

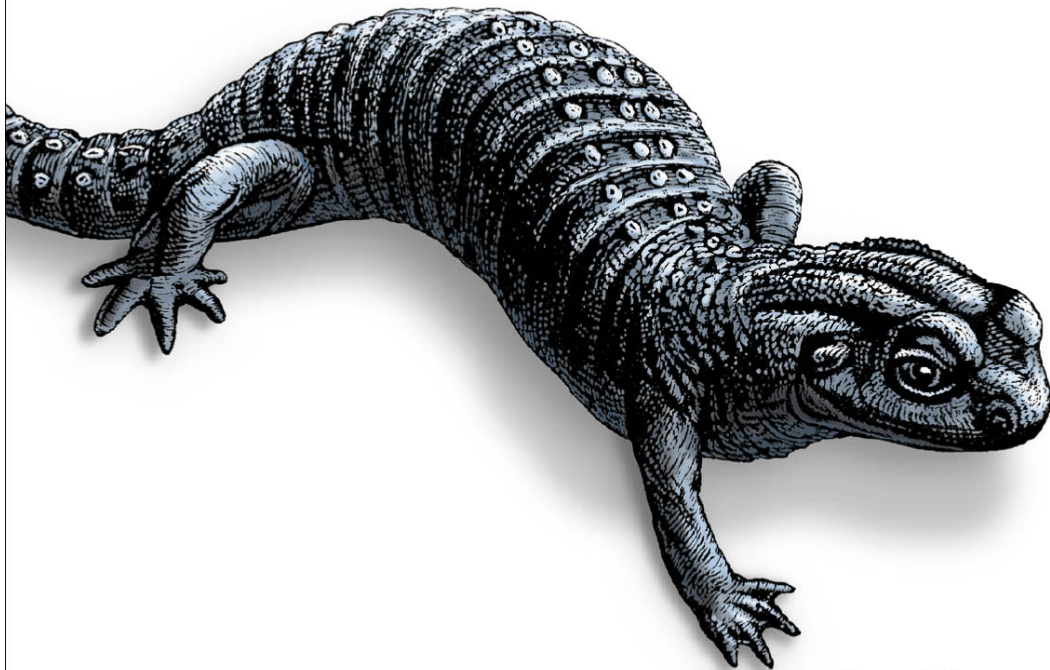
O'REILLY®

Erweiterte
Neuaufgabe
Jetzt zu MySQL, Oracle, SQLite
PostgreSQL, SQL Server

SQL

kurz & gut

O'Reillys Taschenbibliothek



Alice Zhao

Übersetzung von Kathrin Lichtenberg

Inhalt

Cover

Titel

Impressum

Inhalt

Vorwort

1 SQL-Crashkurs

Was ist eine Datenbank?

SQL

NoSQL

Datenbankmanagementsysteme (DBMS)

Eine SQL-Abfrage

SQL-Anweisungen

SQL-Abfragen

Die SELECT-Anweisung

Reihenfolge der Ausführung

Ein Datenmodell

2 Wo kann ich SQL-Code schreiben?

RDBMS-Software

Welches RDBMS sollte man wählen?

Was ist ein Terminal-Fenster?

SQLite

MySQL

Oracle

PostgreSQL

SQL Server

Datenbankwerkzeuge

Ein Datenbankwerkzeug mit einer Datenbank verbinden

Andere Programmiersprachen

Python mit einer Datenbank verbinden

R mit einer Datenbank verbinden

3 Die Sprache SQL

Vergleich mit anderen Sprachen

ANSI-Standards

SQL-Begriffe

Schlüsselwörter und Funktionen

Bezeichner und Aliasse

Anweisungen und Klauseln

Ausdrücke und Prädikate

Kommentare, Anführungszeichen und Whitespace

Teilsprachen

4 SQL-Abfragen: Die Grundlagen

Die SELECT-Klausel

Spalten auswählen

Alle Spalten auswählen

Ausdrücke auswählen

Funktionen auswählen

Aliasse für Spalten

Spalten qualifizieren

Unterabfragen auswählen

DISTINCT

Die FROM-Klausel

Aus mehreren Tabellen

Aus Unterabfragen

Wieso sollte man eine Unterabfrage in der FROM-Klausel benutzen?

Die WHERE-Klausel

Mehrere Prädikate

Auf Unterabfragen filtern

Die GROUP BY-Klausel

Die HAVING-Klausel

Die ORDER BY-Klausel

Die LIMIT-Klausel

5 Erzeugen, Aktualisieren und Löschen

Datenbanken

Datenmodell versus Schema

Namen vorhandener Datenbanken anzeigen

Namen der aktuellen Datenbank anzeigen

Zu einer anderen Datenbank wechseln

Eine Datenbank erzeugen

Eine Datenbank löschen

Tabellen anlegen

Eine einfache Tabelle anlegen

Namen vorhandener Tabellen anzeigen

Erzeugen einer Tabelle, die noch nicht existiert

Eine Tabelle mit Constraints erzeugen

Eine Tabelle mit Primär- und Fremdschlüsseln erzeugen

Eine Tabelle mit einem automatisch generierten Feld erzeugen

Die Ergebnisse einer Abfrage in eine Tabelle einfügen

Daten aus einer Textdatei in eine Tabelle einfügen

Tabellen modifizieren

Eine Tabelle oder Spalte umbenennen

Spalten anzeigen, hinzufügen und löschen

Zeilen anzeigen, hinzufügen und löschen

Constraints anzeigen, hinzufügen und modifizieren

Eine Spalte mit Daten aktualisieren

Eine Zeile mit Daten aktualisieren

Datenzeilen mit den Ergebnissen einer Abfrage aktualisieren

Eine Tabelle löschen

Indizes

Vergleich eines Buchindex mit einem SQL-Index

Einen Index erzeugen, um Abfragen zu beschleunigen

Views

Einen View erzeugen, um die Ergebnisse einer Abfrage zu speichern

Transaktionsmanagement

Prüfen Sie vor einem COMMIT genau

Änderungen mit einem ROLLBACK widerrufen

6 Datentypen

Wie Sie einen Datentyp wählen

Numerische Daten

Numerische Werte

Integer-Datentypen

Dezimal-Datentypen

Gleitkommatypen

String-Daten

String-Werte

Zeichentypen

Unicode-Datentypen

Datum/Zeit-Daten

Datum/Zeit-Werte

Datum/Zeit-Datentypen

Andere Daten

Boolesche Daten

Externe Dateien (Bilder, Dokumente usw.)

7 Operatoren und Funktionen

Operatoren

Logikoperatoren

Vergleichsoperatoren

Mathematische Operatoren

Aggregatfunktionen

Numerische Funktionen

Mathematische Funktionen anwenden

Generieren von Zufallszahlen

Zahlen runden und trunkieren

Daten in einen numerischen Datentyp konvertieren

String-Funktionen

Die Länge eines Strings ermitteln

Die Groß- und Kleinschreibung eines Strings ändern

Unerwünschte Zeichen rund um einen String wegschneiden

Strings verketteten

In einem String nach Text suchen

Einen Teil eines Strings extrahieren

Text in einem String ersetzen

Text aus einem String löschen

Reguläre Ausdrücke verwenden

Daten in einen String-Datentyp konvertieren

Datum/Zeit-Funktionen

Das aktuelle Datum oder die aktuelle Zeit zurückgeben

Ein Datums- oder Zeitintervall addieren oder subtrahieren

Die Differenz zwischen zwei Datums- oder Zeitangaben ermitteln

Einen Teil eines Datums oder einer Zeit extrahieren

Den Wochentag eines Datums ermitteln

Ein Datum auf die nächstgelegene Zeiteinheit runden

Einen String in einen Datum/Zeit-Datentyp konvertieren

Null-Funktionen

Einen alternativen Wert zurückgeben, wenn es einen Null-Wert gibt

8 Erweiterte Abfragekonzepte

Case-Anweisungen

Werte basierend auf einer If-Then-Logik für eine einzelne Spalte anzeigen

Werte basierend auf einer If-Then-Logik für mehrere Spalten anzeigen

Gruppieren und Zusammenfassen

GROUP BY-Grundlagen

Zeilen in einem einzigen Wert oder einer einzigen Liste zusammenfassen

ROLLUP, CUBE und GROUPING SETS

Fensterfunktionen

Aggregatfunktion

Fensterfunktion

Die Zeilen in einer Tabelle ordnen

Den ersten Wert in jeder Gruppe zurückliefern

Den zweiten Wert aus jeder Gruppe zurückliefern

Die ersten zwei Werte aus jeder Gruppe zurückliefern

Den Wert der vorhergehenden Zeile zurückliefern

Den gleitenden Durchschnitt berechnen

Die laufende Gesamtsumme berechnen

Pivoting und Unpivoting

Die Werte einer Spalte in mehrere Spalten zerlegen

Die Werte mehrerer Spalten in einer einzigen Spalte auflisten

9 Arbeiten mit mehreren Tabellen und Abfragen

Tabellen mit Joins zusammenbringen

Join-Grundlagen und INNER JOIN

LEFT JOIN, RIGHT JOIN und FULL OUTER JOIN

USING und NATURAL JOIN

CROSS JOIN und Self Join

Vereinigungsoperatoren

UNION

EXCEPT und INTERSECT

Common Table Expressions

CTEs versus Unterabfragen

Rekursive CTEs

10 Wie mache ich ...?

Zeilen finden, die doppelte Werte enthalten

Alle einmaligen Kombinationen zurückliefern

Nur die Zeilen mit doppelt vorhandenen Werten zurückliefern

Zeilen mit einem Maximalwert für eine andere Spalte auswählen

Text aus mehreren Feldern in einem einzigen Feld verketteten

Text aus Feldern in einer einzigen Zeile verketteten

Text aus Feldern in mehreren Zeilen verketteten

Alle Tabellen finden, die einen bestimmten Spaltennamen enthalten

Eine Tabelle aktualisieren, deren ID einer anderen Tabelle entspricht

Index

Über die Autoren

Kolophon

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 Schema namens 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 waterfall 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.

```
SELECT o.type, w.open_to_public
```

```
FROM owner o
```

```
JOIN waterfall w ON o.id = w.owner_id;
```

oder:

```

SELECT ALL o.type, w.open_to_public

FROM owner o

JOIN waterfall w ON o.id = w.owner_id;

```

```

type      open_to_public

```

```

---- -----

```

```

public    y

```

```

public    y

```

```

public    y

```

```

private   y

```

```

private   y

```

```

private   y

```

```

private   y

```

```

public    y

```

Falls Sie doppelt auftretende Zeilen aus den Ergebnissen entfernen wollen, können Sie das Schlüsselwort **DISTINCT** benutzen. Die folgende Abfrage liefert eine Liste der eindeutigen `type/open_to_public`-Kombinationen zurück.

```

SELECT DISTINCT o.type, w.open_to_public

```

```

FROM owner o

```

```
JOIN waterfall w ON o.id = w.owner_id;
```

```
type      open_to_public
```

```
---- -----
```

```
public    y
```

```
private   y
```

COUNT und DISTINCT

Um die Anzahl der eindeutigen Werte in einer *einzig*en Spalte zu zählen, können Sie die Schlüsselwörter **COUNT** und **DISTINCT** innerhalb der **SELECT**-Klausel kombinieren. Die folgende Abfrage liefert die Anzahl der eindeutigen **type**-Werte zurück.

```
SELECT COUNT(DISTINCT type) AS unique
```

```
FROM owner;
```

```
unique
```

```
---
```

```
2
```

