5.4 Unterabfragen

In diesem Unterabschnitt erweitern wir die bisher behandelten Möglichkeiten der DML-Anweisungen in SQL durch die Möglichkeit, Ergebnisse von Abfragen direkt in anderen Anweisungen zu verwenden.

Wir werden folgende Varianten kennen lernen:

- Unterabfragen, die einen Wert liefern
- ALL-, ANY- und IN-Unterabfragen
- EXISTS-Unterabfragen

5.4.1 Unterabfragen, die einen Wert liefern

Wir benötigen Unterabfragen, um Anweisungen mit Werten zu formulieren, die selbst erst über eine Datenbankabfrage ermittelt werden. Ein Beispiel: »*Zeige die Kunden, die in derselben Stadt wohnen wie der Kunde mit der Kundennummer 106.*« Bei der Formulierung der Abfrage ist der Ort nicht bekannt. Um ihn zu ermitteln, könnten wir zunächst die Abfrage formulieren:

```
SELECT  ort
  FROM  kunde
  WHERE kunden_nr = 106
```

Den Ortsnamen könnten wir dann als Konstante in der Abfrage einsetzen, die die eigentlich erwünschten Ergebnisse liefert:

```
SELECT  kunden_nr, name
  FROM  kunde
  WHERE ort = 'Kayhude'
```

An jeder Stelle in einer SQL-Anweisung, an der ein Ausdruck stehen kann, der einen Wert berechnet, kann stattdessen eine spezielle Unterabfrage stehen, die als Ergebnis einen Wert hat. Eine Unterabfrage, die für einen Wert stehen darf, ist hierbei eine in Klammern gesetzte Abfrage-Anweisung, die zwei Bedingungen zu erfüllen hat

Die SELECT-Klausel muss genau eine Spalte oder virtuelle Spalte enthalten,

Das Ergebnis der SELECT-Klausel darf nicht mehr als ein Tupel enthalten.

Somit können wir unser obiges Problem formulieren als

```
SELECT kunden_nr, name, ort
  FROM kunde
  WHERE ort =
    (SELECT  ort
       FROM  kunde
       WHERE kunden_nr = 106);
```

Falls das Ergebnis der Unterabfrage kein Tupel enthält, wird NULL als Ergebnis zurückgeliefert, sonst der Wert der Spalte dieses Tupels. Die Gültigkeit der ersten Bedingung ist sofort syntaktisch nachprüfbar. Die Gültigkeit der zweiten Bedingung kann in einigen Fällen auch schon vor der Auswertung festgestellt werden, so z.B.

- wenn die virtuelle Spalte der SELECT-Klausel eine Aggregatfunktion enthält und keine GROUP BY-Klausel enthalten ist oder
- wenn die Bedingung in der WHERE-Klausel der Abfrage-Anweisung den Primärschlüssel mit einem Wert vergleicht.

Beispiele für diese beiden Bedingungen sind die folgenden zwei Abfrage-Anweisungen:

```
SELECT AVG(Listenpreis) FROM artikel
SELECT listenpreis FROM Artikel WHERE artikel_Nr = 'G001'
```

In den anderen Fällen wird die Anweisung ausgeführt und es gibt einen Laufzeitfehler, wenn die Unterabfrage eine Tabelle mit mehr als einem Tupel ergibt.

Unterabfragen, die einen Wert liefern, können unter anderem benutzt werden

- in der SELECT-Klausel einer SELECT-Anweisung
- in der WHERE-Klausel einer SELECT-Anweisung
- in der WHERE-Klausel einer DELETE-Anweisung
- in der WHERE-Klausel einer UPDATE-Anweisung
- in der SET-Klausel einer UPDATE-Anweisung
- in der VALUES-Klausel einer INSERT-Anweisung
- in der CHECK-Klausel einer CREATE TABLE-Anweisung

