LEAD und LAG mit T-SQL

Die Funktionen LEAD() und LAG() kennen viele bereits als MDX-Funktionen und wissen, die Möglichkeiten zu nutzen, die diese Funktionen bieten. Seit SQL-Server 2012 stehen diese Funktionen auch in T-SQL für relationale Datenbankabfragen zur Verfügung. Im Folgenden wird die Verwendung dieser Funktionen anhand von Anwendungsbeispielen erläutert.

Seit der Version SQL-Server 2012 stehen die Funktionen LEAD() und LAG() als analytische Funktionen auch in T-SQL zur Verfügung. Ähnlich wie in MDX, wo sich die nachfolgenden bzw. vorausgehenden Elemente in einem Set ermitteln lassen, kann in T-SQL mit LEAD() und LAG() auf nachfolgende bzw. vorausgehende Zeilen einer SELECT-Anweisung zugegriffen werden.

Im Folgenden wird die Verwendung am Beispiel der LAG()-Funktion erläutert, mit der auf den Inhalt von vorausgehenden Zeilen einer SELECT-Anweisung zugegriffen werden kann. Dabei ist die Verwendung der LEAD()-Funktion analog, nur eben in die entgegengesetzte Richtung.

Syntax

Die Syntax der LAG()-Funktion ist recht überschaubar:

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

scalar_expression

Der Rückgabewert auf Basis des angegebenen Offsets – z.B. eine Spalte oder Berechnung aus mehreren Spalteninhalten aus der über das Offset angegebenen vorausgegangenen Zeile.

offset

Abstand der vorausgehenden Zeile, aus der ein Wert abgerufen werden soll. Wenn die vorausgehende Zeile nicht existiert, wird der angegebene default-Wert zurückgegeben. Der Wert für offset muss eine positive ganze Zahl sein. Wird kein offset angegeben, dann wird ein Abstand von 1 verwendet.

default

Gibt den Wert an, der zurückgegeben wird, falls scalar_expression am angegebenen offset NULL ist. Wenn kein Standardwert angegeben ist, wird NULL zurückgegeben. default muss mit scalar_expression typkompatibel sein.

OVER ([partition_by_clause] order_by_clause)

Die partition_by_clause unterteilt das Resultset in Partitionen, auf die die LAG()-Funktion angewendet wird. Ohne Angabe der partition_by_clause wird das gesamte Abfrageergebnis als einzelne Partition verarbeitet. Mit der order_by_clause wird ein Sortierkriterium angegeben, nach dem die Daten innerhalb der Partition(en) sortiert werden. Die Angabe der order_by_clause ist nicht optional!

Beispiele

Es folgen Beispiele auf der Basis der Datenbank AdventureWorks2012. Die Datenbank kann kostenlos bei GitHub heruntergeladen werden:

https://github.com/Microsoft/sql-server-samples/releases/tag/adventureworks

Werte aus verschiedenen Quartalen vergleichen

Mit der folgenden Abfrage werden die Quartals-Verkaufszahlen für einen bestimmten Mitarbeiter ermittelt und der jeweils vorausgehende Wert mit Hilfe der LAG()-Funktion in einer eigenen Spalte ausgegeben.

```
SELECT
        BusinessEntityID,
        QuotaDate AS SalesQuarter,
        SalesQuota AS CurrentQuota,
        LAG(SalesQuota, 1,0) OVER (ORDER BY QuotaDate) AS PreviousQuota
FROM Sales.SalesPersonQuotaHistory
WHERE BusinessEntityID = 275;
```

Im Resultset lässt sich die Anwendung der LAG()-Funktion klar nachvollziehen:

BusinessEntityID	SalesQuarter	CurrentQuota	PreviousQuota
275	2005-07-01 00:00:00.000	367000,00	0,00
275	2005-10-01 00:00:00.000	556000,00	367000,00
275	2006-01-01 00:00:00.000	502000,00	556000,00
275	2006-04-01 00:00:00.000	550000,00	502000,00
275	2006-07-01 00:00:00.000	1429000,00	550000,00
275	2006-10-01 00:00:00.000	1324000,00	1429000,00
275	2007-01-01 00:00:00.000	729000,00	1324000,00
275	2007-04-01 00:00:00.000	1194000,00	729000,00
275	2007-07-01 00:00:00.000	1575000,00	1194000,00
275	2007-10-01 00:00:00.000	1218000,00	1575000,00
275	2008-01-01 00:00:00.000	849000,00	1218000,00
275	2008-04-01 00:00:00.000	869000,00	849000,00

Abbildung 1: Anwendung der LAG()-Funktion

Werte aus verschiedenen Jahren vergleichen mit LAG() und SUM()

