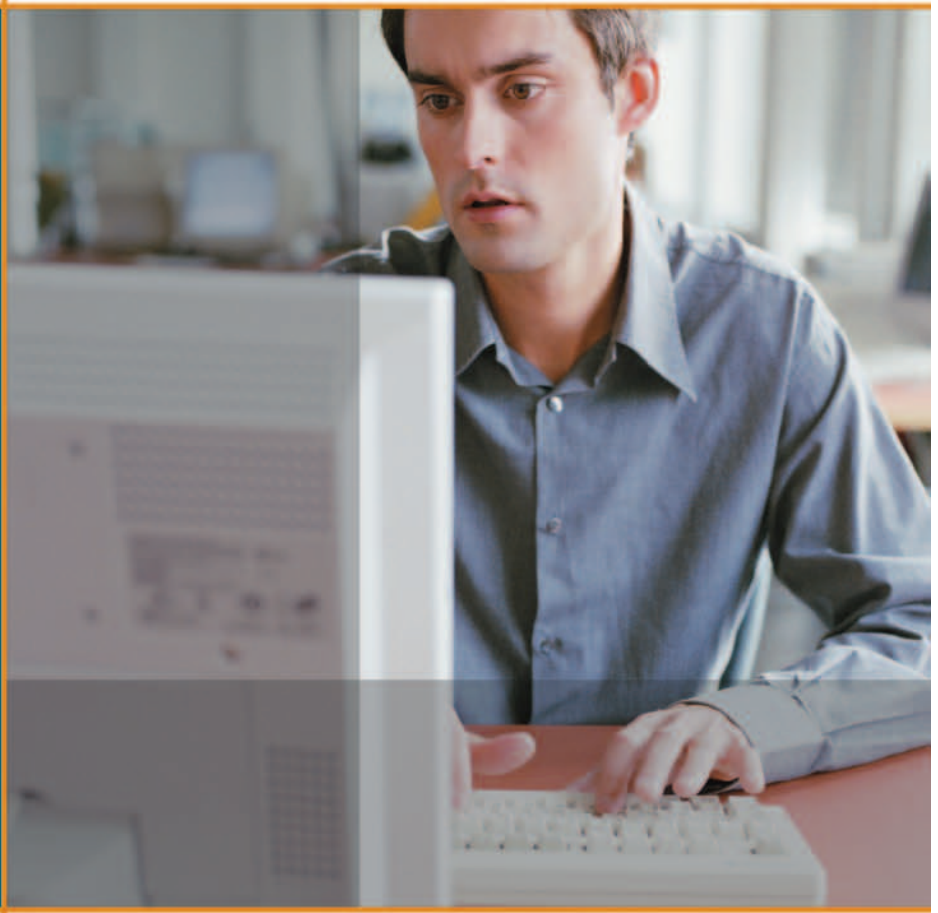
PEARSON Education

ADDISON-WESLEY

MySQL Press



MySQL AB

Das offizielle MySQL® 5 Handbuch

Konfiguration, Administration, Entwicklung und Optimierung

CD Das komplette Buch als PDF

ADDISON-WESLEY

3

Einführung in MySQL: ein MySQL-Tutorial

Dieses Kapitel enthält eine Einführung in MySQL. Hier wird erläutert, wie mithilfe des `mysql`-Clientprogramms eine einfache Datenbank erstellt und verwendet wird. `mysql` (das manchmal auch als „Terminalmonitor“ oder einfach als „Monitor“ bezeichnet wird) ist ein interaktives Programm, das es Ihnen ermöglicht, Verbindungen mit einem MySQL Server herzustellen, Abfragen auszuführen und die Ergebnisse anzuzeigen. `mysql` kann auch im Stapelbetrieb verwendet werden: Sie legen Ihre Abfragen im Voraus in einer Datei ab und weisen `mysql` dann an, den Inhalt dieser Datei auszuführen. Wir werden hier beide Methoden der Verwendung von `mysql` behandeln.

Um eine Liste der Optionen anzuzeigen, die `mysql` bietet, rufen Sie es mit der Option `--help` auf:

```
shell> mysql --help
```

Dieses Kapitel setzt voraus, dass `mysql` auf Ihrem Computer installiert ist und dass ein MySQL Server verfügbar ist, mit dem Sie eine Verbindung herstellen können. Sollte dies nicht der Fall sein, dann wenden Sie sich an Ihren MySQL-Administrator. (Wenn *Sie selbst* der Administrator sind, lesen Sie die erforderlichen Abschnitte dieses Handbuchs, z. B. Kapitel 5 auf Seite 251.)

Dieses Kapitel beschreibt den gesamten Vorgang der Einrichtung und Verwendung einer Datenbank. Wenn Sie nur am Zugriff auf eine vorhandene Datenbank interessiert sind, können Sie die Abschnitte überspringen, die beschreiben, wie Datenbanken und die darin enthaltenen Tabellen erstellt werden.

Da dieses Kapitel lediglich ein Tutorial ist, werden natürlich viele Detailangaben weggelassen. Um mehr zu den hier behandelten Themen zu erfahren, lesen Sie die einschlägigen Abschnitte dieses Handbuchs.

3.1 Verbindung zum Server herstellen und trennen

Um eine Verbindung mit dem Server herzustellen, müssen Sie beim Aufruf von `mysql` normalerweise einen MySQL-Benutzernamen und in aller Regel auch ein Passwort angeben. Wird der Server auf einem anderen System als demjenigen ausgeführt, an dem Sie sich anmelden, dann müssen Sie auch einen Hostnamen angeben. Wenden Sie sich an Ihrem Administrator, um zu erfahren, welche Parameter (Host- und Benutzernamen sowie Passwort) Sie für die Verbindung angeben müssen. Wenn Sie die korrekten Anmeldeinformationen kennen, sollten Sie wie folgt eine Verbindung herstellen können:

```
shell> mysql -h host -u user -p
Enter password: ********
```

`host` und `user` stehen für den Namen des Hosts, auf dem Ihr MySQL Server ausgeführt wird, bzw. den Benutzernamen Ihres MySQL-Kontos. Ersetzen Sie die Werte wie für Ihre Konfiguration erforderlich. Hierbei repräsentiert `********` Ihr Passwort; geben Sie dieses ein, wenn `mysql` die Aufforderung `Enter password:` anzeigt.

Funktioniert alles wie erwartet, dann werden ein paar einleitende Informationen gefolgt von der Eingabeaufforderung `mysql>` angezeigt:

```
shell> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 5.1.5-alpha-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Die Eingabeaufforderung `mysql>` zeigt an, dass `mysql` nun zur Entgegennahme von Befehlen bereit ist.

Wenn Sie sich an dem Computer angemeldet haben, auf dem auch MySQL ausgeführt wird, dann können Sie den Hostnamen weglassen und einfach Folgendes eingeben:

```
shell< mysql -u user -p
```

Wenn Sie beim Anmeldeversuch eine Fehlermeldung wie etwa

ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)

erhalten, bedeutet dies, dass der MySQL Server-Daemon (unter Unix) bzw. der MySQL Server-Dienst (unter Windows) nicht ausgeführt wird. Wenden Sie sich an den Administrator oder lesen Sie den für Ihr Betriebssystem vorgesehenen Abschnitt in Kapitel 2 auf Seite 59.

Hilfe bei anderen Problemen, die beim Anmelden häufig auftreten, finden Sie in Anhang A.2 auf Seite 1678.

Einige MySQL-Installationen gestatten Benutzern die Herstellung einer Verbindung zum auf dem lokalen Host laufenden Server als anonymer (d. h. nicht benannter) Benutzer. Sollte dies bei Ihrem System der Fall sein, dann sollten Sie eine Verbindung herstellen können, indem Sie `mysql` einfach ohne weitere Optionen aufrufen:

```
shell> mysql
```

Wenn Sie erfolgreich eine Verbindung hergestellt haben, können Sie diese jederzeit trennen, indem Sie `QUIT` (oder einfach `\q`) an der Eingabeaufforderung `mysql>` eingeben:

