Analysemöglichkeiten mit SQL: Mehr als SUM und GROUP BY

Carsten Czarski
ORACLE Deutschland B.V. & Co KG
München

Einleitung

Die Verwendung von Berichten und damit von SQL-Abfragen aller Art ist im BI- und DWH-Umfeld alltäglich. Das Zusammenfassen von Daten mit Hilfe von Aggregatsfunktionen wie AVG, SUM, MIN oder MAX ist dabei nichts Besonderes. Der Vortrag stellt verschiedene Aggregatsfunktionen und deren Anwendung in SQL-Abfragen vor. Wir beginnen mit einfachen GROUP BY-Varianten, leiten über zu weniger bekannten Funktionen wie der linearen Regression, betrachten danach analytische Funktione, bei denen das "Datenfenster" zur Anwendung der Aggregatsfunktion dynamisch definiert werden kann und schließen mit SQL-Funktionen für besondere Aufgaben wie Pivoting oder reguläre Ausdrücke.

Der Umgang mit Aggregats- und analytischen Funktionen wird anhand folgender Tabelle demonstriert.

JAHR	MONAT	UMSATZ_EUR	WERBUNG_TV_EUR	WERBUNG_ZEITG_EUR	ANZAHL_VERKAEUFER
2004	11	130000	14000	400	10
2004	12	140000	13200	700	10
2005	1	100000	9000	1000	10
2005	2	103000	9500	900	10
2005	3	112000	10000	1500	10
2005	4	90000	12000	1000	8
2005	5	98000	13000	1000	8
2005	6	70000	4000	5000	9
2005	7	109010	12000	2000	15
2005	8	120000	10000	500	15
2005	9	130000	9000	1000	15
2005	10	150000	15000	1000	15
2005	11	170000	18000	1000	15
2005	12	200000	18000	1000	15
2006	1	160000	10000	1000	10
2006	2	140000	10200	1500	10

Aggregatsfunktionen: GROUP BY und mehr ...

Wir beginnen mit einem sehr einfachen Beispiel. Für jedes Jahr soll die Summe, der Durchschnitt, der minimale und der maximale Umsatz (Spalte UMSATZ_EUR) ermittelt werden. Das erledigt eine ganz normale SQL-Abfrage mit Aggregatsfunktion und GROUP BY.

```
select jahr, sum(umsatz_eur), avg(umsatz_eur), min(umsatz_eur), max(umsatz_eur)
from verkauf group by jahr
order by 1
```

JAHR	SUM(UMSATZ_EUR)	AVG(UMSATZ_EUR)	MIN(UMSATZ_EUR)	MAX(UMSATZ_EUR)
2004	270000,0	135000,0	130000,0	140000,0
2005	1452010,0	121000,8	70000,0	200000,0
2006	300000,0	150000,0	140000,0	160000,0

So weit, so gut. Allerdings erlaubt GROUP BY wesentlich mehr als nur das: Mit der Erweiterung ROLLUP wird der Ergebnismenge nun ein Aggregat "über alles" hinzugefügt.

```
select jahr, sum(umsatz_eur), avg(umsatz_eur), min(umsatz_eur), max(umsatz_eur)
from verkauf group by rollup (jahr)
order by 1
```

JAHR	SUM(UMSATZ_EUR)	AVG(UMSATZ_EUR)	MIN(UMSATZ_EUR)	MAX(UMSATZ_EUR)
2004	270000,0	135000,0	130000,0	140000,0
2005	1452010,0	121000,8	70000,0	200000,0
2006	300000,0	150000,0	140000,0	160000,0
	2022010,0	**126375,6**	**70000,0**	**200000,0**

Ohne ROLLUP bräuchte es hierfür eine zusätzliche SQL-Abfrage, deren Ergebnisse mit UNION ALL angefügt werden. Im Vergleich dazu ist diese Variante wesentlich effizienter; gerade auf großen Datenmengen. Nimmt man auch die Spalte MONAT in die ROLLUP-Klausel, so bekommt man die Zeilen selbst, Aggregate für das einzelne Jahr und "über alles".

```
select
  jahr, monat, sum(umsatz_eur), avg(umsatz_eur), min(umsatz_eur), max(umsatz_eur)
from verkauf group by rollup (jahr,monat)
order by jahr, monat
```

