

Mit Übungen und Lösungen

2. Auflage
Aktualisiert & erweitert

Einführung in SQL



O'REILLY®

Alan Beaulieu
Deutsche Übersetzung
von Dorothea Heymann-Reder & Lars Schulten

Einleitung	IX
1 Der Hintergrund	1
Einführung in Datenbanken	1
Was ist SQL?	7
Was ist MySQL?	13
Weiteres Vorgehen	14
2 Datenbanken erstellen und mit Daten füllen	15
Eine MySQL-Datenbank anlegen	15
Das mysql-Kommandozeilentool	17
MySQL-Datentypen	17
Tabellen anlegen	25
Tabellen füllen und ändern	30
Wenn aus guten Anweisungen schlechte werden	36
Das Bank-Schema	38
3 Datenbankabfragen	41
Die Mechanik von Abfragen	41
Abfrageklauseln	43
Die Select-Klausel	43
Die From-Klausel	49
Die where-Klausel	53
Die Klauseln group by und having	55
Die order by-Klausel	56
Testen Sie Ihr Wissen	60
4 Filtern	63
Bedingungsauswertung	63
Aufbau einer Bedingung	66
Bedingungstypen	66

NULL: Ein böses Wort	76
Testen Sie Ihr Wissen	79
5 Mehrere Tabellen abfragen	81
Was ist ein Join?	81
Joins mit drei oder mehr Tabellen	88
Self-Joins	93
Equi-Joins und Non-Equi-Joins	94
Join-Bedingungen und Filterbedingungen	97
Testen Sie Ihr Wissen	98
6 Umgang mit Mengen	101
Grundlagen der Mengenlehre	101
Mengenlehre in der Praxis	104
Mengenoperatoren	105
Regeln für Mengenoperationen	111
Testen Sie Ihr Wissen	114
7 Daten generieren, konvertieren und manipulieren	115
Der Umgang mit String-Daten	115
Der Umgang mit numerischen Daten	128
Der Umgang mit temporalen Daten	133
Konvertierungsfunktionen	144
Testen Sie Ihr Wissen	145
8 Gruppieren und Aggregieren von Daten	147
Gruppieren von Daten	147
Aggregatfunktionen	150
Gruppen generieren	155
Gruppen-Filterbedingungen	159
Testen Sie Ihr Wissen	161
9 Unterabfragen	163
Was ist eine Unterabfrage?	163
Typen von Unterabfragen	164
Nicht-korrelierte Unterabfragen	165
Korrelierte Unterabfragen	174
Einsatz von Unterabfragen	178
Zusammenfassung zu Unterabfragen	188
Testen Sie Ihr Wissen	189
10 Weitere Joins	191
Outer Joins	191
Cross Joins	201

Natural Joins	207
Testen Sie Ihr Wissen	209
11 Bedingungslogik	211
Was ist Bedingungslogik?	211
Der Case-Ausdruck	212
Beispiele für Case-Ausdrücke	216
Testen Sie Ihr Wissen	223
12 Transaktionen	225
Mehrbenutzerdatenbanken	225
Was ist eine Transaktion?	227
Testen Sie Ihr Wissen	234
13 Indizes und Constraints	235
Indizes	235
Constraints	246
Testen Sie Ihr Wissen	251
14 Views	253
Was sind Views?	253
Warum Views verwenden?	256
Aktualisierbare Views	260
Testen Sie Ihr Wissen	263
15 Metadaten	265
Daten über Daten	265
Information_Schema	266
Mit Metadaten arbeiten	271
Testen Sie Ihr Wissen	278
A ER-Diagramm der Musterdatenbank	279
B MySQL-Erweiterungen für SQL	281
C Lösungen der Übungen	295
Index	317

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.