```
mysql> QUIT
Bye
```

Unter Unix können Sie die Trennung auch mit der Tastenkombination Strg+D durchführen.

Die meisten Beispiele in den nachfolgenden Abschnitten setzen voraus, dass Sie mit dem Server verbunden sind. Dieses wird durch die Eingabeaufforderung `mysql>` angezeigt.

3.2 Anfragen eingeben

Stellen Sie zunächst wie im vorhergehenden Abschnitt beschrieben eine Verbindung zum Server her. Durch diesen Vorgang wird noch keine Datenbank ausgewählt, was aber noch nicht problematisch ist. Wir wollen erst einmal sehen, wie man Abfragen absetzt, statt gleich mit den eigentlichen Datenbankfunktionen – dem Erstellen von Tabellen und dem Einladen von Daten in bzw. Abrufen dieser Daten aus den Tabellen – anzufangen. Dieser Abschnitt erläutert die Grundprinzipien der Befehlseingabe mithilfe verschiedener Abfragen, die Sie ausprobieren können, um sich mit der Funktionsweise von `mysql` vertraut zu machen.

Hier zunächst ein einfacher Befehl, der den Server bittet, seine Versionsnummer und das aktuelle Datum anzugeben. Geben Sie den Befehl wie hier gezeigt an der Eingabeaufforderung `mysql>` ein und betätigen Sie dann die Eingabetaste:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----------------+--------------+
| VERSION()       | CURRENT_DATE |
+-----------------+--------------+
| 5.1.2-alpha-log | 2005-10-11   |
+-----------------+--------------+
1 row in set (0.01 sec)
mysql>
```

Diese Abfrage veranschaulicht mehrere Aspekte von `mysql`:

- Ein Befehl besteht normalerweise aus einer SQL-Anweisung gefolgt von einem Semikolon. (Es gibt eine Reihe von Ausnahmen, bei denen das Semikolon weggelassen werden kann. Ein Beispiel ist das bereits weiter oben erwähnte `QUIT`; weitere werden folgen.)
- Wenn Sie einen Befehl absetzen, sendet `mysql` ihn zur Ausführung an den Server und zeigt die Ergebnisse an. Darauf folgt wieder eine neue Eingabeaufforderung `mysql>`, mit der angezeigt wird, dass nun ein neuer Befehl eingegeben werden kann.
- `mysql` zeigt die Abfrageausgabe in Tabellenform (d. h. als Zeilen und Spalten) an. Die erste Zeile enthält die Spaltenüberschriften. Alle nachfolgenden Zeilen sind Abfrageergebnisse. Normalerweise sind Spaltenüberschriften die Namen der Spalten, die aus den Datenbanktabellen abgerufen werden. Wenn Sie den Wert eines Ausdrucks statt einer Tabellenspalte (wie im obigen Beispiel) abrufen, beschriftet `mysql` die Spalte mit dem Ausdruck selbst.
- `mysql` zeigt an, wie viele Datensätze (Zeilen) zurückgegeben wurden und wie lange die Ausführung der Abfrage dauerte; hierdurch können Sie grob auf die Serverleistung schließen. Die Werte sind allerdings nicht sehr genau, denn sie geben nur eine normale Zeit statt der Prozessor- oder Systemzeit an, die zudem durch Faktoren wie der Serverauslastung und der Netzwerklatenz beeinflusst wird. (Aus Gründen der Übersichtlichkeit haben wir die Zeile „rows in set“ in einigen der in diesem Kapitel aufgeführten Beispiele weggelassen.)

Schlüsselwörter können in beliebiger Groß-/Kleinschreibung angegeben werden. Die folgenden Abfragen sind gleichwertig:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_datum;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Es folgt eine weitere Abfrage. Sie veranschaulicht, wie man `mysql` als einfachen Taschenrechner verwenden kann:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+------------------+---------+
| SIN(PI()/4)      | (4+1)*5 |
+------------------+---------+
| 0.70710678118655 |      25 |
+------------------+---------+
1 row in set (0.02 sec)
```

Die bislang gezeigten Abfragen waren vergleichsweise kurze, einzeilige Anweisungen. Sie können aber auch mehrere Anweisungen in eine Zeile schreiben. Schließen Sie sie jeweils mit einem Semikolon ab:

```
mysql> SELECT VERSION(); SELECT NOW();
+-----------------+
| VERSION()       |
+-----------------+
| 5.1.2-alpha-log |
+-----------------+
1 row in set (0.00 sec)

+---------------------+
| NOW()               |
+---------------------+
| 2005-10-11 15:15:00 |
+---------------------+
1 row in set (0.00 sec)
```

Ein Befehl muss nicht vollständig innerhalb einer einzelnen Zeile angegeben werden; insofern stellen auch längere Befehle über mehrere Zeilen kein Problem dar. `mysql` ermittelt das Ende Ihrer Anweisung anhand des schließenden Semikolons und nicht auf der Basis der Eingabezeile. (Anders gesagt, akzeptiert `mysql` frei formatierte Eingaben: Alle Eingabezeilen werden gesammelt, die Ausführung erfolgt aber erst, nachdem das Semikolon erkannt wurde.)

Hier ist eine einfache mehrzeilige Anweisung:

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+---------------+--------------+
| USER()        | CURRENT_DATE |
+---------------+--------------+
| jon@localhost | 2005-10-11   |
+---------------+--------------+
```

Beachten Sie in diesem Beispiel, wie die Eingabeaufforderung von `mysql>` auf `->` umschaltet, nachdem Sie die erste Zeile einer mehrzeiligen Abfrage eingegeben haben. Auf diese Weise zeigt `mysql` an, dass noch keine vollständige Anweisung erkannt wurde und weitere Eingaben erwartet werden. Diese Form der Eingabeaufforderung ist sehr praktisch, denn sie erlaubt Rückschlüsse auf erforderliche Eingaben. Sie wissen also immer, worauf `mysql` gerade wartet.

Wenn Sie einen Befehl, den Sie gerade eingeben, doch nicht ausführen wollen, können Sie ihn durch Eingabe von \c abbrechen:

```
mysql> SELECT
    -> USER()
    -> \c
mysql>
```

Beachten Sie auch hier die Eingabeaufforderung. Sie schaltet zurück auf `mysql>`, nachdem Sie `\c` eingegeben haben. So wird angezeigt, dass `mysql` auf einen neuen Befehl wartet.

Die folgende Tabelle zeigt alle Eingabeaufforderungen, auf die Sie treffen können, und fasst ferner zusammen, welche Rückschlüsse sie jeweils bezüglich des Zustandes von `mysql` erlauben.

Eingabeaufforderung	Bedeutung
`mysql>`	Bereit für einen neuen Befehl.
`->`	Erwartet die nächste Zeile einer mehrzeiligen Befehlseingabe.
`'>`	Erwartet die nächste Zeile und die Vervollständigung eines Strings, der mit einem einfachen Anführungszeichen (‘'’) begonnen wurde.
`">`	Erwartet die nächste Zeile und die Vervollständigung eines Strings, der mit einem doppelten Anführungszeichen (‘"’) begonnen wurde.
`` `> ``	Erwartet die nächste Zeile und die Vervollständigung eines Bezeichners, der mit einem Backtick (‘`’) begonnen wurde.
`/*>`	Erwartet die nächste Zeile und die Vervollständigung eines Kommentars, der mit `/*` begonnen wurde.

