SQL,Teil 3:
Unterabfragen, Views & Berechnungen

W. Spiegel

Übersicht

- Hinweis . . .
- Unterabfragen (subqueries)
- Virtuelle Spalten: AS
- Logik
- Berechnungen: Aggregatfunktionen in SQL
- GROUP BY & HAVING
- Views (Sichten)
- Beispiel
- Aufgaben

Hinweis . . .

Diesen Hinweis entnehmen wir einer guten Einführung in unser Thema:

> Abfragen erstellen "keine neuen Daten. Vielmehr sind die berechneten Werte temporär. Sie werden bei jeder Durchführung einer Abfrage neu berechnet."

(aus: H. Eirund/U. Kohl: Datenbanken - leicht gemacht, Stuttgart, Leipzig: B. G. Teubner, 2000, S. 88)

Unterabfragen (subqueries)

Unterabfragen (subqueries) sind verschachtelte SQL-Abfragen, Beispiel:

```
SELECT name, lehrernr
      FROM sar_lehrer
      WHERE vorname  IN
      (
            SELECT vorname
            FROM sar_lehrer
            WHERE vorname = 'Walter'
      );
```

Wirkung:

```
name    | lehrernr
====================
Spiegel | 1
```

Wie funktionieren Unterabfragen? Die innere Abfrage liefert eine Menge von Datensätzen zurück (Selektion!), auf der dann die äußere Abfrage arbeitet. Der Operator IN vergleicht Spaltenwerte aus der Spalte vorname mit jedem Element aus der inneren Abfrage. Man beachte, dass die Abfrage oben kein sinnvolles Beispiel für Unterabfragen ist . . .

Virtuelle Spalten: AS

Man kann in SQL **virtuelle Spalten** beziehungsweise **Tabellen** einführen über den Zusatz AS, virtuell, da die Tabellen/Spalten nur für diese Abfrage zur Verfügung stehen. Gerne wird ein **Alias** auch als Abkürzung für einen Tabellennamen benutzt wie im folgenden Beispiel:

```
SELECT  name, thema
        FROM  sar_lehrer AS l, sar_projekt AS p
        WHERE l.prowonr = p.prowonr;
```

In der folgenden Abfrage wird eine virtuelle Spalte eingeführt, ein sinnvolleres Beispiel wäre die Multiplikation mit dem Mehrwertsteuersatz.

```
SELECT name,thema, anzahl * lehrernr AS unsinn
       FROM  sar_lehrer AS l, sar_projekt AS p
       WHERE l.prowonr = p.prowonr;
```

Logik

Beim Join kann man die Bedingung durch logische Operatoren wie AND, OR oder NOT "verfeinern":

```
SELECT name, klasse, thema
       FROM sar_wahl AS w, sar_projekt AS p, sar_schueler AS s
       WHERE w.prowonr = p.prowonr
       AND   w.schuelernr = s.schuelernr
       ORDER BY klasse, name;
```

Das Ergebnis ist wirklich großartig, die Idee zu dieser Abfrage stammt NICHT von mir . . .
⇒ **Frage:** Nach welcher Information wird hier gefragt?

Berechnungen: Aggregatfunktionen in SQL

Man will gelegentlich Berechnungen über alle Datensätze durchführen, die eine bestimmte Bedingung erfüllen, zum Beispiel Minimum oder Maximum-Bestimmung. In SQL benutzt man an dieser Stelle sogenannte **Aggregatfunktionen**.

```
SELECT count(name)
       FROM sar_wahl AS w, sar_projekt AS p, sar_schueler AS s
       WHERE w.prowonr = p.prowonr
       AND   w.schuelernr = s.schuelernr
       AND p.thema = 'Easy-Webdesign';
```

Ergebnis:

```
count(name)
=============
10
```

⇒ **Frage:** Nach welcher Information wird hier gefragt?