Um die Anzahl der eindeutigen Kombinationen aus *mehreren Spalten* zu zählen, können Sie eine **DISTINCT**-Abfrage als Unterabfrage verpacken und dann einen **COUNT** auf der Unterabfrage ausführen. Die folgende Abfrage liefert die Anzahl der eindeutigen **type/open_to_public**-Kombinationen zurück.

```
SELECT COUNT(*) AS num_unique
```

```
FROM (SELECT DISTINCT o.type, w.open_to_public
```

```
FROM owner o JOIN waterfall w
```

```
ON o.id = w.owner_id) my_subquery;
```

```
num_unique
```

```
-----
```

```
2
```

MySQL und *PostgreSQL* unterstützen die Verwendung der `COUNT (DISTINCT)`-Syntax auf mehreren Spalten. Die folgenden zwei Abfragen sind äquivalent zur gezeigten Abfrage, brauchen aber keine Unterabfrage:

– MySQL-Äquivalent

```
SELECT COUNT(DISTINCT o.type, w.open_to_public)
```

```
AS num_unique
```

```
FROM owner o JOIN waterfall w
```

```
ON o.id = w.owner_id;
```

– PostgreSQL-Äquivalent

```
SELECT COUNT(DISTINCT (o.type, w.open_to_public))
```

```
AS num_unique
```

```
FROM owner o JOIN waterfall w
```

```
ON o.id = w.owner_id;
```

num_unique

2

Die FROM-Klausel

Die FROM-Klausel wird verwendet, um die Quelle der Daten festzulegen, die Sie beziehen wollen. Der einfachste Fall ist das Benennen einer einzelnen Tabelle oder eines einzelnen View in der FROM-Klausel der Abfrage.

```
SELECT name
```

```
FROM waterfall;
```

Sie können eine Tabelle oder einen View mittels der Punktnotation entweder mit einem Datenbank- oder einem Schema namen qualifizieren. Die folgende Abfrage bezieht Daten speziell aus der `waterfall`-Tabelle im `sqlbook`-Schema:

```
SELECT name
```

```
FROM sqlbook.waterfall;
```

Aus mehreren Tabellen

Anstatt uns nur auf eine Tabelle zu beschränken, wollen Sie oft Daten aus mehreren Tabellen holen. Die gebräuchlichste Methode hierfür ist die Verwendung einer JOIN-Klausel innerhalb der FROM-Klausel. Die folgende Abfrage bezieht Daten aus den Tabellen `waterfall` und `tour` und zeigt sie in einer einzigen Ergebnistabelle an.

```

SELECT *
FROM waterfall w JOIN tour t
    ON w.id = t.stop;

```

id	name	... name	stop	...
1	Munising Falls	M-28	1	
1	Munising Falls	Munising	1	
2	Tannery Falls	Munising	2	
3	Alger Falls	M-28	3	
3	Alger Falls	Munising	3	
...				

Schauen wir uns die einzelnen Teile des Codeblocks genauer an.

Tabellenaliasse

```

waterfall w JOIN tour t

```

Den Tabellen `waterfall` und `tour` werden die Tabellenaliasse `w` und `t` gegeben. Dies sind temporäre Namen für die Tabellen in der Abfrage. Tabellenaliasse sind in einer JOIN-Klausel nicht erforderlich, helfen aber dabei, Tabellennamen abzukürzen, die in den ON- und SELECT-Klauseln referenziert werden müssen.

JOIN ... ON ...

```
waterfall w JOIN tour t
```

```
ON w.id = t.stop
```

Diese zwei Tabellen werden mit dem Schlüsselwort JOIN zusammengezogen. Einer JOIN-Klausel folgt immer eine ON-Klausel, die festlegt, wie die Tabellen miteinander verknüpft werden sollen. In diesem Fall muss die `id` des Wasserfalls in der Tabelle `waterfall` dem `stop` bei dem Wasserfall in der `tour`-Tabelle entsprechen.



Es kann sein, dass die FROM-, JOIN- und ON-Klauseln auf unterschiedlichen Zeilen stehen oder eingerückt sind. Das ist nicht erforderlich, verbessert aber die Lesbarkeit – vor allem wenn Sie viele Tabellen miteinander verbinden.

Ergebnistabelle

Das Ergebnis einer Abfrage kommt immer in einer einzigen Tabelle. Die Tabelle `waterfall` hat zwölf Spalten, und die Tabelle `tour` hat drei Spalten. Nachdem diese Tabellen zusammengefasst wurden, hat die Ergebnistabelle 15 Spalten.

id	name	... name	stop ...
1	Munising Falls	M-28	1
1	Munising Falls	Munising	1
2	Tannery Falls	Munising	2

3 Alger Falls	M-28	3
---------------	------	---

3 Alger Falls	Munising	3
---------------	----------	---

...

Sie sehen vermutlich, dass es in der Ergebnistabelle zwei Spalten mit dem Namen `name` gibt. Die erste stammt aus der `waterfall`-Tabelle, die zweite aus der `tour`-Tabelle. Um sie in der `SELECT`-Klausel referenzieren zu können, müssen Sie die Spaltennamen qualifizieren.

```
SELECT w.name, t.name
```

```
FROM waterfall w JOIN tour t
```

```
ON w.id = t.stop;
```

```
name          name
```

```
-----
```

```
Munising Falls M-28
```

```
Munising Falls Munising
```

```
Tannery Falls  Munising
```

...

Damit Sie die beiden Spalten unterscheiden können, sollten Sie für die Spaltennamen auch noch Aliasse vergeben.

```
SELECT w.name AS waterfall_name,
```

```

t.name AS tour_name

FROM waterfall w JOIN tour t

ON w.id = t.stop;

waterfall_name  tour_name

```

```

Munising Falls  M-28

Munising Falls  Munising

Tannery Falls   Munising

Alger Falls     M-28

Alger Falls     Munising

...

```

JOIN-Variationen

Falls in dem vorhergehenden Beispiel ein Wasserfall nicht in einer der Touren aufgeführt ist, erscheint er auch nicht in der Ergebnistabelle. Um alle Wasserfälle in den Ergebnissen zu erhalten, müssen Sie eine andere Art von Join verwenden.

JOIN ist standardmäßig ein INNER JOIN

Dieses Beispiel verwendet das einfache Schlüsselwort JOIN, um die Daten aus den zwei Tabellen zusammenzuziehen. Besser ist es jedoch, ausdrücklich die Art von Join anzugeben, die Sie benutzen. JOIN allein bedeutet standardmäßig immer ein INNER JOIN. Das bedeutet, dass nur Einträge, die in beiden Tabellen stehen, in den Ergebnissen zurückgeliefert werden.

In SQL kommt eine Vielzahl an Join-Typen zum Einsatz. Diese werden in »Tabellen mit Joins zusammenbringen« auf Seite 262 in Kapitel 9 ausführlicher

behandelt.

Aus Unterabfragen

Eine Unterabfrage ist eine Abfrage, die in eine andere Abfrage geschachtelt ist. Unterabfragen innerhalb der FROM-Klausel sollten unabhängige SELECT-Anweisungen sein. Das bedeutet, dass sie sich überhaupt nicht auf die äußere Abfrage beziehen und allein ausgeführt werden können.



Eine Unterabfrage innerhalb der FROM-Klausel ist auch als *abgeleitete Tabelle* bekannt, weil die Unterabfrage im Prinzip für die Dauer der Abfrage wie eine Tabelle agiert.

Die folgende Abfrage listet alle Wasserfälle auf, die sich in öffentlicher Hand befinden. Der Unterabfrageteil ist fett gedruckt.

```
SELECT w.name AS waterfall_name,  
  
       o.name AS owner_name  
  
FROM (SELECT * FROM owner WHERE type = 'public') o  
  
     JOIN waterfall w  
  
     ON o.id = w.owner_id;  
  
waterfall_name  owner_name  
-----  
  
Little Miners   Pictured Rocks  
  
Miners Falls   Pictured Rocks
```

Munising Falls Pictured Rocks

Wagner Falls MI DNR

Wichtig ist es, die Reihenfolge zu verstehen, in der die Abfrage ausgeführt wird.

Schritt 1: Ausführen der Unterabfrage

Der Inhalt der Unterabfrage wird zuerst ausgeführt. Sie können sehen, dass dies eine Tabelle ergibt, in der nur die staatlichen Besitzer stehen:

```
SELECT * FROM owner WHERE type = 'public';
```

id	name	phone	type
1	Pictured Rocks	906.387.2607	public
4	MI DNR	906.228.6561	public

Wenn Sie sich noch einmal die Originalabfrage anschauen, bemerken Sie, dass auf die Unterabfrage sofort der Buchstabe o folgt. Dies ist der temporäre Name oder Alias, den wir den Ergebnissen der Unterabfrage zuweisen.



Aliase sind für Unterabfragen innerhalb der FROM-Klausel in *MySQL*, *PostgreSQL* und *SQL Server* notwendig, nicht jedoch in *Oracle* und *SQLite*.

Schritt 2: Ausführen der gesamten Abfrage

Als Nächstes können Sie sich das so vorstellen, dass der Buchstabe o den Platz der Unterabfrage einnimmt. Die Abfrage wird nun wie üblich ausgeführt.

```
SELECT w.name AS waterfall_name,
```

```
       o.name AS owner_name
```

```
FROM o JOIN waterfall w
```

```
     ON o.id = w.owner_id;
```

```
waterfall_name  owner_name
```

```
-----
```

```
Little Miners   Pictured Rocks
```

```
Miners Falls   Pictured Rocks
```

```
Munising Falls Pictured Rocks
```

```
Wagner Falls   MI DNR
```

Unterabfragen versus WITH-Klausel

Eine Alternative zum Schreiben einer Unterabfrage ist es, stattdessen einen allgemeinen Tabellenausdruck (*Common Table Expression*, CTE) mittels einer `WITH`-Klausel zu schreiben. Der Vorteil der `WITH`-Klausel besteht darin, dass die Unterabfrage gleich zu Anfang genannt wird, was saubereren Code ergibt und außerdem die Möglichkeit bietet, die Unterabfrage mehrmals zu referenzieren.

```
WITH o AS (SELECT * FROM owner

           WHERE type = 'public')

SELECT w.name AS waterfall_name,

       o.name AS owner_name

FROM o JOIN waterfall w

     ON o.id = w.owner_id;
```

Die `WITH`-Klausel wird von *MySQL 8.0+* (ab 2018), *PostgreSQL*, *Oracle*, *SQL Server* und *SQLite* unterstützt. »Common Table Expressions« auf Seite 282 in Kapitel 9 bietet weitere Beispiele für diese Technik.

Wieso sollte man eine Unterabfrage in der FROM-Klausel benutzen?

Der größte Vorteil der Verwendung von Unterabfragen besteht darin, dass Sie ein großes Problem in kleinere Probleme verwandeln können. Hier sind zwei Beispiele:

Beispiel 1: Mehrere Schritte, um Ergebnisse zu erhalten

Nehmen wir einmal an, Sie wollten die durchschnittliche Anzahl der Halts einer Tour ermitteln. Zuerst müssten Sie feststellen, wie oft die einzelnen Touren jeweils einen Halt einlegen. Anschließend berechnen Sie den Durchschnitt der Ergebnisse.

