PostgreSQL – Analytische Funktionen

Stephan Karrer

Partitionierung der Ergebnismenge: Fenster-Funktionen

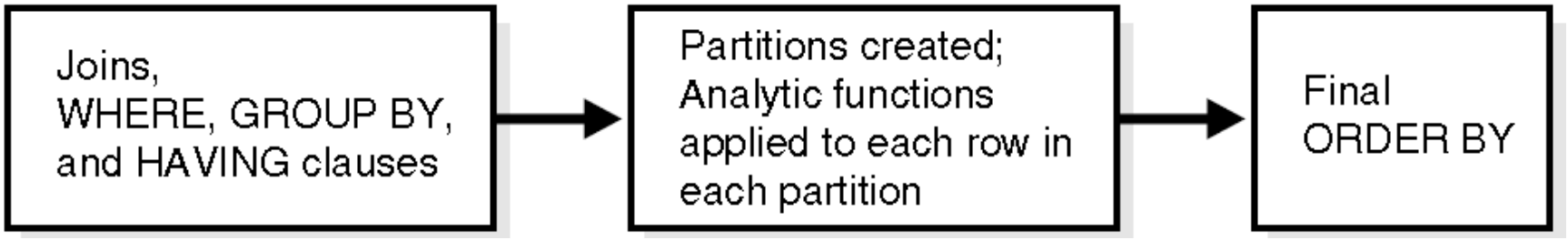
```
SELECT last_name, salary,
       sum(salary) OVER () Gesamt_Sum
    FROM employees;



SELECT last_name, salary, department_id,
       max(salary) OVER (PARTITION BY department_id) Abt_Max
    FROM employees
    ORDER BY department_id;
```

- Das erlaubt uns die Kombination skalarer Werte mit Aggregaten, was bei dem herkömmlichen Gruppierungsvorgehen nicht direkt möglich ist

Einschränkungen



- Da die Partitionierung der Ergebnismenge erst am Ende erfolgt ist die Verwendung der sogenannten „Analytischen Klausel“ nur in der SELECT-Liste erlaubt bzw. kann danach sortiert werden.
- Sollen die Ergebnisse weiterverarbeitet werden, so muss das durch Einbettung in Unterabfrage erfolgen.
- Der Begriff „Partitionierung“ hat hier nichts mit partitionierten Tabellen zu tun (Der ANSI-Standard benutzt den Begriff Fenster bzw. Fensterfunktionen).

Mehrfachgruppierung und -partitionierung

```
SELECT last_name, salary, department_id, job_id, manager_id,
 MAX(salary) OVER (PARTITION BY department_id, job_id) "Dep_Max",
 AVG(salary) OVER (PARTITION BY manager_id) "Man_AVG"
    FROM employees
    ORDER BY department_id, manager_id;
```

- Es ist durchaus Mehrfachgruppierung möglich (wie bei GROUP BY)
- Mehrere, unabhängige Partitionierungen können vorgenommen werden

Partitionierung der Ergebnismenge nach berechneten Werten

```
SELECT last_name, LENGTH(last_name) AS "Length",
       count(*) OVER (PARTITION BY LENGTH(last_name)) as "Count"
  FROM employees
  ORDER BY "Length";
```

- Analog zu GROUP BY kann auch hier die Partitionierung anhand berechneter Werte erfolgen

WINDOW-Klausel

```
SELECT last_name, salary, department_id,
       max(salary) OVER temp_win Abt_Max,
       min(salary) OVER temp_win Abt_Min
    FROM employees
    WINDOW temp_win AS (PARTITION BY department_id)
    ORDER BY department_id;

-- statt
SELECT last_name, salary, department_id,
       max(salary) OVER (PARTITION BY department_id) Abt_Max,
       min(salary) OVER (PARTITION BY department_id) Abt_Min
    FROM employees
    ORDER BY department_id;
```

- Sollen verschiedene Aggregate über dieselbe Partitionierung berechnet werden, kann dem Fenster ein Name gegeben werden.

Filtern der Werte

```
SELECT last_name, salary, department_id,
       max(salary) FILTER (WHERE department_id > 40)
              OVER (PARTITION BY department_id) Abt_Max,
       min(salary) FILTER (WHERE department_id < 50)
             OVER (PARTITION BY department_id) Abt_Min
    FROM employees
    ORDER BY department_id;
```

- Analog zu GROUP BY kann auch hier die Wertemenge für die Aggragation vorab gefiltert werden.
  Hierbei sind unterschiedliche Filter möglich!

- Filterung kann nur bei Aggregatsberechnungs-Funktionen benutzt werden.

