destination database files:

Filename	Size (MB)	Destination Folder	Status
Test_new.mdf	5	C:\Program Files\Microsoft SQL Server\MSSQL11.SQL2...	OK
Test_new_log.ldf	1	C:\Program Files\Microsoft SQL Server\MSSQL11.SQL2...	OK

If the destination database already exists:

☒ Stop the transfer if a database or file with the same name exists at the destination.

☐ Drop any database on the destination server with the same name, then continue with the database transfer, overwriting existing database files.

Buttons: Help, < Back, Next >, Finish >>, Cancel

این عملیات با SQL Server Agent صورت می‌پذیرد به همین خاطر Agent می‌بایست نصب و Start شده باشد.
با انتخاب گزینه Next مراحل بعد را رد کرده تا عملیات آغاز شود.
در مرحله آخر پایگاه داده قبلی را حذف نمایید.

نظرات خوانندگان

نویسنده: اردلان شاه قلی
تاریخ: ۱۱:۵۲ ۱۳۹۲/۰۴/۰۶

خیلی متشکرم.
البته یک سوال هم دارم. تصور کنید حجم پایگاه داده‌ی من 400 گیگابایت می‌باشد و کل فضای خالی موجودم 350 گیگابایت می‌باشد در این صورت امکان استفاده از این روش وجود ندارد؟

گاهی اوقات لازم می‌باشد، در زمان Sort نمودن یک ستون، تمایل داشته باشیم Range خاصی از مقادیر آن ستون در ابتدا قرار گیرد، و عملیات Sort پس از آن Range، اعمال گردد. برای انجام چنین کاری می‌توانید از روش زیر استفاده نمایید:

برای درک مطلب مثالی می‌زنیم:

در ابتدا Script زیر را اجرا نمایید، که شامل یک جدول و درج چند رکورد در آن می‌باشد:

```
Create Table TestSort
(ID int identity(1,1),
Name nvarchar(30),
Color nvarchar(15)
)
```

درج رکورد:

```
Insert into TestSort (Name,color)
Values ('Adjustable Race',null)
,('Bearing Ball',null)
,('Headset Ball Bearings',null)
,('LL Crankarm','Black')
,('ML Crankarm','Black')
,('Chainring','Black')
,('Front Derailleur Cage','Silver')
,('Front Derailleur Linkage','Silver')
,('Lock Ring','Silver')
,('HL Road Frame - Red, 62','Red')
,('HL Road Frame - Red, 48','Red')
,('LL Road Frame - Red, 44','Red')
,('Full-Finger Gloves, M','RED')
,('Road-550-W Yellow, 38','Yellow')
,('Road-550-W Yellow, 40','Yellow')
,('Road-550-W Yellow, 42','Yellow')
,('Classic Vest, S','Blue')
,('Classic Vest, M','Blue')
,('Classic Vest, L','Blue')
```

در جدول TestSort ستونی به نام Color داریم، که نام رنگها در آن درج شده است، رنگهایی همچون ، Black Red و Silver,Blue,Yellow

در ابتدا روی ستون Color بصورت نرمال Sort صعودی انجام می‌دهیم:

```
Select t.ID,t.Name,t.Color from TestSort as t order by t.Color
```

خروجی:

	ID	Name	Color
1	1	Adjustable Race	NULL
2	2	Bearing Ball	NULL
3	3	Headset Ball Bearings	NULL
4	4	LL Crankarm	Black
5	5	ML Crankarm	Black
6	6	Chainring	Black
7	17	Classic Vest, S	Blue
8	18	Classic Vest, M	Blue
9	19	Classic Vest, L	Blue
10	10	HL Road Frame - Red, 62	Red
11	11	HL Road Frame - Red, 48	Red
12	12	LL Road Frame - Red, 44	Red
13	13	Full-Finger Gloves, M	RED
14	7	Front Derailleur Cage	Silver
15	8	Front Derailleur Linkage	Silver
16	9	Lock Ring	Silver
17	14	Road-550-W Yellow, 38	Yell...
18	15	Road-550-W Yellow, 40	Yell...
19	16	Road-550-W Yellow, 42	Yell...

مطابق شکل، زمانی که Sort بصورت صعودی است، رکوردهایی را که ستون Color آنها دارای مقدار Null می‌باشند، در ابتدای جدول قرار گرفته اند.

در ادامه می‌خواهیم، عملیات Sort ی را روی ستون Color انجام دهیم، بطوریکه تمامی رکوردهایی که مقدار ستون Color شان Red است، در ابتدای جدول قرار گیرد، و پس از آن عملیات سورت روی رنگهای دیگر اعمال شود. برای انجام چنین کاری کافست Script زیر را اجرا نمایید:

```
Select t.ID,t.Name,t.Color from TestSort as t
Where t.color is not null
order by Case t.Color
When 'Red' Then Null
Else t.color
End;
```

خروجی:

	ID	Name	Color
1	10	HL Road Frame - Red, 62	Red
2	11	HL Road Frame - Red, 48	Red
3	12	LL Road Frame - Red, 44	Red
4	13	Full-Finger Gloves, M	RED
5	4	LL Crankarm	Black
6	5	ML Crankarm	Black
7	6	Chainring	Black
8	17	Classic Vest, S	Blue
9	18	Classic Vest, M	Blue
10	19	Classic Vest, L	Blue
11	7	Front Derailleur Cage	Silver
12	8	Front Derailleur Linkage	Silver
13	9	Lock Ring	Silver
14	14	Road-550-W Yellow, 38	Yellow
15	15	Road-550-W Yellow, 40	Yellow
16	16	Road-550-W Yellow, 42	Yellow

چطور اینکار انجام شد:

اگر به Script ذکر شده دقت نمایید، در قسمت Order by اشاره کردیم، تمام مقادیر Red در ستون Color به Null تغییر کنند، بنابراین SQL Server، در ابتدا مقادیر Red را یافته آنها را به Null تغییر و سپس عملیات سورت را انجام می‌دهد، برای درک بیشتر عملیاتی را که SQL Server پشت صحنه انجام می‌دهد با Script زیر قابل شبیه سازی میباشد:

```
Select * into Simulation from
(Select t.ID,t.Name,t.Color,
Case t.Color
When 'Red' Then Null
Else t.color
End RedNull
from TestSort as t
Where t.color is not null) A
```

سپس روی ستون RedNull از جدول Simulation سورت انجام می‌دهیم:

```
Select * from Simulation order by Rednull
```

خروجی:

	ID	Name	Color	RedNull
1	10	HL Road Frame - Red, 62	Red	NULL
2	11	HL Road Frame - Red, 48	Red	NULL
3	12	LL Road Frame - Red, 44	Red	NULL
4	13	Full-Finger Gloves, M	RED	NULL
5	4	LL Crankarm	Black	Black
6	5	ML Crankarm	Black	Black
7	6	Chainring	Black	Black
8	17	Classic Vest, S	Blue	Blue
9	18	Classic Vest, M	Blue	Blue
10	19	Classic Vest, L	Blue	Blue
11	7	Front Derailleur Cage	Silver	Silver
12	8	Front Derailleur Linkage	Silver	Silver
13	9	Lock Ring	Silver	Silver
14	14	Road-550-W Yellow, 38	Yellow	Yellow
15	15	Road-550-W Yellow, 40	Yellow	Yellow
16	16	Road-550-W Yellow, 42	Yellow	Yellow

مطابق شکل، پشت صحنه SQL Server چنین کاری را انجام می‌دهد، و در زمان نمایش ستون RedNull پنهان یا حذف می‌گردد، و ستون Color، Name و ID نمایش داده می‌شود. امیدوارم مفید واقع شده باشد.

نظرات خوانندگان

نویسنده: محمد سلم آبادی
تاریخ: ۱۶:۲ ۱۳۹۱/۱۰/۲۹

سلام،
شما از اینکه مقادیر null هنگامی که مرتب سازی غیرنزولی است در ابتدا آمدند این برداشت را داشتین که مقادیر red را ابتدا به null تبدیل کنید....
اما من پیشنهاد دیگری دارم، با کمک اعداد به رنگ ها اولیت می‌دهیم سپس در اولیت‌های یکسان مرتب سازی بر اساس رنگ صورت می‌گیرد.
یعنی:

```
Select ID,t.Name,Color  
from TestSort  
Where t.color is not null  
order by case when color = 'red' then 0 else 1 end,  
color;
```


در 2012 T-SQL قابلیت صفحه بندی، نمایش خروجی یک Query فراهم گردیده است، که برای نرم افزارهای تحت وب بسیار پرکاربرد میباشد، به عنوان مثال، از جمله کاربردهای بارز آن، می‌توان به نمایش نتیجه یک جستجو بصورت صفحه بندی با تعداد رکورد محدود، اشاره نمود.

مایکروسافت برای ایجاد قابلیت صفحه بندی و محدود نمودن نمایش خروجی یک Query، تغییراتی را در Syntax مربوط به Order by ایجاد نموده است، که در ذیل مشاهده می‌نمایید:

```
ORDER BY order_by_expression
      [ COLLATE collation_name ]
      [ ASC | DESC ]
      [ ,...n ]
[ <offset_fetch> ]

<offset_fetch> ::=
{
    OFFSET { integer_constant | offset_row_count_expression } { ROW | ROWS }
    [
        FETCH { FIRST | NEXT } {integer_constant | fetch_row_count_expression } { ROW | ROWS } ONLY
    ]
}
```

OFFSET (نقطه شروع) : شامل یک پارامتر است، بطوریکه، پارامتر فوق می‌تواند یک عدد (integer_constant) یا یک عبارت (offset_row_count_expression) بپذیرد. در اینجا منظور از عبارت می‌تواند یک Subquery باشد، که خروجی آن فقط یک مقدار عددی است. یا یک متغیر و غیره...

در مورد ROW یا ROWS باید بگوییم باهم فرقی ندارند.

FETCH : همانند OFFSET شامل یک پارامتر است، و پارامتر آن می‌تواند یک عدد یا عبارت بپذیرد.

Next یا First نیز با هم تفاوتی ندارند و جهت سازگاری با ANSI می‌باشند.

OFFSET : در واقع تعداد سطر قابل حذف، پیش از نمایش اولین سطر در خروجی را بیان می‌کند.

FETCH : بیانگر تعداد رکورد قابل نمایش در یک صفحه می‌باشد.

برای درک بیشتر مثالی می‌زنیم:

ابتدا بوسیله Script زیر یک جدول ایجاد می‌نماییم، سپس چند رکورد درون آن درج می‌کنیم:

```
Create Table Testoffset
(BusinessEntityID int,
FirstName varchar(100) ,
LastName varchar(100)
);

Insert into Testoffset (BusinessEntityID,FirstName,LastName)
Values(1,'Ken','Sánchez')
,(2,'Terri','Duffy')
,(3,'Roberto','Tamburello')
,(4,'Rob','Walters')
,(5,'Gail','Erickson')
,(6,'Jossef','Goldberg')
,(7,'Dylan','Miller')
,(8,'Diane','Margheim')
,(9,'Gigi','Matthew')
,(10,'Michael','Raheem')
```

در ادامه Script زیر را اجرا نمایید، تا تعداد رکوردهای درج شده را مشاهده کنید:

	BusinessEntityID	FirstName	LastName
1	1	Ken	S?nchez
2	2	Teri	Duffy
3	3	Roberto	Tamburello
4	4	Rob	Walters
5	5	Gail	Erickson
6	6	Jossef	Goldberg
7	7	Dylan	Miller
8	8	Diane	Margheim
9	9	Gigi	Matthew
10	10	Michael	Raheem

در شکل، سه سطر (منظور رکورد 4 و 5 و 6) در کادر قرمز رنگ دیده می‌شود، می‌خواهیم Script ی ایجاد نماییم، که فقط سه سطر فوق را نمایش دهد. بنابراین خواهیم داشت:

```
SELECT BusinessEntityID, FirstName, LastName
FROM Testoffset
ORDER BY BusinessEntityID
OFFSET 3 ROWS
FETCH First 3 ROWS only
```

خروجی:

	BusinessEntityID	FirstName	LastName
1	4	Rob	Walters
2	5	Gail	Erickson
3	6	Jossef	Goldberg

اگر به Query اجرا شده دقت کنیم. در قسمت Order By جلوی Offset مقدار 3 اختصاص داده شده بود، یعنی نقطه شروع از سطر چهارم می‌باشد، به عبارت دیگر مقداری که به Offset اختصاص داده می‌شود، به SQL Server می‌فهماند، چه تعداد رکورد را نمایش ندهد. اگر شکل اول و دوم را با هم مقایسه نمایید، براحتی متوجه می‌شوید که OFFSET نقطه شروع را مشخص کرده است. مقداریکه برای Fetch در نظر گرفته شده بود برابر 3 است، که بیانگر تعداد سطر نمایش داده شده در خروجی از نقطه آغازین (offset) می‌باشد. امیدوارم مفید واقع شده باشد.

نظرات خوانندگان

نویسنده: Ara

تاریخ: ۲۲:۳۴ ۱۳۹۱/۰۹/۲۸

بعضی وقتها مایکروسافت یکسری کارها رو اونقدر سخت می کنه !

paging هم یکی از همونها بود تازه داره پخته می شه

این syntax من رو یاد mysql انداخت

```
SELECT column FROM table  
LIMIT 10 OFFSET 10
```

نویسنده: شیرزادبان

تاریخ: ۰:۱۰ ۱۳۹۱/۰۹/۲۹

سلام

می خواستم بدونم امکان هست مرتب سازی را به صورت پارامتری ارسال کنیم. یعنی می خواهیم به نوعی صفحه بندی را در وب همراه با مرتب سازی دلخواه کاربر انجام بدهم. متشکرم

نویسنده: فرهاد فرهمندخواه

تاریخ: ۸:۵ ۱۳۹۱/۰۹/۲۹

سلام

من منظور سؤال شما رو، بدرستی متوجه نشدم، به هر حال اگر بخواهید با کد نویسی سمت سرور، Script خود را Generate نمایید، اینکار، بستگی به نگرش کدنویسی تان و Interface ی که در اختیار کاربر قرار می دهید، دارد.
اگر بخواهید در SQL Server اینکار را انجام دهید، با استفاده از Case ، در قسمت Order By می توانید اینکار را انجام دهید. به عنوان مثال:

```
DECLARE @Varsort varchar(50)  
DECLARE @Varsort1 varchar(50)  
  
SET @Varsort=''  
SET @Varsort1='BusinessEntityID'  
  
SELECT BusinessEntityID, FirstName, LastName  
FROM Testoffset  
ORDER BY case when @Varsort='Firstname'then Firstname End ASC,  
           case when @Varsort1= 'BusinessEntityID'then BusinessEntityID End ASC  
OFFSET 3 ROWS  
FETCH First 3 ROWS only
```

امیدوارم پاسخ تان را گرفته باشید.

نویسنده: سعید شیرزادبان

تاریخ: ۲۰:۴۸ ۱۳۹۱/۰۹/۲۹

سلام

ممنون از جواب شما. فقط اگر تعداد ستونها بیشتر بود و یا اینکه کاربر درخواست مرتب سازی بصورت صعودی و نزولی را هم داشت (در این مثال شما به صورت پیش فرض مرتب سازی صعودی را اعمال کردید) راهکاری دارید؟
با تشکر

نویسنده: Farahmandkhah
تاریخ: ۱۳۹۱/۰۹/۳۰ ۶:۵۶

سلام

دوست عزیز، مثالی که برای شما زده شد، امکان سورت دلخواه را فراهم می‌کند، شما می‌توانید، هر کدام را به دلخواه نزولی یا صعودی نمایید، محدودیتی ندارد، برای مطالعه بیشتر می‌توانید به آدرس زیر مراجعه نمایید: [Dynamic/Conditional Order By](#)

[Clause in SQL Server/T-SQL](#)

یادآور شوم، چنانچه Performance برای شما اهمیت دارد، بهتر است از Case در Order by استفاده ننمایید، و بهتر است در زمان Run Time از طریق کد نویسی سمت سرور، Script خود را Generate نمایید، به عنوان مثال

```
SELECT BusinessEntityID, FirstName, LastName
FROM Testoffset
ORDER BY BusinessEntityID Desc,Firstname ASC
OFFSET 3 ROWS
FETCH First 3 ROWS only
```

موفق باشید.

Window Function ها برای اولین بار در نسخه SQL Server 2005 ارائه گردیدند، و در ورژن‌های جدیدتر SQL Server، به تعداد این فانکشنها افزوده شده است.

تعریف Window Function :

معمولا از این نوع فانکشنها روی مجموعه ای از ROWهای یک جدول، در جهت اعمال عملیتهای محاسباتی، ارزیابی داده ها، رتبه بندی و غیره... استفاده می‌گردد، به بیان ساده‌تر بوسیله Window Function ها می‌توان، ROWهای یک جدول را گروه بندی نمود. و روی گروه‌ها از توابع جمعی (Aggregate Functions) استفاده کرد. این نوع فانکشنها از قابلیت و انعطاف پذیری زیادی برخوردار می‌باشند، و بوسیله آنها می‌توان نتایج (خروجی) بسیار مفیدی از Query ها، بدست آورد، معمولا از این نوع فانکشنها در Data Mining (داده کاوی) و گزارشگیری‌ها استفاده می‌گردد. و آگاهی و روش استفاده از Window Function ها برای برنامه نویسان و DBA ها، می‌تواند بسیار مفید باشد.

مفهوم Window Function مطابق استاندارد ISO و ANSI می‌باشد، و دیتابیس هایی همچون Oracle, DB2, Sybase از آن پشتیبانی می‌نمایند. برای اطلاعات بیشتر می‌توانید به سایت‌های زیر مراجعه کنید: [SQL:2003](#) و [SQL:2008](#)

کلمه "Window" در Window Function، به مجموعه ROW هایی اشاره می‌کند، که محاسبات و ارزیابی و غیره... روی آنها اعمال می‌گردد.

Window Function ها برای ارائه قابلیت‌های خود، از Over Clause استفاده می‌کنند. اگر مقاله [آشنایی با Row_Number, Rank, Dense_Rank, NTILE](#) را مطالعه کرده باشید، می‌توان هریک از آنها را یک Window Function دانست. برای شروع، به بررسی Over Clause می‌پردازیم، و Syntax آن به شرح ذیل می‌باشد:

```
OVER (
    [ <PARTITION BY clause> ]
    [ <ORDER BY clause> ]
    [ <ROW or RANGE clause> ]
)

<PARTITION BY clause> ::=
PARTITION BY value_expression , ... [ n ]

<ORDER BY clause> ::=
ORDER BY order_by_expression
    [ COLLATE collation_name ]
    [ ASC | DESC ]
    [ ,...n ]

<ROW or RANGE clause> ::=
{ ROWS | RANGE } <window frame extent>

<window frame extent> ::=
{ <window frame preceding>
  | <window frame between>
}

<window frame between> ::=
BETWEEN <window frame bound> AND <window frame bound>

<window frame bound> ::=
{ <window frame preceding>
  | <window frame following>
}

<window frame preceding> ::=
{
    UNBOUNDED PRECEDING
  | <unsigned_value_specification> PRECEDING
  | CURRENT ROW
}
```

```

<window frame following> ::=
{
    UNBOUNDED FOLLOWING
    | <unsigned_value_specification> FOLLOWING
    | CURRENT ROW
}

<unsigned value specification> ::=
{ <unsigned integer literal> }

```

OVER دارای سه آرگومان اختیاری است که هر کدام را به تفصیل بررسی می‌کنیم:

1- PARTITION BY clause : بوسیله این پارامتر می‌توانیم Row های یک جدول را گروه بندی نماییم. این پارامتر یک value_expression می‌پذیرد. یک Value_expression می‌تواند نام یک ستون ، یک Scalar Function ، Scalar Subquery و غیره باشد.

2- ORDER BY clause : از نامش مشخص است و برای Sort استفاده می‌شود، و ویژگی‌های Order By در آن اعمال می‌گردد. به جز Offset.

3- ROW or RANGE clause : این پارامتر بیشتر برای محدود نمودن Row در یک Partition (گروه) مورد استفاده قرار می‌گیرد، به عنوان مثال نقطه شروع و پایان را می‌توان بوسیله پارامتر فوق تعیین نمود.

Row و Range نسبت به هم یک تفاوت عمده دارند، و آن این است که، اگر از ROW Clause استفاده نمایید، ارتباط ROW های قبلی یا بعدی، نسبت به ROW جاری، بصورت فیزیکی (physical association) سنجیده می‌شود، بطوریکه با استفاده از Range Clause ارتباط سطرهای قبلی و بعدی، نسبت به سطر جاری بصورت منطقی (logical association) در نظر گرفته می‌شود. ممکن است درک این مطلب کمی سخت باشد، در ادامه با مثالهایی که بررسی می‌نماییم، براحتی تفاوت این دو را متوجه می‌شوید.

Row یا Range در قالب‌های متفاوتی مقدار می‌پذیرند، که هر کدام را بررسی می‌کنیم:

UNBOUNDED PRECEDING : بیانگر اولین سطر Partition می‌باشد. UNBOUNDED PRECEDING فقط نقطه شروع را مشخص می‌نماید.

UNBOUNDED FOLLOWING : بیانگر آخرین سطر Partition می‌باشد. UNBOUNDED FOLLOWING فقط نقطه پایانی را مشخص می‌نماید.

CURRENT ROW : اولین سطر جاری یا آخرین سطر جاری را مشخص می‌نماید.

n PRECEDING یا <unsigned value specification> PRECEDING : تعداد سطرهای قبل از سطر جاری را تعیین می‌کند، n یا <unsigned value specification> تعداد سطرهای قبل از سطر جاری را تعیین می‌نماید. از n PRECEDING نمی‌توان برای Range استفاده نمود.

n FOLLOWING یا <unsigned value specification> FOLLOWING : تعداد سطرهای بعد از سطر جاری را تعیین می‌کند، n یا <unsigned value specification> تعداد سطرهای بعد از سطر جاری را تعیین می‌نماید. از n FOLLOWING نمی‌توان برای Range استفاده نمود.

<window frame bound> AND <window frame bound> BETWEEN : از چارچوب فوق برای Range و Row می‌توان استفاده نمود، و نقطه آغازین و نقطه پایانی توسط قالب فوق تعیین می‌گردد. نکته قابل توجه آن است که نقطه پایانی نمی‌تواند، کوچکتر از نقطه آغازین گردد.

در ادامه برای درک هرچه بیشتر تعاریف بیان شده، چندین مثال می‌زنیم و هر کدام را بررسی می‌نماییم:

در ابتدا Script زیر را اجرا نمایید، که شامل جدولی به نام Revenue (سود، درآمد) و درج چند درکورد در آن:

```

CREATE TABLE REVENUE
(
    [DepartmentID] int,
    [Revenue] int,
    [Year] int
);

insert into REVENUE
values (1,10030,1998),(2,20000,1998),(3,40000,1998),
(1,20000,1999),(2,60000,1999),(3,50000,1999),
(1,40000,2000),(2,40000,2000),(3,60000,2000),
(1,30000,2001),(2,30000,2001),(3,70000,2001)

```

مثال اول : می‌خواهیم براساس فیلد DepartmentID جدول Revenue را Partition بندی نماییم و از توابع جمعی AVG و SUM روی فیلد درآمد (Revenue) استفاده کنیم.
ابتدا Script زیر را اجرا می‌کنیم:

```
select *,
avg(Revenue) OVER (PARTITION by DepartmentID) as AverageRevenue,
sum(Revenue) OVER (PARTITION by DepartmentID) as TotalRevenue
from REVENUE
order by departmentID, year;
```

خروجی بصورت زیر خواهد بود:

	DepartmentID	Revenue	Year	AverageRevenue	TotalRevenue
1	1	10030	1998	25007	100030
2	1	20000	1999	25007	100030
3	1	40000	2000	25007	100030
4	1	30000	2001	25007	100030
5	2	20000	1998	37500	150000
6	2	60000	1999	37500	150000
7	2	40000	2000	37500	150000
8	2	30000	2001	37500	150000
9	3	40000	1998	55000	220000
10	3	50000	1999	55000	220000
11	3	60000	2000	55000	220000
12	3	70000	2001	55000	220000

مطابق شکل، جدول براساس فیلد DepartmentID به سه Partition تقسیم شده است، و عملیات میانگین و جمع روی فیلد Revenue انجام شده است و عملیات Sort روی هر گروه بطور مستقل انجام گرفته است. چنین کاری را نمی‌توانستیم بوسیله Group By انجام دهیم.

مثال دوم : نحوه استفاده از ROWS PRECEDING در این مثال قصد داریم عملیات جمع را روی فیلد Revenue انجام دهیم. بطوریکه جمع هر مقدار برابر است با سه مقدار قبلی + مقدار جاری:
لطفا رکوردهای زیر را به جدول فوق درج نمایید:

```
insert into REVENUE
values(1,90000,2002),(2,20000,2002),(3,80000,2002),
(1,10300,2003),(2,1000,2003),(3,90000,2003),
(1,10000,2004),(2,10000,2004),(3,10000,2004),
(1,20000,2005),(2,20000,2005),(3,20000,2005),
(1,40000,2006),(2,30000,2006),(3,30000,2006),
(1,70000,2007),(2,40000,2007),(3,40000,2007),
(1,50000,2008),(2,50000,2008),(3,50000,2008),
(1,20000,2009),(2,60000,2009),(3,60000,2009),
(1,30000,2010),(2,70000,2010),(3,70000,2010),
(1,80000,2011),(2,80000,2011),(3,80000,2011),
(1,10000,2012),(2,90000,2012),(3,90000,2012)
```

سپس Script زیر را اجرا می‌نماییم:

```
select Year, DepartmentID, Revenue,
sum(Revenue) OVER (PARTITION by DepartmentID ORDER BY [YEAR]
ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) as Prev3
From REVENUE order by departmentID, year;
```

خروجی :

	Year	DepartmentID	Revenue	Prev3
1	1998	1	10030	10030
2	1999	1	20000	30030
3	2000	1	40000	70030
4	2001	1	30000	100030
5	2002	1	90000	180000
6	2003	1	10300	170300
7	2004	1	10000	140300
8	2005	1	20000	130300
9	2006	1	40000	80300

در Script بالا، جدول را براساس فیلد DepartmentID گروه بندی می‌کنیم، که سه گروه ایجاد می‌شود، هر گروه را بطور مستقل، روی فیلد Year بصورت صعودی مرتب می‌نماییم. حال برای آنکه بتوانیم سیاست جمع، روی فیلد Revenue، را پیاده سازی نماییم، قطعه کد زیر را در Script بالا اضافه کردیم.

```
ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) as Prev3
```

برای شرح چگونگی استفاده از PRECEDING، فقط به شرح گروه اول بسنده می‌کنیم. مقدار جمع فیلد Revenue سطر اول، که قبل از آن سطر وجود ندارد، برابر است با مقدار خود، یعنی 10030، مقدار جمع فیلد Revenue سطر دوم برابر است با حاصل جمع مقدار فیلد Revenue سطر اول و دوم، یعنی 30030. این روند تا سطر چهار ادامه دارد، اما برای بدست آوردن مقدار جمع فیلد Revenue سطر پنجم، مقدار جمع فیلد Revenue سطر دوم، سوم، چهارم و پنجم در نظر گرفته می‌شود، و مقدار فیلد Revenue سطر اول در حاصل جمع در نظر گرفته نمی‌شود، بنابراین مقدار جمع فیلد Revenue سطر پنجم برابر است با 180000. در صورت مسئله گفته بودیم، مقدار جمع فیلد Revenue هر سطر جاری برابر است با حاصل جمع مقدار سطر جاری و مقادیر سه سطر ماقبل خود.