Weitere Aggregatfunktionen:

- **sum** (anzahl)
- **min** (anzahl)
- **max** (anzahl)
- **avg** (anzahl) (Für die Bestimmung des Mittelwertes)

GROUP BY & HAVING

Abfragen geben Datensätze ungeordnet zurück, wenn ich aber beispielsweise wissen möchte, wieviel Schüler ein bestimmtes Projekt gewählt haben, muss ich die Datensätze irgendwie gruppieren, für die Gruppenbildung entscheidend sind hierbei gleiche Werte in einer bestimmten Spalte, Beispiel:

```
SELECT thema, count(*)
       FROM sar_wahl AS w, sar_projekt AS p, sar_schueler AS s
       WHERE w.prowonr = p.prowonr
       AND   w.schuelernr = s.schuelernr
       GROUP BY thema;
```

Ergebnis:

```
thema                | count(*)
================================
Easy-Webdesign       | 9.0
HTML für Einsteiger  | 7.0
So ein Theater       | 8.0
```

Die Bedingung unter WHERE sucht die Schülerwahlen aus, dann wird nach dem gewählten Projekt gruppiert und die Anzahl der Schülerwahlen summiert. Man kann die Gruppenbildung kombinieren mit einer "Selektion" über den HAVING-Befehl:

```
SELECT thema, count(*)
       FROM sar_wahl AS w, sar_projekt AS p, sar_schueler AS s
       WHERE w.prowonr = p.prowonr
       AND   w.schuelernr = s.schuelernr
       GROUP BY thema
       HAVING count(*) < 9;
```

Ergebnis:

```
thema                | count(*)
================================
HTML für Einsteiger  | 7.0
So ein Theater       | 8.0
```

Über den BETWEEN-Operator kann man auch ein Intervall angeben:

```
SELECT thema, count(*)
       FROM sar_wahl AS w, sar_projekt AS p, sar_schueler AS s
       WHERE w.prowonr = p.prowonr
       AND   w.schuelernr = s.schuelernr
       GROUP BY thema
       HAVING count(*) BETWEEN 4 AND 7;
```

Ergebnis:

```
thema                | count(*)
================================
HTML für Einsteiger  | 7.0
```

Views (Sichten)

Manchmal ist in SQL eine eingeschränkte Sicht auf die Daten notwendig, sei es aus datenschutzrechtlichen Gründen oder aus eher praktischen Erwägungen, in SQL kann man an dieser Stelle Views (Sichten) benutzen:

```
CREATE VIEW prowo
       AS SELECT prowonr, thema, anzahl
       FROM sar_projekt;
```

Ergebnis ist eine Projekt-Tabelle mit drei Spalten, wie man der Abfrage SELECT * FROM prowo; entnimmt.
PS: Views lassen sich mit dem DROP-Befehl wieder löschen: DROP VIEW prowo;
sie verhalten sich also genauso wie Tabellen. Was aber passiert beim Einfügen neuer Datensätzte mittels INSERT? (Aufgabe!)

Beispiel

Siehe die Datei **"beispiel_sql_03.txt"** in der Zip-Datei prowo.zip (in gfsqlite als SQL-Datei einlesen, davor die prowo-Datenbank öffnen), vergleiche auch im Datenbanken-Reader S. 93-96 (Abschnitt 5.1 & 5.2).

Aufgaben

1. Beantworte die Fragen im Text!
2. Kopiere die Datei **"beispiel_sql_03.txt"** vom Webserver in dein Verzeichnis, und öffne sie in gfsqlite als SQL-Datei (Davor die prowo-Datenbank öffnen!). Versuche die Ausgabe im unteren Fenster den einzelnen Befehlen zuzuordnen.
3. Sammle Erfahrungen mit dem IN- sowie dem BETWEEN-Operator

4. Denke dir Abfragen mit den logischen Operatoren AND, OR, NOT aus.
5. Erzeuge mit VIEW eine neue Sicht auf die Datenbank und versuche, Daten einzufügen!
6. Vereinfache dir das Leben mit virtuellen Spalten (**AS**), und benutze an sinnvoller Stelle die Aggregatfunktionen.
7. Gib deine GROUP BY-Abfrage ein . . .
8. Suche nach einer sinnvollen Unterabfrage . . .

W. Spiegel, E-Mail: walter.spiegel@web.de