Die folgende Abfrage ermittelt die Anzahl der Halts der einzelnen Touren:

```
SELECT name, MAX(stop) as num_stops
```

FROM tour

GROUP BY name;

name	num_stops
------	-----------

M-28	11
------	----

Munising	6
----------	---

US-2	14
------	----

Sie könnten dann die Abfrage in eine Unterabfrage umwandeln und eine weitere Abfrage um diese herum schreiben, um den Durchschnitt zu ermitteln:

```
SELECT AVG(num_stops) FROM
```

```
(SELECT name, MAX(stop) as num_stops
```

```
FROM tour
```

```
GROUP BY name) tour_stops;
```

```
AVG(num_stops)
```

```
10.33333333333333
```

Beispiel 2: Tabelle in FROM-Klausel ist zu groß

Das ursprüngliche Ziel bestand darin, alle Wasserfälle aufzulisten, die in öffentlicher Hand sind. Das lässt sich tatsächlich ohne Unterabfrage und stattdessen mit einem Join erledigen:

```
SELECT w.name AS waterfall_name,
       o.name AS owner_name
FROM   owner o
       JOIN waterfall w ON o.id = w.owner_id
WHERE  o.type = 'public';

waterfall_name  owner_name
-----
Little Miners   Pictured Rocks
Miners Falls    Pictured Rocks
Munising Falls  Pictured Rocks
Wagner Falls    MI DNR
```

Nehmen wir einmal an, dass die Abfrage sehr lange für ihre Verarbeitung braucht. Das kann passieren, wenn Sie riesige Tabellen zusammenführen (wir sprechen hier von Tabellen mit einer Zeilenanzahl im zweistelligen Millionenbereich). Es gibt mehrere Möglichkeiten, die Abfrage so umzuschreiben, dass sie schneller läuft. Eine dieser Möglichkeiten ist eine Unterabfrage.

Da wir nur an staatlichen Eigentümern interessiert sind, können wir zuerst eine Unterabfrage schreiben, die alle Privatbesitzer herausfiltert. Die kleinere

owner-Tabelle würde dann mit der waterfall-Tabelle zusammengeführt werden, was weniger Zeit beansprucht und dieselben Ergebnisse liefert.

```
SELECT w.name AS waterfall_name,  
  
       o.name AS owner_name  
  
FROM   (SELECT * FROM owner  
  
        WHERE type = 'public') o  
  
JOIN waterfall w ON o.id = w.owner_id;
```

```
waterfall_name  owner_name  
  
-----  
  
Little Miners   Pictured Rocks  
  
Miners Falls   Pictured Rocks  
  
Munising Falls Pictured Rocks  
  
Wagner Falls   MI DNR
```

Das sind nur zwei Beispiele dafür, wie man eine größere Abfrage mithilfe von Unterabfragen in kleinere Schritte zerlegen kann.

Die WHERE-Klausel

Die WHERE-Klausel dient dazu, die Abfrageergebnisse auf die Zeilen zu beschränken, die uns wirklich interessieren. Oder einfacher gesagt: Dies ist die Stelle, an der wir Daten filtern können. Sie wollen nur ganz selten alle Zeilen einer Tabelle anzeigen. Meist reichen Ihnen die Zeilen, die bestimmte Kriterien erfüllen.



Wenn Sie eine Tabelle mit Millionen von Zeilen untersuchen, sollten Sie niemals ein `SELECT * FROM my_table;` verwenden, weil das unnötig lange für die Ausführung braucht.

Stattdessen ist es eine gute Idee, die Daten vorher zu filtern. Zwei Methoden sind hier gebräuchlich:

Filtern anhand einer Spalte in der WHERE-Klausel

Besser noch, filtern Sie anhand einer Spalte, die bereits indiziert wurde, um die Datenabfrage noch mehr zu beschleunigen.

```
SELECT *  
  
FROM my_table  
  
WHERE year_id = 2021;
```

*Anzeigen der obersten Zeilen der Daten mit der LIMIT-Klausel (oder WHERE ROWNUM <= 10 in Oracle oder SELECT TOP 10 * in SQL Server)*

```
SELECT *  
  
FROM my_table  
  
LIMIT 10;
```

Die folgende Abfrage sucht alle Wasserfälle, die nicht das Wort *Falls* im Namen haben. Mehr zum Schlüsselwort `LIKE` finden Sie in Kapitel 7.

```
SELECT id, name  
  
FROM waterfall  
  
WHERE name NOT LIKE '%Falls%';
```

```
id      name  
--  -----
```

```
7 Little Miners
```

```
14 Rapid River Fls
```

Der fett dargestellte Abschnitt wird oft als bedingte Anweisung oder Prädikat bezeichnet. Das Prädikat nimmt einen logischen Vergleich für jede Datenzeile vor, der TRUE/FALSE/UNKNOWN ergibt.

Die `waterfall`-Tabelle enthält 16 Zeilen. Für jede Zeile wird geprüft, ob der Wasserfallname *Falls* enthält. Ist *Falls* nicht enthalten, ist das `name NOT LIKE '%Falls%'`-Prädikat TRUE, und die Zeile wird in den Ergebnissen zurückgeliefert. Das war in den zwei gezeigten Zeilen der Fall.

Mehrere Prädikate

Es ist möglich, mehrere Prädikate mit *Operatoren* wie AND oder OR zu kombinieren. Das folgende Beispiel zeigt Wasserfälle ohne *Falls* in ihren Namen, die außerdem keinen Besitzer haben:

```
SELECT id, name

FROM waterfall

WHERE name NOT LIKE '%Falls%'

      AND owner_id IS NULL;
```

```
id      name
--  -----
```

```
14 Rapid River Fls
```

Näheres zu Operatoren finden Sie in Operatoren in Kapitel 7.

Auf Unterabfragen filtern

Eine Unterabfrage ist eine Abfrage, die in eine andere Abfrage geschachtelt ist. Häufig findet man sie im Zusammenhang mit der WHERE-Klausel. Das folgende Beispiel ruft öffentlich zugängliche Wasserfälle ab, die sich im County Alger befinden:

```
SELECT w.name

FROM   waterfall w

WHERE  w.open_to_public = 'y'

      AND w.county_id IN (

          SELECT c.id FROM county c

          WHERE c.name = 'Alger');
```

name

Munising Falls

Tannery Falls

Alger Falls

...



Anders als Unterabfragen in der SELECT-Klausel oder der FROM-Klausel erfordern Unterabfragen in der WHERE-Klausel keinen Alias. Tatsächlich erhalten Sie eine Fehlermeldung, falls Sie einen Alias einsetzen.

Warum sollte man eine Unterabfrage in der WHERE-Klausel benutzen?

Das ursprüngliche Ziel war es, die öffentlich zugänglichen Wasserfälle abzufragen, die im County Alger liegen. Würden Sie diese Abfrage von Grund auf neu schreiben, würden Sie vermutlich folgendermaßen starten:

```
SELECT w.name

FROM   waterfall w

WHERE  w.open_to_public = 'y';
```

An dieser Stelle haben Sie alle Wasserfälle, die öffentlich zugänglich sind. Nun müssen Sie versuchen, diejenigen zu finden, die speziell im County Alger liegen. Sie wissen, dass die `waterfall`-Tabelle keine Spalte mit County-Namen hat, die `county`-Tabelle dagegen schon.

Sie haben zwei Möglichkeiten, um den County-Namen in die Ergebnisse zu bekommen. Entweder Sie schreiben eine Unterabfrage in der WHERE-Klausel, die speziell die Alger-County-Informationen abrufen, oder Sie fassen die Tabellen `waterfall` und `county` zusammen:

– Unterabfrage in WHERE-Klausel

```
SELECT w.name

FROM   waterfall w

WHERE  w.open_to_public = 'y'
```

```
AND w.county_id IN (  
  
    SELECT c.id FROM county c  
  
    WHERE c.name = 'Alger');
```

oder:

```
– JOIN-Klausel
```

```
SELECT w.name
```

```
FROM waterfall w INNER JOIN county c
```

```
    ON w.county_id = c.id
```

```
WHERE w.open_to_public = 'y'
```

```
    AND c.name = 'Alger';
```

```
name
```

```
-----
```

```
Munising Falls
```

```
Tannery Falls
```

```
Alger Falls
```

```
...
```

Die zwei Abfragen liefern die gleichen Ergebnisse. Der Vorteil der ersten Herangehensweise besteht darin, dass Unterabfragen oft leichter zu verstehen sind als Joins. Die zweite Vorgehensweise hat dagegen den Vorteil, dass Joins üblicherweise schneller ausgeführt werden als Unterabfragen.

Funktionierend > Optimieren

Beim Schreiben von SQL-Code gibt es oft mehrere Möglichkeiten, das Gleiche zu erreichen.

Ihre oberste Priorität sollte es immer sein, *funktionierenden* Code zu schreiben. Es ist egal, wenn er sehr lange für die Ausführung braucht oder hässlich ist ... er funktioniert!

Falls Sie Zeit haben, besteht der nächste Schritt darin, den Code zu *optimieren*. Sie können seine Leistung verbessern, indem Sie ihn etwa mit einem Join umschreiben, oder Sie machen ihn mit Einrückungen und Großschreibungen lesbarer usw.

Setzen Sie sich nicht unter Druck, von Anfang an den absolut optimalen Code zu schreiben, sondern bemühen Sie sich zunächst um Code, der funktioniert. Eleganter Code kommt dann mit zunehmender Erfahrung.

Andere Möglichkeiten, Daten zu filtern

Die WHERE-Klausel ist nicht die einzige Stelle in einer SELECT-Anweisung zum Filtern von Datenzeilen.

- FROM-Klausel: Wenn man Tabellen miteinander verbindet, gibt die ON-Klausel an, wie sie verknüpft werden sollen. Hier können Sie Bedingungen festlegen, mit denen Sie die Datenzeilen einschränken, die durch die Abfrage zurückgeliefert werden (siehe »Tabellen mit Joins zusammenbringen« auf Seite 262 in Kapitel 9 für nähere Informationen).
- HAVING-Klausel: Wenn es Aggregationen in der SELECT-Anweisung gibt, dann geben Sie in der HAVING-Klausel an, wie die Aggregationen gefiltert werden sollen (siehe »Die HAVING-Klausel« auf Seite 92 für weitere Einzelheiten).
- LIMIT-Klausel: Um eine bestimmte Anzahl von Zeilen anzuzeigen, können Sie die LIMIT-Klausel verwenden. In *Oracle* wird dies mit WHERE ROWNUM erledigt und in *SQL Server* mit SELECT TOP (siehe »Die LIMIT-Klausel« auf Seite 97 in diesem Kapitel für weitere Einzelheiten).

Die GROUP BY-Klausel

Zweck der GROUP BY-Klausel ist es, Zeilen in Gruppen zu sammeln und die Zeilen innerhalb der Gruppen irgendwie zusammenzufassen, sodass am Ende

nur eine Zeile pro Gruppe zurückgegeben wird. Man bezeichnet das manchmal als »Slicing« der Zeilen und als »Rollup« der Zeilen in den einzelnen Gruppen.

Die folgende Abfrage ermittelt die Anzahl der Wasserfälle in den einzelnen Touren:

```
SELECT  t.name AS tour_name,
        COUNT(*) AS num_waterfalls
FROM    waterfall w INNER JOIN tour t
        ON w.id = t.stop
GROUP BY t.name;
```

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

Es gibt hier zwei Teile, die für uns hier interessant sind:

- *Das Sammeln der Zeilen*, das in der GROUP BY-Klausel erfolgt.
- *Das Zusammenfassen der Zeilen* innerhalb der Gruppen, das in der SELECT-Klausel festgelegt wird.