Mehrzeilige Anweisungen treten häufig ungewollt auf, wenn Sie eigentlich nur einen einzeiligen Befehl absetzen wollen, aber das abschließende Semikolon vergessen. In diesem Fall erwartet `mysql` eine weitere Eingabe:

```
mysql> SELECT USER()
    ->
```

Wenn das geschieht (d. h., wenn Sie glauben, dass Sie einen vollständigen Befehl eingegeben haben, aber die Eingabeaufforderung -> erscheint), dann wartet `mysql` in aller Regel auf das Semikolon. Bemerken Sie nicht sofort, welche Eingabeaufforderung hier angezeigt wird, dann sitzen Sie womöglich eine Zeit lang vor dem Computer, bevor Sie feststellen, was passiert ist. Geben Sie einfach ein Semikolon ein, um die Anweisung abzuschließen – sie wird dann von `mysql` ausgeführt:

```
mysql> SELECT USER()
    -> ;
+---------------+
| USER()        |
+---------------+
| jon@localhost |
+---------------+
```

Die Eingabeaufforderungen `'>` und `">` erscheinen bei der Erfassung von Strings (MySQL erwartet also die Vervollständigung eines Strings). In MySQL können Sie Strings entweder in ‘'’ oder ‘"’ setzen (z. B. `'hello'` oder `"goodbye"`). `mysql` erlaubt die Eingabe von Strings, die sich über mehrere Zeilen erstrecken. Wenn Sie die Eingabeaufforderung `'>` oder `">` sehen, bedeutet dies, dass Sie eine Zeile mit einem String eingegeben haben, der mit dem Anführungszeichen ‘'’ oder ‘"’

beginnt, diesen String aber noch nicht mit dem zugehörigen schließenden Anführungszeichen beendet haben. Hierdurch ist häufig erkennbar, dass Sie ein Anführungszeichen einzugeben vergessen haben. Ein Beispiel:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND alter < 30;
    '>
```

Wenn Sie diese SELECT-Anweisung eingeben und dann die Eingabetaste betätigen, werden Sie eine Zeit lang warten, ohne dass etwas passiert. Wenn Sie sich wundern, warum die Verarbeitung der Abfrage so lange dauert, werden Sie bald feststellen, dass die Eingabeaufforderung '> angezeigt wird. Sie besagt, dass mysql den Rest eines nicht abgeschlossenen Strings erwartet. (Erkennen Sie den Fehler in der Anweisung? Beim String 'Smith fehlt das zweite einzelne Anführungszeichen.)

Was können Sie nun tun? Die einfachste Möglichkeit besteht darin, den Befehl abzubrechen. Sie können in diesem Fall aber nicht einfach \c eingeben, da mysql dies als Teil des Strings interpretieren würde, den Sie vermeintlich eingeben. Geben Sie stattdessen zuerst das schließende Anführungszeichen (damit mysql weiß, dass der String abgeschlossen ist) und erst dann \c ein:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND alter < 30;
    '> '\c
mysql>
```

Die Eingabeaufforderung schaltet nun auf mysql> zurück und zeigt so an, dass mysql für einen neuen Befehl bereit ist.

Die Eingabeaufforderung `> ähnelt '> und ">, gibt aber an, dass Sie mit einem Backtick einen Bezeichner begonnen, aber noch nicht beendet haben.

Zu wissen, was die Eingabeaufforderungen '>, "> und `> bedeuten, ist wichtig, denn wenn Sie versehentlich einen nicht abgeschlossenen String eingeben, werden alle nachfolgend eingegebenen Zeilen von mysql ignoriert – einschließlich der Zeile mit dem Befehl QUIT. Dies kann insbesondere dann recht verwirrend sein, wenn Sie nicht wissen, dass Sie ein schließendes Anführungszeichen eingeben müssen, bevor Sie den aktuellen Befehl abbrechen können.

3.3 Eine Datenbank erzeugen und benutzen

Nachdem Sie nun wissen, wie Sie Befehle eingeben, sind Sie so weit, dass Sie auf eine Datenbank zugreifen können.

Angenommen, Sie halten bei sich zu Hause mehrere Haustiere (Ihre kleine „Menagerie“) und wollen nun verschiedene Informationen zu diesen Tieren verwalten. Dies können Sie tun, indem Sie Tabellen erstellen, die die gewünschten Informationen aufnehmen sollen, und diese Tabellen dann mit den erforderlichen Daten bestücken. Danach können Sie zu Ihren Tieren verschiedene Arten von Fragen beantworten, indem Sie Daten aus den Tabellen abrufen. In diesem Abschnitt erläutern wir, wie man

- eine Datenbank erstellt,
- eine Tabelle erstellt,
- Daten in eine Tabelle lädt,
- Daten auf verschiedene Weisen aus der Tabelle abruft,
- mehrere Tabellen verwendet.

Die Menageriedatenbank ist (bewusst) einfach gehalten, aber es ist auch nicht schwierig, sich Situationen aus dem täglichen Leben vorzustellen, in denen eine ähnliche Art von Datenbank zum Einsatz kommen könnte. Eine solche Datenbank könnte etwa von einem Landwirt, der seinen Tierbestand organisieren möchte, oder von einem Tierarzt zur Patientenverwaltung verwendet werden. Eine Menageriedistribution mit einigen der in den folgenden Abschnitten verwendeten Abfragen und Beispieldaten finden Sie auf der MySQL-Website. Sie ist in komprimierter Form als `tar`- (http://www.mysql.com/Downloads/Contrib/Examples/menagerie.tar.gz) und Zip-Archiv (http://www.mysql.com/Downloads/Contrib/Examples/menagerie.zip) verfügbar.

Verwenden Sie die `SHOW`-Anweisung, um zu ermitteln, welche Datenbanken derzeit auf dem Server vorhanden sind:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
| tmp      |
+----------+
```

Die Liste der Datenbanken sieht auf Ihrem Computer wahrscheinlich etwas anders aus, aber die Datenbanken `mysql` und `test` sind höchstwahrscheinlich vorhanden. Die Datenbank `mysql` ist erforderlich, da sie die Benutzerberechtigungen beschreibt. Die Datenbank `test` hingegen wird häufig als Spielplatz für Benutzer verwendet, die Dinge ausprobieren möchten.

Beachten Sie, dass Ihnen unter Umständen nicht alle Datenbanken angezeigt werden, wenn Ihnen die Berechtigung `SHOW DATABASES` fehlt. Siehe auch Abschnitt 13.5.1.3 auf Seite 907.

Wenn die Datenbank `test` vorhanden ist, versuchen Sie sie aufzurufen:

```
mysql> USE test
Database changed
```

Beachten Sie, dass `USE` ebenso wie `QUIT` kein Semikolon braucht. (Sie können solche Anweisungen aber nichtsdestoweniger mit einem Semikolon abschließen – hierdurch wird kein Schaden angerichtet.) Die `USE`-Anweisung ist auch auf andere Weise besonders: Sie muss in einer einzigen Zeile angegeben werden.

Sie können die Datenbank `test` für die folgenden Beispiele verwenden (vorausgesetzt, Sie haben Zugriff darauf); beachten Sie aber, dass alles, was Sie in dieser Datenbank erstellen, von allen anderen Benutzern, die darauf zugreifen dürfen, entfernt werden kann. Aus diesem Grund sollten Sie Ihren MySQL-Administrator besser bitten, eine eigene Datenbank verwenden zu dürfen. Wenn Sie Ihre Datenbank `menagerie` nennen wollen, dann muss der Administrator folgenden Befehl ausführen:

```
mysql> GRANT ALL ON menagerie.* TO 'your_mysql_name'@'your_client_host';
```

Hierbei ist `your_mysql_name` der Ihnen zugewiesene MySQL-Benutzername und `your_client_host` der Name des Hosts, von dem aus Sie die Verbindung zum Server herstellen.

3.3.1 Eine Datenbank erzeugen und auswählen

Wenn Ihr Administrator beim Konfigurieren der Berechtigungen eine Datenbank für Sie erstellt hat, können Sie diese sofort nutzen. Andernfalls müssen Sie sie selbst einrichten:

```
mysql> CREATE DATABASE menagerie;
```

Unter Unix wird bei Datenbanknamen – anders als bei SQL-Schlüsselwörtern – die Groß-/Kleinschreibung unterschieden. Deswegen muss der Datenbankname immer als `menagerie` und nicht als `Menagerie`, `MENAGERIE` oder in einer anderen Variante angegeben werden. Gleiches gilt für Tabellennamen. (Unter Windows gilt diese Einschränkung nicht; allerdings müssen Sie innerhalb einer Abfrage eine Datenbank oder Tabelle konsistent mit derselben Schreibung bezeichnen. Wir empfehlen jedoch aus verschiedenen Gründen, immer dieselbe Schreibweise zu nutzen, die beim Einrichten der Datenbank verwendet wurde.)

Hinweis: Wenn Sie beim Erstellen einer Datenbank eine Fehlermeldung wie etwa

ERROR 1044 (42000): Access denied for user 'monty'@'localhost' to database 'menagerie'

erhalten, bedeutet dies, dass Ihr Benutzerkonto nicht die zur Datenbankerstellung erforderlichen Berechtigungen hat. Besprechen Sie dies mit Ihrem Administrator oder lesen Sie Abschnitt 5.8 auf Seite 347.

Wenn Sie eine Datenbank erstellen, wird diese nicht automatisch für die Verwendung ausgewählt; Sie müssen dies ausdrücklich tun. Um `menagerie` also zur aktuellen Datenbank zu machen, verwenden Sie folgenden Befehl:

