Aggregatfunktionen in SQL

Michael Dienert

14. April 2008

1 Definition von Aggregatfunktionen

Ihren Namen haben die Aggregatfunktionen vom englischen Verb *to aggregate*, was auf deutsch *anhäufen, vereinigen, zusammenballen*, heisst. Hier nun die Definition der Aggregatfunktionen:

> Aggregatfunktionen bilden **genau einen** Ergebniswert aus einer **Vielzahl** von Eingabewerten.
> Die fünf wichtigsten SQL-Aggregatfunktionen sind:
> **COUNT, SUM, AVG, MAX, MIN**

1.1 Ein Beispiel

Tabelle Wetter

stadt	temp_min	temp_max
FR	-1	19
S	-5	12
MA	-2	13
KA	-2	14

Die Abfrage

```
SELECT MAX(temp_min) FROM wetter;
```

liefert als Ergebnis -1, Freiburg ist also die Stadt, mit der höchsten Minimaltemperatur. Möchte man die ganze zugehörige Zeile ausgeben, lässt sich das mit einer *Unterabfrage* (Sub-Query) erreichen. Hier ein Beispiel, in dem schrittweise eine Unterabfrage zusammengestellt wird:

- Die Abfrage von oben

  ```
  SELECT MAX(temp_min) FROM wetter;
  ```

  liefert als Ergebnis z.b. den Wert **3**.

- Da man nun diesen Wert kennt, kann man gezielt die zugehörige Stadt abfragen:

  ```
  SELECT * FROM wetter WHERE temp_min = 3;
  ```

- Diese beiden Abfragen kann man jetzt zusammenfassen:

```
SELECT * FROM wetter WHERE temp_min =
   (SELECT MAX(temp_min) FROM wetter);
```

1.2 Aufgaben

Unter

`http://msv.wara.de/~dienert/archive/wetter.sql`

ist ein SQL-Skript abgelegt, mit dem die Datenbank `wetter` erzeugt wird. Kopiere diese Datei mit dem Kommando `wget` in Dein Heimatverzeichnis, lege Dich selbst als postgres-Benutzer an (`su postgres ..., createuser` *meinAnmeldename*, `exit`) und spiele die Datei mit

```
psql -f wetter.sql postgres
```

ein.

1. Warum meldet der postgres-Server beim ersten Einspielen der datei `wetter.sql` einen Fehler? Die Antwort auf diese Frage steht in der Datei `wetter.sql` selbst.
2. Verbinde Dich mit der Datenbank `wetter` mit dem Kommando `psql wetter` und teste alle Abfragen aus Abschnitt 1.
3. Warum funktioniert folgende Abfrage nicht:

   ```
   SELECT stadt, max(temp_min) FROM wetter;
   ```

   Hinweis: Wieviele Ergebniszeilen haben die Abfragen

   ```
   SELECT MAX(temp_min) FROM wetter;
   ```

   und

   ```
   SELECT stadt FROM wetter;
   ```

   jeweils?

2 GROUP BY und HAVING - Statements

2.1 Eine erweiterte Temperaturtabelle

Messen wir die Temperaturen in den Städten an mehreren Tagen, kommen wir zu einer neuem Messwertetabelle:

Tabelle Wetter

Stadt	Temp_min	Temp_max
freiburg	-1	19
stuttgart	-5	12
mannheim	-2	13
karlsruhe	-2	14
freiburg	-0.5	17
stuttgart	-3	13
mannheim	-2	10
karlsruhe	-2	12
freiburg	0.5	21
stuttgart	-0.5	14
mannheim	-1	13.5
karlsruhe	-1.5	13.5
freiburg	3	14
stuttgart	1.5	16
mannheim	0.5	13
karlsruhe	2	14

Das Berechnen absoluter Minimal- oder Maximalwerte kann genau gleich wie oben erfolgen. Wie sieht es aber aus, wenn ich z.b. die durchschnittliche Minimaltemperatur der einzelnen Städte haben möchte? Für diese Aufgabe müsste ich die Messwerte, die zu jeweils einer Stadt gehören zusammenfassen, also **gruppieren** und dann den jeweiligen Durchschnitt dieser Gruppen berechnen.