Schritt 1: Das Sammeln der Zeilen

In der GROUP BY-Klausel:

```
GROUP BY t.name
```

geben wir an, dass wir gern alle Datenzeilen anschauen und die Wasserfälle der M-28-Tour in eine Gruppe setzen würden, alle Wasserfälle der Munising-Tour in eine Gruppe und so weiter. Hinter den Kulissen werden die Daten folgendermaßen gruppiert:

```
tour_name  waterfall_name
```

```
-----
```

```
M-28      Munising Falls
```

```
M-28      Alger Falls
```

```
M-28      Scott Falls
```

```
M-28      Canyon Falls
```

```
M-28      Agate Falls
```

```
M-28      Bond Falls
```

```
Munising  Munising Falls
```

```
Munising  Tannery Falls
```

```
Munising  Alger Falls
```

```
Munising  Wagner Falls
```

```
Munising  Horseshoe Falls
```

```
Munising  Miners Falls
```

US-2	Bond Falls
US-2	Fumee Falls
US-2	Kakabika Falls
US-2	Rapid River Fls

Schritt 2: Das Zusammenfassen der Zeilen

In der SELECT-Klausel:

```
SELECT t.name AS tour_name,  
  
       COUNT(*) AS num_waterfalls
```

geben wir an, dass wir für jede Gruppe oder für jede Tour die Datenzeilen in der Gruppe zählen wollen. Da jede Zeile einen Wasserfall repräsentiert, würde dies die Gesamtanzahl der Wasserfälle in den jeweiligen Touren repräsentieren.

Die COUNT () -Funktion ist offiziell als *Aggregatfunktion* bekannt oder als eine Funktion, die viele Zeilen mit Daten in einem einzelnen Wert zusammenfasst. Weitere Aggregatfunktionen finden Sie in »Aggregatfunktionen« auf Seite 190 in Kapitel 7.



In diesem Beispiel liefert `COUNT(*)` die Anzahl der Wasserfälle entlang der einzelnen Touren zurück. Das funktioniert hier aber nur, weil jede Datenzeile in den Tabellen `waterfall` und `tour` einen einzelnen Wasserfall repräsentiert.

Wäre einer der Wasserfälle in mehreren Zeilen aufgeführt, würde `COUNT(*)` einen größeren Wert liefern als erwartet. In diesem Fall könnten Sie stattdessen möglicherweise `COUNT(DISTINCT waterfall_name)` verwenden, um die eindeutigen Wasserfälle zu finden. Näheres dazu erfahren Sie in `COUNT` und `DISTINCT`.

Die wichtigste Erkenntnis hier ist, dass Sie die Ergebnisse der Aggregatfunktion unbedingt noch einmal überprüfen müssen, um sicherzugehen, dass sie die Daten auch wirklich so zusammenfasst, wie Sie es vorgesehen haben.

Nachdem nun mit der `GROUP BY`-Klausel die Gruppen erzeugt wurden, wird die Aggregatfunktion einmal auf jede Gruppe angewendet:

```
tour_name  COUNT(*)
-----
M-28                6
M-28
M-28
M-28
M-28
M-28
Munising                6
Munising
```

Munising

Munising

Munising

Munising

US-2 4

US-2

US-2

US-2

Alle Spalten, auf die die Aggregatfunktion nicht angewendet wurde, in diesem Fall also die Spalte `tour_name`, werden in einem Wert zusammengefasst:

```
tour_name  COUNT(*)
```

```
----  ---
```

```
M-28      6
```

```
Munising  6
```

```
US-2      4
```



Dieses Zusammenfassen vieler Zeilen in einer Gesamtzeile bedeutet, dass die **SELECT**-Klausel beim Anwenden einer **GROUP BY**-Klausel *nur* dies enthalten sollte:

- Alle Spalten, die in der **GROUP BY**-Klausel aufgelistet sind: **t.name**
- Aggregationen: **COUNT(*)**

```
SELECT t.name AS tour_name,  
  
        COUNT(*) AS num_waterfalls  
  
...  
  
GROUP BY t.name;
```

Geschieht das nicht, erhalten Sie entweder eine Fehlermeldung, oder es werden unkorrekte Werte zurückgeliefert.

GROUP BY in der Praxis

Folgende Schritte sollten Sie unternehmen, wenn Sie ein **GROUP BY** nutzen:

1. Stellen Sie fest, welche Spalte(n) Sie verwenden wollen, um Ihre Daten (z. B. den Namen der Tour) herauszulösen oder zu gruppieren.
2. Stellen Sie fest, wie Sie die Daten in den einzelnen Zeilen zusammenfassen wollen (z. B. Zählen der Wasserfälle in den einzelnen Touren).

Wenn Sie dies entschieden haben:

1. Listen Sie in der **SELECT**-Klausel die Spalte(n) auf, anhand deren Sie gruppieren wollen (z. B. den Namen der Tour), und die Aggregation(en), die Sie in den einzelnen Gruppen berechnen wollen (z. B. Anzahl der Wasserfälle).
2. Listen Sie in der **GROUP BY**-Klausel alle Spalten auf, die keine Aggregationen sind (z. B. den Namen der Tour).

Komplexere Gruppierungssituationen, einschließlich **ROLLUP**, **CUBE** und **GROUPING SETS**, finden Sie in »Gruppieren und Zusammenfassen« auf Seite 235 in Kapitel 8.

Die HAVING-Klausel

Die **HAVING**-Klausel setzt Restriktionen auf die Zeilen, die von einer **GROUP BY**-Abfrage zurückgeliefert werden. Mit anderen Worten, sie erlaubt Ihnen, die

Ergebnisse zu filtern, nachdem ein GROUP BY angewandt wurde.



Eine HAVING-Klausel folgt immer unmittelbar auf eine GROUP BY-Klausel. Ohne GROUP BY-Klausel kann es keine HAVING-Klausel geben.

Dies ist eine Abfrage, die die Anzahl der Wasserfälle auf jeder Tour mittels einer GROUP BY-Klausel auflistet:

```
SELECT  t.name AS tour_name,
        COUNT(*) AS num_waterfalls
FROM    waterfall w INNER JOIN tour t
        ON w.id = t.stop
```

GROUP BY t.name;

```
tour_name  num_waterfalls
```

```
-----
```

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

Nehmen wir einmal an, wir wollten nur die Touren auflisten, die exakt sechs Halts haben. Dazu würden Sie eine HAVING-Klausel hinter der GROUP BY-

Klausel einfügen:

```
SELECT  t.name AS tour_name,  
        COUNT(*) AS num_waterfalls  
FROM    waterfall w INNER JOIN tour t  
        ON w.id = t.stop  
  
GROUP BY t.name  
  
HAVING  COUNT(*) = 6;
```

tour_name num_waterfalls

M-28 6

Munising 6

WHERE versus HAVING

Aufgabe beider Klauseln ist das Filtern von Daten. Falls Sie versuchen:

- auf bestimmten Spalten zu filtern, schreiben Sie Ihre Bedingung in die WHERE-Klausel.
- auf Aggregationen zu filtern, schreiben Sie Ihre Bedingungen in die HAVING-Klausel.

Der Inhalt von WHERE- und HAVING-Klauseln kann nicht vertauscht werden:

- Setzen Sie niemals eine Bedingung mit einer Aggregation in die WHERE-Klausel. Sie erhalten einen Fehler.
- Setzen Sie niemals eine Bedingung in die HAVING-Klausel, die keine Aggregation betrifft. Solche Bedingungen werden in der WHERE-Klausel viel effizienter ausgewertet.

Sie werden bemerken, dass sich die HAVING-Klausel auf die Aggregation COUNT(*) bezieht:

```
SELECT COUNT(*) AS num_waterfalls
```

...

```
HAVING COUNT(*) = 6;
```

und nicht den Alias:

```
# Code läuft nicht
```

```
SELECT COUNT(*) AS num_waterfalls
```

...

```
HAVING num_waterfalls = 6;
```

Das liegt an der Ausführungsreihenfolge der Klauseln. Die SELECT-Klausel steht vor der HAVING-Klausel. Allerdings wird die SELECT-Klausel tatsächlich *nach* der HAVING-Klausel ausgeführt.

Das bedeutet, dass der Alias `num_waterfalls` in der SELECT-Klausel zu dem Zeitpunkt, an dem die HAVING-Klausel ausgeführt wird, noch nicht existiert. Die HAVING-Klausel muss sich stattdessen auf die reine Aggregation `COUNT(*)` beziehen.



MySQL und SQLite sind Ausnahmen und erlauben Aliasse (num_waterfalls) in der HAVING-Klausel.

Die ORDER BY-Klausel

Die ORDER BY-Klausel wird verwendet, um anzugeben, wie die Ergebnisse einer Abfrage sortiert werden sollen.

Die folgende Abfrage liefert eine Liste mit Eigentümern und Wasserfällen zurück, die unsortiert ist:

```
SELECT COALESCE(o.name, 'Unknown') AS owner,  
  
       w.name AS waterfall_name  
  
FROM   waterfall w  
  
       LEFT JOIN owner o ON w.owner_id = o.id;
```

owner	waterfall_name
-----	-----
Pictured Rocks	Munising Falls
Michigan Nature	Tannery Falls
AF LLC	Alger Falls
MI DNR	Wagner Falls

Unknown

Horseshoe Falls

...

Die COALESCE-Funktion

Die COALESCE-Funktion ersetzt alle NULL-Werte in einer Spalte durch einen anderen Wert. In diesem Fall wandelte sie die NULL-Werte in der Spalte o.name in den Text Unknown um.

Würde die COALESCE-Funktion hier nicht verwendet werden, wären Wasserfälle ohne Besitzer nicht in den Ergebnissen aufgetaucht. Stattdessen steht bei ihnen nun, dass sie einen unbekanntem Besitzer haben, also Unknown. Sie können daher in die Ergebnisse einsortiert werden.

Mehr dazu finden Sie in Kapitel 7.

Die folgende Abfrage liefert die gleiche Liste zurück, nun jedoch zuerst alphabetisch nach dem Besitzer und dann nach dem Wasserfall sortiert:

```
SELECT    COALESCE(o.name, 'Unknown') AS owner,
          w.name AS waterfall_name
FROM      waterfall w
          LEFT JOIN owner o ON w.owner_id = o.id
```

```
ORDER BY owner, waterfall_name;
```

```
owner           waterfall_name
```

```
-----
```

```
AF LLC          Alger Falls
```

```
MI DNR          Wagner Falls
```

Michigan Nature Tannery Falls

Michigan Nature Twin Falls #1

Michigan Nature Twin Falls #2

...

Die Standardsortierung erfolgt in aufsteigender Reihenfolge, das heißt, Text verläuft von A nach Z, und Zahlen verlaufen von der niedrigsten zur höchsten. Sie können die Schlüsselwörter ASCENDING und DESCENDING (die als ASC und DESC abgekürzt werden können) verwenden, um die Sortierung auf den einzelnen Spalten zu kontrollieren.

Das Folgende ist eine Modifizierung der vorherigen Sortierung, allerdings werden dieses Mal die Namen der Besitzer in umgekehrter Reihenfolge sortiert:

```
SELECT COALESCE(o.name, 'Unknown') AS owner,
```

```
       w.name AS waterfall_name
```

...

```
ORDER BY owner DESC, waterfall_name ASC;
```

```
owner           waterfall_name
```

```
-----
```

```
Unknown        Agate Falls
```

```
Unknown        Bond Falls
```

```
Unknown        Canyon Falls
```

...

Sie können nach Spalten und Ausdrücken sortieren, die nicht in Ihrer SELECT-Liste stehen:

```
SELECT    COALESCE(o.name, 'Unknown') AS owner,
          w.name AS waterfall_name

FROM      waterfall w

          LEFT JOIN owner o ON w.owner_id = o.id

ORDER BY o.id DESC, w.id;
```

```
owner           waterfall_name
```

```
-----
```

```
MI DNR          Wagner Falls
```

```
AF LLC          Alger Falls
```

```
Michigan Nature Tannery Falls
```

...

