عنوان مقاله: تغییرات زبان T-SQL در SQL Server 2022 – قسمت دوم

نویسنده مقاله: تیم فنی نیک‌آموز

تاریخ انتشار: ۱۳ دی ۱۴۰۱

منبع: https://nikamooz.com/ t-sql-language-changes-in-sql-server-۲۰۲۲-part-۲

SQL Server در ورژن ۲۰۲۲ تغییراتی داشته است که من در مقاله قبلی به تغییرات زبان T-SQL پرداختم. در این مقاله قصد دارم تغییرات دیگر T-SQL را که در ورژن ۲۰۲۲ در پلتفرم پایگاه داده ایجاد شده را پوشش دهم.

در مقاله قبلی موضوعاتی از جمله DISTINCT FROM، DATE_BUCKET، GENERATE_SERIES ، GREATEST/LEAST، STRING_SPLIT و DATETRUNC مورد بررسی قرار گرفت. در این مقاله APPROX_PERCENTILE_CONT، APPROX_PERCENTILE_DISC و توابع bit manipulation مورد بررسی قرار خواهد گرفت. همچنین تغییراتی که روی FIRST_VALUE ، LAST_VALUE و LTRIM/RTRIM/TRIM اتفاق افتاده بررسی می‌شود.

در این مقاله من یک نگاه ساده به این ویژگی‌های جدید دارم و هنوز در حال آزمایش و یادگیری این ویژگی‌های جدید هستم. ارزیابی من از SQL Server 2022 از دیدگاه شخصی است که با ویژگی‌های جدید در این پلتفرم کار می‌کند تا در نهایت تصمیم بگیرد که آیا ارتقا دادن به ورژن ۲۰۲۲ ارزش دارد یا خیر. بنابراین در حال بررسی این نکته هستم که تغییرات جدید کجاها ممکن است به درد بخورد و باعث تسهیل در کدنویسی شود. من برای آزمایش‌های خود با SQL Server 2022 RC0 کار می‌کنم.

APPROX_PERCENTILE_CONT

در حال حاضر تابع PERCENTILE_CONT در T-SQL وجود دارد. این تابع برای برگرداندن مقدار از مجموعه‌ای از داده‌ها است که براساس درصد محاسبه شده است. تابع تحلیلیPERCENTILE_CONT، محاسبه percentile (درصد) را براساس توزیع پیوسته‌ای از مقدار ستون انجام می‌دهد. این کار شبیه به پیدا کردن میانه با مقدار درصد است. این تابع یک تابع window است که به عبارت OVER() نیاز دارد. یک کوئری نمونه از Docs در پایگاه داده AdventureWorks اجرا می‌کنم که خروجی به صورت زیر است:

```
SELECT DISTINCT Name AS DepartmentName
      ,PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY ph.Rate)
                            OVER (PARTITION BY Name) AS MedianCont
      ,PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY ph.Rate)
                            OVER (PARTITION BY Name) AS MedianDisc
FROM HumanResources.Department AS d
INNER JOIN HumanResources.EmployeeDepartmentHistory AS dh
    ON dh.DepartmentID = d.DepartmentID
INNER JOIN HumanResources.EmployeePayHistory AS ph
    ON ph.BusinessEntityID = dh.BusinessEntityID
WHERE dh.EndDate IS NULL;
```

lts Messages

DepartmentName	MedianCont	MedianDisc
Document Control	16.8269	16.8269
Engineering	34.375	32.6923
Executive	54.32695	48.5577
Facilities and Maintenance	9.25	9.25
Finance	19	19.00
Human Resources	17.42785	16.5865

تابع APPROX_PERCENTILE_CONT مقدار تقریبی همان تابع PERCENTILE_CONT را به دست می‌آورد و نیاز به خواندن همه داده‌ها ندارد. Microsoft Docs درمورد عملکرد این تابع به این صورت توضیح می‌دهد:

این تابع یک مقدار تقریبی را از مجموعه‌ای از مقادیر که براساس مقدار درصدی مشخص شده و براساس یک ویژگی خاص مرتب‌سازی شده، برمی‌گرداند. از آنجایی که این یک تابع تقریبی است، خروجی در محدوده‌ای از خطا قرار دارد. مقدار برگردانده شده توسط این تابع براساس توزیع پیوسته مقادیر ستون است و نتیجه interpolate می‌شود. به همین دلیل، خروجی ممکن است یکی از مقادیر موجود در مجموعه داده نباشد. یکی از موارد استفاده رایج برای این تابع، اجتناب از داده‌های پرت است. این تابع می‌تواند به عنوان جایگزینی برای تابع PERCENTILE_CONT برای مجموعه داده‌های بزرگ استفاده شود. چون در مقیاس بزرگ، خطای ناچیز قابل چشم‌پوشی است در مقایسه با دریافت مقدار دقیق، درحالی که زمان طولانی برای دریافت مقدار، سیستم باید معطل بماند.

