Tipps & Tricks: Analytische Funktionen

Bereich:	DBA, SQL	Erstellung:	03/2005 HA
Versionsinfo:	10.1, 10.2, 11.1, 11.2	Letzte Überarbeitung:	06/2009 MA

Analytische Funktionen

Analytische Funktionen fassen, ähnlich wie die Gruppenfunktionen, mehr als eine Zeile der Ergebnismenge zu einem Rückgabewert zusammen. In der Tat kann eine ganze Reihe von Funktionen sowohl als Gruppenfunktionen als auch als analytische Funktionen eingesetzt werden. Im Gegensatz zu Gruppenfunktionen wird aber für jede Zeile der Ergebnismenge einzeln ein Wert ausgegeben.
Optional kann (über die analytische Klausel, s.u.) angegeben werden, welcher Anteil der Ergebnismenge in das Resultat einfließen bzw. wonach gruppiert werden soll. Ohne diese Angaben wird die komplette Ergebnismenge mit einbezogen. Analytische Funktionen sind v.a. für den Einsatz im Data Warehouse-Bereich gedacht.

Syntax:

Da analytische Funktionen erst unmittelbar vor der ORDER-BY-Klausel ausgewertet werden, sind sie nur in der SELECT-Liste und in der ORDER-BY-Klausel zulässig. Die generelle Syntax des Aufrufs sieht, schematisch dargestellt, folgendermaßen aus:

funktion([argumente]) **OVER** ([*analytische Klausel*])

Die analytische Klausel kann folgende Klauseln enthalten:

- **PARTITION BY:** Sie dient zur Gruppierung der Ergebnismenge, ähnlich der GROUP BY-Klausel bei Gruppenfunktionen. Nur diejenigen Zeilen der Ergebnismenge, die im angegebenen Kriterium übereinstimmen, werden zusammengefasst. Ohne Angabe dieser Klausel wird die komplette Ergebnismenge als eine Partition betrachtet.
- **ORDER BY:** Sie gibt eine Sortierung der Zeilen innerhalb der Partition vor, die allerdings nicht zwingend der Ausgabe-Reihenfolge entsprechen muss(!). Stattdessen bewirkt diese Klausel, dass nicht mehr alle Zeilen einer Partition zusammengefasst werden, sondern nur noch ein bestimmtes "Fenster" daraus. Standardmäßig besteht dieses Fenster aus allen Zeilen der jeweiligen Partition bis zur jeweils aktuellen Zeile. Das entspricht dem Default der Windowing-Klausel (s.u.).
- **"Windowing Klausel":** Damit kann das "Fenster" der Ergebnismenge genauer definiert werden, das zu Teilergebnissen zusammengefasst werden soll. Voraussetzung für diese Klausel ist die Angabe der ORDER-BY-Klausel. Bei diesem Fenster kann es sich um eine Einschränkung der Zeilenzahl (Schlüsselwort **ROWS**) oder um eine logische Einschränkung auf Werte-Basis (Schlüsselwort **RANGE**) handeln.

Angegeben werden entweder Start- und Endpunkt oder nur der Startpunkt des Fensters. Wird nur der Startpunkt angegeben, so ist der Endpunkt automatisch die aktuelle Zeile.

Mögliche Werte für Start- oder Endpunkt sind:

- **UNBOUNDED PRECEDING:** alle bisherigen Zeilen ab Beginn der Partition
- **<n> PRECEDING:** die letzten n Zeilen vor der aktuellen Zeile
- **UNBOUNDED FOLLOWING:** alle nachfolgenden Zeilen bis zum Ende der Partition
- **<n> FOLLOWING:** die nächsten n Zeilen nach der aktuellen Zeile

- **CURRENT_ROW:** die aktuelle Zeile

Die Angabe erfolgt in der Form:

- **ROWS | RANGE BETWEEN** <startpunkt> **AND** <endpunkt>
- **ROWS | RANGE** <startpunkt>

Bei Auslassung der Windowing-Klausel lautet der Default "RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW".

Je nach Funktion kann eine Klausel unzulässig oder sogar nötig sein.
Die Klauseln sollen anhand von Beispielen verdeutlicht werden. Als "Prototyp" für die analytischen Funktionen wird die Funktion AVG zur Berechnung des Durchschnitts verwendet, die allgemein bekannt sein dürfte, wenn auch in der Syntax als Gruppenfunktion.