Sie können auch nach numerischer Spaltenposition sortieren:

```
SELECT COALESCE(o.name, 'Unknown') AS owner,
          w.name AS waterfall_name
```

...

ORDER BY 1 DESC, 2 ASC;

owner	waterfall_name
-------	----------------

Unknown	Agate Falls
---------	-------------

Unknown	Bond Falls
---------	------------

Unknown	Canyon Falls
---------	--------------

...

Da die Zeilen einer SQL-Tabelle ungeordnet sind, könnten die Ergebnisse jedes Mal, wenn Sie eine Abfrage ausführen, in einer anderen Reihenfolge angezeigt werden – es sei denn, Sie fügen eine `ORDER BY`-Klausel hinzu.

ORDER BY kann nicht in einer Unterabfrage verwendet werden

Von den sechs Hauptklauseln kann nur `ORDER BY` nicht in einer Unterabfrage verwendet werden. Leider können Sie das Ordnen der Zeilen einer Unterabfrage nicht erzwingen.

Um dieses Problem zu umgehen, müssten Sie Ihre Abfrage mit einer anderen Logik neu schreiben, damit Sie nicht versucht sind, eine `ORDER BY`-Klausel in der Unterabfrage zu verwenden, und eine `ORDER BY`-Klausel nur in die äußere Abfrage einfügen.

Die LIMIT-Klausel

Wenn man sich schnell mal eine Tabelle anschauen möchte, ist es ganz praktisch, die Anzahl der ausgegebenen Zeilen zu beschränken und nicht die ganze Tabelle anzuzeigen.

MySQL, *PostgreSQL* und *SQLite* unterstützen die `LIMIT`-Klausel. *Oracle* und *SQL Server* verwenden eine andere Syntax mit derselben Funktionalität:

– MySQL, PostgreSQL und SQLite

```
SELECT *
```

```
FROM owner
```

```
LIMIT 3;
```

– Oracle

```
SELECT *
```

```
FROM owner
```

```
WHERE ROWNUM <= 3;
```

– SQL Server

```
SELECT TOP 3 *
```

```
FROM owner;
```

id	name	phone	type
1	Pictured Rocks	906.387.2607	public
2	Michigan Nature	517.655.5655	private
3	AF LLC		private

Eine andere Möglichkeit, die Anzahl der zurückgegebenen Zeilen zu beschränken, besteht darin, auf einer Spalte in der WHERE-Klausel zu filtern. Die

Filterung wird sogar noch schneller ausgeführt, wenn die Spalte indiziert ist.

Index

Symbole

- \d (Ziffern) in regulären Ausdrücken 208, 212
- _ (Unterstrich) mit LIKE verwenden 188
- , (Komma) Separator 66, 118
- , (Komma) Trennzeichen 240, 300
- ;(Semikolon) Beenden von SQL-Anweisungen 58
- !-= (Ungleichheit) Operator 182
- . (Punktnotation) 71
- " (einfache Anführungszeichen)
 - einfaches Anführungszeichen in String-Wert einbetten 157
 - String-Werte einschließen 62, 157
 - Escape-Folgen in 159
- "" (doppelte Anführungszeichen)
 - Bezeichner einschließen 57, 62, 157
 - Spaltenaliasse 70
 - Textwerte in Dateien einschließen 118
- () (runde Klammern)
 - Reihenfolge von Operationen anzeigen 181
 - Unterabfragen einschließen 73
- \ (Backslash) reguläre Ausdrücke in MySQL schützen 208
- (Minuszeichen) Subtraktions-operator 188
- * (Asterisk)
 - alle Spalten auswählen 67
 - Multiplikationsoperator 188
- Kommentar für eine einzelne Codezeile 24, 61
- / (Schrägstrich) Divisionsoperator 188
- /* */ Kommentar für mehrere Codezeilen 62

- % (Prozentzeichen)
 - mit LIKE verwenden 188
 - Modulo-Operator 188
 - verwenden mit LIKE 186
- + (Pluszeichen) Additionsoperator 179, 188
- <- (kleiner als) Operator 182
- <=- (kleiner als oder gleich) Operator 182
- <=>- (nullsichere Gleichheit) Operator in MySQL 183
- <>- (Ungleichheit) Operator 183
- = (Gleichheitszeichen)
 - beliebtester Vergleichsoperator 61
 - Equi-Joins 271
 - Gleichheitsoperator 179, 182
- >- (größer als) Operator 182
- >=- (größer als oder gleich) Operator 182
- || (Verkettung) Operator 200, 298
- ~ (Tilde) eingeschränkte Unterstützung für reguläre Ausdrücke in PostgreSQL 211
- \$\$ (Dollarzeichen) Strings in PostgreSQL einschließen 158

A

- Abfragekonzepte, erweiterte 231–259
 - CASE-Anweisungen 232–235
 - Fensterfunktionen 243–255
 - Gruppieren und Zusammenfassen 235–243
 - Pivoting und Unpivoting 255–259
- Abfragen 22–25, 65–98
 - beschleunigen durch einen Index 135
 - Ergebnisse in eine Tabelle einfügen 116
 - Grundlagen 65
 - mit UNION kombinieren 277
 - View erzeugen zum Sichern der Ergebnisse von 139
 - Zeilen aktualisieren mit den Ergebnissen von 131
- abgeleitete Tabellen 80
- abschließende Leerzeichen
 - aus String-Länge ausschließen 197
 - mit RTRIM entfernen 200

- absoluter Wert 192
- Aggregatfunktionen 190–192
 - COUNT 90
 - versus Fensterfunktionen 243
 - mehrere, in GROUP BY 237
 - Zeilen in einem einzigen Wert oder einer einzigen Liste zusammenfassen 238, 300
- aktuelles Datum oder aktuelle Zeit holen 214–215
- Aliasse 58
 - Spaltenaliasse 69
 - Spalten versus Tabellen 72
 - Unterabfragen mit Aliassen versehen 81
- ALL-Schlüsselwort 74
- ALTER-Rechte 121
- ALTER TABLE-Anweisungen 121, 123
- AND-Operator 85, 179, 180
- Anführungszeichen in SQL 62
- ANSI-Standards 52–55
 - entscheiden, welche im eigenen SQL-Code befolgt werden 53, 54
- Anweisungen 22, 58
 - Klauseln in 59
- ARRAY_AGG-Funktion 190, 235, 299
- ASCENDING-(ASC-)Schlüsselwort 96
- ASCII- versus Unicode-Codierung 161
- AS-Schlüsselwort
 - mit Tabellenaliasen verwenden 72
 - nutzen beim Umbenennen von Spalten 58
 - Spaltenaliasse 69, 72
- Atomarität 141
- Attribute 26
- Ausdrücke 60
- Ausführungsreihenfolge
 - SELECT- und HAVING-Klauseln 94
 - Vereinigungsoperatoren 281
- AUTOINCREMENT 115
- AVG-Funktion 190

ROWS BETWEEN-Klausel, verwenden mit 252

B

bedingte Anweisungen 61, 182

siehe auch Prädikate 85

Befehlsprompt 31

für MySQL 33

für Oracle 33

für PostgreSQL 34

für SQLite 32

für SQL Server 35

Benutzernamen

für Python-Verbindung zu einer Datenbank 42, 47

zum Verbinden mit dem Datenbankwerkzeug 39

Bereiche

mit BETWEEN testen, ob ein Wert in diesen fällt 184

RANGE BETWEEN versus ROWS BETWEEN 254

BETWEEN-Operator 179, 184

Bezeichner 57

Benennung 57

doppelte Anführungszeichen (""") einschließen 62

Beziehung, definiert 27

Bilder 147

in externen Dateien 175

binäre Datentypen 176

Binärwerte 176

externe Dateien speichern als 175

Bits 156

bitweise Operatoren 189

BLOB-Datentyp 177

boolesche Datentypen 174

bytea-Datentyp (PostgreSQL) 177

Bytes 156

C

CASCADE-Schlüsselwort 132

- Vorsicht mit 133
- CASE-Anweisungen 231, 232–235
 - Alternative zu PIVOT-Operation 256
 - statt der Oracle-Funktion DECODE verwenden 54
- Case Insensitivity
 - LIKE-Muster in MySQL, SQL Server und SQLite 187
 - reguläre Ausdrücke in MySQL 207
 - in SQL 23
- Case Sensitivity
 - LIKE-Muster in Oracle und PostgreSQL 187
 - reguläre Ausdrücke in PostgreSQL 210
- CAST-Funktion
 - Datum/Zeit-Wert referenzieren 166
 - Datumswert referenzieren 164
 - in einen numerischen Datentyp konvertieren 195–197
 - in einen String-Datentyp konvertieren 213
 - String in Datum/Zeit-Datentyp konvertieren 224
 - Zeitwert referenzieren 165
- CHAR-Datentyp 160
- CHARINDEX-Funktion 201
- CHECK-Constraint 110
 - DROP CHECK und DROP CONSTRAINT 128
- Cloud, Datenbanken in der 38
- Cloud-basierte Speicherlösungen
 - Notwendigkeit von SQL-Abfragen auf 11
- COALESCE-Funktion 95, 228, 287
- Codebeispiele aus diesem Buch 15
- COMMIT-Befehl 142
 - Änderungen bestätigen mit 144
 - kein ROLLBACK nach 144
- Common Table Expressions (CTEs) 282–291
 - nicht rekursive 282–285
 - rekursive 285–291
 - versus Unterabfragen 283–285
 - Vorteile von CTEs 284

- CONCAT-Funktion 298
- Constraints 108, 112, 125–129
 - aus einer Tabelle löschen 128
 - eindeutige Werte in Spalte verlangen 110
 - Fremdschlüssel, Tabelle löschen mit 132
 - Fremdschlüssel festlegen 113
 - für eine Tabelle anzeigen 126
 - in einer Tabelle modifizieren 127
 - löschen vor dem Löschen einer Spalte 123
 - NULL-Werte in Spalte verbieten 109
 - Primärschlüssel festlegen 111
 - Standardwerte in Spalte setzen 109
 - Werte beschränken in Spalte 110
 - zu einer Tabelle hinzufügen 127
- CONSTRAINT-Schlüsselwort 108
- CONVERT-Funktion 225
- COUNT-Funktion 90, 190
 - HAVING COUNT in Abfrage 296
 - HAVING-Klausel bezieht sich auf 94
 - mit DISTINCT verwenden 75, 294
- Create, Read, Update, and Delete *siehe* CRUD-Operationen 22
- CREATE-Anweisungen 138
 - Ausführungsrechte 103, 106, 136, 138
 - für Datenbanken 103
 - für Indizes 135
 - für Tabellen 105, 107
- CROSS JOIN 263, 264, 266, 272
 - Syntax 265
- CRUD-Operationen 22, 99–145
 - für Datenbanken 99–104
 - für Indizes 133–136
 - für Tabellen 104–133
 - mit Transaktionsmanagement verwenden 141–145
 - für Views 136–141
- CSV-Dateien, Daten in eine Tabelle einfügen aus 117–120