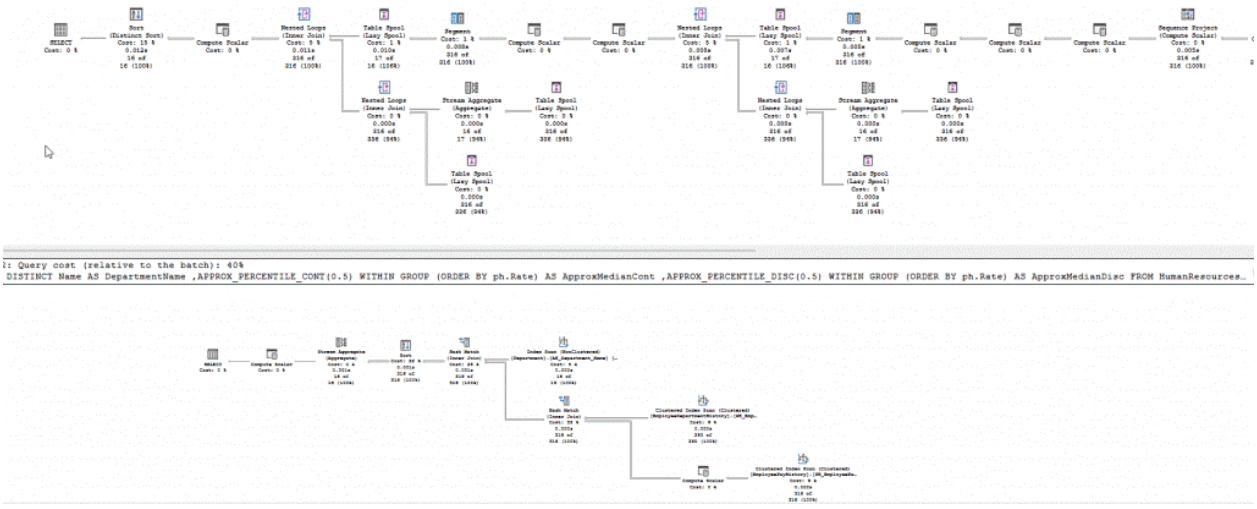
اساساً، اگر بتوانید یک مقدار تقریبی را بپذیرید و بخواهید منابع مورد نیاز برای محاسبات را به حداقل برسانید، این تابع مناسب است. این تابع، یک تابع window نیست و از عبارت ORDER استفاده نمی‌کند. یک عبارت grouping وجود دارد و می‌توان یک دستور را در آنجا مشخص نمود. کد زیر را روی مجموعه داده AdventureWorks اجرا می‌کنم، همان نتایج قبلی مشاهده می‌شود.

```
SELECT DISTINCT Name AS DepartmentName
      ,APPROX_PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY ph.Rate) AS ApproxMedianCont
      ,APPROX_PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY ph.Rate) AS ApproxMedianDisc
FROM HumanResources.Department AS d
INNER JOIN HumanResources.EmployeeDepartmentHistory AS dh
    ON dh.DepartmentID = d.DepartmentID
INNER JOIN HumanResources.EmployeePayHistory AS ph
    ON ph.BusinessEntityID = dh.BusinessEntityID
WHERE dh.EndDate IS NULL
GROUP BY Name;
GO
```

Messages

DepartmentName	ApproxMedianCont	ApproxMedianDisc
Document Control	16.8269	16.8269
Engineering	34.375	32.6923
Executive	54.32695	48.5577
Facilities and Maintenance	9.25	9.25
Finance	19	19.00
Human Resources	17.42785	16.5865
Information Services	32.4519	32.4519
Marketing	14.4231	14.4231

اگر به execution plan ها برای دو کوئری بالا نگاهی بیاندازیم، می‌بینیم که نسخه تقریبی از طرح بسیار ساده‌تری استفاده می‌کند. طرح بالایی تابع PERCENTILE_CONT و طرح پایینی، تابع APPROX_PERCENTILE_CONT است.



