O'REILLY®

3. Auflage

Einführung in SQL

Daten erzeugen, bearbeiten und abfragen

Alan Beaulieu
Übersetzung
von Thomas Demmig

Inhalt

KAPITEL 4

Filtern

In manchen Fällen wird mit allen Zeilen einer Tabelle gearbeitet, nämlich:

- Wenn alle Daten aus einer Tabelle geleert werden sollen, um neue Data-Warehouse-Feeds zu präsentieren.
- Wenn alle Zeilen einer Tabelle modifiziert werden sollen, nachdem eine neue Spalte hinzugefügt wurde.
- Wenn alle Zeilen aus einer Message-Queue-Tabelle abgeholt werden sollen.

In Fällen wie diesen benötigt eine SQL-Anweisung keine `where`-Klausel, da keine Zeilen von der Betrachtung ausgeschlossen werden müssen. Meist ist jedoch nur ein Teil der Tabellenzeilen von Interesse. Daher haben alle SQL-Datenanweisungen (außer `insert`) eine optionale `where`-Klausel mit einer oder mehreren Filterbedingungen, die die Zahl der von der SQL-Anweisung betrachteten Zeilen eingrenzen. Zusätzlich enthält die `select`-Anweisung eine `having`-Klausel, um Filterbedingungen zu formulieren, die für gruppierte Daten gelten sollen. Dieses Kapitel untersucht die verschiedenen Filterbedingungen, die in den `where`-Klauseln der `select`-, `up date`- und `delete`-Anweisungen stehen können; den Einsatz von Filterbedingungen in der `having`-Klausel einer `select`-Anweisung werden wir uns in Kapitel 8 ansehen.

Bedingungsauswertung

Eine `where`-Klausel kann eine oder mehrere durch die Operatoren `and` und `or` getrennte Bedingungen enthalten. Wenn mehrere Bedingungen lediglich durch den `and`-Operator getrennt werden, müssen alle diese Bedingungen `true` sein, damit die betreffende Zeile in die Ergebnismenge aufgenommen wird. Betrachten Sie die folgende `where`-Klausel:

```
WHERE first_name = 'STEVEN' AND create_date > '2006-01-01'
```

Diese beiden Bedingungen führen dazu, dass nur Zeilen berücksichtigt werden, bei denen der Vorname Steven ist und das Erstellungsdatum nach dem 1. Januar 2006 liegt. Dieses Beispiel verwendet zwar nur zwei Bedingungen, aber im Prinzip können Sie beliebig viele Bedingungen in die `where`-Klausel schreiben. Wenn alle durch den Operator `and` getrennt sind, müssen auch alle `true` sein, damit die Zeile in die Ergebnismenge kommt.

Sind sämtliche Bedingungen der `where`-Klausel durch den Operator `or` getrennt, muss hingegen nur eine von ihnen `true` sein, damit die betreffende Zeile in der Ergebnismenge Aufnahme findet. Betrachten Sie die folgenden beiden Bedingungen:

```
WHERE first_name = 'STEVEN' OR create_date > '2006-01-01'
```

Jetzt kann eine Zeile auf mehreren Wegen in die Ergebnismenge gelangen:

- Der Vorname ist Steven, und das Erstellungsdatum liegt nach dem 1. Januar 2006.
- Der Vorname ist Steven, und das Erstellungsdatum liegt vor dem 1. Januar 2006, oder es ist dieses Datum.
- Der Vorname ist nicht Steven, aber das Erstellungsdatum liegt nach dem 1. Januar 2006.

Tabelle 4-1 zeigt die möglichen Resultate einer `where`-Klausel mit zwei durch `or` getrennten Bedingungen.

Tabelle 4-1: Auswertung von zwei Bedingungen mit dem or-Operator

Zwischenergebnis	Endergebnis
`WHERE true OR true`	`true`
`WHERE true OR false`	`true`
`WHERE false OR true`	`true`
`WHERE false OR false`	`false`

Im obigen Beispiel ist eine Zeile nur in einem Fall aus der Ergebnismenge ausgeschlossen: wenn der Vorname nicht Steven ist und das Erstellungsdatum vor dem 1. Januar 2006 liegt oder dieses Datum ist.

Verwendung von Klammern

Wenn Ihre `where`-Klausel mindestens drei Bedingungen und sowohl `and` als auch `or` enthält, sollten Sie durch Klammern klarstellen, was Sie bezwecken, damit sowohl die Datenbank als auch nachfolgende Generationen Ihren Code noch lesen können. Die folgende `where`-Klausel erweitert das obige Beispiel um die Prüfung, ob der Vorname Steven oder der Nachname Young ist und das Erstellungsdatum nach dem 1. Januar 2006 liegt:

```
WHERE (first_name = 'STEVEN' OR last_name = 'YOUNG')

  AND create_date > '2006-01-01'
```

Jetzt gibt es drei Bedingungen: Damit eine Zeile in die Ergebnismenge gelangt, müssen zuerst die erste oder zweite Bedingung (oder beide) `true` und dann die dritte ebenfalls `true` sein. Tabelle 4-2 zeigt die möglichen Resultate dieser `where`-Klausel.

Tabelle 4-2: Auswertung von drei Bedingungen mit and und or

Zwischenergebnis	Endergebnis
`WHERE (true OR true) AND true`	`true`
`WHERE (true OR false) AND true`	`true`
`WHERE (false OR true) AND true`	`true`
`WHERE (false OR false) AND true`	`false`
`WHERE (true OR true) AND false`	`false`
`WHERE (true OR false) AND false`	`false`
`WHERE (false OR true) AND false`	`false`
`WHERE (false OR false) AND false`	`false`

Man sieht: Je mehr Bedingungen eine `where`-Klausel aufweist, umso mehr Kombinationen muss der Server auswerten. In diesem Fall ergeben nur drei der acht möglichen Kombinationen das Endergebnis `true`.

