KAPITEL 3

Datenbankabfragen

In den ersten beiden Kapiteln dieses Buchs haben Sie hier und dort bereits ein paar Beispiele für Datenbankabfragen (`select`-Anweisungen) gesehen. Nun ist es an der Zeit, sich die verschiedenen Teile der `select`-Anweisung und ihre Wechselbeziehungen einmal genauer anzuschauen. Am Ende dieses Kapitels sollten Sie ein grundlegendes Verständnis für das Auslesen, Verknüpfen, Filtern, Gruppieren und Sortieren von Daten haben – diese Themen werden dann im Detail in den Kapiteln Kapitel 4 bis Kapitel 10 behandelt.

Die Mechanik von Abfragen

Bevor wir die `select`-Anweisung in ihre Einzelteile zerlegen, ist es vielleicht interessant zu sehen, wie Abfragen vom MySQL-Server (oder jedem x-beliebigen Datenbankserver) überhaupt ausgeführt werden. Wenn Sie das `mysql`-Kommandozeilentool verwenden (und davon gehe ich aus), haben Sie sich bereits mit Ihrem Benutzernamen und zugehörigem Passwort in den MySQL-Server eingeloggt (und möglicherweise auch mit einem Hostnamen, wenn der MySQL-Server auf einem anderen Computer läuft). Sobald der Server die Richtigkeit Ihrer Anmeldedaten geprüft hat, wird eine *Datenbankverbindung* für Sie eingerichtet. Diese Verbindung wird von der Anwendung gehalten, die sie angefordert hatte (in diesem Fall das `mysql`-Tool), bis entweder die Anwendung die Verbindung freigibt (wenn Sie `quit` eingeben) oder der Server selbst die Verbindung trennt (wenn er heruntergefahren wird). Jeder Verbindung mit dem MySQL-Server wird ein Identifier zugewiesen, den Sie beim Einloggen auch sehen:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 8.0.15 MySQL Community Server - GPL

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

Hier ist meine Verbindungs-ID die 11. Diese Information kann für den Datenbankadministrator bedeutungsvoll sein, wenn es Schwierigkeiten gibt, weil beispielsweise eine schlecht geformte Abfrage stundenlang läuft und man sie lieber abbrechen möchte.

Sobald der Server den Benutzernamen und das Passwort geprüft und Ihnen eine Verbindung zugeteilt hat, können Sie Abfragen ausführen (und natürlich auch andere SQL-Anweisungen). Immer wenn eine Abfrage an den Server geschickt wird, prüft dieser vor der Ausführung Folgendes:

- Haben Sie die Berechtigung, diese Anweisung auszuführen?
- Haben Sie die Berechtigung, auf die gewünschten Daten zuzugreifen?
- Ist die Syntax Ihrer Anweisung korrekt?

Wenn Ihre Anweisung diese drei Prüfungen besteht, wird sie an die *Abfrageoptimierung* übergeben, die den effizientesten Ausführungspfad für sie ermittelt. Die Optimierung fragt beispielsweise, in welcher Reihenfolge die in Ihrer `from`-Klausel genannten Tabellen verbunden werden und welche Indizes zur Verfügung stehen. Dann sucht sie einen *Ausführungsplan* aus, der anschließend vom Server bei der Bearbeitung Ihrer Abfrage eingehalten wird.

Viele interessieren sich für das faszinierende Thema, wie der Datenbankserver Ausführungspläne wählt und wie man diese Wahl beeinflussen kann. Wer MySQL nutzt, kann dies in *High Performance MySQL* (O'Reilly Verlag) von Baron Schwartz und anderen nachlesen. Darin lernen Sie unter anderem, wie man Indizes generiert, Ausführungspläne analysiert, die Optimierung durch Abfragehinweise beeinflusst und die Startparameter des Servers tunt. Wenn Sie Oracle Database oder SQL Server einsetzen, haben Sie die Auswahl aus zig Büchern über Datenbank-Tuning.

Wenn der Server die Abfrage fertig ausgeführt hat, gibt er der aufrufenden Anwendung eine *Ergebnismenge* zurück (in diesem Fall wieder dem `mysql`-Tool). Wie in Kapitel 1 bereits gesagt, ist eine Ergebnismenge auch nur eine Tabelle mit Zeilen und Spalten. Scheitert Ihre Abfrage, zeigt Ihnen das `mysql`-Tool die gleiche Meldung wie am Ende des folgenden Beispiels:

```
mysql> SELECT first_name, last_name
    -> FROM customer
    -> WHERE last_name = 'ZIEGLER';
Empty set (0.02 sec)
```

Gibt die Abfrage eine oder mehrere Zeilen zurück, formatiert das `mysql`-Tool sie, indem es Spaltenüberschriften angibt und mit den Symbolen –, | und + Kästen um die Spalten zeichnet:

```
mysql> SELECT *
    -> FROM category;
```

```
+-------------+-------------+---------------------+
| category_id | name        | last_update         |
+-------------+-------------+---------------------+
|           1 | Action      | 2006-02-15 04:46:27 |
|           2 | Animation   | 2006-02-15 04:46:27 |
|           3 | Children    | 2006-02-15 04:46:27 |
|           4 | Classics    | 2006-02-15 04:46:27 |
|           5 | Comedy      | 2006-02-15 04:46:27 |
|           6 | Documentary | 2006-02-15 04:46:27 |
|           7 | Drama       | 2006-02-15 04:46:27 |
|           8 | Family      | 2006-02-15 04:46:27 |
|           9 | Foreign     | 2006-02-15 04:46:27 |
|          10 | Games       | 2006-02-15 04:46:27 |
|          11 | Horror      | 2006-02-15 04:46:27 |
|          12 | Music       | 2006-02-15 04:46:27 |
|          13 | New         | 2006-02-15 04:46:27 |
|          14 | Sci-Fi      | 2006-02-15 04:46:27 |
|          15 | Sports      | 2006-02-15 04:46:27 |
|          16 | Travel      | 2006-02-15 04:46:27 |
+-------------+-------------+---------------------+
16 rows in set (0.02 sec)
```

Diese Abfrage gibt alle Spalten aller Zeilen aus der `category`-Tabelle zurück. Nachdem die letzte Datenzeile angezeigt wurde, zeigt das `mysql`-Tool an, wie viele Zeilen zurückgeliefert wurden. Hier sind es 16.

Abfrageklauseln

Die `select`-Anweisung besteht aus mehreren Komponenten oder *Klauseln*. In MySQL ist nur eine von ihnen obligatorisch (nämlich die `select`-Klausel), aber normalerweise werden mindestens zwei oder drei der sechs möglichen Klauseln zusätzlich verwendet. Tabelle 3-1 zeigt die verschiedenen Klauseln und ihre Zwecke.