APPROX_PERCENTILE_DISC

تفاوت توابع PERCENTILE_DISC و APPROX_PERCENTILE_DISC نیز همانند روال تابع APPROX_PERCENTILE_CONT است که در بخش قبل توضیح داده شد. یعنی تابع APPROX_PERCENTILE_DISC مقداری را به صورت تقریبی برمی‌گرداند اما چون سریع‌تر از تابع PERCENTILE_DISC خروجی را مشخص می‌کند، منابع کمتری مصرف می‌کند و سیستم کمتر درگیر محاسبات خواهد بود. البته این مزیت به قیمت دقیق نبودن خروجی است که در بخش قبل هم اشاره شد، برای مجموعه داده‌های بزرگ، قابل پذیرش است.

همانند توابع بخش قبل، در AdventureWorks به ازای توابع دقیق و هم برای توابع تقریبی مقادیر یکسانی دریافت می‌کنم. شکل‌های طرح اجرا نیز یکسان به نظر می‌رسند.

عملیات بیتی

تعدادی توابع بیتی در ورژن ۲۰۲۲ اضافه شده است که به شما امکان می‌دهد مقادیر تغییر بیت را انجام دهید. رمزگذاری و دستکاری بیت به عنوان مجموعه‌ای از toggle ها استفاده می‌شود که تنظیم آن سریع‌تر و آسان‌تر از استفاده از مقادیر ستون جداگانه است. به عنوان مثال، @@options در واقع یکسری بیت است که تنظیم شده است.

برای توضیح ساده این موضوع، یک مثال می‌آورم. به عنوان مثال، اگر من ٦٤ را در نظر بگیرم، این عدد به صورت مقدار باینری، به صورت مجموعه‌ای از ۷ بیت ذخیره می‌شود. این مقدار به صورت ۱۰۰۰۰۰۰ است. می‌توانم با استفاده از تابع AND (&) آن در SELECT به صورت زیر اضافه کنم.

اجازه دهید به یک عدد نگاه کنیم. به عنوان مثال، اگر من ٦٤ را در نظر بگیرم، این عدد در یک مقدار باینری به عنوان مجموعه‌ای از ۷ بیت ذخیره می‌شود. مقدار باینری برابر ۱۰۰۰۰۰۰ است. به خروجی دستور SELECT زیر نگاه کنید، متوجه می‌شوید که هر بیت براساس توان ۲ است.

```
SELECT 64,
       CAST(64 & 64 AS BIT) AS bit7,
       CAST(64 & 32 AS BIT) AS bit6,
       CAST(64 & 16 AS BIT) AS bit5,
       CAST(64 & 8 AS BIT) AS bit4,
       CAST(64 & 4 AS BIT) AS bit3,
       CAST(64 & 2 AS BIT) AS bit2,
       CAST(64 & 1 AS BIT) AS bit1;
```

lts Messages

(No column name)	bit7	bit6	bit5	bit4	bit3	bit2	bit1
64	1	0	0	0	0	0	0

توابع بعدی که در این مقاله در مورد آن صحبت می‌کنیم، با عملیات‌های مختلف بر این بیت‌ها تأثیر می‌گذارند.

LEFT_SHIFT()

همان‌طور که ممکن است حدس زده باشید، تابع LEFT_SHIFT() برای جابجایی بیت‌ها به سمت چپ طراحی شده است. در این صورت، اگر این تابع را روی مقدار 64 اجرا کنم، نتیجه زیر ایجاد می‌شود:

```
SELET LEFT_SHIFT(64,1) as NewValue

NewValue
--------
128
```

فرمت تابع به دو پارامتر نیاز دارد.

- یک عدد صحیح یا یک مقدار باینری
- تعداد بیت برای جابجایی به سمت چپ

این تابع مقدار باینری یا عدد صحیح (integer) جدید را برمی‌گرداند. شیفت می‌تواند روی یک مقدار باینری اعمال شود، بنابراین مقدار خروجی BIGINT خواهد بود.