Verwendung des Operators not

Hoffentlich war das obige Beispiel mit den drei Bedingungen gut zu verstehen. Betrachten Sie gleichwohl folgende Abänderung:

```
WHERE NOT (first_name = 'STEVEN' OR last_name = 'YOUNG')

  AND create_date > '2006-01-01'
```

Haben Sie sie erkannt? Ich habe vor dem ersten Satz an Bedingungen einen `not`-Operator eingefügt. Nun suche ich nicht mehr nach Personen, deren Vorname Steven oder deren Nachname Young lautet und deren Datensatz nach dem 1. Januar 2006 erstellt wurde, sondern ich erhalte nur die Zeilen, bei denen der Vorname nicht Steven oder der Nachname Young ist und deren Datensatz nach dem 1. Januar 2006 angelegt wurde. Tabelle 4-3 zeigt die möglichen Resultate für dieses Beispiel.

Tabelle 4-3: Auswertung von drei Bedingungen mit and, or und not

Zwischenergebnis	Endergebnis
`WHERE NOT (true OR true) AND true`	`false`
`WHERE NOT (true OR false) AND true`	`false`
`WHERE NOT (false OR true) AND true`	`false`
`WHERE NOT (false OR false) AND true`	`true`
`WHERE NOT (true OR true) AND false`	`false`
`WHERE NOT (true OR false) AND false`	`false`
`WHERE NOT (false OR true) AND false`	`false`
`WHERE NOT (false OR false) AND false`	`false`

Der `not`-Operator ist für einen Datenbankserver kein Problem, aber ein Mensch kann eine `where`-Klausel mit diesem Operator nicht mehr so leicht durchschauen. Daher trifft man ihn nur selten an. In diesem Fall könnte man die `where`-Klausel auch ohne `not`-Operator schreiben:

```
WHERE first_name <> 'STEVEN' AND last_name <> 'YOUNG'
```

```
  AND create_date > '2006-01-01'
```

Dem Server ist es sicherlich egal, aber Sie als Leser können diese Version der `where`-Klausel bestimmt besser verstehen.

Aufbau einer Bedingung

Da Sie nun gesehen haben, wie mehrere Bedingungen vom Server ausgewertet werden, gehen wir wieder einen Schritt zurück und betrachten, was eigentlich eine einzelne Bedingung ausmacht. Eine Bedingung besteht aus einem oder mehreren *Ausdrücken*, die durch einen oder mehrere *Operatoren* verbunden sind. Ein Ausdruck kann Folgendes sein:

- eine Zahl
- eine Spalte einer Tabelle oder View
- ein String-Literal wie beispielsweise `'Maple Street'`
- eine eingebaute Funktion wie beispielsweise `concat('Learning', ' ', 'SQL')`
- eine Unterabfrage
- eine Liste von Ausdrücken wie beispielsweise `('Boston', 'New York', 'Chicago')`

Folgende Operatoren sind in Bedingungen zulässig:

- Vergleichsoperatoren wie beispielsweise `=`, `!=`, `<`, `>`, `<>`, `LIKE`, `IN` und `BETWEEN`
- arithmetische Operatoren wie beispielsweise `+`, `–`, `*` und `/`

Der folgende Abschnitt zeigt, wie diese Ausdrücke und Operatoren kombiniert werden können, um verschiedene Arten von Bedingungen zu konstruieren.

Bedingungstypen

Es gibt viele verschiedene Wege, unerwünschte Daten herauszufiltern. Sie können bestimmte Werte, Wertemengen oder Wertebereiche ein- oder ausschließen oder mit diversen Mustererkennungstechniken Teilübereinstimmungen in String-Daten ausfindig machen. Im Folgenden werden diese Bedingungstypen genauer erklärt.

Gleichheitsbedingungen

Ein großer Prozentsatz der Filterbedingungen, die Sie schreiben oder über die Sie stolpern werden, hat die Form '*spalte = ausdruck*':

```
title = 'RIVER OUTLAW'

fed_id = '111-11-1111'

amount = 375.25

film_id = (SELECT film_id FROM film WHERE title = 'RIVER
OUTLAW')
```

Solche Bedingungen bezeichnet man als *Gleichheitsbedingungen*, da sie einen Ausdruck mit einem anderen gleichsetzen. Die ersten drei Beispiele setzen eine Spalte mit einem Literal gleich (zweimal ein String und einmal eine Zahl), und das vierte Beispiel setzt eine Spalte mit dem Rückgabewert einer Unterabfrage gleich. Die folgende Abfrage verwendet zwei Gleichheitsbedingungen: eine in der `on`-Klausel (eine Join-Bedingung) und die andere in der `where`-Klausel (eine Filterbedingung):

```
mysql> SELECT c.email

    -> FROM customer c

    ->   INNER JOIN rental r

    ->   ON c.customer_id = r.customer_id

    -> WHERE date(r.rental_date) = '2005-06-14';

+-------------+

| email                                |
```

```
+--------------+
| CATHERINE.CAMPBELL@sakilacustomer.org |
| JOYCE.EDWARDS@sakilacustomer.org      |
| AMBER.DIXON@sakilacustomer.org        |
| JEANETTE.GREENE@sakilacustomer.org    |
| MINNIE.ROMERO@sakilacustomer.org      |
| GWENDOLYN.MAY@sakilacustomer.org      |
| SONIA.GREGORY@sakilacustomer.org      |
| MIRIAM.MCKINNEY@sakilacustomer.org    |
| CHARLES.KOWALSKI@sakilacustomer.org   |
| DANIEL.CABRAL@sakilacustomer.org      |
| MATTHEW.MAHAN@sakilacustomer.org      |
| JEFFERY.PINSON@sakilacustomer.org     |
| HERMAN.DEVORE@sakilacustomer.org      |
| ELMER.NOE@sakilacustomer.org          |
| TERRANCE.ROUSH@sakilacustomer.org     |
| TERRENCE.GUNDERSON@sakilacustomer.org |
```

```
+-------------+

16 rows in set (0.03 sec)
```

Diese Abfrage zeigt die E-Mail-Adressen aller Kunden, die am 14. Juni 2005 einen Film ausgeliehen haben.