Tabelle 3-1: Abfrageklauseln

Name der Klausel	Zweck
`select`	Legt fest, welche Spalten in die Ergebnismenge kommen.
`from`	Sagt, aus welchen Tabellen die Daten genommen und wie die Tabellen verbunden werden sollen.
`where`	Filtert unerwünschte Daten heraus.
`group by`	Fasst Zeilen nach gemeinsamen Spaltenwerten zusammen.
`having`	Filtert unerwünschte Gruppen heraus.
`order by`	Sortiert die Zeilen der Ergebnismenge nach einer oder mehr Spalten.

Alle Klauseln aus Tabelle 3-1 sind auch in der ANSI-Spezifikation erwähnt. Die folgenden Abschnitte beschreiben nur diese sechs wichtigsten Klauseln.

Die select-Klausel

Obwohl die `select`-Klausel die erste Klausel jeder `select`-Anweisung ist, wird sie vom Datenbankserver als eine der letzten ausgewertet. Denn ehe Sie bestimmen können, was die Ergebnismenge tatsächlich enthalten wird, müssen Sie herausfinden, was sie enthalten *könnte*. Um die Rolle der `select`-Klausel genau zu verstehen, müssen Sie daher zunächst die `from`-Klausel besser kennenlernen. Hier ist zunächst einmal eine Abfrage:

```
mysql> SELECT *
    -> FROM language;
+-------------+----------+---------------------+
| language_id | name     | last_update         |
+-------------+----------+---------------------+
|           1 | English  | 2006-02-15 05:02:19 |
|           2 | Italian  | 2006-02-15 05:02:19 |
|           3 | Japanese | 2006-02-15 05:02:19 |
|           4 | Mandarin | 2006-02-15 05:02:19 |
|           5 | French   | 2006-02-15 05:02:19 |
|           6 | German   | 2006-02-15 05:02:19 |
+-------------+----------+---------------------+
6 rows in set (0.03 sec)
```

In dieser Abfrage listet die `from`-Klausel eine einzige Tabelle auf (`language`), und die `select`-Klausel besagt, dass *alle* Spalten (gekennzeichnet durch *) der `language`-Tabelle in die Ergebnismenge aufgenommen werden. Diese Abfrage könnte man in Worten wie folgt formulieren:

> *Zeige mir alle Spalten der* `language`*-Tabelle.*

Sie können nicht nur durch das Sternchensymbol alle Spalten auswählen, sondern auch explizit die Spalten aufführen, für die Sie sich interessieren:

```
mysql> SELECT language_id, name, last_update
    -> FROM language;
+-------------+----------+---------------------+
| language_id | name     | last_update         |
+-------------+----------+---------------------+
|           1 | English  | 2006-02-15 05:02:19 |
|           2 | Italian  | 2006-02-15 05:02:19 |
|           3 | Japanese | 2006-02-15 05:02:19 |
|           4 | Mandarin | 2006-02-15 05:02:19 |
|           5 | French   | 2006-02-15 05:02:19 |
|           6 | German   | 2006-02-15 05:02:19 |
+-------------+----------+---------------------+
6 rows in set (0.00 sec)
```

Das Ergebnis ist dasselbe wie bei der ersten Abfrage, da alle Spalten der `language`-Tabelle (`language_id`, `name` und `last_update`) in der `select`-Klausel benannt werden. Sie können jedoch auch nur eine Teilmenge der `language`-Tabellenspalten betrachten:

```
mysql> SELECT name
    -> FROM language;
+----------+
| name     |
+----------+
| English  |
| Italian  |
| Japanese |
| Mandarin |
| French   |
| German   |
+----------+
6 rows in set (0.00 sec)
```

Die Aufgabe der `select`-Klausel könnte man also folgendermaßen definieren:

> Die `select`-Klausel stellt fest, welche von allen möglichen Spalten in die Ergebnismenge der Abfrage aufgenommen werden.

Wenn Sie nur Spalten von Tabellen auswählen könnten, die in der `from`-Klausel auftauchen, wäre das kein großes Kunststück. Doch interessant wird es, wenn Sie in die `select`-Klausel Dinge wie diese aufnehmen:

- Literale, wie beispielsweise Zahlen oder Strings
- Ausdrücke, wie `transaction.amount * -1`
- Aufrufe eingebauter Funktionen, wie etwa `ROUND(transaction.amount, 2)`
- Aufrufe benutzerdefinierter Funktionen

Die nächste Abfrage demonstriert, wie man eine Tabellenspalte, ein Literal, einen Ausdruck und eine eingebaute Funktion in eine einzige Abfrage der `language`-Tabelle einfließen lässt:

```
mysql> SELECT language_id,
    ->   'COMMON' language_usage,
    ->   language_id * 3.1415927 lang_pi_value,
    ->   upper(name) language_name
    -> FROM language;
+-------------+----------------+---------------+---------------+
| language_id | language_usage | lang_pi_value | language_name |
+-------------+----------------+---------------+---------------+
|           1 | COMMON         |     3.1415927 | ENGLISH       |
|           2 | COMMON         |     6.2831854 | ITALIAN       |
|           3 | COMMON         |     9.4247781 | JAPANESE      |
|           4 | COMMON         |    12.5663708 | MANDARIN      |
|           5 | COMMON         |    15.7079635 | FRENCH        |
|           6 | COMMON         |    18.8495562 | GERMAN        |
+-------------+----------------+---------------+---------------+
6 rows in set (0.04 sec)
```

Ausdrücke und eingebaute Funktionen werden weiter unten genauer erklärt. Ich wollte Ihnen jedoch bereits jetzt ein Gefühl dafür vermitteln, was man in einer `select`-Klausel alles unterbringen kann. Wenn Sie nur eine eingebaute Funktion

ausführen oder einen einfachen Ausdruck auswerten müssen, können Sie die `from`-Klausel ganz beiseitelassen. Ein Beispiel:

```
mysql> SELECT version(),
    ->   user(),
    ->   database();
+-----------+----------------+------------+
| version() | user()         | database() |
+-----------+----------------+------------+
| 8.0.15    | root@localhost | sakila     |
+-----------+----------------+------------+
1 row in set (0.00 sec)
```

Da diese Abfrage nur drei eingebaute Funktionen aufruft und keine Daten aus irgendwelchen Tabellen abholt, ist keine `from`-Klausel erforderlich.