اگر من این تابع را روی عدد ۳۲ با شیفت ۱ اجرا کنم، در نهایت به عدد ٦٤ می‌رسم. این همان عددی است که انتظار داریم. اگر بخواهید به این دو مجموعه بیت با هم نگاه کنید به مثال زیر دقت کنید. کوئری زیر را اجرا می‌کنم:

```
DECLARE @number INT = 32, @new int;
SELECT left_shift (@number, 1)
SELECT @new = left_shift (@number, 1)
SELECT @number,
       CAST (@number & 128 AS BIT) AS bit8,
       CAST (@number & 64 AS BIT) AS bit7,
       CAST (@number & 32 AS BIT) AS bit6,
       CAST (@number & 16 AS BIT) AS bit5,
       CAST (@number & 8 AS BIT) AS bit4,
       CAST (@number & 4 AS BIT) AS bit3,
       CAST (@number & 2 AS BIT) AS bit2,
       CAST (@number & 1 AS BIT) AS bit1
UNION
SELECT @new,
       CAST (@new & 128 AS BIT) AS bit8,
       CAST (@new & 64 AS BIT) AS bit7,
       CAST (@new & 32 AS BIT) AS bit6,
       CAST (@new & 16 AS BIT) AS bit5,
       CAST (@new & 8 AS BIT) AS bit4,
       CAST (@new & 4 AS BIT) AS bit3,
       CAST (@new & 2 AS BIT) AS bit2,
       CAST (@new & 1 AS BIT) AS bit1;
```

نتایج به صورت زیر است. همان‌طور که می‌بینید، عدد ۱ به اندازه یک بیت به سمت چپ منتقل می‌شود. شیفت از عدد صفر برای پر کردن از سمت راست استفاده می‌کند.

(No column name)	bit8	bit7	bit6	bit5	bit4	bit3	bit2	bit1
32	0	0	1	0	0	0	0	0
64	0	1	0	0	0	0	0	0

اگر این روال را با عدد ۲۷ تکرار کنم، متوجه می‌شوید که خروجی دو برابر، یعنی ٥٤ می‌شود LEFT_SHIFT(). راهی برای ضرب در ۲ است. به عبارت دیگر، مقدار صحیح برای هر تغییر بیت، دو برابر می‌شود.

(No column name)	bit8	bit7	bit6	bit5	bit4	bit3	bit2	bit1
27	0	0	0	1	1	0	1	1
54	0	0	1	1	0	1	1	0

RIGHT_SHIFT()

همان طور که ممکن است حدس زده باشید، تابع RIGHT_SHIFT() برعکس LEFT_SHIFT() عمل می‌کند. این تابع بیت‌ها را به سمت راست حرکت می‌دهد و برای هر بیت جابجا شده، تقسیم بر ۲ برای عدد صحیح خواهد بود.

همان کد بالا استفاده کنیم، تابع RIGHT_SHIFT() را جایگزین می‌کنم. همان‌طور که مشاهده می‌کنید، اگر با ٦٤ شروع کنم و ۲ بار به راست شیفت کنم، 16 دریافت می‌کنم. این یعنی ۲/٦٤ = ۳۲ و ۲/۳۲ که برابر ١٦ خواهد بود. کوئری به صورت زیر است:

```
DECLARE @number INT = 64, @new INT, @shifts INT = 2;
SELECT right_shift (@number, @shifts)
SELECT @new = right_shift (@number, @shifts)
SELECT @number,
       CAST (@number & 128 AS BIT) AS bit8,
       CAST (@number & 64 AS BIT) AS bit7,
```

```
        CAST (@number & 32 AS BIT) AS bit6,
        CAST (@number & 16 AS BIT) AS bit5,
        CAST (@number & 8 AS BIT) AS bit4,
        CAST (@number & 4 AS BIT) AS bit3,
        CAST (@number & 2 AS BIT) AS bit2,
        CAST (@number & 1 AS BIT) AS bit1
UNION
SELECT @new,
        CAST (@new & 128 AS BIT) AS bit8,
        CAST (@new & 64 AS BIT) AS bit7,
        CAST (@new & 32 AS BIT) AS bit6,
        CAST (@new & 16 AS BIT) AS bit5,
        CAST (@new & 8 AS BIT) AS bit4,
        CAST (@new & 4 AS BIT) AS bit3,
        CAST (@new & 2 AS BIT) AS bit2,
        CAST (@new & 1 AS BIT) AS bit1;
```

و نتایج به صورت تصویری به صورت زیر خواهد بود. توجه کنید عدد ۱ در بیت ۷ قرار داشته به اندازه ۲ مکان به سمت راست جابه‌جا شده است.