مثال سوم: نحوه استفاده از ROWS FOLLOWING، این مثال عکس مثال دوم است، یعنی حاصل جمع مقدار فیلد Revenue هر سطر برابر است با حاصل جمع سطر جاری با سه سطر بعد از خود. بنابراین Script زیر را اجرا نمایید:

```
select Year, DepartmentID, Revenue,
sum(Revenue) OVER (PARTITION by DepartmentID ORDER BY [YEAR]
ROWS BETWEEN CURRENT ROW AND 3 FOLLOWING) as Next3
From REVENUE order by departmentID, year;
```

خروجی :

	Year	DepartmentID	Revenue	Next3
1	1998	1	10030	100030
2	1999	1	20000	180000
3	2000	1	40000	170300
4	2001	1	30000	140300
5	2002	1	90000	130300
6	2003	1	10300	80300
7	2004	1	10000	140000
8	2005	1	20000	180000

مطابق شکل مقدار جمع فیلد اول برابر است با حاصل جمع مقدار سطر جاری و سه سطر بعد از آن. نکته ای که در مثالهای دوم و سوم، می بایست به آن توجه نمود، این است که در زمان استفاده از Row یا Range ، استفاده از Order by در Partition الزامی است، در غیر این صورت با خطا مواجه می شوید.

```

select Year, DepartmentID, Revenue,
       sum(Revenue) OVER (PARTITION by DepartmentID --ORDER BY [YEAR]
                          ROWS BETWEEN CURRENT ROW AND 3 FOLLOWING) as Next3
From REVENUE order by departmentID, year;

```

Msg 10756, Level 15, State 1, Line 2
Window frame with ROWS or RANGE must have an ORDER BY clause.

نحوه استفاده از UNBOUNDED PRECEDING ، این امکان در T-SQL Server 2012 افزوده شده است. مثال چهار: در این مثال می خواهیم کمترین سود بدست آمده در چند سال را بدست آوریم: ابتدا Script زیر را اجرا نمایید:

```

select Year, DepartmentID, Revenue,
       min(Revenue) OVER (PARTITION by DepartmentID ORDER BY [YEAR]
                          ROWS UNBOUNDED PRECEDING) as MinRevenueToDate
From REVENUE order by departmentID, year;

```

خروجی:

	Year	DepartmentID	Revenue	MinRevenueToDate
1	1998	1	10030	10030
2	1999	1	20000	10030
3	2000	1	40000	10030
4	2001	1	30000	10030
5	2002	1	90000	10030
6	2004	1	10000	10000
7	2005	1	20000	10000
8	2006	1	40000	10000
9	2007	1	70000	10000
10	2008	1	50000	10000
11	2009	1	20000	10000
12	2010	1	30000	10000
13	2011	1	80000	10000
14	2012	1	10000	10000
15	1998	2	20000	20000
16	1999	2	60000	20000

طبق تعریف UNBOUNDED PRECEDING اولین سطر هر Partition را مشخص می‌نماید، و چون از PRECEDING استفاده کرده ایم، بنابراین مقایسه همیشه بین سطر جاری و سطرها قبل از آن انجام می‌پذیرد. بنابراین خواهیم داشت، کمترین مقدار فیلد Revenue در سطر اول، برابر با مقدار خود می‌باشد، چون هیچ سطری ماقبل از آن وجود ندارد. در سطر دوم مقایسه کمترین مقدار، بین 20000 و 10030 انجام می‌گیرد، که برابر است با 10030، در سطر سوم، مقایسه بین مقادیر سطر اول، دوم و سطر سوم صورت می‌گیرد، یعنی کمترین مقدار بین 40000، 20000 و 10030، بنابراین کمترین مقدار سطر سوم برابر است با 10030. به بیان ساده‌تر برای بدست آوردن کمترین مقدار هر سطر، مقدار سطر جاری با مقادیر همه سطرها ماقبل خود مقایسه می‌گردد.

برای بدست آوردن کمترین مقدار در سطر ششم، مقایسه بین مقادیر سطرها اول، دوم، سوم، چهارم، پنجم و ششم صورت می‌گیرد که عدد 10000 بدست می‌آید و الی آخر...

نکته: اگر در Over Clause شرط Order by را اعمال نماییم، اما از Row یا Range استفاده نکنیم، SQL Server بصورت پیش فرض از قالب زیر استفاده می‌نماید:

```
RANGE UNBOUNDED PRECEDING AND CURRENT ROW
```

برای روشن‌تر شدن مطلب فوق مثالی می‌زنیم:

ابتدا Script زیر را اجرا نمایید، که شامل ایجاد یک جدول و درج چند رکورد در آن می‌باشد:

```
CREATE TABLE Employees (
    EmployeeId INT IDENTITY PRIMARY KEY,
    Name VARCHAR(50),
    HireDate DATE NOT NULL,
    Salary INT NOT NULL
)
GO
INSERT INTO Employees (Name, HireDate, Salary)
VALUES
```

```
( 'Alice', '2011-01-01', 20000),
( 'Brent', '2011-01-15', 19000),
( 'Carlos', '2011-02-01', 22000),
( 'Donna', '2011-03-01', 25000),
( 'Evan', '2011-04-01', 18500)
GO
```

سپس Script زیر را اجرا نمایید:

```
SELECT
    Name,
    Salary,
    AVG(Salary) OVER(ORDER BY HireDate) AS avgSalary
FROM Employees
GO
```

خروجی :

	Name	Salary	avgSalary
1	Alice	20000	20000
2	Brent	19000	19500
3	Carlos	22000	20333
4	Donna	25000	21500
5	Evan	18500	20900

حال اگر Script زیر را نیز اجرا نمایید، خروجی آن مطابق شکل بالا خواهد بود:

```
SELECT
    Name,
    Salary,
    AVG(Salary) OVER(ORDER BY HireDate
        RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS avgSalary
FROM Employees
GO
```

توضیح درباره Script بالا، در این روش برای بدست آوردن میانگین هر سطر، مقدار سطر جاری با مقادیر سطرهای ماقبل خود جمع و تقسیم بر تعداد سطر می‌شود.

سطر دوم $19000 + 20000$ تقسیم بر دو برابر است با 19500

میانگین سطر پنجم، حاصل جمع فیلد Salary همه مقادیر سطرها تقسیم بر 5

*** اگر بخواهید بوسیله Over Clause، میانگین همه سطرها یکسان باشد می‌توانید از Script زیر استفاده نمایید:

```
SELECT
    Name,
    Salary,
    AVG(Salary) OVER(ORDER BY HireDate
        RANGE
        BETWEEN UNBOUNDED PRECEDING
        AND UNBOUNDED FOLLOWING
    ) AS avgSalary
FROM Employees
GO
```

خروجی :

	Name	Salary	avgSalary
1	Alice	20000	20900
2	Brent	19000	20900
3	Carlos	22000	20900
4	Donna	25000	20900
5	Evan	18500	20900

منظور از ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING یعنی در محاسبه میانگین برای هر سطر تمامی مقادیر سطرهای دیگر در نظر گرفته شود.
پایان بخش اول
امیدوارم مفید واقع شده باشد.

نظرات خوانندگان

نویسنده: ناصر
تاریخ: ۱۳۹۱/۰۹/۱۸ ۰:۲۳

بسیار عالی. فقط خواستم تشکری کرده باشم (:

نویسنده: reza
تاریخ: ۱۳۹۱/۰۹/۱۸ ۰:۳۳

```
SQLQuery2.sql - (L...Er-PC\na3Er (51))* SQLQuery1.sql - (L...Er-PC\na3Er (54))* NA3ER-PC.learn - dbo.REVENUE
select Year, DepartmentID, Revenue,
sum(Revenue) OVER (PARTITION by DepartmentID ORDER BY [YEAR]
ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) as Prev3
From REVENUE order by departmentID, year;
```

Messages

Msg 102, Level 15, State 1, Line 3
Incorrect syntax near 'ROWS'.

علت این خطا چیست ؟ آیا این دستور مال 2012 هستش ؟ چون من تو 2008 تست میکنم

نویسنده: فرهاد فرهمندخواه
تاریخ: ۱۳۹۱/۰۹/۱۸ ۶:۸

سلام

Syntax ارائه شده Over Clause در SQL Server 2008 شامل دو پارامتر است :

```
Ranking Window Functions
< OVER_CLAUSE > :: =
    OVER ( [ PARTITION BY value_expression , ... [ n ] ]
    <ORDER BY_Clause> )

Aggregate Window Functions
< OVER_CLAUSE > :: =
    OVER ( [ PARTITION BY value_expression , ... [ n ] ] )
```

در SQL Server 2012 پارامتر <ROW or RANGE clause> به Syntax فوق افزوده شد. بنابراین، Query هایی که از Row یا Range

استفاده کرده اند، در SQL Server 2008 با خطا مواجه می‌شوند.

نویسنده: معتمدی
تاریخ: ۱۳۹۱/۰۹/۱۸ ۱۲:۵۹

با سلام

آیا استفاده از این query ها برای محیط‌های transactional هم مناسب است؟ یا بیشتر در database های آماری جهت تهیه گزارشات کاربرد دارد؟

از نظر زمان و هزینه‌ی اجرا و نیز تخصیص منابع سرور به آنها می‌پرسم.

نویسنده: فرهاد فرهمندخواه
تاریخ: ۱۳۹۱/۰۹/۱۸ ۱۴:۵۷

سلام

سرعت و قابلیت اجرایی Over Clause به مراتب از Group by بهتر است. بطوریکه اگر یک عملیات یکسان را، بطور جداگانه، هم با Over Clause و هم با Group By انجام دهید. و در Execution Plan مشاهده نمایید، تفاوت را حس خواهید نمود. سایت زیر یک مثال ساده در این رابطه قرار داده است: [Windowing functions - Underappreciated](#)

در مورد اینکه برای محیط‌های Transactional مناسب است یا نه، عوامل زیادی در آن دخیل است و بسته به حجم داده‌ای مورد انتظار شما در خروجی دارد، بطور مثال اگر بخواهید یک گزارش 400 صفحه‌ای ایجاد نمایید، بطور حتم در چنین محیط‌هایی هیچ Script مناسبی نیست، اما بطور قاطع می‌توان گفت که Window Function ها از کارایی بسیار خوبی برخوردار هستند، و انجام عملیات‌های پیچیده محاسباتی را برای ما آسانتر نموده‌اند.

نویسنده: فاطمه
تاریخ: ۱۳۹۱/۱۰/۲۳ ۱۷:۲۲

بسیار عالی بود
با تشکر

نویسنده: zarei
تاریخ: ۱۳۹۲/۰۲/۲۱ ۲۲:۱۴

سلام

ممنون از آموزش مفیدتون . سوال من اینه که اگه بخواهیم رکوردهای تکراری حذف بشن باید چیکار کنیم ؟ مثلا من میخام مجموع مبلغ بدهکار برای یک کد کل و در یک سند را در یک ردیف و همین مورد برای مجموع مبلغ بستانکار را نیز در یک رکورد یا ردیف دیگر بدهد . در صورتی که اگر از توابع Over() Sum استفاده کنیم به ازای هر کد کل در آن سند یک رکورد در خروجی داریم (چه بدهکار و چه بستانکار)

نویسنده: فرهاد فرهمندخواه
تاریخ: ۱۳۹۲/۰۲/۲۲ ۸:۱۹

سلام

اگر سؤال شما را درست متوجه شده باشم، فکر می‌کنم، می‌بایست از Pivot استفاده نمایید، مقاله زیر می‌تواند در درک Pivot به شما کمک نماید. [Pivot](#)

موفق باشید.

نویسنده: محمد شهریاری
تاریخ: ۹:۰ ۱۳۹۳/۰۲/۲۱

با سلام
آیا امکان استفاده از scaler function ها هم هست ؟

ممنون

نویسنده: فرهاد
تاریخ: ۱۵:۱۶ ۱۳۹۳/۰۲/۲۲

سلام
معمولا از توابع Aggregate Functions می توان در Window Function استفاده نمود: [^](#)

فرض کنید که می‌خواهیم خروجی از جدول خود را به صورت XML نمایش یا از طریق وب سرویس در برنامه مان استفاده نماییم. اولین راهی که به ذهنمان می‌رسد خودمان رشته xml را با حلقه ای ایجاد نماید یا استفاده از فضای نام System.Xml و کلاس‌های نوشته شده برای این کار. اما خود Sql Server امکانات ویژه ای برای کار با ساختار xml مهیا نموده که براحتی می‌توانید خروجی xml از داده هایتان ایجاد نمایید.

برای این کار از عبارت [For XML](#) در Select می‌توان استفاده نمود. برای مثال برای بدست آوردن ساختار ساده از For Xml Auto استفاده نمایید

```
SELECT BusinessEntityID, PersonType, Title, FirstName, MiddleName, LastName
FROM Person
WHERE BusinessEntityID = 10001
FOR XML AUTO
```

که خروجی بصورت **node attribute** زیر می‌باشد:

```
<Person.Person BusinessEntityID="10001" PersonType="IN" FirstName="Carolyn" LastName="Alonso" />
```

اما اگر بخواهیم خروجی به صورت node Elements باشد کفایت از پارامتر **Elements** استفاده نمایید

```
SELECT BusinessEntityID, PersonType, Title, FirstName, MiddleName, LastName
FROM Person
WHERE BusinessEntityID = 10001
FOR XML AUTO, ELEMENTS
```

خروجی بصورت زیر می‌باشد:

```
<Person.Person>
  <BusinessEntityID>10001</BusinessEntityID>
  <PersonType>IN</PersonType>
  <FirstName>Carolyn</FirstName>
  <LastName>Alonso</LastName>
</Person.Person>
```

اگر بخواهیم node attributes و node elements با هم ترکیب کنیم بصورت زیر عمل می‌کنیم:

```
SELECT BusinessEntityID AS '@ID', PersonType, Title, FirstName, MiddleName, LastName
FROM Person
WHERE BusinessEntityID = 10001
FOR XML ELEMENTS
```

خروجی بصورت زیر است:


```
<Person ID="10001">
  <PersonType>IN</PersonType>
  <FirstName>Carolyn</FirstName>
  <LastName>Alonso</LastName>
</Person>
```

حال می‌خواهیم همه nodeها را یک node ریشه قرار دهیم برای این کار از پارامتر ROOT در کنار AUTO به صورت زیر استفاده نماییم:

```
SELECT *
FROM Person
WHERE BusinessEntityID = 15291
FOR XML AUTO , ROOT('Persons')
```

اما اگر بخواهیم نام جدول را با نام دلخواه خود تغییر دهیم از پارامتر PATH به جای AUTO به صورت زیر استفاده نماییم:

```
SELECT *
FROM Person
WHERE BusinessEntityID = 15291
FOR XML PATH('P') , ROOT('Persons')
```

نظرات خوانندگان

نویسنده: احسان میرسعیدی
تاریخ: ۱۳۹۱/۰۹/۱۹ ۰:۲۶

بسیار تکنیک کارامدی بود. واقعا متشکرم

نویسنده: فرهاد فرهمندخواه
تاریخ: ۱۳۹۱/۰۹/۱۹ ۷:۲۹

سلام
مرسی از مطلب مفید تان

نویسنده: بهراد
تاریخ: ۱۳۹۱/۰۹/۱۹ ۱۱:۸

بسیار عالی بود ، خیلی برام مفید واقع شد
راه مشابه ای برای خروجی Json نیست؟
ممنون

نویسنده: مجتبی کاویانی
تاریخ: ۱۳۹۱/۰۹/۱۹ ۱۱:۵۷

ممنون. هنوز به صورت native نه اما از تیم microsoft در این [لینک](#) خواسته شده که For Json را هم اضافه کند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۹/۱۹ ۱۱:۵۹

« [استفاده از JSON در SQL Server](#) »

نویسنده: مهدی ناظم السادات
تاریخ: ۱۳۹۲/۰۹/۱۷ ۱۸:۵۸

حالا دستور Insert نداریم که بشه یه فایل xml رو مثل فایل backup روی دیتابیس restore کنیم؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۹/۱۸ ۰:۵۹

- می‌تونم با کدنویسی اینکار رو انجام بدم:

```
var reportData = new DataSet();
reportData.ReadXml("yourfile.xml");
var connection = new SqlConnection("DB ConnectionString");
var sbc = new SqlBulkCopy(connection);
sbc.DestinationTableName = "yourXMLTable";
```

- یا می‌تونم از [import و export](#) خود SQL Server استفاده کنی.

- و یا از [OPENXML](#) میشه استفاده کرد:

```
INSERT Customers
SELECT *
FROM OPENXML ...
```

نویسنده: ناصر نیازی
تاریخ: ۱۳۹۲/۱۱/۲۹ ۲۱:۵۱

تشکر فراوان از این مطلب فوق العاده کاربردی. یه نکته کوچک به ذهنم رسید. سومین قطعه کدی که نوشتید در جدول من کار نکرد به نظرم اومد که شاید اینچنین بوده باشه :
کد شما

FOR XML ELEMENTS

کدی که من مد نظرم هست و در جدول من کار می‌کنه :

for xml auto, ELEMENTS

قبل از مطالعه این بخش لطفاً [آشنایی با Window Function ها در SQL Server بخش اول](#) را مطالعه نمایید.

در [بخش اول](#)، در مورد Syntax مربوط به Over Clause صحبت کردیم، و برای درک استفاده از Over Clause، مثالهایی را بررسی نمودیم، در این بخش نیز، به تفاوت Row Clause و Range Clause می‌پردازیم. مثال: با ایجاد یک Script، عملیات جمع روی یک فیلد خاص، بوسیله Row Clause و Range Clause انجام می‌دهیم. تا تفاوت آنها را درک نماییم.

در ادامه Script زیر را اجرا نمایید:

```
DECLARE @Test TABLE
(
    RowID INT IDENTITY,
    FName VARCHAR(20),
    Salary SMALLINT
);
INSERT INTO @Test (FName, Salary)
VALUES ('George', 800),
('Sam', 950),
('Diane', 1100),
('Nicholas', 1250),
('Samuel', 1250),
('Patricia', 1300),
('Brian', 3000),
('Thomas', 1600),
('Fran', 2450),
('Debbie', 2850),
('Mark', 2975),
('James', 3000),
('Cynthia', 3000),
('Christopher', 5000);

SELECT RowID, FName, Salary,
       SumByRows = SUM(Salary) OVER (ORDER BY Salary ROWS UNBOUNDED PRECEDING),
       SumByRange = SUM(Salary) OVER (ORDER BY Salary RANGE UNBOUNDED PRECEDING)
FROM @Test
ORDER BY RowID;
```

خروجی بصورت زیر خواهد بود:

	RowID	FName	Salary	SumByRows	SumByRange
1	1	George	800	800	800
2	2	Sam	950	1750	1750
3	3	Diane	1100	2850	2850
4	4	Nicholas	1250	4100	5350
5	5	Samuel	1250	5350	5350
6	6	Patricia	1300	6650	6650
7	7	Brian	1500	8150	8150
8	8	Thomas	1600	9750	9750
9	9	Fran	2450	12200	12200
10	10	Debbie	2850	15050	15050
11	11	Mark	2975	18025	18025
12	12	James	3000	21025	24025
13	13	Cynthia	3000	24025	24025
14	14	Christopher	5000	29025	29025

با مشاهده شکل بالا، به وضوح می‌توان تفاوت Row و Range را تشخیص داد. در Script بالا از UNBOUNDED PRECEDING استفاده کردیم، و مفهوم قالب آن به شرح ذیل می‌باشد:

مقدار فیلد Salary سطر جاری = جمع مقادیر فیلد Salary همه سطرهای ماقبل، سطر جاری + مقدار فیلد Salary سطر جاری
Row Clause بصورت فیزیکی به سطرها می‌نگرد و قالب بیان شده در Script را، روی تمامی سطرها، نسبت به جایگاه آنها در جدول، به ترتیب اعمال می‌نماید. و در شکل نیز قابل مشاهده می‌باشد، یعنی به چیدمان سطرها در خروجی که بصورت فیزیکی نمایش داده شده است، توجه می‌کند، و حاصل جمع هر سطر برابر است با حاصل جمع سطرهای ماقبل + سطر جاری
اما Range Clause: به چیدمان فیزیکی سطرها توجه نمی‌کند، بلکه بصورت منطقی به مقدار فیلد Salary سطرها توجه می‌نماید، یعنی مقادیری که در یک محدوده (Range) قرار دارند، حاصل جمع آنها، یکی است.
مقدار فیلد Salary سطر چهار و پنج برابر است با 1250 بنابراین حاصل جمع آنها برابر هم می‌باشد. و بصورت زیر محاسبه می‌شود:

$$5350 = 1250 + 1250 + 1100 + 950 + 800$$

روش بیان شده، در مورد سطرهای 12 و 13 نیز صادق است.

امیدوارم با مثالهایی که در بخش اول و بخش دوم بررسی نمودیم، روش استفاده از Over Clause را درک کرده باشیم.

Window Function ها را به چهار بخش تقسیم بندی شده اند، که به شرح ذیل می‌باشد:

1- [Ranking functions](#) (توابع رتبه بندی)، که بررسی نمودیم.

2- [NEXT VALUE FOR](#)، که در بحث ایجاد Sequence آن را بررسی نمودیم.

3- [Aggregate Functions](#) (توابع جمعی)، اکثرا با اینگونه توابع آشنا هستیم.

4- [Analytic Functions](#) (توابع تحلیلی) که در بخش بعدی آن را بررسی می‌نماییم.

یکی از منابع بسیار مفید در مورد Window Function ها کتاب [Microsoft SQL Server 2012 High-Performance T-SQL Using](#)

[Window Functions](#)، می‌باشد، که بطور کامل به Window Function ها اختصاص دارد و تکنیک‌های بسیار مفیدی را بیان می‌کند.

مطالعه آن به علاقمندان، پیشنهاد می‌گردد.

موفق باشید.

نظرات خوانندگان

نویسنده: محمد صاحب
تاریخ: ۱۳۹۱/۰۹/۱۹ ۱۲:۲۱

دوست عزیز ممنون...
من قسمت Row و Range رو که شما توضیح دادی درست متوجه نشدم سرچی که زدم متوجه شدم این قابلیت تقریباً شبیه قسمت WITH TIES تو Select هست. برای مثال اگه بخواهیم 3 شاگرد برتر کلاس رو کوئری بنویسیم اگه تو کلاس 3 نفر معدل 18 داشته باشن (با توجه به اینکه یک معدل 20 و 19 داریم) 2 نفر از شاگردها که معدل 18 دارن تو این کوئری نمایان (TOP 3) و... برداشت منم از Range اینه که بواسطه ی برابر بودن تاریخها این 2 مقدار به هم گره خوردن و هنگام محاسبه مقدار یکسانی را تولید میکنن.

نویسنده: فرهاد فرهمندخواه
تاریخ: ۱۳۹۱/۰۹/۱۹ ۱۴:۲۱

سلام
اگر سؤال شما رو درست متوجه شده باشم، با یک مثال مفهوم Range رو بررسی می‌کنیم:

```
CREATE TABLE #Transactions
(
  AccountId INTEGER,
  TranDate DATE,
  TranAmt NUMERIC(8, 2)
);
INSERT INTO #Transactions
SELECT *
FROM ( VALUES ( 1, '2011-01-15', 50), ( 1, '2011-01-17', 500), ( 1, '2011-01-17', 500),
  ( 1, '2011-01-16', 500), ( 1, '2011-01-24', 75), ( 1, '2011-01-26', 125),
  ( 1, '2011-02-28', 500), ( 2, '2011-01-01', 500), ( 2, '2011-01-15', 50),
  ( 2, '2011-01-22', 25), ( 2, '2011-01-23', 125), ( 2, '2011-01-26', 200),
  ( 2, '2011-01-29', 250), ( 3, '2011-01-01', 500), ( 3, '2011-01-15', 50 ),
  ( 3, '2011-01-22', 5000), ( 3, '2011-01-25', 550), ( 3, '2011-01-27', 95 ),
  ( 3, '2011-01-30', 2500)
) dt (AccountId, TranDate, TranAmt);
```

روی جدول فوق دو نوع Script اجرا می‌کنیم، مثال اول، براساس AccountID جدول را گروه بندی می‌نماییم. سپس هر گروه را براساس تاریخ Sort می‌کنیم، و در هر گروه مقدار Sum آن را بدست می‌آوریم:

```
SELECT
  AccountId,
  TranDate,
  TranAmt,
  Sum(TranAmt) OVER(partition by Accountid ORDER BY TranDate RANGE UNBOUNDED PRECEDING) AS SumAmt
FROM #Transactions
GO
```

خروجی :

	AccountId	TranDate	TranAmt	SumAmt
1	1	2011-01-15	50.00	50.00
2	1	2011-01-16	500.00	550.00
3	1	2011-01-17	500.00	1550.00
4	1	2011-01-17	500.00	1550.00
5	1	2011-01-24	75.00	1625.00
6	1	2011-01-26	125.00	1750.00
7	1	2011-02-28	500.00	2250.00

مطابق شکل Sort براساس TranDate است، که چهار مقدار 500 در سه بازه تاریخی دیده می‌شود، حال محاسبه جمع هر سطر بصورت زیر است:

سطر دوم با وجود اینکه مقدار آن 500 است و در بازه تاریخی 2011-01-16 قرار دارد: مقدار آن برابر است با $500 + 50 = 550$

سطر سوم و چهارم که در بازه تاریخی 2011-01-17 می‌باشد (به عبارتی در یک محدوده می‌باشند): برابر است با :

$$500 + 500 + 500 + 50 = 1550$$

در اینجا چیزی حذف نشده، حاصل جمع سطر سوم و چهارم، چون در یک محدوده (Range) می‌باشد، برابر است با حاصل جمع سطرهای ما قبل یعنی سطر اول و دوم ($500 + 50 = 550$) + حاصل جمع تمامی سطرهای آن محدوده (Range)، یعنی سطر سوم و چهارم ($500 + 500 = 1000$)

مثال دیگر، در این حالت Sort روی فیلد TranAMT انجام می‌شود، و جدول همچنان روی فیلد AccountId گروه بندی می‌شود، بنابراین خواهیم داشت:

```
SELECT
    AccountId,
    TranDate,
    TranAmt,
    Sum(TranAmt) OVER(partition by AccountId ORDER BY TranAmt RANGE UNBOUNDED PRECEDING) AS SumAmt
FROM #Transactions
GO
```

خروجی :

	AccountId	TranDate	TranAmt	SumAmt
1	1	2011-01-15	50.00	50.00
2	1	2011-01-24	75.00	125.00
3	1	2011-01-26	125.00	250.00
4	1	2011-02-28	500.00	2250.00
5	1	2011-01-17	500.00	2250.00
6	1	2011-01-17	500.00	2250.00
7	1	2011-01-16	500.00	2250.00
8	2	2011-01-22	25.00	25.00
9	2	2011-01-15	50.00	75.00
10	2	2011-01-23	125.00	200.00

در شکل، مقدار جمع هیچ سطرى حذف نشده است، و Top ی هم در کار نیست.
حال اگر مثال فوق را روی میانگین در نظر بگیرید، باز هم تمام مقادیر، در محاسبه میانگین تاثیر گذار میباشند.

