SQL Server – Unterabfragen

Stephan Karrer

Unterabfragen

- Können wie gewöhnliche Ausdrücke verwendet werden in:
  - WHERE-Klausel
  - HAVING-Klausel
  - FROM-Klausel
  - …
    (in der Regel überall, wo das Ergebnis der Abfrage syntaktisch passt und weiterverarbeitet werden kann)
  - Unterabfragen sind neben Joins das Arbeitskonstrukt in SQL!

```
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 ist in der Unterabfrage nicht möglich.
- **Vorsicht:**
  Unterabfragen können keinen, einen Wert (skalar), 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!
```

- Statt dem Schlüsselwort ANY kann gleichwertig das Schlüsselwort SOME verwendet werden.

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.
  - Somit muss dass mühsam via Kombination der Vergleiche der Einzelspalten erfolgen!

- ANSI nennt das Tabellen-wertige Unterabfrage
(sonst üblicherweise multi column)

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;
```

- Unterabfragen können in CASE-Anweisungen verwendet werden.
- Sie müssen nicht notwendigerweise skalar sein (IN, ANY, ALL können 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);
```

- Unterabfragen können in ORDER BY -Klauseln verwendet werden. sofern sie Werte abliefern, die als Sortierkriterium taugen.
- Im obigen Beispiel handelt es sich speziell um eine korrelierte Unterabfrage.

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 CROSS bzw. OUTER APPLY).

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

```
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 Unterabfragen.
- Diese Form der Unterabfrage wird auch als Derived Table oder Inline View bezeichnet.

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 (
         SELECT SUM(dept_total)/COUNT(*) avg
            FROM dept_costs)

SELECT * FROM dept_costs
  WHERE dept_total >
                     (SELECT avg FROM avg_cost)
  ORDER BY department_name;
```

- Statt zu Schachteln werden die Unterabfragen vorweg geschrieben, wobei die jeweils Nachfolgende auf dem Ergebnis der Vorherigen aufbauen kann.
- ANSI nennt das Subquery Factoring Clause.

Verwendung von CROSS APPLY bei Unterabfragen

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

- Unterabfragen hinter der FROM-Klausel können üblicherweise keinen Parameter einer Abfrage auf derselben Ebene benutzen, d.h. sich korrelieren.
- Mittels CROSS APPLY ist das aber möglich, wobei die Unterabfrage einen Alias haben muss .
- Der CROSS APPLY Join liefert alle Zeilen der vorangehenden Abfrage, zu denen mindestens eine Zeile durch die korrelierte Unterabfrage geliefert wird.
  Das Ganze entspricht somit eher einem INNER-Join als einem CROSS-Join.
- Der ANSI-Standard sieht dafür den JOIN mit dem Schlüsselwort LATERAL vor,
  was durch SQL Server nicht unterstützt wird.

Verwendung von OUTER APPLY bei Unterabfragen

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

- Statt CROSS APPLY kann auch OUTER APPLY verwendet werden.

- Der OUTER APPLY Join liefert alle Zeilen der vorangehenden Abfrage, egel ob eine Zeile durch die korrelierte Unterabfrage geliefert wird.
  Das Ganze entspricht somit einem LEFT OUTER-Join.

Verwendung von CROSS APPLY mit Table-Function

```
SELECT *
FROM sys.dm_exec_cached_plans AS cp
CROSS APPLY sys.dm_exec_query_plan(cp.plan_handle);
```

- Ein interessanter Anwendungsfall ist die Kombination mit SQL-Funktionen die Zeilenmengen als Ergebnis liefern (Table-Value-Functions). Deren Eingabeparameter können dadurch korreliert sein.
- Die Programmierung von solchen SQL-Funktionen ist nicht Bestandteil dieses Seminars!
- Zum obigen Beispiel: Die System-Funktion **`sys.dm_exec_query_plan`** liefert die XML-Darstellung zum Ausführungsplan und wird versorgt mit den im Cache befindlichen Ausführungsplänen.