JAHR	MONAT	SUM(UMSATZ_EUR)	AVG(UMSATZ_EUR)	MIN(UMSATZ_EUR)	MAX(UMSATZ_EUR)
2004	11	130000,0	130000,0	130000,0	130000,0
2004	12	140000,0	140000,0	140000,0	140000,0
2004		270000,0	135000,0	130000,0	140000,0
2005	1	100000,0	100000,0	100000,0	100000,0
:	:	:	:	:	:
2005	12	200000,0	200000,0	200000,0	200000,0
2005		1452010,0	121000,8	70000,0	200000,0
2006	1	160000,0	160000,0	160000,0	160000,0
2006	2	140000,0	140000,0	140000,0	140000,0
2006		300000,0	150000,0	140000,0	160000,0
		2022010,0	126375,6	70000,0	200000,0

Es ist auch möglich, nur die Zwischenaggregate pro Jahr zu berechnen und das Aggregat über alles zu unterdrücken. Dazu braucht es aber die Klausel GROUPING SETS: Mit dieser können Sie gewünschten Gruppen explizit anfordern. GROUP BY ROLLUP (JAHR, MONAT) bedeutet das gleiche wie GROUP BY GROUPING SETS (JAHR, MONAT),

(JAHR), (). Wir brauchen ein GROUPING SET pro Monat (JAHR, MONAT) und eins pro Jahr (JAHR). Die Gruppe "über alles" () brauchen wir nicht.

```
select
  jahr, monat, sum(umsatz_eur), avg(umsatz_eur),
  min(umsatz_eur), max(umsatz_eur)
from verkauf group by grouping sets (jahr,monat), (jahr)
order by jahr, monat

 JAHR MONAT SUM(UMSATZ_EUR) AVG(UMSATZ_EUR) MIN(UMSATZ_EUR) MAX(UMSATZ_EUR)
----- ----- --------------- --------------- --------------- ---------------
 2004    11        130000,0        130000,0        130000,0        130000,0
 2004    12        140000,0        140000,0        140000,0        140000,0
 2004              270000,0        135000,0        130000,0        140000,0

 2005     1        100000,0        100000,0        100000,0        100000,0
    :     :               :               :               :               :
 2005    12        200000,0        200000,0        200000,0        200000,0
 2005             1452010,0        121000,8         70000,0        200000,0

 2006     1        160000,0        160000,0        160000,0        160000,0
 2006     2        140000,0        140000,0        140000,0        140000,0
 2006              300000,0        150000,0        140000,0        160000,0
```

Liegen noch mehr "Dimensionsspalten" vor, so lassen sich mit ROLLUP und den anderen GROUP BY-Klauseln CUBE und GROUPING SETS Aggregate für alle möglichen Spaltenkombinationen berechnen.

Weitere Aggregatsfunktionen

Die verwendeten Aggregatsfunktionen SUM, AVG, MIN und MAX sind sicherlich die am häufigsten verwendeten. Aber die Oracle-Datenbank bietet noch mehr an: Ab Oracle11g Release 2 können Zeichenketten mit der LISTAGG-Funktion zusammengefasst werden; das ist ein für den Entwickler sehr nützliches und häufig benötigtes Feature.

```
select deptno, listagg(ename,',') within group (order by ename) emplist
from emp
group by deptno
/

DEPTNO EMPLIST
------ --------------------------------------------------
    10 ADAMS,BLAKE,FORD,KING,MILLER
    20 JONES,SCOTT,SMITH
    30 ALLEN,JAMES,MARTIN,TURNER,WARD
    40 CLARK
```