نویسنده: محمد صاحب
تاریخ: ۱۴:۵۹ ۱۳۹۱/۰۹/۱۹

ممنون...

« سطرهای فیزیکی » و « سطرهای منطقی » که شما گفتید برا من گیج کننده بود من با مثال Top ی که زدم فرق رو متوجه شدم
گفتم عنوان کنم دیگران هم استفاده کنن. البته این پست شما خیلی بهتر موضوع رو توضیح داد.

نویسنده: محمد سلم ابادی
تاریخ: ۱۲:۱۱ ۱۳۹۱/۱۰/۳۰

سلام،

ممنون از مطالب مفیدتون.

آیا دو دستور زیر با هم یکسان هستند یا خیر؟

range unbounded preceding

range between unbounded preceding and current row

و کوئری اولتون باید مساله running total باشه، که به سادگی توسط Over Clause حل شده.
کوئری زیر روشی بوده که قبل از نسخه 2012 برای حل اینگونه مسائل مورد استفاده قرار میگرفته

```
SELECT AccountId,
       TranDate,
       TranAmt,
       D.sumAmt AS older_method
       Sum(TranAmt) OVER(partition by Accountid
                        ORDER BY TranDate
                        RANGE UNBOUNDED PRECEDING) AS SumAmt
FROM #Transactions AS T1
CROSS APPLY (SELECT SUM(T2.TranAmt)
             FROM #Transactions AS T2
             WHERE T2.AccountId = T1.AccountId
             AND T2.TranDate <= T1.TranDate) AS D(SumAmt)
ORDER BY T1.AccountId, T1.TranDate;
```

نویسنده: فرهاد فرهمندخواه
تاریخ: ۱۳:۲۵ ۱۳۹۱/۱۰/۳۰

سلام

در جواب سؤال شما باید بگویم هر دو دستور یکی میباشند و هر دو از اولین سطر تا سطر جاری را در نظر میگیرند.

نویسنده: zarei
تاریخ: ۱۹:۳ ۱۳۹۲/۰۲/۲۲

سلام

من یه کوئری توسط Over Sum() .. نوشتم که تو در تو هست که ترتیب جمع دستور بیرونی برام مهمه .

```
;WITH cteBed ([Counter], id_doc , [Year] ,id_Total , date_duc ,Number_Temp , number_fix , sumbed ,
sumbes , row_no ) AS (
SELECT [Counter], d.id_doc , d.[Year] ,r.id_Total , d.date_duc ,d.Number_Temp ,d.number_fix ,
SUM( r.Mablagh_bed) OVER(PARTITION BY d.[Year] ,r.id_Total , d.Number_Temp) AS sumbed ,
sumbes= 0,
```



```

ROW_NUMBER() OVER (PARTITION BY d.[Year] ,r.id_Total , d.date_duc , d.Number_Temp , d.number_fix ORDER
BY d.date_duc )AS row_no
FROM tbl_Records r JOIN tbl_Documents d ON d.id_doc = r.id_doc ) ,

cteBes ([Counter], id_doc , [Year] ,id_Total , date_duc ,Number_Temp , number_fix , sumbed , sumbes ,
row_no) AS (
SELECT [Counter], d.id_doc , d.[Year] ,r.id_Total , d.date_duc ,d.Number_Temp ,d.number_fix , sumbed
= 0 ,
SUM( r.Mablagh_bes ) OVER(PARTITION BY d.[Year] ,r.id_Total , d.Number_Temp ) AS sumbes,
ROW_NUMBER() OVER (PARTITION BY d.[Year] ,r.id_Total , d.date_duc ,d.Number_Temp , d.number_fix ORDER
BY d.date_duc )AS row_no
FROM tbl_Records r JOIN tbl_Documents d ON d.id_doc = r.id_doc )

SELECT [Counter], id_doc , [Year] ,id_Total , date_duc ,Number_Temp , number_fix , sumbed , sumbes ,
amountBed ,amountBes
,SUM(amountBed)OVER( ORDER BY [Year] ,id_Total , date_duc , number_Temp , number_Fix ROWS BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW ) AS bed
,SUM(amountBes)OVER( ORDER BY [Year] ,id_Total , date_duc , number_Temp , number_Fix ROWS BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW ) AS bes
FROM (
SELECT [Counter], id_doc , [Year] ,id_Total , date_duc ,Number_Temp , number_fix , sumbed , sumbes ,
amountBed = CASE WHEN id_Total LIKE '1%' OR Id_Total LIKE '2%' OR Id_Total LIKE '7%' OR Id_Total
LIKE '8%' THEN (tt.sumbed-tt.sumbes) ELSE 0 END ,
amountBes=CASE WHEN Id_Total LIKE '3%' OR Id_Total LIKE '4%' OR Id_Total LIKE '5%' OR Id_Total LIKE
'6%' OR Id_Total LIKE '9%' THEN (tt.sumbes-tt.sumbed)ELSE 0 END ,
ROW_NUMBER() OVER (PARTITION BY [Year] ,id_Total , date_duc , Number_Temp , number_fix ORDER BY
date_duc )AS row_no
FROM (
SELECT * FROM cteBed cb WHERE cb.row_no = 1
UNION ALL
SELECT * FROM cteBes cb WHERE cb.row_no = 1
) AS tt ([Counter], id_doc , [Year] ,id_Total , date_duc ,Number_Temp , number_fix , sumbed ,
sumbes,row_no ) WHERE not(sumbed = 0 AND sumbes = 0)
) AS rr

```

اگرچه توپه دستور Select از Row_Number استفاده کرده باشم ، اول خروجی رو بدست میاره بعد خروجی رو بر حسب نوع مرتب سازی مربوط به Row_Number مرتب میکنه ؟ و دیگه اینکه خروجی دستور اول که شامل Row_Number هست بعد از مرتب شدن به همون صورت به دست دستور دوم (یا همون Select بیرونی) میرسه یا باز باید روی اون نیز مرتب سازی انجام بدم ؟ اصلاً جای ستونی که مربوط به Row_Number هست اول یا آخر فرق میکنه ؟

اینارو به این خاطر پرسیدم ، چون هر بار داده هام جواب متفاوتی میداد و نتونستم تشخیص بدم . ممنون

نویسنده: فرهاد فرهمندخواه
تاریخ: ۸:۱۴ ۱۳۹۲/۰۲/۲۵

سلام

جواب سوال اول: در Syntax تابع Row_Number عملیات order by اجباری است، بنابراین عملیات سورت در ابتدا انجام می‌شود و سپس Row_Number (اعداد ترتیبی) روی رکوردها اعمال می‌گردد.

در سایت مایکروسافت به خوبی اشاره شده است که هیچ تضمینی وجود ندارد، خروجی یک Query با استفاده از Row_number در هر بار اجرا، با اجرای قبلی یکی باشد مگر آنکه موارد زیر را رعایت کرده باشید:

1- مقادیر ستونی که برای قسمت Partition در نظر گرفته اید، منحصر بفرد باشد.

2- مقادیری که برای قسمت Order by در نظر گرفته اید منحصر بفرد باشد.

3- ترکیب مقادیر Partition و Order by نیز مقدار منحصر بفردی را ایجاد نماید.

جواب سوال دوم: جای ستون Row_number در زمان نمایش اهمیتی ندارد.

پیشنهاد دوستانه:

1- تاجایی که امکان دارد از OR در Query های خود استفاده ننمایید، باعث افزایش زمان اجرای Query شما می‌شود و هزینه بالایی دارد.

2- از Like نیز در نوشتن Query های خود اجتناب کنید.

برای اطلاعات بیشتر در مورد Row_Number به آدرس زیر مراجعه نمایید: [Row_Number\(\)](#)

موفق باشید.

نویسنده: zarei

تاریخ: ۱۳۹۲/۰۲/۲۷ ۱:۳۸

ممنون از راهنماییتون . ولی برای رسیدن به پاسخ راه دیگه ای به ذهنم نرسید (استفاده از OR و Like) همیشه خواهش کنم راه جایگزین رو بهم بگید ؟ آموزشی در خصوص بررسی مفهومی Plan های تولیدی SQL سراغ دارید ؟ دیگه اینکه از کجا میتونم به طور دقیق و مفهومی هزینه استفاده از دستورات SQL رو مثل OR و ... رو بخونم ؟ ممنون

نویسنده: فرهاد فرهمندخواه

تاریخ: ۱۳۹۲/۰۲/۲۷ ۲۲:۱۹

سلام

مقاله زیر به خوبی طرز استفاده از Execution Plan را آموزش می دهد. [How to read SQL Server graphical query execution plans](#)

دو کتاب زیر، جهت مطالعه و بهینه سازی در ایجاد Query مفید است: [SQL Server 2012 T-SQL Recipes](#)

[SQL Server 2012 Query Performance Tuning](#)

موفق باشید.

در مطلب قبلی با استفاده از دستور For XML خروجی xml تولید کردیم اما با همین دستور می‌توان تا حدودی خروجی Json نیز تولید نمود. البته به صورت native هنوز در sql server این امکان وجود ندارد که با رای دادن به این [لینک](#) از تیم ماکروسافت بخواهید که این امکان را در نسخه بعدی اضافه کند. برای این کار یک جدول موقت ایجاد کرده و چند رکورد در آن درج می‌کنیم:

```
declare @t table(id int, name nvarchar(max), active bit)
insert @t values (1, 'Group 1', 1), (2, 'Group 2', 0)
```

حال با استفاده از همان for xml و پارامتر type که نوع خروجی xml را خودمان می‌توانیم تعیین نماییم و پارامتر Path این کار را بصورت زیر انجام می‌دهیم:

```
select '[' + STUFF((
    select
        '{ "id":' + cast(id as varchar(max))
        + ', "name":' + name + ', "active":' + cast(active as varchar(max))
        + '}'
    from @t t1
    for xml path(''), type
).value('','varchar(max)'), 1, 1, '') + ']'
```

توجه کنید در این جا از پارامتر path بدون نام استفاده شده است و از تابع STUFF برای در یک رشته در رشته دیگر استفاده شده است. خروجی در زیر آورده شده است:

```
[{"id":1,"name":"Group 1","active":1},{"id":2,"name":"Group 2","active":0}]
```

حالت پیشرفته‌تر آن است که بتوانیم یک join را بصورت فرزندان آن در json نمایش دهیم قطعه کد زیر را مشاهده فرمایید:

```
declare @group table(id int, name nvarchar(max), active bit)
insert @group values (1, 'Group 1', 1), (2, 'Group 2', 0)

declare @member table(id int, groupid int, name nvarchar(max))
insert @member values (1, 1, 'Ali'), (2, 1, 'Mojtaba'), (3, 2, 'Hamid')

select '[' + STUFF((
    select
        '{ "id":' + cast(g.id as varchar(max))
        + ', "name":' + g.name + ', "members": { "children": [' +
        (select + STUFF((
            select
                '{ "id":' + cast(m.id as varchar(max))
                + ', "name":' + m.name + '}'
            from @member m
            where m.groupid = g.id
            for xml path(''), type
        ).value('','varchar(max)'), 1, 1, '')
        + ']} '
        + ', "active":' + cast(g.active as varchar(max))
        + '}'
    from @group g
    for xml path(''), type
).value('','varchar(max)'), 1, 1, '') + ']'
```

خروجی json بصورت زیر است:

```
[{"id":1,"name":"Group 1","members":
  { "children": [{"id":1,"name":"Ali"}, {"id":2,"name":"Mojtaba"}]}
,"active":1},
{"id":2,"name":"Group 2","members":
  { "children": [{"id":3,"name":"Hamid"}]}
,"active":0}]
```

حالت‌های خاص و پیشرفته‌تر را با امکانات t-sql خودتان می‌توانید به همین شکل تولید نمایید.

[آشنایی با Window Function ها در SQL Server بخش اول](#)

[آشنایی با Window Function ها در SQL Server بخش دوم](#)

در این بخش به دو Function از Analytic Function ها (توابع تحلیلی)، یعنی Lead Function و LAG Function می پردازیم. قبل از اینکه به توابع ذکر شده بپردازیم، باید عرض کنم، شرح عملکرد اینگونه توابع کمی مشکل می باشد، بنابراین با ذکر مثال و توضیح آنها، سعی می کنیم، قابلیت هریک را بررسی و درک نماییم.

Lead Function:

این فانکشن در SQL Server 2012 ارائه شده است، و امکان دسترسی، به Data های سطر بعدی نسبت به سطر جاری را در نتیجه یک پرس و جو (Query)، ارائه می دهد. بدون آنکه از Self-join استفاده نمایید، Syntax تابع فوق بصورت زیر است:

```
LEAD ( scalar_expression [ ,offset ] , [ default ] )
      OVER ( [ partition_by_clause ] order_by_clause )
```

شرح Syntax:

Scalar_expression: در Scalar_expression، نام یک فیلد یا ستون درج می شود، و مقدار برگشتی فیلد مورد نظر، به مقدار تعیین شده offset نیز بستگی دارد. خروجی Scalar_expression فقط یک مقدار است.

offset: منظور از Offset در این Syntax همانند عملکرد Offset در Syntax مربوط به Over می باشد. یعنی هر عددی برای offset در نظر گرفته شود، بیانگر نقطه آغازین سطر بعدی یا قبلی نسبت به سطر جاری است. به بیان دیگر، عدد تعیین شده در Offset به Sql server می فهماند چه تعداد سطر را در محاسبه در نظر نگیرد.

Default: زمانی که برای Offset مقداری را تعیین می نمایید، SQL Server به تعداد تعیین شده در Offset، سطرها را در نظر نمی گیرد، بنابراین مقدار خروجی Scalar_expression بطور پیش فرض Null در نظر گرفته می شود، چنانچه بخواهید، مقداری غیر از Null درج نمایید، می توانید مقدار دلخواه را در قسمت Default وارد کنید.

OVER ([partition_by_clause] order_by_clause): در [بخش اول](#) بطور کامل توضیح داده شده است.

برای درک بهتر Lead Function چند مثال را بررسی می نماییم:

ابتدا Script زیر را اجرا می نماییم، که شامل ایجاد یک جدول و درج 18 رکورد در آن:

```
Create Table TestLead_LAG
(SalesOrderID int not null,
 SalesOrderDetailID int not null ,
 OrderQty smallint not null);
GO
Insert Into TestLead_LAG
Values (43662,49,1),(43662,50,3),(43662,51,1),
(43663,52,1),(43664,53,1),(43664,54,1),
(43667,77,3),(43667,78,1),(43667,79,1),
(43667,80,1),(43668,81,3),(43669,110,1),
(43670,111,1),(43670,112,2),(43670,113,2),
(43670,114,1),(43671,115,1),(43671,116,2)
```

مثال: قصد داریم در هر سطر مقدار بعدی فیلد SalesOrderDetailID در فیلد دیگری به نام LeadValue نمایش دهیم، بنابراین Script زیر را ایجاد می کنیم:

```
SELECT s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty,
LEAD(SalesOrderDetailID) OVER (ORDER BY SalesOrderDetailID) LeadValue
FROM TestLead_LAG s
WHERE SalesOrderID IN (43670, 43669, 43667, 43663)
```

ORDER BY s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty

خروجی بصورت زیر خواهد بود:

	SalesOrderID	SalesOrderDetailID	OrderQty	LeadValue
1	43663	52	1	77
2	43667	77	3	78
3	43667	78	1	79
4	43667	79	1	80
5	43667	80	1	110
6	43669	110	1	111
7	43670	111	1	112
8	43670	112	2	113
9	43670	113	2	114
10	43670	114	1	NULL

مطابق شکل، براحتی واضح است، که در هر سطر مقدار بعدی فیلد SalesOrderDetailID در فیلد LeadValue درج و نمایش داده می‌شود. فقط در سطر 10، چون مقدار بعدی برای فیلد SalesOrderDetailID وجود ندارد، SQL Server مقدار فیلد LeadValue را، Null در نظر می‌گیرد.

در این مثال فقط از آرگومان Scalar_expression، استفاده کردیم، و Offset و Default را مقدار دهی ننمودیم، بنابراین SQL Server بطور پیش فرض هیچ سطر را حذف نمی‌کند و مقدار Default را Null در نظر می‌گیرد.

مثال دوم: قصد داریم در هر سطر مقدار دو سطر بعدی فیلد SalesOrderDetailID را در فیلد LeadValue نمایش دهیم، و در صورت وجود نداشتن مقدار فیلد SalesOrderDetailID، مقدار پیش فرض صفر، در فیلد LeadValue قرار دهیم، بنابراین Script آن بصورت زیر خواهد شد:

```
SELECT s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty,
LEAD(SalesOrderDetailID,2,0) OVER (ORDER BY SalesOrderDetailID) LeadValue
FROM TestLead_LAG s
WHERE SalesOrderID IN (43670, 43669, 43667, 43663)
ORDER BY s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty
```

خروجی:

	SalesOrderID	SalesOrderDetailID	OrderQty	LeadValue
1	43663	52	1	78
2	43667	77	3	79
3	43667	78	1	80
4	43667	79	1	110
5	43667	80	1	111
6	43669	110	1	112
7	43670	111	1	113
8	43670	112	2	114
9	43670	113	2	0
10	43670	114	1	0

در صورت مسئله بیان کرده بودیم، در هر سطر، مقدار فیلد SalesOrderDetailID دو سطر بعدی، را نمایش دهیم، بنابراین مقداری که برای Offset در نظر می‌گیریم، برابر دو خواهد بود، سپس گفته بودیم، چنانچه در هر سطر مقدار فیلد SalesOrderDetailID وجود نداشت، بجای مقدار پیش فرض Null، از مقدار صفر استفاده شود، بنابراین به Default مقدار صفر را نسبت دادیم.

```
LEAD(SalesOrderDetailID,2,0)
```

در شکل، مطابق صورت مسئله، مقدار فیلد LeadValue سطر اول برابر است با 78، به بیان ساده‌تر برای بدست آوردن مقدار فیلد LeadValue هر سطر، می‌بایست هر سطر را به علاوه 2 (Offset) نمایش، تا سطر بعدی بدست آید، سپس مقدار SalesOrderDetailID را در فیلد LeadValue قرار می‌دهیم. به سطر 9 و 10 توجه نمایید، که مقدار فیلد LeadValue آنها برابر با صفر است، واضح است، سطر 10 + 2 برابر است با 12 (12=2+10)، چنین سطری در خروجی نداریم، بنابراین بطور پیش فرض مقدار LeadValue توسط Sql Server برابر Null در نظر گرفته می‌شود، اما نمی‌خواستیم، که این مقدار Null باشد، بنابراین به آرگومان Default مقدار صفر را نسبت دادیم، تا SQL Server، به جای استفاده از Null، مقدار در نظر گرفته شده صفر را استفاده نماید. اگر چنین فانکشنی وجود نداشت، برای شبیه سازی آن می‌بایست از Join روی خود جدول استفاده می‌نمودیم، و یکسری محاسبات دیگر، که کار را سخت می‌نمود، مثال دوم را با Script زیر می‌توان شبیه سازی نمود:

```
WITH cteLead
AS
(
SELECT SalesOrderID,SalesOrderDetailID,OrderQty,
ROW_NUMBER() OVER (ORDER BY SalesOrderDetailID) AS sn
FROM TestLead_LAG
WHERE
SalesOrderID IN (43670, 43669, 43667, 43663)
)
SELECT m.SalesOrderID, m.SalesOrderDetailID, m.OrderQty,
case when sLead.SalesOrderDetailID is null Then 0 Else sLead.SalesOrderDetailID END as
leadvalue
FROM cteLead AS m
LEFT OUTER JOIN cteLead AS sLead ON sLead.sn = m.sn+2
ORDER BY m.SalesOrderID, m.SalesOrderDetailID, m.OrderQty
```

جدول موقتی ایجاد نمودیم، که ROW_Number را در آن اضافه کردیم، سپس جدول ایجاد شده را با خود Join کردیم، و گفتیم، که مقدار فیلد LeadValue هر سطر برابر است با مقدار فیلد SalesOrderDetailID دو سطر بعد از آن. و با Case نیز مقدار پیش فرض صفر در نظر گرفتیم.

:LAG Function

این فانکشن نیز در SQL Server 2012 ارائه شده است، و امکان دسترسی، به Dataهای سطر قبلی نسبت به سطر جاری را در نتیجه یک پرس و جو (Query)، ارائه می‌دهد. بدون آنکه از Self-join استفاده نمایید، Syntax آن شبیه به فانکشن Lead میباشد و بصورت زیر است:

```
LAG (scalar_expression [,offset] [,default])
OVER ( [ partition_by_clause ] order_by_clause )
```

Syntax مربوط به فانکشن LAG را شرح نمی‌دهم، بدلیل آنکه شبیه به فانکشن Lead می‌باشد، فقط تفاوت آن در Offset است، Offset در فانکشن LAG روی سطرهاى ماقبل سطر جاری اعمال می‌گردد. مثال دوم را برای حالت LAG Function شبیه سازی می‌نمایم:

```
SELECT s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty,
LAG(SalesOrderDetailID,2,0) OVER (ORDER BY SalesOrderDetailID) LAGValue
FROM TestLead_LAG s
WHERE SalesOrderID IN (43670, 43669, 43667, 43663)
ORDER BY s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty
go
```

خروجی :

	SalesOrderID	SalesOrderDetailID	OrderQty	LAGValue
1	43663	52	1	0
2	43667	77	3	0
3	43667	78	1	52
4	43667	79	1	77
5	43667	80	1	78
6	43669	110	1	79
7	43670	111	1	80
8	43670	112	2	110
9	43670	113	2	111
10	43670	114	1	112

همانطور که گفتیم، LAG Function عکس LEAD Function میباشد. یعنی مقدار فیلد LAGValue سطر جاری برابر است با مقدار SalesOrderDetailID دو سطر ما قبل خود. مقدار فیلد LAGValue دو سطر اول و دوم نیز برابر صفر است، چون دو سطر ماقبل آنها وجود ندارد، و مقدار صفر نیز بدلیل این است که Default را برابر صفر در نظر گرفته بودیم. مثال: در این مثال از LAG Function و Lead Function بطور همزمان استفاده می‌کنیم، با این تفاوت، که از گروه بندی نیز استفاده شده است:

Script زیر را اجرا نمایید:

```
SELECT s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty,
Lead(SalesOrderDetailID) OVER (PARTITION BY SalesOrderID ORDER BY SalesOrderDetailID) LeadValue,
LAG(SalesOrderDetailID) OVER (PARTITION BY SalesOrderID ORDER BY SalesOrderDetailID) LAGValue
FROM TestLead_LAG s
WHERE SalesOrderID IN (43670, 43669, 43667, 43663)
ORDER BY s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty
go
```


خروجی:

	SalesOrderID	SalesOrderDetailID	OrderQty	LeadValue	LAGValue
1	43663	52	1	NULL	NULL
2	43667	77	3	78	NULL
3	43667	78	1	79	77
4	43667	79	1	80	78
5	43667	80	1	NULL	79
6	43669	110	1	NULL	NULL
7	43670	111	1	112	NULL
8	43670	112	2	113	111
9	43670	113	2	114	112
10	43670	114	1	NULL	113

با بررسی هایی که در مثالهای قبل نمودیم، خروجی زیر را می توان براحتی تشخیص داد، و توضیح بیشتری نمی دهیم. موفق باشید.

نظرات خوانندگان

نویسنده: محمد

تاریخ: ۱۸:۵۳ ۱۳۹۱/۱۰/۲۸

سلام،

این توابع واقعا کار رو آسون کردن، ما رو از بکارگیری چندین بار self join بی نیاز کردن.

بطور نمونه اگه بخواهیم مقدار SalesOrderDetailID سطر قبلی، دو سطر قبلی، سطر بعدی و دو سطر بعدی را بدست بیاریم در نسخه 2008 ساده ترین و مناسب ترین کوئری این هست:

```
WITH cteLead
AS
(
SELECT SalesOrderID,SalesOrderDetailID,OrderQty,
      ROW_NUMBER() OVER (PARTITION BY SalesOrderID
                        ORDER BY SalesOrderDetailID) AS sn
FROM TestLead_LAG
WHERE
SalesOrderID IN (43670, 43669, 43667, 43663)
)
SELECT m.SalesOrderID, m.SalesOrderDetailID, m.OrderQty,
      COALESCE(sLead1.SalesOrderDetailID, 0) as leadvalue1,
      COALESCE(sLead2.SalesOrderDetailID, 0) as leadvalue2,
      COALESCE(sLag1.SalesOrderDetailID, 0) as lagvalue2,
      COALESCE(sLag2.SalesOrderDetailID, 0) as lagvalue2
FROM cteLead AS m
LEFT OUTER JOIN cteLead AS sLead1
ON m.sn = sLead1.sn - 1
AND m.SalesOrderID = sLead1.SalesOrderID
LEFT OUTER JOIN cteLead AS sLead2
ON m.sn = sLead2.sn - 2
AND m.SalesOrderID = sLead2.SalesOrderID
LEFT OUTER JOIN cteLead AS sLag1
ON m.sn = sLag1.sn + 1
AND m.SalesOrderID = sLag1.SalesOrderID
LEFT OUTER JOIN cteLead AS sLag2
ON m.sn = sLag2.sn + 2
AND m.SalesOrderID = sLag2.SalesOrderID
ORDER BY m.SalesOrderID, m.SalesOrderDetailID, m.OrderQty;
```

در حالی که با دو تابعی که شما در اینجا پوشش دادین میشه کوئری فوق را فوق العاده تر نمود:

```
SELECT s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty,
      Lead(SalesOrderDetailID, 1, 0) OVER (PARTITION BY SalesOrderID ORDER BY SalesOrderDetailID)
      LeadValue1,
      LAG(SalesOrderDetailID, 1, 0) OVER (PARTITION BY SalesOrderID ORDER BY SalesOrderDetailID)
      LAGValue1,
      Lead(SalesOrderDetailID, 2, 0) OVER (PARTITION BY SalesOrderID ORDER BY SalesOrderDetailID)
      LeadValue2,
      LAG(SalesOrderDetailID, 2, 0) OVER (PARTITION BY SalesOrderID ORDER BY SalesOrderDetailID)
      LAGValue2,
FROM TestLead_LAG s
WHERE SalesOrderID IN (43670, 43669, 43667, 43663)
ORDER BY s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty
```

نویسنده: حسین

تاریخ: ۱۴:۵۸ ۱۳۹۱/۱۰/۳۰

جالب بود مرسی

برای مطالعه این بخش لازم است، به Syntax مربوط به Over آشنا باشیم، در [بخش اول](#) بطور کامل به Syntax مربوط به Over پرداختیم.