Spaltenaliase

Obwohl das `mysql`-Tool für die Rückgabespalten der Abfragen Überschriften generiert, können Sie diesen auch eigene Beschriftungen zuweisen. Es mag vielleicht seltener vorkommen, dass Sie einer Spalte, die einen unzutreffenden oder mehrdeutigen Bezeichner hat, einen eigenen Namen geben möchten, aber fast immer werden Sie dies tun, wenn Ihre Ergebnismenge Spalten enthält, die durch Ausdrücke oder eingebaute Funktionen generiert wurden. Einen neuen Namen vergeben Sie als *Spaltenalias* hinter den einzelnen Elementen der `select`-Klausel. Hier sehen Sie die vorige Abfrage der `language`-Tabelle mit Spaltenaliasen für drei Spalten:

```
mysql> SELECT language_id,
    ->   'COMMON' language_usage,
    ->   language_id * 3.1415927 lang_pi_value,
    ->   upper(name) language_name
    -> FROM language;
+-------------+----------------+---------------+---------------+
| language_id | language_usage | lang_pi_value | language_name |
+-------------+----------------+---------------+---------------+
|           1 | COMMON         |     3.1415927 | ENGLISH       |
|           2 | COMMON         |     6.2831854 | ITALIAN       |
|           3 | COMMON         |     9.4247781 | JAPANESE      |
|           4 | COMMON         |    12.5663708 | MANDARIN      |
|           5 | COMMON         |    15.7079635 | FRENCH        |
|           6 | COMMON         |    18.8495562 | GERMAN        |
+-------------+----------------+---------------+---------------+
6 rows in set (0.04 sec)
```

Wenn Sie nun die Überschriften betrachten, erkennen Sie, dass die Spaltenaliase `language_usage`, `lang_pi_value` und `language_name` nach der zweiten, dritten und vierten Spalte angegeben wurden. Wahrscheinlich stimmen Sie mir zu, dass die Ausgabe durch die Aliase leichter zu verstehen ist und dass man auch im Programm leichter mit ihr umgehen könnte, wenn man die Abfrage mit Java oder Python absetzte anstatt interaktiv mit dem `mysql`-Tool. Damit die Spaltenaliase noch

etwas besser erkennbar sind, können Sie vor dem Aliasnamen auch das Schlüsselwort as angeben, wie Sie es in folgendem Beispiel sehen:

```
mysql> SELECT language_id,
    ->   'COMMON' AS language_usage,
    ->   language_id * 3.1415927 AS lang_pi_value,
    ->   upper(name) AS language_name
    -> FROM language;
```

Als Grund für die Verwendung des Schlüsselworts as wird häufig genannt, dass es die Lesbarkeit verbessert. Trotzdem habe ich mich dafür entschieden, es in den Beispielen in diesem Buch nicht zu verwenden.

Duplikate entfernen

In manchen Fällen kann eine Abfrage doppelte Datenzeilen zurückliefern. Wenn Sie zum Beispiel die IDs aller Schauspieler abfragen wollten, die in einem Film auftauchen, erhielten Sie Folgendes:

```
mysql> SELECT actor_id FROM film_actor ORDER BY actor_id;
+----------+
| actor_id |
+----------+
|        1 |
|        1 |
|        1 |
|        1 |
|        1 |
|        1 |
|        1 |
|        1 |
|        1 |
|        1 |
...
|      200 |
|      200 |
|      200 |
|      200 |
|      200 |
|      200 |
|      200 |
|      200 |
|      200 |
+----------+
5462 rows in set (0.01 sec)
```

Da manche Schauspieler in mehr als einem Film mitspielen, steht dort auch mehrfach die gleiche Schauspieler-ID. In diesem Fall möchten Sie jedoch vermutlich die Menge der unterschiedlichen Schauspieler sehen, anstatt die Schauspieler-ID für jeden Film anzeigen zu lassen, in dem sie vorkommen. Dies erzielen Sie, indem Sie direkt hinter das `select` das Schlüsselwort `distinct` setzen:

```
mysql> SELECT DISTINCT actor_id FROM film_actor ORDER BY actor_id;
+----------+
| actor_id |
+----------+
|        1 |
|        2 |
|        3 |
|        4 |
|        5 |
|        6 |
|        7 |
|        8 |
|        9 |
|       10 |
...
|      192 |
|      193 |
|      194 |
|      195 |
|      196 |
|      197 |
|      198 |
|      199 |
|      200 |
+----------+
200 rows in set (0.01 sec)
```

Die neue Ergebnismenge enthält nun 200 Zeilen (eine pro Schauspieler) anstatt 5462 (eine pro Auftauchen eines Schauspielers in einem Film).

Wollen Sie einfach eine Liste aller Schauspieler haben, können Sie auch die Tabelle `actor` abfragen, statt alle Zeilen in `film_actor` auszulesen und die Doubletten zu entfernen.

Wenn Sie nicht möchten, dass der Server doppelte Daten eliminiert, oder wenn Sie sicher sind, dass in der Ergebnismenge keine Doppelnennungen vorkommen, können Sie anstelle von `distinct` das Schlüsselwort `all` verwenden. Da dieses jedoch der Default ist und daher nicht extra erwähnt werden muss, schreibt kaum ein Programmierer `all` in seine Abfragen.

Um eine `distinct`-Ergebnismenge zu generieren, müssen die Daten sortiert werden, was bei großen Ergebnismengen viel Zeit in Anspruch nimmt. Bitte fallen Sie nicht darauf herein, `distinct` nur zu benutzen, um Duplikate auszuschließen. Stattdessen nehmen Sie sich besser Zeit, die Daten, mit denen Sie arbeiten, genau zu verstehen. So können Sie erkennen, ob Duplikate möglich sind oder nicht.

Die from-Klausel

Bisher betrachteten wir Abfragen, deren `from`-Klauseln nur eine einzige Tabelle enthielten. Zwar definieren die meisten SQL-Bücher die `from`-Klausel einfach als Liste von einer oder mehreren Tabellen, aber ich würde diese Definition gern ausweiten:

> *Die `from`-Klausel definiert die Tabellen, die von einer Abfrage benutzt werden, und besitzt zusätzlich Mittel, die Tabellen zu verknüpfen.*

Diese Definition setzt sich aus zwei getrennten, aber zusammenhängenden Konzepten zusammen, die in den folgenden Abschnitten genauer erläutert werden.

Tabellen

Bei dem Wort *Tabelle* denken die meisten Menschen an zusammengehörige Zeilen, die gemeinsam in einer Datenbank gespeichert sind. Das beschreibt aber nur eine Art von Tabellen. Ich verwende diesen Begriff gern in einem allgemeineren Sinn, in dem die Art und Weise, wie die Daten gespeichert werden, keine Rolle spielen. Stattdessen konzentriere ich mich nur auf die Menge zusammengehöriger Zeilen. Auf diese umfassendere Definition passen jedoch vier Arten von Tabellen:

- permanente Tabellen (die mit `create table` angelegt wurden)
- abgeleitete Tabellen (Zeilen, die von einer Unterabfrage zurückgegeben und im Speicher gehalten werden)
- temporäre Tabellen (volatile Daten, die im Speicher gehalten werden)
- virtuelle Tabellen (die mit `create view` angelegt wurden)

Jeden dieser Tabellentypen kann man in der `from`-Klausel einer Abfrage angeben. Da Sie inzwischen wissen, wie man eine permanente Tabelle in einer `from`-Klausel verwendet, werde ich jetzt auf die drei anderen Tabellentypen näher eingehen.