Ungleichheitsbedingungen

Ebenfalls ziemlich gebräuchlich ist die *Ungleichheitsbedingung*, die fordert, dass zwei Ausdrücke *nicht* gleich sind. Hier habe ich die Filterbedingung aus der `where`-Klausel der obigen Abfrage in eine Ungleichheitsbedingung umgewandelt:

```
mysql> SELECT c.email

    -> FROM customer c

    ->   INNER JOIN rental r

    ->   ON c.customer_id = r.customer_id

    -> WHERE date(r.rental_date) <> '2005-06-14';

+------------+

| email                           |

+------------+

| MARY.SMITH@sakilacustomer.org     |

| MARY.SMITH@sakilacustomer.org     |

| MARY.SMITH@sakilacustomer.org     |
```

```
| MARY.SMITH@sakilacustomer.org     |
| MARY.SMITH@sakilacustomer.org     |
| MARY.SMITH@sakilacustomer.org     |
| MARY.SMITH@sakilacustomer.org     |
| MARY.SMITH@sakilacustomer.org     |
| MARY.SMITH@sakilacustomer.org     |
| MARY.SMITH@sakilacustomer.org     |
…
| AUSTIN.CINTRON@sakilacustomer.org |
| AUSTIN.CINTRON@sakilacustomer.org |
| AUSTIN.CINTRON@sakilacustomer.org |
| AUSTIN.CINTRON@sakilacustomer.org |
| AUSTIN.CINTRON@sakilacustomer.org |
| AUSTIN.CINTRON@sakilacustomer.org |
| AUSTIN.CINTRON@sakilacustomer.org |
| AUSTIN.CINTRON@sakilacustomer.org |
+-------------+
```

```
16028 rows in set (0.03 sec)
```

Diese Abfrage zeigt alle E-Mail-Adressen für Filmausleihvorgänge, die an einem Datum stattfanden, das nicht der 14. Juni 2005 war. Bei der Konstruktion von Ungleichheitsbedingungen können Sie entweder `!=` oder `<>` als Operator verwenden.

Daten mit Gleichheitsbedingungen modifizieren

Gleichheits- und Ungleichheitsbedingungen werden gern auch zur Modifikation von Daten eingesetzt. Angenommen, die Videothek wirft einmal jährlich alte Kontendatensätze über Bord. Ihre Aufgabe ist es, aus der `rental`-Tabelle die Zeilen zu entfernen, bei denen der Ausleihvorgang im Jahr 2004 lag. Hier sehen Sie eine Möglichkeit, dies zu tun:

```
DELETE FROM rental

WHERE year(rental_date) = 2004;
```

Diese Anweisung enthält eine Gleichheitsbedingung, das folgende Beispiel nutzt dagegen zwei Ungleichheitsbedingungen, um alle Zeilen zu entfernen, bei denen das Ausleihdatum nicht in den Jahren 2005 oder 2006 lag:

```
DELETE FROM rental

WHERE year(rental_date) <> 2005 AND year(rental_date) <> 2006;
```

Bei der Formulierung von `delete`- und `update`-Anweisungen werde ich versuchen, jede Anweisung so zu schreiben, dass keine Zeilen modifiziert werden. So bleiben die Daten bei der Ausführung der Anweisungen unverändert, und die Ausgaben Ihrer `select`-Anweisungen entsprechen immer den in diesem Buch gezeigten.

Da MySQL-Sessions standardmäßig im Auto-Commit-Modus ablaufen (siehe Kapitel 12), könnten Sie, falls meine Anweisungen die Beispieldaten ändern würden, diese Änderungen nicht wieder zurückrollen (ungeschehen machen). Sie selbst können natürlich mit Ihren Beispieldaten machen, was Sie wollen; meinetwegen können Sie

sie auch löschen und die Skripte erneut ausführen, aber ich für meinen Teil werde darauf achten, dass die Daten intakt bleiben.

Wertebereichsbedingungen

Sie können in einer Bedingung nicht nur prüfen, ob ein Ausdruck gleich (oder ungleich) einem anderen Ausdruck ist, sondern auch, ob ein Ausdruck in einem bestimmten Wertebereich liegt. Diese Art von Bedingung findet man häufig im Umgang mit numerischen oder temporalen Daten. Betrachten Sie folgende Abfrage:

```
mysql> SELECT customer_id, rental_date

    -> FROM rental

    -> WHERE rental_date < '2005-05-25';

+-----+-------+

| customer_id | rental_date         |

+-----+-------+

|         130 | 2005-05-24 22:53:30 |

|         459 | 2005-05-24 22:54:33 |

|         408 | 2005-05-24 23:03:39 |

|         333 | 2005-05-24 23:04:41 |

|         222 | 2005-05-24 23:05:21 |

|         549 | 2005-05-24 23:08:07 |

|         269 | 2005-05-24 23:11:53 |
```

```
|         239 | 2005-05-24 23:31:46 |

+-----+-------+

8 rows in set (0.00 sec)
```

Diese Abfrage findet alle Ausleihvorgänge, die vor dem 25. Mai 2005 stattfanden. Doch eventuell möchten Sie nicht nur eine Obergrenze für das Ausleihdatum angeben, sondern auch eine Untergrenze:

```
mysql> SELECT customer_id, rental_date

    -> FROM rental

    -> WHERE rental_date <= '2005-06-16'

    ->   AND rental_date >= '2005-06-14';

+-----+-------+

| customer_id | rental_date         |

+-----+-------+

|         416 | 2005-06-14 22:53:33 |

|         516 | 2005-06-14 22:55:13 |

|         239 | 2005-06-14 23:00:34 |

|         285 | 2005-06-14 23:07:08 |

|         310 | 2005-06-14 23:09:38 |
```

```
|         592 | 2005-06-14 23:12:46 |

…

|         148 | 2005-06-15 23:20:26 |

|         237 | 2005-06-15 23:36:37 |

|         155 | 2005-06-15 23:55:27 |

|         341 | 2005-06-15 23:57:20 |

|         149 | 2005-06-15 23:58:53 |

+-----+-------+

364 rows in set (0.00 sec)
```

Diese Version der Abfrage findet alle Filme, die am 14. oder 15. Juni 2005 ausgeliehen wurden.

