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.

KAPITEL 10

Wie mache ich ...?

Dieses Kapitel soll als schnelle Referenz für häufig gestellte SQL-Fragen dienen, die mehrere Konzepte miteinander kombinieren:

- Zeilen finden, die doppelte Werte enthalten
- Zeilen mit dem Maximalwert für eine andere Spalte auswählen
- Text aus mehreren Feldern in einem einzigen Feld verketten
- Alle Tabellen finden, die einen bestimmten Spaltennamen enthalten
- Eine Tabelle aktualisieren, deren ID einer anderen Tabelle entspricht

Zeilen finden, die doppelte Werte enthalten

Die folgende Tabelle listet sieben Teesorten auf sowie die Temperaturen, bei denen sie ziehen sollten. Beachten Sie, dass es zwei Gruppen mit doppelten `tea`/`temperature`-Werten gibt; diese werden fett dargestellt.

```
SELECT * FROM teas;

+----+--------+-------------+
| id | tea    | temperature |
+----+--------+-------------+
|  1 | green  |         170 |
|  2 | black  |         200 |
|  3 | black  |         200 |
|  4 | herbal |         212 |
```

```
|  5 | herbal |         212 |
|  6 | herbal |         210 |
|  7 | oolong |         185 |
+----+--------+-------------+
```

Dieser Abschnitt behandelt zwei unterschiedliche Szenarien:

- Zurückliefern aller einmaligen tea/temperature-Kombinationen
- Zurückliefern nur der Zeilen mit doppelten tea/temperature-Werten

Alle einmaligen Kombinationen zurückliefern

Um alle doppelt vorhandenen Werte auszuschließen und nur die einmaligen Zeilen einer Tabelle zurückzuliefern, verwenden Sie das Schlüsselwort DISTINCT.

```
SELECT DISTINCT tea, temperature
FROM teas;

+--------+-------------+
| tea    | temperature |
+--------+-------------+
| green  |         170 |
| black  |         200 |
| herbal |         212 |
| herbal |         210 |
| oolong |         185 |
+--------+-------------+
```

Potenzielle Erweiterungen

Um die Anzahl der einmaligen Zeilen in einer Tabelle zurückzuliefern, setzen Sie die Schlüsselwörter COUNT und DISTINCT zusammen ein. Näheres finden Sie im Abschnitt »DISTINCT« in Kapitel 4.

Nur die Zeilen mit doppelt vorhandenen Werten zurückliefern

Die folgende Abfrage identifiziert die Zeilen in der Tabelle, in denen doppelt vorhandene Werte stehen.

```
WITH dup_rows AS (
     SELECT tea, temperature,
            COUNT(*) as num_rows
     FROM teas
     GROUP BY tea, temperature
     HAVING COUNT(*) > 1)

SELECT t.id, d.tea, d.temperature
FROM teas t INNER JOIN dup_rows d
     ON t.tea = d.tea
     AND t.temperature = d.temperature;

+----+--------+-------------+
| id | tea    | temperature |
+----+--------+-------------+
|  2 | black  |         200 |
|  3 | black  |         200 |
|  4 | herbal |         212 |
|  5 | herbal |         212 |
+----+--------+-------------+
```

Erklärung

Der größte Teil der Arbeit geschieht in der dup_rows-Abfrage. Alle tea/temperature-Kombinationen werden gezählt, und dann werden mit der HAVING-Klausel nur die Kombinationen aufgehoben, die mehr als einmal vorkommen. So sieht dup_rows aus:

```
+--------+-------------+----------+
| tea    | temperature | num_rows |
+--------+-------------+----------+
| black  |         200 |        2 |
| herbal |         212 |        2 |
+--------+-------------+----------+
```

Der Zweck des JOIN in der zweiten Hälfte der Abfrage ist, die id-Spalte zurück in die fertige Ausgabe zu ziehen.

Schlüsselwörter in der Abfrage

- **WITH dup_rows** ist der Beginn einer Common Table Expression, die es Ihnen erlaubt, innerhalb einer einzigen Abfrage mit mehreren SELECT-Anweisungen zu arbeiten.

- **`HAVING COUNT(*) > 1`** nutzt die `HAVING`-Klausel, die es Ihnen erlaubt, auf einer Aggregation wie `COUNT()` zu filtern.
- **`teas t INNER JOIN dup_rows d`** verwendet einen `INNER JOIN`, der es Ihnen erlaubt, die `teas`-Tabelle und die `dup_rows`-Abfrage zusammenzuführen.

Potenzielle Erweiterungen

Um bestimmte Zeilen aus einer Tabelle zu löschen, verwenden Sie eine `DELETE`-Anweisung. Näheres finden Sie in Kapitel 5.

