KAPITEL 4

SQL-Abfragen: Die Grundlagen

Eine SELECT-Anweisung, die üblicherweise aus sechs Klauseln besteht, bezeichnet man auch als Abfrage. Die einzelnen Klauseln werden jeweils in einem eigenen Abschnitt dieses Kapitels näher behandelt:

1. SELECT
2. FROM
3. WHERE
4. GROUP BY
5. HAVING
6. ORDER BY

Der letzte Abschnitt in diesem Kapitel befasst sich mit der LIMIT-Klausel, die von *MySQL*, *PostgreSQL* und *SQLite* unterstützt wird.

Die Codebeispiele in diesem Kapitel beziehen sich auf vier Tabellen:

`waterfall`
: Wasserfälle auf der Oberen Halbinsel von Michigan

`owner`
: Besitzer der Wasserfälle

`county`
: Counties, in denen sich die Wasserfälle befinden

`tour`
: Touren, die an mehreren Wasserfällen haltmachen

Hier ist eine Beispielabfrage, die unsere sechs wichtigsten Klauseln verwendet. Sie wird gefolgt von den Abfrageergebnissen, die man auch als *Ergebnismenge* bezeichnet.

```
-- Touren mit zwei oder mehr öffentlichen Wasserfällen
SELECT   t.name AS tour_name,
         COUNT(*) AS num_waterfalls
FROM     tour t LEFT JOIN waterfall w
         ON t.stop = w.id
WHERE    w.open_to_public = 'y'
GROUP BY t.name
HAVING   COUNT(*) >= 2
ORDER BY tour_name;

tour_name  num_waterfalls
---------- --------------
M-28                    6
Munising                6
US-2                    4
```

Eine Datenbank *abzufragen*, bedeutet, Daten aus einer Datenbank, meist aus einer oder mehreren Tabellen, abzurufen oder zu beziehen.

Es ist auch möglich, einen *View* statt einer Tabelle abzufragen. Views sehen wie Tabellen aus und werden aus Tabellen abgeleitet, enthalten selbst aber keine Daten. Mehr zu Views finden Sie in »Views« auf Seite 136 in Kapitel 5.

Die SELECT-Klausel

Die SELECT-Klausel gibt die Spalten an, die eine Anweisung für Sie zurückliefern soll.

In der SELECT-Klausel folgt auf das Schlüsselwort SELECT eine Liste mit Spaltennamen und/oder Ausdrücken, die durch Kommata voneinander getrennt sind. Jeder Spaltenname und/oder Ausdruck wird dann zu einer Spalte in den Ergebnissen.

Spalten auswählen

Die einfachste SELECT-Klausel listet einen oder mehrere Spaltennamen aus den Tabellen in der FROM-Klausel auf:

```
SELECT id, name
FROM owner;
```

```
id    name
----- ----------------
    1 Pictured Rocks
    2 Michigan Nature
    3 AF LLC
    4 MI DNR
    5 Horseshoe Falls
```

Alle Spalten auswählen

Um alle Spalten aus einer Tabelle zurückzuliefern, können Sie einen einzelnen Asterisk verwenden, anstatt alle Spaltennamen anzugeben:

```
SELECT *
FROM owner;

id    name             phone         type
----- ---------------- ------------- --------
    1 Pictured Rocks   906.387.2607  public
    2 Michigan Nature  517.655.5655  private
    3 AF LLC                         private
    4 MI DNR           906.228.6561  public
    5 Horseshoe Falls  906.387.2635  private
```

Der Asterisk ist eine hilfreiche Abkürzung, wenn Sie Abfragen testen, weil Sie sich damit eine Menge Tipparbeit ersparen. Allerdings ist es riskant, den Asterisk in Ihrem Produktionscode einzusetzen, weil sich die Spalten in einer Tabelle mit der Zeit ändern können, sodass Ihr Code fehlerhaft wird, wenn es weniger oder mehr Spalten gibt als erwartet.

Ausdrücke auswählen

Sie können nicht nur einfach Spalten auflisten, sondern auch komplexere Ausdrücke in der SELECT-Klausel angeben, um sich in den Ergebnissen bestimmte Spalten ausgeben zu lassen.