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 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: 6.0.3-alpha-community MySQL Community Server (GPL)
```

```
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 benutzt, 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 tut. Wenn Sie Oracle Database oder SQL Server benutzen, 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 emp_id, fname, lname
-> FROM employee
-> WHERE lname = 'Bkadfl';
Empty set (0.00 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 fname, lname
-> FROM employee;
+-----+-----+
| fname | lname |
+-----+-----+
| Michael | Smith |
| Susan | Barker |
| Robert | Tyler |
| Susan | Hawthorne |
| John | Gooding |
```

Helen	Fleming
Chris	Tucker
Sarah	Parker
Jane	Grossman
Paula	Roberts
Thomas	Ziegler
Samantha	Jameson
John	Blake
Cindy	Mason
Frank	Portman
Theresa	Markham
Beth	Fowler
Rick	Tulman

-----+
 18 rows in set (0.00 sec)

Diese Abfrage gibt die Vor- und Nachnamen aller Mitarbeiter aus der `employee`-Tabelle zurück. Nachdem die letzte Datenzeile angezeigt wurde, zeigt das `mysql`-Tool an, wie viele Zeilen zurückgeliefert wurden. Hier sind es 18.

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. Zusätzlich dazu kennt MySQL eine Reihe von eigenen Klauseln, die in Anhang B genauer beschrieben werden. Die folgenden Abschnitte beschreiben nur die 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 enthal-

ten *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 department;
+-----+
| dept_id | name          |
+-----+
|      1  | Operations    |
|      2  | Loans         |
|      3  | Administration|
+-----+
3 rows in set (0.04 sec)
```

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

Zeige mir alle Spalten der `department`-Tabelle.

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

```
mysql> SELECT dept_id, name
-> FROM department;
+-----+
| dept_id | name          |
+-----+
|      1  | Operations    |
|      2  | Loans         |
|      3  | Administration|
+-----+
3 rows in set (0.01 sec)
```

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

```
mysql> SELECT name
-> FROM department;
+-----+
| name          |
+-----+
| Operations    |
| Loans         |
| Administration|
+-----+
3 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 `employee`-Tabelle einfließen lässt:

```
mysql> SELECT emp_id,
-> 'ACTIVE',
-> emp_id * 3.14159,
-> UPPER(lname)
-> FROM employee;
```

emp_id	ACTIVE	emp_id * 3.14159	UPPER(lname)
1	ACTIVE	3.14159	SMITH
2	ACTIVE	6.28318	BARKER
3	ACTIVE	9.42477	TYLER
4	ACTIVE	12.56636	HAWTHORNE
5	ACTIVE	15.70795	GOODING
6	ACTIVE	18.84954	FLEMING
7	ACTIVE	21.99113	TUCKER
8	ACTIVE	25.13272	PARKER
9	ACTIVE	28.27431	GROSSMAN
10	ACTIVE	31.41590	ROBERTS
11	ACTIVE	34.55749	ZIEGLER
12	ACTIVE	37.69908	JAMESON
13	ACTIVE	40.84067	BLAKE
14	ACTIVE	43.98226	MASON
15	ACTIVE	47.12385	PORTMAN
16	ACTIVE	50.26544	MARKHAM
17	ACTIVE	53.40703	FOWLER
18	ACTIVE	56.54862	TULMAN

18 rows in set (0.05 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 beiseite lassen. Ein Beispiel:

```
mysql> SELECT VERSION(),
-> USER(),
-> DATABASE();
```

```

+-----+-----+-----+
| version() | user() | database() |
+-----+-----+-----+
| 6.0.3-alpha-community | lrngsql@localhost | bank |
+-----+-----+-----+
1 row in set (0.05 sec)

```

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

Spalten-Aliase

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 *Spalten-Alias* hinter den einzelnen Elementen der select-Klausel. Hier sehen Sie die vorige Abfrage der employee-Tabelle mit Spalten-Aliassen für drei Spalten:

```

mysql> SELECT emp_id,
-> 'ACTIVE' status,
-> emp_id * 3.14159 empid_x_pi,
-> UPPER(lname) last_name_upper
-> FROM employee;

```

emp_id	status	empid_x_pi	last_name_upper
1	ACTIVE	3.14159	SMITH
2	ACTIVE	6.28318	BARKER
3	ACTIVE	9.42477	TYLER
4	ACTIVE	12.56636	HAWTHORNE
5	ACTIVE	15.70795	GOODING
6	ACTIVE	18.84954	FLEMING
7	ACTIVE	21.99113	TUCKER
8	ACTIVE	25.13272	PARKER
9	ACTIVE	28.27431	GROSSMAN
10	ACTIVE	31.41590	ROBERTS
11	ACTIVE	34.55749	ZIEGLER
12	ACTIVE	37.69908	JAMESON
13	ACTIVE	40.84067	BLAKE
14	ACTIVE	43.98226	MASON
15	ACTIVE	47.12385	PORTMAN
16	ACTIVE	50.26544	MARKHAM
17	ACTIVE	53.40703	FOWLER
18	ACTIVE	56.54862	TULMAN