Die Funktionen zur einfachen linearen Regression zeigen, wie mächtig der Standardumfang der Datenbank ist. Sie ermitteln den absoluten Betrag und die Steigung einer linearen Funktion, die anhand der Daten mit der "Methode der kleinsten Quadrate" gebildet wurde.

```
select
  regr_intercept(umsatz_eur, werbung_tv_eur) as abs_betrag,
  regr_slope(umsatz_eur, werbung_tv_eur) as steigung
from verkauf
/

    ABS_BETRAG       STEIGUNG
-------------- --------------
       46341,7            6,9
```

Die Regressionsfunktion für den Umsatz anhand der Ausgaben für TV-Werbung lautet also:

```
UMSATZ_EUR := 46341.7 + 6.9 * WERBUNG_TV_EUR
```

Und natürlich kann dies - direkt in der gleichen SQL-Abfrage - auch angewendet werden. So macht das folgende Beispiel (anhand des Regressionsmodells) eine Prognose für den Umsatz bei Ausgaben von 200.000 für TV-Werbung.

```
with regr as (
 select
  regr_intercept(umsatz_eur, werbung_tv_eur) as abs_betrag,
  regr_slope(umsatz_eur, werbung_tv_eur) as steigung
 from verkauf
)
select abs_betrag + (steigung * 200000)
from regr
/

ABS_BETRAG+(STEIGUNG*200000)
----------------------------
                   1416638,3
```

Eigene Aggregatsfunktionen erstellen

Wie in der Dokumentation erkennbar ist, stellt die Datenbank eine ganze Menge Aggregatsfunktionenen bereit - aber es fehlt auch was: So gibt es keine Aggregatsfunktion für das "Produkt". Man könnte sich nun zwar eine einfache PL/SQL-Funktion schreiben, die wäre aber nicht in jeder SQL-Abfrage im Zusammenspiel mit GROUP BY nutzbar. Doch auch hier bietet die Oracle-Datenbank Abhilfe: Hält man sich an ein bestimmtes Programmierschema, so kann man sich eigene Aggregatsfunktionen bauen - und diese Aggregatsfunktionen können genauso genutzt werden, wie die eingebauten Pendants.

Das Programmierschema ist im kaum bekannten, aber dafür hochinteressanten Handbuch Data Cartridge Developers' Guide beschrieben.

- Aggregatsfunktion für das Produkt (AGG_PRODUCT):
  http://sql-plsql-de.blogspot.co.uk/2011/02/sql-aggregatsfunktion-product-fehlt.html
- LISTAGG-Funktion für 9*i*, 10*g* und 11*g*R1-Datenbanken:
  http://sql-plsql-de.blogspot.co.uk/2007/03/group-by-wird-zusammen-mit.html

- Aggregatsfunktion für die "Entropie": http://ora-sql-plsql.blogspot.co.uk/2012/02/aggregatsfunktion-zur-berechnung-der.html

- Ein Anwendungsbeispiel: Zinsberechnung für ein Sparkonto: http://sql-plsql-de.blogspot.co.uk/2011/03/noch-ein-szenario-mit-user-defined.html

Hat man, zum Beispiel, die Aggregatsfunktion AGG_PRODUCT eingespielt, so lässt sich diese ganz normal mit GROUP BY nutzen. Auch die beschriebenen GROUP BY-Klauseln ROLLUP, GROUPING SETS und CUBE können verwendet werden.

```
select jahr, agg_product(zinssatz)
from zinsen_jahr
group by rollup (jahr)

   JAHR AGG_PRODUCT(ZINSSATZ)
------- --------------------
   2008      0,03000000000000
   2009      0,02050000000000
   2010      0,02400000000000
   2011      0,02750000000000
   2012      0,02450000000000
             0,00000000994455
```

Basiernd darauf ließen sich bspw. viele Kennzahlen aus der Finanzmathematik recht einfach in benutzerdefinierte Aggregatsfunktionen überführen.