(No column name)	bit8	bit7	bit6	bit5	bit4	bit3	bit2	bit1
16	0	0	0	1	0	0	0	0
64	0	1	0	0	0	0	0	0

با دو شیفت، عدد ٥٤ به چه عددی تبدیل می‌شود؟ عدد ۲۷ به طور مساوی بر ۲ تقسیم نمی‌شود. ممکن است حدس زده باشید که بخش عدد صحیح نتیجه یعنی (۲٦) را دریافت می‌کنیم و اعشار حذف می‌شود. در اینجا نتایج bit shift آمده است.

(No column name)	bit8	bit7	bit6	bit5	bit4	bit3	bit2	bit1
13	0	0	0	0	1	1	0	1
54	0	0	1	1	0	1	1	0

یک مثال دیگر برای دیدن شیفت‌ها، عدد ۱۰۰ را دو بار به سمت راست شیفت می‌دهیم. همه ۱ ها به سمت راست حرکت می‌کنند و ۰ ها از سمت چپ وارد می‌شوند.

(No column name)	bit8	bit7	bit6	bit5	bit4	bit3	bit2	bit1
25	0	0	0	1	1	0	0	1
100	0	1	1	0	0	1	0	0

BIT_COUNT()

تابع BIT_COUNT() برای شمارش تعداد بیت‌های تنظیم شده روی ۱ در یک عبارت عددی یا باینری طراحی شده است. دقیقاً نمی‌دانم کاربرد این تابع کجا می‌تواند باشد. شاید اگر چیزی شبیه حضور در کلاسی داشته باشیم که هر دانش آموز را در مکانی خاص داشته باشیم، این تابع استفاده شود.

به این مثال دقت کنید، این تابع به صورت زیر خروجی می‌دهد:

```
SELECT bit_count(64)
SELECT bit_count(32)
SELECT bit_count(27)
SELECT bit_count(54)
SELECT bit_count(25)
```

Messages

(No column name)
1

(No column name)
1

(No column name)
4

(No column name)
4

(No column name)
3

GET_BIT()

همان طور که انتظار می‌رود، GET_BIT() مقدار یک عبارت عددی (یا باینری) را برمی‌گرداند. مانند LEFT_SHIFT()، به پارامتر دومی نیاز داریم که افست بیت خواهد بود. به عنوان مثال اگر بخواهم بیت ٤ را از عدد ۱۰۰ دریافت کنم، انتظار دارم عدد صفر را برگرداند. البته دقت داشته باشید مکان بیت‌ها از صفر شروع می‌شود یعنی جایگاه اول، در واقع بیت صفرم است و جایگاه دوم بیت یکم است و به همین ترتیب.

```
DECLARE @number INT = 100, @new INT, @bit INT = 3;
SELECT @new = GET_BIT(@number, @bit)

SELECT @number AS Expression,
       @new AS BitValue,
       CAST(@number & 128 AS BIT) AS bit7,
       CAST(@number & 64 AS BIT) AS bit6,
       CAST(@number & 32 AS BIT) AS bit5,
       CAST(@number & 16 AS BIT) AS bit4,
       CAST(@number & 8 AS BIT) AS bit3,
       CAST(@number & 4 AS BIT) AS bit2,
       CAST(@number & 2 AS BIT) AS bit1,
       CAST(@number & 1 AS BIT) AS bit0
```

ults Messages

Expression	BitValue	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
100	0	0	1	1	0	0	1	0	0

SET_BIT()

تابع SET_BIT() دقیقاً نقطه مقابل GET_BIT() است. این تابع به شما امکان می‌دهد مقدار بیت را در عبارت تنظیم کنید.