```

+-----+-----+-----+
18 rows in set (0.00 sec)

```

Wenn Sie nun die Überschriften betrachten, erkennen Sie, dass die zweite, dritte und vierte Spalte jetzt vernünftige Namen haben, anstatt nur nach der Funktion oder dem Ausdruck zu heißen, durch den sie generiert wurden. Betrachten Sie einmal die select-Klausel, sehen Sie, dass die Aliase `status`, `empid_x_pi` und `last_name_upper` hinter der zweiten, dritten und vierten Spalte stehen. 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 C# absetzte anstatt interaktiv mit dem `mysql`-Tool. Damit die Spalten-Aliase noch etwas besser erkennbar sind, können Sie vor dem Aliasnamen auch das Schlüsselwort `AS` angeben, wie Sie es in folgendem Beispiel sehen:

```
mysql> SELECT emp_id,  
-> 'ACTIVE' AS status,  
-> emp_id * 3.14159 AS empid_x_pi,  
-> UPPER(lname) AS last_name_upper  
-> FROM employee;
```

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 Kunden abfragen wollten, die Konten besitzen, erhielten Sie Folgendes:

```
mysql> SELECT cust_id  
-> FROM account;  
+-----+  
| cust_id |  
+-----+  
| 1 |  
| 1 |  
| 1 |  
| 2 |  
| 2 |  
| 3 |  
| 3 |  
| 4 |  
| 4 |  
| 4 |  
| 5 |  
| 6 |  
| 6 |  
| 7 |  
| 8 |  
| 8 |  
| 9 |  
| 9 |
```

```

|      9 |
|     10 |
|     10 |
|     11 |
|     12 |
|     13 |
+-----+
24 rows in set (0.00 sec)

```

Da manche Kunden mehrere Konten haben, wird jede Kunden-ID für jedes Konto des betreffenden Kunden extra aufgeführt. In diesem Fall möchten Sie jedoch vermutlich eine Menge unterschiedlicher Kunden, die Konten besitzen, anstatt die Kunden-ID für jede Zeile der account-Tabelle einmal anzeigen zu lassen. Dies erzielen Sie, indem Sie direkt hinter das select das Schlüsselwort `distinct` setzen:

```

mysql> SELECT DISTINCT cust_id
-> FROM account;

```

```

+-----+
| cust_id |
+-----+
|      1 |
|      2 |
|      3 |
|      4 |
|      5 |
|      6 |
|      7 |
|      8 |
|      9 |
|     10 |
|     11 |
|     12 |
|     13 |
+-----+
13 rows in set (0.01 sec)

```

Die neue Ergebnismenge enthält 13 Zeilen (eine pro Kunde) anstatt 24 (eine pro Konto).

Wenn Sie nicht möchten, dass der Server doppelte Daten eliminiert, oder wenn Sie sicher sind, dass in der Ergebnismenge keine Doppelnennungen vorkommen, können Sie an Stelle 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` zu benutzen, nur 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 an verwandte 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 verwandter Zeilen. Auf diese umfassendere Definition passen jedoch drei Arten von Tabellen:

- permanente Tabellen (die mit `create table` angelegt wurden)
- temporäre Tabellen (Zeilen, die von einer Unterabfrage zurückgegeben 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 beiden anderen Tabellentypen näher eingehen.

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. Doch in der from-Klausel hat eine Unterabfrage die Rolle, eine temporäre 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 e.emp_id, e.fname, e.lname  
-> FROM (SELECT emp_id, fname, lname, start_date, title  
-> FROM employee) e;
```

emp_id	fname	lname
1	Michael	Smith
2	Susan	Barker
3	Robert	Tyler
4	Susan	Hawthorne
5	John	Gooding
6	Helen	Fleming

7	Chris	Tucker
8	Sarah	Parker
9	Jane	Grossman
10	Paula	Roberts
11	Thomas	Ziegler
12	Samantha	Jameson
13	John	Blake
14	Cindy	Mason
15	Frank	Portman
16	Theresa	Markham
17	Beth	Fowler
18	Rick	Tulman

-----+-----+-----+
18 rows in set (0.00 sec)

In diesem Beispiel gibt eine Unterabfrage der `employee`-Tabelle fünf Spalten zurück, und die *übergeordnete Abfrage* referenziert drei der fünf vorhandenen Spalten. Die Unterabfrage wird von der übergeordneten Abfrage durch ihren Alias referenziert, in diesem Fall `e`. Dieses Beispiel ist grob vereinfacht und nicht sonderlich nützlich, aber in Kapitel 9 werden Unterabfragen eingehender behandelt.

