Oracle SQL – Aggregierte Abfragen 2

Stephan Karrer

Erweiterung der GROUP BY Klausel: ROLLUP - Operator

```
SELECT department_id,
       job_id,
       SUM(salary)
FROM employees
GROUP BY ROLLUP
       (department_id,
       job_id)
ORDER BY department_id;
```

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
10		8750
20	ANALYST	6000
20	CLERK	1900
20	MANAGER	2975
20		10875
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600
30		9400
		29025

Erweiterung der GROUP BY Klausel: CUBE - Operator

```
SELECT department_id,
       job_id,
       SUM(salary)
FROM employees
GROUP BY CUBE
       (department_id,
       job_id)
ORDER BY department_id;
```

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
10		8750
20	ANALYST	6000
20	CLERK	1900
20	MANAGER	2975
20		10875
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600
30		9400
	ANALYST	6000
	CLERK	4150
	MANAGER	8275
	PRESIDENT	5000
	SALESMAN	5600
		29025

Verwendung der Funktion GROUPING

```
SELECT   department_id DEPTID, job_id JOB,
         SUM(salary),
         GROUPING(department_id) GRP_DEPT,
         GROUPING(job_id) GRP_JOB
FROM     employees
WHERE    department_id < 50
GROUP BY ROLLUP(department_id, job_id);
```

- Die Funktion GROUPING liefert 1, sofern der angezeigte NULL-Wert durch die Aggregation zustande kam, ansonsten 0
- Der Parameter der Funktion muss ein Gruppierungskriterium sein

Verwendung der Funktion GROUPING_ID

```
SELECT   department_id DEPTID, job_id JOB,
         SUM(salary),
         GROUPING(department_id) GRP_DEPT,
         GROUPING(job_id) GRP_JOB,
         GROUPING_ID(department_id, job_id)
FROM     employees
WHERE    department_id < 50
GROUP BY CUBE(department_id, job_id);
```

- GROUPING_ID erzeugt einen Bitvektor, in dem jede Stelle per 1 angibt, ob auf dem Kriterium aggregiert wurde, und liefert die entsprechende Dezimalzahl zurück.
- Dies kann bei vielen Gruppierungskriterien effizienter sein, als die Verwendung einzelner GROUPING-Funktionen

Verwendung von Grouping Sets

```
SELECT   department_id, job_id,
         manager_id,avg(salary)
FROM     employees
GROUP BY GROUPING SETS
((department_id,job_id), (job_id,manager_id));
```

- Mittels GROUPING SETS werden genau die gewünschten Gruppierungen definiert

- Effizienz:
  Die Basismenge muss nur einmal durchsucht werden, statt viele Ergebnisse zu kombinieren

Ebenen überspringen: zusammengesetzte Spalten

```
SELECT   department_id, job_id, manager_id,
         SUM(salary)
FROM     employees
GROUP BY ROLLUP( department_id,(job_id, manager_id));
```

- Gruppen von Spalten werden als Einheit definiert
- Dadurch werden bei der Aggregation Detailebenen übersprungen

Konkatenation von Gruppierungen

```
SELECT   department_id, job_id, manager_id, SUM(salary),
         grouping(department_id), grouping(job_id),
grouping(manager_id)
FROM     employees
GROUP BY department_id,
         ROLLUP(job_id),
         CUBE(manager_id)
ORDER BY department_id, job_id, manager_id;
```

- Die einzelnen Auswertungsarten lassen sich auch kombinieren
- Somit gibt es meist mehrere Möglichkeiten um die gewünschte Ergebnismenge zusammen zu stellen