```
mysql> USE menagerie
Database changed
```

Ihre Datenbank muss nur einmal erstellt werden, sie muss aber jedes Mal, wenn Sie eine `mysql`-Sitzung beginnen, zur Verwendung ausgewählt werden. Dies tun Sie durch Absetzen einer `USE`-Anweisung wie im Beispiel gezeigt. Alternativ können Sie die Datenbank auch auf der Befehlszeile auswählen, wenn Sie `mysql` aufrufen. Geben Sie auf alle ggf. erforderlichen Verbindungsparameter folgend einfach den Datenbanknamen ein. Ein Beispiel:

```
shell> mysql -h host -u user -p menagerie
Enter password: ********
```

Beachten Sie, dass `menagerie` im gerade gezeigten Befehl *nicht* Ihr Passwort ist. Wenn Sie Ihr Passwort auf der Befehlszeile nach der Option `-p` angeben wollen, müssen Sie dies ohne zwischengeschaltetes Leerzeichen tun (z. B. als `-pmypassword`, nicht jedoch als `-p mypassword`). Allerdings wird von der Übermittlung des Passworts auf der Befehlszeile ohnehin abgeraten, weil es so anderen Benutzern offenbart werden könnte, die an Ihrem Computer angemeldet sind.

3.3.2 Eine Tabelle erzeugen

Das Erstellen einer Datenbank ist ganz einfach. Noch aber ist die Datenbank leer, wie Sie mit `SHOW TABLES` nachprüfen können:

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

Schwieriger zu entscheiden ist hingegen, wie die Struktur der Datenbank aussehen soll: Welche Tabellen brauchen Sie? Und welche Spalten sollen in diesen Tabellen enthalten sein?

Sie brauchen eine Tabelle, die einen Datensatz für jedes Ihrer Haustiere enthält. Diese Tabelle könnte man `haustier` nennen; sie sollte zumindest den Namen jedes Tieres enthalten. Da der Name an sich aber nicht besonders interessant ist, sollte die Tabelle auch weitere Informationen enthalten. Wenn beispielsweise mehrere Personen in Ihrer Familie Haustiere halten, würde es sich anbieten, die Besitzer der einzelnen Tiere aufzuführen. Ferner könnten einige grundlegende Beschreibungen eingetragen werden, z. B. die Tierart und das Geschlecht.

Und das Alter der Tiere? Das könnte zwar interessant sein, die Speicherung in einer Datenbank ist jedoch problematisch. Da sich das Alter im Laufe der Zeit ändert, müssten Sie Ihre Datensätze regelmäßig aktualisieren. Stattdessen ist es besser, einen festen Wert wie etwa das Geburtsdatum zu speichern. Wann immer Sie dann das Alter eines Tieres in Erfahrung bringen wollten, müssten Sie nur die Differenz zwischen dem aktuellen Datum und dem Geburtsdatum errechnen. MySQL bietet Funktionen, die derartige Rechenaufgaben erledigen können, sodass dies nicht weiter schwierig ist. Das Speichern des Geburtsdatums statt des Alters hat aber auch weitere Vorteile:

- Sie können die Datenbank für Aufgaben wie das automatische Erinnern an anstehende Tiergeburtstage verwenden. (Wenn Sie der Ansicht sind, dass eine solche Abfrage etwas albern ist, dann beachten Sie, dass Sie dieselbe Frage im Kontext einer Unternehmensdatenbank verwenden könnten, um Kunden zu ermitteln, denen Sie demnächst Geburtstagsgrüße senden wollen – die persönliche Note mit freundlicher Unterstützung Ihrer Datenbank.)
- Sie können das Alter auch in Relation zu anderen als dem aktuellen Datum berechnen. Wenn Sie etwa das Sterbedatum eines Tieres in der Datenbank ablegen, können Sie ganz leicht berechnen, wie alt es bei seinem Tod war.

Sie können sich wahrscheinlich eine Reihe andere Informationstypen vorstellen, die in der Tabelle `haustier` von Nutzen sein könnten, aber die bisher genannten sollen fürs Erste ausreichen: Name, Besitzer, Tierart, Geschlecht, Geburts- und Sterbedatum.

Das Layout Ihrer Tabelle legen Sie mit einer `CREATE TABLE`-Anweisung fest:

```
mysql> CREATE TABLE haustier (name VARCHAR(20), besitzer VARCHAR(20),
    -> gattung VARCHAR(20), geschlecht CHAR(1), geburtstag DATE, todestag DATE);
```

`VARCHAR` ist eine geeignete Wahl für die Spalten `name`, `besitzer` und `gattung`, denn die Werte dieser Spalten können in der Länge variieren. Die Längen in diesen Spaltendefinitionen müssen weder identisch sein noch alle den Wert `20` haben. Sie können eine beliebige Länge zwischen `1` und `65535` eingeben – je nachdem, was Ihnen am sinnvollsten erscheint. Haben Sie jedoch eine falsche Entscheidung getroffen und es stellt sich später heraus, dass Sie ein längeres Feld benötigen, dann bietet MySQL hierfür die `ALTER TABLE`-Anweisung an.

Zur Auswahl des Geschlechts Ihrer Tiere bieten sich mehrere Wertetypen an, z. B. `'m'` und `'w'` oder auch `'männlich'` und `'weiblich'`. Am einfachsten ist die Verwendung der Einzelzeichen `'m'` und `'w'`.

Der Datentyp `DATE` ist offensichtlich am geeignetsten für die Spalten `geburtstag` und `todestag`.

Wenn Sie eine Tabelle erstellt haben, sollte die Anweisung `SHOW TABLES` eine entsprechende Ausgabe erzeugen:

```
mysql> SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| haustier            |
+---------------------+
```

Um zu überprüfen, dass Ihre Tabelle wie gewünscht erstellt worden ist, verwenden Sie eine `DESCRIBE`-Anweisung:

```
mysql> DESCRIBE haustier;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| name       | varchar(20) | YES  |     | NULL    |       |
| besitzer   | varchar(20) | YES  |     | NULL    |       |
| gattung    | varchar(20) | YES  |     | NULL    |       |
| geschlecht | char(1)     | YES  |     | NULL    |       |
| geburtstag | date        | YES  |     | NULL    |       |
| todestag   | date        | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

Sie können `DESCRIBE` auch jederzeit aufrufen, wenn Sie die Spaltennamen Ihrer Tabelle oder die verwendeten Datentypen vergessen haben.

Weitere Informationen zu MySQL-Datentypen finden Sie in Kapitel 11 auf Seite 691.

3.3.3 Daten in Tabellen einladen

Nachdem Sie Ihre Tabelle erstellt haben, müssen Sie sie mit Daten füllen. Zu diesem Zweck sind die Anweisungen `LOAD DATA` und `INSERT` vorhanden.

Nehmen wir einmal an, dass sich die Angaben zu Ihren Haustieren wie in der folgenden Tabelle gezeigt zusammenfassen lassen (beachten Sie dabei, dass MySQL Datumsangaben im Format `'YYYY-MM-DD'` erwartet; dies unterscheidet sich unter Umständen von dem Format, mit dem Sie vertraut sind).

name	besitzer	gattung	geschlecht	geburtstag	todestag
Fluffy	Harold	Katze	w	1993-02-04	
Claws	Gwen	Katze	m	1994-03-17	
Buffy	Harold	Hund	w	1989-05-13	
Fang	Benny	Hund	m	1990-08-27	
Bowser	Diane	Hund	m	1979-08-31	1995-07-29
Chirpy	Gwen	Vogel	w	1998-09-11	
Whistler	Gwen	Vogel		1997-12-09	
Slim	Benny	Schlange	m	1996-04-29	

Da wir anfangs eine leere Tabelle haben, können Sie diese zunächst ganz einfach ausfüllen, indem Sie eine Textdatei erstellen, die jeweils eine Zeile pro Tier enthält. Die Inhalte der Datei können Sie dann mit einer einzigen Anweisung in die Tabelle einladen.