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 `employee`-Tabelle abfragt und einen Aufruf einer eingebauten Funktion enthält:

```
mysql> CREATE VIEW employee_vw AS
-> SELECT emp_id, fname, lname,
-> YEAR(start_date) start_year
-> FROM employee;
Query OK, 0 rows affected (0.10 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 emp_id, start_year
-> FROM employee_vw;
+-----+-----+
| emp_id | start_year |
+-----+-----+
| 1 | 2005 |
| 2 | 2006 |
| 3 | 2005 |
| 4 | 2006 |
| 5 | 2007 |
```

6	2008
7	2008
8	2006
9	2006
10	2006
11	2004
12	2007
13	2004
14	2006
15	2007
16	2005
17	2006
18	2006

18 rows in set (0.07 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 den Kapiteln 5 und 10 genauer erläutert. Hier sehen Sie vorab bereits ein kleines Beispiel, nur für den Fall, dass ich Ihre Neugier geweckt habe:

```
mysql> SELECT employee.emp_id, employee.fname,
-> employee.lname, department.name dept_name
-> FROM employee INNER JOIN department
-> ON employee.dept_id = department.dept_id;
```

emp_id	fname	lname	dept_name
1	Michael	Smith	Administration
2	Susan	Barker	Administration
3	Robert	Tyler	Administration
4	Susan	Hawthorne	Operations
5	John	Gooding	Loans
6	Helen	Fleming	Operations
7	Chris	Tucker	Operations
8	Sarah	Parker	Operations
9	Jane	Grossman	Operations
10	Paula	Roberts	Operations
11	Thomas	Ziegler	Operations
12	Samantha	Jameson	Operations
13	John	Blake	Operations
14	Cindy	Mason	Operations

15	Frank	Portman	Operations
16	Theresa	Markham	Operations
17	Beth	Fowler	Operations
18	Rick	Tulman	Operations

18 rows in set (0.05 sec)

Da diese Abfrage Daten aus den Tabellen `employee` (`emp_id`, `fname`, `lname`) und `department` (`name`) anzeigt, müssen beide Tabellen in der `from`-Klausel genannt werden. Der Mechanismus für eine Verknüpfung der beiden Tabellen (ein sogenannter *Join*) ist die Zugehörigkeit des Mitarbeiters zu einer Abteilung, die in der `employee`-Tabelle gespeichert ist. Der Datenbankserver wird angewiesen, den Wert der Spalte `dept_id` aus der Tabelle `employee` zu verwenden, um in der Tabelle `department` die betreffende Abteilung nachzuschlagen. Die Join-Bedingungen für die beiden Tabellen findet man in der `on`-Subklausel der `from`-Klausel. Hier lautet die Join-Bedingung `ON employee.dept_id = department.dept_id`. Noch einmal: Kapitel 5 beschreibt gründlich, wie man mehrere Tabellen verbindet.

Tabellen-Aliase 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 `employee.emp_id`.
- Sie weisen den Tabellen *Aliasnamen* zu und verwenden in der Abfrage die Aliase.

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

```
SELECT e.emp_id, e.fname, e.lname,
       d.name dept_name
FROM employee e INNER JOIN department d
     ON e.dept_id = d.dept_id;
```

Wenn Sie sich die `from`-Klausel genau anschauen, erkennen Sie, dass die `employee`-Tabelle den Alias `e` und die `department`-Tabelle den Alias `d` 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 Tabellen-Aliasen das Schlüsselwort `AS` verwenden, wie wir es zuvor bereits für Spalten-Aliase gezeigt haben:

```
SELECT e.emp_id, e.fname, e.lname,
       d.name dept_name
FROM employee AS e INNER JOIN department AS d
     ON e.dept_id = d.dept_id;
```