در کد زیر بیت ۳ در عدد ۱۰۰ تنظیم می‌شود. مقدار بیت ۳ برابر صفر است و اگر تابع SET_BIT() پارامتر سوم نداشته باشد، روی عدد ۱ تنظیم می‌شود. این کار باعث می‌شود عدد ۸ (۲ به توان ۳) به عدد اولیه یعنی ۱۰۰ اضافه شود.

```
DECLARE @number INT = 100, @new INT, @bit INT = 3;
SELECT @new = SET_BIT(@number, @bit)
SELECT @number AS Expression,
       CAST(@number & 128 AS BIT) AS bit7,
       CAST(@number & 64 AS BIT) AS bit6,
       CAST(@number & 32 AS BIT) AS bit5,
       CAST(@number & 16 AS BIT) AS bit4,
       CAST(@number & 8 AS BIT) AS bit3,
       CAST(@number & 4 AS BIT) AS bit2,
       CAST(@number & 2 AS BIT) AS bit1,
       CAST(@number & 1 AS BIT) AS bit0
UNION
SELECT @new,
       CAST(@new & 128 AS BIT) AS bit7,
       CAST(@new & 64 AS BIT) AS bit6,
       CAST(@new & 32 AS BIT) AS bit5,
       CAST(@new & 16 AS BIT) AS bit4,
       CAST(@new & 8 AS BIT) AS bit3,
       CAST(@new & 4 AS BIT) AS bit2,
       CAST(@new & 2 AS BIT) AS bit1,
       CAST(@new & 1 AS BIT) AS bit0
   ;
GO
```

نتیجه به صورت زیر خواهد بود:

Expression	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
100	0	1	1	0	0	1	0	0
108	0	1	1	0	1	1	0	0

این تابع در واقع راهی برای اضافه یا کم کردن یک مقدار از یک عدد صحیح است.

FIRST_VALUE () / LAST_VALUE ()

البته این توابع در چند ورژن قبلی SQL Server بوده است اما در ورژن ۲۰۲۲ تغییراتی داشته است. در واقع دو عبارت جدید اضافه شده است. توضیحات MS Docs در مورد این دو تابع به این صورت است:

- IGNORE NULLS: هنگام محاسبه اولین مقدار روی یک پارتیشن، مقادیر null در مجموعه داده نادیده گرفته شود.

- RESPECT NULLS: هنگام محاسبه اولین مقدار روی یک پارتیشن، به مقادیر null در مجموعه داده توجه شود.

این بدان معناست که اگر مقادیر NULL در یک پارتیشن (مجموعه داده) وجود داشته باشد، می‌توانیم آن‌ها را در نظر بگیریم و یا در نظر نگیریم. به عنوان مثال، من یک مجموعه داده در یک جدول مجازی دارم که دارای چند bucket از داده‌ها است. به مثال زیر دقت کنید:

```
SELECT bucket,
FIRST_VALUE(val) OVER (PARTITION BY A.bucket ORDER BY A.bucketorder ROWS BETWEEN
UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS firstvalue,
LAST_VALUE(val) OVER (PARTITION BY A.bucket ORDER BY A.bucketorder ROWS BETWEEN
UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS lastvalue
 FROM ( VALUES
         ('a', 1,  'first'),
         ('a', 2,  NULL),
         ('a', 3,  'LAST'),
         ('b', 1,  NULL),
         ('b', 2,  'mid'),
         ('b', 3,  'last'),
         ('c', 1,  'first'),
         ('c', 2,  'mid1'),
         ('c', 3,  'mid2'),
         ('c', 4,  NULL)
 ) A(bucket, bucketorder, val)
```

اگر کد بالا را در ورژن ۲۰۱۹ اجرا کنم، خروجی به صورت زیر خواهد بود:

bucket	firstvalue	lastvalue
a	first	LAST
a	first	LAST
a	first	LAST
b	NULL	last
b	NULL	last
b	NULL	last
c	first	NULL
c	first	NULL
c	first	NULL
c	first	NULL

من می‌توانم در ورژن ۲۰۲۲ عبارت IGNORE NULLS را اضافه کنم. این عبارت بعد از تابع و قبل از عبارت OVER() اضافه شده است. کد به صورت زیر خواهد بود:

```
SELECT bucket,
FIRST_VALUE(val) IGNORE NULLS OVER (PARTITION BY A.bucket ORDER BY A.bucketorder
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS firstvalue,
LAST_VALUE(val) IGNORE NULLS  OVER (PARTITION BY A.bucket ORDER BY A.bucketorder
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS lastvalue
 FROM ( VALUES
         ('a', 1,  'first'),
         ('a', 2,  NULL),
         ('a', 3,  'LAST'),
         ('b', 1,  NULL),
         ('b', 2,  'mid'),
         ('b', 3,  'last'),
         ('c', 1,  'first'),
         ('c', 2,  'mid1'),
         ('c', 3,  'mid2'),
         ('c', 4,  NULL)
 ) A(bucket, bucketorder, val)
```

