

SQL Server – Aggregierte Abfragen

Stephan Karrer

Gruppenfunktionen (Aggregatfunktionen)

Funktion	Kommentar
AVG ([ALL DISTINCT] expression)	Mittelwert, NULL-Werte werden ignoriert
CHECKSUM_AGG ([ALL DISTINCT] expression)	Prüfsumme, NULL-Werte werden ignoriert
COUNT ({ [[ALL DISTINCT] expression] * }) COUNT_BIG ({ [ALL DISTINCT] expression } *)	Anzahl von Einträgen
GROUPING (column_name)	Verwendung bei CUBE- oder ROLLUP-Operator
MAX ([ALL DISTINCT] expression)	Maximum
MIN ([ALL DISTINCT] expression)	Minimum
SUM ([ALL DISTINCT] expression)	Summe, NULL-Werte werden ignoriert
STDEV ([ALL DISTINCT] expression) STDEVP ([ALL DISTINCT] expression)	Standardabweichung
VAR ([ALL DISTINCT] expression) VARP ([ALL DISTINCT] expression)	Varianz

Einfache Verwendung von Gruppenfunktionen

```
SELECT COUNT(*) "Anzahl Zeilen" FROM employees;

/* Max. Schachtelungstiefe ist 2 */
SELECT AVG( ISNULL(salary, 0)) FROM employees;

SELECT MAX(salary) FROM employees
       WHERE job_id = 'IT_PROG' ;
```

Verwendung von Gruppenfunktionen

```
SELECT [ALL|DISTINCT] Auswahlliste
      FROM Quelle
      [WHERE Where-Klausel]
      [GROUP BY (Group-by-Attribut)
               [HAVING Having-Klausel]]
      [ORDER BY (Sortierungsattribut) [ASC|DESC]]
```

- WHERE:
schränkt die Ausgangsmenge ein.
- GROUP BY:
zerlegt die Ausgangsmenge in Gruppen. Je Gruppe wird aggregiert.
- HAVING:
nachträgliche Filterung der Ergebnisse.
- SELECT:
in der Auswahlliste können nur die Gruppierungsattribute und Aggregate verwendet werden !!

Verwendung von Gruppenfunktionen: GROUP BY und HAVING

```
SELECT department_id, COUNT(DISTINCT job_id)
      FROM employees
      GROUP BY department_id;
```

```
SELECT job_id, SUM(salary)
      FROM employees
      WHERE department_id <> 100
      GROUP BY job_id
      ORDER BY job_id DESC;
```

```
SELECT job_id, SUM(salary)
      FROM employees
      WHERE department_id <> 100
      GROUP BY job_id
      HAVING SUM(salary) > 20000
      ORDER BY job_id DESC;
```

Nach mehreren Spalten gruppieren

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

- Es wird anhand der tiefsten Gruppenbildung aggregiert.

Gruppierung nach berechneten Werten

```
SELECT length(last_name) as "Length", count(*) as "Count"  
  FROM employees  
 GROUP BY LENGTH(last_name)  
 ORDER BY "Length";
```

- Generell kann nach Ausdrücken und damit nach berechneten Werten aggregiert werden.

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