3 Zusammenfassen gleicher Werte

> Das SQL-Statement GROUP BY wird benutzt, um die Zeilen einer Tabelle zusammenzufassen, die die gleichen Spaltenwerte haben.

Die SQL-Abfrage

```
SELECT stadt from wetter;
```

liefert als Ergebnis eine liste mit 16 Einträgen: jede Stadt kommt 4 mal vor.
Nun kann man alle Ergebniszeilen zusammenfassen, bei denen der Städtenamen gleich ist:

```
SELECT stadt FROM wetter GROUP BY stadt;
```

Und schliesslich den Durchschnitt berechnen:

```
SELECT stadt, AVG(temp_min) FROM wetter GROUP BY stadt;
```

> Eine Aggregations-Funktion wird über alle Zeilen berechnet, die eine Gruppe bilden. Für jede Gruppe gibt es ein Ergebnis der Aggregations-Funktion.

Nun fragt man sich vielleicht, wie dann das Beispiel ganz oben (`SELECT MAX(temp_min) FROM wetter;`) funktioniert. Es enthält ja eine Aggregationsfunktion aber keinen GROUP BY - Ausdruck:

> Wird eine Aggregations-Funktion ohne GROUP BY verwendet, werden alle selektierten Zeilen als eine Gruppe betrachtet.

3.1 Aufgaben

Teste alle Abfragen aus Abschnitt 3

4 Zugriff auf nicht gruppierte Attribute

Die Verwendung von GROUP BY schränkt den Zugriff auf die anderen Attriute der Tabelle allerdings ein:

> Alle Spalten, die nicht gruppiert werden, können nur noch in Aggregat-Funktionen verwendet werden!

Folgende Abfrage ist also nicht möglich:

```
SELECT stadt, temp_min FROM wetter GROUP BY stadt;
```

Das ergibt unter postgres folgende Fehlermeldung:

```
ERROR: column "wetter.temp_min" must appear in the GROUP BY clause
  or be used in an aggregate function)
```

Warum ist das so? Ganz einfach: durch die Gruppierung der Städte soll das RDBMS 4 Werte für die Städte aber 16 Werte für die Temperaturen ausgeben, also eine Tabelle erzeugen, die gleichzeitig 4 und 16 Zeilen hat, was natürlich nicht klappen kann.

4.1 Aufgaben

Teste die Abfragen aus Abschnitt 4.

Warum funktioniert folgende Abfrage und warum hat sie 14 Ergebniszeilen:

```
SELECT stadt, temp_min FROM wetter GROUP BY stadt, temp_min;
```

5 Filterung der Gruppen

Nun möchten wir dieses Ergebnis einschränken: es sollen nur die Städte aufgelistet werden, bei denen die Durchschnittstemperatur über -1°C liegt. Leider können wir diese Filterfunktion *nicht* mit WHERE erzeugen:

> WHERE selektiert die Zeilen **bevor** die Aggregatfunktion berechnet und mit GROUP BY Gruppen gebildet werden.
> Das heisst mit anderen Worten: WHERE bestimmt, welche Zeilen in die Berechnung der Aggregationsfunktion einfliessen.

Wir wollen aber nach der Berechnung der Mittelwerte filtern. Hierfür muss man das HAVING-Schlüsselwort verwenden:

Mit dem Schlüsselwort HAVING werden ganze Gruppen gefiltert, **nachdem** sie mit GROUP BY gebildet wurden.
In der praktischen Anwendung enthält die HAVING-Anweisung immer eine Aggregationsfunktion.

Hier nun das Beispiel:

```
SELECT AVG(temp_min), stadt
  FROM wetter
  GROUP BY stadt HAVING AVG(temp_min) > -1;
```

5.1 Aufgabe

Teste auch dieses Beispiel