خروجی به صورت زیر خواهد بود. دقت داشته باشید روی همان مجموعه داده اجرا شده است.

bucket	firstvalue	lastvalue
a	first	LAST
a	first	LAST
a	first	LAST
b	mid	last
b	mid	last
b	mid	last
c	first	mid2
c	first	mid2
c	first	mid2
c	first	mid2

اگر RESPECT NULLS را اضافه کنم، همان نتایج سال ۲۰۱۹ را دریافت می‌کنم. یعنی عبارت RESPECT NULLS به صورت پیش فرض اعمال شده است.

LTRIM()

تغییری در تابع LTRIM() در SQL Server 2022 وجود دارد که به شما امکان می‌دهد کاراکترهای خاصی را برش (TRIM) دهید. پیش فرض یک space است، اما می‌توانید آن را تغییر دهید. به مثال‌های زیر دقت کنید:

```
SELECT STRINGVALUE,
  LEN(StringValue) AS OriginalLength,
  LTRIM(STRINGVALUE,' '),
  LEN(LTRIM(StringValue,' ')) AS TrimLength
FROM (
 VALUES
  ('   String1'),
  (' String2'),
  ('   String3   '),
  ('aaaString4bbbb'),
  ('   String5bbbb'),
  ('bbbString4bbbb')
 ) A(StringValue)
GO
```

در این کد، من space ها را از سمت چپ رشته‌ها حذف می‌کنم. اگر به رشته نگاه کنید، ٤ مقدار وجود دارد که در سمت چپ فاصله دارند. (string1, string2, string2, and string5) وقتی این کد را اجرا می‌کنم، نتایج به صورت زیر خواهد بود:

STRINGVALUE	OriginalLength	(No column name)	TrimLength
String1	10	String1	7
String2	8	String2	7
String3	10	String3	7
aaaString4bbbb	14	aaaString4bbbb	14
String5bbbb	14	String5bbbb	11
bbbString4bbbb	14	bbbString4bbbb	14

همان‌طور که می‌بینید space ها حذف شدند. در مثال زیر کاراکتر دیگری را حذف می‌کنم. کاراکتر a را با کد زیر حذف می‌کنم:

```
SELECT STRINGVALUE,
  LEN(StringValue) AS OriginalLength,
  LTRIM(STRINGVALUE,'a'),
  LEN(LTRIM(StringValue,'a')) AS TrimLength
FROM (
 VALUES
  ('   String1'),
  (' String2'),
  ('   String3   '),
  ('aaaString4bbbb'),
  ('   String5bbbb'),
  ('bbbString4bbbb')
 ) A(StringValue)
GO
```

در نتایج، فقط string4 در ابتدا دارای a است و هر ۳ مورد حذف می‌شوند.

STRINGVALUE	OriginalLength	(No column name)	TrimLength
String1	10	String1	10
String2	8	String2	8
String3	10	String3	10
aaaString4bbbb	14	String4bbbb	11
String5bbbb	14	String5bbbb	14
bbbString4bbbb	14	bbbString4bbbb	14

کد زیر چندین کاراکتر را حذف می‌کند:

```
SELECT STRINGVALUE,
  LEN(StringValue) AS OriginalLength,
  LTRIM(STRINGVALUE,'The'),
  LEN(LTRIM(StringValue,'The')) AS TrimLength
FROM (
 VALUES
  ('The quick brown fox'),
  ('The end of the story'),
  ('TheThe Boat')
 ) A(StringValue)
GO
```

نتایج به صورت زیر خواهد بود:

STRINGVALUE	OriginalLength	(No column name)	TrimLength
The quick brown fox	19	quick brown fox	16
The end of the story	20	end of the story	17
TheThe Boat	11	Boat	5

توجه داشته باشید، این کد کلمه "The" را حذف نمی‌کند، بلکه به دنبال هر یک از کاراکترهای ابتدای رشته می‌گردد و آن‌ها را حذف می‌کند. بنابراین، اگر یک "t"، "h" یا "e" در ابتدا باشد، حذف می‌شوند. این بدان معنی است که SELECT LTRIM('tehhts', 'the') فقط یک "s" را برمی‌گرداند.