Abgeleitete (von Unterabfragen generierte) Tabellen

Eine Unterabfrage ist eine in einer anderen Abfrage enthaltene Abfrage. Unterabfragen stehen in runden Klammern und kommen in diversen Teilen einer `select`-Anweisung vor. In der `from`-Klausel hat eine Unterabfrage jedoch die Rolle, eine abgeleitete Tabelle anzulegen, die für alle anderen Klauseln der Abfrage sichtbar ist und mit den anderen in der `from`-Klausel aufgeführten Tabellen interagieren kann. Ein einfaches Beispiel:

```
mysql> SELECT concat(cust.last_name, ', ', cust.first_name) full_name
    -> FROM
    ->  (SELECT first_name, last_name, email
    ->   FROM customer
    ->   WHERE first_name = 'JESSIE'
    ->  ) cust;
```

```
+---------------+
| full_name     |
+---------------+
| BANKS, JESSIE |
| MILAM, JESSIE |
+---------------+
2 rows in set (0.00 sec)
```

In diesem Beispiel gibt eine Unterabfrage der customer-Tabelle drei Spalten zurück, und die *übergeordnete Abfrage* referenziert zwei der drei vorhandenen Spalten. Die Unterabfrage wird von der übergeordneten Abfrage durch ihren Alias referenziert, in diesem Fall cust. Die Daten von cust werden während des Ausführens der Abfrage im Speicher gehalten und dann verworfen. Dieses Beispiel ist grob vereinfacht und nicht sonderlich nützlich, aber in Kapitel 9 werden Unterabfragen eingehender behandelt.

Temporäre Tabellen

Auch wenn sich die Implementierung unterscheidet, erlaubt jede relationale Datenbank das Definieren volatiler oder temporärer Tabellen. Diese sehen wie permanente Tabellen aus, aber darin eingefügte Daten verschwinden irgendwann wieder (meist am Ende einer Transaktion oder wenn Ihre Datenbanksession beendet wird). Hier ein einfaches Beispiel, das zeigt, wie Schauspieler, deren Nachname mit J beginnt, temporär gespeichert werden können:

```
mysql> CREATE TEMPORARY TABLE actors_j
    ->  (actor_id smallint(5),
    ->   first_name varchar(45),
    ->   last_name varchar(45)
    ->  );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO actors_j
    -> SELECT actor_id, first_name, last_name
    -> FROM actor
    -> WHERE last_name LIKE 'J%';
Query OK, 7 rows affected (0.03 sec)
Records: 7  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM actors_j;
+----------+------------+-----------+
| actor_id | first_name | last_name |
+----------+------------+-----------+
|      119 | WARREN     | JACKMAN   |
|      131 | JANE       | JACKMAN   |
|        8 | MATTHEW    | JOHANSSON |
|       64 | RAY        | JOHANSSON |
|      146 | ALBERT     | JOHANSSON |
|       82 | WOODY      | JOLIE     |
|       43 | KIRK       | JOVOVICH  |
+----------+------------+-----------+
7 rows in set (0.00 sec)
```

Die sieben Zeilen werden temporär im Speicher gehalten und verschwinden, sobald Ihre Session geschlossen ist.

Die meisten Datenbankserver löschen die temporäre Tabelle, wenn die Session endet. Eine Ausnahme bildet Oracle, bei dem die Definition der temporären Tabelle für zukünftige Sessions erhalten bleibt.

Views

Eine View ist eine im Data Dictionary gespeicherte Abfrage. Sie sieht aus und verhält sich wie eine Tabelle, aber mit einer View sind keine Daten assoziiert (daher bezeichne ich sie als *virtuelle* Tabelle). Wenn Sie eine Abfrage auf einer View ausführen, wird diese Abfrage mit der View-Definition verschmolzen, um die Abfrage anzulegen, die letztlich ausgeführt wird.

Zur Veranschaulichung sehen Sie hier eine View-Definition, die die `customer`-Tabelle abfragt und vier der verfügbaren Spalten enthält:

```
mysql> CREATE VIEW cust_vw AS
    -> SELECT customer_id, first_name, last_name, active
    -> FROM customer;
Query OK, 0 rows affected (0.12 sec)
```

Nachdem die View erstellt wurde, werden keine weiteren Daten mehr generiert oder gespeichert: Die `select`-Anweisung wird einfach zur späteren Verwendung vom Server gespeichert. Da diese View nun angelegt ist, kann man sie abfragen:

```
mysql> SELECT first_name, last_name
    -> FROM cust_vw
    -> WHERE active = 0;
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| SANDRA     | MARTIN    |
| JUDITH     | COX       |
| SHEILA     | WELLS     |
| ERICA      | MATTHEWS  |
| HEIDI      | LARSON    |
| PENNY      | NEAL      |
| KENNETH    | GOODEN    |
| HARRY      | ARCE      |
| NATHAN     | RUNYON    |
| THEODORE   | CULP      |
| MAURICE    | CRAWLEY   |
| BEN        | EASTER    |
| CHRISTIAN  | JUNG      |
| JIMMIE     | EGGLESTON |
| TERRANCE   | ROUSH     |
+------------+-----------+
15 rows in set (0.00 sec)
```

Views werden aus verschiedenen Gründen erzeugt, etwa um Spalten vor Benutzern zu verbergen oder um komplexe Datenbankentwürfe zu vereinfachen.