Die folgende Anweisung enthält einen Ausdruck zum Berechnen eines Bevölkerungsrückgangs von 10%, gerundet auf null Dezimalstellen:

```
SELECT name, ROUND(population * 0.9, 0)
FROM county;

name       ROUND(population * 0.9, 0)
---------- --------------------------
Alger                            8876
Baraga                           7871
Ontonagon                        7036
...
```

Funktionen auswählen

Ausdrücke in der SELECT-Liste beziehen sich üblicherweise auf Spalten in den Tabellen, aus denen Sie die Daten beziehen, es gibt aber auch Ausnahmen. So ist zum Beispiel eine gebräuchliche Funktion, die sich nicht auf irgendwelche Tabellen bezieht, die Funktion, die das aktuelle Datum zurückgibt:

```
SELECT CURRENT_DATE;

CURRENT_DATE
-------------
2021-12-01
```

Der gezeigte Code funktioniert in *MySQL*, *PostgreSQL* und *SQLite*. Äquivalenten Code für andere RDBMS gibt es in »Datum/Zeit-Funktionen« auf Seite 213 in Kapitel 7.

Die meisten Abfragen enthalten sowohl eine SELECT- als auch eine FROM-Klausel, doch für bestimmte Datenbankfunktionen wie etwa CURRENT_DATE ist nur die SELECT-Klausel erforderlich.

Es ist auch möglich, Ausdrücke in die SELECT-Klausel einzusetzen, die *Unterabfragen* sind (das sind Abfragen, die in andere Abfragen geschachtelt sind). Mehr dazu finden Sie in »Unterabfragen auswählen« auf Seite 72.

Aliasse für Spalten

Der Zweck eines *Spaltenalias* besteht darin, einer Spalte oder einem Ausdruck, die bzw. der in der SELECT-Klausel aufgeführt ist, einen

temporären Namen zu geben. Dieser temporäre Name oder Spaltenalias wird dann in den Ergebnissen als Spaltenname angezeigt.

Beachten Sie, dass diese Namensänderung nicht von Bestand ist, da die Spaltennamen in den Originaltabellen unverändert bleiben. Der Alias existiert nur in der Abfrage.

Dieser Code zeigt drei Spalten an.

```
SELECT id, name,
       ROUND(population * 0.9, 0)
FROM county;

id    name       ROUND(population * 0.9, 0)
----- ---------- --------------------------
    2 Alger                            8876
    6 Baraga                           7871
    7 Ontonagon                        7036
...
```

Nehmen wir einmal an, wir wollten die Spaltennamen in den Ergebnissen umbenennen. `id` ist zu uneindeutig, und wir würden der Spalte gern einen aussagekräftigeren Namen geben. `ROUND(population * 0.9, 0)` ist zu lang und der Name soll einfacher sein.

Um einen Spaltenalias zu erzeugen, setzen Sie hinter einen Spaltennamen oder Ausdruck entweder einen Aliasnamen oder das Schlüsselwort `AS` und einen Aliasnamen.

```
-- alias_name
SELECT id county_id, name,
       ROUND(population * 0.9, 0) estimated_pop
FROM county;
```

oder:

```
-- AS alias_name
SELECT id AS county_id, name,
       ROUND(population * 0.90, 0) AS estimated_pop
FROM county;

county_id  name       estimated_pop
---------- ---------- -------------
         2 Alger                8876
         6 Baraga               7871
         7 Ontonagon            7036
...
```

Beide Möglichkeiten werden in der Praxis eingesetzt, um Aliasse zu erzeugen. In der SELECT-Klausel ist die zweite Option beliebter, weil das Schlüsselwort AS es visuell leichter macht, Spaltennamen und Aliasse in einer langen Liste aus Spaltennamen zu unterscheiden.

Ältere Versionen von *PostgreSQL* verlangen die Verwendung von AS, wenn ein Spaltenalias erzeugt werden soll.

Auch wenn Spaltenaliasse nicht notwendig sind, werden sie beim Arbeiten mit Ausdrücken doch dringend empfohlen, um den Spalten in den Ergebnissen vernünftige Namen zu geben.