در این بخش دو فانکشن دیگر از توابع تحلیلی (Analytic functions) به نامهای First_Value و Last_Value را بررسی می‌نماییم.

First_Value

این فانکشن نیز همانند دیگر فانکشنهای تحلیلی در نسخه SQL Server 2012 ارائه گردیده است. و اولین مقدار از یک مجموعه مقادیر را بر می‌گرداند. و Syntax آن بصورت ذیل می‌باشد:

```
FIRST_VALUE ( [scalar_expression ]
OVER ( [ partition_by_clause ] order_by_clause [ rows_range_clause ] )
```

شرح Syntax:

1- Scalar_expression : مقدار آن می‌تواند نام یک فیلد یا Subquery باشد.

2- Over : در [بخش اول](#) بطور مفصل آن را بررسی نمودیم.

قبل از بررسی تابع First_Value، ابتدا Script زیر را اجرا نمایید، که شامل یک جدول و درج چند رکورد در آن است.

```
Create Table Test_First_Last_Value
(SalesOrderID int not null,
SalesOrderDetailID int not null ,
OrderQty smallint not null);
GO
Insert Into Test_First_Last_Value
Values (43662,49,1),(43662,50,3),(43662,51,1),
(43663,52,1),(43664,53,1),(43664,54,1),
(43667,77,3),(43667,78,1),(43667,79,1),
(43667,80,1),(43668,81,3),(43669,110,1),
(43670,111,1),(43670,112,2),(43670,113,2),
(43670,114,1),(43671,115,1),(43671,116,2)
```

مثال: ابتدا Scriptی ایجاد می‌نماییم، بطوریکه جدول Test_Firts_Last_Value را براساس فیلد SalesOrderID گروه بندی نموده و اولین مقدار فیلد SalesOrderDetailID در هرگروه را مشخص نماید.

```
SELECT s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty,
FIRST_VALUE(SalesOrderDetailID) OVER (PARTITION BY SalesOrderID
ORDER BY SalesOrderDetailID) FstValue
FROM Test_First_Last_Value s
WHERE SalesOrderID IN (43670, 43669, 43667, 43663)
ORDER BY s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty
```

خروجی:

	SalesOrderID	SalesOrderDetailID	OrderQty	FstValue
1	43663	52	1	52
2	43667	77	3	77
3	43667	78	1	77
4	43667	79	1	77
5	43667	80	1	77
6	43669	110	1	110
7	43670	111	1	111
8	43670	112	2	111
9	43670	113	2	111
10	43670	114	1	111

مطابق Script چهار گروه در خروجی ایجاد شده است و در فیلد FstValue ، اولین مقدار هر گروه نمایش داده می‌شود. اگر بخش‌های قبلی Window Function ها را مطالعه کرده باشید، تحلیل این تابع کار بسیار ساده ای است.

Last_Value

این تابع نیز در نسخه SQL Server 2012 ارائه گردیده است. و آخرین مقدار از یک مجموعه مقادیر را بر می‌گرداند، به عبارتی فانکشن Last_Value عکس فانکشن First_Value عمل می‌نماید و Syntax آن به شرح ذیل می‌باشد:

```
LAST_VALUE ( [scalar_expression]
OVER ( [ partition_by_clause ] order_by_clause rows_range_clause )
```

شرح Syntax تابع Last_Value شبیه به تابع First_Value می‌باشد.

مثال: همانند مثال قبل Script یی ایجاد می‌نماییم، بطوریکه جدول Test_Firts_Last_Value را براساس فیلد SalesOrderID گروه بندی نموده و آخرین مقدار فیلد SalesOrderDetailID در هر گروه را مشخص نماید.

```
SELECT s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty,
LAST_VALUE(SalesOrderDetailID) OVER (PARTITION BY SalesOrderID
ORDER BY SalesOrderDetailID RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
LstValue
FROM Test_First_Last_Value s
WHERE SalesOrderID IN (43670, 43669, 43667, 43663)
ORDER BY s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty
```

خروجی:

	SalesOrderID	SalesOrderDetailID	OrderQty	LstValue
1	43663	52	1	52
2	43667	77	3	80
3	43667	78	1	80
4	43667	79	1	80
5	43667	80	1	80
6	43669	110	1	110
7	43670	111	1	114
8	43670	112	2	114
9	43670	113	2	114
10	43670	114	1	114

خروجی جدول، به چهار گروه تقسیم، و آخرین مقدار هر گروه، در فیلد LstValue نمایش داده شده است. در این مثال نیز تحلیلی نخواهیم داشت، چون فرض بر آن است که بخش‌های قبلی را مطالعه نموده ایم.

موفق باشید.

نظرات خوانندگان

نویسنده: محمد

تاریخ: ۱۰:۳۹ ۱۳۹۱/۱۰/۲۸

سلام،

مطلب اول: قسمت order by در ماده over در هر دو کوئری به چه جهت آمده است؟

مطلب دوم: first_value چه مزیتی نسبت به min() over دارد، منظورم اینه که میشه خروجی کوئری اولتون رو با این کوئری بدست آورد:

```
SELECT s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty,
      MIN(SalesOrderDetailID) OVER (PARTITION BY SalesOrderID) FstValue
FROM Test_First_Last_Value s
WHERE SalesOrderID IN (43670, 43669, 43667, 43663)
ORDER BY s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty
```

نویسنده: محمد

تاریخ: ۱۲:۱۱ ۱۳۹۱/۱۰/۲۸

سلام،

من SQL Server 2012 ندارم، ولی تا اونجایی که متوجه شدم بر اساس شواهد دو کوئری زیر باید یک نتیجه رو برگردانند. منظورم اینکه که با first_value همیشه last_value هم شبیه سازی کرد، فقط کافیست که در ماده order by از کلید واژه DESC استفاده بشه. اگه من اشتباه میکنم لطفا راهنمایی بفرمایید.

```
SELECT s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty,
      LAST_VALUE(SalesOrderDetailID) OVER (PARTITION BY SalesOrderID
      ORDER BY SalesOrderDetailID) LstValue
FROM Test_First_Last_Value s
WHERE SalesOrderID IN (43670, 43669, 43667, 43663)
ORDER BY s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty

SELECT s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty,
      FIRST_VALUE(SalesOrderDetailID) OVER (PARTITION BY SalesOrderID
      ORDER BY SalesOrderDetailID DESC) FstValue
FROM Test_First_Last_Value s
WHERE SalesOrderID IN (43670, 43669, 43667, 43663)
ORDER BY s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty
```

نویسنده: فرهاد فرهمندخواه

تاریخ: ۱۱:۵۳ ۱۳۹۱/۱۰/۲۹

سلام

جواب سؤال اول: در Syntax تابع First_value استفاده از Order by اجباری می باشد.

جواب سؤال دوم:

First_Value اولین مقدار یا اولین Row در یک گروه را مشخص می کند و به مفهوم کوچکترین مقدار نمی باشد، شاید، مثالی که در مقاله زدم شما را به اشتباه انداخت، در زیر با یک مثال First_value و Min را مقایسه می کنیم.

ابتدا یک جدول و چند رکورد، در آن درج می کنیم:

```
CREATE TABLE Employees (
  EmployeeId INT IDENTITY PRIMARY KEY,
  Name VARCHAR(50),
  HireDate DATE NOT NULL,
  Salary INT NOT NULL
)
```

GO

```
INSERT INTO Employees (Name, HireDate, Salary)
VALUES
    ('Alice', '2011-01-01', 20000),
    ('Brent', '2011-01-15', 19000),
    ('Carlos', '2011-02-01', 22000),
    ('Donna', '2011-03-01', 25000),
    ('Evan', '2011-04-01', 18500)
GO
```

در ادامه Script زیر را اجرا می‌کنیم:

```
Select EmployeeId, Name, Salary, HireDate,
    First_VALUE(HireDate) OVER(ORDER BY Salary RANGE BETWEEN UNBOUNDED PRECEDING
                                AND UNBOUNDED FOLLOWING) AS First,
    Min(HireDate) OVER(ORDER BY Salary
                       RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Min
FROM Employees
ORDER BY EmployeeId
GO
```

خروجی بصورت زیر می‌شود:

	EmployeeId	Name	Salary	HireDate	First	Min
1	1	Alice	20000	2011-01-01	2011-04-01	2011-01-01
2	2	Brent	19000	2011-01-15	2011-04-01	2011-01-01
3	3	Carlos	22000	2011-02-01	2011-04-01	2011-01-01
4	4	Donna	25000	2011-03-01	2011-04-01	2011-01-01
5	5	Evan	18500	2011-04-01	2011-04-01	2011-01-01

در شکل بالا تفاوت Min و First_Value بطور کامل مشخص است، اگر به Query دقت نمایید، Sort براساس Salary انجام شده است، برای حالت First_value مقدار فیلد HireDate در اولین رکورد، برابر است با 2011-04-01، بنابراین سورت روی نمایش First_value تاثیر گذار است، بطوریکه Sort برای حالت Min، تاثیر گذار نمی‌باشد، و تابع Min، کوچکترین مقدار، از مقادیر ستون HireDate را بدست می‌آورد، به بیان ساده‌تر در حالت استفاده از Min، عملیات Sort بیهوده می‌باشد. چون تابع MIN روی کل مقادیر یک گروه یا ستون تاثیر می‌گذارد.

نویسنده: فرهاد فرهمندخواه

تاریخ: ۱۳۹۱/۱۰/۲۹ ۱۲:۵۱

سلام

شما می‌توانید، با دستکاری Queryها خروجی‌های یکسانی را ایجاد نمایید، دو Query که ایجاد نمودید، خروجی یکسانی ندارند. Query دوم شما با خروجی Last_Value، مقاله یکسان است، اما باید بگویم که مفهوم Last_Value این است که آخرین سطر در یک گروه را بر می‌گرداند. بهتر است [بخش اول](#) را مطالعه نمایید.

علت یکسان نبودن نتیجه دو Query شما در نحوه Sort و مفهوم First_value و Last_Value می‌باشد:

نکته: اگر در Over Clause شرط Order by اعمال نماییم، اما از Row یا Range استفاده نکنیم، SQL Server بصورت پیش فرض از قالب زیر استفاده می‌نماید:

RANGE UNBOUNDED PRECEDING AND CURRENT ROW

نویسنده: محمد

تاریخ: ۱۳:۲۳ ۱۳۹۱/۱۰/۲۹

ممون از پاسختون، الان متوجه تفاوتشون شدم.

ستون older_method باید مقادیرش دقیقا مشابه با first_value شما باشد:

```
Select EmployeeId,Name,Salary,HireDate,
      First_VALUE(HireDate) OVER(ORDER BY Salary RANGE BETWEEN UNBOUNDED PRECEDING
                                AND UNBOUNDED FOLLOWING) AS First,
      D.HireDate AS older_method
FROM Employees
CROSS APPLY (SELECT TOP 1 HireDate
              FROM Employees
              --WHERE E1.EmployeeId = E2.EmployeeId
              ORDER BY Salary ASC) AS D
ORDER BY EmployeeId;
```

نویسنده: محمد

تاریخ: ۱۳:۳۱ ۱۳۹۱/۱۰/۲۹

ممون از شما، من مطالب بخش اول رو مطالعه کردم.

عبارت RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING به معنای تمام سطرهاى جدول هست ديگه درسته. یعنی تمام سطرهاى جدول از اولین گرفته، جاری گرفته و آخرین رو پوشش میده.

با این توضیحات باید دو کوئری زیر اینبار جواب یکسانی بدهند:

```
SELECT s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty,
      FIRST_VALUE(SalesOrderDetailID) OVER (PARTITION BY SalesOrderID
      ORDER BY SalesOrderDetailID RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
      LstValue
FROM Test_First_Last_Value s
WHERE SalesOrderID IN (43670, 43669, 43667, 43663)
ORDER BY s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty

SELECT s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty,
      LAST_VALUE(SalesOrderDetailID) OVER (PARTITION BY SalesOrderID
      ORDER BY SalesOrderDetailID DESC RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
      LstValue
FROM Test_First_Last_Value s
WHERE SalesOrderID IN (43670, 43669, 43667, 43663)
ORDER BY s.SalesOrderID,s.SalesOrderDetailID,s.OrderQty
```

دو کوئری کاملا یکسان هستند به غیر از اینکه در کوئری دوم یک DESC اضافه شده و نام تابع از first به last تغییر کرده است.

عنوان: استفاده از SQLDom برای آنالیز عبارات T-SQL

نویسنده: وحید نصیری

تاریخ: ۲۲:۱۰ ۱۳۹۱/۱۰/۰۶

آدرس: www.dotnettips.info

برچسب‌ها: SQL Server, T-SQL

به همراه بسته [Features pack](#) اس کیوال سرور 2012، دو بسته SqlDom.msi نیز وجود دارند (نسخه‌های [x86](#) و [x64](#)). این بسته حاوی اسمبلی Microsoft.SqlServer.TransactSql.ScriptDom.dll می‌باشد که نهایتاً در آدرس Program Files\Microsoft SQL Server\110\SDK\Assemblies Server\110\SDK\Assemblies کپی خواهد شد.

به کمک آن می‌توان عبارات پیچیده T-SQL را Parse و آنالیز کرد. البته باید در نظر داشت هرچند این بسته جهت SQL Server 2012 ارائه شده اما این اسمبلی با نگارش‌های 2005 به بعد اس کیوال سرور کاملاً سازگار است و اساساً نیازی هم به SQL Server ندارد. در ادامه مروری خواهیم داشت بر نحوه استفاده از آن.

یافتن کوثری‌های * Select در بین انبوهی از اسکریپت‌ها به کمک SQLDom

در مورد [مضرات کوثری‌های * select](#) پیشتر مطلبی را در این سایت خوانده‌اید. در ادامه قصد داریم به کمک امکانات اسمبلی Microsoft.SqlServer.TransactSql.ScriptDom.dll، تعدادی عبارت T-SQL را آنالیز کرده و مشخص کنیم که آیا حاوی * select هستند یا خیر. کد کامل آن‌را در ذیل مشاهده می‌کنید:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using Microsoft.SqlServer.TransactSql.ScriptDom;

namespace DbCop
{
    // Microsoft® SQL Server® 2012 Transact-SQL ScriptDom
    // SQL Server 2012 managed parser, Supports SQL Server 2005+
    // SQLDom.msi (redist x86/x64)
    // http://www.microsoft.com/en-us/download/details.aspx?id=29065
    // X86: http://go.microsoft.com/fwlink/?LinkID=239634&clcid=0x409
    // X64: http://go.microsoft.com/fwlink/?LinkID=239635&clcid=0x409
    // Program Files\Microsoft SQL
    Server\110\SDK\Assemblies\Microsoft.SqlServer.TransactSql.ScriptDom.dll

    class Program
    {
        static void Main()
        {
            const string tSql = @"
-- select * in PROCEDURE
CREATE PROCEDURE dbo.SelectStarTest
AS
SELECT * FROM dbo.tbl1
go

-- select * in PROCEDURE with TableVar
Create ProcEDURE SelectAll
AS
Declare @X table(Id integer)
Select * from @x
go

-- select * in PROCEDURE with ctex
CREATE PROCEDURE dbo.SelectAllCte
AS
WITH ctex
AS (
SELECT * FROM sys.objects
)
SELECT * FROM ctex
go

-- normal select *
select * from tbl1;
select * from dbo.tbl2;
";
```

```

        IList<ParseError> errors;
        TSqLScript sqlFragment;
        using (var reader = new StringReader(tSql))
        {
            var parser = new TSqL110Parser(initialQuotedIdentifiers: true);
            sqlFragment = (TSqLScript)parser.Parse(reader, out errors);
        }

        if (errors != null && errors.Any())
        {
            var sb = new StringBuilder();
            foreach (var error in errors)
                sb.AppendLine(error.Message);

            throw new InvalidOperationException(sb.ToString());
        }

        var i = 0;
        foreach (var batch in sqlFragment.Batches)
        {
            Console.WriteLine("Batch: {0}, Statement(s): {1}", ++i, batch.Statements.Count);
            foreach (var statement in batch.Statements)
            {
                processStatement(statement);
            }
            Console.WriteLine();
        }

        Console.WriteLine("\nPress a key...");
        Console.Read();
    }

    private static void processStatement(TSqLStatement statement)
    {
        var createProcedureStatement = statement as CreateProcedureStatement;
        if (createProcedureStatement != null)
        {
            var statementList = createProcedureStatement.StatementList;
            foreach (var procedureStatement in statementList.Statements)
            {
                processStatement(procedureStatement);
            }
        }

        var selectStatement = statement as SelectStatement;
        if (selectStatement != null)
        {
            var query = selectStatement.QueryExpression;
            var selectElements = ((QuerySpecification)query).SelectElements;
            foreach (var selectElement in selectElements)
            {
                var expression = selectElement as SelectStarExpression;
                if (expression == null) continue;
                Console.WriteLine(
                    "`Select *` detected @StartOffset:{0}, Line:{1}, T-SQL: {2}",
                    expression.StartOffset,
                    expression.StartLine,
                    statementToString(selectStatement));
            }
        }
    }

    private static string statementToString(TSqLStatement selectStatement)
    {
        var text = new StringBuilder();
        for (var i = selectStatement.FirstTokenIndex; i <= selectStatement.LastTokenIndex; i++)
        {
            text.Append(selectStatement.ScriptTokenStream[i].Text);
        }
        return text.ToString();
    }
}

```

توضیحات:

پس از نصب SQLDom.msi، ارجاعی را به اسمبلی زیر اضافه نمایید تا بتوانید کد فوق را کامپایل کنید:

Program Files\Microsoft SQL Server\110\SDK\Assemblies\Microsoft.SqlServer.TransactSql.ScriptDom.dll

کار با ایجاد وهله‌ای از TSql110Parser شروع می‌شود. متد Parse آن، آرگومانی از نوع TextReader را قبول می‌کند. برای مثال با استفاده از StringReader می‌توان محتوای یک متغیر رشته‌ای را به آن ارسال کرد و یا توسط StreamReader یک فایل sql را. پس از فراخوانی متد Parse، بهتر است بررسی شود که آیا عبارت T-SQL دریافتی معتبر بوده است یا خیر. اینکار را توسط لیستی از ParseErrorهای دریافتی می‌توان انجام داد.

خروجی متد Parse، حاوی یک سری Batch آنالیز شده است. هر عبارت Go در اینجا یک Batch را تشکیل می‌دهد. سپس در داخل هر batch به دنبال batch.Statements گشت تا بتوان به عبارات T-SQL آن‌ها دسترسی یافت.

در ادامه کار اصلی توسط متد processStatement صورت می‌گیرد. عبارات دریافتی، در حالت کلی از نوع TSqlStatement هستند اما در اصل می‌توانند یکی از مشتقات آن نیز باشند. در اینجا فقط دو مورد CreateProcedureStatement و SelectStatement بررسی شده‌اند (مطابق رشته tSql ابتدای مثال). هر دو عبارت، از کلاس TSqlStatement مشتق شده‌اند.

در متد processStatement عبارات select معمولی و همچنین آن‌هایی که داخل رویه‌های ذخیره شده تعریف شده‌اند، استخراج شده و در نهایت بررسی می‌شوند که آیا از نوع SelectStarExpression هستند یا خیر (همان * select صورت مساله). خروجی مثال فوق به شرح زیر است:

```
Batch: 1, Statement(s): 1
`Select *` detected @StartOffset:140, Line:5, T-SQL: SELECT * FROM dbo.tbl1

Batch: 2, Statement(s): 1
`Select *` detected @StartOffset:368, Line:12, T-SQL: Select * from @x

Batch: 3, Statement(s): 1
`Select *` detected @StartOffset:659, Line:22, T-SQL: WITH ctex
      AS (
        SELECT * FROM sys.objects
      )
      SELECT * FROM ctex

Batch: 4, Statement(s): 2
`Select *` detected @StartOffset:753, Line:26, T-SQL: select * from tbl1;
`Select *` detected @StartOffset:791, Line:27, T-SQL: select * from dbo.tbl2;
```

عنوان: محاسبه تعداد تکرار یک کلمه در یک رشته

نویسنده: محمد سلیم آبادی

تاریخ: ۱۳۹۱/۱۰/۲۹ ۱۶:۴۰

آدرس: www.dotnettips.info

برچسب‌ها: T-SQL

گاهی در راه حل‌هایمان نیاز داریم که تعداد تکرار یک کلمه در یک رشته را بدست آوریم. مثلاً در عبارت "محمد محمد علی محمد محمد علی رضا جواد" کلمه محمد 4 بار تکرار شده و کلمه علی 2 دفعه. هدف ما پیدا کردن این اعداد هست.

برای بدست آوردن تعداد تکرار یک کلمه در یک رشته مراحل زیر را طی کنید:

1- اگر در رشته بین کلمات تنها یک فاصله بود آن را به دو فاصله تبدیل کنید.

2- اگر ابتدا و انتهای رشته فاصله وجود نداشت فاصله اضافه کنید.

3- طول رشته بعد از حذف کلمات مشابه کلمه مورد نظر را از طول رشته تفریق کنید و عدد حاصل را تقسیم به طول کلمه مورد نظر کنید

```
DECLARE @S VARCHAR(50) = 'ali ali ali ali hasan reza ali javad reza reza'
```

```
SELECT D.value, E.cnt
FROM (VALUES ('ali'),
             ('hasan'),
             ('reza'),
             ('javad')) AS D(value)
CROSS APPLY
(SELECT ' ' + REPLACE(@s, ' ', ' ') + ' ') AS C(String)
CROSS APPLY
(SELECT (LEN(String) - LEN(REPLACE(String, ' ' + D.value + ' ', ''))) / (LEN(D.value) + 2)) AS
E(cnt)
ORDER BY cnt DESC;
```

در اسکریپت فوق تعداد تکرار شدن برخی از کلمات موجود در رشته مذکور مشخص خواهد شد. خروجی کوئری فوق برابر است با:

	value	cnt
1	ali	5
2	reza	3
3	javad	1
4	hasan	1

نظرات خوانندگان

نویسنده: سعید
تاریخ: ۱۷:۲ ۱۳۹۱/۱۰/۲۹

به نظر من راه بهتر و عمومی حل این نوع مسایل استفاده از تابع [split](#) یا [tokenizer](#) هست بعد پردازش روی نتیجه نهایی.

نویسنده: محمد سلم آبادی
تاریخ: ۱۸:۵۰ ۱۳۹۱/۱۰/۲۹

Split کردن به عنوان روش کلی خیلی خوب، اما وقتی مساله رو میشه بدون درگیر شدن با بحث Splitting بسادگی حل نمود چرا باید سراغ روش پر هزینه تر رفت؟! فقط توجه داشته باشید که ما قصد داریم تنها تکرار یک کلمه را بررسی کنیم نه تمام کلمات. بله اگه قرار بر این باشد که تمام کلمات را تفکیک شده داشته باشیم و تعداد آنها را بدست آوریم ناچارا باید سراغ splitting رفت.

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

فرض کنید می‌خواهیم سطرهای جدول را 6 تا 6 تا سوا کنیم و به هر کدام یک عددی انتساب دهیم و هر قسم تولید شده را نیز 2 تا 2 تا سوا کنیم و بهش عدد انتساب دهیم.

به تصویر زیر توجه بفرمایید. ابتدا داده‌ها به دو دسته شش‌تایی تقسیم شدن (ستون ntl)، سپس هر کدام از این دسته‌ها نیز به سه دسته دوتایی تقسیم شدن (ستون grp) هدف ما تولید دو ستون ntl و grp توسط query می‌باشد.

	nbr	ntl	grp
1	1	1	1
2	2	1	1
3	3	1	2
4	4	1	2
5	5	1	3
6	6	1	3
7	7	2	1
8	8	2	1
9	9	2	2
10	10	2	2
11	11	2	3
12	12	2	3

برای بدست آوردن مقادیر دو ستون مذکور روش‌های متنوعی وجود دارد که برخی از آنها را در اینجا پوشش میدهم. قبل از هر چیزی ابتدا جدول را ایجاد و 12 سطر زیر را در آن انتشار دهید:

```
CREATE TABLE T (nbr INT NOT NULL);
INSERT T VALUES (1), (2), (3), (4), (5), (6),
(7), (8), (9), (10), (11), (12);
```

روش اول:

این روش، تعمیم پذیری و پویایی ندارد و برای هر سناریویی مناسب نخواهد بود. ولی از آنجایی که دیدم کوئری زیر میتواند یک نمونه از کاربرد Ntile باشه آن را مطرح کردم.

```
SELECT nbr, ntl, NTILE(3) OVER(PARTITION BY ntl ORDER BY nbr) AS grp
FROM (
    SELECT nbr, NTILE(2) OVER(ORDER BY nbr) ntl
    FROM T
) AS D;
```

تابع ntile داخلی سطرهای جدول را به دو قسم تقسیم می‌کند و برای قسم اول عدد 1 و برای قسم دوم عدد 2 را در نظر می‌گیرد.

تابع ntile بیرونی بر اساس دو عدد 1 و 2 گروه بندی انجام داده و هر گروه را به 3 قسمت تقسیم می‌کند. قسمت اول 1، دوم 2 و

سوم 3 خواهد بود.

لازم به ذکر است که باید خارج قسمت تقسیم تعداد سطرها بر عدد n_{tile} یک عدد صحیح باشد تا خروجی مناسب داشته باشیم. و همچنین بایستی بدانیم که تعداد سطرهای جدول چنانست تا آن را به گونه ای تقسیم کنیم که خارج قسمت برابر شود با عدد مورد نظر ما یعنی 6.

روش دوم:

در این روش بر خلاف روش قبل که همه چیز توسط تابع بدست می‌آید باید خودمان دست به کار شویم و فرمولی را بدست آوریم که نتیجه مورد نظر را تولید کند.

برای حل این مساله ابتدا باید سطرهای جدول را 6 تا 6 تا سوا کنیم و عناصر هر دسته را شماره گذاری کنیم (از 1 تا 6 بر اساس ترتیب مقدار n_{br}) سپس با کمک سایر فرمول‌ها دسته‌ها را دوتا دوتا شماره گذاری میکنیم.

به تصویر زیر توجه بفرمایید:

	nbr	mk1	mk2	grp2	grp2
1	1	1	1	1	1
2	2	2	2	1	1
3	3	3	3	2	2
4	4	4	4	2	2
5	5	5	5	3	3
6	6	6	6	3	3
7	7	1	1	1	1
8	8	2	2	1	1
9	9	3	3	2	2
10	10	4	4	2	2
11	11	5	5	3	3
12	12	6	6	3	3

در کادر نارنجی رنگ همانطور که اشاره شد ما سطرهای شمارگذاری شده ای داریم که در رنج 1 تا 6 هستند. و در کادر بنفش ستون مورد نظر ما قرار دارد. ستون بنفش با کمک ستون نارنجی بدست آمده است. اگر تقسیم صحیح را div و باقیمانده صحیح را mod بگیریم فرول‌های مورد نظر به این شرح خواهد بود:

$$(nbr - 1) \bmod 6 + 1$$

$$\left(((nbr - 1) \bmod 6 + 1) + 1 \right) \div 2$$

طبق کوئری زیر ستون نارنجی (rnk1/rnk2) را به دو طریق می توان ایجاد نمود و ستون بنفش (grp1/grp2) را نیز به دو طریق میتوان ایجاد نمود.