Tabellenverknüpfungen

Die zweite Abweichung von der einfachen Definition der `from`-Klausel ist folgende Vorschrift: Wenn mehrere Tabellen in der `from`-Klausel auftauchen, müssen auch die Bedingungen für eine *Verknüpfung* dieser Tabellen mit angegeben werden. Dies ist kein Gebot von MySQL oder einem anderen Datenbankserver, sondern eine vom ANSI anerkannte Methode, mehrere Tabellen miteinander zu verbinden, und es ist die portabelste Methode auf den verschiedenen Datenbankservern. Tabellen-Joins werden in Kapitel 5 und Kapitel 10 genauer erläutert. Hier sehen Sie vorab bereits ein kleines Beispiel – für den Fall, dass ich Ihre Neugier geweckt habe:

```
mysql> SELECT customer.first_name, customer.last_name,
    ->   time(rental.rental_date) rental_time
    -> FROM customer
    ->   INNER JOIN rental
    ->   ON customer.customer_id = rental.customer_id
    -> WHERE date(rental.rental_date) = '2005-06-14';
+------------+-----------+-------------+
| first_name | last_name | rental_time |
+------------+-----------+-------------+
| JEFFERY    | PINSON    | 22:53:33    |
| ELMER      | NOE       | 22:55:13    |
| MINNIE     | ROMERO    | 23:00:34    |
| MIRIAM     | MCKINNEY  | 23:07:08    |
| DANIEL     | CABRAL    | 23:09:38    |
| TERRANCE   | ROUSH     | 23:12:46    |
| JOYCE      | EDWARDS   | 23:16:26    |
| GWENDOLYN  | MAY       | 23:16:27    |
| CATHERINE  | CAMPBELL  | 23:17:03    |
| MATTHEW    | MAHAN     | 23:25:58    |
| HERMAN     | DEVORE    | 23:35:09    |
| AMBER      | DIXON     | 23:42:56    |
| TERRENCE   | GUNDERSON | 23:47:35    |
| SONIA      | GREGORY   | 23:50:11    |
| CHARLES    | KOWALSKI  | 23:54:34    |
| JEANETTE   | GREENE    | 23:54:46    |
+------------+-----------+-------------+
16 rows in set (0.01 sec)
```

Da diese Abfrage Daten aus den Tabellen `customer` (`first_name`, `last_name`) und `rental` (`rental_date`) anzeigt, müssen beide Tabellen in der `from`-Klausel genannt werden. Der Mechanismus für eine Verknüpfung der beiden Tabellen (ein sogenannter *Join*) ist die Kunden-ID, die sowohl in der `customer`- wie auch in der `rental`-Tabelle gespeichert ist. Der Datenbankserver wird angewiesen, den Wert der Spalte `customer_id` aus der Tabelle `customer` zu verwenden, um alle Ausleihen in der Tabelle `rental` zu finden. Die Join-Bedingungen für die beiden Tabellen findet man

in der on-Subklausel der from-Klausel. Hier lautet die Join-Bedingung ON customer.customer_id = rental.customer_id. Die where-Klausel ist nicht Teil des Joins und nur aufgenommen worden, um die Ergebnismenge schön klein zu halten, da es mehr als 16.000 Zeilen in der rental-Tabelle gibt. Noch einmal: Kapitel 5 beschreibt ausführlich, wie man mehrere Tabellen verbindet.

Tabellenaliase definieren

Werden mehrere Tabellen in einer einzigen Abfrage verbunden, muss herauszufinden sein, welche Tabelle gemeint ist, wenn in den Klauseln select, where, group by, having und order by Spalten angesprochen werden. Wenn Sie eine Tabelle außerhalb der from-Klausel referenzieren, haben Sie zwei Möglichkeiten:

- Sie verwenden den vollständigen Namen der Tabelle, wie in customer.customer_id.
- Sie weisen den Tabellen *Aliasnamen* zu und verwenden in der Abfrage die Aliase.

In der obigen Abfrage nutzte ich in der select- und der on-Klausel den vollständigen Namen der Tabelle. Mit Aliasen würde die gleiche Abfrage folgendermaßen aussehen:

```
SELECT c.first_name, c.last_name,
  time(r.rental_date) rental_time
FROM customer c
  INNER JOIN rental r
  ON c.customer_id = r.customer_id
WHERE date(r.rental_date) = '2005-06-14';
```

Wenn Sie sich die from-Klausel genau anschauen, erkennen Sie, dass die customer-Tabelle den Alias c und die rental-Tabelle den Alias r bekommen hat. Diese Aliase werden dann in der on-Klausel verwendet, um die Join-Bedingung zu definieren, und in der select-Klausel, um anzugeben, welche Spalten in die Ergebnismenge aufgenommen werden. Ich hoffe, Sie werden mir zustimmen, dass die Aliase die Anweisung kompakter machen, ohne Verwirrung zu stiften (solange die Aliasnamen sinnvoll gewählt werden). Zusätzlich können Sie bei Ihren Tabellenaliasen das Schlüsselwort as verwenden, wie wir es zuvor bereits für Spaltenaliase gezeigt haben:

```
SELECT c.first_name, c.last_name,
  time(r.rental_date) rental_time
FROM customer AS c
  INNER JOIN rental AS r
  ON c.customer_id = r.customer_id
WHERE date(r.rental_date) = '2005-06-14';
```

Ich habe festgestellt, dass rund die Hälfte der Datenbankentwickler, mit denen ich zusammengearbeitet habe, das Schlüsselwort as für ihre Spalten- und Tabellenaliase nutzen, die andere Hälfte dagegen nicht.

Die where-Klausel

In manchen Fällen wollen Sie alle Zeilen einer Tabelle abfragen, insbesondere bei kleinen Tabellen wie language. Doch meist möchte man nicht jede Zeile abrufen, sondern uninteressante Zeilen beiseitelassen. Dies ist die Aufgabe der where-Klausel.

> *Die* where*-Klausel ist der Mechanismus, mit dem unerwünschte Zeilen aus der Ergebnismenge herausgefiltert werden.*

Wenn Sie beispielsweise einen Film ausleihen wollen, aber nur an solchen interessiert sind, die mit G bewertet wurden und mindestens eine Woche lang behalten werden können, lassen Sie mit der where-Klausel der folgenden Abfrage nur die Filme ausgeben, die diese Bedingungen erfüllen:

```
mysql> SELECT title
    -> FROM film
    -> WHERE rating = 'G' AND rental_duration >= 7;
+-------------------------+
| title                   |
+-------------------------+
| BLANKET BEVERLY         |
| BORROWERS BEDAZZLED     |
| BRIDE INTRIGUE          |
| CATCH AMISTAD           |
| CITIZEN SHREK           |
| COLDBLOODED DARLING     |
| CONTROL ANTHEM          |
| CRUELTY UNFORGIVEN      |
| DARN FORRESTER          |
| DESPERATE TRAINSPOTTING |
| DIARY PANIC             |
| DRACULA CRYSTAL         |
| EMPIRE MALKOVICH        |
| FIREHOUSE VIETNAM       |
| GILBERT PELICAN         |
| GRADUATE LORD           |
| GREASE YOUTH            |
| GUN BONNIE              |
| HOOK CHARIOTS           |
| MARRIED GO              |
| MENAGERIE RUSHMORE      |
| MUSCLE BRIGHT           |
| OPERATION OPERATION     |
| PRIMARY GLASS           |
| REBEL AIRPORT           |
| SPIKING ELEMENT         |
| TRUMAN CRAZY            |
| WAKE JAWS               |
| WAR NOTTING             |
+-------------------------+
29 rows in set (0.00 sec)
```

Hier wurden 971 aus den 1.000 Datensätzen der film-Tabelle von der where-Klausel herausgefiltert. Diese where-Klausel enthält zwei *Filterbedingungen*, aber wenn Sie möchten, können Sie so viele Bedingungen wie nötig aufführen. Die Bedingungen werden durch Operatoren wie and, or und not getrennt (in Kapitel 4 werden die where-Klausel und ihre Filterbedingungen eingehend erläutert).