Erstellen Sie also eine Textdatei namens `haustier.txt` mit einem Datensatz pro Zeile, bei dem die einzelnen Werte durch Tabulatorzeichen voneinander getrennt sind; die Werte müssen dabei in der Reihenfolge eingegeben werden, in der Sie in der `CREATE TABLE`-Anweisung aufgeführt wurden. Bei fehlenden Werten (z. B. wenn das Geschlecht nicht bekannt oder das Ableben des Tieres noch nicht erfolgt ist) können Sie `NULL`-Werte eingeben. Diese geben Sie als `\N` (Backslash, großes N) in Ihre Textdatei ein. So würde der Datensatz für Whistler, den Vogel, wie folgt aussehen (der Whitespace zwischen den Werten ist ein Tabulatorzeichen):

name	besitzer	gattung	geschlecht	geburtstag	todestag
Whistler	Gwen	Vogel	\N	1997-12-09	\N

Um die Textdatei `haustier.txt` in die Tabelle `haustier` zu laden, genügt die folgende Anweisung:

```
mysql> LOAD DATA LOCAL INFILE '/path/haustier.txt' INTO TABLE haustier;
```

Beachten Sie, dass Sie, wenn Sie die Datei unter Windows mit einem Editor erstellt haben, der `\r\n` als Zeilenbegrenzer verwendet, Folgendes verwenden:

```
mysql> LOAD DATA LOCAL INFILE '/path/haustier.txt' INTO TABLE haustier
    -> LINES TERMINATED BY '\r\n';
```

(Auf einem Apple Macintosh unter Mac OS X würden Sie hingegen eher `LINES TERMINATED BY '\r'` benutzen.)

Sie können das Trennzeichen für Spaltenwerte und den Zeilenbegrenzer bei Bedarf auch ausdrücklich in der `LOAD DATA`-Anweisung angeben, die Standardeinstellungen sind jedoch das Tabulator- bzw. das Zeilenvorschubzeichen. Diese Einstellungen sind zum Einlesen unserer Datei `haustier.txt` geeignet.

Schlägt die Anweisung fehl, dann liegt das wahrscheinlich daran, dass bei der MySQL-Installation standardmäßig nicht die Funktionalität für lokale Dateien aktiviert ist. Wie Sie dies ändern, erfahren Sie in Abschnitt 5.7.4 auf Seite 346.

Zum sukzessiven Einfügen neuer Datensätze ist die `INSERT`-Anweisung am praktischsten. In ihrer einfachsten Form geben Sie Werte für jede Spalte in der Reihenfolge an, in der die Spalten in der `CREATE TABLE`-Anweisung aufgeführt waren. Nehmen wir etwa an, Diane bekommt einen neuen Hamster namens „Puffball“. Nun könnten Sie den erforderlichen neuen Datensatz wie folgt mithilfe einer `INSERT`-Anweisung angeben:

```
mysql> INSERT INTO haustier
    -> VALUES ('Puffball','Diane','Hamster','w','1999-03-30',NULL);
```

Beachten Sie, dass die String- und Datenwerte hier in Anführungszeichen gesetzt werden. Mit der `INSERT`-Anweisung können Sie `NULL` auch direkt eingeben, um einen fehlenden Wert darzustellen. `\N` würden Sie hier – anders als bei `LOAD DATA` – nicht verwenden.

Aus diesem Beispiel sollte klar werden, dass die Erstellung der Basisdatensätze mithilfe einzelner `INSERT`-Anweisungen erheblich mehr Tipparbeit bedeuten würde als die Verwendung einer einzelnen `LOAD DATA`-Anweisung.

3.3.4 Informationen aus einer Tabelle abfragen

Mit der SELECT-Anweisung können Sie Daten aus einer Tabelle abrufen. Die allgemeine Form dieser Anweisung sieht wie folgt aus:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

what_to_select gibt hierbei an, was Sie sehen wollen. Dies kann eine Liste mit Spalten oder alternativ * sein, wenn Sie alle Spalten anzeigen wollen. *which_table* gibt die Tabelle an, aus der Sie Daten abrufen wollen. Die WHERE-Klausel ist optional. Ist sie vorhanden, dann gibt *conditions_to_satisfy* eine oder mehrere Bedingungen an, die Datensätze erfüllen müssen, um abgerufen zu werden.

3.3.4.1 Alle Daten auswählen

Die einfachste Form von SELECT ruft alle in der Tabelle vorhandenen Daten ab:

```
mysql> SELECT * FROM haustier;
+----------+----------+----------+------------+------------+------------+
| name     | besitzer | gattung  | geschlecht | geburtstag | todestag   |
+----------+----------+----------+------------+------------+------------+
| Fluffy   | Harold   | Katze    | w          | 1993-02-04 | NULL       |
| Claws    | Gwen     | Katze    | m          | 1994-03-17 | NULL       |
| Buffy    | Harold   | Hund     | w          | 1989-05-13 | NULL       |
| Fang     | Benny    | Hund     | m          | 1990-08-27 | NULL       |
| Bowser   | Diane    | Hund     | m          | 1979-08-31 | 1995-07-29 |
| Chirpy   | Gwen     | Vogel    | w          | 1998-09-11 | NULL       |
| Whistler | Gwen     | Vogel    | NULL       | 1997-12-09 | NULL       |
| Slim     | Benny    | Schlange | m          | 1996-04-29 | NULL       |
| Puffball | Diane    | Hamster  | w          | 1999-03-30 | NULL       |
+----------+----------+----------+------------+------------+------------+
```

Diese Verwendung von SELECT ist nützlich, wenn Sie beispielsweise Ihre Tabelle überprüfen wollen, nachdem Sie gerade die ersten Daten eingeladen haben. Nehmen wir etwa an, dass Sie nicht ganz sicher sind, ob Sie für Bowser das richtige Geburtsdatum eingegeben haben. Dem Stammbaum entnehmen Sie, dass das korrekte Geburtsjahr nicht 1979, sondern 1989 lautet.

Nun gibt es zwei Möglichkeiten, den Fehler zu beheben:

- Sie korrigieren den Datensatz in der Datei `haustier.txt`, leeren die Tabelle mit DELETE und laden die Daten dann mit LOAD DATA neu ein:

  ```
  mysql> DELETE FROM haustier;
  mysql> LOAD DATA LOCAL INFILE 'haustier.txt' INTO TABLE haustier;
  ```

  Allerdings müssen Sie bei dieser Vorgehensweise auch die Daten für Puffball neu eingeben.

- Sie ändern nur den fehlerhaften Datensatz. Hierzu verwenden Sie eine UPDATE-Anweisung:

  ```
  mysql> UPDATE haustier SET geburtstag = '1989-08-31' WHERE name = 'Bowser';
  ```

  Mit UPDATE ändern Sie nur den fraglichen Datensatz – Sie müssen die Tabellendaten in diesem Fall nicht neu laden.

3.3.4.2 Bestimmte Zeilen auswählen

Wie im vorherigen Abschnitt gezeigt, ist das Abrufen der Daten einer ganzen Tabelle recht einfach. Sie lassen einfach die `WHERE`-Klausel in der `SELECT`-Anweisung weg. In der Regel wollen Sie aber nicht die gesamte Tabelle anzeigen – insbesondere dann nicht, wenn diese sehr groß ist. Stattdessen werden Sie meist an der Beantwortung einer bestimmten Frage interessiert sein. In diesem Fall müssen Sie Beschränkungen für die gewünschten Daten definieren. Betrachten wir einmal einige Auswahlabfragen dahingehend, welche Fragen diese zu Ihren Haustieren beantworten.

Sie können die Auswahl in Ihrer Tabelle auch auf bestimmte Datensätze beschränken. Wollen Sie z. B. die Änderungen am Geburtsdatum von Bowser überprüfen, dann wählen Sie den entsprechenden Datensatz wie folgt aus:

```
mysql> SELECT * FROM haustier WHERE name = 'Bowser';
+--------+----------+----------+------------+------------+------------+
| name   | besitzer | gattung  | geschlecht | geburtstag | todestag   |
+--------+----------+----------+------------+------------+------------+
| Bowser | Diane    | Hund     | m          | 1989-08-31 | 1995-07-29 |
+--------+----------+----------+------------+------------+------------+
```

Die Ausgabe bestätigt die korrekte Änderung der Jahresangabe von 1979 auf 1989.