```
SELECT nbr, rnk1, rnk2,
       (rnk1 + 1) / 2 AS grp2,
       (rnk1 - 1) / 2 + 1 AS grp2
FROM
(
  SELECT nbr,
         ROW_NUMBER() OVER(PARTITION BY (nbr + 5) / 6 ORDER BY nbr) rnk1,
         (nbr - 1) % 6 + 1 AS rnk2
  FROM t
)d
```

حذف نمودن کاراکترهای ناخواسته توسط Recursive CTE قسمت اول

عنوان:

نویسنده: محمد سلیم آبادی

تاریخ: ۱۵:۳۵ ۱۳۹۱/۱۱/۰۱

آدرس: www.dotnettips.info

برچسب‌ها: SQL Server, T-SQL, recursive cte, replace

شاید برایتان تا حالا پیش آمده باشد که بخواهید یکسری کاراکترهای ناخواسته و اضافه را از یک رشته حذف کنید. بطور مثال تمام کاراکترهایی غیر عددی را باید از یک رشته حذف نمود تا آن رشته قابلیت تبدیل به نوع integer را بدست بیاورد.

اگر تعداد کاراکترهای ناخواسته محدود و مشخص هستند می‌توانید با دستور REPLACE آنها را حذف کنید، مثلاً می‌خواهیم هر سه کاراکتر ~!@ از رشته حذف شوند:

```
DECLARE @s VARCHAR(50) = '~~~~~!@@@@@ salam';
SET @s = REPLACE(REPLACE(REPLACE(@s, '~', ''), '!', ''), '@', '');
SELECT @s AS new_string
```

ولی هنگامی که کاراکترها نامحدود بوده امکان نوشتن تابع REPLACE به کرات بی معنا است در این حالت باید دنبال روشی پویا و تعمیم پذیر بود.

با جستجویی که در اینترنت انجام دادم متوجه شدم تکنیک WHILE یا همون loop یکی از روش‌های رایج برای انجام اینکار هست، که احتمالاً به دلیل سهولت در بکارگیری و سادگی آن بوده که عمومیت پیدا کرده است. مستقل از این صحبت‌ها هدف معرفی یک روش مجموعه گرا (set-based) برای این مساله می‌باشد.

حذف کاراکترها ناخواسته با تکنیک Recursive CTE

راه حل بر اساس جدول زیر است:

```
CREATE TABLE test_string
(id integer not null primary key,
 string_value varchar(500) not null);

INSERT INTO test_string
VALUES (1, '##### salam 12345'),
(2, 'good $$$$ &&&& bye 00000');
```

حالا فرض کنید می‌خواهیم هر کاراکتری غیر از حروف الفبای انگلیسی و فاصله(space) از رشته حذف شود. پس دو داده فوق به صورت salam و good bye در انتها در خواهند آمد. برای حذف کاراکترهای ناخواسته فوق query زیر را اجرا کنید.

```
WITH CTE (ID, MyString, Ix) AS
(
    SELECT id,
           string_value,
           PATINDEX('%[^a-z ]%', string_value)
    FROM   test_string

    UNION ALL

    SELECT id,
           CAST(REPLACE(MyString, SUBSTRING(MyString, Ix, 1), '') AS VARCHAR(500)),
           PATINDEX('%[^a-z ]%', REPLACE(MyString, SUBSTRING(MyString, Ix, 1), ''))
    FROM   CTE
    WHERE  Ix > 0
)
SELECT *
FROM     cte
--WHERE  Ix = 0;
ORDER BY id, CASE WHEN Ix = 0 THEN 1 ELSE 0 END, Ix;
```

توضیح query:

در قسمت anchor اندیس اولین کاراکتر ناخواسته (خارج از رنج حروف الفبا و فاصله) بدست می‌آید. سپس در قسمت recursive هر کاراکتری که برابر باشد با کاراکتر ناخواسته ای که در مرحله قبل بدست آمده از رشته حذف می‌شود این عملیات توسط تابع replace صورت می‌گیرد و اندیس کاراکتر ناخواسته بعدی بعد از حذف کاراکتر ناخواسته قبلی بدست می‌آید که به مرحله بعد منتقل می‌شود. این مراحل تا آنجایی پیش می‌رود که دیگر کاراکتر ناخواسته ای در رشته وجود نداشته باشد.

به جدول زیر توجه بفرمایید (خروجی query فوق)

	ID	MyString	lx
1	1	@@@@ ##### salam 12345	1
2	1	##### salam 12345	2
3	1	salam 12345	9
4	1	salam 2345	9
5	1	salam 345	9
6	1	salam 45	9
7	1	salam 5	9
8	1	salam	0
9	2	good \$\$\$\$ &&&& bye 00000	6
10	2	good &&&& bye 00000	7
11	2	good bye 00000	12
12	2	good bye	0

نتیجه مطلوب ما آن دو سطری است که در کادر بنفش هستند. که اگر به ستون lx اشان توجه کنید مقدارش برابر با 0 است.

لطفا به سطر اول جدول توجه بفرمایید مشاهده می‌شود که هر 4 کاراکتر @ یکبار از رشته حذف شدند که بدلیل استفاده از تابع REPLACE میباشد.

نظرات خوانندگان

نویسنده: سید حمزه
تاریخ: ۱۶:۵۳ ۱۳۹۲/۰۶/۲۷

سلام
خیلی جامع بود
فقط من متوجه نشدم دقیقا کجای کوئریم باید اینو بنویسم.
و همینطور میخواستم اگر بشه یک فانکشن بسازم و از اون هم توی select استفاده کنم.
ممنون

syntax کامل LIKE:

```
match_expression [ NOT ] LIKE pattern [ ESCAPE escape_character ]
```

چهار کاراکتر وجود دارد که الگوهای جستجو را توسط آنها تعریف می‌کنند. این کاراکترها wildcard نام دارند. زمانی که کاراکترهای wildcard موجود در الگوی ما به عنوان کاراکترهای عادی در نظر گرفته شده اند نه wildcard آنگاه باید به SQL اطلاع دهیم که آنها را با کاراکترهای wildcard اشتباه نگیرد، برای این منظور چندین راه حل وجود دارد که در ادامه شرح خواهیم داد.

فرض کنید الگوی شما این مقدار باشد:
"تنها دو کاراکتر % کاراکترهای wildcard هستند"

```
%[abcde_ ]%
```

روش اول: ESCAPE

```
WHERE string_value LIKE '%@[abcde@_ ]%' ESCAPE '@'
```

دو کاراکتر [و _ علامت گذاری (بوسیله @) و توسط ماده ESCAPE به عنوان کاراکترهای معمولی شناخته شدند.

روش دوم: Bracket

```
WHERE string_value LIKE '%[[]abcde[_ ]%'
```

دو کاراکتر [و _ را داخل [] قرار دادیم.

روش سوم: CHARINDEX

```
WHERE CHARINDEX('[abcde_ ]', string_value) > 0
```

عبارت را بدون هیچ گونه تغییری در تابع استفاده کردیم. این روش تنها زمانی مورد استفاده می‌تواند قرار گیرد که کاراکتر % در ابتدا یا انتهای (یا هر دو) آن قرار گرفته باشد.

تمرین: عباراتی را پیدا کنید که الگوی زیر در آن پیدا شود، فقط آخرین کاراکتر wildcard می‌باشد یعنی کاراکتر %

```
[[[%^ab'c'de]]]%
```

برای تست راه حلتان از متغیر جدولی زیر استفاده کنید:

```
declare @t table (s varchar(50))
insert @t values ('[[[%^ab'c'de]]]'),
('[[[%^ab'c'de]]]'), ('[[[%^ab'c'de]]]'),
('[[[%^abc'de]]]');
```


در این بخش فانکشن دیگری از توابع تحلیلی به نام CUME_DIST را بررسی می‌نماییم.

CUME_DIST

بوسیله تابع CUME_DIST می‌توان ارزیابی نمود، در یک گروه، چه درصد از مقادیر، مساوی یا کوچکتر از مقدار سطر جاری می‌باشند، به این تابع cumulative distribution نیز گفته می‌شود.
Syntax تابع CUME_DIST به صورت زیر است:

```
CUME_DIST( )  
OVER ( [ partition_by_clause ] order_by_clause )
```

شرح Syntax:

1- Partition By Clause : بوسیله پارامتر فوق می‌توانید، نتیجه پرس جو (Query)، خود را دسته بندی نمایید.

2- order by clause : همانطور که از نامش مشخص است، جهت مرتب نمودن خروجی Query می‌باشد.

معمولا شرح عملکرد توابع تحلیلی، کمی مشکل است. بنابراین برای درک، عملکرد تابع CUME_DIST چند مثال را بررسی می‌کنیم.
در ابتدا بوسیله Script زیر یک جدول ایجاد و 10 رکورد در آن درج می‌کنیم:

```
Create Table TestCUME_DIST  
(SalesOrderID int not null,  
OrderQty smallint not null,  
ProductID int not null  
);  
GO  
Insert Into TestCUME_DIST  
Values (43663,1,760),(43667,3,710),(43667,1,773),  
(43667,1,775),(43667,1,778),(43669,1,747),  
(43670,1,709),(43670,2,710),(43670,2,773),(43670,1,776)
```

مثال اول: Script زیر را اجرا می‌کنیم، سپس خروجی آن را بررسی می‌نماییم:

```
SELECT SalesOrderID, OrderQty,  
CUME_DIST() OVER(ORDER BY SalesOrderID) AS [CUME_DIST]  
FROM TestCUME_DIST ORDER BY [CUME_DIST] DESC
```

پس از اجرا خروجی بصورت زیر خواهد بود:

	SalesOrderID	OrderQty	CUME_DIST
1	43670	1	1
2	43670	2	1
3	43670	2	1
4	43670	1	1
5	43669	1	0.6
6	43667	3	0.5
7	43667	1	0.5
8	43667	1	0.5
9	43667	1	0.5
10	43663	1	0.1

در ادامه اجازه دهید، مقادیری که در فیلد CUME_DIST بدست آمده است را بصورت تصویری بررسی کنیم.
مقادیر سطر اول تا چهارم:

	SalesOrderID	OrderQty	CUME_DIST
1	43670	1	1
2	43670	2	1
3	43670	2	1
4	43670	1	1
5	43669	1	0.6
6	43667	3	0.5
7	43667	1	0.5
8	43667	1	0.5
9	43667	1	0.5
10	43663	1	0.1

سوال اول: چند سطر، SalesOrderID آن برابر ۴۳۶۷۰ می باشد؟

جواب: چهار سطر (Row)، فرض می کنیم $C1=4$

سوال دوم: چند سطر (Row)، زیر SalesOrderID=43670 قرار دارد؟

جواب: شش سطر (Row)، فرض می کنیم $C2=6$

سوال سوم: تعداد کل سطرها (Rows)، چند عدد می باشد؟

جواب: ۱۰ سطر (Rows)، فرض می کنیم $C3=10$

برای بدست آوردن CUME_DIST چهار سطر اول از فرمول زیر پیروی می نماییم:

$$\text{Rows} = (C1+C2)/C3 = (4+6)/10 = 1$$

*** برای بدست آوردن CUME_DIST سطر پنجم نیز خواهیم داشت:

$$\text{Rows} = (c1+c2)/c3 \text{ بنابراین خواهیم داشت: } \text{Rows} = (1+5)/10 = 6/10$$

مثال دوم : ابتدا Script زیر را اجرا نمایید:

```
SELECT SalesOrderID, OrderQty, ProductID,
       CUME_DIST() OVER(PARTITION BY SalesOrderID ORDER BY ProductID ) AS [CUME_DIST]
FROM TestCUME_DIST
WHERE SalesOrderID IN (43670, 43669, 43667, 43663)
ORDER BY SalesOrderID DESC, [CUME_DIST] DESC
```

خروجی :

	SalesOrderID	OrderQty	ProductID	CUME_DIST
1	43670	1	776	1
2	43670	2	773	0.75
3	43670	2	710	0.5
4	43670	1	709	0.25
5	43669	1	747	1
6	43667	1	778	1
7	43667	1	775	0.75
8	43667	1	773	0.5
9	43667	3	710	0.25
10	43663	1	760	1

همانگونه که ملاحظه می‌کنید، در این مثال، خروجی، براساس SalesOrderID به چهار گروه تقسیم می‌شود و عملیات مرتب سازی روی فیلد ProductID انجام می‌گیرد، بنابراین CUME_DIST، روی هر گروه بر روی فیلد ProductID محاسبه می‌شود.

گروه اول : نحوه محاسبه Cume_DIST سطر اول:

سوال: چه تعداد از مقادیر ProductID آن برابر 776 می‌باشد؟

جواب: فقط مقدار سطر اول، بنابراین خواهیم داشت $C1=1$

سوال: چه تعداد از مقادیر کوچکتر از ProductID=776 می‌باشد؟

جواب: مقدار سه سطر، در واقع مقادیر سطر دوم، سوم و چهارم کوچکتر از مقدار سطر اول می‌باشند، $C2=3$

سوال: تعداد کل سطرهای گروه اول چه مقدار می‌باشد؟

جواب: 4 سطر

بنابراین برای بدست آوردن CUME_DIST سطر اول خواهیم داشت:

$$\text{Rows} = (1+3)/4 = 1$$

محاسبه سطر دوم از گروه اول بدون شرح:

$$\text{Rows} = (1+2)/4 = 0.75$$

امیدوارم مفید واقع شده باشد.

قبل از اینکه این موضوع را بررسی کنیم باید با دستور Delete و Truncate آشنا شویم.

بررسی دستور Delete :

همانگونه که می‌دانیم از این دستور برای حذف رکوردها استفاده می‌کنند. با اجرای دستور Delete به راحتی می‌توانید تعدادی از رکوردهای یک جدول را حذف کنید. ساده‌ترین شکل استفاده از دستور Delete به صورت زیر می‌باشد.

```
DELETE FROM table_name  
WHERE some_column=some_value
```

برای مثال در صورتیکه بخواهیم مشتریانی را حذف کنیم که کشور (Country) آنها USA است باید از دستور زیر استفاده کنیم.

```
DELETE FROM Customers  
WHERE Country='USA'  
GO
```

با اجرای این دستور هر رکوردی که در شرط مربوطه صدق کند حذف خواهد شد. (خوب این رو که همه می‌دانند) اما نکته مهمی که دستور Delete دارد این است کار این دستور به شکل Transactional می‌باشد. یعنی یا کلیه رکوردهایی که Country آنها USA است حذف می‌شود و یا هیچکدام از آنها. پس اگر شما 200000 رکورد داشته باشید که در این شرط صدق کند اگر وسط کار Delete (البته اگر عملیات حذف طولانی باشد) منصرف شوید می‌توانید با Cancel کردن این دستور عملیات Rollback Transaction را به خودکار توسط SQL Server داشته باشید. در صورتیکه عملیات Cancel را انجام دهید SQL Server از Log File برای بازگرداندن مقادیر حذف شده استفاده خواهد کرد.

اما نکته دیگری که دستور Delete دارد این است که این دستور Log کلیه رکوردهایی را که قرار است حذف کند در Log File می‌نویسد. این Log شامل اصل رکورد، تاریخ و زمان حذف، نام کاربر و... می‌باشد. شاید الان متوجه شوید که دستور Delete چرا در برخی از مواقع که قرار است حجم زیادی از اطلاعات را حذف نماید به گندی این کار را انجام می‌دهد. (چون باید Log رکوردهای حذف شده در Log File نوشته شود).



بررسی دستور Truncate Table:

Truncate در لغت به معنی بریدن و کوتاه کردن می‌باشد. با استفاده از دستور Truncate Table می‌توانید محتوای کلیه رکوردهای موجود در یک جدول را در کسری از ثانیه حذف کنید. نکته مهمی که باید درباره دستور Truncate Table بدانید این است که تاثیر استفاده از این دستور بر روی کلیه رکوردها بوده و به

هیچ عنوان نمی‌توان برای این دستور شرط (Where Clause) اعمال نمود.

شکل کلی استفاده از این دستور به صورت زیر می‌باشد.

```
TRUNCATE TABLE table_name
GO
```

برای مثال اگر بخواهیم کلیه رکوردهای موجود در جدول Customers را حذف نماییم کافی است با استفاده از این دستور اینکار را انجام دهید

```
TRUNCATE TABLE Customers
GO
```

با اجرای این دستور در کسری از ثانیه کلیه رکوردهای جدول Customers حذف خواهد شد. (بهتر است از این دستور زمانی استفاده کنید که بخواهید ساختار جدول شما باقی بماند)

اما در مورد دستور Truncate Table و Delete باید به نکات زیر توجه کنید. 1- دستور Truncate Table فاقد قسمت شرط

(Where Clause) می‌باشد در صورتیکه دستور Delete دارای قسمت شرط (Where Clause) است

2- دستور Truncate Table در Log File آدرس Page و مقدار فضای آزاد شده (کمترین میزان Log) را می‌نویسد اما در صورتیکه دستور Delete در Log هر رکوردی را که

قرار است حذف شود را در Log File ثبت می‌نماید.

3- دستور Truncate Table باعث می‌شود که Page‌های متعلق به جدول deallocate شوند. deallocate شدن Page‌ها این معنی را می‌دهد که رکوردهای موجود در جدول واقعاً حذف نشوند بلکه Extent‌های مربوط به آن Page‌ها علامت Empty خورده تا دفعات بعد مورد استفاده قرار گیرند اما دستور Delete به طور فیزیکی محتوای Page‌ها مربوط به جدول را خالی می‌کند.

نکته : پس از Truncate شدن رکوردها امکان بازگشت آنها وجود ندارد.

4- در صورتیکه جدول شما دارای ایندکس باشد. دستور Truncate Table آزاد کردن فضای مربوط به ایندکس را در یک مرحله انجام می‌دهد (مطابق بند 3) همچنین Log مربوط به این حذف به شکل حداقل (مطابق بند 2) در Log File ثبت می‌شود. اما دستور Delete هر رکوردی را که از ایندکس حذف می‌کند در Log File ثبت می‌کند.

5- Trigger مربوط به دستور Delete به هیچ عنوان هنگام اجرای دستور Truncate Table فعال نمی‌شود. در صورتیکه با اجرای دستور Delete تریگر آن فعال خواهد شد.

6- در صورتیکه جدول شما دارای Reference (Relation) باشد امکان استفاده از دستور Truncate Table وجود ندارد. لازم به ذکر است حتی اگر Reference را غیر فعال کنید

باز هم امکان استفاده از دستور Truncate Table وجود نخواهد داشت و تلاش برای اجرای دستور Truncate Table باعث نمایش خطای زیر خواهد شد.

```
TRUNCATE TABLE CUSTOMERS
GO
```

100 %
Messages

Msg 4712, Level 16, State 1, Line 1
Cannot truncate table 'Customers' because it is being referenced by a FOREIGN KEY constraint.

در صورتیکه در دستور Delete امکان حذف رکوردها به ازای جداولی که دارای Relation هستند وجود دارد. فقط باید به این نکته توجه کنید که ترتیب حذف رکوردها از جداول Master و Detail را رعایت کنید.

7- دستور Truncate Table مقدار Identity را Reset کرده و آن را به Seed (هسته/مقدار اولیه) برمیگرداند. در صورتیکه دستور Delete تأثیری بر روی مقدار Identity ندارد

8- دستور Truncate Table تنها توسط کاربرانی قابل اجرا است که نقش DB_Owner و یا SysAdmin را داشته باشند در صورتیکه دستور Delete توسط هر کاربری که مجوز Delete بر روی جدول را داشته باشد قابل اجرا می‌باشد.

9- پس از اجرای دستور Truncate Table تعداد رکوردهای حذف شده نمایش داده نمی‌شود. در صورتیکه هنگام اجرای دستور Delete تعداد رکوردهای حذف شده نمایش داده می‌شود.

```
TRUNCATE TABLE Customers2
GO
```

100 %

Messages

Command(s) completed successfully.

```
DELETE FROM Customers2
GO
```

100 %

Messages

(91 row(s) affected)

نظرات خوانندگان

نویسنده: محمد مهدی ملائیان

تاریخ: ۱۵:۳۰ ۱۳۹۲/۰۴/۰۲

مطلب مفیدی بود. منتظر مطالب شما هستم.

Identity یکی از Attribute‌هایی که در SQL Server به ازای Column‌های عددی می‌توان در نظر گرفت. به طور خیلی ساده هنگامی که این Attribute به ازای یک فیلد عددی تنظیم گردد. چنانچه رکوردی در جدول مربوط به Identity درج شود فیلد Identity مقداری را به طور اتوماتیک دریافت خواهد نمود. نحوه دریافت مقدار به ازای فیلد Identity با توجه به آخرین مقدار آن و گام افزایش است که در هنگام ایجاد identity تعریف می‌گردد.

برای ایجاد یک فیلد از نوع Identity می‌توانید زمانی که جدول خود را ایجاد می‌کنید این Attribute را به فیلد مورد نظر خود تخصیص دهید.

مثال 1: این مثال نحوه ایجاد یک فیلد از نوع Identity را نمایش می‌دهد.

```
USE tempdb
GO
CREATE TABLE Customers1
(
    ID INT IDENTITY, -- ID INT IDENTITY(1,1)
    Name NVARCHAR(100),
    [Address] NVARCHAR(200)
)
GO
```

همانطور که در مثال 1 مشاهده می‌کنید فیلد ID از نوع Identity تعریف شده است. در این حالت (ID int IDENTITY) مقدار شروع و گام افزایش به ازای این فیلد 1 در نظر گرفته خواهد شد. در این صورت اگر چند رکورد زیر را به ازای این جدول درج کنید. مقدار Identity به صورت زیر خواهد بود.

```
INSERT INTO Customers1 (Name,[Address]) VALUES
(N'میانه',N'مسعود'),
(N'میانه',N'فرید'),
(N'میانه',N'احمد')
GO
SELECT * FROM Customers1
```

Results		Messages	
	ID	Name	Address
1	1	مسعود	میانه
2	2	فرید	میانه
3	3	احمد	میانه

مثال 2: این مثال نحوه ایجاد یک فیلد از نوع Identity به همراه مقدار شروع و گام افزایش را مشخص می‌کند.

```
USE tempdb
GO
CREATE TABLE Customers2
(
    ID INT IDENTITY(100,2),
    Name NVARCHAR(100),

```



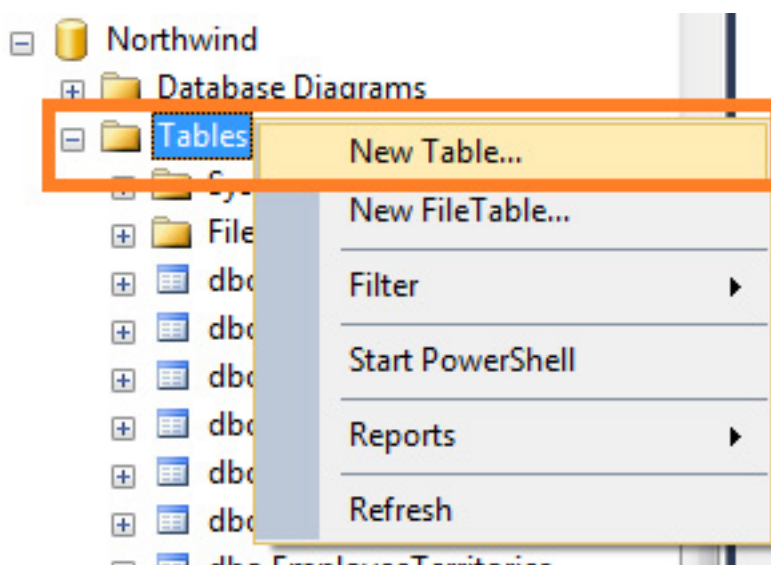
```
[Address] NVARCHAR(200)
)
GO
```

همانطور که در مثال 2 مشاهده می‌کنید فیلد ID از نوع Identity تعریف شده است و مقدار شروع آن از 100 و همچنین گام افزایش 2 در نظر گرفته شده است. در این صورت اگر چند رکورد زیر را به ازای این جدول درج کنید. مقدار Identity به صورت زیر خواهد بود.