Schauen wir, was passiert, wenn wir den Operator zwischen den beiden Bedingungen von and nach or ändern:

```
mysql> SELECT title
    -> FROM film
    -> WHERE rating = 'G' OR rental_duration >= 7;
+--------------------------+
| title                    |
+--------------------------+
| ACE GOLDFINGER           |
| ADAPTATION HOLES         |
| AFFAIR PREJUDICE         |
| AFRICAN EGG              |
| ALAMO VIDEOTAPE          |
| AMISTAD MIDSUMMER        |
| ANGELS LIFE              |
| ANNIE IDENTITY           |
|...                       |
| WATERSHIP FRONTIER       |
| WEREWOLF LOLA            |
| WEST LION                |
| WESTWARD SEABISCUIT      |
| WOLVES DESIRE            |
| WON DARES                |
| WORKER TARZAN            |
| YOUNG LANGUAGE           |
+--------------------------+
340 rows in set (0.00 sec)
```

Verbinden Sie zwei Bedingungen mit dem and-Operator, müssen *alle* Bedingungen true sein, um eine Zeile in die Ergebnismenge aufzunehmen – verwenden Sie or, muss nur *eine* der Bedingungen true sei, damit die Zeile dazugehört. Das erklärt, warum die Ergebnismenge von 29 auf 340 Zeilen angewachsen ist.

Was sollte man also tun, wenn man sowohl and als auch or in der where-Klausel verwenden muss? Gute Frage. In einem solchen Fall setzen Sie runde Klammern ein, um die Bedingungen zu Gruppen zusammenzufassen. Die nächste Abfrage besagt, dass nur mit G bewertete Filme, die für sieben oder mehr Tage ausleihbar sind, oder PG-13-bewertete Filme, die für drei oder weniger Tage zu leihen sind, in die Ergebnismenge kommen:

```
mysql> SELECT title, rating, rental_duration
    -> FROM film
    -> WHERE (rating = 'G' AND rental_duration >= 7)
    ->   OR (rating = 'PG-13' AND rental_duration < 4);
```

```
+------------------------+--------+-----------------+
| title                  | rating | rental_duration |
+------------------------+--------+-----------------+
| ALABAMA DEVIL          | PG-13  |               3 |
| BACKLASH UNDEFEATED    | PG-13  |               3 |
| BILKO ANONYMOUS        | PG-13  |               3 |
| BLANKET BEVERLY        | G      |               7 |
| BORROWERS BEDAZZLED    | G      |               7 |
| BRIDE INTRIGUE         | G      |               7 |
| CASPER DRAGONFLY       | PG-13  |               3 |
| CATCH AMISTAD          | G      |               7 |
| CITIZEN SHREK          | G      |               7 |
| COLDBLOODED DARLING    | G      |               7 |
|...                                              |
| TREASURE COMMAND       | PG-13  |               3 |
| TRUMAN CRAZY           | G      |               7 |
| WAIT CIDER             | PG-13  |               3 |
| WAKE JAWS              | G      |               7 |
| WAR NOTTING            | G      |               7 |
| WORLD LEATHERNECKS     | PG-13  |               3 |
+------------------------+--------+-----------------+
68 rows in set (0.00 sec)
```

Verwenden Sie immer Klammern, um separate Gruppen von Bedingungen zusammenzufassen oder verschiedene Operatoren zu nutzen. So sind Sie, der Datenbankserver und andere Bearbeiter, die später Ihren Code modifizieren sollen, immer auf demselben Stand.

Die Klauseln group by und having

Alle bisherigen Abfragen haben unbearbeitete Rohdaten geliefert. Doch gelegentlich möchte man auch Trends in den Daten ausfindig machen, und dazu muss der Datenbankserver die Daten ein wenig aufbereiten, ehe die Ergebnismenge abgeliefert werden kann. Ein möglicher Mechanismus hierfür ist die `group by`-Klausel, die Daten nach Spaltenwerten zusammenfasst. Vielleicht möchten Sie alle Kunden finden, die schon 40 oder mehr Filme ausgeliehen haben. Statt alle 16.044 Zeilen in der `rental`-Tabelle zu durchforsten, können Sie eine Abfrage schreiben, die den Server anweist, alle Ausleihvorgänge nach Kunden zu gruppieren, die Anzahl der Ausleihen für jeden Kunden zu zählen und nur die zurückzugeben, deren Ausleihzähler mindestens 40 beträgt. Wenn Sie die `group by`-Klausel verwenden, um Zeilengruppen zu bilden, können Sie auch eine `having`-Klausel einsetzen: Diese filtert die gruppierten Daten in der gleichen Weise, wie es die `where`-Klausel mit Rohdaten tut.

So kann die Abfrage aussehen:

```
mysql> SELECT c.first_name, c.last_name, count(*)
    -> FROM customer c
    ->   INNER JOIN rental r
    ->   ON c.customer_id = r.customer_id
```

```
    -> GROUP BY c.first_name, c.last_name
    -> HAVING count(*) >= 40;
+------------+-----------+----------+
| first_name | last_name | count(*) |
+------------+-----------+----------+
| TAMMY      | SANDERS   |       41 |
| CLARA      | SHAW      |       42 |
| ELEANOR    | HUNT      |       46 |
| SUE        | PETERS    |       40 |
| MARCIA     | DEAN      |       42 |
| WESLEY     | BULL      |       40 |
| KARL       | SEAL      |       45 |
+------------+-----------+----------+
7 rows in set (0.03 sec)
```

Ich wollte diese beiden Klauseln nur kurz ansprechen, damit Sie später in diesem Buch bereits darauf vorbereitet sind. Sie sind allerdings etwas fortgeschrittener als die beiden `select`-Klauseln. Daher bitte ich Sie um Verständnis, dass ich erst in Kapitel 8 genau sagen werde, wann und wie man `group by` und `having` einsetzt.

Die order by-Klausel

Im Allgemeinen haben die Zeilen der Ergebnismenge einer Abfrage keine besondere Reihenfolge. Wenn Sie eine sortierte Ergebnismenge wünschen, müssen Sie den Server mit der `order by`-Klausel veranlassen, die Ergebnisse zu sortieren.