Dynamische Aggregatsfenster: Analytische Funktionen

Das mit der GROUP-BY-Klausel definierte "Fenster", über welches die Aggregate gebildet werden, ist eher statischer Natür: Zur Abfragezeit werden die Daten in feste Abschnitte, eben die Gruppen, aufgeteilt. Manche Aufgaben, wie das Berechnen eines gleitenden Durchschnitts, erfordern jedoch mehr Möglichkeiten: Hier helfen die analytischen Varianten weiter - jede Aggregatsfunktion steht in einer analytischen Variante bereit - und darüber hinaus gibt es noch zusätzliche analytische Funktionen.

Ein wichtiges Merkmal der analytischen Funktionen ist, dass die Aggregate für jede Zeile der Ergebnismenge berechnet werden - im Gegensatz zu den "klassischen" Aggregatsfunktionen, welche das Aggregat nur einmal pro Gruppe ausrechnen und für jede Gruppe nur noch eine Zeile ausgeben. Allen analytischen Funktionen ist die Klausel OVER gemein - in dieser wird das Aggregats-"Fenster" definiert (analytische Funktionen werden manchmal auch als Window-Funktionen) bezeichnet. Die OVER-Klausel besteht wiederum aus drei Teilen:

- Mit **PARTITION BY** werden die Daten in Gruppen unterteilt. Dies ähnelt sehr stark dem klassischen **GROUP BY**
- Mit **ROWS BETWEEN** wird das Aggregatsfenster über eine bestimmte Menge an Zeilen, ausgehend vom "über die Ergebnismenge laufenden Cursor" gebildet.
- Mit **ORDER BY** wird die Sortierung innerhalb des Aggregatfensters festgelegt.

Diese doch eher theoretischen Beschreibungen werden nun anhand eines Beispiels deutlich gemacht. Über die Umsatzzahlen (**UMSATZ_EUR**) der Tabelle **VERKAUF** soll ein gleitender Durchschnitt gebildet werden. Die Periode soll dabei drei Monate sein - achten Sie auf das **ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING**. Diese Klausel legt fest, dass das Aggregatsfenster für jede Zeile immer von der Zeile davor bis zur Zeile danach gehen soll. Eine solche Klausel erfordert natürlich eine Sortierung - das wird mit dem **ORDER BY** innerhalb der OVER-Klausel erledigt.

```
select
 JAHR,
 MONAT,
 UMSATZ_EUR,
 AVG(UMSATZ_EUR) over (order by JAHR, MONAT rows between 1 preceding and 1
following) as GL_DS_UMSATZ
from VERKAUF
order by jahr, monat

      JAHR      MONAT UMSATZ_EUR GL_DS_UMSATZ
---------- ---------- ---------- ------------
      2009         11     130000   135000,000
      2009         12     140000   123333,333
      2010          1     100000   114333,333
         :          :          :            :
      2010         12     200000   185000,000
      2011          1              200000,000
      2011          2
```

Analog dazu kann die "laufende Summe" gebildet werden - für jedes Jahr sollen die Umsatzzahlen fortlaufend aufsummiert werden; allerdings soll die Summierung im neuen Jahr neu beginnen. Dazu benötigen wir alle drei Elemente der OVER-Klausel wie folgt.

```
select
 JAHR,
 MONAT,
 UMSATZ_EUR,
 SUM(UMSATZ_EUR) over (partition by jahr order by MONAT rows between
unbounded preceding and current row) as SUM_UMSATZ
from VERKAUF
order by jahr, monat
/

      JAHR      MONAT UMSATZ_EUR SUM_UMSATZ
---------- ---------- ---------- ----------
      2009         11     130000     130000
      2009         12     140000     270000

      2010          1     100000     100000
      2010          2     103000     203000
         :          :          :          :
      2010         12     200000    1452010

      2011          1      50000      50000
      2011          2     150000     200000
```