Der between-Operator

Wenn Sie *sowohl* eine obere *als auch* eine untere Grenze für Ihren Wertebereich haben, können Sie statt zwei separaten Bedingungen auch eine einzige Bedingung mit dem Operator `between` formulieren:

```
mysql> SELECT customer_id, rental_date

    -> FROM rental

    -> WHERE rental_date BETWEEN '2005-06-14' AND '2005-06-
    16';

+-----+-------+
```

```
| customer_id | rental_date         |
+-----+-------+
|         416 | 2005-06-14 22:53:33 |
|         516 | 2005-06-14 22:55:13 |
|         239 | 2005-06-14 23:00:34 |
|         285 | 2005-06-14 23:07:08 |
|         310 | 2005-06-14 23:09:38 |
|         592 | 2005-06-14 23:12:46 |
…
|         148 | 2005-06-15 23:20:26 |
|         237 | 2005-06-15 23:36:37 |
|         155 | 2005-06-15 23:55:27 |
|         341 | 2005-06-15 23:57:20 |
|         149 | 2005-06-15 23:58:53 |
+-----+-------+
364 rows in set (0.00 sec)
```

Bei der Verwendung des `between`-Operators sind einige Dinge zu beachten: Geben Sie immer zuerst (hinter `between`) die Untergrenze und als Zweites (hinter `and`) die Obergrenze des Wertebereichs an. Hier sehen Sie, was passiert, wenn Sie versehentlich zuerst die Obergrenze angeben:

```
mysql> SELECT customer_id, rental_date

    -> FROM rental

    -> WHERE rental_date BETWEEN '2005-06-16' AND '2005-06-
    14';

Empty set (0.00 sec)
```

Es werden keine Daten zurückgegeben, da der Server in Wirklichkeit aus Ihrer einen Bedingung zwei Bedingungen (mit den Operatoren <= und >=) macht, wie in:

```
SELECT customer_id, rental_date

    -> FROM rental

    -> WHERE rental_date >= '2005-06-16'

    ->   AND rental_date <= '2005-06-14'

Empty set (0.00 sec)
```

Da es unmöglich ein Datum geben kann, das sowohl größer als der 16. Juni 2005 als auch kleiner als der 14. Juni 2005 ist, gibt die Abfrage die leere Menge zurück. Und das bringt mich zur zweiten Falle von `between`: Denken Sie stets daran, dass Ober- und Untergrenze *inklusiv* sind, dass ihre Werte also im Wertebereich enthalten sind. In diesem Fall will ich alle Ausleihvorgänge vom 14. oder 15. Juni 2005 haben, also gebe ich `2005-06-14` als Unter- und `2005-06-16` als Obergrenze an. Da ich die Stunden-, Minuten- und Sekundenangaben des Datumswerts nicht festlege, wird die Zeit auf Mitternacht gelegt, und der

effektive Bereich ist 2005-06-14 00:00:00 bis 2005-06-16 00:00:00. Damit sind alle Ausleihvorgänge vom 14. oder 15. Juni eingeschlossen.

Neben Datumsintervallen können Bedingungen auch Zahlenintervalle angeben. Diese sind leicht verständlich, wie das folgende Beispiel zeigt:

```
mysql> SELECT customer_id, payment_date, amount

    -> FROM payment

    -> WHERE amount BETWEEN 10.0 AND 11.99;

+-----+-------+---+

| customer_id | payment_date        | amount |

+-----+-------+---+

|           2 | 2005-07-30 13:47:43 |  10.99 |

|           3 | 2005-07-27 20:23:12 |  10.99 |

|          12 | 2005-08-01 06:50:26 |  10.99 |

|          13 | 2005-07-29 22:37:41 |  11.99 |

|          21 | 2005-06-21 01:04:35 |  10.99 |

|          29 | 2005-07-09 21:55:19 |  10.99 |

…

|         571 | 2005-06-20 08:15:27 |  10.99 |

|         572 | 2005-06-17 04:05:12 |  10.99 |
```

```
|         573 | 2005-07-31 12:14:19 |  10.99 |

|         591 | 2005-07-07 20:45:51 |  11.99 |

|         592 | 2005-07-06 22:58:31 |  11.99 |

|         595 | 2005-07-31 11:51:46 |  10.99 |

+-----+-------+---+

114 rows in set (0.01 sec)
```

Alle Bezahlvorgänge zwischen $10,00 und $11,99 werden gemeldet. Denken Sie auch hier daran, den kleineren Betrag als Erstes anzugeben.

String-Bereiche

Während Datums- und Zahlenintervalle leicht zu verstehen sind, lassen sich Bedingungen, die nach String-Intervallen suchen, schon etwas schwerer veranschaulichen. Angenommen, Sie suchten nach Kunden, deren Nachname in einen bestimmten Bereich fällt. Diese Abfrage liefert alle Kunden, bei denen die Anfangsbuchstaben der Nachnamen in dem Bereich zwischen FA und FR liegen:

```
mysql> SELECT last_name, first_name

    -> FROM customer

    -> WHERE last_name BETWEEN 'FA' AND 'FR';

+----+----+

| last_name  | first_name |

+----+----+
```

FARNSWORTH	JOHN
FENNELL	ALEXANDER
FERGUSON	BERTHA
FERNANDEZ	MELINDA
FIELDS	VICKI
FISHER	CINDY
FLEMING	MYRTLE
FLETCHER	MAE
FLORES	JULIA
FORD	CRYSTAL
FORMAN	MICHEAL
FORSYTHE	ENRIQUE
FORTIER	RAUL
FORTNER	HOWARD
FOSTER	PHYLLIS
FOUST	JACK
FOWLER	JO

```
| FOX         | HOLLY      |

+----+----+

18 rows in set (0.00 sec)
```