CTEs *siehe* Common Table Expressions 281

CUBE-Schlüsselwort 242

CURRENT_DATE-Funktion 60, 214

CURRENT_TIME-Funktion 214

CURRENT_TIMESTAMP-Funktion 214

D

Data Control Language (DCL) 63

Data Definition Language (DDL) 63

Data Manipulation Language (DML) 63

Data Query Language (DQL) 63

DATE-Datentyp 147

DATEDIFF-Funktion 217

Dateipfade zum Desktop (Beispiel) 119

Datenanalyse-Workflow 39

Datenbankdateien 38

Datenbanken 26, 99–104

- abfragen 66

- anlegen 37

- Anzeigen der Namen vorhandener 101

- Anzeigen des Namens der aktuellen 102

- anzeigen in MySQL 33

- anzeigen in Oracle 34

- anzeigen in PostgreSQL 34

- anzeigen in SQLite 32

- anzeigen in SQL Server 35

Datenmodell 26

Datenmodell versus Schema 100

- erzeugen 103

- löschen 103

NoSQL 18

Schemata 99

SQL 17

Tabellen qualifizieren mit Datenbanknamen 71

- über 17

- zu einer anderen wechseln 102

Datenbankmanagementsysteme (DBMS) 19

Datenbankobjekte 99

Datenbanktreiber 36

 Treiber für Python installieren 41

 Treiber für R installieren 46

Datenbankwerkzeuge 29, 36–39

 mit einer Datenbank verbinden 37

 Vergleich 37

Daten filtern

 Alternativen zur WHERE-Klausel 88

 mittels HAVING-Klausel 92

 mittels WHERE-Klausel 84–88

 auf Spalten filtern 84

 WHERE-Klausel verwenden

 auf Unterabfragen filtern 86–88

Datenmodelle 25–27, 99

 versus Schemata 100

Datentypen 147–178

 andere Daten 173–178

 boolesche Datentypen 174

 externe Dateien 175–178

 Datum/Zeit-Daten 163–173

 Datum/Zeit-Datentypen 167–173

 für eine Spalte wählen 149

 numerische Daten 150–156

 Dezimal-Datentypen 153

 Gleitkommatypen 154

 Integer-Datentypen 151–153

 von Spalten 105

 String-Daten 157–163

 Unicode-Datentypen 161

 Zeichentypen 159–161

DATE-Schlüsselwort 164

DATETIME-Datentyp 168

DATETIME-Schlüsselwort 166

Datum 163

Datentypen *siehe* Datum/Zeit-Datentypen 167

Datumswerte 163

Datum/Zeit-Daten 163–173

Datum/Zeit-Funktionen 163, 213–228

aktuelles Datum oder Zeit zurückgeben 214–215

Datum auf die nächstgelegene Zeiteinheit runden 223

Datums- oder Zeitintervall subtrahieren 215

Differenz zwischen zwei Daten/Zeiten ermitteln 219

Differenz zwischen zwei Datumsangaben ermitteln 216

Differenz zwischen zwei Zeitangaben ermitteln 218

in SQLite 172

String in Datum/Zeit-Datentyp konvertieren 224–228

CAST-Funktion verwenden 224

Datumsfunktion auf String-Spalte anwenden 227

STRING_TO_DATE, TO_DATE und CONVERT verwenden 225

Teil eines Datums oder einer Zeit extrahieren 220–222

Wochentag für ein Datum ermitteln 222

Datum/Zeit-Werte 163–167

Datum/Zeit-Datentypen 167–173

Datums- und Zeitwerte 166

Datumswerte 163

Zeitwerte 165

Datumsangaben

Datum/Zeit-Formatspezifikatoren 226

Datumseinheiten in unterschiedlichen RDBMS 220

Datumsformate 226

fehlende Daten mit rekursiver CTE einsetzen 286–288

String in Datum-Datentyp konvertieren 226

DB-Dateien *siehe* Datenbankdateien 38

DCL (Data Control Language) 63

DDL (Data Definition Language) 63

DECIMAL-Datentyp 155

DEFAULT-Constraint 109

deklarative Programmierung 52

- DELETE-Anweisungen 108, 125, 144, 145
 - Operatoren und Funktionen in 179
- DENSE_RANK-Funktion 245
 - versus ROW_NUMBER und RANK 247
- DESCENDING-(DESC-)Schlüsselwort 96
- Dezimale
 - Dezimal-Datentypen 153
 - Dezimalwerte 151
- DISTINCT-Schlüsselwort 74, 294
 - mit COUNT in SELECT-Klausel verwenden 75, 294
 - verwenden mit COUNT 90
- DML (Data Manipulation Language) 63
- Dokumente, speichern zur Verwendung in SQL 175
- doppelte Genauigkeit (Gleitkommatentypen) 155
- DOUBLE-Datentyp 155
- DQL (Data Query Language) 63
- DROP-Anweisungen
 - DROP CHECK und DROP CONSTRAINT 128
 - für Datenbanken 103
 - für Indizes 136
 - für Tabellen 108, 132
 - für Views 140
- Duplikate
 - doppelte Zeilen mit DISTINCT aus Ergebnis entfernen 75, 294
 - Zeilenduplikate mit UNION ALL bewahren 278
 - Zeilenduplikate mit UNION ausschließen 277
 - Zeilen mit doppelten Werten finden 293–296

E

- eindeutige Zeilen *siehe* DISTINCT-Schlüsselwort 75
- einfache Genauigkeit (Gleitkommatentypen) 155
- entfernte Server, Datenbanken auf 38
- Ergebnismengen 65
- Erweiterungen für SQL 52
- Escape-Folgen

nur für Strings, die in einfache Anführungszeichen eingeschlossen sind, nicht in Dollarzeichen (\$\$) 159

Escape-Folgen in einem String 158

ESCAPE-Schlüsselwort 188

EXCEPT-Operator 280

Ausführungsreihenfolge 281

existierende Tabelle trunkieren 125

EXISTS-Operator 184

JOIN versus 184

NOT EXISTS 185, 186

externe Dateien (Bilder, Dokumente usw.) 175

EXTRACT-Funktion 227

F

FALSE- und TRUE-Werte 174

fehlende Daten

aus CSV-Datei, Interpretationen durch RDBMS 120

fehlende Werte durch anderen Wert ersetzen 109

Fenster, Definition 244

Fensterfunktionen 231, 243–255

Aggregatfunktionen versus 243

ersten Wert in jeder Gruppe zurückliefern 247

erste zwei Werte aus jeder Gruppe zurückliefern 250

genauere Betrachtung 244

gleitenden Durchschnitt berechnen 252

laufende Gesamtsumme berechnen 253

vorhergehenden Zeilenwert zurückliefern 251

Zeilen in einer Tabelle ordnen 245

zweiten Wert aus jeder Gruppe zurückliefern 248

Festkommazahlen 153

FIRST_VALUE-Funktion 247

FLOAT-Datentyp 155

Fremdschlüssel 27

festlegen 113

Tabelle löschen mit Fremdschlüsselverweis 132

FROM-Klausel 24, 59, 76–84, 88

- Daten aus mehreren Tabellen, mittels JOIN 77
 - Tabellenaliasse 77
- Reihenfolge der Ausführung in SELECT-Anweisung 25
- Tabellenaliasse, definiert in 72
- Unterabfragen in 80–84
 - Vorteile von 82–84
- FULL OUTER JOIN 264, 269
- Funktionen 56, 179, 189–229
 - Aggregat 190, 192
 - Datum/Zeit 213–228
 - gebräuchlichste 180
 - Null 228
 - numerische 192–197
 - Operatoren versus 179
 - String 197–213
- funktionierender Code, schreiben, dann optimieren 87

G

- Gleitkommatentypen 154
- Gleitkommawerte 151
- go-Befehl (SQL Server) 36
- GREATEST-Funktion 191
- Groß- und Kleinschreibung
 - Aliasse in doppelten Anführungszeichen 70
- Groß- und Kleinschreibung, für einen String ändern 198
- GROUP_CONCAT-Funktion 190, 238, 299
- GROUP BY-Klausel 24, 59, 89–92, 236–240
 - in Abfrage Zeilen mit Maximalwert wählen 297
- HAVING-Klausel nach 93
- nach mehreren Spalten gruppieren 237
- Nichtaggregatspalten in 191
- Reihenfolge der Ausführung in SELECT-Anweisung 25
- Sammeln von Zeilen 89
- Schritte beim Verwenden 92
- Zeilen zu einem einzigen Wert oder einer einzigen Liste zusammenfassen 238
- zum Anlegen einer zusammenfassenden Tabelle 236

- zusammenfassen der Zeilen in Gruppen 90
- GROUPING SETS-Schlüsselwörter 242
- Gruppieren und Zusammenfassen 231, 235–243
 - GROUP BY-Klausel 236
 - mit ROLLUP, CUBE und GROUPING SETS 240
 - CUBE-Schlüsselwort 242
 - GROUPING SETS-Schlüsselwörter 242
 - ROLLUP-Schlüsselwort 241
- GUIs (grafische Benutzeroberflächen)
 - in Datenbankwerkzeugen 29
 - Oberfläche des Datenbankwerkzeugs verwenden zum Modifizieren einer Tabelle 124

H

- Häufig gestellte Fragen (Frequently Asked Questions, FAQs) 293–304
 - alle Tabellen mit einem bestimmten Spaltennamen finden 300–302
 - Text aus mehreren Feldern in einem einzigen Feld verketteten 298
 - aus Feldern in einer einzigen Zeile 298
 - aus Feldern in mehreren Zeilen 299
 - Zeilen mit doppelten Werten finden 293–296
 - alle einmaligen Kombinationen zurückliefern 294
 - nur Zeilen mit doppelten Werten zurückliefern 294–296
 - Zeilen mit einem Maximalwert für eine andere Spalte auswählen 296–298
- HAVING-Klausel 24, 92–94
 - in Abfrage zum Finden doppelter Werte 295
 - Ergebnisse filtern von GROUP BY 93
 - HAVING COUNT in Abfrage 296
 - muss immer auf GROUP BY folgen 93
 - Operatoren und Funktionen in 179
 - Reihenfolge der Ausführung in SELECT-Anweisung 25
 - verwendet mit SELECT, Ausführungsreihenfolge 94
 - WHERE versus 93
 - zum Filtern von Daten 88
- Hexadezimalwerte 176
- Homebrew-Paketmanager (Linux und macOS) 31

I

- IF EXISTS-Schlüsselwörter 132
- IF NOT EXISTS-Schlüsselwörter 107
- imperative Programmierung 51
- Indizes 133–136
 - beschränkte Anzahl von 135
 - Buch- versus SQL-Index 133
 - löschen 136
 - zum Beschleunigen von Abfragen erzeugen 135
- INNER JOIN 79, 263, 264, 267–268
 - in Abfrage Zeilen mit Maximalwert wählen 297
 - in Abfrage zum Finden von doppelten Werten 296
- IN-Operator 185
 - NOT IN 181, 186
- INSERT-Anweisungen 105, 116, 125
 - Operatoren und Funktionen in 179
- Installation, RDBMS 31
- INSTR-Funktion 201
- Integer
 - Auswahl an Integer-Datentypen 149
 - durch ein Integer dividieren 189
 - Integer-Datentypen 151
 - Integer-Werte 150
- INTEGER-Datentyp 105, 147, 149, 172
- Interpunktion in Spaltenaliassen 70
- INTERSECT-Operator 281
 - Ausführungsreihenfolge 281
- Intervalle
 - Datums- oder Zeitintervall subtrahieren 215
 - INTERVAL-Datentyp 173
- IS NOT NULL-Operator 186
- IS NULL-Operator 61, 186

J

- Joins 262–265
 - in Abfrage Zeilen mit Maximalwert wählen 297
 - Abfragen umschreiben mit 83