Bei String-Vergleichen wird die Groß-/Kleinschreibung normalerweise ignoriert, weswegen Sie den Namen als `'bowser'`, `'BOWSER'` usw. angeben können. Das Abfrageergebnis ist stets gleich.

Sie können Bedingungen aber nicht nur für `name`, sondern für jede beliebige Spalte angeben. Wenn Sie beispielsweise wissen wollen, welche Tiere im Jahre 1998 oder danach geboren wurden, testen Sie die Spalte `geburtstag`:

```
mysql> SELECT * FROM haustier WHERE geburtstag >= '1998-1-1';
+----------+----------+---------+-------------+------------+----------+
| name     | besitzer | gattung | geschlecht  | geburtstag | todestag |
+----------+----------+---------+-------------+------------+----------+
| Chirpy   | Gwen     | Vogel   | w           | 1998-09-11 | NULL     |
| Puffball | Diane    | Hamster | w           | 1999-03-30 | NULL     |
+----------+----------+---------+-------------+------------+----------+
```

Sie können auch Bedingungen kombinieren, um etwa nach weiblichen Hunden zu suchen:

```
mysql> SELECT * FROM haustier WHERE gattung = 'Hund' AND geschlecht = 'w';
+-------+-----------+---------+-------------+------------+----------+
| name  | besitzer  | gattung | geschlecht  | geburtstag | todestag |
+-------+-----------+---------+-------------+------------+----------+
| Buffy | Harold    | Hund    | w           | 1989-05-13 | NULL     |
+-------+-----------+---------+-------------+------------+----------+
```

Obige Abfrage verwendet den Operator `AND` (logisches Und). Es gibt analog auch einen Operator `OR` (logisches Oder):

```
mysql> SELECT * FROM haustier WHERE gattung = 'Schlange' OR gattung = 'Vogel';
+----------+----------+----------+-------------+------------+----------+
| name     | besitzer | gattung  | geschlecht  | geburtstag | todestag |
+----------+----------+----------+-------------+------------+----------+
| Chirpy   | Gwen     | Vogel    | w           | 1998-09-11 | NULL     |
| Whistler | Gwen     | Vogel    | NULL        | 1997-12-09 | NULL     |
| Slim     | Benny    | Schlange | m           | 1996-04-29 | NULL     |
+----------+----------+----------+-------------+------------+----------+
```

AND und OR können auch gemischt verwendet werden, wobei AND jedoch Vorrang vor OR hat. Wenn Sie beide Operatoren verwenden, bietet sich die Verwendung von Klammern an, um exakt anzugeben, wie die Bedingungen gruppiert werden sollen:

```
mysql> SELECT * FROM haustier WHERE (gattung = 'Katze' AND geschlecht = 'm')
    -> OR (gattung = 'Hund' AND geschlecht = 'w');
+-------+----------+---------+------------+------------+----------+
| name  | besitzer | gattung | geschlecht | geburtstag | todestag |
+-------+----------+---------+------------+------------+----------+
| Claws | Gwen     | Katze   | m          | 1994-03-17 | NULL     |
| Buffy | Harold   | Hund    | w          | 1989-05-13 | NULL     |
+-------+----------+---------+------------+------------+----------+
```

3.3.4.3 Bestimmte Spalten auswählen

Wenn Sie nicht die gesamten Datensätze aus Ihrer Tabelle sehen wollen, dann führen Sie in der jeweiligen Abfrage die gewünschten Spalten durch Kommata getrennt auf. Wollen Sie also etwa wissen, wann Ihre Tiere geboren wurden, dann wählen Sie die Spalten name und geburtstag:

```
mysql> SELECT name, geburtstag FROM haustier;
+----------+------------+
| name     | geburtstag |
+----------+------------+
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Puffball | 1999-03-30 |
+----------+------------+
```

Um herauszufinden, wem welches Tier gehört, setzen Sie folgende Abfrage ab:

```
mysql> SELECT besitzer FROM haustier;
+----------+
| besitzer |
+----------+
| Harold   |
| Gwen     |
| Harold   |
| Benny    |
| Diane    |
| Gwen     |
| Gwen     |
| Benny    |
| Diane    |
+----------+
```

Beachten Sie, dass diese einfache Abfrage die Spalte besitzer jedes Datensatzes abruft; einige davon tauchen mehrfach auf. Um die Ausgabe zu optimieren, können Sie jeden eindeutigen Ergebnisdatensatz genau einmal ausgeben, indem Sie einfach das Schlüsselwort DISTINCT hinzufügen:

```
mysql> SELECT DISTINCT besitzer FROM haustier;
+----------+
| besitzer |
+----------+
| Benny    |
| Diane    |
| Gwen     |
| Harold   |
+----------+
```

Mithilfe einer `WHERE`-Klausel können Sie Datensatz- und Spaltenauswahl kombinieren. Um z. B. die Geburtsdaten von Hunden und Katzen anzuzeigen, verwenden Sie folgende Abfrage:

```
mysql> SELECT name, gattung, geburtstag FROM haustier
    -> WHERE gattung = 'Hund' OR gattung = 'Katze';
+--------+---------+------------+
| name   | gattung | geburtstag |
+--------+---------+------------+
| Fluffy | Katze   | 1993-02-04 |
| Claws  | Katze   | 1994-03-17 |
| Buffy  | Hund    | 1989-05-13 |
| Fang   | Hund    | 1990-08-27 |
| Bowser | Hund    | 1989-08-31 |
+--------+---------+------------+
```

3.3.4.4 Zeilen sortieren

Sie haben in den vorangegangenen Beispielen vielleicht bereits festgestellt, dass die Ergebnisdatensätze nicht in einer bestimmten Reihenfolge angezeigt werden. Häufig ist es jedoch einfacher, das Abfrageergebnis zu durchsuchen, wenn die Datensätze sinnvoll sortiert werden. Um ein Ergebnis zu sortieren, verwenden Sie die `ORDER BY`-Klausel.

Nachfolgend sind die Geburtstage der Tiere nach Datum sortiert aufgelistet:

```
mysql> SELECT name, geburtstag FROM haustier ORDER BY geburtstag;
+----------+------------+
| name     | geburtstag |
+----------+------------+
| Buffy    | 1989-05-13 |
| Bowser   | 1989-08-31 |
| Fang     | 1990-08-27 |
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Slim     | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy   | 1998-09-11 |
| Puffball | 1999-03-30 |
+----------+------------+
```

Bei zeichenbasierten Spalten erfolgt die Sortierung – wie bei allen anderen Vergleichsoperationen auch – normalerweise ohne Berücksichtigung der Groß-/Kleinschreibung. Das bedeutet, dass die Reihenfolge bei Spalten, die bis auf die Groß-/Kleinschreibung identisch sind, nicht definiert ist. Sie können die Sortierung einer Spalte unter Berücksichtigung der Groß-/Kleinschreibung durch Verwendung von `BINARY` wie folgt erzwingen: `ORDER BY BINARY` *`col_name`*.