Auf der gleichen Datenbasis wie im vorherigen Beispiel, sollen nun die Jahreswerte des Mitarbeiters ermittelt und zusammen mit dem entsprechenden Vorjahreswert und der Abweichung zum Vorjahreswert ausgegeben werden.

```
SELECT
        BusinessEntityID,
        YEAR(QuotaDate) AS SalesYear,
        SUM(SalesQuota) AS CurrentQuota,
        LAG(SUM(SalesQuota), 1,0) OVER (ORDER BY YEAR(QuotaDate)) AS
PreviousQuota,
        SUM(SalesQuota)
      - LAG(SUM(SalesQuota), 1,0) OVER (ORDER BY YEAR(QuotaDate)) AS
Deviation
FROM Sales.SalesPersonQuotaHistory
WHERE BusinessEntityID = 275
GROUP BY BusinessEntityID, YEAR(QuotaDate);
```

An diesem Beispiel lässt sich gut nachvollziehen, dass auch die Verwendung von Aggregat-Funktionen innerhalb der LAG()-Funktion keinerlei Probleme bereitet:

BusinessEntityID	SalesYear	CurrentQuota	PreviousQuota	Deviation
275	2005	923000,00	0,00	923000,00
275	2006	3805000,00	923000,00	2882000,00
275	2007	4716000,00	3805000,00	911000,00
275	2008	1718000,00	4716000,00	-2998000,00

Abbildung 2: Verwendung von Aggregat-Funktionen innerhalb der LAG()-Funktion

Gruppierter Wertevergleich mit PARTITION_BY

Nun sollen die Jahreswerte mit dem Vergleichswert des Vorjahres für alle Mitarbeiter ermittelt werden. Dafür wird der PARTITION_BY Parameter verwendet.

```
SELECT
        BusinessEntityID,
        YEAR(QuotaDate) AS SalesYear,
        SUM(SalesQuota) AS CurrentQuota,
        LAG(SUM(SalesQuota), 1,0)
          OVER (PARTITION BY BusinessEntityID ORDER BY YEAR(QuotaDate)) AS
PreviousQuota
FROM Sales.SalesPersonQuotaHistory
GROUP BY BusinessEntityID, YEAR(QuotaDate);
```

BusinessEntityID	SalesYear	CurrentQuota	PreviousQuota
280	2006	1001000,00	634000,00
280	2007	1183000,00	1001000,00
280	2008	733000,00	1183000,00
281	2005	1009000,00	0,00
281	2006	2813000,00	1009000,00
281	2007	2644000,00	2813000,00
281	2008	1338000,00	2644000,00
282	2005	1292000,00	0,00
282	2006	2114000,00	1292000,00
282	2007	2293000,00	2114000,00
282	2008	1399000,00	2293000,00
283	2005	579000,00	0,00
283	2006	1371000,00	579000,00
283	2007	1438000,00	1371000,00
283	2008	637000,00	1438000,00
284	2006	321000,00	0,00
284	2007	1481000,00	321000,00
284	2008	951000,00	1481000,00
285	2007	172000,00	0,00
285	2008	33000,00	172000,00
286	2007	867000,00	0,00
286	2008	820000,00	867000,00
287	2006	108000,00	0,00
287	2007	651000,00	108000,00
287	2008	117000,00	651000,00
288	2007	1294000,00	0,00
288	2008	993000,00	1294000,00
289	2006	3160000,00	0,00
289	2007	5142000,00	3160000,00
289	2008	2212000,00	5142000,00
290	2006	1002000,00	0,00
290	2007	2940000,00	1002000,00
290	2008	1615000,00	2940000,00

Abbildung 3: Verwendung von PARTITION_BY Parameter

Fazit

Die mit SQL-Server 2012 neu eingeführten analytischen Funktionen LEAD() und LAG() lassen sich an vielen Stellen verwenden, an denen sonst eine Tabelle mit sich selbst über JOIN verbunden werden müsste. Neben der Vereinfachung bringt die Verwendung von LEAD() bzw. LAG() auch Performance-Vorteile mit sich.

In unseren Projekten lässt sich die LAG()-Funktion immer dann einsetzen, wenn wir mit Bestandsdaten aus einem Vorsystem versorgt werden und die Daten im ETL-Prozess dekumulieren müssen. Eine einfachere Variante als die Verwendung von LAG() gibt es nicht.

Auch in bestehenden Projekten sollten wir prüfen, ob Datenbankabfragen zur Dekumulation bestehen. Es wäre interessant zu wissen, wie hoch der Performance-Gewinn ist, wenn man die self-joins durch Verwendung der LAG()-Funktion ersetzt. Schneller wird es auf alle Fälle sein.