- anstelle einer korrelierten Unterabfrage verwenden 73
- CROSS JOIN 272
- in doppelte Zeilenabfrage 295
- EXISTS versus JOIN 184
- FULL OUTER JOIN 269
- Grundlagen von 265
- INNER JOIN 265, 267–268
- JOIN ... ON ...-Syntax 78, 267
- Join-Bedingung 263
- JOIN ist standardmäßig INNER JOIN 79, 263
- JOIN-Klausel 262
 - Analyse der 262
- JOIN-Klausel in FROM-Klausel verwenden 77
- Join-Typen 263
- JOIN versus UNION 275
- LEFT JOIN 268
- RIGHT JOIN 269
- Self Join 273–275
- Syntax zum Zusammenführen von Tabellen mit Joins 264
- Unterabfragen umschreiben mit 74
- USING- und NATURAL JOIN-Abkürzungen 270–272

K

- Klauseln 23, 59, 64, 65
 - Beispielabfrage mit den sechs wichtigsten Klauseln 65
 - Operatoren und Funktionen in 179
 - Reihenfolge der Ausführung in SELECT-Anweisung 25
 - Reihenfolge der Klauseln in SELECT-Anweisung 24
- Kommentare 61
- Konstanten *siehe auch* Literale 148
- korrelierte Unterabfrage 73
 - Leistungsprobleme mit 73

L

- LAG-Funktion 251, 287
- LAST_VALUE-Funktion 247

- LEAD-Funktion 252
- LEAST-Funktion 191
- LEFT JOIN 264, 268, 287
- LEFT OUTER JOIN 269
- Leistung
 - Abfragen beschleunigen mit Indizes 135
 - Code optimieren 88
 - korrelierte Unterabfragen, Probleme mit 73
- LENGTH-Funktion 197, 213
 - LEN-Funktion in SQL Server 213
- LIKE-Operator 183, 186
 - eingeschränkte Unterstützung für reguläre Ausdrücke in SQL Server 212
 - in Strings nach Text suchen 201
 - NOT LIKE 188
- LIMIT-Klausel 88, 97
- LISTAGG-Funktion 190, 235, 299
- Literale 148
- localhost 42, 47
- Logikoperatoren 180
- LOWER-Funktion 198
- LTRIM- und RTRIM-Funktionen 200

M

- mathematische Funktionen 192
- mathematische Operatoren 188
- MAX-Funktion 190, 191, 238
- mehrdeutiger Spaltenname (Fehler) 71
- Microsoft SQL Server *siehe* SQL Server 20
- MIN-Funktion 190, 191
- MongoDB 19
- Mustererkennung
 - LIKE-Operator *siehe auch* reguläre Ausdrücke 186
- MySQL
 - MySQL Workbench 37
 - Schemata und Datenbanken 101
 - SQL-Code schreiben mit 32

N

Namenskonventionen

- Bezeichner 57

- Spaltenaliasse 70

NATURAL JOIN 265

- einsetzen, um INNER JOIN zu ersetzen 271

- Vorsicht bei 272

Nichtbeachtung der Groß- und Kleinschreibung

- Schlüsselwörter in SQL 56

nichtkorrelierte Unterabfragen 73

nicht rekursive CTEs 282

NoSQL 18

NOT NULL-Constraint 109, 112

NOT-Operator 180

NTH_VALUE-Funktion 248

NULL-Werte 61

- aus CSV-Datei, Interpretationen durch RDBMS 120

- ersetzt durch COALESCE-Funktion 95

- in Spalte erlauben 109

- IS NULL- und IS NOT NULL-Operatoren 186

- NOT IN- versus NOT EXISTS-Operator 186

- Null-Funktionen 228

- NULL-Literal 148

- NULL- und NOT NULL-Spalten-Constraints 109

numerische Daten 150–156

- Dezimal-Datentypen 153

- Gleitkommatentypen 154

- Integer-Daten 151

- numerische Spalte mit String-Spalte vergleichen 195

- numerische Werte 150

numerische Funktionen 56, 150, 192–197

- CAST, in einen numerischen Datentyp konvertieren 195–197

- Mathematik 192

- Zahlen runden und trunkieren 195

- Zufallszahlen generieren 194

NVARCHAR-Datentyp 163

 VARCHAR versus 162

O

Object Relational Mappers (ORMs) 44

ON-Klausel 78

 in Joins durch USING ersetzen 270

 JOIN ... ON ...-Syntax, mehrere Bedingungen in 268

 Operatoren in 179

Operatoren 179–189

 versus Funktionen 179

 gebräuchlichste 180

 in Join-Bedingungen 263

 Logik 180

 math 188

 versus Prädikate 182

 Vereinigung 275–281

 Vergleich 182–188

 zum Kombinieren von Prädikaten verwenden 85

Oracle

 Oracle SQL Developer 37

 Procedural Language SQL (PL/SQL) 52

 Schemata und Benutzer 101

 SQL-Code schreiben mit 33

 verglichen mit anderen RDBMS 20

Oracle Database *siehe* Oracle 21

ORDER BY-Klausel 24, 94–97, 245

 alphabetische Sortierung in aufsteigender Reihenfolge 95

 darf nicht in Unterabfragen benutzt werden 97

 Reihenfolge der Ausführung in SELECT-Anweisung 25

 sortieren in absteigender Reihenfolge 96

 sortieren nach Spalten und Ausdrücken, die nicht in SELECT stehen 96

 in UNION-Abfragen 278

OR-Operator 85, 179, 180, 181

OVER ... PARTITION BY ...-Syntax 244, 298

OVER-Schlüsselwort 245

P

pandas-DataFrame 44

PARTITION BY-Schlüsselwörter 245, 254

Passwörter

- für Python-Verbindung zu einer Datenbank 42

- für R-Verbindung an eine Datenbank 47

- zum Verbinden mit dem Datenbankwerkzeug 39

Pivoting und Unpivoting 231, 255–259

- Werte einer Spalte in mehrere Spalten zerlegen 255–257

- Werte mehrerer Spalten in einer einzigen Spalte auflisten 257–259

PIVOT-Operation 255–257

- CASE-Anweisung als Alternative zu 256

PL/SQL (Procedural Language SQL) 52

POSITION-Funktion 201

POSIX-Syntax, reguläre Ausdrücke 209, 211

PostgreSQL

- pgAdmin 37

- SQL-Code schreiben mit 34

- Standarddatenbank, postgres 104

- verglichen mit anderen RDBMS 20

Prädikate 60, 85

- mehrere kombinieren mittels Operatoren 85

- Operatoren versus 182

Präzision 154

- Gleitkommatentypen mit doppelter Genauigkeit 155

- Gleitkommatentypen mit einfacher Genauigkeit 155

Primärschlüssel 26, 111–112

- Best Practices 112

- Fremdschlüssel verweisen auf 113

Programmiersprachen

- SQL-Code schreiben in 29, 39–49

 - Python 40–45

 - R 45–49

- Vergleich von SQL mit anderen 51

Python 51

- mit einer Datenbank verbinden 40–43
- mit RDBMS verbinden und SQL-Code schreiben in 30
- SQLAlchemy-Paket 44
- SQL-Code schreiben in 43–45
- Verwendung von SQL mit 13

R

R, Sprache

- Verwendung von SQL mit 13

RANK-Funktion 245

- versus ROW_NUMBER und DENSE_RANK 247

RDBMS

- ANSI SQL versus RDBMS-spezifisches SQL 53

- definiert 20

- für ein RDBMS entscheiden 30

 - MySQL 32

 - Oracle 33

 - PostgreSQL 34

 - SQLite 31

 - SQL Server 35

- SQLAlchemy verwenden mit 45

- SQL-Code schreiben mit 30

- Variationen in SQL-Syntax 20, 304

- Vergleiche von 20

RDBMS-Software 29

REAL-Datentyp 172

RECURSIVE-Schlüsselwort 283

REGEXP_REPLACE-Funktion 205

- in Oracle 208

- in PostgreSQL 211

REGEXP-Funktion (MySQL) 207

reguläre Ausdrücke 183, 205–212

- in Oracle für die Suche nach Teilstrings verwenden 202

- in Oracle zum Extrahieren von Teilstrings verwenden 204

- in MySQL 207

- in Oracle 208

- in PostgreSQL 210
- in SQL Server 212
- wichtige Hinweise zu 206
- Reihenfolge der Ausführung SELECT-Anweisung 25
- rekursive CTEs 282, 285–291
 - alle Eltern von Kindzeilen zurückliefern 288–291
 - fehlende Zeilen in einer Datensequenz einsetzen 285–288
- relationale Datenbanken 17
- relationale Datenbankmanagementsysteme *siehe* RDBMS 20
- REPLACE-Funktion 204
- RIGHT JOIN 264, 269
- RIGHT OUTER JOIN 269
- ROLLBACK-Befehl 142
 - Änderungen widerrufen mit 144
- ROLLUP-Schlüsselwort 241
- ROW_NUMBER-Funktion 244, 245
 - in einer Fensterfunktion 244
 - in Unterabfrage verwenden, um mehrere Ränge aus jeder Gruppe zurückzuliefern 250
 - versus RANK und DENSE_RANK 247
- ROWS BETWEEN-Klausel 252
- ROWS BETWEEN UNBOUNDED-Klausel 253
- R-Sprache
 - mit RDBMS verbinden und SQL-Code schreiben in 30
 - SQL-Code schreiben in 48
 - verbinden mit einer Datenbank 45–48
- RTRIM- und LTRIM-Funktionen 200

S

- Schemata 99
 - Datenmodelle versus 100
 - in SQL-Datenbanken 18
 - Tabellen qualifizieren mit Schemanamen 71
 - variierende Definitionen in RDBMS 101
- Schlüsselwörter 23, 55, 64
 - Nichtbeachtung der Groß- und Kleinschreibung in SQL 56
 - Operatoren als 180

- SELECT-Anweisungen 23, 58, 143
 - Ergebnisse aus zwei oder mehr mit UNION kombinieren 276
 - Klauseln in 23, 59
 - Operatoren und Funktionen in 179
- SELECT-Klausel 24, 59, 66–76
 - COUNT und DISTINCT in 75
 - DISTINCT in 74
 - Operatoren und Funktionen in 179
 - Reihenfolge der Ausführung in SELECT-Anweisung 25
 - Spaltenalias
 - Aliasse mit Groß- und Kleinschreibung und Interpunktion 70
 - Spaltenaliasse 69
 - Spalten qualifizieren 70
 - Tabellen qualifizieren 71
 - Unterabfragen auswählen 72–74
- Self Join 265, 273–275
- Semi-Join 185
- SIMILAR TO, eingeschränkte Unterstützung für reguläre Ausdrücke in PostgreSQL 211
- Skala 154
- SMALLINT-Datentyp 150
- sortieren
 - aufsteigende und absteigende Reihenfolge 96
 - nach Spalten und Ausdrücken, die nicht in der SELECT-Liste stehen 96
- Spalten 26
 - Anzeigen von Spaltennamen in SQLite-Ausgabe 32
 - auf Spalten filtern in WHERE-Klausel 84
 - aus einer Tabelle löschen 123
 - Daten aktualisieren in 129
 - einer Tabelle hinzufügen 122
 - für eine Tabelle anzeigen 122
 - mehrere, eindeutige Kombinationen zählen 76
 - mehrere, in GROUP BY 237
 - Spaltennamen qualifizieren 70
 - umbenennen 121
 - umbenennen mit Aliassen 58

