SQL,Teil 2: SELECT

W. Spiegel

Übersicht

- SELECT
- Mehrfache Werte vermeiden: SELECT DISTINCT
- Ausgabe ordnen: ORDER BY
- Projektion
- Selektion: WHERE
- Join mit SELECT
- Beispiel
- Aufgaben

SELECT

Die SELECT-Anweisung ist in SQL eine sehr flexible Anweisung. Hier die einfachste Form: SELECT * FROM sar_schueler;

Wirkung: Aus der Tabelle sar_schueler werden sämtliche Datensätze angezeigt, der ***** ist ein Joker: es werden alle Spalten der Tabelle angezeigt. Die Reihenfolge der Spalten entspricht in gfsqlite der Vereinbarung in der CREATE-Anweisung:

```
CREATE TABLE sar_schueler (schuelernr integer primary key,
name varchar(20), vorname varchar(20), Klasse varchar(5),
Geschlecht varchar(1))
```

Will man diese Reihenfolge verändern, so ruft man beispielsweise auf:

```
SELECT name,vorname,geschlecht,klasse,schuelernr FROM sar_schueler;
```

Mehrfache Werte vermeiden: SELECT DISTINCT

Manchmal kann es in einer Tabelle mehrfache Einträge zu einer Eigenschaft geben. Beispiel: In der Tabelle sar_wahl gibt es für jeden Schüler zwei Einträge: seine beiden Wahlen eben. Will man Schüler nur einmal anzeigen lassen (Beispielsweise sinnvoll bei der Frage, ob alle Schüler erfasst wurden), dann verwendet man den Zusatz DISTINCT:

```
SELECT DISTINCT schuelernr
FROM sar_wahl;
```

Ausgabe ordnen: ORDER BY

Die Wirkung des Befehls eben ist nicht so, wie man sich das dachte, denn die Ausgabe ist ungeordnet, also geben wir ein:

```
SELECT DISTINCT schuelernr
FROM sar_wahl
ORDER BY schuelernr;
```

Jetzt ist die Ausgabe ok, aber eigentlich sollten sie ja umgekehrt geordnet sein:

```
SELECT DISTINCT schuelernr
FROM sar_wahl
ORDER BY schuelernr DESC;
```

Wir können auch mehrfach sortieren und müssen nicht immer den Spaltennamen im Kopf haben:

```
SELECT schuelernr,prowonr
FROM sar_wahl
ORDER BY 1,2;
```

Wirkung: Sortiert die Zeilen aus der Tabelle sar_wahl nach der Schülernummer und innerhalb der Schülernummer nach der Projektnummer. Die **1** bezieht sich also auf die erste Spalte: schuelernr, die **2** auf die zweite Spalte (prowonr)

Projektion

ist in SQL einfach: wir wollen einzelne Spalten ausblenden, kein Problem:

```
SELECT name,vorname,klasse FROM sar_schueler;
```

und schon haben wir zwei Spalten (Welche?) ausgeblendet.

Selektion: WHERE

Bei der Selektion geht es darum, Datensätze (sprich: Zeilen) auszuwählen, wir müssen also eine Bedingung angeben:

```
SELECT name,vorname FROM sar_lehrer
WHERE name='Spiegel';
```

Die Bedingung folgt hinter dem WHERE. Gebe ich den Namen falsch an, so kommt die interessante Mitteilung

```
Keine Zeilen gefunden . . .
```

Ich kann Bedingungen auch schachteln, aber das verschieben wir auf's nächste Mal!

Join mit SELECT

Die Selektion ist der Schlüssel zum **Join** . Schauen wir uns erst das **kartesische Produkt** zweier Tabellen an:

```
SELECT name, thema
FROM sar_lehrer, sar_projekt;
```

Hier wird jeder Eintrag aus der Tabelle sar_lehrer kombiniert mit jedem Eintrag aus der Tabelle sar_projekt, gibt es also insgesamt vier Datensätze (= Zeilen) in der Tabelle sar_lehrer und fünf Projekte, so hat meine Ergebnistabelle insgesamt 4*5=20 (sinnlose!) Zeilen. Besser geht es so:

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

Hier werden genau diejenigen Datensätze (=Zeilen) ausgewählt, deren prowonr in beiden Tabellen identisch ist (**Join!**). Damit die Ausgabe "schön" aussieht, schreiben wir natürlich

```
SELECT name, thema
FROM sar_lehrer, sar_projekt
WHERE sar_lehrer.prowonr = sar_projekt.prowonr
ORDER BY name DESC;
```

Das Ergebnis ist so großartig, dass ich es euch nicht vorenthalten kann:

```
name    | thema
================================
Spiegel | So ein Theater
Noether | HTML für Einsteiger
Hilbert | HTML für Einsteiger
Hegel   | Easy-Webdesign
Born    | Easy-Webdesign
```

Wir erkennen (hoffentlich), dass man Abfragen beliebig komplex gestalten kann. Hier der "Beweis":

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

Beachte: Wir haben es hier mit einer KURZschreibweise zu tun. Statt dem vollen Tabellennamen sar_lehrer genügt auch ein l. Das AS ist ein sogenannter **Tabellenalias** , man vereinbart damit einen Kurznamen für eine Tabelle. Das funktioniert auch mit Spalten, so dass man auf die Schnelle eine neue virtuelle Spalte definieren kann, mehr hierzu demnächst!

Beispiel

Siehe die Datei **"beispiel_sql_02.txt"** in der Zip-Datei prowo.zip (in gfsqlite als SQL-Datei einlesen, davor die prowo-Datenbank öffnen), vergleiche auch im Datenbanken-Reader S. 90-95 (Abschnitt 5.1 & 5.2), allerdings bezogen auf MS-SQL (=Access).

Aufgaben

1. Kopiere die Datei **"beispiel_sql_02.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.

2. Ordne die einzelnen Tabellen der prowo-DB nach unterschiedlichen Merkmalen (auch rückwärts), und probiere auch Ordnen nach mehreren Eigenschaften.

3. Zeit für die Selektion: Denke dir zu jeder Tabelle der prowo-DB mindestens drei SELECT-Abfragen inclusive WHERE aus, die du in einer Datei mit Namen drei_30.txt in deinem Verzeichnis speicherst. Lade diese Datei anschliessend wieder über "Datei → **SQL-Datei einlesen**" (Tipp: zuvor das untere Fenster löschen)

4. Warum macht der Join nur über die prowonr Sinn? Gibt es weitere Tabellen, wo man einen Join versuchen könnte? Falls ja, welche?

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