Oracle SQL – Joins

Stephan Karrer

Beziehungen zwischen Tabellen

Tabelle: `EMPLOYEES`

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
174	Ellen	Abel	80
142	Curtis	Davies	50
102	Lex	De Haan	90
104	Bruce	Ernst	60
202	Pat	Fay	20
206	William	Gietz	110

Primärschlüssel **Fremdschlüssel**

Tabelle: `DEPARTMENTS`

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

Primärschlüssel

- Die Daten werden in der Regel auf mehrere Tabellen verteilt, um Redundanzen zu vermeiden (sog. Normalisierung)
- Der Wert in der Fremdschlüsselspalte der Tabelle "EMPLOYEES" verweist auf den zugehörigen Datensatz (Primärschlüssel) in der Tabelle "DEPARTMENTS"
- Diese Daten wieder zusammenzuführen ist der häufigste Anwendungsfall des Joins

Joins unter Oracle

Syntax
FROM first_table join_type second_table [ON join_condition] *(ANSI-Syntax 92)*
FROM first_table, second_table [WHERE join_condition] *(ANSI-Syntax 89)*

- In Spezialfällen ist Oracle-eigene Syntax möglich
- Unterstützte Arten:
  - Inner Join (Equi Join als Spezialform) mit beliebigen Bedingungen
  - Self Join
  - Cross Join (Kartesisches Produkt)
  - Ein- und zweiseitige Outer Joins mit beliebigen Bedingungen

Equi-Join (Spezialform des Inner-Join)

```
SELECT  employees.employee_id, employees.last_name,
        employees.department_id, departments.location_id
FROM employees INNER JOIN departments
   ON (employees.department_id = departments.department_id);


SELECT  e.employee_id, e.last_name, e.department_id,
        d.location_id
FROM employees e JOIN departments d
   ON (e.department_id = d.department_id);
```

- Sofern Spaltennamen in beiden Tabellen gleich sind, müssen diese qualifiziert werden!
  - Hier bieten sich Aliase an.
  - Es kann grundsätzlich alles qualifiziert werden, das erhöht die Lesbarkeit.
  - Das Schlüsselwort „AS“ für Tabellen-Aliase ist bei Oracle nicht erlaubt.
- Inner Join ist Standard, es reicht das Schlüsselwort JOIN.

Equi-Join (Spezialform des Inner-Join)

```
SELECT e.last_name, d.department_name
   FROM   employees e JOIN departments d
          ON e.department_id = d.department_id
             AND e.manager_id = d.manager_id
ORDER BY e.last_name, d.department_name;
```

- Es können auch mehrere Join-Spalten verwendet werden.

Equi-Join über WHERE-KLausel: alte Schreibweise (ANSI 89)

```
SELECT  e.employee_id, e.last_name, e.department_id,
        d.location_id
  FROM employees e, departments d
  WHERE e.department_id = d.department_id
        AND d.location_id > 1000;
```

- Hier kann nicht syntaktisch zwischen der Join-Bedingung und den sonstigen Filterbedingungen unterschieden werden.
- Da kein Schlüsselwort für den Join-Typ verwendet wird, sind nur Inner und Cross Joins formulierbar.
- So gut wie alle Hersteller unterstützen beide Schreibweisen (ANSI 89 und 92)!

Natural JOIN

```
SELECT e.last_name, d.department_name
   FROM   employees e NATURAL JOIN departments d
ORDER BY e.last_name, d.department_name;

-- entspricht
SELECT e.last_name, d.department_name
   FROM   employees e JOIN departments d
          ON e.department_id = d.department_id
             AND e.manager_id = d.manager_id
ORDER BY e.last_name, d.department_name;
```

- JOIN erfolgt automatisch über alle gleich benannten Spalten!
- Ist zwar ANSI, wird aber eher nicht verwendet!