Zeilen mit einem Maximalwert für eine andere Spalte auswählen

Die folgende Tabelle listet Angestellte und die von ihnen ausgeführten Verkäufe auf. Sie wollen für jeden Angestellten die neueste Anzahl an Verkäufen zurückliefern. Dieser Wert ist jeweils fett gedruckt.

```
SELECT * FROM sales;

+------+----------+------------+-------+
| id   | employee | date       | sales |
+------+----------+------------+-------+
|    1 | Emma     | 2021-08-01 |     6 |
|    2 | Emma     | 2021-08-02 |    17 |
|    3 | Jack     | 2021-08-02 |    14 |
|    4 | Emma     | 2021-08-04 |    20 |
|    5 | Jack     | 2021-08-05 |     5 |
|    6 | Emma     | 2021-08-07 |     1 |
+------+----------+------------+-------+
```

Lösung

Die folgende Abfrage liefert die Anzahl der Verkäufe zurück, die jeder Angestellte zum neuesten Verkaufsdatum ausgeführt hat (d.h. zum größten Datumswert jedes Verkäufers).

```
SELECT s.id, r.employee, r.recent_date, s.sales
FROM (SELECT employee, MAX(date) AS recent_date
      FROM sales
      GROUP BY employee) r
INNER JOIN sales s
           ON r.employee = s.employee
           AND r.recent_date = s.date;

+------+----------+-------------+-------+
| id   | employee | recent_date | sales |
+------+----------+-------------+-------+
|    5 | Jack     | 2021-08-05  |     5 |
|    6 | Emma     | 2021-08-07  |     1 |
+------+----------+-------------+-------+
```

Erklärung

Der Schlüssel zu diesem Problem liegt darin, es in zwei Teile zu zerlegen. Das erste Ziel ist, für jeden Angestellten das neueste Verkaufsdatum zu identifizieren. So sieht die Ausgabe der Unterabfrage `r` aus:

```
+----------+-------------+
| employee | recent_date |
+----------+-------------+
| Emma     | 2021-08-07  |
| Jack     | 2021-08-05  |
+----------+-------------+
```

Das zweite Ziel ist, die Spalten `id` und `sales` zurück in das Endergebnis zu ziehen. Das wird durch den `JOIN` in der zweiten Hälfte der Abfrage erledigt.

Schlüsselwörter in der Abfrage

- **`GROUP BY employee`** verwendet die `GROUP BY`-Klausel, die die Tabelle anhand von `employee` zerlegt und für jeden Angestellten **`MAX(date)`** sucht.
- **`r INNER JOIN sales s`** benutzt einen `INNER JOIN`, der es Ihnen erlaubt, die Unterabfrage `r` und die `sales`-Tabelle zusammenzuführen.

Potenzielle Erweiterungen

Eine Alternative zur GROUP BY-Lösung besteht darin, eine Fensterfunktion (OVER ... PARTITION BY ...) mit einer FIRST_VALUE-Funktion zu verwenden, die die gleichen Ergebnisse zurückliefern würde. Näheres finden Sie im Abschnitt »Fensterfunktionen« auf Seite 243 in Kapitel 8.

Text aus mehreren Feldern in einem einzigen Feld verketten

Dieser Abschnitt behandelt zwei unterschiedliche Szenarien:

- Text aus Feldern *in einer einzigen Zeile* in einem einzigen Wert verketten
- Text aus Feldern *in mehreren Zeilen* in einem einzigen Wert verketten

Text aus Feldern in einer einzigen Zeile verketten

Die folgende Tabelle enthält zwei Spalten, und Sie wollen sie in einer einzigen Spalte verketten.

```
+----+---------+        +-----------+
| id | name    |        | id_name   |
+----+---------+        +-----------+
|  1 | Boots   |  --->  | 1_Boots   |
|  2 | Pumpkin |        | 2_Pumpkin |
|  3 | Tiger   |        | 3_Tiger   |
+----+---------+        +-----------+
```

Verwenden Sie die CONCAT-Funktion oder den Verkettungsoperator (||), um die Werte zusammenzubringen:

```
-- MySQL, PostgreSQL und SQL Server
SELECT CONCAT(id, '_', name) AS id_name
FROM my_table;

-- Oracle, PostgreSQL und SQLite
SELECT id || '_' || name AS id_name
FROM my_table;
```

```
+-----------+
| id_name   |
+-----------+
| 1_Boots   |
| 2_Pumpkin |
| 3_Tiger   |
+-----------+
```

Potenzielle Erweiterungen

Kapitel 7, behandelt weitere Möglichkeiten zum Arbeiten mit String-Werten zusätzlich zu `CONCAT`, darunter:

- Ermitteln der Länge eines Strings
- Suchen von Wörtern in einem String
- Extrahieren von Text aus einem String

Text aus Feldern in mehreren Zeilen verketten

Die folgende Tabelle listet auf, wie viele Kalorien jede Person verbraucht hat. Sie wollen die Kalorien jeder Person in einer einzigen Zeile verketten.

```
+------+----------+      +------+----------+
| name | calories |      | name | calories |
+------+----------+      +------+----------+
| ally |       80 | ---> | ally | 80,75,90 |
| ally |       75 |      | jess | 100,92   |
| ally |       90 |      +------+----------+
| jess |      100 |
| jess |       92 |
+------+----------+
```

