Oracle SQL – Unterabfragen

Stephan Karrer

Unterabfragen

- Können wie gewöhnliche Ausdrücke verwendet werden in:
  - WHERE-Klausel
  - HAVING-Klausel
  - FROM-Klausel

```
SELECT last_name,salary
       FROM employees
       WHERE salary > (SELECT salary
                       FROM employees
                       WHERE last_name = 'Abel'  );
SELECT last_name,salary
       FROM employees
       WHERE salary = (SELECT MAX(salary)
                       FROM employees);
```

Regeln bei der Verwendung von Unterabfragen

```
SELECT last_name,salary
       FROM employees
       WHERE salary = (SELECT MAX(salary)
                       FROM employees);
```

- Unterabfragen werden geklammert
- Bei Vergleichsoperatoren ist es besser lesbar, die Unterabfrage auf die rechte Seite zu schreiben
- ORDER BY wird in der Unterabfrage in der Regel nicht benötigt (es sei denn, für Top-N-Analyse)
- **Vorsicht:**
  Unterabfragen können keinen, einen Wert (skalar, single row), aber auch viele Werte (multiple row) bzw. Tupel (multiple column) zurückgeben !

Unterabfrage in HAVING-KLausel

```
SELECT department_id, MIN(salary)
   FROM employees
   GROUP BY department_id
   HAVING MIN(salary) > (SELECT MIN(salary)
                         FROM employees
                         WHERE department_id = 50) ;
```

- Unterabfragen können in der HAVING-Klausel benutzt werden
- Typischer Einsatz: Aggregierten Wert der äußeren Abfrage vergleichen

Unterabfragen die mehrere Zeilen zurückgeben (multiple row)

Operator	Bedeutung
IN	Ein Elememt aus der Ergebnisliste muß gleich sein
ANY	Irgendein Wert aus der Ergebnisliste
ALL	Alle Werte der Ergebnisliste

```
SELECT last_name,salary  FROM employees
       WHERE salary < ANY (SELECT salary  FROM employees
                           WHERE department_id = 50);

SELECT last_name  FROM employees
       WHERE employee_id NOT IN
              (SELECT DISTINCT manager_id FROM employees
                     WHERE manager_id IS NOT NULL);
-- Die Prüfung auf NULL ist wichtig!
```

Unterabfragen die mehrere Spalten liefern

```
SELECT  employee_id, manager_id, department_id
FROM    employees
WHERE   (manager_id, department_id) IN
                        (SELECT manager_id, department_id
                         FROM   employees
                         WHERE  employee_id IN (199,174))
AND employee_id NOT IN (199,174);
```

- Bei Oracle, PostgreSQL können die Ergebnis-Tupel direkt verarbeitet werden. Leider nein bei SQL Server bis Version 2019..
- ANSI nennt das Tabellen-wertige Unterabfrage (sonst üblicherweise multi column)

EXISTS – Bedingung für Unterabfragen

```
SELECT 'es gibt Mitarbeiter ohne Abteilung' "Text"
   FROM DUAL
   WHERE
     EXISTS (SELECT 1 FROM employees
                 WHERE department_id is null);
```

- Prüft, ob überhaupt eine Zeile durch die Unterabfrage geliefert wird (quasi Prüfung auf NULL)
- Sofern eine Zeile in der Unterabfrage getroffen wird, wird die Auswertung der Unterabfrage beendet

CASE mit Unterabfrage

```
SELECT employee_id, last_name,
       (CASE
        WHEN department_id =
                  (SELECT department_id
                   FROM departments
                      WHERE location_id = 1800)
             THEN 'Canada' ELSE 'USA' END) location
FROM   employees;
```

- Skalare Unterabfragen können in CASE-Anweisungen verwendet werden

ORDER BY mit Unterabfrage

```
SELECT   employee_id, last_name
FROM     employees e
ORDER BY (SELECT department_name
          FROM departments d
          WHERE e.department_id = d.department_id);
```

- Skalare Unterabfragen können in ORDER BY -Klauseln verwendet werden

Korrelierte Unterabfragen

```
SELECT department_id, last_name, salary
    FROM employees x
    WHERE salary > (SELECT AVG(salary)
                    FROM employees
                    WHERE x.department_id = department_id)
    ORDER BY department_id;
```

- Die Unterabfrage verwendet eine Spalte aus einer Tabelle, die auch in der äußeren Abfrage benutzt wird
- Performanz-Thema:
  Die Unterabfrage wird für jede getroffene Zeile der äußeren Abfrage ausgeführt
- Korrelierte Unterabfragen können so nicht hinter FROM verwendet werden (siehe auch Lateral-Klausel)

EXISTS – Bedingung für Unterabfragen

```
SELECT department_id
   FROM departments d
   WHERE
     EXISTS  (SELECT * FROM employees e
                 WHERE d.department_id = e.department_id);
```

- Prüft, ob überhaupt eine Zeile durch die Unterabfrage geliefert wird (quasi Prüfung auf NULL)
- Sofern eine Zeile in der Unterabfrage getroffen wird, wird die Auswertung der Unterabfrage beendet

Verwendung von Unterabfragen in der FROM-Klausel (Inline View)

```
SELECT a.department_id "Department",
       a.num_emp/b.total_count "%_Employees",
       a.sal_sum/b.total_sal "%_Salary"
FROM
  (SELECT department_id, COUNT(*) num_emp, SUM(salary) sal_sum
     FROM employees
     GROUP BY department_id) a,
  (SELECT COUNT(*) total_count, SUM(salary) total_sal
     FROM employees) b
ORDER BY a.department_id;
```

- Typischer Einsatz: JOIN der Ergebnismengen von Unterabfrage

Anti-Join über Unterabfrage

```
SELECT * FROM employees
    WHERE department_id NOT IN
          (SELECT department_id FROM departments
             WHERE location_id = 1700)
    ORDER BY last_name;
```

- Ein Outer-JOIN liefert auch die Zeilen, die das JOIN-Kriterium erfüllen
- Mit Hilfe einer Unterabfrage kann man nur die nicht in Frage kommenden Zeilen erhalten

Verwendung der WITH-Klausel bei Unterabfragen

```
WITH

  dept_costs AS (

         SELECT department_name, SUM(salary) dept_total

            FROM employees e, departments d

            WHERE e.department_id = d.department_id

            GROUP BY department_name),

  avg_cost AS (
```