```
INSERT INTO Customers2 (Name,[Address]) VALUES
(N'مسعود',N'میانه'),
(N'فرید',N'میانه'),
(N'احمد',N'میانه')
GO
SELECT * FROM Customers2
```

	ID	Name	Address
1	100	مسعود	میانه
2	102	فرید	میانه
3	104	احمد	میانه

مثال 3: این مثال نحوه تنظیم یک فیلد به صورت Identity را در محیط SSMS (SQL Server Management Studio) آموزش می‌دهد. 1- برای شروع کار همانند تصویر زیر بر روی قسمت Table کلیک راست کنید و گزینه New Table... را انتخاب کنید.



2- پس از نمایش پنجره زیر فیلدی را که می‌خواهید از نوع Identity باشد را انتخاب کرده و در قسمت Column Properties

خصیصه Is Identity را برابر Yes قرار دهید تا فیلد مورد نظر شما از نوع Identity در نظر گرفته شود. لازم به ذکر است که Identity Seed مقدار شروع و Identity Increment گام افزایش را مشخص می‌نماید.

TAHERI-PC.Northwind - dbo.Table_1* × Identity.sql - (loca... \Administrator (51))

Column Name	Data Type	Allow Nulls
ID	int	<input type="checkbox"/>
Name	nvarchar(100)	<input checked="" type="checkbox"/>
Address	nvarchar(100)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Column Properties

Deterministic	Yes
DTS-published	No
Full-text Specification	No
Has Non-SQL Server Subscriber	No
Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1
Indexable	Yes
Is Columnset	No

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۳/۲۱ ۱۷:۱

آیا فیلد Identity به خودی خود زمانیکه primary key و unique نباشه (مانند مثال‌های بالا) ارزش و کاربردی داره؟

نویسنده: فرید طاهری
تاریخ: ۱۳۹۲/۰۳/۲۲ ۹:۲۳

این موضوع بستگی به سناریو شما داره
اما معمولا در بیشتر مواقع Identity را به شکل Unique در نظر می‌گیرند ذکر این نکته هم ضروری است که
1- در SQL Server معمولا Primary Key بوسیله یک Unique Clustered Index هندل می‌شود
(هر چند می‌شود اون رو به صورت یک Unique Non Clustered Index در نظر گرفت)
2- Clustered Index ترتیب و چینش فیزیکی رکوردها را مشخص می‌کند یعنی اگر Identity به عنوان کلاستر ایندکس باشد
چینش و ترتیب فیزیکی رکوردها بر اساس Identity خواهد بود (سطح leaf Level مربوط به ایندکس که در کلاستر ایندکس
همان Data Level است)

همانگونه که می‌دانید مقدار Identity پس از درج به آن تخصیص می‌یابد چنانچه بخواهید به این مقدار دسترسی پیدا کنید چندین روش به ازای اینکار وجود دارد که ما در این مقاله سه روش معمول را بررسی خواهیم نمود.

1- استفاده از متغیر سیستمی @@Identity

2- استفاده از تابع Scope_Identity ()

3- استفاده از تابع Ident_Current

هر سه این توابع مقدار Identity ایجاد شده برای جداول را نمایش می‌دهند. اما تفاوت هایی باهم دارند که در ادامه مقاله این تفاوت‌ها بررسی شده است.

1- متغیر سیستمی @@Identity : این متغیر سیستمی حاوی آخرین Identity ایجاد شده به ازای Session جاری شما است. لازم به ذکر است اگر به واسطه Insert شما، Identity دیگری در یک حوزه دیگر (مانند یک Trigger) ایجاد شود مقدار موجود در این متغیر حاوی آخرین Identity ایجاد شده است. (یعنی Identity ایجاد شده توسط آن تریگر و نه خود جدول). لازم به ذکر است این موضوع به طور کامل در ادامه مقاله شرح داده شده است.

2- استفاده از تابع Scope_Identity () : با استفاده از این تابع می‌توانیم آخرین Identify ایجاد شده به ازای Session جاری را بدست آوریم. لازم به ذکر است مقادیر Identity ایجاد شده توسط سایر حوزه‌ها تاثیر در مقدار بازگشتی توسط این تابع ندارد. در ادامه مقاله این موضوع به طور کامل بررسی شده است.

3- استفاده از تابع ident_Current : این تابع آخرین مقدار Identity موجود در یک جدول را نمایش می‌دهد. ذکر این نکته ضروری است که Identity ایجاد شده توسط سایر Session‌ها هم روی خروجی این تابع تاثیرگذار است. چون این تابع آخرین Identity موجود در جدول را به شما نمایش می‌دهد و نه Identity ایجاد شده به ازای یک Session را.

برای بدست آوردن یک Identity کافی است که پس از درج رکورد در جدول مورد نظر متغیر سیستمی @@Identity و یا توابع Scope_Identity و یا Ident_Current را همانند مثال زیر Select کنید.

```
USE TEMPDB
GO
IF OBJECT_ID(N'Employees', N'U') IS NOT NULL
    DROP TABLE Employees1;
GO
CREATE TABLE Employees
(
    ID int IDENTITY,
    FirstName NVARCHAR(50),
    LastName NVARCHAR(50)
)
GO
INSERT INTO Employees (FirstName,LastName) VALUES (N'طاهری',N'مسعود')
GO
SELECT @@IDENTITY AS [@@IDENTITY]
SELECT SCOPE_IDENTITY() AS [SCOPE_IDENTITY()]
SELECT IDENT_CURRENT('Employees1') AS [IDENT_CURRENT('Employees1')]
GO
```

خروجی دستورات بالا پس از درج رکورد مورد نظر به صورت زیر است.

Results		Messages	
@@IDENTITY			
1	1		
SCOPE_IDENTITY()			
1	1		
IDENT_CURRENT('Employees')			
1	1		

اما ممکن است از خودتان این سوال را بپرسید که آیا این توابع در سطح شبکه آخرین مقدار Identity درج شده توسط سایر Sessionها را نمایش می‌دهند و یا Session جاری را؟ (منظور Sessionی که درخواست مقدار موجود در identity را نموده است).

برای دریافت پاسخ این سوال مطابق مراحل اسکریپت‌های زیر را اجرا نمایید.

1- ایجاد جدول Employees1

```
USE TEMPDB GO
IF OBJECT_ID('Employees1', 'U') IS NOT NULL
    DROP TABLE Employees1;
GO
CREATE TABLE Employees1
(
    ID int IDENTITY(1,1),
    FirstName NVARCHAR(50),
    LastName NVARCHAR(50)
)
GO
```

همانطور که مشاهده می‌کنید مقدار شروع برای Identity برابر 1 و گام افزایش هم برابر 1 در نظر گرفته شده است ((Identity(1,1)).

2- در Session جدید دستورات زیر را اجرا نمایید. (درج رکورد جدید در جدول Employees1 و واکنش مقدار Identity)

USE tempdb

```
GO
INSERT INTO Employees1(FirstName,LastName) VALUES (N'طاهری',N'فرید')
GO
SELECT @@IDENTITY AS [@@IDENTITY]
SELECT SCOPE_IDENTITY() AS [SCOPE_IDENTITY()]
SELECT IDENT_CURRENT('Employees1') AS [IDENT_CURRENT('Employees1')]
GO
```

SPID مربوط به
Connection جاری

SQLQuery2.sql - (lo...Administrator (55))* X SQLQuery1.sql - (lo...Administrator (53))*

```
USE tempdb
GO
INSERT INTO Employees1(FirstName,LastName) VALUES (N'طاهری',N'فرید')
GO
SELECT @@IDENTITY AS [@@IDENTITY]
SELECT SCOPE_IDENTITY() AS [SCOPE_IDENTITY()]
SELECT IDENT_CURRENT('Employees1') AS [IDENT_CURRENT('Employees1')]
GO
```

100 %

Results Messages

@@IDENTITY	
1	1

SCOPE_IDENTITY()	
1	1

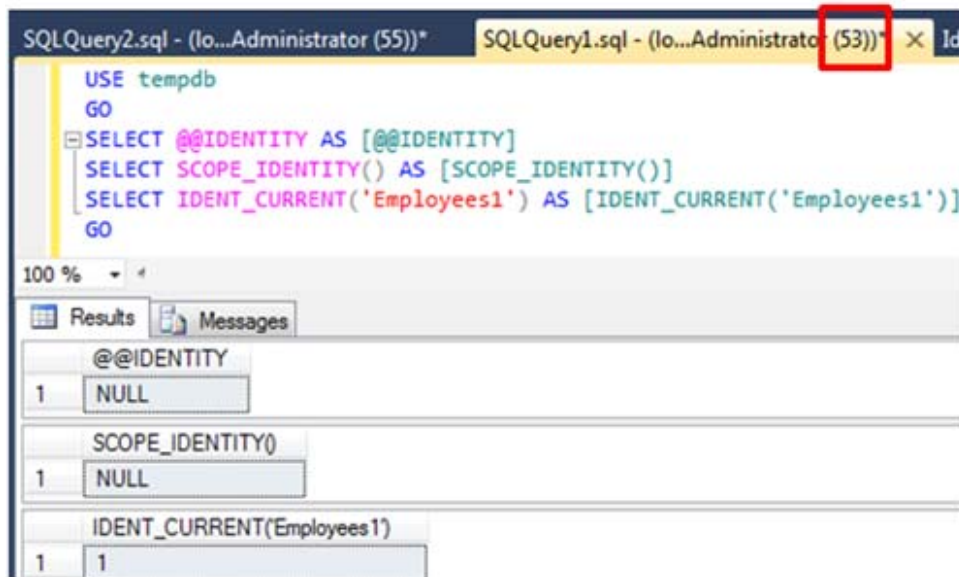
IDENT_CURRENT('Employees1')	
1	1

همانگونه که ملاحظه می‌کنید @@Identity , Scope_Identity() و Ident_Current هر سه مقدار Identity (عدد 1) ایجاد شده بوسیله دستور Insert را به شما نمایش می‌دهند.

1- و در انتها در یک Session دیگر دستورات زیر را اجرا نمایید. (واکشی مقدار Identity)

```
USE tempdb
GO
SELECT @@IDENTITY AS [@@IDENTITY]
SELECT SCOPE_IDENTITY() AS [SCOPE_IDENTITY()]
SELECT IDENT_CURRENT('Employees1') AS [IDENT_CURRENT('Employees1')]
GO
```

SPID مربوط به
Connection جاری



همانطور که مشاهده می‌کنید در این Session ما از SQL خواسته‌ایم آخرین مقدار Identity را به ما نشان داده شود. باید به این نکته توجه کنید با توجه به اینکه در این Session عملیات درجی هنوز انجام نگرفته است که ما Identity ایجاد شده را مشاهده نماییم. بنابراین صرفاً تابع Idet_Current مقدار Identity موجود در جدول را به ما نمایش می‌دهد.

پس می‌توان به این نکته رسید که

@@Identity و Scope_Identity ایجاد به ازای Session جاری را نمایش داده و به مقادیر تولید شده توسط سایر Session های دیگر دسترسی ندارد.

Ident_Current : آخرین Identity موجود در جدول را به شما نمایش می‌دهد. بنابراین باید این نکته را در نظر داشته باشید که Identity ها ایجاد شده توسط سایر Session ها روی مقدار بازگشتی این تابع تأثیرگذار است.

اما یکی دیگر از مباحث مهم درباره Identity تأثیر Scope بر مقدار Identity است (یعنی چه!). برای اینکه با مفهوم این موضوع آشنا شوید اسکریپت‌های مربوط به مثال زیر را بدقت اجرا کنید.

1- ایجاد جدول Employees1

```
USE TEMPDB
GO
IF OBJECT_ID(N'Employees1', N'U') IS NOT NULL
    DROP TABLE Employees1;
GO
CREATE TABLE Employees1
(
    ID int IDENTITY(1,1),
    FirstName NVARCHAR(50),
    LastName NVARCHAR(50)
)
GO
```

همانطور که مشاهده می‌کنید مقدار شروع برای Identity برابر 1 و گام افزایش هم برابر 1 در نظر گرفته شده است)
(Identity(1,1) .

2- ایجاد جدول Employees2

```
USE TEMPDB
GO
IF OBJECT_ID(N'Employees2', N'U') IS NOT NULL
    DROP TABLE Employees2;
GO
CREATE TABLE Employees2
(
    ID int IDENTITY(100,1),
    FirstName NVARCHAR(50),
    LastName NVARCHAR(50)
)
GO
```

همانطور که مشاهده می‌کنید مقدار شروع برای Identity برابر 100 و گام افزایش هم برابر 1 در نظر گرفته شده است)
(Identity(100,1) .

3- ایجاد یک Trigger به ازای جدول Employees1

```
USE tempdb
GO
CREATE TRIGGER Employees1_Insert ON Employees1 FOR INSERT
AS
BEGIN
    INSERT Employees2(FirstName,LastName)
    SELECT FirstName,LastName FROM INSERTED
END;
GO
```

Trigger ایجاد شده به ازای جدول Employees1 به ازای عملیات Insert اجرا می‌شود. همچنین مقادیر درج شده در جدول Employees1 بوسیله جدول Inserted در دسترس است. لازم به ذکر است جدول Inserted یک جدول موقت بوده که توسط Trigger ایجاد شده و داخل خود آن معتبر است.

هدف ما از ایجاد این Trigger تهیه یک کپی از رکوردهایی که در جدول Employees1 درج می‌شوند است. این کپی قرار است با استفاده از دستور Insert...Select در جدول Employees2 ایجاد گردد.

4- درج یک رکورد در جدول Employees1 و واکنشی مقدار Identity

```
USE tempdb
GO
INSERT INTO Employees1(FirstName,LastName) VALUES (N'طاهری',N'مسعود')
GO
SELECT @@IDENTITY AS [@@IDENTITY]
SELECT SCOPE_IDENTITY() AS [SCOPE_IDENTITY()]
SELECT IDENT_CURRENT('Employees1') AS [IDENT_CURRENT('Employees1')]
SELECT IDENT_CURRENT('Employees2') AS [IDENT_CURRENT('Employees2')]
GO
```


Results		Messages	
		@@IDENTITY	
1	100		
		SCOPE_IDENTITY()	
1	1		
		IDENT_CURRENT('Employees1')	
1	1		
		IDENT_CURRENT('Employees2')	
1	100		

مقادیر استخراج شده به ازای Identity به شرح زیر است

1- Identity@@ : پس از درج رکورد در جدول Employees1 متغیر سیستمی @@Identity مقدار 100 را نمایش داده است دلیل این موضوع بر می‌گردد به Trigger موجود در جدول Employees1.

با توجه به اینکه جدول Employees1 دارای یک فیلد Identity بوده است هنگام درج رکورد در جدول مقدار @@Identity=1 است اما چون این جدول دارای Trigger ی است که این Trigger خود با جدولی دیگری درگیر است که دارای Identity است مقدار متغیر @@identity خواهد شد.

2- Scope_Identity() : مقدار نمایش داده شده توسط تابع Scope_Identity() برابر با مقدار Identity تخصیص (عدد 1) داده شده به ازای رکورد شما می‌باشد که این موضوع در اغلب موارد مد نظر برنامه‌نویسان می‌باشد.

3- Ident_Current('Employees1') : مقدار نمایش شده توسط تابع Ident_Current آخرین مقدار Identity (عدد 1) موجود در جدول Employees1 است.

4- Ident_Current('Employees2') : مقدار نمایش شده توسط تابع Ident_Current آخرین مقدار Identity (عدد 100) موجود در جدول Employees2 است.

چند نکته مهم

1- مقدار بازگردانده شده توسط تابع Ident_Current آخرین مقدار Identity موجود در جدول مورد نظر شما بوده است و عملیات درج سایر کاربران در این مقدار تاثیر گذار است.

2- برای بدست آوردن مقدار Identity درست‌تر است از تابع Scope_Identity() استفاده نماییم. معمولاً در بیشتر مواقع مقدار بازگردانده شده توسط این تابع مد نظر برنامه‌نویسان است.

3- EntityFramework و Nhibernate هم برای بدست آوردن Identity از تابع Scope_Identity استفاده می‌کند.

نظرات خوانندگان

نویسنده: کاربر

تاریخ: ۴:۵۰ ۱۳۹۲/۰۴/۳۱

بین دوست من مطلبتون رو خوندم هم اینو و هم قبلی رو، ازش خوشم اومد اما چیزی راجب درج صریح یا بروز رسانی مقادیر Identity ننوشته بودین. یا اینکه همیشه در یک جدول دو identity property داشت.

من بلام با set identity_insert table_name on/off کاری کنم که خودم دستی مقداری را برای خصیصه identity لحاظ کنم. ولی متاسفانه نتونستم مقدار یک ستون با خصیصه Identity رو بروز رسانی (یا همون update) کنم. لطفا بهم بگید که اصلا این کار ممکنه یا من بلد نیستم. البته براساس query زیر بمن SQL Server گفته که همیشه این ستون را update کرد که ظاهرا هم همین طور(ستون id همانطور که در پیام آمده از نوع identity هست)

```
update t
set id = new_id
from (select id, row_number() over(order by id) new_id from #temp)t
--Cannot update identity column 'id'.
```

اصلا اجازه بدین یه جور دیگه سوال رو مطرح کنم من نیاز دارم تمام مقادیر identity رو بروز رسانی کنم تا کاملا پشت سر هم و متوالی بشن این کار را میتونم با یک تابع row_number و یک derived table انجام بدم (اگر بذارن!) همانطور که قبلا نشان دادم، یا با روش زیر این کار را بکنم که البته اجرا نمیشه به این دلیل که در یک جدول همیشه دو identity property داشت. با فرض اجرا شدن دستور select into باز هم در دستور update با مشکل بر می خوردیم (چون همیشه ستون id را بروز رسانی کرد)

```
select id, identity(int, 1,1) new_id
into #temptable
from #temp
order by id asc

/*
cannot add identity column, using the SELECT INTO statement, to table '#temptable',
which already has column 'id' that inherits the identity property.
*/
update t
set id = new_id
from #temp t
join #temptable d
on t.id = d.id;
```

البته یک راهی برای حل این مساله هست اونم اینه که ابتدا بیاییم تمام داده ها جدول را در جدول دیگه ای درج کنیم سپس تمام داده های جدول را حذف کنیم سپس داده های حذف شده را با id جدید و مرتب شده در جدول اول درج کنیم. به این شکل

```
declare @t table(id int)

insert into @t
select id from #temp

delete from #temp

set identity_insert #temp on
insert #temp (id)
select row_number() over(order by id) from @t
set identity_insert #temp off
```

اما مشکلی که وجود داره اینه که اگر جدول ما parent باشه با مشکل واجه میشیم تمام سطرهای جداول child یتیم میشن.

من قصد ندارم صورت مساله نقد و بررسی بشه و اصولی بودن یا صحیح بودنش مورد ارزیابی قرار بگیره فقط برام این یک سوال شده.

مساله عمومی که راجب این ستون وجود داره استفاده کردن از Gap های حاصل شده در این ستون برای درج های بعدی است. که query آن نیز بسیار ساده و در دسترس است. آیا شما میدانید که چگونه این مشکل با sequence ای که در نسخه 2012 معرفی شده است حل می شود؟

نویسنده: وحید نصیری
تاریخ: ۱۴:۵۵ ۱۳۹۲/۰۴/۳۱

- خیر. چندین نوع استراتژی برای تعیین PK وجود دارند که یکی از آن ها فیلدهای Identity است و این تنها روش و الزاما بهترین روش نیست.
- مثلا زمانی که با ORM ها کار می کنید استفاده از فیلدهای Identity در حین ثبت تعداد بالایی از رکوردها مشکل ساز می شوند. چون این فیلدها تحت کنترل دیتابیس هستند و نه برنامه، ORM نیاز دارد پس از هربار Insert یکبار آخرین Id را از بانک اطلاعاتی واکنشی کند. همین مساله یعنی افت سرعت در تعداد بالای Insert ها (چون یکبار کوئری Insert باید ارسال شود و یکبار هم یک Select اضافی دوم برای دریافت Id تولیدی توسط دیتابیس).
- روش دوم تعیین PK استفاده از نوع Guid است. در این حالت، هم مشکل حذف رکوردها و خالی شدن یک شماره را در این بین ندارید و هم چون عموما تحت کنترل برنامه است، سرعت کار کردن با آن بالاتر است. فقط تنها مشکل آن زیبا نبودنش است در مقایسه با یک عدد ساده فیلدهای Identity.

در مورد فیلدهای Identity، [تغییر شماره Id](#) به صلاح نیست چون:
الف) همانطور که عنوان کردید روابط بین جداول را به هم خواهد ریخت.
ب) در یک وب سایت و یا هر برنامه ای، کلا آدرس ها و ارجاعات قدیمی را از بین می برد. مثلا فرض کنید شماره این مطلب 1381 است و شما آن را یادداشت کرده اید. در روزی بعد، برنامه نویسی شماره Id ها را کلا ریست کرده. در نتیجه یک هفته بعد شما به شماره 1381 ایی خواهید رسید که تطابقی با مطلب مدنظر شما ندارد (حالا فرض کنید که این عدد شماره پرونده یک شخص بوده یا شماره کاربری او و نتایج و خسارات حاصل را در نظر بگیرید).
ج) این خوب است که در بین اطلاعات یک ردیف خالی وجود دارد. چون بر این اساس می توان بررسی کرد که آیا واقعا رکوردی حذف شده یا خیر. گاهی از اوقات کاربران ادعا می کنند که اطلاعات ارسالی آن ها نیست در حالیکه نبود این رکوردها به دلیل حذف بوده و نه عدم ثبت آن ها. با بررسی این Id ها می شود با کاربران در این مورد بحث کرد و پاسخ مناسبی را ارائه داد.
و اگر شماره ای که به کاربر نمایش می دهید فقط یک شماره ردیف است (و از این لحاظ می خواهید که حتما پشت سرهم باشد)، بهتر است یک View جدید ایجاد کنید تا این Id خود افزاینده را تولید کند (بدون استفاده از pk جدول).

پ.ن.

هدف من از این توضیحات صرفا عنوان این بود که به PK به شکل یک فیلد *read only* نگاه کنید. این دقیقا برخوردی است که Entity framework با این مفهوم دارد و صحیح است و اصولی. اگر در یک کشور هر روزه عده ای به رحمت ایزدی می روند به این معنا نیست که سازمان ثبت احوال باید شماره شناسنامه ها را هر ماه ریست کند!

نویسنده: فرید طاهری
تاریخ: ۲۰:۰۰ ۱۳۹۲/۰۴/۳۱

با تشکر از آقای نصیری و پاسخ مناسبی که ارائه کرده اند
در مورد استفاده از GUID به جای identity باید به یک نکته هم اشاره کنم که در بیشتر مواقع اگر مقدار GUID ی که به ازای یک فیلد UNIQUEIDENTIFIER تنظیم می کنید به صورت SEQUENTIAL نباشد باعث Fragment شدن ایندکس خواهد شد.
برای مقایسه بهتر بین Fragmentation ایندکس مربوط به Identity و GUID به مثال زیر دقت کنید. هر دو مثال فیلد ID خود را به شکل Clustered Index دارند بعد از درج تعدادی رکورد مساوی در دو جدول Fragmentation مربوط به جدولی که دارای GUID است به شدت بالا است که این موضوع باعث کاهش کارایی خواهد شد

```

IF OBJECT_ID('TABLE_GUID')>0
DROP TABLE TABLE_GUID
GO
CREATE TABLE TABLE_GUID
(
ID UNIQUEIDENTIFIER PRIMARY KEY,
FirstName NVARCHAR(1000),
LastName NVARCHAR(1000)
)
GO
IF OBJECT_ID('TABLE_IDENTITY')>0
DROP TABLE TABLE_IDENTITY
GO
CREATE TABLE TABLE_IDENTITY
(
ID INT IDENTITY PRIMARY KEY,
FirstName NVARCHAR(1000),
LastName NVARCHAR(1000)
)
GO
INSERT INTO TABLE_GUID(ID,FirstName,LastName) VALUES
(NEWID(),REPLICATE('FARID*',100),REPLICATE('Taheri*',100))
GO 10000

INSERT INTO TABLE_IDENTITY(FirstName,LastName) VALUES
(REPLICATE('FARID*',100),REPLICATE('Taheri*',100))
GO 10000

--Fragmentation بررسی وضعیت
SELECT * FROM sys.dm_db_index_physical_stats(DB_ID(),OBJECT_ID('TABLE_GUID'),NULL,NULL,'DETAILED')
DBCC SHOWCONTIG(TABLE_GUID)
GO
SELECT * FROM sys.dm_db_index_physical_stats(DB_ID(),OBJECT_ID('TABLE_IDENTITY'),NULL,NULL,'DETAILED')
DBCC SHOWCONTIG(TABLE_IDENTITY)
GO

```

خوب برای اینکه Fragmentation این نوع جداول را رفع کنید چند راه داریم

1- تولید GUID به صورت Sequential (لازم می‌دانم اشاره کنم این قابلیت در SQL Server وجود دارد ولی مقدار تولید شده باید به شکل یک Default Constraint باشد که این موضوع نیازمند این است که شما اگر در سورس به این GUID نیاز پیدا کنید مجبور به زدن Select و... شوید. اگر بخواهید در سورس این کار را انجام دهید باید از Extention‌هایی که برای اینکار وجود دارند استفاده کنید فکر کنم Nhibernate این حالت رو پشتیبانی کنه در مورد EF دقیقاً اطلاع ندارم باید اهل فن نظر بدن)

2- تنظیم مقدار Fillfactor به ازای ایندکس

3-Rebuild و یا Reorganize دوره ای ایندکس

مقدمه

تکنولوژی CTE از نسخه 2005 SQL Server رسمیت یافته است و شامل یک result set موقتی [1] است که دارای نام مشخص بوده و می‌توان از آن در دستورات SELECT, INSERT, UPDATE, DELETE استفاده کرد. همچنین از CTE می‌توان در دستور CREATE VIEW و دستور SELECT مربوط به آن استفاده کرد. در نسخه 2008 SQL Server نیز امکان استفاده از CTE در دستور MERGE فراهم شده است.

در SQL Server از دو نوع CTE بازگشتی [2] و غیر بازگشتی [3] پشتیبانی می‌شود. در این مقاله سعی شده است نحوه تعریف و استفاده از هر دو نوع آن آموزش داده شود.

انواع روش‌های ایجاد جداول موقت

برای استفاده از جداول موقت در سرور اسکوال، سه راه زیر وجود دارد.

روش اول: استفاده از دستوری مانند زیر است که سبب ایجاد جدول موقت در بانک سیستمی tempdb می‌شود. زمانی که شما ارتباط خود را با سرور SQL قطع می‌کنید به صورت اتوماتیک جداول موقت شما از بانک tempdb حذف می‌شوند. این روش در برنامه نویسی پیشنهاد نمی‌شود و فقط در کارهای موقت و آزمایشی مناسب است.

```
SELECT * INTO #temptable FROM [Northwind].[dbo].[Products]
UPDATE #temptable SET [UnitPrice] = [UnitPrice] + 10
```

روش دوم: استفاده از متغیر نوع Table است، که نمونه آن در مثال زیر دیده می‌شود. زمانیکه از محدوده [4] جاری کد [5] خودتان خارج شوید آن متغیر نیز از حافظه پاک می‌شود. از این روش، عموماً در کدهای Stored Procedure ها و UserDefined Function ها استفاده می‌شود.