Aliasse mit Groß- und Kleinschreibung sowie Interpunktion

Wie die Spaltenaliasse county_id und estimated_pop zeigen, besteht die Konvention, beim Benennen von Spaltenaliassen Kleinbuchstaben zu verwenden sowie für Leerzeichen Unterstriche zu setzen.

Sie können auch Aliasse erzeugen, die Großbuchstaben, Leerzeichen und Interpunktionszeichen enthalten. Verwenden Sie hierzu doppelte Anführungszeichen, wie dieses Beispiel demonstriert:

```
SELECT id AS "Waterfall #",
  name AS "Waterfall Name"
FROM waterfall;

Waterfall #  Waterfall Name
------------ ---------------
           1 Munising Falls
           2 Tannery Falls
           3 Alger Falls
...
```

Spalten qualifizieren

Nehmen wir einmal an, Sie schrieben eine Abfrage, die Daten aus zwei Tabellen bezieht, die beide eine Spalte namens name enthalten. Falls Sie nur name in der SELECT-Klausel angeben, weiß der Code nicht, welche Tabelle Sie meinen.

Um dieses Problem zu lösen, können Sie einen Spaltennamen durch seinen Tabellennamen *qualifizieren*. Mit anderen Worten, Sie geben einer Spalte mithilfe der *Punktnotation* ein Präfix, um näher zu bestimmen, zu welcher Tabelle sie gehört, wie in `table_name.column_name`.

Im folgenden Beispiel wird eine einzelne Tabelle abgefragt. Es ist also eigentlich nicht nötig, die Spalten hier zu qualifizieren, es dient lediglich der Demonstration. Und so würden Sie eine Spalte durch ihren Tabellennamen qualifizieren:

```
SELECT owner.id, owner.name
FROM owner;
```

Falls Sie in SQL einen Fehler erhalten, wenn Sie einen *mehrdeutigen Spaltennamen* referenzieren, bedeutet dies, dass mehrere Tabellen in Ihrer Abfrage eine Spalte desselben Namens enthalten und Sie nicht angegeben haben, auf welche Tabelle/Spalte Sie sich beziehen. Sie können den Fehler beheben, indem Sie den Spaltennamen qualifizieren.

Tabellen qualifizieren

Wenn Sie einen Spaltennamen anhand seines Tabellennamens qualifizieren, können Sie auch diese Tabelle anhand ihres Datenbank- oder Schemanamens qualifizieren. Die folgende Abfrage bezieht Daten speziell aus der `owner`-Tabelle im `sqlbook`-Schema:

```
SELECT sqlbook.owner.id, sqlbook.owner.name
FROM sqlbook.owner;
```

Der gezeigte Code ist recht lang, da `sqlbook.owner` mehrmals wiederholt wird. Um sich das Tippen zu ersparen, können Sie einen *Tabellenalias* angeben. Das folgende Beispiel gibt der Tabelle `owner` den Alias `o`:

```
SELECT o.id, o.name
FROM sqlbook.owner o;
```

oder:

```
SELECT o.id, o.name
FROM owner o;
```

Spaltenaliasse versus Tabellenaliasse

Spaltenaliasse werden innerhalb der SELECT-Klausel definiert, um eine Spalte in den Ergebnissen umzubenennen. Es ist üblich, AS zu verwenden, wird aber nicht verlangt.

```
-- Spaltenalias
SELECT num AS new_col
FROM my_table;
```

Tabellenaliasse werden innerhalb der FROM-Klausel definiert, um einen temporären Namen für eine Tabelle zu erzeugen. Es ist üblich, AS wegzulassen, obwohl es auch mit AS funktioniert.

```
-- Tabellenalias
SELECT *
FROM my_table mt;
```

Unterabfragen auswählen

Eine *Unterabfrage* ist eine Abfrage, die in eine andere Abfrage geschachtelt ist. Unterabfragen können in verschiedenen Klauseln zu finden sein, einschließlich der SELECT-Klausel.