Beispiele

```
-- zeige die Abweichungen vom durchschnittlichen Listenpreis
SELECT artikel_nr, listenpreis,
       listenpreis-(SELECT AVG(listenpreis) FROM artikel) AS Abweichung
  FROM Artikel;
-- Zeige alle Artikel, die teurer sind als der Durchschnitt
SELECT artikel_nr, listenpreis
  FROM Artikel
  WHERE listenpreis > (SELECT AVG(listenpreis) FROM artikel);
-- Zeige den (oder die) teuersten Artikel
SELECT artikel_nr, listenpreis
  FROM Artikel
  WHERE listenpreis = (SELECT MAX(listenpreis) FROM artikel);
-- Lösche alle Artikel, die teurer sind als der Durchschnitt
DELETE FROM Artikel
  WHERE listenpreis > (SELECT AVG(listenpreis) FROM artikel);
```

```
-- Verbillige alle Artikel, die teurer sind als der
-- Durchschnitt, um 10 %
UPDATE Artikel
  SET listenpreis = 0.9*listenpreis
  WHERE listenpreis > (SELECT AVG(listenpreis) FROM artikel);
-- Setze für Herrn Berger die Adresse, die Herr Stein hat
UPDATE Kunde
  SET strasse =
       (SELECT Strasse FROM Kunde WHERE name LIKE 'Stein%'),
      plz =
       (SELECT plz      FROM Kunde WHERE name LIKE 'Stein%'),
      ort =
       (SELECT ort      FROM Kunde WHERE name LIKE 'Stein%')
  WHERE name LIKE 'Berger%';
-- Füge eine Bestellung mit der Nummer 176 für den Kunden Berger ein
INSERT INTO bestellung (bestell_nr, kunden_nr)
  VALUES (   176,
          (SELECT kunden_nr FROM kunde
                  WHERE NAME LIKE 'Berger%')
  );
-- Diese Abfrage führt zu einem Laufzeitfehler, da die
-- Unterabfrage mehr als ein Tupel liefert.

INSERT INTO bestellung (bestell_nr, kunden_nr)
  VALUES (   176,
          (SELECT kunden_nr FROM kunde
                  WHERE NAME LIKE 'St%')
  );
-- Diese Unterabfrage in der CHECK-Klausel
-- könnte Teil einer CREATE TABLE-Anweisung sein
...
   CHECK(stockwerk BETWEEN 0 AND
         (SELECT MAX(stockwerk) FROM haus))
```

Als Erweiterung dieses Konzeptes erlaubt SQL auch Unterabfragen, die mehr als eine Spalte enthalten. Diese Tupel können dann u.a. mit Tupeln verglichen werden. Ein Beispiel:

```
SELECT kunden_nr, name, plz, ort
FROM   kunde
WHERE (plz,ort) =
   (SELECT  plz,ort
    FROM kunde
    WHERE name = 'Stein, Peter')
```

5.4.2 Unterabfragen, die eine Relation liefern

Liefert die Unterabfrage eine Ergebnistabelle mit mehr als einer Zeile, so ist der Einsatz von Mengenoperatoren in der WHERE-Klausel der Hauptabfrage erforderlich. Folgende Operatoren werden zur Formulierung von Bedingungen angeboten, wobei θ für einen der Vergleichsoperatoren (=, <>, >, >=, <=, <) steht:

IN	prüft, ob ein Wert in der Ergebnismenge der Unterabfrage enthalten ist.
EXISTS	prüft, ob die Unterabfrage mindestens eine Zeile erbringt, die der Bedingung genügt.

Tabelle 5.2: Operatoren für Mengenvergleiche

θ ANY	prüft, ob die Bedingung für irgendeine Zeile der Unterabfrage zutrifft.
θ SOME	SOME ist ein anderer Name für ANY.
θ ALL	prüft, ob die Bedingung für alle Zeilen der Unterabfrage zutrifft.

Tabelle 5.2: Operatoren für Mengenvergleiche (Forts.)

Wir erläutern im Folgenden die einzelnen Operatoren an Beispielen.

IN

Gesucht sind die Bestellungen, in denen der Artikel mit der Nummer »K003« enthalten ist. Wir ermitteln in der Unterabfrage die Bestellnummern der Positionen, bei denen `artikel_nr` den Wert `'K003'` hat. Dieses können mehrere Positionen sein.

```
SELECT bestell_nr, bestelldatum
    FROM bestellung
    WHERE bestell_nr IN
      (SELECT bestell_nr
       FROM  position
       WHERE artikel_nr = 'K003');
 bestell_nr bestelldatum
========== ============
        151 2000-04-28
        152 2000-04-30
```

Zur Erläuterung betrachten wir die Unterabfrage einmal für sich. Sie liefert mehrere Zeilen:

```
SELECT bestell_nr
    FROM  position
    WHERE artikel_nr = 'K003';
 bestell_nr
==========
        151
        152
```

5.4.6 Unterabfragen in der CREATE TABLE-Anweisung

Wir haben in Kapitel 4 Assertions als Mittel zur Kontrolle tabellenübergreifender Konsistenzbedingungen behandelt, die aber von vielen DBMS nicht unterstützt werden. Eine Alternative dazu bietet die Möglichkeit, in CHECK-Klauseln Unterabfragen einzubauen. Das folgende Beispiel ist unter SYBASE realisierbar:

```
CREATE TABLE kunde2(
   kunden_nr          Kunden_Key     NOT NULL,
   status             Kundenstatus NOT NULL,
   kname              PersonenName   NOT NULL,
   strasse            Strassenname   NOT NULL,
   plz                Postleitzahl   NOT NULL,
   ort                ortsname       NOT NULL,
   letzte_bestellung  DATE,
   letzte_werbeaktion DATE,
   zahlungsart           Zahlungsart    NOT NULL,
   PRIMARY KEY (kunden_nr),
   CHECK (zahlungsart <>'B' OR
               kunden_nr IN
                  (SELECT kunden_nr FROM girokonto g))
   );
```

Der Nachteil dieser Lösung gegenüber der Assertion ist, dass die CHECK-Klausel genau zu einer Tabelle gehört, was nicht bei allen tabellenübergreifenden Integritätsbedingungen sinnvoll ist.