```
DECLARE @tempTable TABLE
(
    [ProductID] [int] NOT NULL,
    [ProductName] [nvarchar](40) NOT NULL,
    [UnitPrice] [money] NULL
)

INSERT INTO @tempTable
SELECT
    [ProductID],
    [ProductName],
    [UnitPrice]
FROM [Northwind].[dbo].[Products]

UPDATE @temptable SET [UnitPrice] = [UnitPrice] + 10
```

روش سوم: استفاده از CTE است که مزیت‌هایی نسبت به دو روش قبلی دارد و در بخش بعدی به نحوه تعریف و استفاده از آن خواهیم پرداخت.

کار با CTE

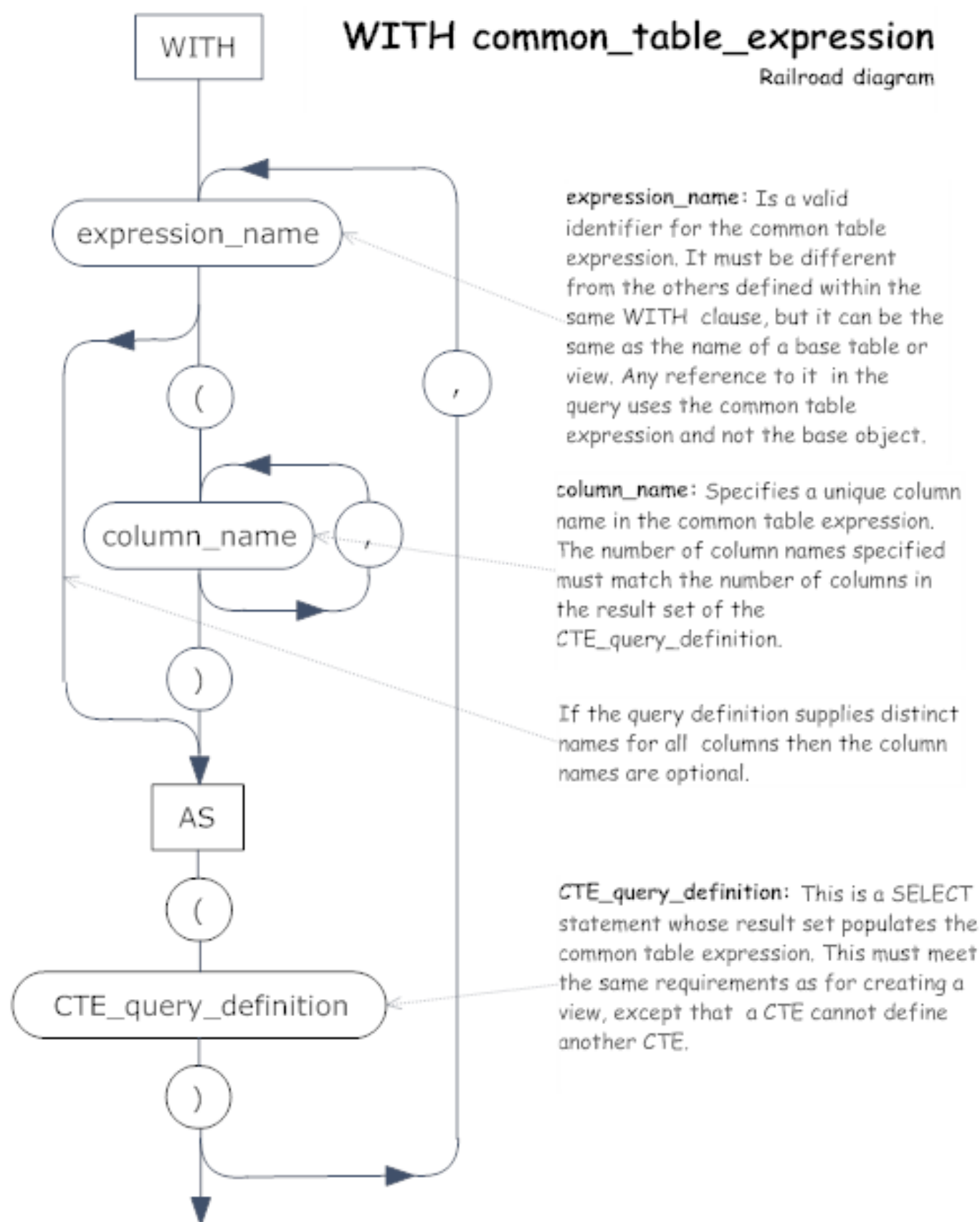
ساده -ترین شکل تعریف یک CTE به صورت زیر است:

```
WITH yourName [(Column1, Column2, ...)]
AS
(
```

```
) your query
```

با کلمه WITH شروع شده و یک نام اختیاری به آن داده می-شود. سپس فهرست فیلدهای جدول موقت را درون زوج پرانتز، مشخص می-کنید. تعریف این فیلدها اختیاری است و اگر حذف شود، فیلدهای جدول موقت، مانند فیلدهای کوئری مربوطه خواهد بود.

your query شامل دستوری است که سبب تولید یک result set می-شود. قواعد تعریف این کوئری مشابه قواعد تعریف کوئری است که در دستور CREATE VIEW کاربرد دارد.



همانطور که از این تصویر مشخص است می-توان چندین بلوک از این ساختار را به دنبال هم تعریف نمود که با کاما از هم جدا می-شوند. در واقع یکی از کاربردهای CTE ایجاد قطعات کوچکی است که امکان استفاده مجدد را به شما داده و می-تواند سبب خواناتر شدن کدهای پیچیده شود.

یکی دیگر از کاربردهای CTE آنجایی است که شما نمی-خواهید یک شی View عمومی تعریف کنید و در عین حال می-خواهید از مزایای Viewها بهرمنند شوید.

و همچنین از کاربردهای دیگر CTE تعریف جدول موقت و استفاده از آن جدول به صورت همزمان در یک دستور است.

بعد از آنکه CTE یا CTEهای خودتان را تعریف کردید آنگاه می-توانید مانند جداول معمولی از آنها استفاده کنید. استفاده از این جداول توسط دستوری خواهد بود که دقیقاً بعد از تعریف CTE نوشته می-شود.

ایجاد یک CTE غیر بازگشتی[6]

مثال اول، یک CTE غیر بازگشتی ساده را نشان میدهد.

```
WITH temp
AS
(
    SELECT
        [ProductName],
        [UnitPrice]
    FROM [Northwind].[dbo].[Products]
)
SELECT * FROM temp
ORDER BY [UnitPrice] DESC
```

مثال دوم نمونه-ای دیگر از یک CTE غیر بازگشتی است.

```
WITH orderSales (OrderID, Total)
AS
(
    SELECT
        [OrderID],
        SUM([UnitPrice]*[Quantity]) AS Total
    FROM [Northwind].[dbo].[Order Details]
    GROUP BY [OrderID]
)
SELECT
    O.[ShipCountry],
    SUM(OS.[Total]) AS TotalSales
FROM [Northwind].[dbo].[Orders] AS O INNER JOIN [orderSales] AS OS
ON O.[OrderID] = OS.[OrderID]
GROUP BY O.[ShipCountry]
ORDER BY TotalSales DESC
```

هدف این کوئری، محاسبه کل میزان فروش کالاها، به ازای هر کشور می-باشد. ابتدا از جدول Order Details مجموع فروش هر سفارش محاسبه شده و نتیجه آن در یک CTE به نام orderSales قرار می-گیرد و از JOIN این جدول موقت با جدول Orders محاسبه نهایی انجام شده و نتیجه-ای مانند این تصویر حاصل می-شود.

Results Messages		
	ShipCountry	TotalSales
1	USA	263566.98
2	Germany	244640.63
3	Austria	139496.63
4	Brazil	114968.48
5	France	85498.76
6	Venezuela	60814.89
7	UK	60616.51
8	Sweden	59523.70
9	Ireland	57317.39
10	Canada	55334.10
11	Belgium	35134.98
12	Denmark	34782.25

نتیجه خروجی

مثال سوم استفاده از دو CTE را به صورت همزمان نشان می-دهد:

```
WITH customerList
AS
(
    SELECT
        [CustomerID],
        [ContactName]
    FROM [Northwind].[dbo].[Customers]
    WHERE [Country] = 'UK'
)
,orderList
AS
(
    SELECT
        [CustomerID],
        [OrderDate]
    FROM [Northwind].[dbo].[Orders]
    WHERE YEAR([OrderDate]) < 2000
)
SELECT
    cl.[ContactName],
    YEAR(ol.[OrderDate]) AS SalesYear
FROM customerList AS cl JOIN orderList AS ol
ON cl.[CustomerID] = ol.[CustomerID]
```

مثال چهارم استفاده مجدد از یک CTE را نشان می-دهد. فرض کنید جدولی به نام digits داریم که فقط یک فیلد digit دارد و دارای 10 رکورد با مقادیر 0 تا 9 است. مانند تصویر زیر

	digit
	0
	1
	2
	3
	4
	5
	6
▶	7
	8
	9
*	NULL

نتیجه خروجی

حال می-خواهیم از طریق CROSS JOIN اعداد 1 تا 100 را با استفاده از مقادیر این جدول تولید کنیم. کد زیر آنرا نشان می-دهد:

```
WITH digitList
AS
(
    SELECT [digit] from [digits]
)
SELECT
    a.[digit] * 10 + b.[digit] + 1 AS [Digit]
FROM [digitList] AS a CROSSJOIN [digitList] AS b
```

در این کد یک CTE تعریف شده و دو بار مورد استفاده قرار گرفته است. مثلاً اگر بخواهید اعداد 1 تا 1000 را تولید کنید می-توانید سه بار از آن استفاده کنید. حاصل این دستور result set مانند زیر است.

Results		Messages
	Digit	
1	1	
2	2	
3	3	
4	4	
5	5	
6	6	
7	7	
8	8	
9	9	
10	10	
11	11	
12	12	
13	13	

نتیجه

87	87
88	88
89	89
90	90
91	91
92	92
93	93
94	94
95	95
96	96
97	97
98	98
99	99
100	100

نتیجه

حتی می-توان از یک CTE در کوئری CTE بعدی مانند کد زیر استفاده کرد.

```
WITH CTE_1 AS
(
    ....
),
CTE_2 AS
(
    SELECT ... FROM CTE_1 JOIN ...
)
SELECT *
FROM FOO
LEFTJOIN CTE_1
LEFTJOIN CTE_2
```

ایجاد یک CTE بازگشتی[7]

از CTE بازگشتی برای پیمایش جدولی استفاده می-شود که رکوردهای آن دارای رابطه سلسله مراتبی یا درختی است. نمونه این جدول، جدول کارمندان است که مدیر هر کارمند نیز مشخص شده است یا جدولی که ساختار سازمانی را نشان می-دهد یا جدولی که موضوعات درختی را در خود ذخیره کرده است. یکی از مزایای استفاده از CTE بازگشتی، سرعت کار آن در مقایسه با روش-های پردازشی دیگر است.

ساختار کلی یک دستور CTE بازگشتی به صورت زیر است.

```
WITH cteName AS
(
    query1
    UNION ALL
    query2
)
```

در بدنه CTE حداقل دو عضو[8] (کوئری) وجود دارد که بایستی با یکی از عبارت-های زیر به هم متصل شوند.

```
UNION
UNION ALL
INTERSECT
EXCEPT
```

query1 شامل دستوری است که اولین سری از رکوردهای result set نهایی را تولید می-کند. اصطلاحاً به این کوئری anchor member می-گویند.

بعد از دستور query1، حتماً بایستی از UNION ALL و امثال آنها استفاده شود.

سپس query2 ذکر می-شود. اصطلاحاً به این کوئری recursive member گفته می-شود. این کوئری شامل دستوری است که سطوح بعدی درخت را تولید خواهد کرد. این کوئری دارای شرایط زیر است.

حتماً بایستی به CTE که همان cteName است اشاره کرده و در جایی از آن استفاده شده باشد. به عبارت دیگر از رکوردهای موجود در جدول موقت استفاده کند تا بتواند رکوردهای بعدی را تشخیص دهد.

حتماً بایستی مطمئن شوید که شرایط کافی برای پایان حلقه پیمایش رکوردها را داشته باشد در غیر این صورت سبب تولید حلقه بی پایان[9] خواهد شد.

بدنه CTE می-تواند حاوی چندین anchor member و چندین recursive member باشد ولی فقط recursive memberها هستند که به CTE اشاره می-کنند.

برای آنکه نکات فوق روشن شود به مثال-های زیر توجه کنید.

فرض کنید جدولی از کارمندان و مدیران آنها داریم که به صورت زیر تعریف و مقداردهی اولیه شده است.

```
IF OBJECT_ID('Employees','U') IS NOT NULL
DROPTABLE dbo.Employees
GO

CREATETABLE dbo.Employees
(
    EmployeeID int NOT NULL PRIMARY KEY,
    FirstName varchar(50) NOT NULL,
    LastName varchar(50) NOT NULL,
    ManagerID int NULL
)
GO

INSERT INTO Employees VALUES (101,'Alireza','Nematollahi',NULL)
INSERT INTO Employees VALUES (102,'Ahmad','Mofarrahzadeh',101)
INSERT INTO Employees VALUES (103,'Mohammad','BozorgGhommi',102)
INSERT INTO Employees VALUES (104,'Masoud','Narimani',103)
INSERT INTO Employees VALUES (105,'Mohsen','Hashemi',103)
INSERT INTO Employees VALUES (106,'Aref','Partovi',102)
INSERT INTO Employees VALUES (107,'Hosain','Mahmoudi',106)
INSERT INTO Employees VALUES (108,'Naser','Pourali',106)
INSERT INTO Employees VALUES (109,'Reza','Bagheri',102)
INSERT INTO Employees VALUES (110,'Abbas','Najafian',102)
```

مثال اول: می-خواهیم فهرست کارمندان را به همراه نام مدیر آنها و شماره سطح درخت نمایش دهیم. کوئری زیر نمونه‌ای از یک کوئری بر اساس CTE بازگشتی می-باشد.

```
WITH cteReports(EmpID, FirstName, LastName, MgrID, EmpLevel)
AS
(
    SELECT EmployeeID, FirstName, LastName, ManagerID, 1
    FROM Employees
    WHERE ManagerID IS NULL
    UNION ALL
    SELECT e.EmployeeID, e.FirstName, e.LastName, e.ManagerID, r.EmpLevel + 1
    FROM Employees e INNER JOIN cteReports r
    ON e.ManagerID = r.EmpID
)
SELECT
    FirstName + ' ' + LastName AS FullName,
    EmpLevel,
    (SELECT FirstName + ' ' + LastName FROM Employees
    WHERE EmployeeID = cteReports.MgrID) AS Manager
FROM cteReports
ORDER BY EmpLevel, MgrID
```

کوئری اول در بدنه CTE رکورد مدیری را می-دهد که ریشه درخت بوده و بالاسری ندارد و شماره سطح این رکورد را 1 در نظر می-گیرد.

کوئری دوم در بدنه CTE از یک JOIN بین Employees و cteReports استفاده کرده و کارمندان زیر دست هر کارمند قبلی (فرزندان) را بدست آورده و مقدار شماره سطح آنرا به صورت Level+1 تنظیم می-کند. در نهایت با استفاده از CTE و یک subquery جهت بدست آوردن نام مدیر هر کارمند، نتیجه نهایی تولید می-شود.

مثال دوم: می-خواهیم شناسه یک کارمند را بدهیم و نام او و نام مدیران وی را به عنوان جواب در خروجی بگیریم.

```
WITH cteReports(EmpID, FirstName, LastName, MgrID, EmpLevel)
AS
(
    SELECT EmployeeID, FirstName, LastName, ManagerID, 1
    FROM Employees
    WHERE EmployeeID = 110
    UNION ALL
    SELECT e.EmployeeID, e.FirstName, e.LastName, e.ManagerID, r.EmpLevel + 1
    FROM Employees e INNER JOIN cteReports r
    ON e.EmployeeID = r.MgrID
```

```
)
SELECT
FirstName + ' ' + LastName AS FullName,
EmpLevel
FROMcteReports
ORDERBY EmpLevel
```

اگر دقت کنید اولین تفاوت در خط اول مشاهده می-شود. در اینجا مشخص می-کند که اولین سری از رکوردها چگونه انتخاب شود. مثلاً کارمندی را می-خواهیم که شناسه آن 110 باشد. دومین تفاوت اصلی این کوئری با مثال قبلی، در قسمت دوم دیده می-شود. شما می-خواهید مدیر (پدر) کارمندی که در آخرین پردازش در جدول موقت قرار گرفته است را استخراج کنید.

- a temporary named result set [1]
- recursive [2]
- nonrecursive [3]
- Scope [4]
- [5]مثلاً محدوده کدهای یک روال یا یک تابع
- nonrecursive [6]
- recursive [7]
- member [8]
- Infinite loop [9]

نظرات خوانندگان

نویسنده: همراز
تاریخ: ۱۱:۷ ۱۳۹۳/۰۱/۲۶

ضمن تشکر از پست مفید جناب عمران یکی از استفاده‌های CTE افزودن شماره ردیف به ساختار خروجی و محدود کردن نتیجه باتوجه به شماره ردیف است.
مثلاً ردیف 20 تا 30 ... که البته با پارامتر پاس می‌شوند.

```
WITH RCTE AS
(
SELECT TOP (100)
ROW_NUMBER() OVER (ORDER BY Invoice.InsertDate ASC) AS RowNumber,
Invoice.ID,
Invoice.PreInvoiceNo,
Invoice.InvoiceNo,
Invoice.IssueDate,
Invoice.CustomerID, ...
FROM
Invoice
WHERE
Invoice.HistorySequence = 1
)
SELECT DISTINCT
RCTE.ID,
RCTE.PreInvoiceNo,
RCTE.InvoiceNo,
(dbo.fnc_Calendar_Gregorian_to_Persian(RCTE.IssueDate) + 'T' + CONVERT(CHAR(8), RCTE.IssueDate, 14)) AS
IssueDate,
RCTE.CustomerID,
Customer.NameEn AS CustomerNameEn,
Customer.NameFa AS CustomerNameFa,
FROM
RCTE
INNER JOIN Customer ON RCTE.CustomerID = Customer.ID
WHERE
RowNumber BETWEEN @StartFrom AND (@RowCount + @StartFrom - 1)
```

استفاده شده از SQL 2008

روش کار : 1- دریافت پارامتر ورودی به صورت رشته

2- درج عناوین اعداد، ارزش مکانی اعداد صحیح و اعشاری هرکدام در یک جدول

3- جدا کردن ارقام صحیح و اعشاری

4- جداکردن سه رقم سه رقم اعداد صحیح و انتقال آنها به جدول مربوطه

5- Join جداول عناوین و ارقام جدا شده

6- ارسال ارقام اعشاری به همین تابع

7- مشخص کردن ارزش مکانی رقم اعشار

8- اتصال رشته حروف صحیح و اعشاری

در آخر این مطلب کد این تابع را به صورت کامل، برای دانلود قرار داده ام.

بررسی قسمت‌های مختلف کد

برای اینکه محدودیتی در تعداد ارقام صحیح و اعشاری نداشته باشیم، پارامتر ورودی را از نوع VARCHAR می‌گیریم. پس باید ورودی را بررسی کنیم تا رشته عددی باشد.

بررسی رشته ورودی:

```
-- @pNumber پارامتر ورودی
IF LEN(ISNULL(@pNumber, '')) = 0 RETURN NULL

IF (PATINDEX('%[^0-9.-]%', @pNumber) > 0)
  OR (LEN(@pNumber) - LEN(REPLACE(@pNumber, '-', '')) > 1)
  OR (LEN(@pNumber) - LEN(REPLACE(@pNumber, '.', '')) > 1)
  OR (CHARINDEX('-', @pNumber) > 1)
RETURN 'خطا'

IF PATINDEX('%[^0]%', @pNumber) = 0 RETURN 'صفر'
IF (CHARINDEX('.', @pNumber) = 1) SET @pNumber='0'+@pNumber

DECLARE @Negative AS VARCHAR(5) = '';
IF LEFT(@pNumber, 1) = '-';
BEGIN
  SET @pNumber = SUBSTRING(@pNumber, 2, 100)
  SET @Negative = 'منفی'
END
```

- بررسی NULL، خالی بودن و یا داشتن فاصله در رشته، با دانستن اینکه تابع LEN فاصله‌های آخر یک رشته را در نظر نمی‌گیرد.

- بررسی رشته ورودی برای پیدا کردن کاراکتر غیر عددی، نقطه و منفی. بررسی تعداد علامت منفی و نقطه که بیشتر از یک مورد نباشند، و در نهایت بررسی اینکه علامت منفی در ابتدای رشته ورودی باشد.

- بررسی صفر بودن ورودی (0)، مقدار ورودی شروع شونده با ممیز (0.213) و مقدار عددی منفی (-21210.0021). چیز دیگری به ذهنم نرسید!

درج عناوین در جداول مربوطه:

فکر کنم اینجا به علت وجود کاراکترهای فارسی و انگلیسی کد کمی بهم ریخته نمایش داده می‌شود.

```
DECLARE @NumberTitle TABLE (val INT, Title NVARCHAR(100));
```



```

DECLARE @Number AS INT
DECLARE @MyNumbers TABLE (id INT IDENTITY(1, 1), Val1 INT, Val2 INT, Val3 INT)

WHILE (@pNumber) <> '0'
BEGIN
    SET @number = CAST(SUBSTRING(@pNumber, LEN(@pNumber) - 2, 3) AS INT)

    INSERT INTO @MyNumbers
    SELECT (@Number % 1000) - (@Number % 100),
    CASE
    WHEN @Number % 100 BETWEEN 10 AND 19 THEN @Number % 100
    ELSE (@Number % 100) - (@Number % 10)
    END,
    CASE
    WHEN @Number % 100 BETWEEN 10 AND 19 THEN 0
    ELSE @Number % 10
    END
    END

    IF LEN(@pNumber) > 2
        SET @pNumber = LEFT(@pNumber, LEN(@pNumber) - 3)
    ELSE
        SET @pNumber = '0'
    END
END

```

سطری که تمام مقادیر آن صفر باشد برای ما بی ارزش محسوب می‌شود، مانند سطر یک در عکس زیر (جدول MyNumbers) برای عدد **1200955000** :

	id	Val1	Val2	Val3
1	1	0	0	0
2	2	900	50	5
3	3	200	0	0
4	4	0	0	1

استفاده از JOIN : JOIN کردن جدول اعداد با عناوین عددی براساس ارزش آن‌ها و JOIN جدول اعداد با جدول ارزش مکانی براساس ID به صورت نزولی (شماره سطر).

```

DECLARE @Str AS NVARCHAR(2000) = '';
SELECT @Str += REPLACE(REPLACE(LTRIM(RTRIM(nt1.Title + ' ' + nt2.Title + ' ' + nt3.title)), ' ', ','), ', ', ' و ')
+ ' ' + pt.title + ' و '
FROM @MyNumbers AS mn
INNER JOIN @PositionTitle pt
ON pt.id = mn.id
INNER JOIN @NumberTitle nt1
ON nt1.val = mn.Val1
INNER JOIN @NumberTitle nt2
ON nt2.val = mn.Val2
INNER JOIN @NumberTitle nt3
ON nt3.val = mn.Val3
WHERE (nt1.val + nt2.val + nt3.val > 0)
ORDER BY pt.id DESC

```

Replace داخلی: جایگزین کردن "دو فاصله‌ی خالی" با "یک فاصله‌ی خالی"

Replace بیرونی: جایگزینی فاصله‌های خالی با ' و '

همانطور که در بالا اشاره کردم سطرهایی که Val3, Val2, Val1 آن صفر باشد برای ما بی ارزش هستند، پس آنها را با شرط نوشته شده حذف می‌کنیم.

بدست آوردن مقدار اعشاری: خوب! حالا نوبت به عدد اعشاری می‌رسد. برای بدست آوردن حروف، مقدار اعشاری بدست آمده

را به همین تابع ارسال می‌کنیم و برای بدست آوردن عنوان ارزش مکانی، براساس طول اعشار (ID) آن را در جدول مربوطه پیدا می‌کنیم.

اگر عدد ورودی مثلاً 0.355 باشد، تابع باید صفر اول را شناسایی و قسمت عناوین اعشاری را به آن اضافه کند، که این کار با شرط ذیل انجام می‌شود.

اگر رشته اعشار بدون مقدار باشد، تابع مقدار NULL بر می‌گرداند (قسمت بررسی رشته ورودی) و هر رشته ای که با NULL جمع شود برابر با NULL خواهد بود. در این صورت با توجه به کد زیر مقداری به رشته Str به عنوان قسمت اعشاری، اضافه نمی‌گردد.

```
IF @IntegerNumber='0'  
SET @Str=CASE WHEN PATINDEX('%[^0]%', @DecimalNumber) > 0 THEN @Negative ELSE '' END + 'صفر'  
ELSE  
SET @Str = @Negative + LEFT (@Str, LEN(@Str) -2)  
  
DECLARE @PTitle NVARCHAR(100)=ISNULL((SELECT Title FROM @DecimalTitle WHERE id=LEN(@DecimalNumber)), '')  
SET @Str += ISNULL(' ممیز '+[dbo].[fnNumberToWord_Persian](@DecimalNumber) + ' '+@PTitle, '')  
RETURN @str
```

مثال: رشته ' 5445789240.54678000000000 ' رشتۀ

پنج میلیارد و چهارصد و چهل و پنج میلیون و هفتصد و هشتاد و نه هزار و دویست و چهل ممیز پنجاه و چهار هزار و ششصد و هفتاد و هشت صد-هزارم

[دانلود فایل](#)

نظرات خوانندگان

نویسنده: میثم چگینی
تاریخ: ۱۱:۴۱ ۱۳۹۲/۰۸/۰۱

با تشکر

یک مورد کوچک داره که برای نمایش درست نیاز به اضافه کردن N قبل از تمام کلمات فارسی می‌باشد
'صفر' N

نویسنده: حمیدرضا عابدینی
تاریخ: ۱۰:۵۳ ۱۳۹۲/۰۸/۰۳

حق با شماست!

من چون از Collation نوع Persian_100_CI_AI استفاده کردم همه‌ی Nها را حذف کردم. ولی برای دیگر موارد برای اینکه به صورت علامت ؟ ظاهر نشود، همانطور که شما گفتید باید از N استفاده کرد.

ممنون

نویسنده: مهیار
تاریخ: ۲۰:۳۸ ۱۳۹۲/۱۱/۱۵

سلام،

ممنون از راه حلی که ارائه دادید.

برای حذف صفرهای غیرضروری موجود در اعشار پیشنهاد من کد زیر است که ساده‌تر و خواناتر است:

```
declare @value varchar(50) = '0.1010000000'  
select replace(rtrim(replace(@value, '0', ' ')), ' ', '0');
```

توضیح کد: ابتدا تمام صفرهای موجود در رشته را تبدیل به کاراکتر Space میکنیم توسط تابع Replace سپس با تابع Rtrim تمام کاراکترهای Space آخر رشته را Remove می‌کنیم و مجدد کاراکترهای Space را برمیگردانیم به صفر.

نویسنده: مهیار
تاریخ: ۲۱:۳۷ ۱۳۹۲/۱۱/۱۵

کد مربوط به حلقه While اتان هم کمی ساده‌تر کردم مخصوصاً اولین دستور حلقه که مربوط میشه به انتخاب سه رقم آخر رشته. ضمناً نیازی نیست که صراحتاً متغیر را به integer تبدیل کنید. با صفر جمع یا با یک ضرب کنید تا بصورت Implicit تبدیل صورت بگیره:

```
WHILE (@pNumber) <> '0'  
BEGIN  
    SET @number = RIGHT(@pNumber, 3) + 0  
  
    INSERT INTO @MyNumbers  
    SELECT  
        @Number / 100 * 100,  
        CASE  
            WHEN nbr BETWEEN 10 AND 19 THEN nbr  
            ELSE nbr / 10 * 10  
        END,  
        CASE  
            WHEN nbr BETWEEN 10 AND 19 THEN 0  
            ELSE nbr % 10  
        END  
    FROM (SELECT @Number % 100)S(nbr);  
  
    IF LEN(@pNumber) > 2
```

```
        SET @pNumber = LEFT(@pNumber, LEN(@pNumber) - 3)
    ELSE
        SET @pNumber = '0'
END
```

نویسنده: حمیدرضا عابدینی
تاریخ: ۲۳:۳۲ ۱۳۹۲/۱۱/۲۰

سلام مهیار جان!
کد بسیار خوب و خوانایی گذاشتید، ممنونم!
در نتیجه:

```
DECLARE @IntegerNumber NVARCHAR(100),
        @DecimalNumber NVARCHAR(100),
        @PointPosition INT =case CHARINDEX('.', @pNumber) WHEN 0 THEN LEN(@pNumber)+1 ELSE CHARINDEX('.',
        @pNumber) END