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

Die where-Klausel

Die bisher in diesem Kapitel gezeigten Abfragen wählten jede Zeile aus den Tabellen `employee`, `department` oder `account` aus (mit Ausnahme des `distinct`-Beispiels weiter oben in diesem Kapitel). Doch meist möchte man nicht *jede* Zeile abrufen, sondern uninteressante Zeilen beiseite lassen. 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 aus der `employee`-Tabelle nur die Daten derjenigen Mitarbeiter abrufen möchten, die Chefkassierer (*Head Teller*) sind, können Sie dies mit der `where`-Klausel der folgenden Abfrage tun:

```
mysql> SELECT emp_id, fname, lname, start_date, title
-> FROM employee
-> WHERE title = 'Head Teller';
```

emp_id	fname	lname	start_date	title
6	Helen	Fleming	2008-03-17	Head Teller
10	Paula	Roberts	2006-07-27	Head Teller
13	John	Blake	2004-05-11	Head Teller
16	Theresa	Markham	2005-03-15	Head Teller

```
4 rows in set (1.17 sec)
```

Hier wurden 4 der 18 Mitarbeiterdatensätze von der `where`-Klausel herausgefiltert. Diese `where`-Klausel enthält zwar nur eine einzige *Filterbedingung*, 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). Hier sehen Sie eine Erweiterung der obigen Abfrage mit einer zweiten Bedingung, die besagt, dass nur Mitarbeiter von Interesse sind, die nach dem 1. Januar 2006 ihre Arbeit angetreten haben:

```
mysql> SELECT emp_id, fname, lname, start_date, title
-> FROM employee
-> WHERE title = 'Head Teller'
-> AND start_date > '2006-01-01';
```

emp_id	fname	lname	start_date	title
6	Helen	Fleming	2008-03-17	Head Teller
10	Paula	Roberts	2006-07-27	Head Teller

```
2 rows in set (0.01 sec)
```

Die erste Bedingung (`title = 'Head Teller'`) hatte 14 der 18 Mitarbeiterdatensätze herausgefiltert und die zweite (`start_date > '2006-01-01'`) nun noch einmal zwei, sodass zwei Zeilen in der endgültigen Ergebnismenge verbleiben. Wir wollen sehen, was passiert, wenn wir den Operator zwischen den beiden Bedingungen von `and` auf `or` umstellen.

```
mysql> SELECT emp_id, fname, lname, start_date, title
-> FROM employee
-> WHERE title = 'Head Teller'
-> OR start_date > '2006-01-01';
```

emp_id	fname	lname	start_date	title
2	Susan	Barker	2006-09-12	Vice President
4	Susan	Hawthorne	2006-04-24	Operations Manager
5	John	Gooding	2007-11-14	Loan Manager
6	Helen	Fleming	2008-03-17	Head Teller
7	Chris	Tucker	2008-09-15	Teller
8	Sarah	Parker	2006-12-02	Teller
9	Jane	Grossman	2006-05-03	Teller
10	Paula	Roberts	2006-07-27	Head Teller
12	Samantha	Jameson	2007-01-08	Teller
13	John	Blake	2004-05-11	Head Teller
14	Cindy	Mason	2006-08-09	Teller
15	Frank	Portman	2007-04-01	Teller
16	Theresa	Markham	2005-03-15	Head Teller
17	Beth	Fowler	2006-06-29	Teller
18	Rick	Tulman	2006-12-12	Teller

15 rows in set (0.00 sec)

Die Ausgabe zeigt, dass alle vier Chefkassierer in der Ergebnismenge aufgeführt werden, allerdings zusammen mit allen anderen Angestellten, die seit dem 1. Januar 2006 angefangen haben. Auf 15 der 18 Mitarbeiter in der `employee`-Tabelle trifft mindestens eine der beiden Bedingungen zu. Also: Wenn Sie Bedingungen mit dem `and`-Operator trennen, müssen *alle* diese Bedingungen `true` sein, damit die betreffende Zeile in die Ergebnismenge aufgenommen wird; wenn Sie hingegen `or` verwenden, genügt es schon, wenn nur *eine* der Bedingungen `true` 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 verwenden Sie runde Klammern, um die Bedingungen zu Gruppen zusammenzufassen. Die nächste Abfrage besagt, dass nur Chefkassierer, die nach dem 1. Januar 2006 ins Unternehmen kamen, *oder* Kassierer, die nach dem 1. Januar 2007 ihre Arbeit aufnahmen, in die Ergebnismenge kommen:

```
mysql> SELECT emp_id, fname, lname, start_date, title
-> FROM employee
-> WHERE (title = 'Head Teller' AND start_date > '2006-01-01')
-> OR (title = 'Teller' AND start_date > '2007-01-01');
```

```

+-----+-----+-----+-----+-----+
| emp_id | fname | lname | start_date | title |
+-----+-----+-----+-----+-----+
|      6 | Helen | Fleming | 2008-03-17 | Head Teller |
|      7 | Chris | Tucker | 2008-09-15 | Teller |
|     10 | Paula | Roberts | 2006-07-27 | Head Teller |
|     12 | Samantha | Jameson | 2007-01-08 | Teller |
|     15 | Frank | Portman | 2007-04-01 | Teller |
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