RTRIM()

مانند LTRIM()، RTRIM() نیز تغییر کرده است تا اجازه دهد یک کاراکتر برای حذف مشخص شود. در اینجا چند مثال ارائه می‌کنم:

```
SELECT STRINGVALUE,
  LEN(StringValue) AS OriginalLength,
  RTRIM(STRINGVALUE,'b'),
  LEN(RTRIM(StringValue,'b')) AS TrimLength
FROM (
 VALUES
  ('   String1'),
  (' String2'),
  ('   String3   '),
  ('aaaString4bbbb'),
  ('   String5bbbb'),
  ('String6bbbb')
 ) A(StringValue)
GO
```

نتایج شبیه به LTRIM() هستند، اما از سمت راست رشته‌ها، کاراکترهایی حذف شده است.

STRINGVALUE	OriginalLength	(No column name)	TrimLength
String1	10	String1	10
String2	8	String2	8
String3	10	String3	10
aaaString4bbbb	14	aaaString4	10
String5bbbb	14	String5	10
String6bbbb	11	String6	7

این تابع یک عملکرد مفید ارائه داده است زیرا گاهی اوقات کاراکترهای اضافی را دریافت می‌کنیم که توسط مشتریان وارد می‌شوند و می‌خواهیم کاراکترهایی غیر از کاراکتر space را حذف کنیم.

TRIM()

تابع TRIM() نیز به صورت مشابه دو تابع قبلی بهبود یافته است، اما با سینتکسی متفاوت. سینتکس جدید برای ورژن ۲۰۲۲ در Docs به صورت زیر نشان داده شده است:

TRIM ([LEADING | TRAILING | BOTH] [characters FROM] string)

در سینتکس بالا از LEADING برای حالت LTRIM، از TRAILING برای حالت RTRIM و از BOTH برای حالت TRIM استفاده می‌شود. فرض کنید من یک رشته دارم که شامل کاراکترهایی است که می‌خواهم حذف کنم. به مثال زیر دقت کنید:

```
SELECT STRINGVALUE,
  LEN(StringValue) AS OriginalLength,
  TRIM (BOTH 'b' FROM STRINGVALUE),
  LEN (TRIM(BOTH 'b' FROM STRINGVALUE)) AS TrimLength
FROM (
 VALUES
  ('  b String1b'),
  (' String2b'),
  ('b   String3   '),
  ('aaaString4bbbb'),
  ('   String5bbbb'),
  ('bString6bbbb')
 ) A(StringValue)
GO
```

اگر به رشته‌ها نگاه کنید، متوجه می‌شوید که در ابتدا یا انتهای آن‌ها کاراکتر ‘b’ وجود دارد. می‌خواهیم b ها حذف شوند. نتایج به صورت زیر است:

STRINGVALUE	OriginalLength	(No column name)	TrimLength
b String1b	12	b String1	11
String2b	9	String2	8
b String3	11	String3	10
aaaString4bbbb	14	aaaString4	10
String5bbbb	14	String5	10
bString6bbbb	12	String6	7

در رشته اول، اولین b را حذف نمی‌کند؛ زیرا ابتدای آن با کاراکتر space شروع شده و ما در پارامتر جستجوی خود space را اضافه نکرده بودیم. اما در رشته سوم b حذف شده است. در رشته ششم هر دو حالت leading و trailing درنظر گرفته شده است و همه b ها را حذف کرده است.

نکته مهم در این تابع آن است که تصمیم صحیح بگیرید که با هر پارامتر جستجو از کدام حالت‌های leading، trailing و یا both استفاده کنید تا نتیجه دلخواهتان به دست آید.

خلاصه

برخی از این توابع که به تازگی در ورژن ۲۰۲۲ اضافه شدند، عملکردهای جالبی دارند. توابع تقریبی، کابردی هستند و می‌توانند منابع را ذخیره کنند .موارد تغییر بیت باعث می‌شود به این فکر کنم که کجا ممکن است در رمزگذاری برخی سوئیچ‌ها استفاده کنم. عملکردهای TRIM پیشرفت‌های خوبی داشته‌اند و به نظر می‌رسد در تسهیل کدنویسی کمک کنند.