Es gibt zwar fünf Kunden, deren Nachname mit FR beginnt, aber sie sind nicht in der Ergebnismenge enthalten, da ein Name wie FRANKLIN außerhalb des Bereichs liegt. Wir könnten aber vier der fünf Kunden mit aufnehmen, indem wir die rechte Bereichsgrenze auf FRB erweitern:

```
mysql> SELECT last_name, first_name

    -> FROM customer

    -> WHERE last_name BETWEEN 'FA' AND 'FRB';

+----+----+

| last_name  | first_name |

+----+----+

| FARNSWORTH | JOHN       |

| FENNELL    | ALEXANDER  |

| FERGUSON   | BERTHA     |

| FERNANDEZ  | MELINDA    |

| FIELDS     | VICKI      |

| FISHER     | CINDY      |
```

| FLEMING | MYRTLE |

| FLETCHER | MAE |

| FLORES | JULIA |

| FORD | CRYSTAL |

| FORMAN | MICHEAL |

| FORSYTHE | ENRIQUE |

| FORTIER | RAUL |

| FORTNER | HOWARD |

| FOSTER | PHYLLIS |

| FOUST | JACK |

| FOWLER | JO |

| FOX | HOLLY |

| FRALEY | JUAN |

| FRANCISCO | JOEL |

| FRANKLIN | BETH |

| FRAZIER | GLENDA |

+----+----+

```
22 rows in set (0.00 sec)
```

Um mit String-Bereichen arbeiten zu können, müssen Sie die Reihenfolge der Zeichen Ihres Zeichensatzes kennen (die Sortierreihenfolge der Zeichen eines Zeichensatzes nennt man auch *Kollation*).

Mitgliedschaftsbedingungen

Es gibt Fälle, in denen man einen Ausdruck nicht auf einen Einzelwert oder Wertebereich eingrenzen möchte, sondern auf eine endliche Wertemenge. Wenn Sie beispielsweise alle Filme finden möchten, deren Klassifizierung nach Altersempfehlungen entweder `'G'` oder `'PG'` ist, tun Sie Folgendes[1]:

```
mysql> SELECT title, rating

    -> FROM film

    -> WHERE rating = 'G' OR rating = 'PG';

+---------+---+

| title                    | rating |

+---------+---+

| ACADEMY DINOSAUR         | PG     |

| ACE GOLDFINGER           | G      |

| AFFAIR PREJUDICE         | G      |

| AFRICAN EGG              | G      |

| AGENT TRUMAN             | PG     |
```

```
| ALAMO VIDEOTAPE           | G      |

| ALASKA PHANTOM            | PG     |

| ALI FOREVER               | PG     |

| AMADEUS HOLY              | PG     |

…

| WEDDING APOLLO            | PG     |

| WEREWOLF LOLA             | G      |

| WEST LION                 | G      |

| WIZARD COLDBLOODED        | PG     |

| WON DARES                 | PG     |

| WONDERLAND CHRISTMAS      | PG     |

| WORDS HUNTER              | PG     |

| WORST BANGER              | PG     |

| YOUNG LANGUAGE            | G      |

+---------+---+

372 rows in set (0.00 sec)
```

Diese `where`-Klausel (zwei durch `or` verknüpfte Bedingungen) war zwar nicht schwer zu konstruieren, aber stellen Sie sich eine Menge von 10 oder 20

Ausdrücken vor! Für solche Situationen können Sie stattdessen den Operator `in` nutzen:

```
SELECT title, rating

FROM film

WHERE rating IN ('G','PG');
```

Mit dem `in`-Operator können Sie eine Menge von beliebig vielen Ausdrücken in einer einzigen Bedingung unterbringen.

Unterabfragen

Sie können nicht nur selbst eine Menge von Ausdrücken wie beispielsweise `('G', 'PG')` schreiben, sondern auch eine solche Menge direkt von einer Unterabfrage erzeugen lassen. Vielleicht gehen Sie ja davon aus, dass jeder Film, der den String `'PET'` enthält, ein Kinderfilm ist. Dann könnten Sie eine Unterabfrage auf der `film`-Tabelle ausführen, um alle Bewertungen zu diesen Filmen abzurufen und damit dann alle Filme mit diesen Bewertungen auszugeben:

```
mysql> SELECT title, rating

    -> FROM film

    -> WHERE rating IN (SELECT rating FROM film WHERE title
    LIKE '%PET%');

+---------+---+

| title                    | rating |

+---------+---+

| ACADEMY DINOSAUR         | PG     |
```

| ACE GOLDFINGER | G |

| AFFAIR PREJUDICE | G |

| AFRICAN EGG | G |

| AGENT TRUMAN | PG |

| ALAMO VIDEOTAPE | G |

| ALASKA PHANTOM | PG |

| ALI FOREVER | PG |

| AMADEUS HOLY | PG |

…

| WEDDING APOLLO | PG |

| WEREWOLF LOLA | G |

| WEST LION | G |

| WIZARD COLDBLOODED | PG |

| WON DARES | PG |

| WONDERLAND CHRISTMAS | PG |

| WORDS HUNTER | PG |

| WORST BANGER | PG |

```
| YOUNG LANGUAGE             | G      |

+---------+---+

372 rows in set (0.00 sec)
```

Die Unterabfrage gibt eine Menge mit `'G'` und `'PG'` zurück, und die Hauptabfrage schaut nach, ob die Spalte `rating` in dieser Rückgabemenge zu finden ist.