```

Verwenden Sie immer Klammern, um separate Gruppen von Bedingungen zusammenzufassen oder verschiedene Operatoren zu verwenden. 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 keine Liste von Mitarbeitern und zugehörigen Abteilungen, sondern eine Liste der Abteilungen mit der jeweiligen Anzahl der Mitarbeiter. Wenn Sie die `group by`-Klausel verwenden, können Sie auch eine `having`-Klausel einsetzen: Diese filtert die gruppierten Daten in der gleichen Weise, wie es die `where`-Klausel mit Rohdaten tut.

Werfen Sie hier einen kurzen Blick auf eine Abfrage, die alle Angestellten in allen Abteilungen zählt und die Namen der Abteilungen zurückliefert, in denen mehr als zwei Personen arbeiten:

```

mysql> SELECT d.name, count(e.emp_id) num_employees
-> FROM department d INNER JOIN employee e
-> ON d.dept_id = e.dept_id
-> GROUP BY d.name
-> HAVING count(e.emp_id) > 2;
+-----+-----+
| name          | num_employees |
+-----+-----+
| Administration |          3 |
| Operations    |         14 |
+-----+-----+
2 rows in set (0.00 sec)

```

Ich wollte diese beiden Klauseln nur kurz ansprechen, damit Sie später in diesem Buch bereits darauf vorbereitet sind. Die beiden Klauseln 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` verwendet.

Die order by-Klausel

Im Allgemeinen haben die Zeilen der Ergebnismenge einer Abfrage keine besondere Reihenfolge. Wenn Sie eine geordnete 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 eine frühere Abfrage der `account`-Tabelle:

```
mysql> SELECT open_emp_id, product_cd  
-> FROM account;
```

open_emp_id	product_cd
10	CHK
10	SAV
10	CD
10	CHK
10	SAV
13	CHK
13	MM
1	CHK
1	SAV
1	MM
16	CHK
1	CHK
1	CD
10	CD
16	CHK
16	SAV
1	CHK
1	MM
1	CD
16	CHK
16	BUS
10	BUS
16	CHK
13	SBL

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

Wenn Sie versuchen, die Daten für jeden Mitarbeiter zu analysieren, ist es hilfreich, die Ergebnisse anhand der Spalte `open_emp_id` zu sortieren. Dazu genügt es, diese Spalte in die `order by`-Klausel aufzunehmen:

```
mysql> SELECT open_emp_id, product_cd  
-> FROM account  
-> ORDER BY open_emp_id;
```

open_emp_id	product_cd
-------------	------------

```

+-----+
|      1 | CHK |
|      1 | SAV |
|      1 | MM  |
|      1 | CHK |
|      1 | CD  |
|      1 | CHK |
|      1 | MM  |
|      1 | CD  |
|     10 | CHK |
|     10 | SAV |
|     10 | CD  |
|     10 | CHK |
|     10 | SAV |
|     10 | CD  |
|     10 | BUS |
|     13 | CHK |
|     13 | MM  |
|     13 | SBL |
|     16 | CHK |
|     16 | CHK |
|     16 | SAV |
|     16 | CHK |
|     16 | BUS |
|     16 | CHK |
+-----+
24 rows in set (0.00 sec)