> *Die* `order by`*-Klausel ist der Mechanismus, mit dem die Ergebnismenge sortiert wird, und zwar entweder nach rohen Spaltendaten oder anhand von Ausdrücken, die auf Spaltendaten basieren.*

Als Beispiel sehen Sie hier noch einmal die bereits erwähnte Abfrage für alle Kunden, die am 14. Juni 2005 einen Film ausgeliehen haben:

```
mysql> SELECT c.first_name, c.last_name,
    ->   time(r.rental_date) rental_time
    -> FROM customer c
    ->   INNER JOIN rental r
    ->   ON c.customer_id = r.customer_id
    -> WHERE date(r.rental_date) = '2005-06-14';
+------------+-----------+-------------+
| first_name | last_name | rental_time |
+------------+-----------+-------------+
| JEFFERY    | PINSON    | 22:53:33    |
| ELMER      | NOE       | 22:55:13    |
| MINNIE     | ROMERO    | 23:00:34    |
| MIRIAM     | MCKINNEY  | 23:07:08    |
| DANIEL     | CABRAL    | 23:09:38    |
| TERRANCE   | ROUSH     | 23:12:46    |
| JOYCE      | EDWARDS   | 23:16:26    |
| GWENDOLYN  | MAY       | 23:16:27    |
| CATHERINE  | CAMPBELL  | 23:17:03    |
| MATTHEW    | MAHAN     | 23:25:58    |
```

```
| HERMAN     | DEVORE    | 23:35:09    |
| AMBER      | DIXON     | 23:42:56    |
| TERRENCE   | GUNDERSON | 23:47:35    |
| SONIA      | GREGORY   | 23:50:11    |
| CHARLES    | KOWALSKI  | 23:54:34    |
| JEANETTE   | GREENE    | 23:54:46    |
+------------+-----------+-------------+
16 rows in set (0.01 sec)
```

Möchten Sie die Ergebnisse alphabetisch nach Nachnamen sortiert haben, können Sie die `order by`-Klausel um `last_name` ergänzen:

```
mysql> SELECT c.first_name, c.last_name,
    ->   time(r.rental_date) rental_time
    -> FROM customer c
    ->   INNER JOIN rental r
    ->   ON c.customer_id = r.customer_id
    -> WHERE date(r.rental_date) = '2005-06-14'
    -> ORDER BY c.last_name;
+------------+-----------+-------------+
| first_name | last_name | rental_time |
+------------+-----------+-------------+
| DANIEL     | CABRAL    | 23:09:38    |
| CATHERINE  | CAMPBELL  | 23:17:03    |
| HERMAN     | DEVORE    | 23:35:09    |
| AMBER      | DIXON     | 23:42:56    |
| JOYCE      | EDWARDS   | 23:16:26    |
| JEANETTE   | GREENE    | 23:54:46    |
| SONIA      | GREGORY   | 23:50:11    |
| TERRENCE   | GUNDERSON | 23:47:35    |
| CHARLES    | KOWALSKI  | 23:54:34    |
| MATTHEW    | MAHAN     | 23:25:58    |
| GWENDOLYN  | MAY       | 23:16:27    |
| MIRIAM     | MCKINNEY  | 23:07:08    |
| ELMER      | NOE       | 22:55:13    |
| JEFFERY    | PINSON    | 22:53:33    |
| MINNIE     | ROMERO    | 23:00:34    |
| TERRANCE   | ROUSH     | 23:12:46    |
+------------+-----------+-------------+
16 rows in set (0.01 sec)
```

Große Kundenlisten enthalten oft mehrere Personen mit dem gleichen Nachnamen (auch wenn das hier nicht der Fall ist). Dann möchten Sie vielleicht das Sortierkriterium um den Vornamen der Person erweitern.

Das erreichen Sie, indem Sie in der `order by`-Klausel die Spalte `first_name` zusätzlich nach der Spalte `last_name` angeben:

```
mysql> SELECT c.first_name, c.last_name,
    ->   time(r.rental_date) rental_time
    -> FROM customer c
    ->   INNER JOIN rental r
    ->   ON c.customer_id = r.customer_id
```

```
    -> WHERE date(r.rental_date) = '2005-06-14'
    -> ORDER BY c.last_name, c.first_name;
+------------+-----------+-------------+
| first_name | last_name | rental_time |
+------------+-----------+-------------+
| DANIEL     | CABRAL    | 23:09:38    |
| CATHERINE  | CAMPBELL  | 23:17:03    |
| HERMAN     | DEVORE    | 23:35:09    |
| AMBER      | DIXON     | 23:42:56    |
| JOYCE      | EDWARDS   | 23:16:26    |
| JEANETTE   | GREENE    | 23:54:46    |
| SONIA      | GREGORY   | 23:50:11    |
| TERRENCE   | GUNDERSON | 23:47:35    |
| CHARLES    | KOWALSKI  | 23:54:34    |
| MATTHEW    | MAHAN     | 23:25:58    |
| GWENDOLYN  | MAY       | 23:16:27    |
| MIRIAM     | MCKINNEY  | 23:07:08    |
| ELMER      | NOE       | 22:55:13    |
| JEFFERY    | PINSON    | 22:53:33    |
| MINNIE     | ROMERO    | 23:00:34    |
| TERRANCE   | ROUSH     | 23:12:46    |
+------------+-----------+-------------+
16 rows in set (0.01 sec)
```

Die Reihenfolge, in der die Spalten in Ihrer `order by`-Klausel aufgeführt sind, ist entscheidend, wenn Sie mehr als eine Spalte nutzen. Würden Sie die Reihenfolge in diesem Beispiel vertauschen, würde Amber Dixon in der Ergebnisliste als Erstes erscheinen.

Auf- und absteigende Sortierung

Beim Sortieren haben Sie die Möglichkeit, mit den Schlüsselwörtern `asc` und `desc` eine *aufsteigende* oder eine *absteigende* Reihenfolge festzulegen. Nach Voreinstellung wird ohnehin aufsteigend sortiert, sodass Sie eigentlich nur das Schlüsselwort `desc` einsetzen müssen, sofern Sie eine absteigende Sortierung wünschen. Die folgende Beispielabfrage listet alle Kunden auf, die am 14. Juni 2005 Filme ausgeliehen haben – in absteigender Reihenfolge der Ausleihzeit:

```
mysql> SELECT c.first_name, c.last_name,
    ->   time(r.rental_date) rental_time
    -> FROM customer c
    ->   INNER JOIN rental r
    ->   ON c.customer_id = r.customer_id
    -> WHERE date(r.rental_date) = '2005-06-14'
    -> ORDER BY time(r.rental_date) desc;
+------------+-----------+-------------+
| first_name | last_name | rental_time |
+------------+-----------+-------------+
| JEANETTE   | GREENE    | 23:54:46    |
| CHARLES    | KOWALSKI  | 23:54:34    |
| SONIA      | GREGORY   | 23:50:11    |
```

```
| TERRENCE   | GUNDERSON | 23:47:35    |
| AMBER      | DIXON     | 23:42:56    |
| HERMAN     | DEVORE    | 23:35:09    |
| MATTHEW    | MAHAN     | 23:25:58    |
| CATHERINE  | CAMPBELL  | 23:17:03    |
| GWENDOLYN  | MAY       | 23:16:27    |
| JOYCE      | EDWARDS   | 23:16:26    |
| TERRANCE   | ROUSH     | 23:12:46    |
| DANIEL     | CABRAL    | 23:09:38    |
| MIRIAM     | MCKINNEY  | 23:07:08    |
| MINNIE     | ROMERO    | 23:00:34    |
| ELMER      | NOE       | 22:55:13    |
| JEFFERY    | PINSON    | 22:53:33    |
+------------+-----------+-------------+
16 rows in set (0.01 sec)
```