Die Standardreihenfolge bei der Sortierung ist aufsteigend, d. h., die kleinsten Werte werden zuerst aufgeführt. Um in umgekehrter (absteigender) Reihenfolge zu sortieren, fügen Sie das Schlüsselwort `DESC` zum Namen der Spalte zu, nach der die Sortierung erfolgt:

```
mysql> SELECT name, geburtstag FROM haustier ORDER BY geburtstag DESC;
+----------+------------+
| name     | geburtstag |
+----------+------------+
| Puffball | 1999-03-30 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Claws    | 1994-03-17 |
| Fluffy   | 1993-02-04 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Buffy    | 1989-05-13 |
+----------+------------+
```

Sie können auch nach mehreren Spalten und diese jeweils mit eigener Reihenfolge sortieren. Um beispielsweise nach der Tierart in aufsteigender, dann dem Geburtsdatum innerhalb der Tierart in absteigender Reihenfolge (d. h. das jüngste Tier zuerst nennend) zu sortieren, verwenden Sie die folgende Abfrage:

```
mysql> SELECT name, gattung, geburtstag FROM haustier
    -> ORDER BY gattung, geburtstag DESC;
+----------+----------+------------+
| name     | gattung  | geburtstag |
+----------+----------+------------+
| Chirpy   | Vogel    | 1998-09-11 |
| Whistler | Vogel    | 1997-12-09 |
| Claws    | Katze    | 1994-03-17 |
| Fluffy   | Katze    | 1993-02-04 |
| Fang     | Hund     | 1990-08-27 |
| Bowser   | Hund     | 1989-08-31 |
| Buffy    | Hund     | 1989-05-13 |
| Puffball | Hamster  | 1999-03-30 |
| Slim     | Schlange | 1996-04-29 |
+----------+----------+------------+
```

Beachten Sie, dass das Schlüsselwort `DESC` nur für den Spaltennamen gilt, dem es direkt vorangestellt ist (`geburtstag`); die Sortierreihenfolge der Spalte `gattung` wird hiervon nicht berührt.

3.3.4.5 Datumsberechnungen

MySQL bietet eine Anzahl von Funktionen, mit denen Sie datumsbezogene Berechnungen durchführen können, um etwa Altersangaben zu ermitteln oder Teile aus Datumsangaben zu extrahieren.

Um zu bestimmen, wie alt Ihre Haustiere jeweils sind, berechnen Sie die Differenz im Jahresbestandteil des aktuellen und des Geburtsdatums und ziehen den Wert `1` ab, sofern das aktuelle Datum im Kalenderjahr vor dem Geburtsdatum liegt. Die folgende Abfrage zeigt für jedes Haustier das Geburtsdatum, das aktuelle Datum und das Alter in Jahren an.

```
mysql> SELECT name, geburtstag, CURDATE(),
    -> (YEAR(CURDATE())-YEAR(geburtstag))
    -> - (RIGHT(CURDATE(),5)<RIGHT(geburtstag,5))
    -> AS alter
    -> FROM haustier;
+----------+------------+------------+-------+
| name     | geburtstag | CURDATE()  | alter |
+----------+------------+------------+-------+
| Fluffy   | 1993-02-04 | 2003-08-19 |    10 |
| Claws    | 1994-03-17 | 2003-08-19 |     9 |
| Buffy    | 1989-05-13 | 2003-08-19 |    14 |
| Fang     | 1990-08-27 | 2003-08-19 |    12 |
| Bowser   | 1989-08-31 | 2003-08-19 |    13 |
| Chirpy   | 1998-09-11 | 2003-08-19 |     4 |
| Whistler | 1997-12-09 | 2003-08-19 |     5 |
| Slim     | 1996-04-29 | 2003-08-19 |     7 |
| Puffball | 1999-03-30 | 2003-08-19 |     4 |
+----------+------------+------------+-------+
```

Hierbei extrahiert `YEAR()` die Jahreszahl aus dem Datum, während mit `RIGHT()` die fünf Zeichen ganz rechts im Datum (`MM-DD`, also der Monat und der Tag) ermittelt werden. Der Teil des Ausdrucks, der die Werte `MM-DD` vergleicht, ist entweder `1` oder `0`. Hiermit wird von der Jahresdifferenz ggf. ein Jahr abgezogen, sofern `CURDATE()` (das aktuelle Datum) im Kalenderjahr vor `geburtstag` liegt. Der gesamte Ausdruck wirkt ein wenig unübersichtlich, weswegen ein *Alias* (`alter`) verwendet wird, um die Beschriftung der Ausgabespalte sinnvoller zu gestalten.

Die Abfrage funktioniert zwar, aber das Ergebnis ließe sich einfacher erfassen, wenn die Datensätze in einer bestimmten Reihenfolge angezeigt würden. Dies ist durch Ergänzen der Klausel `ORDER BY name` möglich, die die Ausgaben dem Namen nach sortiert:

```
mysql> SELECT name, geburtstag, CURDATE(),
    -> (YEAR(CURDATE())-YEAR(geburtstag))
    -> - (RIGHT(CURDATE(),5)<RIGHT(geburtstag,5))
    -> AS alter
    -> FROM haustier ORDER BY name;
+----------+------------+------------+-------+
| name     | geburtstag | CURDATE()  | alter |
+----------+------------+------------+-------+
| Bowser   | 1989-08-31 | 2003-08-19 |    13 |
| Buffy    | 1989-05-13 | 2003-08-19 |    14 |
| Chirpy   | 1998-09-11 | 2003-08-19 |     4 |
| Claws    | 1994-03-17 | 2003-08-19 |     9 |
| Fang     | 1990-08-27 | 2003-08-19 |    12 |
| Fluffy   | 1993-02-04 | 2003-08-19 |    10 |
| Puffball | 1999-03-30 | 2003-08-19 |     4 |
| Slim     | 1996-04-29 | 2003-08-19 |     7 |
| Whistler | 1997-12-09 | 2003-08-19 |     5 |
+----------+------------+------------+-------+
```

Um die Ausgabe nach dem Alter statt dem Namen zu sortieren, verwenden Sie einfach eine andere `ORDER BY`-Klausel:

```
mysql> SELECT name, geburtstag, CURDATE(),
    -> (YEAR(CURDATE())-YEAR(geburtstag))
    -> - (RIGHT(CURDATE(),5)<RIGHT(geburtstag,5))
    -> AS alter
    -> FROM haustier ORDER BY alter;
```

```
+----------+------------+------------+-------+
| name     | geburtstag | CURDATE()  | alter |
+----------+------------+------------+-------+
| Chirpy   | 1998-09-11 | 2003-08-19 |     4 |
| Puffball | 1999-03-30 | 2003-08-19 |     4 |
| Whistler | 1997-12-09 | 2003-08-19 |     5 |
| Slim     | 1996-04-29 | 2003-08-19 |     7 |
| Claws    | 1994-03-17 | 2003-08-19 |     9 |
| Fluffy   | 1993-02-04 | 2003-08-19 |    10 |
| Fang     | 1990-08-27 | 2003-08-19 |    12 |
| Bowser   | 1989-08-31 | 2003-08-19 |    13 |
| Buffy    | 1989-05-13 | 2003-08-19 |    14 |
+----------+------------+------------+-------+
```

Eine ähnliche Abfrage kann verwendet werden, um das erreichte Alter bereits verstorbener Tiere zu bestimmen. Welche Tiere dies sind, ermitteln Sie durch die Überprüfung, ob für `todestag` der Wert `NULL` lautet. Bei denjenigen Datensätzen, bei denen der Wert nicht `NULL` ist, berechnen Sie die Differenz zwischen den Werten `todestag` und `geburtstag`:

```
mysql> SELECT name, geburtstag, todestag,
    -> (YEAR(todestag)-YEAR(geburtstag)) - (RIGHT(todestag,5)<RIGHT(geburtstag,5))
    -> AS alter
    -> FROM haustier WHERE todestag IS NOT NULL ORDER BY alter;
+--------+------------+------------+-------+
| name   | geburtstag | todestag   | alter |
+--------+------------+------------+-------+
| Bowser | 1989-08-31 | 1995-07-29 |     5 |
+--------+------------+------------+-------+
```

Die Abfrage verwendet `todestag IS NOT NULL` statt `todestag <> NULL`, da `NULL` ein Sonderwert ist, der nicht mithilfe der üblichen Vergleichsoperatoren verglichen werden kann. Wir werden später noch darauf eingehen. Siehe auch Abschnitt 3.3.4.6 auf Seite 217.