Zunächst werden die Daten mit der **PARTITION BY**-Klausel nach Jahren gruppiert. Jede Gruppe wird intern nach dem Monat sortiert und das "Aggregatsfenster" läuft vom Anfang (**UNBOUNDED PRECEDING**) bis zur aktuellen Zeile. Für häufige Aufgabenstellungen gibt es eigene analytische Funktionen. So berechnet **RATIO_TO_REPORT** für jede Zeile den prozentualen Anteil an der Gesamtmenge. Angewendet auf das Umsatzbeispiel sähe die Abfrage wie folgt aus:

```
select
 JAHR,
 MONAT,
 UMSATZ_EUR,
 RATIO_TO_REPORT(UMSATZ_EUR) over (partition by jahr) * 100 as
ANTEIL_UMSATZ
from VERKAUF
order by jahr, monat
/

      JAHR      MONAT UMSATZ_EUR ANTEIL_UMSATZ
---------- ---------- ---------- -------------
      2009         11     130000         48,15
      2009         12     140000         51,85

      2010          1     100000         06,89
         :          :          :             :
      2010         12     200000         13,77

      2011          1      50000         25,00
      2011          2     150000         75,00
```

Analytische Funktionen lassen sich auch mit den klassichen Pendants kombinieren. So wäre es nett, wenn diesem Bericht noch eine Zeile mit den Summen für das jeweilige Jahr hinzugefügt würde. Das geht mit der am Anfang besprochenen Funktion **ROLLUP**.

```
with zahlen as (
 select
  JAHR,
  MONAT,
  UMSATZ_EUR,
  RATIO_TO_REPORT(UMSATZ_EUR) over (partition by jahr) * 100 as
ANTEIL_UMSATZ
 from VERKAUF
)
select
 jahr,
 monat,
 sum(umsatz_eur) umsatz_eur,
 sum(anteil_umsatz) anteil_umsatz
from zahlen
group by grouping sets(jahr, monat), (jahr)
order by jahr, monat
/
```

```
      JAHR      MONAT UMSATZ_EUR ANTEIL_UMSATZ
---------- ---------- ---------- -------------
      2009         11     130000         48,15
      2009         12     140000         51,85
      2009                270000        100,00

      2010          1     100000         06,89
      2010          2     103000         07,09
         :          :          :             :
      2010         12     200000         13,77
      2010                1452010        100,00

      2011          1      50000         25,00
      2011          2     150000         75,00
      2011                200000        100,00
```

Kreuztabellen ganz einfach: mit PIVOT und UNPIVOT

Das Generieren einer *Pivot-* oder *Kreuztabelle* kommt in der Praxis nahezu alltäglich vor. *Pivoting* überführt mehrere Zeilen in zusätzliche Spalten (dabei können Aggregationen durchgeführt werden), *Unpivoting* wandelt dagegen Spalten in Zeilen um: Die Tabelle wird in die andere Richtung „gedreht".

Pivoting und Unpivoting kann natürlich auch in älteren Datenbankversionen durchgeführt werden; dazu sind allerdings teilweise recht komplexe SQL-Anweisungen nötig. Oracle11*g* stellt dazu die sehr einfach nutzbaren SQL-Klauseln PIVOT und UNPIVOT bereit. Listing 3 illustriert die Nutzung der PIVOT-Klausel: Dabei werden die Inhalte der Spalte JOB als zusätzliche Spalten erzeugt; diese enthalten dann die aufsummierten Gehälter (SAL) nach Abteilung (DEPTNO).

```
select deptno, sum(clerk), sum(salesman), sum(manager)
from emp pivot (
  sum(sal) for JOB in ('CLERK' as "CLERK", 'SALESMAN' as "SALESMAN", 'MANAGER' as "MANAGER")
)
group by deptno

DEPTNO     SUM(CLERK) SUM(SALESMAN) SUM(MANAGER)
---------- ---------- ------------- ------------
        30        950          5600         2850
        20       1900                       2975
        10       1300                       2450
```

SQL MODEL Klausel

Bereits mit Oracle10*g* wurde die *SQL Model Clause* eingeführt – sie macht es möglich, im Ergebnis einer SQL-Abfrage so mit Formeln zu rechnen, wie man es von einem Tabellenkalkulationsprogramm gewohnt ist.