Absteigende Sortierreihenfolgen werden oft für Rankings verwendet, etwa in der Art »zeige mir die 5 höchsten Kontensalden«. MySQL kennt zudem eine `limit`-Klausel, mit der Sie Daten sortieren und dann alle außer den ersten *x* Zeilen wieder verwerfen können.

Sortieren nach numerischen Platzhaltern

Wenn Sie in Ihrer `select`-Klausel die Daten nach Spalten sortieren, können Sie diese Spalten statt nach Namen auch nach ihrer *Position* in der `select`-Klausel referenzieren. Nutzen wir das vorige Beispiel und schreiben wir in die `order by`-Klausel eine absteigende Reihenfolge für das dritte Element der `select`-Klausel:

```
mysql> SELECT c.first_name, c.last_name,
    ->   time(r.rental_date) rental_time
    -> FROM customer c
    ->   INNER JOIN rental r
    ->   ON c.customer_id = r.customer_id
    -> WHERE date(r.rental_date) = '2005-06-14'
    -> ORDER BY 3 desc;
+------------+-----------+-------------+
| first_name | last_name | rental_time |
+------------+-----------+-------------+
| JEANETTE   | GREENE    | 23:54:46    |
| CHARLES    | KOWALSKI  | 23:54:34    |
| SONIA      | GREGORY   | 23:50:11    |
| TERRENCE   | GUNDERSON | 23:47:35    |
| AMBER      | DIXON     | 23:42:56    |
| HERMAN     | DEVORE    | 23:35:09    |
| MATTHEW    | MAHAN     | 23:25:58    |
| CATHERINE  | CAMPBELL  | 23:17:03    |
| GWENDOLYN  | MAY       | 23:16:27    |
| JOYCE      | EDWARDS   | 23:16:26    |
| TERRANCE   | ROUSH     | 23:12:46    |
| DANIEL     | CABRAL    | 23:09:38    |
| MIRIAM     | MCKINNEY  | 23:07:08    |
```

```
| MINNIE     | ROMERO    | 23:00:34    |
| ELMER      | NOE       | 22:55:13    |
| JEFFERY    | PINSON    | 22:53:33    |
+------------+-----------+-------------+
16 rows in set (0.01 sec)
```

Dieses Feature sollten Sie jedoch sparsam einsetzen, da manchmal seltsame Dinge passieren können, wenn man der `select`-Klausel eine Spalte hinzufügt, ohne die Nummern in der `order by`-Klausel entsprechend abzuändern. Wenn ich mal eben eine Abfrage schreibe, referenziere ich Spalten gelegentlich auch numerisch, aber wenn ich richtigen Code schreibe, referenziere ich Spalten grundsätzlich über den Namen.

Testen Sie Ihr Wissen

Die folgenden Übungen sollen zum besseren Verständnis der `select`-Anweisung und ihrer Klauseln beitragen. Die Lösungen finden Sie in Anhang B.

Übung 3-1

Fragen Sie die Schauspieler-ID (Actor-ID), den Vornamen und den Nachnamen aller Schauspieler ab. Sortieren Sie die Daten zuerst nach Nachnamen und dann nach Vornamen.

Übung 3-2

Fragen Sie die Schauspieler-ID (Actor-ID), den Vor- und den Nachnamen aller Schauspieler ab, deren Nachname gleich `'WILLLIAMS'` oder `'DAVIS'` ist.

Übung 3-3

Schreiben Sie für die `rental`-Tabelle eine Abfrage, die die IDs der Kunden liefert, die am 5. Juli 2005 einen Film ausgeliehen haben (nutzen Sie die Spalte `rental.rental_date`; mit der `date()`-Funktion können Sie den Zeitanteil ignorieren). Für jeden Kunden soll eine einzige Zeile zurückgeliefert werden.

Übung 3-4

Füllen Sie die (durch `<Zahl>` gekennzeichneten) Lücken für diese Abfrage mehrerer Datensätze aus, um die unten gezeigten Ergebnisse zu erhalten:

```
mysql> SELECT c.email, r.return_date
    -> FROM customer c
    ->   INNER JOIN rental <1>
    ->   ON c.customer_id = <2>
    -> WHERE date(r.rental_date) = '2005-06-14'
    -> ORDER BY <3> <4>;
```

```
+--------------------------------------+---------------------+
| email                                | return_date         |
+--------------------------------------+---------------------+
| DANIEL.CABRAL@sakilacustomer.org     | 2005-06-23 22:00:38 |
| TERRANCE.ROUSH@sakilacustomer.org    | 2005-06-23 21:53:46 |
| MIRIAM.MCKINNEY@sakilacustomer.org   | 2005-06-21 17:12:08 |
| GWENDOLYN.MAY@sakilacustomer.org     | 2005-06-20 02:40:27 |
| JEANETTE.GREENE@sakilacustomer.org   | 2005-06-19 23:26:46 |
| HERMAN.DEVORE@sakilacustomer.org     | 2005-06-19 03:20:09 |
| JEFFERY.PINSON@sakilacustomer.org    | 2005-06-18 21:37:33 |
| MATTHEW.MAHAN@sakilacustomer.org     | 2005-06-18 05:18:58 |
| MINNIE.ROMERO@sakilacustomer.org     | 2005-06-18 01:58:34 |
| SONIA.GREGORY@sakilacustomer.org     | 2005-06-17 21:44:11 |
| TERRENCE.GUNDERSON@sakilacustomer.org | 2005-06-17 05:28:35 |
| ELMER.NOE@sakilacustomer.org         | 2005-06-17 02:11:13 |
| JOYCE.EDWARDS@sakilacustomer.org     | 2005-06-16 21:00:26 |
| AMBER.DIXON@sakilacustomer.org       | 2005-06-16 04:02:56 |
| CHARLES.KOWALSKI@sakilacustomer.org  | 2005-06-16 02:26:34 |
| CATHERINE.CAMPBELL@sakilacustomer.org | 2005-06-15 20:43:03 |
+--------------------------------------+---------------------+
16 rows in set (0.03 sec)
```