Beispiel 1: Ohne analytische Klausel

```
SQL> SELECT ename, job, sal,
     AVG(sal) OVER() durchschnitt
     FROM emp;

ENAME      JOB              SAL DURCHSCHNITT
---------- --------- ---------- ------------
SMITH      CLERK            800   2073,21429
ALLEN      SALESMAN        1600   2073,21429
WARD       SALESMAN        1250   2073,21429
JONES      MANAGER         2975   2073,21429
MARTIN     SALESMAN        1250   2073,21429
BLAKE      MANAGER         2850   2073,21429
CLARK      MANAGER         2450   2073,21429
SCOTT      ANALYST         3000   2073,21429
KING       PRESIDENT       5000   2073,21429
TURNER     SALESMAN        1500   2073,21429
ADAMS      CLERK           1100   2073,21429
JAMES      CLERK            950   2073,21429
FORD       ANALYST         3000   2073,21429
MILLER     CLERK           1300   2073,21429
```

Beispiel 2: Mit PARTITION BY- Klausel

```
SQL> SELECT ename, job, sal,
     AVG(sal) OVER(PARTITION BY job) durchschnitt
     FROM emp;

ENAME      JOB              SAL DURCHSCHNITT
---------- --------- ---------- ------------
SCOTT      ANALYST         3000         3000
FORD       ANALYST         3000         3000
```

```
SMITH      CLERK            800       1037,5
ADAMS      CLERK           1100       1037,5
MILLER     CLERK           1300       1037,5
JAMES      CLERK            950       1037,5
JONES      MANAGER         2975   2758,33333
CLARK      MANAGER         2450   2758,33333
BLAKE      MANAGER         2850   2758,33333
KING       PRESIDENT       5000         5000
ALLEN      SALESMAN        1600         1400
MARTIN     SALESMAN        1250         1400
TURNER     SALESMAN        1500         1400
WARD       SALESMAN        1250         1400
```

Beispiele 3 und 4: Mit PARTITION BY- und ORDER BY- Klauseln

```
SQL> SELECT ename, job, sal,
     AVG(sal) OVER(PARTITION BY job ORDER BY ename) durchschnitt
     FROM emp;

ENAME      JOB              SAL DURCHSCHNITT
---------- --------- ---------- ------------
FORD       ANALYST          3000         3000
SCOTT      ANALYST          3000         3000
ADAMS      CLERK            1100         1100
JAMES      CLERK             950         1025
MILLER     CLERK            1300   1116,66667
SMITH      CLERK             800       1037,5
BLAKE      MANAGER          2850         2850
CLARK      MANAGER          2450         2650
JONES      MANAGER          2975   2758,33333
KING       PRESIDENT        5000         5000
ALLEN      SALESMAN         1600         1600
MARTIN     SALESMAN         1250         1425
TURNER     SALESMAN         1500         1450
```

```
SQL> SELECT ename, job, sal,
     AVG(sal) OVER(PARTITION BY job ORDER BY ename
     ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
     durchschnitt
     FROM emp;

ENAME      JOB              SAL DURCHSCHNITT
---------- --------- ---------- ------------
FORD       ANALYST          3000         3000
SCOTT      ANALYST          3000         3000
ADAMS      CLERK            1100       1037,5
JAMES      CLERK             950       1037,5
MILLER     CLERK            1300       1037,5
```

```
SMITH      CLERK            800      1037,5
BLAKE      MANAGER         2850   2758,33333
CLARK      MANAGER         2450   2758,33333
JONES      MANAGER         2975   2758,33333
KING       PRESIDENT       5000         5000
ALLEN      SALESMAN        1600         1400
MARTIN     SALESMAN        1250         1400
TURNER     SALESMAN        1500         1400
WARD       SALESMAN        1250         1400
```

Übersicht über analytische Funktionen:

Bei den folgenden Funktionen, die sowohl als analytische Funktionen als auch als Gruppenfunktionen zulässig sind, sind alle aufgelisteten Klauseln zulässig, keine ist verpflichtend. Da ihr Gebrauch als Gruppenfunktion weitgehend bekannt sein dürfte, wird auf sie nicht näher eingegangen.

- **AVG**: Ermittlung des Durchschnitts
- **COUNT**: Ermittlung der Anzahl an Werten
- **MIN**: Ermittlung des Minimums
- **MAX**: Ermittlung des Maximums
- **SUM**: Aufsummierung der Werte

DENSE_RANK, RANK und **ROW_NUMBER**

An diese Funktionen wird kein Argument übergeben; die ORDER BY-Klausel ist verpflichtend, eine Windowing-Klausel dagegen ist nicht zulässig. Sie liefern einen Rang bzw. eine Zeilennummer innerhalb der Partition zurück, wobei die ORDER-By-Klausel vorgibt, wonach sich dieser Rang / diese Zeilennummer richten soll.
Bei gleichem Wert geben RANK und DENSE_RANK auch den gleichen Rang zurück; ROW_NUMBER dagegen setzt die Zählung einfach fort, wobei die Reihenfolge nicht-deterministisch ist.
Der Unterschied zwischen RANK und DENSE_RANK besteht darin, dass RANK nach gleichen Werten eine Lücke in der Rangfolge lässt, DENSE_RANK dagegen nicht.