Verwendung von not in

Manchmal möchte man wissen, ob ein bestimmter Ausdruck in einer Menge von Ausdrücken vorhanden ist, manchmal aber auch, ob er darin *nicht* vorhanden ist. In solchen Situationen leistet der Operator `not in` gute Dienste:

```
SELECT title, rating

FROM film

WHERE rating NOT IN ('PG-13','R', 'NC-17');
```

Diese Abfrage findet alle Konten, die *nicht* mit `'PG-13'`, `'R'` oder `'NC-17'` bewertet wurden, und liefert damit die gleichen 372 Zeilen wie die vorige Abfrage.

Bedingungen abgleichen

Bisher haben Sie Bedingungen zu sehen bekommen, die einen genauen String, einen String-Bereich oder eine String-Menge identifizieren. Es gibt jedoch noch einen letzten Bedingungstyp, der teilweise Übereinstimmungen von Strings erkennen kann. Sie können zum Beispiel alle Kunden ausfindig machen, deren Nachname mit »Q« beginnt. Mit einer eingebauten Funktion können Sie den ersten Buchstaben der Spalte `last_name` herausziehen, wie in folgender Anweisung:

```
mysql> SELECT last_name, first_name
```

```
    -> FROM customer

    -> WHERE left(last_name, 1) = 'Q';

+-----+----+

| last_name   | first_name |

+-----+----+

| QUALLS      | STEPHEN    |

| QUINTANILLA | ROGER      |

| QUIGLEY     | TROY       |

+-----+----+

3 rows in set (0.00 sec)
```

Die eingebaute Funktion `left()` kann dies zwar tun, gibt Ihnen aber wenig Flexibilität. Stattdessen können Sie auch Suchausdrücke mit Wildcard-Zeichen formulieren, wie es im nächsten Abschnitt beschrieben wird.

Verwendung von Wildcards

Wenn Sie sich für teilweise Übereinstimmungen von Strings interessieren, könnte Sie Folgendes interessieren:

- Strings mit einem bestimmten Anfangs-/Endbuchstaben
- Strings, die mit einem bestimmten Teilstring anfangen oder enden
- Strings, in denen irgendwo ein bestimmtes Zeichen vorkommt
- Strings, in denen irgendwo ein bestimmter Teilstring vorkommt
- Strings, die unabhängig von den enthaltenen Zeichen ein bestimmtes Format aufweisen

Mit bestimmten Suchausdrücken können Sie diese und andere Teilübereinstimmungen von Strings mithilfe der Wildcard-Zeichen aus Tabelle 4-4 ausfindig machen.

Tabelle 4-4: Wildcard-Zeichen

Wildcard-Zeichen	Passt auf
_	genau ein Zeichen
%	beliebig viele Zeichen (einschließlich null)

Der Unterstrich ist Platzhalter für ein einzelnes und das Prozentzeichen für beliebig viele Zeichen. Wenn Sie Bedingungen mit solchen Suchausdrücken formulieren, verwenden Sie den `like`-Operator:

```
mysql> SELECT last_name, first_name

    -> FROM customer

    -> WHERE last_name LIKE '_A_T%S';

+----+----+

| last_name | first_name |

+----+----+

| MATTHEWS  | ERICA      |

| WALTERS   | CASSANDRA  |

| WATTS     | SHELLY     |

+----+----+

3 rows in set (0.00 sec)
```

Der Suchausdruck im obigen Beispiel spezifiziert Strings, die an zweiter Stelle ein »A« und an vierter Stelle ein »T« aufweisen, worauf beliebig viele Zeichen folgen – aber am Ende muss ein »S« stehen. Tabelle 4-5 zeigt einige weitere Suchausdrücke zusammen mit ihren Interpretationen.

Tabelle 4-5: Beispiele für Suchausdrücke

Suchausdruck	Interpretation
F%	Strings, die mit »F« anfangen
%t	Strings, die mit »t« aufhören
%bas%	Strings, die den Teilstring »bas« enthalten
_ _t_	vier Zeichen lange Strings mit einem »t« an dritter Position
_ _ _-_ _-_ _ _ _	elf Zeichen lange Strings mit Bindestrichen an der vierten und siebten Position

Die Wildcard-Zeichen funktionieren gut für einfache Suchausdrücke, aber wenn es etwas komplizierter wird, können Sie auch mehrere Suchausdrücke miteinander verbinden:

```
mysql> SELECT last_name, first_name

    -> FROM customer

    -> WHERE last_name LIKE 'Q%' OR last_name LIKE 'Y%';

+-----+----+

| last_name   | first_name |

+-----+----+

| QUALLS      | STEPHEN    |

| QUIGLEY     | TROY       |
```

```
| QUINTANILLA | ROGER      |

| YANEZ       | LUIS       |

| YEE         | MARVIN     |

| YOUNG       | CYNTHIA    |

+-----+----+

6 rows in set (0.00 sec)
```

Diese Abfrage findet alle Kunden, deren Nachnamen mit »Q« oder »Y« anfangen.

Verwendung von regulären Ausdrücken

Wenn Ihnen Wildcards nicht flexibel genug sind, können Sie Suchausdrücke mit regulären Ausdrücken konstruieren. Ein regulärer Ausdruck ist im Wesentlichen ein Suchausdruck auf Steroiden. Wenn Sie SQL-Neuling sind, aber bereits mit Sprachen wie Perl programmiert haben, kennen Sie sich mit regulären Ausdrücken bestens aus. Sollten Sie jedoch noch nie mit ihnen zu tun gehabt haben, lesen Sie am besten *Reguläre Ausdrücke* von Jeffrey Friedl (O'Reilly), da dieses Thema viel zu umfangreich ist, um es in diesem Buch behandeln zu können.