Sortieren

```
SELECT last_name, salary, department_id,
       MAX(salary) OVER (PARTITION BY department_id
                         ORDER BY last_name) Abt_Max
    FROM employees
    ORDER BY department_id;
```

- Sortierung kann innerhalb der Partition erfolgen
- Dann verhält sich die analytische Funktion (Aggregatsfunktion) dynamisch !!
- Globale Sortierung sollte das berücksichtigen

Partitionierung der Ergebnismenge mit dynamischen Fenster

```
SELECT last_name,  salary,
       MAX(salary)
       OVER (ORDER BY salary
                ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING) n1
  FROM employees;

SELECT last_name,  department_id, salary,
       MAX(salary)
       OVER (PARTITION BY department_id ORDER BY salary
              ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING) n1
  FROM employees;
```

Möglichkeiten dynamischer Fenster - 1

```
SELECT last_name,  salary,
 SUM(salary) OVER
    (ORDER BY salary ROWS UNBOUNDED PRECEDING) "Unb_Pre",
 SUM(salary) OVER
    (ORDER BY salary ROWS BETWEEN CURRENT ROW
                             AND UNBOUNDED FOLLOWING) "Unb_Fol",
 SUM(salary) OVER
    (ORDER BY salary ROWS BETWEEN 2 PRECEDING
                             AND 2 FOLLOWING) "5_Window",
 SUM(salary) OVER
    (ORDER BY salary RANGE BETWEEN 100 PRECEDING
                             AND 100 FOLLOWING) "Range_Window"
FROM employees;
```

- Fehlt das Schlüsselwort BETWEEN, so ist die Grenze die jeweils aktuelle Zeile.
- Mittels RANGE kann der Wertebereich (abhängig vom jeweiligen Datentyp) addressiert werden.

Details siehe
https://www.postgresql.org/docs/18/sql-expressions.html#SYNTAX-WINDOW-FUNCTIONS

Möglichkeiten dynamischer Fenster - 2

```
SELECT last_name, salary, department_id, job_id,
       sum(salary) OVER (PARTITION BY department_id ORDER BY job_id
                                      GROUPS CURRENT ROW) "Sum_Group"
    FROM employees
    ORDER BY department_id;

SELECT last_name, salary, job_id,
       sum(salary) OVER (ORDER BY job_id
              GROUPS BETWEEN 1 PRECEDING AND 1 FOLLOWING) "Sum_Group"
    FROM employees
    ORDER BY job_id;
```

- Eine Gruppe besteht aus allen Zeilen mit dem gleichen Wert bzgl. der Sortierung.
- Falls BETWEEN angeben wird, so adressiert das die Gruppen davor bzw. dahinter.

  Details siehe
  https://www.postgresql.org/docs/18/sql-expressions.html#SYNTAX-WINDOW-FUNCTIONS
  und
  https://modern-sql.com/caniuse/over_groups_between

Möglichkeiten dynamischer Fenster - 3

```
SELECT last_name,  salary,
 SUM(salary) OVER (ORDER BY salary ROWS BETWEEN 2 PRECEDING
                      AND 2 FOLLOWING EXCLUDE CURRENT ROW) "4_Window"
FROM employees;

SELECT last_name, salary, department_id, job_id,
    sum(salary) OVER (PARTITION BY department_id ORDER BY job_id
        GROUPS CURRENT ROW EXCLUDE TIES) "Sum_Single"
    FROM employees
    ORDER BY department_id;
```

- Mittels EXCLUDE können Zeilen oder Gruppen aus dem Bereich ausgeschlossen werden.

  Details siehe
  https://www.postgresql.org/docs/18/sql-expressions.html#SYNTAX-WINDOW-FUNCTIONS

Rangfolge-Funktionen

```
SELECT last_name, salary, department_id
       , row_number() OVER (PARTITION BY  department_id
            ORDER BY salary DESC NULLS LAST) "RowNumber"
       , rank() OVER (PARTITION BY  department_id
            ORDER BY salary DESC NULLS LAST) "Rank"
       , dense_rank() OVER (PARTITION BY  department_id
            ORDER BY salary DESC NULLS LAST) "DenseRank"
       , percent_rank() OVER (PARTITION BY  department_id
            ORDER BY salary DESC NULLS LAST) AS "%Rank"
       , ntile(5) OVER (PARTITION BY  department_id
            ORDER BY salary DESC NULLS LAST) AS "Bucket"
    FROM employees
    ORDER BY "Rank";
```

- Ranking innerhalb des Fensters

Anwendung spezieller analytischer Funktionen

```
SELECT last_name, salary, department_id,
       first_value(salary) OVER (PARTITION BY  department_id
                                 ORDER BY salary DESC NULLS LAST) "First",
       last_value(salary) OVER (PARTITION BY  department_id
                                 ORDER BY salary DESC NULLS LAST) "Last",
       lead(salary, 1, 0) OVER (PARTITION BY  department_id
                                 ORDER BY salary DESC NULLS LAST) "Lead",
       lag(salary, 1,999) OVER (PARTITION BY  department_id
                                 ORDER BY salary DESC NULLS LAST) "Lag",
       nth_value (last_name, 3) OVER (PARTITION BY  department_id
                                 ORDER BY salary DESC NULLS LAST) "Nth"
    FROM employees
    ORDER BY department_id;
```

- Der Zugriff kann auf alle Spalten der jeweiligen Zeile erfolgen.