Beispiel:

```
SQL> SELECT ename, job, sal,
     DENSE_RANK() OVER(PARTITION BY job ORDER BY sal) drang,
     RANK() OVER(PARTITION BY job ORDER BY sal) rang,
     ROW_NUMBER() OVER(PARTITION BY job ORDER BY sal) nummer
     FROM emp
     WHERE job IN ('ANALYST', 'SALESMAN');

ENAME      JOB              SAL      DRANG       RANG     NUMMER
---------- --------- ---------- ---------- ---------- ----------
SCOTT      ANALYST          3000          1          1          1
FORD       ANALYST          3000          1          1          2
WARD       SALESMAN         1250          1          1          1
MARTIN     SALESMAN         1250          1          1          2
TURNER     SALESMAN         1500          2          3          3
ALLEN      SALESMAN         1600          3          4          4
```

CUME_DIST, NTILE und **RATIO_TO_REPORT**

Diese Funktionen sagen etwas aus über die Verteilung der Werte:
Bei CUME_DIST, das ohne Übergabe eines Arguments aufgerufen wird, liegt der Rückgabewert zwischen 0 (exclusive) und 1 (inclusive). Er gibt den Anteil der Zeilen an, die kleiner oder gleich dem jeweiligen Wert des ORDER BY-Ausdrucks sind.
An NTILE dagegen wird mitgegeben, in wie viele gleich große Buckets der Wertebereich aufgeteilt werden soll. Returnwert ist die Nummer des Buckets, zu dem der Wert gehört. Da sich die Anzahl der zu einem Bucket zusammengefassten Werte um maximal 1 unterscheiden darf, können gleiche Werte auch in unterschiedlichen Buckets landen.
RATIO_TO_REPORT gibt den Anteil des Wertes an der Gesamtsumme der Partition zurück, wobei mitgegeben werden muss, wonach dieser Anteil berechnet werden soll. Für diese Funktion sind alle Klauseln optional und zulässig.
Für CUME_DIST und NTILE dagegen ist die ORDER BY-Klausel wiederum verpflichtend, eine Windowing-Klausel erneut nicht zulässig.

Beispiel:

```
SQL>  SELECT ename, job, sal,
      CUME_DIST() OVER(PARTITION BY job ORDER BY sal) cume,
      NTILE(3) OVER (PARTITION BY job ORDER BY sal) bucket,
      RATIO_TO_REPORT(sal) OVER() anteil
      FROM emp
      WHERE job IN ('ANALYST', 'SALESMAN');

ENAME      JOB         SAL       CUME     BUCKET     ANTEIL
---------- --------- ----- ---------- ---------- ----------
SCOTT      ANALYST    3000          1          1  ,25862069
FORD       ANALYST    3000          1          2  ,25862069
WARD       SALESMAN   1250         ,5          1 ,107758621
MARTIN     SALESMAN   1250         ,5          1 ,107758621
TURNER     SALESMAN   1500        ,75          2 ,129310345
ALLEN      SALESMAN   1600          1          3 ,137931034
```

LAG und **LEAD**

Diese beiden Funktionen bieten ohne Self-Join Zugriff auf andere Zeilen in der Ergebnismenge. Die ORDER BY-Klausel ist wiederum verpflichtend, eine Windowing-Klausel erneut nicht zulässig. Das erste Argument gibt an, welcher Wert zurückgegeben werden soll. Optional kann noch angegeben werden, wie viele Zeilen (Default: 1) nach vorne (LEAD) bzw. nach hinten (LAG) gegangen werden soll, und was als Default bei Überschreiten der Partition ausgegeben werden soll.

Beispiel:

```
SQL> SELECT ename, job, sal,
     LAG(sal,2,0) OVER(ORDER BY sal) zwei_vorher,
     LEAD(ename) OVER(ORDER BY sal) naechster
     FROM emp
     WHERE job IN ('ANALYST', 'SALESMAN');
```

```
ENAME      JOB         SAL ZWEI_VORHER NAECHSTER
---------- --------- ----- ---------- ----------
WARD       SALESMAN   1250          0 MARTIN
MARTIN     SALESMAN   1250          0 TURNER
TURNER     SALESMAN   1500       1250 ALLEN
ALLEN      SALESMAN   1600       1250 SCOTT
SCOTT      ANALYST    3000       1500 FORD
FORD       ANALYST    3000       1600
```

Diese Liste ist unvollständig. Es gibt noch eine Reihe weiterer analytischer Funktionen, v.a. zur Berechnung statistischer Werte (Standardabweichung, Varianz, Korrelationskoeffizient...). Eine vollständige Liste sowie weitere Informationen zur Syntax finden Sie in der Oracle Dokumentation.