Die obige Abfrage (alle Kunden zu finden, deren Nachnamen mit »Q« oder »Y« anfangen) würde in der MySQL-Implementierung von regulären Ausdrücken folgendermaßen aussehen:

```
mysql> SELECT last_name, first_name

    -> FROM customer

    -> WHERE last_name REGEXP '^[QY]';

+-----+----+

| last_name   | first_name |
```

```
+-----+----+

| YOUNG       | CYNTHIA    |

| QUALLS      | STEPHEN    |

| QUINTANILLA | ROGER      |

| YANEZ       | LUIS       |

| YEE         | MARVIN     |

| QUIGLEY     | TROY       |

+-----+----+

6 rows in set (0.16 sec)
```

Der Operator `regexp` nimmt einen regulären Ausdruck (hier `'^[QY]'`) und wendet ihn auf den Ausdruck auf der linken Seite der Bedingung an (die Spalte `last_name`). Jetzt enthält die Abfrage eine einzelne Bedingung mit einem regulären Ausdruck anstelle zweier mit Wildcards.

Oracle Database und Microsoft SQL Server unterstützen ebenfalls reguläre Ausdrücke. Bei Oracle Database verwendet man die Funktion `regexp_like` anstelle des Operators `regexp` und bei SQL Server den Operator `like`.

NULL: ein böses Wort

Ich habe mich darum gedrückt, solange ich konnte, aber jetzt lässt es sich nicht mehr umgehen, dieses Thema, das Unsicherheit, Angst und Entsetzen hervorruft: der `null`-Wert. `null` ist die Abwesenheit eines Werts. Bevor ein Angestellter gefeuert wird, wird seine `end_date`-Spalte in der `employee`-Tabelle auf `null` gesetzt. Es gibt einfach keinen Wert, den man der Spalte `end_date` in dieser Situation vernünftigerweise zuweisen könnte. `null` ist jedoch eine wackelige Angelegenheit, da es mehrere Varianten davon gibt:

Nicht zutreffend

Zum Beispiel die Employee-ID-Spalte für ein am Geldautomaten getätigtes Bankgeschäft.

Wert noch nicht bekannt

Wenn zum Beispiel zu dem Zeitpunkt, da die Daten eines Kunden angelegt werden, dessen Federal ID noch unbekannt ist.

Wert nicht definiert

Wenn zum Beispiel ein Konto für ein Produkt angelegt wird, das noch nicht in die Datenbank eingetragen wurde.

Manche Theoretiker sind der Ansicht, es müsse für alle diese (und andere) Situationen jeweils unterschiedliche Ausdrücke geben, aber die meisten Praktiker sind sich einig, dass das viel zu verwirrend wäre.

Bei der Arbeit mit `null` müssen Sie an Folgendes denken:

- Ein Ausdruck kann null *sein*, aber nicht *gleich* null.
- null und null sind nie gleich.

Ob ein Ausdruck `null` ist, kann man mit dem Operator `is null` testen:

```
mysql> SELECT rental_id, customer_id

    -> FROM rental

    -> WHERE return_date IS NULL;

+----+-----+

| rental_id | customer_id |

+----+-----+

|     11496 |         155 |
```

```
|     11541 |         335 |

|     11563 |          83 |

|     11577 |         219 |

|     11593 |          99 |

…

|     15867 |         505 |

|     15875 |          41 |

|     15894 |         168 |

|     15966 |         374 |

+-----+-----+

183 rows in set (0.01 sec)
```

Diese Abfrage liefert alle Ausleihvorgänge, bei denen Filme nie zurückgegeben wurden. Hier sehen Sie die gleiche Abfrage mit `= null` anstelle von `is null`:

```
mysql> SELECT rental_id, customer_id

    -> FROM rental

    -> WHERE return_date = NULL;

Empty set (0.01 sec)
```

Die Abfrage wird korrekt geparst und ausgeführt, gibt aber keine Zeilen zurück. Unerfahrenen SQL-Programmierern unterläuft häufig dieser Ausrutscher, und der Datenbankserver meldet dann keinen Fehler. Seien Sie also vorsichtig bei der Formulierung von Bedingungen, die `null` testen sollen.

Wenn Sie sehen möchten, ob einer Spalte ein Wert zugewiesen wurde, verwenden Sie den Operator `is not null`:

```
mysql> SELECT rental_id, customer_id, return_date

    -> FROM rental

    -> WHERE return_date IS NOT NULL;

+-----+-----+-------+

| rental_id | customer_id | return_date         |

+-----+-----+-------+

|         1 |         130 | 2005-05-26 22:04:30 |

|         2 |         459 | 2005-05-28 19:40:33 |

|         3 |         408 | 2005-06-01 22:12:39 |

|         4 |         333 | 2005-06-03 01:43:41 |

|         5 |         222 | 2005-06-02 04:33:21 |

|         6 |         549 | 2005-05-27 01:32:07 |

|         7 |         269 | 2005-05-29 20:34:53 |

…
```

```
|     16043 |         526 | 2005-08-31 03:09:03 |

|     16044 |         468 | 2005-08-25 04:08:39 |

|     16045 |          14 | 2005-08-25 23:54:26 |

|     16046 |          74 | 2005-08-27 18:02:47 |

|     16047 |         114 | 2005-08-25 02:48:48 |

|     16048 |         103 | 2005-08-31 21:33:07 |

|     16049 |         393 | 2005-08-30 01:01:12 |

+-----+-----+-------+

15861 rows in set (0.02 sec)
```

Diese Version der Abfrage liefert alle Ausleihvorgänge, bei denen die Filme zurückgegeben wurden, was den Großteil der Zeilen in der Tabelle betrifft (15.861 von 16.044).