- Spaltenaliasse 79, 262
 - erzeugen 69
 - mit Groß- und Kleinschreibung und Interpunktion 70
 - mit Unterabfragen verwenden in SELECT-Klausel 73
 - versus Tabellenaliasse 72
- Spaltennamen, suchen nach 300
- Spaltennamen qualifizieren 71, 78
- SQL
 - integrale Rolle in der Datenlandschaft 11
 - online nach Syntax suchen 20
 - Sprachzusammenfassung 64
 - Teilsprachen 63
 - über 17
 - Vergleich mit anderen Programmiersprachen 51
- SQLAlchemy-Paket (Python) 44
- SQL-Anweisungen 22
- SQLite
 - am schnellsten einzurichtendes RDBMS 30
 - Datenbank gespeichert in externen Dateien 101
 - DB Browser for SQLite 37
 - verglichen mit anderen RDBMS 20
- SQL Server 21
 - SQL-Code schreiben mit 35
 - SQL Server Management Studio 37
 - Standarddatenbank, master 104
 - verglichen mit anderen RDBMS 20
- Start Fridays with grandma's homemade oatmeal (Eselsbrücke für Klauseln) 24
- START TRANSACTION 141
- Sternschema 100
- STR_TO_DATE-Funktion 225
- STRING_AGG-Funktion 190, 235, 299
- String-Funktionen 56, 157, 197–213
 - Groß- und Kleinschreibung eines Strings ändern 198
 - in einen String-Datentyp konvertieren 212
 - Länge eines Strings ermitteln 197

- nach Text in Strings suchen 201
- reguläre Ausdrücke verwenden 205–212
- Strings verketteten 200
- Teil eines Strings extrahieren 203
- Text in einem String ersetzen 204
- Text löschen aus einem String 205
- unerwünschte Zeichen rund um String wegschneiden 198–200

Strings 157–163

- in Datum/Zeit-Datentyp konvertieren 224–228
- in Dezimal umwandeln, um mit einer Zahl zu vergleichen 196
- in einfache Anführungszeichen (') einschließen 62
- String-Spalte mit numerischer Spalte vergleichen 195
- String-Werte 157, 159
 - Alternative zur Verwendung von einfachen Anführungszeichen 157
 - Escape-Folgen für 158
- String-zu-Datum/Zeit-Funktionen 166
- String-zu-Datum-Funktionen 165
- String-zu-Zeit-Funktionen 165
- Unicode-Datentypen 161
- Zeichendatentypen 159

Strings verketteten

- || (Verkettung) Operator 200
- CONCAT-Funktion 200
- Text aus mehreren Feldern in einem einzigen Feld 298–300
 - Text aus Feldern in einer einzigen Zeile verketteten 298
 - Text aus Feldern in mehreren Zeilen verketteten 299–300

SUBSTR- oder SUBSTRING-Funktion 203

SUM-Funktion 190, 243

- mit ROWS BETWEEN UNBOUNDED-Klausel verwenden 253

Sweaty feet will give horrible odors (Eselsbrücke für Klauseln) 24

T

Tabellen 17, 26

- Abfrageergebnisse einfügen in 116–117
- abgeleitete Tabellen 80
- anzeigen in MySQL 33

- anzeigen in Oracle 34
- anzeigen in PostgreSQL 34
- anzeigen in SQLite 32
- anzeigen in SQL Server 35
- arbeiten mit mehreren 261–291
- einfache Tabelle erzeugen 105
- erzeugen zum Füllen der neuen Datenbank 37
- mit UNION kombinieren und Zeilenduplikate eliminieren 277
- modifizieren 120–133
 - Constraints anzeigen 126
 - Constraints löschen 128
 - Datenspalte aktualisieren 129
 - Datentyp für eine Spalte ändern 150
 - Datenzeilen aktualisieren 130
 - ein Constraint hinzufügen 127
 - ein Constraint modifizieren 127
 - eine Spalte hinzufügen 122
 - eine Tabelle löschen 132
 - Spalte aus einer Tabelle löschen 123
 - Spalten anzeigen 122
 - Spalte oder Tabelle umbenennen 121–122
 - Tabelle mit Fremdschlüsselverweis löschen 132
 - Zeilen anzeigen 125
 - Zeilen hinzufügen 125
 - Zeilen löschen 125
 - Zeilen mit Abfrageergebnissen aktualisieren 131
- Namen vorhandener Tabellen anzeigen 107
- SQL-Anforderungen für 104
- Tabelle erzeugen, die noch nicht existiert 107
- Tabelle erzeugen mit automatisch generiertem Feld 114
- Tabelle erzeugen mit Constraints
 - eindeutige Werte in Spalte verlangen 110
 - NULL-Werte in Spalte verbieten 109
 - Standardwerte in Spalte setzen 109
 - Werte in Spalte beschränken mit CHECK 110

- Tabelle erzeugen mit Primär- und Fremdschlüsseln 111–114
 - Fremdschlüssel festlegen 113
 - Primärschlüssel festlegen 111
- Tabelle mit Constraints erzeugen 108–111
- Tabellennamen qualifizieren 71
- Textdateidaten einfügen in 117–120
 - umbenennen mit Aliassen 58
- Views versus 66, 137
- Tabellenaliasse 71
 - in FROM-Klausel verwenden 77
 - Spaltenaliasse versus 72
- Tabellennamen anzeigen 300
- Tabellennamen qualifizieren 71
- TCL (Transaction Control Language) 64
- Teilsprachen (SQL) 63
- Teilstrings, in Strings suchen 202
- Terminal-Fenster, SQL-Code schreiben in 29, 31
- Text
 - alle Tabellen mit einem bestimmten Spaltennamen finden 300–302
 - aus mehreren Feldern in einem einzigen Feld verketteten 298–300
 - Daten aus Textdatei in eine Tabelle einfügen 117–120
 - ersetzen in einem String 204
 - in einem String suchen 201
- TEXT-Datentyp 160, 172
- TIME-Schlüsselwort 165
- TIMESTAMP-Datentyp 168
- TIMESTAMP-Schlüsselwort 166
- TINYINT-Datentyp 150
- TO_DATE-Funktion 225
- Transaction Control Language (TCL) 64
- Transaktion, definiert 141
- Transaktionsmanagement 141–145
 - Änderungen vor COMMIT genau überprüfen 142
 - Änderungen widerrufen mit ROLLBACK 144
 - kein ROLLBACK nach COMMIT 144

- Vorteile des 142
- Trennzeichen 240, 300
- TRIM-Funktion 197, 198
 - führende oder abschließende Zeichen entfernen 199
- TRUE- und FALSE-Werte 174
- Typaffinitäten (SQLite) 105

U

- Umbenennen
 - von Spalten 121
 - von Tabellen 121
- Unicode-Datentypen 161
 - ASCII- versus Unicode-Codierung 161
- UNION ALL, Alternative zu UNPIVOT 258
- UNIQUE-Constraint 110, 112
- UNPIVOT-Operation 257–259
 - UNION ALL als Alternative zu 258
- Unterabfragen
 - anstelle eines View verwenden 137
 - Common Table Expressions versus 283–285
 - Daten filtern auf 248, 249
 - in DISTINCT einpacken und COUNT einsetzen auf 76
 - in FROM-Klausel 80–84
 - Ausführungsreihenfolge für Beispielabfrage 80–82
 - Vorteile der Verwendung von 82–84
 - keine ORDER BY-Klauseln in 97
 - korrelierte, Leistungsprobleme mit 73
 - korrelierte versus nichtkorrelierte 73
 - in SELECT-Klausel 72
 - temporär mit Aliassen benennen 58
 - versus Views 138
 - in WHERE-Klausel 86
 - Vorteile von 86
 - versus WITH-Klausel 81
- Unveränderlichkeit, Primärschlüssel 112
- UPDATE-Anweisungen 129

- Datenzeilen aktualisieren 130
- eine Datenspalte aktualisieren 129
- kein Widerruf der Ergebnisse möglich 303
- Operatoren und Funktionen in 179
- Tabelle aktualisieren, deren ID einer anderen Tabelle entspricht 302
- Werte aktualisieren anhand einer Abfrage 131
- UPPER-Funktion 179, 198
- USING-Klausel, ON in Joins ersetzen 265, 270
- UTF (Unicode Transformation Format) 162

V

- VARBINARY-Datentyp (SQL Server) 177
- VARCHAR-Datentyp 105, 147, 159
 - versus NVARCHAR 162
 - VARCHAR2 in Oracle 115
- Verbindungen (Datenbank)
 - für Python einrichten 41–43
 - für R einrichten 45–48
 - verbinden des Datenbankwerkzeugs mit der Datenbank 38
- Vereinigungsoperatoren 275–281
 - Ausführungsreihenfolge 281
 - EXCEPT 280
 - INTERSECT 281
 - JOIN versus UNION 275
 - übereinstimmende Datentypen erforderlich 278
 - UNION 276
 - mit anderen Klauseln 278
 - mit mehr als zwei Tabellen 279
 - UNION ALL 278, 287
- Vergleichsoperatoren 182–188
 - BETWEEN 184
 - EXISTS 184
 - IN 185
 - IS NULL 186
 - LIKE 186
 - Schlüsselwörter 183

Symbole 182

verteilte Datenverarbeitungs-Frameworks, SQL-artige Schnittstellen 11

Views 136–141

abfragen 66

aktualisieren 140

erzeugen, um den Zugriff auf Tabellen zu beschränken 137

erzeugen zum Speichern von Abfrageergebnissen 139

löschen 140

Unterabfragen versus 138

vorhandene anzeigen 139

vi-Texteditor 35

vorhandene Tabelle trunkieren 108

W

WHERE-Klausel 24, 59, 84–88

auf Spalten filtern 84, 98

auf Unterabfragen filtern 86

in CROSS JOINS 273

DELETE-Anweisung nutzen 144

versus HAVING 93

Operatoren und Funktionen in 179

Reihenfolge der Ausführung in SQL-Code 25

in Self Join 274

in UPDATE-Anweisung 130

weglassen in DELETE-Anweisung 125

Whitespace

Leerzeichen in Spaltenaliasnamen 70

Leerzeichen um einen String herum entfernen 198

in SQL 62, 64

Wie mache ich ... *siehe* Häufig gestellte Fragen 293

WITH-Klausel

in Common Table Expressions 282

RDBMS unterstützen 82

Unterabfragen versus 81

Y

YEAR-Funktion 179, 221

Z

Zahlen runden 195

Zählen startet in SQL bei 1 202

Zahlen trunkieren 195

Zeichen, rund um Strings wegschneiden 198–200

Zeichentypen 159

- ASCII- versus Unicode-Codierung 162

- Unicode-Codierungen 161

- VARCHAR, CHAR und TEXT 159

Zeilen

- aus einer Tabelle löschen 125

- für eine Tabelle anzeigen 125

- in einer Tabelle ordnen 245

- in eine Tabelle einfügen 125

- mit Abfrageergebnissen aktualisieren 131

- ROWS BETWEEN versus RANGE BETWEEN 254

- sammeln in GROUP BY 89

- Werte aktualisieren in 130

- zu einem einzigen Wert oder einer einzigen Liste zusammenfassen 238

Zeit

- Datentypen *siehe* Datum/Zeit-Datentypen 167

- String in einen Zeit-Datentyp konvertieren 225

- Zeiteinheiten in unterschiedlichen RDBMS 221

- Zeitformate 226

- Zeitwerte 165

Zeitangabefunktionen 56

Zeit *siehe auch* Datum/Zeit-Daten 163

Ziffern 156

Zufallszahlengenerator 194

zusammengesetzte Indizes 135

zusammengesetzter Schlüssel 112

Zuweisungsoperatoren 189