SET @pNumber=replace(rtrim(replace(@pNumber,'0',' ')), ' ','0');
SET @IntegerNumber= LEFT(@pNumber, @PointPosition - 1)
SET @DecimalNumber= SUBSTRING(@pNumber, @PointPosition+1 , LEN(@pNumber))

SET @pNumber= @IntegerNumber
```

عنوان:	استفاده از توابع Scalar بجای case
نویسنده:	فرهود جعفری
تاریخ:	۲۱:۲۵ ۱۳۹۲/۱۱/۰۶
آدرس:	www.dotnettips.info
گروه‌ها:	T-SQL, querying, SQL

گاهی از اوقات نیاز است در کوئری‌ها از بین چندین مقدار یکی انتخاب و بجای مقدار اصلی، رشته یا عبارتی جایگزین، نوشته شود. پر استفاده‌ترین راه حل پیشنهادی، استفاده از عبارت case در داخل کوئری هست که بر اساس موارد ممکن، عبارتهای برگشتی نوشته می‌شود. این راه حل خوبی به نظر می‌رسد اما اگر تعداد گزینه‌ها زیاد شود باعث شلوغ شدن متن کوئری و اشکال در بازبینی و نگهداری آن خواهد شد.

یک راه حل دیگر استفاده از توابع نوع Scalar می‌باشد؛ به این صورت که میتوان مقدار استخراج شده از جدول را به تابع تعریف شده فرستاد و در ازاء، مقدار بازگشتی مناسبی را در خروجی مشاهده کرد. حال به یک مثال توجه کنید:

```
Select Case Gen when 0 then 'مرد' when 1 then 'زن' end As Gen From Table
```

اکنون استفاده از تابع:

```
CREATE FUNCTION fcGenName
(
    @Gen tinyint
)
RETURNS nvarchar(20)
AS
BEGIN
    -- Declare the return variable here
    DECLARE @gen nvarchar(20)

    -- Add the T-SQL statements to compute the return value here
    set @gen = (SELECT case @Gen when 0 then 'مرد' when 1 then 'زن' end as d)

    -- Return the result of the function
    RETURN @gen
END
```

و فراخوانی تابع در متن کوئری :

```
Select fcGenName(Gen) From Table
```

نظرات خوانندگان

نویسنده: م رحمانی
تاریخ: ۱۳۹۲/۱۱/۰۷ ۱۲:۲۲

سلام و تشکر
سؤال: این ارجاع به یک تابع تأثیر تو کارایی نداره؟

نویسنده: فرهود جعفری
تاریخ: ۱۳۹۲/۱۱/۰۷ ۱۴:۵۳

با سلام
ظاهراً در تعداد رکوردهای پایین مشکلی نداره اما در تعداد رکوردهای بالا احتمال کاهش سرعت اجرا دور از ذهن نیست. به هر حال من دقیق تست نکردم اما روی شیوه دیگه هم دارم کار می‌کنم که اون توابع برگشتی از نوع جدول هست که با این شیوه اساساً فرق داره

نویسنده: زاهدیان فرد
تاریخ: ۱۳۹۲/۱۱/۱۹ ۱۱:۴۴

استفاده از function خوبه مزیتش این که میشه جاهای مختلف استفاده کرد! ولی در تعداد رکورد پایین، چون در رکوردهای زیاد سرعت کوئری به شدت افت میکنه! روش اول بنظر من بهتر

نویسنده: زاهدیان فرد
تاریخ: ۱۳۹۲/۱۱/۱۹ ۱۲:۱۰

در 2012 SQL server تابعی اضافه شده به اسم IIF که بجای

```
SELECT CASE @GEN WHEN 0 THEN 'Male' ELSE 'Woman' AS Gender
```

از این می‌توان استفاده کرد

```
SELECT IIF(Gen=0, 'Male', 'Woman')
```

نویسنده: محمد سلیم آبادی
تاریخ: ۱۳۹۲/۱۱/۱۹ ۱۵:۳

در نسخه 2012 جهت سهولت در مهاجرت پایگاه داده‌های Access به SQL Server از توابع CHOOSE و IIF حمایت شده.

منتها تابع IIF چندان انعطاف پذیر نیست. مثلاً اگر بخواهید به ازای چند حالت مشخص از داده‌های یک فیلد یک مقدار را برگردانید مجبورید چند تابع IIF تودرتو بنویسید. تودرتو بودن این تابع هم به 10 سطح محدود میشه. اما CASE قابلیت‌ها و انعطاف پذیری بیشتری داره.

سوال میشه گاهی یک از این دو Performance یا کارایی بهتر دارد، در جواب میشه گفت هر دو برابر اند. در واقع IIF هنگام اجرا تبدیل به فرم CASE خواهد شد.

فرض کنید یک نظر سنجی تلوزیونی تنظیم کردیم که مردم از طریق پیامک نظر خودشان را به ما اعلام میکنند. شش گزینه هم داریم. برای انتخاب هر گزینه کفایت از اعداد 1 تا 6 استفاده کنیم. حال هنگام نمایش می‌خواهیم به جای اعداد مقدار متناظر ظاهر شود:

```

Use Tempdb
Go

CREATE TABLE [Sample] (value int);
INSERT INTO [Sample] VALUES (1),(2),(3),(4),(5),(6);
Go

--simple CASE Expression
SELECT value,
       CASE Value
         WHEN 1 THEN 'Very Bad'
         WHEN 2 THEN 'Bad'
         WHEN 3 THEN 'Not Bad'
         WHEN 4 THEN 'Good'
         WHEN 5 THEN 'Very Good'
         WHEN 6 THEN 'Excellent'
       ELSE NULL
     END AS [Result]
FROM [Sample];

--CHOOSE Scalar Function
SELECT value,
       CHOOSE(value,'Very Bad','Bad','Not Bad','Good','Very Good','Excellent')
FROM [Sample];

--nested IIF Scalr Function
SELECT value,
       IIF(value = 1, 'Very Bad',
         IIF(value = 2, 'Bad',
           IIF(value = 3, 'Not Bad',
             IIF(value = 4, 'Good',
               IIF(value = 5, 'Very Good', 'Excellent')
             )
           )
         )
       )
FROM [Sample];

```


مقدمه SQL Server، با هر تقاضا به عنوان یک واحد مستقل رفتار می‌کند. در وضعیت‌های پیچیده‌ای که فعالیت‌ها توسط مجموعه‌ای از دستورات SQL انجام می‌شود، به طوری که یا همه باید اجرا شوند یا هیچکدام اجرا نشوند، این روش مناسب نیست. در چنین وضعیت‌هایی، نه تنها تقاضاهای موجود در یک دنباله به یکدیگر بستگی دارند، بلکه شکست یکی از تقاضاهای موجود در دنباله، به معنای این است که کل تقاضاهای موجود در دنباله باید لغو شوند، و تغییرات حاصل از تقاضاهای اجرا شده در آن دنباله خنثی شوند تا بانک اطلاعاتی به حالت قبلی برگردد.

1- تراکنش چیست؟ تراکنش شامل مجموعه‌ای از یک یا چند دستور SQL است که به عنوان یک واحد عمل می‌کنند. اگر یک دستور SQL در این واحد با موفقیت اجرا نشود، کل آن واحد خنثی می‌شود و داده‌هایی که در اجرای آن واحد تغییر کرده‌اند، به حالت اول برگردانده می‌شود. بنابراین تراکنش وقتی موفق است که هر یک از دستورات آن با موفقیت اجرا شوند. برای درک مفهوم تراکنش مثال زیر را در نظر بگیرید: سهامدار A در معامله‌ای 400 سهم از شرکتی را به سهامدار B می‌فروشد. در این سیستم، معامله وقتی کامل می‌شود که حساب سهامدار A به اندازه 400 بدهکار و حساب سهامدار B همزمان به اندازه 400 بستانکار شود. اگر هر کدام از این مراحل با شکست مواجه شود، معامله انجام نمی‌شود.

2- خواص تراکنش هر تراکنش دارای چهار خاصیت است (معروف به ACID) که به شرح زیر می‌باشند:

2-1- خاصیت یکپارچگی (Atomicity) یکپارچگی به معنای این است که تراکنش باید به عنوان یک واحد منسجم (غیر قابل تفکیک) در نظر گرفته شود. در مثال مربوط به مبادله سهام، یکپارچگی به معنای این است که فروش سهام توسط سهامدار A و خرید آن سهام توسط سهامدار B، مستقل از هم قابل انجام نیستند و برای این که تراکنش کامل شود، هر دو عمل باید با موفقیت انجام شوند.

اجرای یکپارچه، یک عمل "همه یا هیچ" است. در عملیات یکپارچه، اگر هر کدام از دستورات موجود در تراکنش با شکست مواجه شوند، اجرای تمام دستورات قبلی خنثی می‌شود تا به جامعیت بانک اطلاعاتی آسیب نرسد.

2-2- خاصیت سازگاری (Consistency) سازگاری زمانی وجود دارد که هر تراکنش، سیستم را در یک حالت سازگار قرار دهد (چه تراکنش به طور کامل انجام شود و چه در اثر وجود خطایی خنثی گردد). در مثال مبادله سهام، سازگاری به معنای آن است که هر بدهکاری مربوط به حساب فروشنده، موجب همان میزان بستانکاری در حساب خریدار می‌شود. در SQL Server، سازگاری با راهکار ثبت فایل سابقه انجام می‌گیرد که تمام تغییرات را در بانک اطلاعاتی ذخیره می‌کند و جزئیات را برای ترمیم تراکنش ثبت می‌نماید. اگر سیستم در اثر تراکنش خراب شود، فرآیند ترمیم SQL Server با استفاده از این اطلاعات، تعیین می‌کند که آیا تراکنش با موفقیت انجام شده است یا خیر، و در صورت عدم موفقیت آن را خنثی می‌کند. خاصیت سازگاری تضمین می‌کند که بانک اطلاعاتی هیچگاه تراکنش‌های ناقص را نشان نمی‌دهد.

2-3- خاصیت تفکیک (Isolation) تفکیک موجب می‌شود هر تراکنش در فضای خودش و جدا از سایر تراکنش‌های دیگری که در سیستم انجام می‌گیرد، اجرا شود و نتایج هر تراکنش فقط در صورت کامل شدن آن قابل مشاهده است. اگر چندین تراکنش همزمان در سیستم در حال اجرا باشند، اصل تفکیک تضمین می‌کند که اثرات یک تراکنش تا کامل شدن آن، قابل مشاهده نیست. در مثال مربوط به مبادله سهام، اصل تفکیک به معنای این است که تراکنش بین دو سهامدار، مستقل از تمام تراکنش‌های دیگری است که در سیستم به مبادله سهام می‌پردازند و اثر آن وقتی برای افراد قابل مشاهده است که آن تراکنش کامل شده باشد. این اصل در مواردی که سیستم همزمان از چندین کاربر پشتیبانی می‌کند، مفید است.

2-4- پایداری (Durability) پایداری به معنای این است که تغییرات حاصل از نهای شدن تراکنش، حتی در صورت خرابی سیستم نیز پایدار می‌ماند. اغلب سیستم‌های مدیریت بانک اطلاعاتی رابطه‌ای، از طریق ثبت تمام فعالیت‌های تغییر دهنده‌ی داده‌ها در بانک اطلاعاتی، پایداری را تضمین می‌کنند. در صورت خرابی سیستم یا رسانه ذخیره سازی داده‌ها، سیستم قادر است آخرین

بهنگام سازی موفق را هنگام راه اندازی مجدد، بازیابی کند. در مثال مربوط به مبادله سهام، پایداری به معنای این است که وقتی انتقال سهام از سهامدار A به B با موفقیت انجام گردید، حتی اگر سیستم بعداً خراب شد، باید نتیجه‌ی آن را منعکس سازد.

3- مشکلات همزمانی (Concurrency Effects):

Dirty Read 3-1: زمانی روی می‌دهد که تراکنشی رکوردی را می‌خواند، که بخشی از تراکنشی است که هنوز تکمیل نشده است، اگر آن تراکنش Rollback شود اطلاعاتی از بانک اطلاعاتی دارید که هرگز روی نداده است. اگر سطح جداسازی تراکنش (پیش فرض) Read Committed باشد، این مشکل بوجود نمی‌آید.

Non-Repeatable Read 3-2: زمانی ایجاد می‌شود که رکوردی را دو بار در یک تراکنش می‌خوانید و در این اثنا یک تراکنش مجزای دیگر داده‌ها را تغییر می‌دهد. برای پیشگیری از این مسئله باید سطح جداسازی تراکنش برابر با Repeatable Read یا Serializable باشد.

Phantoms 3-3: با رکوردهای مرموزی سروکار داریم که گویی تحت تاثیر عبارات Update و Delete صادر شده قرار نگرفته اند. به طور خلاصه شخصی عبارت Insert را درست در زمانی که Update مان در حال اجرا بوده انجام داده است، و با توجه به اینکه ردیف جدیدی بوده و قفل وجود نداشته، به خوبی انجام شده است. تنها چاره این مشکل تنظیم سطح Serializable است و در این صورت بهنگام رسانی‌های جداول نباید درون بخش Where قرار گیرد، در غیر این صورت Lock خواهند شد.

Lost Update 3-4: زمانی روی می‌دهد که یک Update به طور موفقیت آمیزی در بانک اطلاعاتی نوشته می‌شود، اما به طور اتفاقی توسط تراکنش دیگری بازنویسی می‌شود. راه حل این مشکل بستگی به کد شما دارد و بایست به نحوی تشخیص دهید، بین زمانی که داده‌ها را می‌خوانید و زمانی که می‌خواهید آنرا بهنگام کنید، اتصال دیگری رکورد شما را بهنگام کرده است.

4- منابع قابل قفل شدن 6 منبع قابل قفل شدن برای SQL Server وجود دارد و آن‌ها سلسله مراتبی را تشکیل می‌دهند. هر چه

سطح قفل بالاتر باشد، Granularity کمتری دارد. در ترتیب آبشاری Granularity عبارتند از:

- **Database:** کل بانک اطلاعاتی قفل شده است، معمولاً طی تغییرات Schema بانک اطلاعاتی روی می‌دهد.
- **Table:** کل جدول قفل شده است، شامل همه اشیای مرتبط با جدول.
- **Extent:** کل Extent (متشکل از هشت Page) قفل شده است.
- **Page:** همه داده‌ها یا کلیدهای Index در آن Page قفل شده اند.
- **Key:** قفل در کلید مشخصی یا مجموعه کلید هایی Index وجود دارد. ممکن است سایر کلیدها در همان Index Page تحت تاثیر قرار نگیرند.
- **Row or Row Identifier (RID):** هر چند قفل از لحاظ فنی در Row Identifier قرار می‌گیرد ولی اساساً کل ردیف را قفل می‌کند.

5- تسریع قفل (Lock Escalation) و تاثیرات قفل روی عملکرد اگر تعداد آیتم‌های قفل شده کم باشد نگهداری سطح بهتری از Granularity (مثلاً RID به جای Page) معنی دار است. هرچند با افزایش تعداد آیتم‌های قفل شده، سربار مرتبط با نگهداری آن قفل‌ها در واقع باعث کاهش عملکرد می‌شود، و می‌تواند باعث شود قفل به مدت طولانی‌تری در محل باشد (هر چه قفل به مدت طولانی‌تری در محل باشد، احتمال این که شخصی آن رکورد خاص را بخواهد بیشتر است). هنگامی که تعداد قفل نگهداری شده به آستانه خاصی برسد آن گاه قفل به بالاترین سطح بعدی افزایش می‌یابد و قفل‌های سطح پایین‌تر نباید به شدت مدیریت شوند (آزاد کردن منابع و کمک به سرعت در مجادله). توجه شود که تسریع مبتنی بر تعداد قفل هاست و نه تعداد کاربران.

6- حالات قفل (Lock Modes): همانطور که دامنه وسیعی از منابع برای قفل شدن وجود دارد، دامنه ای از حالات قفل نیز وجود دارد.

Shared Locks (S) 6-1: زمانی استفاده می‌شود، که فقط باید داده‌ها را بخوانید، یعنی هیچ تغییری ایجاد نخواهید کرد. Shared Lock با سایر Shared Lock‌های دیگر سازگار است، البته قفل‌های دیگری هستند که با Shared Lock سازگار نیستند. یکی از کارهایی که Shared Lock انجام می‌دهد، ممانعت از انجام Dirty Read از طرف کاربران است.

6-2- Exclusive Locks (X): این قفل‌ها با هیچ قفل دیگری سازگار نیستند. اگر قفل دیگری وجود داشته باشد، نمی‌توان به Exclusive Lock دست یافت و همچنین در حالی که Exclusive Lock فعال باشد، به هر قفل جدیدی از هر شکل اجازه ایجاد شدن در منبع را نمی‌دهند.

این قفل از اینکه دو نفر همزمان به حذف کردن، بهنگام رسانی و یا هر کار دیگری مبادرت ورزند، پیشگیری می‌کند.

6-3- Update Locks (U): این قفل‌ها نوعی پیوند میان Exclusive Locks و Shared Locks هستند. برای انجام Update باید بخش Where را (در صورت وجود) تایید اعتبار کنید، تا دریابید فقط چه ردیف‌هایی را می‌خواهید بهنگام رسانی کنید. این بدان معنی است که فقط به Shared Lock نیاز دارید، تا زمانی که واقعاً بهنگام رسانی فیزیکی را انجام دهید. در زمان بهنگام سازی فیزیکی نیاز به Exclusive Lock دارید. Update Lock نشان دهنده این واقعیت است که دو مرحله مجزا در بهنگام رسانی وجود دارد، Shared Lock ای دارید که در حال تبدیل شدن به Exclusive Lock است. Update Lock تمامی Update Lock‌های دیگر را از تولید شدن باز می‌دارند، و همچنین فقط با Shared Lock و Intent Shared Lock‌ها سازگار هستند.

6-4- Intent Locks: با سلسله مراتب شی سر و کار دارد. بدون Intent Lock، اشیای سطح بالاتر نمی‌دانند چه قفلی را در سطح پایین‌تر داشته‌اید. این قفل‌ها کارایی را افزایش می‌دهند و 3 نوع هستند:

6-4-1- Intent Shared Lock (IS: Shared Lock): در نقطه پایین‌تری در سلسله مراتب، تولید شده یا در شرف تولید است. این نوع قفل تنها به Page و Table اعمال می‌شود.

6-4-2- Intent Exclusive Lock (IX): همانند Intent Shared Lock است اما در شرف قرار گرفتن در آیتم سطح پایین‌تر است.

6-4-3- Shared With Intent Exclusive (SIX: Shared Lock): در پایین سلسله مراتب شی تولید شده یا در شرف تولید است اما Intent Lock قصد اصلاح داده‌ها را دارد بنابراین در نقطه مشخصی تبدیل به Intent Exclusive Lock می‌شود.

6-5- Schema Locks: به دو شکل هستند:

6-5-1- Schema Modification Lock (Sch-M): تغییر Schema به شی اعمال شده است. هیچ پرس و جویی یا سایر عبارت‌های Create، Alter و Drop نمی‌توانند در مورد این شی در مدت قفل Sch-M اجرا شوند. با همه حالات قفل ناسازگار است.

6-5-2- Schema Stability Lock (Sch-S): بسیار شبیه به Shared Lock است، هدف اصلی این قفل پیشگیری از Sch-M است وقتی که قبلاً قفل‌هایی برای سایر پرس و جو-ها (یا عبارت‌های Create، Alter و Drop) در شی فعال شده‌اند. این قفل با تمامی انواع دیگر قفل سازگار است به جز با Sch-M.

6-6- Bulk Update Locks (BU): این قفل‌ها بارگذاری موازی داده‌ها را امکان‌پذیر می‌کنند، یعنی جدول در مورد هر فعالیت نرمال (عبارات T-SQL) قفل می‌شود، اما چندین عمل bcp یا Bulk Insert را می‌توان در همان زمان انجام داد. این قفل فقط با Sch-S و سایر قفل‌های BU سازگار است.

7- سطوح جداسازی (Isolation Level):

7-1- Read Committed (وضعیت پیش فرض): با Read Committed همه Shared Lock‌های ایجاد شده، به محض اینکه عبارت ایجاد کننده آنها تکمیل شود، به طور خودکار آزاد می‌شوند. به طور خلاصه قفل‌های مرتبط با عبارت Select به محض تکمیل عبارت Select آزاد می‌شوند و SQL Server منتظر پایان تراکنش نمی‌ماند. اگر تراکنش پرس و جویی را انجام می‌دهد که داده‌ها را اصلاح می‌کند (Insert، Delete و Update) قفل‌ها برای مدت تراکنش نگه داشته می‌شوند. با این سطح پیش فرض، می‌توانید مطمئن شوید جامعیت کافی برای پیشگیری از Dirty Read دارید، اما همچنان Non-Repeatable Read می‌تواند روی دهد.

7-2-Read Uncommitted: خطرناک‌ترین گزینه از میان تمامی گزینه‌ها است، اما بالاترین عملکرد را به لحاظ سرعت دارد. در واقع با این تنظیم سطح تجربه همه مسائل متعدد هم زمانی مانند Dirty Read امکان پذیر است. در واقع با تنظیم این سطح به SQL Server اعلام می‌کنیم هیچ قفلی را تنظیم نکرده و به هیچ قفلی اعتنا نکند، بنابراین هیچ تراکنش دیگری را مسدود نمی‌کنیم. می‌توانید همین اثر Read Uncommitted را با اضافه کردن نکته بهینه ساز **NOLOCK** در پرس و جوها بدست آورید.

7-3-Repeatable Read: سطح جداسازی را تا حدودی افزایش می‌دهد و سطح اضافی محافظت همزمانی را با پیشگیری از Dirty Read و همچنین Non-Repeatable Read فراهم می‌کند. پیشگیری از Non-Repeatable Read بسیار مفید است اما حتی نگه داشتن Shared Lock تا زمان پایان تراکنش می‌تواند دسترسی کاربران به اشیا را مسدود کند، بنابراین به بهره وری لطمه وارد می‌کند. نکته بهینه ساز برای این سطح **REPEATABLE READ** است.

7-4-Serializable: این سطح از تمام مسائل هم زمانی پیشگیری می‌کند به جز برای Lost Update. این تنظیم سطح به واقع بالاترین سطح آنچه را که سازگاری نامیده می‌شود، برای پایگاه داده فراهم می‌کند. در واقع فرآیند بهنگام رسانی برای کاربران مختلف به طور یکسان عمل می‌کند به گونه ای که اگر همه کاربران یک تراکنش را در یک زمان اجرا می‌کردند، این گونه می‌شد «پردازش امور به طور سریالی». با استفاده از نکته بهینه ساز **SERIALIZABLE** یا **HOLDLOCK** در پرس و جو شبیه سازی می‌شود.

7-5-Snapshot: جدترین سطح جداسازی است که در نسخه 2005 اضافه شد، که شبیه ترکیبی از Read Committed و Read Uncommitted است. به طور پیش فرض در دسترس نیست، در صورتی در دسترس است که گزینه **ALLOW_SNAPSHOT_ISOLATION** برای بانک اطلاعاتی فعال شده باشد. (برای هر بانک اطلاعاتی موجود در تراکنش) Snapshot مشابه Read Uncommitted هیچ قفلی ایجاد نمی‌کند. تفاوت اصلی آن‌ها در این است که تغییرات صورت گرفته در بانک اطلاعاتی را در زمان‌های متفاوت تشخیص می‌دهند. هر تغییر در بانک اطلاعاتی بدون توجه به زمان یا Commit شدن آن، توسط پرس و جو هایی که سطح جداسازی Read Uncommitted را اجرا می‌کنند، دیده می‌شود. با Snapshot فقط تغییراتی که قبل از شروع تراکنش، Commit شده اند، مشاهده می‌شود. از شروع تراکنش Snapshot، تمامی داده‌ها دقیقاً مشاهده می‌شوند، زیرا در شروع تراکنش Commit شده اند. **نکته:** در حالی که Snapshot توجهی به قفل‌ها و تنظیمات آنها ندارد، یک حالت خاص وجود دارد. چنانچه هنگام انجام Snapshot یک عمل Rollback (بازیافت) بانک اطلاعاتی در جریان باشد، تراکنش Snapshot قفل‌های خاصی را برای عمل کردن به عنوان یک مکان نگهدار و سپس انتظار برای تکمیل Rollback تنظیم می‌کند. به محض تکمیل Rollback، قفل حذف شده و Snapshot به طور طبیعی به جلو حرکت خواهد کرد.

Isolation level	Dirty read	Non-Repeatable read	Phantom
Read uncommitted	Yes	Yes	Yes
Read committed	No	Yes	Yes
Repeatable read	No	No	Yes
Snapshot	No	No	No
Serializable	No	No	No

نظرات خوانندگان

نویسنده: محمد

تاریخ:

۱۳۹۳/۰۲/۰۵ ۱۲:۳۷

سلام ، خسته نباشید. ممنون از زحماتی که برای این مقاله کشیدید. من به سوال داشتم درباره تراکنش (Transaction) : من توی پروژه ام از sqlTransaction و isolation سریالایز استفاده کردم در این بلاک، 9 جدول من مورد عملیات درج و یا ویرایش قرار می‌گیره (و البته دو وب سرویس نیز صدا زده می‌شه) و معمولا 2 یا 3 ثانیه برای ثبت این تراکنش زمان صرف میشه . تا اینجا مشکلی نیست ولی اگر چند کاربر مثلا 5 نفر این عملیات انجام بدن در بعضی از مواقع (به ندرت) به ترتیب عملیات برای هر نفر انجام میشه (همین توقع از سریالایز می‌ره). ولی در اکثر مواقع خطا deadlock می‌ده و یا خطا timeout و همچنین جاهای مختلف برنامه که به یکی از این 9 جدول select می‌شه به انها هم همین خطاها رو می‌ده . لطفا در صورت امکان راه حلی برای مشکل من بگین . ممنون

نویسنده: محمد رجبی

تاریخ:

۱۳۹۳/۰۲/۰۵ ۱۵:۴۵

با سلام و سپاس از محبت تان، چند نکته به ذهن من میرسد، که شاید راهگشا باشد:

- تعداد دستورات درون بلاک Transaction تا جایی که میسر است، کمینه باشد.
- عملیات مربوط به وب سرویس، خارج از Transaction انجام گیرد. (به عنوان یک نکته در کار با Transaction باید عملیات Input و Output خارج از Transaction انجام گیرد).
- طبیعتا هر چه درجه Isolation بالاتر باشد، Lock سختگیرانه‌تر برخورد می‌کند.
- برای مشکل بن بست، اگر می‌توانید ترتیب در اختیار گرفتن جداول را (پس از شناسائی محل بن بست) تغییر دهید و یا از جداول Temp استفاده کنید.
- چنانچه در قسمت هایی از برنامه همانطور که اشاره کردید فقط نیاز به خواندن مقادیر جداول دارید، از بهینه ساز پرس و جوی nolog استفاده کنید. برای مثال:

```
select * from tbl with (NOLOCK)
```