Verwenden Sie eine Funktion wie `GROUP_CONCAT`, `LISTAGG`, `ARRAY_AGG` oder `STRING_AGG`, um die Liste anzulegen.

```
SELECT name,
       GROUP_CONCAT(calories) AS calories_list
FROM workouts
GROUP BY name;

+------+---------------+
| name | calories_list |
+------+---------------+
```

```
| ally | 80,75,90      |
| jess | 100,92        |
+------+---------------+
```

Dieser Code funktioniert in *MySQL* und *SQLite*. Ersetzen Sie in den anderen RDBMS `GROUP_CONCAT(calories)` durch Folgendes:

Oracle
```
LISTAGG(calories, ',')
```

PostgreSQL
```
ARRAY_AGG(calories)
```

SQL Server
```
STRING_AGG(calories, ',')
```

Potenzielle Erweiterungen

Der Abschnitt »Zeilen in einem einzigen Wert oder einer einzigen Liste zusammenfassen« in Kapitel 8 bietet nähere Informationen dazu, wie Sie andere Trennzeichen neben dem Komma (`,`) benutzen, wie Sie die Werte sortieren und wie Sie eindeutige Werte zurückliefern können.

Alle Tabellen finden, die einen bestimmten Spaltennamen enthalten

Stellen Sie sich vor, Sie hätten eine Datenbank mit vielen Tabellen. Sie wollen schnell alle Tabellen finden, die einen Spaltennamen mit dem Wort `city` darin enthalten.

Lösung

In den meisten RDBMS gibt es eine besondere Tabelle, die alle Tabellen- und Spaltennamen enthält. Tabelle 10-1 zeigt, wie Sie in den einzelnen RDBMS diese Tabelle abfragen.

Die letzte Zeile jedes Codeblocks ist optional. Sie können sie einfügen, wenn Sie die Ergebnisse auf eine bestimmte Datenbank oder einen Benutzer eingrenzen wollen. Wenn Sie diese Zeile weglassen, werden alle Tabellen zurückgeliefert.

Tabelle 10-1: Alle Tabellen finden, die einen bestimmten Spaltennamen enthalten

RDBMS	Code

MySQL	`SELECT table_name, column_name`
	`FROM information_schema.columns`
	`WHERE column_name LIKE '%city%'`
	`  AND table_schema = 'my_db_name';`

Oracle	`SELECT table_name, column_name`
	`FROM all_tab_columns`
	`WHERE column_name LIKE '%CITY%'`
	`  AND owner = 'MY_USER_NAME';`

PostgreSQL, SQL Server	`SELECT table_name, column_name`
	`FROM information_schema.columns`
	`WHERE column_name LIKE '%city%'`
	`  AND table_catalog = 'my_db_name';`

Die Ausgabe zeigt alle Spaltennamen an, die den Begriff `city` enthalten, und gibt auch die Tabellen an, in denen sie sich befinden:

```
+------------+-------------+
| TABLE_NAME | COLUMN_NAME |
+------------+-------------+
| customers  | city        |
| employees  | city        |
| locations  | metro_city  |
+------------+-------------+
```

SQLite besitzt keine Tabelle, die alle Spaltennamen enthält. Stattdessen können Sie manuell alle Tabellen anzeigen lassen und dann die Spaltennamen in jeder Tabelle zeigen:

```
.tables
pragma table_info(my_table);
```

Potenzielle Erweiterungen

Kapitel 5, behandelt weitere Möglichkeiten, mit Datenbanken und Tabellen zu interagieren, einschließlich:

- Betrachten existierender Datenbanken

- Betrachten existierender Tabellen
- Betrachten der Spalten einer Tabelle

Kapitel 7, betrachtet weitere Möglichkeiten neben `LIKE`, nach Text zu suchen, darunter:

- = für die Suche nach einem exakten Treffer
- `IN` für die Suche nach mehreren Begriffen
- Regular Expressions für die Suche nach einem Muster

Eine Tabelle aktualisieren, deren ID einer anderen Tabelle entspricht

Stellen Sie sich vor, Sie hätten zwei Tabellen: `products` und `deals`. Sie wollen die Namen in der `deals`-Tabelle mit den Namen der Objekte in der `products`-Tabelle aktualisieren, die eine passende `id` haben.

```
SELECT * FROM products;

+------+--------------------+
| id   | name               |
+------+--------------------+
|  101 | Mac and cheese mix |
|  102 | MIDI keyboard      |
|  103 | Mother's day card  |
+------+--------------------+

SELECT * FROM deals;

+------+--------------+
| id   | name         |
+------+--------------+
|  102 | Tech gift    | --> MIDI keyboard
|  103 | Holiday card | --> Mother's day card
+------+--------------+
```

Lösung

Nutzen Sie eine `UPDATE`-Anweisung, um Werte in einer Tabelle mittels der `UPDATE ... SET ...`-Syntax zu modifizieren. Tabelle 10-2 zeigt, wie Sie dies in den einzelnen RDBMS erledigen.