```

Nun lässt sich einfacher erkennen, welche Konten von welchem Angestellten eröffnet wurden. Noch besser wäre es vielleicht, wenn man gewährleisten könnte, dass die Kontenarten für jeden Mitarbeiter in der gleichen Reihenfolge angezeigt werden. Dies erreichen Sie, indem Sie die Spalte `product_cd` hinter der Spalte `open_emp_id` in der `order by`-Klausel aufführen:

```

mysql> SELECT open_emp_id, product_cd
-> FROM account
-> ORDER BY open_emp_id, product_cd;
+-----+
| open_emp_id | product_cd |
+-----+
|          1 | CD         |
|          1 | CD         |
|          1 | CHK        |
|          1 | CHK        |
|          1 | CHK        |
|          1 | MM         |
|          1 | MM         |
|          1 | SAV        |
|         10 | BUS        |
|         10 | CD         |
|         10 | CD         |
|         10 | CHK        |
|         10 | CHK        |
|         10 | SAV        |

```

```

+-----+
|      10 | SAV |
|      13 | CHK |
|      13 | MM  |
|      13 | SBL |
|      16 | BUS |
|      16 | CHK |
|      16 | SAV |
+-----+
24 rows in set (0.00 sec)

```

Die Ergebnismenge wurde nun zuerst nach Mitarbeiter-ID und dann nach Kontenart sortiert. Dabei ist bedeutsam, in welcher Reihenfolge die Spalten in der `order by`-Klausel aufgeführt werden.

Auf- und absteigende Sortierung

Beim Sortieren haben Sie die Möglichkeit, mit den Schlüsselwörtern `asc` und `desc` eine *aufsteigende* oder *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 Konten nach ihrem Saldo auf, beginnend mit dem höchsten Saldo:

```

mysql> SELECT account_id, product_cd, open_date, avail_balance
-> FROM account
-> ORDER BY avail_balance DESC;

```

```

+-----+
| account_id | product_cd | open_date | avail_balance |
+-----+
|      29 | SBL      | 2004-02-22 | 50000.00 |
|      28 | CHK      | 2003-07-30 | 38552.05 |
|      24 | CHK      | 2002-09-30 | 23575.12 |
|      15 | CD       | 2004-12-28 | 10000.00 |
|      27 | BUS      | 2004-03-22 | 9345.55 |
|      22 | MM       | 2004-10-28 | 9345.55 |
|      12 | MM       | 2004-09-30 | 5487.09 |
|      17 | CD       | 2004-01-12 | 5000.00 |
|      18 | CHK      | 2001-05-23 | 3487.19 |
|       3 | CD       | 2004-06-30 | 3000.00 |
|       4 | CHK      | 2001-03-12 | 2258.02 |
|      13 | CHK      | 2004-01-27 | 2237.97 |
|       8 | MM       | 2002-12-15 | 2212.50 |
|      23 | CD       | 2004-06-30 | 1500.00 |
|       1 | CHK      | 2000-01-15 | 1057.75 |
|       7 | CHK      | 2002-11-23 | 1057.75 |
|      11 | SAV      | 2000-01-15 | 767.77 |
|      10 | CHK      | 2003-09-12 | 534.12 |
|       2 | SAV      | 2000-01-15 | 500.00 |
|      19 | SAV      | 2001-05-23 | 387.99 |
|       5 | SAV      | 2001-03-12 | 200.00 |
+-----+

```

```

+-----+-----+-----+-----+
|      21 | CHK | 2003-07-30 | 125.67 |
|      14 | CHK | 2002-08-24 | 122.37 |
|      25 | BUS | 2002-10-01 | 0.00   |
+-----+-----+-----+-----+
24 rows in set (0.05 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. In Anhang B wird die `limit`-Klausel zusammen mit anderen nicht dem ANSI-Standard entsprechenden Erweiterungen vorgestellt.

Sortieren mit Ausdrücken

Resultate anhand von Spaltendaten zu sortieren ist ja schön und gut, aber manchmal möchte man doch gern ein anderes Sortierkriterium anwenden, eines, das nicht in der Datenbank gespeichert ist und möglicherweise auch nirgendwo sonst in der Abfrage erscheint. In solche Fällen können Sie in Ihre `order by`-Klausel auch einen Ausdruck schreiben. Vielleicht möchten Sie ja Ihre Kundendaten nach den letzten drei Ziffern der Steuernummer sortieren (in Amerika und in unserem Beispiel ist das die Federal ID):