JOIN mit USING-Klausel

```
SELECT e.last_name, d.department_name
FROM   employees e JOIN departments d
       USING (department_id)
ORDER BY e.last_name, d.department_name;
```

- Entspricht dem NATURAL JOIN mit Einschränkung der zu verwendenden Spalten.
- Ist zwar ANSI, wird aber eher nicht verwendet.

Mehrfach-Join

```
/* Das Schlüsselwort "ON" muss nach dem jeweiligen "JOIN"
   folgen  */

SELECT e.last_name, d.department_name, l.city, c.country_name
  FROM employees e INNER JOIN departments d
                     ON e.department_id = d.department_id
                   INNER JOIN locations l
                     ON d.location_id = l.location_id
                   INNER JOIN countries c
                     ON l.country_id = c.country_id ;
```

- Wenn es sich um einen Inner-Join handelt, ist die Reihenfolge egal (sprich der Server kann prinzipiell optimieren).

Mehrfach-Join (geschachtelt)

```
SELECT e.last_name, d.department_name, l.city, c.country_name
  FROM (employees e INNER JOIN departments d
                    ON e.department_id = d.department_id)
     INNER JOIN
       (locations l INNER JOIN countries c
                    ON l.country_id = c.country_id )
     ON d.location_id = l.location_id;
```

- Wer es lieber geschachtelt mag (??), kann das mit Klammerung schreiben.
  - Das wirkt sich nicht auf die interne Ausführungs-Reihenfolge aus (was immer wieder gerne behauptet wird)!

Mehrfach-Join (alte Syntax)

```
SELECT e.last_name, d.department_name, l.city, c.country_name
  FROM employees e, departments d, locations l, countries c
  WHERE e.department_id = d.department_id
        AND d.location_id = l.location_id
        AND l.country_id = c.country_id ;
```

- Selbstverständlich kann auch die ältere Syntax verwendet werden.

Non-Equi-Join

```
SELECT e.last_name, e.department_id,
       d.department_id, d.department_name
   FROM employees e JOIN departments d
        ON e.department_id > d.department_id
   WHERE e.department_id = 50 and e.employee_id = 140;
```

- Grundsätzlich sind beliebige JOIN-Bedingungen formulierbar (allgemein Theta-Join genannt).

Outer-Join (LEFT, RIGHT, FULL)

```
SELECT  e.employee_id, e.last_name, e.department_id,
        d.department_id, d.department_name

        FROM employees e LEFT OUTER JOIN departments d
        ON (e.department_id = d.department_id);


SELECT  e.employee_id, e.last_name, d.department_id,
        d.department_name

        FROM employees e FULL OUTER JOIN departments d
        ON (e.department_id = d.department_id);
```

- Es sollen auch die Zeilen berücksichtigt werden, die keinen Partner finden können!
- Selbstverständlich sind auch bei Outer-Joins beliebige JOIN-Bedingungen möglich.
- Ebenso Mehrfach-Joins mit Inner- und Outer-Joins.
  - Sobald Outer-Joins im Spiel sind, ist die Ausführungsreihenfolge nicht mehr beliebig!

Cross Join (Kartesisches Produkt)

```
SELECT  e.employee_id, e.last_name, e.department_id,
        d.department_id, d.department_name
        FROM employees e CROSS JOIN departments d;


-- bzw.
SELECT  e.employee_id, e.last_name, e.department_id,
        d.department_id, d.department_name
        FROM employees e, departments d;
```

- Jede Zeile mit jeder Zeile zu kombinieren macht nur in Ausnahmefällen Sinn!

Cross Join (Kartesisches Produkt)

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

- Nachträgliche Filterung macht keinen Sinn!
  (Auch wenn der Server eventuell durch Query-Umformung optimiert.)