Bevor wir das Thema `null` wieder verlassen, möchte ich eine weitere mögliche Falle aufzeigen. Angenommen, Sie müssten alle Ausleihvorgänge finden, die nicht zwischen Mai und August 2005 zurückgegeben wurden. Instinktiv tun Sie zunächst Folgendes:

```
mysql> SELECT rental_id, customer_id, return_date

    -> FROM rental

    -> WHERE return_date NOT BETWEEN '2005-05-01' AND '2005-
    09-01';

+-----+-----+-------+
```

```
| rental_id | customer_id | return_date         |
+-----+-----+-------+
|     15365 |         327 | 2005-09-01 03:14:17 |
|     15388 |          50 | 2005-09-01 03:50:23 |
|     15392 |         410 | 2005-09-01 01:14:15 |
|     15401 |         103 | 2005-09-01 03:44:10 |
|     15415 |         204 | 2005-09-01 02:05:56 |
…
|     15977 |         550 | 2005-09-01 22:12:10 |
|     15982 |         370 | 2005-09-01 21:51:31 |
|     16005 |         466 | 2005-09-02 02:35:22 |
|     16020 |         311 | 2005-09-01 18:17:33 |
|     16033 |         226 | 2005-09-01 02:36:15 |
|     16037 |          45 | 2005-09-01 02:48:04 |
|     16040 |         195 | 2005-09-02 02:19:33 |
+-----+-----+-------+
62 rows in set (0.01 sec)
```

Es stimmt zwar, dass diese 62 Ausleihvorgänge außerhalb unseres Zeitfensters von Mai bis August beendet wurden, aber wenn Sie die Daten genau anschauen, erkennen Sie, dass alle Zeilen im `return_date` nicht `null` sind. Was ist aber mit den 183 Ausleihvorgängen, deren Filme nie zurückgegeben wurden? Man kann durchaus sagen, dass diese 183 Zeilen ebenfalls in der Ergebnismenge enthalten sein sollten, weil die Filme nicht zwischen Mai und August zurückkamen. Um also die korrekte Antwort zu finden, müssen Sie auch die Möglichkeit einbeziehen, dass manche Zeilen in der Spalte `return_date` den Wert `null` aufweisen:

```
mysql> SELECT rental_id, customer_id, return_date

    -> FROM rental

    -> WHERE return_date IS NULL

    ->   OR return_date NOT BETWEEN '2005-05-01' AND '2005-09-
    01';

+----+-----+-------+

| rental_id | customer_id | return_date         |

+----+-----+-------+

|     11496 |         155 | NULL                |

|     11541 |         335 | NULL                |

|     11563 |          83 | NULL                |

|     11577 |         219 | NULL                |

|     11593 |          99 | NULL                |
```

…

```
|     15939 |         382 | 2005-09-01 17:25:21 |
|     15942 |         210 | 2005-09-01 18:39:40 |
|     15966 |         374 | NULL                |
|     15971 |         187 | 2005-09-02 01:28:33 |
|     15973 |         343 | 2005-09-01 20:08:41 |
|     15977 |         550 | 2005-09-01 22:12:10 |
|     15982 |         370 | 2005-09-01 21:51:31 |
|     16005 |         466 | 2005-09-02 02:35:22 |
|     16020 |         311 | 2005-09-01 18:17:33 |
|     16033 |         226 | 2005-09-01 02:36:15 |
|     16037 |          45 | 2005-09-01 02:48:04 |
|     16040 |         195 | 2005-09-02 02:19:33 |
+-------+-----+-------+

245 rows in set (0.01 sec)
```

Nun enthält die Ergebnismenge die 62 Ausleihvorgänge, bei denen die Filme außerhalb des Fensters von Mai bis August zurückgegeben wurden, und die 183 Fälle, bei denen die Filme nie zurückkamen, was zusammen 245 Zeilen ergibt.

Wenn Sie mit unbekannten Datenbanken arbeiten, sollten Sie immer zuerst herausfinden, welche Spalten einer Tabelle `null`-Werte erlauben. So können Sie in Ihren Filterbedingungen Vorkehrungen treffen, damit Ihnen keine Daten durchrutschen.

Testen Sie Ihr Wissen

Die folgenden Übungen prüfen, ob Sie Filterbedingungen verstanden haben. Die Lösungen finden Sie in Anhang B.

Für die ersten beiden Übungen wird die folgende Untermenge an Zeilen aus der `payment`-Tabelle verwendet:

```
+----+-----+---+-------+

| payment_id | customer_id | amount | date(payment_date) |

+----+-----+---+-------+

|        101 |           4 |   8.99 | 2005-08-18         |

|        102 |           4 |   1.99 | 2005-08-19         |

|        103 |           4 |   2.99 | 2005-08-20         |

|        104 |           4 |   6.99 | 2005-08-20         |

|        105 |           4 |   4.99 | 2005-08-21         |

|        106 |           4 |   2.99 | 2005-08-22         |

|        107 |           4 |   1.99 | 2005-08-23         |

|        108 |           5 |   0.99 | 2005-05-29         |

|        109 |           5 |   6.99 | 2005-05-31         |
```

```
|        110 |           5 |   1.99 | 2005-05-31          |
|        111 |           5 |   3.99 | 2005-06-15          |
|        112 |           5 |   2.99 | 2005-06-16          |
|        113 |           5 |   4.99 | 2005-06-17          |
|        114 |           5 |   2.99 | 2005-06-19          |
|        115 |           5 |   4.99 | 2005-06-20          |
|        116 |           5 |   4.99 | 2005-07-06          |
|        117 |           5 |   2.99 | 2005-07-08          |
|        118 |           5 |   4.99 | 2005-07-09          |
|        119 |           5 |   5.99 | 2005-07-09          |
|        120 |           5 |   1.99 | 2005-07-09          |
+----+-----+---+-------+
```

Übung 4-1

Welche Payment-IDs würden durch folgende Filterbedingungen zurückgegeben?

```
customer_id <> 5 AND (amount > 8 OR date(payment_date) =
'2005-08-23')
```

Übung 4-2

Welche Payment-IDs würden durch diese Filterbedingungen zurückgegeben?

```
customer_id = 5 AND NOT (amount > 6 OR date(payment_date) =
'2005-06-19')
```

Übung 4-3

Erstellen Sie eine Abfrage, die alle Zeilen aus der `payment`-Tabelle liefert, bei denen der Betrag entweder 1,98, 7,98 oder 9,98 ist.

Übung 4-4

Erstellen Sie eine Abfrage, die alle Kunden findet, deren Nachname an zweiter Stelle ein »A« und dann irgendwo nach dem »A« ein »W« enthält.

Index

Symbole

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z