Im folgenden Beispiel wollen wir zusätzlich zu `id`, `name` und `population` auch die durchschnittliche Bevölkerungszahl aller Counties sehen. Indem wir eine Unterabfrage einfügen, erzeugen wir eine neue Spalte in den Ergebnissen für die durchschnittliche Bevölkerungszahl.

```
SELECT id, name, population,
       (SELECT AVG(population) FROM county)
       AS average_pop
FROM county;

id    name       population  average_pop
----- ---------- ----------- ------------
    2 Alger             9862        18298
    6 Baraga            8746        18298
    7 Ontonagon         7818        18298
...
```

Ein paar Anmerkungen:

- Eine Unterabfrage muss in runde Klammern eingeschlossen sein.
- Wenn man eine Unterabfrage in der SELECT-Klausel schreibt, wird dringend empfohlen, einen Spaltenalias anzugeben, was mit average_pop hier geschehen ist. Auf diese Weise trägt die Spalte in den Ergebnissen einen einfachen Namen.
- Es gibt nur einen Wert in der Spalte average_pop, der in allen Zeilen wiederholt wird. Wenn man eine Unterabfrage in die SELECT-Klausel einfügt, muss das Ergebnis der Unterabfrage eine einzelne Spalte und entweder keine oder eine Zeile zurückgeben, wie in der folgenden Unterabfrage demonstriert wird, die die durchschnittliche Bevölkerungszahl berechnet.

```
SELECT AVG(population) FROM county;

AVG(population)
---------------
          18298
```

- Falls die Unterabfrage keine Zeilen zurückliefert, wird die neue Spalte mit NULL-Werten gefüllt.

Nichtkorrelierte versus korrelierte Unterabfragen

Das gezeigte Beispiel ist eine *nichtkorrelierte Unterabfrage*, was bedeutet, dass die Unterabfrage sich nicht auf die äußere Abfrage bezieht. Die Unterabfrage kann für sich allein, also unabhängig von der äußeren Abfrage, ausgeführt werden.

Die andere Art von Unterabfrage wird *korrelierte Unterabfrage* genannt und ist eine, die sich auf die Werte in der äußeren Abfrage bezieht. Sie verlangsamt oft signifikant die Verarbeitungszeit, sodass es am besten ist, die Abfrage umzuschreiben und stattdessen einen JOIN zu verwenden. Es folgt ein Beispiel einer korrelierten Unterabfrage mit effizienterem Code.

Leistungsprobleme mit korrelierten Unterabfragen

Die folgende Abfrage gibt die Anzahl der Wasserfälle der einzelnen Besitzer aus. Beachten Sie, dass sich der o.id = w.owner_id-Schritt in

der Unterabfrage auf die owner-Tabelle in der äußeren Abfrage bezieht, wodurch dies zu einer korrelierten Unterabfrage wird.

```
SELECT o.id, o.name,
       (SELECT COUNT(*) FROM waterfall w
       WHERE o.id = w.owner_id) AS num_waterfalls
FROM owner o;

id    name             num_waterfalls
----- ---------------- --------------
    1 Pictured Rocks                3
    2 Michigan Nature               3
    3 AF LLC                        1
    4 MI DNR                        1
    5 Horseshoe Falls               0
```

Eine bessere Vorgehensweise wäre es, die Abfrage mit einem Join umzuschreiben. Auf diese Weise werden die Tabellen zuerst zusammengefasst, bevor der Rest der Abfrage ausgeführt wird. Das geht viel schneller, als für jede Datenzeile immer wieder eine Unterabfrage auszuführen. Mehr zu Joins finden Sie in »Tabellen mit Joins zusammenbringen« auf Seite 262 in Kapitel 9.

```
SELECT   o.id, o.name,
         COUNT(w.id) AS num_waterfalls
FROM     owner o LEFT JOIN waterfalls w
         ON o.id = w.owner_id
GROUP BY o.id, o.name

id    name             num_waterfalls
----- ---------------- --------------
    1 Pictured Rocks                3
    2 Michigan Nature               3
    3 AF LLC                        1
    4 MI DNR                        1
    5 Horseshoe Falls               0
```

DISTINCT

Wenn eine Spalte in der SELECT-Klausel aufgeführt ist, werden standardmäßig alle Zeilen zurückgeliefert. Um expliziter zu sein, können Sie das Schlüsselwort ALL einfügen, aber das ist wirklich optional. Die folgenden Abfragen liefern jede type/open_to_public-Kombination zurück.