Self-Join

```
SELECT  e.last_name AS emp, m.last_name AS man
        FROM employees e INNER JOIN employees m
        ON (e.manager_id = m.employee_id) ;
```

- Die zu kombinierenden Zeilen können durchaus aus einer Tabelle kommen.
  - grundsätzlich kann der Server beliebige Zeilenmengen per Join kombinieren (siehe Unterabfragen).
- Hier ist Qualifizierung via Alias Pflicht!

Mehrfach-Join mit derselben Tabelle

```
SELECT e1.last_name, d.department_name
   FROM        employees e1
        JOIN   departments d
          ON e1.department_id = d.department_id
        JOIN   employees e2
          ON e2.manager_id = d.manager_id
ORDER BY e1.last_name, d.department_name;
```

- Falls erforderlich kann dieselbe Tabelle mehrfach mit unterschiedlichen Aliasen verwendet werden.

Join-Bedingungen – Teil 1

```
SELECT  e.employee_id, e.last_name, e.department_id,
        d.location_id
    FROM employees e JOIN departments d
        ON e.department_id = d.department_id
        WHERE d.location_id > 1000;


SELECT  e.employee_id, e.last_name, e.department_id,
        d.location_id
    FROM employees e JOIN departments d
        ON e.department_id = d.department_id
           AND d.location_id > 1000;
```

- Sowohl in der WHERE- als auch in der ON-Klausel können (mehrere) Bedingungen formuliert werden
- Zuerst Filterung und dann Join?

Join-Bedingungen – Teil 2

```
SELECT d.department_name, d.department_id, e.last_name
  FROM departments d LEFT OUTER JOIN employees e
        ON  d.department_id = e.department_id
            AND   d.department_id in (10,40);


SELECT d.department_name, d.department_id, e.last_name
  FROM departments d LEFT OUTER JOIN employees e
        ON  d.department_id = e.department_id
     WHERE   d.department_id in (10,40);
```

- Vorsicht: die beiden SELECT-Anweisungen sind nicht gleichwertig!
- Laut ANSI: Die WHERE-Klausel filtert stets das Ergebnis des Ausdrucks hinter der FROM-Klausel.

Interne Verarbeitung – Nested Loop

- Grundsätzlich gibt es 3 Varianten
  (*siehe auch:*
  https://de.wikipedia.org/wiki/Joinalgorithmen
  https://learn.microsoft.com/de-de/sql/relational-databases/performance/joins)

- Variante 1: Nested Loops Joins

```
for each row R1 in the outer table
     for each row R2 in the inner table
    if R1 joins with R2 return (R1, R2)
```

  - grundsätzlich immer anwendbar
    (setzt keine Gleichheitsbeziehung voraus)

  - gute Performance bei kleinen Zeilenmengen

  - gute Performance, wenn äußere Menge klein und innere Menge anhand Index zugegriffen wird.

Interne Verarbeitung – Sort-Merge Join

- Variante 2: Sort-Merge Join, setzt Sortierung beider Zeilenmengen voraus

```
get first row R1 from input 1
get first row R2 from input 2
   while not at the end of either input  begin
        if R1 joins with R2
          then return (R1, R2)
          else if R1 < R2
                  then get next row R1 from input 1
                  else get next row R2 from input 2
   end
```

  - eignet sich nur für Gleichheitsbedingung
  - gute Performance bei großen Zeilenmengen, wenn Sortierung vorliegt

Interne Verarbeitung – Hash-Join

- Variante 3: Hash-Join

```
for each row R1 in the build table
    calculate hash value on R1 join key(s)
    insert R1 into the appropriate hash bucket
end
for each row R2 in the probe table
   calculate hash value on R2 join key(s)
   for each row R1 in the corresponding hash bucket
   if R1 joins with R2
    return (R1, R2)
end
```

  - eignet sich nur für Gleichheitsbedingung
  - gute Performance bei großen Zeilenmengen
  - aber: es muss zuerst der Hash-Table komplett anhand der einen Zeilenmenge aufgebaut werden, bevor im 2. Schritt die Hashwerte der anderen Menge dagegen geprüft werden.