Die SQL Model Clause betrachtet die Ergebnismenge einer Abfrage wie ein Arbeitsblatt einer Tabellenkalkulation. Nach dem Schlüsselwort **model** werden die Dimensionen (**dimensions**) und die Werte (**measures**) deklariert. Die Dimensionen dienen zum "Ansprechen" der Werte, die selbst durch die nachfolgenden Rules verändert werden können. Listing 5 zeigt zunächste die Struktur einer SQL-Abfrage mit der MODEL-Klausel.

```
select zeile, a, b, c, d, e, f from emp
model
 dimension by (rownum zeile)
 measures (
  empno a, ename b, hiredate c, sal d,
  comm e, cast(null as number) f
 )
 rules upsert (
   -- Hier werden "Formeln" als "Rules" eingebeben!
   b[2] = 'SCHMIDT',
   f[1] = (d[1] / d[2] - 1) * 100,
   d[15] = avg(d)[ANY]
 )
order by zeile
```

Listing 5: SQL MODEL Klausel in Aktion

Anstelle des Kommentars in den Klammern von RULES UPSERT können nun "Formeln" ähnlich zur Nutzung in einer Tabellenkalkulation angegeben werden. Und natürlich bezieht sich das "RULES UPSERT" allein auf die Ergebnismenge der SQL-Abfrage - in der Tabelle werden keine Daten geändert. Einige Beispiele

In der zweiten Zeile der Ergebnismenge soll der Name (Spalte "B") nach SCHMIDT geändert werden:
b[2] = 'SCHMIDT'

Die Spalte "F" ist zunächst leer (SQL NULL). Dort soll nun für die erste Zeile berechnet werden, um wie viel Prozent das Gehalt (Spalte "D") über dem Gehalt der zweiten Zeile liegt.
f[1] = (d[b="KING"] / d[2] - 1) * 100

Schließlich soll eine neue Zeile angefügt werden, die in der Spalte "D" den Durchschnitt über alle Gehälter enthält.
d[15] = avg(d)[ANY]

Das Ergebnis dieser SQL-Abfrage sieht dann wie folgt aus:

```
ZEILE     A B          C                     D     E     F
----- ----- ---------- ------------------- ----- ----- -----
    1  7369 SMITH      17.12.1980 00:00:00   800         -50
    2  7499 SCHMIDT    20.02.1981 00:00:00  1600   300
    3  7521 WARD       22.02.1981 00:00:00  1250   500
    4  7566 JONES      02.04.1981 00:00:00  2975
    5  7654 MARTIN     28.09.1981 00:00:00  1250  1400
:
   13  7902 FORD       03.12.1981 00:00:00  3000
   14  7934 MILLER     23.01.1982 00:00:00  1300
   15                                       2073

15 Zeilen ausgewählt.
```

Listing 6: Ergebnismenge der SQL-Abfrage mit MODEL-Klausel

Weitere Informationen

Viele der hier vorgestellten Beispiele sind der deutschen APEX Community und dem Blog "SQL und PL/SQL in Oracle" entnommen. Darüber hinaus gibt es mittlerweile einige deutschsprachige Informationsquellen rund um die Oracle-Datenbank.

[1] Deutschsprachige APEX Community
http://tinyurl.com/apexcommunity

[2] Oracle Dokumentation: SQL Language Reference
http://docs.oracle.com/cd/E11882_01/server.112/e26088/toc.htm

[3] Oracle Dokumentation: Data Warehousing Guide
http://docs.oracle.com/cd/E11882_01/server.112/e25554/toc.htm

[4] Oracle Dokumentation: Data Cartridge Developers' Guide
http://docs.oracle.com/cd/E11882_01/appdev.112/e10765/toc.htm

[5] Blog "SQL und PL/SQL in Oracle"
http://sql-plsql-de.blogspot.com

Kontaktadresse:

Carsten Czarski
ORACLE Deutschland B.V. & Co KG
Riesstr. 25, 80992 München

Telefon:	+49 (0) 89 1430 2116
E-Mail	carsten.czarski@oracle.com
Internet:	www.oracle.de
Blog des Autors	http://sql-plsql-de.blogspot.com
Twitter	@cczarski