```

mysql> SELECT cust_id, cust_type_cd, city, state, fed_id
-> FROM customer
-> ORDER BY RIGHT(fed_id, 3);
+-----+-----+-----+-----+
| cust_id | cust_type_cd | city       | state | fed_id |
+-----+-----+-----+-----+
|      1 | I             | Lynnfield | MA    | 111-11-1111 |
|     10 | B             | Salem   | NH    | 04-1111111 |
|      2 | I             | Woburn   | MA    | 222-22-2222 |
|     11 | B             | Wilmington | MA    | 04-2222222 |
|      3 | I             | Quincy   | MA    | 333-33-3333 |
|     12 | B             | Salem   | NH    | 04-3333333 |
|     13 | B             | Quincy   | MA    | 04-4444444 |
|      4 | I             | Waltham  | MA    | 444-44-4444 |
|      5 | I             | Salem   | NH    | 555-55-5555 |
|      6 | I             | Waltham  | MA    | 666-66-6666 |
|      7 | I             | Wilmington | MA    | 777-77-7777 |
|      8 | I             | Salem   | NH    | 888-88-8888 |
|      9 | I             | Newton   | MA    | 999-99-9999 |
+-----+-----+-----+-----+
13 rows in set (0.24 sec)

```

Diese Abfrage nutzt die eingebaute Funktion `right()`, um die letzten drei Ziffern der Spalte `fed_id` zu extrahieren, und sortiert dann die Zeilen anhand dieses Werts.

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. Ein

Beispiel: Angenommen, Sie möchten Ihre Daten anhand der zweiten und fünften Rückgabespalte einer Abfrage sortieren, dann könnten Sie wie folgt vorgehen:

```
mysql> SELECT emp_id, title, start_date, fname, lname
-> FROM employee
-> ORDER BY 2, 5;
```

emp_id	title	start_date	fname	lname
13	Head Teller	2004-05-11	John	Blake
6	Head Teller	2008-03-17	Helen	Fleming
16	Head Teller	2005-03-15	Theresa	Markham
10	Head Teller	2006-07-27	Paula	Roberts
5	Loan Manager	2007-11-14	John	Gooding
4	Operations Manager	2006-04-24	Susan	Hawthorne
1	President	2005-06-22	Michael	Smith
17	Teller	2006-06-29	Beth	Fowler
9	Teller	2006-05-03	Jane	Grossman
12	Teller	2007-01-08	Samantha	Jameson
14	Teller	2006-08-09	Cindy	Mason
8	Teller	2006-12-02	Sarah	Parker
15	Teller	2007-04-01	Frank	Portman
7	Teller	2008-09-15	Chris	Tucker
18	Teller	2006-12-12	Rick	Tulman
11	Teller	2004-10-23	Thomas	Ziegler
3	Treasurer	2005-02-09	Robert	Tyler
2	Vice President	2006-09-12	Susan	Barker

18 rows in set (0.00 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 C.

Übung 3-1

Fragen Sie die Personalnummer (Employee ID), den Vornamen und den Nachnamen aller Bankangestellten ab. Sortieren Sie die Daten zuerst nach Nachnamen und dann nach Vornamen.

Übung 3-2

Fragen Sie die Kontonummer (Account ID), die Kundennummer (Customer ID) und den Saldo (Balance) für alle Konten ab, deren Status gleich 'ACTIVE' und deren Saldo größer als \$2.500 ist.

Übung 3-3

Schreiben Sie für die account-Tabelle eine Abfrage, die die IDs der Mitarbeiter liefert, die Konten eröffnet haben (verwenden Sie dazu die Spalte account.open_emp_id). Für jeden Angestellten 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 p.product_cd, a.cust_id, a.avail_balance
-> FROM product p INNER JOIN account <1>
-> ON p.product_cd = <2>
-> WHERE p.<3> = 'ACCOUNT'
-> ORDER BY <4>, <5>;
```

product_cd	cust_id	avail_balance
CD	1	3000.00
CD	6	10000.00
CD	7	5000.00
CD	9	1500.00
CHK	1	1057.75
CHK	2	2258.02
CHK	3	1057.75
CHK	4	534.12
CHK	5	2237.97
CHK	6	122.37
CHK	8	3487.19
CHK	9	125.67
CHK	10	23575.12
CHK	12	38552.05
MM	3	2212.50
MM	4	5487.09
MM	9	9345.55
SAV	1	500.00
SAV	2	200.00
SAV	4	767.77
SAV	8	387.99

21 rows in set (0.09 sec)

Dies ist ein Auszug aus dem Buch "Einführung in SQL, 2. Auflage", ISBN 978-3-89721-937-3
<http://www.oreilly.de/catalog/earning/sql2ger/>
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009