



SQL Server – 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 SQL Server

Syntax

```
FROM first_table join_type second_table [ON (join_condition)]  
(ANSI-Syntax 99)
```

```
FROM first_table, second_table WHERE join_condition  
(ANSI-Syntax 92)
```

Unterstützte Arten:

- Inner Join (Equi Join als Spezialform)
- Self Join
- Cross Join (Kartesisches Produkt)
- Ein- und zweiseitige Outer Joins
- Inner und Outer Joins mit beliebigen Bedingungen

Nicht unterstützt wird NATURAL JOIN (ANSI)

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
```

Equi-Join über WHERE-Klausel: alte Schreibweise (ANSI 92)

```
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;
```

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
```

- Sowohl in der WHERE- als auch in der ON-Klausel können (mehrere) Bedingungen formuliert werden

- Zuerst Filterung und dann Join?
 ON e.department_id = d.department_id
 AND d.location_id > 1000;

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!

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 ;
```


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;
```

Outer Joins

```
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!

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 ;
```

- Das macht keinen Sinn!

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
- Hier ist Qualifizierung via Alias Pflicht!

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)
    get next row R2 from input 2
    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
  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.