Was aber, wenn Sie nun wissen wollen, welche Tiere im nächsten Monat Geburtstag haben? Für diese Art der Berechnung sind Jahr und Tag irrelevant: Sie müssen lediglich den Monat aus der Spalte `geburtstag` extrahieren. MySQL bietet mehrere Funktionen zur Extraktion von Datumsbestandteilen, z. B. `YEAR()`, `MONTH()` und `DAYOFMONTH()`. `MONTH()` ist für unsere Belange die passende Funktion. Um zu sehen, wie sie funktioniert, führen Sie eine einfache Abfrage aus, die den Wert von `geburtstag` und `MONTH(geburtstag)` anzeigt:

```
mysql> SELECT name, geburtstag, MONTH(geburtstag) FROM haustier;
+----------+------------+-------------------+
| name     | geburtstag | MONTH(geburtstag) |
+----------+------------+-------------------+
| Fluffy   | 1993-02-04 |                 2 |
| Claws    | 1994-03-17 |                 3 |
| Buffy    | 1989-05-13 |                 5 |
| Fang     | 1990-08-27 |                 8 |
| Bowser   | 1989-08-31 |                 8 |
| Chirpy   | 1998-09-11 |                 9 |
| Whistler | 1997-12-09 |                12 |
| Slim     | 1996-04-29 |                 4 |
| Puffball | 1999-03-30 |                 3 |
+----------+------------+-------------------+
```

Das Suchen von Tieren, die nächsten Monat Geburtstag haben, ist ebenfalls ganz einfach. Nehmen wir einmal an, es wäre April. In diesem Fall ist der Monatswert 4 – Sie suchen also nach Tieren, die im Mai (d. h. im Monat 5) geboren sind. Das geht wie folgt:

```
mysql> SELECT name, geburtstag FROM haustier WHERE MONTH(geburtstag) = 5;
+-------+------------+
| name  | geburtstag |
+-------+------------+
| Buffy | 1989-05-13 |
+-------+------------+
```

Wenn der aktuelle Monat Dezember ist, gibt es eine kleine Komplikation. Sie können nämlich nicht einfach die Monatsnummer (12) um den Wert 1 erhöhen – die Suche nach im Monat 13 geborenen Tieren brächte kein Ergebnis, weil es diesen Monat nicht gibt. Schließlich suchen Sie nach Tieren, die im Januar (Monat 1) geboren sind.

Sie können die Abfrage so verfassen, dass sie unabhängig vom aktuellen Monat funktioniert, sodass Sie die Nummer eines bestimmten Monats gar nicht verwenden müssen. DATE_ADD() erlaubt Ihnen das Addieren eines bestimmten Zeitintervalls für ein gegebenes Datum. Wenn Sie einen Monat zum aktuellen Wert von CURDATE() hinzuaddieren wollen, extrahieren Sie den Monatsbestandteil mit MONTH(). Das Ergebnis ist der Monat, in dem Sie nach Geburtstagen suchen müssen:

```
mysql> SELECT name, geburtstag FROM haustier
    -> WHERE MONTH(geburtstag) = MONTH(DATE_ADD(CURDATE(),INTERVAL 1 MONTH));
```

Eine andere Möglichkeit, diese Aufgabe zu lösen, besteht darin, 1 hinzuzuaddieren und dann den auf den aktuellen Monat folgenden Monat zu ermitteln, wobei mithilfe der Modulofunktion (MOD) der Monatswert auf 0 gesetzt wird, wenn er derzeit 12 beträgt:

```
mysql> SELECT name, geburtstag FROM haustier
    -> WHERE MONTH(geburtstag) = MOD(MONTH(CURDATE()), 12) + 1;
```

Beachten Sie, dass MONTH eine Zahl zwischen 1 und 12 zurückgibt. Ferner gibt MOD(something,12) eine Zahl zwischen 0 und 11 zurück. Insofern muss die Addition nach MOD() erfolgen, andernfalls würden wir von November (11) direkt zu Januar (1) springen.

3.3.4.6 Mit NULL-Werten arbeiten

Wenn Sie mit dem Wert NULL noch keine Erfahrung haben, kann Ihnen manche Überraschung bevorstehen. Vom Konzept her bezeichnet NULL einen „fehlenden unbekannten Wert“ und wird etwas anders behandelt als andere Werte. Um auf NULL zu prüfen, können Sie arithmetische Vergleichsoperatoren wie =, < oder <> nicht verwenden. Um dies zu demonstrieren, geben Sie einmal Folgendes ein:

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+----------+-----------+----------+----------+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+----------+-----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
+----------+-----------+----------+----------+
```

Offensichtlich lassen sich aus diesen Vergleichen keine sinngebenden Ergebnisse ziehen. Verwenden Sie stattdessen die Operatoren `IS NULL` und `IS NOT NULL`:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
+-----------+---------------+
```

Beachten Sie, dass `0` oder `NULL` in MySQL „falsch" entspricht, während alles andere wahr ist. Der standardmäßig „wahr" entsprechende Wert aus einer booleschen Operation ist `1`.

Die Sonderbehandlung für `NULL` ist der Grund dafür, warum im vorherigen Abschnitt die Frage, welche Tiere nicht mehr leben, mit `todestag IS NOT NULL` statt mit `todestag <> NULL` ermittelt werden musste.

Zwei `NULL` in einer `GROUP BY`-Klausel werden als gleichberechtigt betrachtet.

Wenn Sie eine `ORDER BY`-Klausel verwenden, werden `NULL`-Werte zuerst angezeigt, sofern Sie `ORDER BY ... ASC` angeben; im umgekehrten Fall `ORDER BY ... DESC` erscheinen sie hingegen am Ende der Ergebnislisten.

Ein häufiger Fehler beim Umgang mit `NULL` besteht in der Annahme, dass es nicht möglich ist, eine Null oder einen Leer-String in eine Spalte einzufügen, die als `NOT NULL` definiert ist. Das stimmt nicht. Es handelt sich dabei vielmehr um echte Werte, während `NULL` die Bedeutung „kein Wert" hat. Sie können dies ganz einfach wie folgt durch Verwendung von `IS [NOT] NULL` testen:

```
mysql> SELECT 0 IS NULL, 0 IS NOT NULL, '' IS NULL, '' IS NOT NULL;
+-----------+---------------+------------+----------------+
| 0 IS NULL | 0 IS NOT NULL | '' IS NULL | '' IS NOT NULL |
+-----------+---------------+------------+----------------+
|         0 |             1 |          0 |              1 |
+-----------+---------------+------------+----------------+
```

Es ist also durchaus möglich, eine Null oder einen Leer-String in eine `NOT NULL`-Spalte einzusetzen, denn diese Werte sind in der Tat `NOT NULL`. Siehe auch Anhang A.5.3 auf Seite 1704.

3.3.4.7 Übereinstimmende Suchmuster

MySQL ermöglicht einen Mustervergleich sowohl nach SQL-Standard als auch basierend auf erweiterten regulären Ausdrücken, wie man es von Unix-Hilfsprogrammen wie `vi`, `grep` und `sed` her kennt.

Beim SQL-Mustervergleich können Sie '_' für eine Übereinstimmung mit einzelnen Zeichen sowie '%' für eine Übereinstimmung mit einer beliebigen Anzahl von Zeichen (einschließlich 0 Zeichen) verwenden. In MySQL wird bei SQL-Mustern standardmäßig keine Unterscheidung der Groß-/Kleinschreibung vorgenommen. Einige Beispiele sind hier aufgeführt. Achten Sie darauf, in Verbindung mit SQL-Mustern nicht = oder <> zu verwenden, sondern die Vergleichsoperatoren `LIKE` bzw. `NOT LIKE`.

Namen, die mit 'b' beginnen, finden Sie so:

```
mysql> SELECT * FROM haustier WHERE name LIKE 'b%';
+--------+----------+---------+------------+------------+------------+
| name   | besitzer | gattung | geschlecht | geburtstag | todestag   |
+--------+----------+---------+------------+------------+------------+
| Buffy  | Harold   | Hund    | w          | 1989-05-13 | NULL       |
| Bowser | Diane    | Hund    | m          | 1989-08-31 | 1995-07-29 |
+--------+----------+---------+------------+------------+------------+
```

Namen, die auf 'fy' enden, finden Sie so:

```
mysql> SELECT * FROM haustier WHERE name LIKE '%fy';
+--------+----------+---------+------------+------------+----------+
| name   | besitzer | gattung | geschlecht | geburtstag | todestag |
+--------+----------+---------+------------+------------+----------+
| Fluffy | Harold   | Katze   | w          | 1993-02-04 | NULL     |
| Buffy  | Harold   | Hund    | w          | 1989-05-13 | NULL     |
+--------+----------+---------+------------+------------+----------+
```

Und Namen, die 'w' enthalten, finden Sie so:

```
mysql> SELECT * FROM haustier WHERE name LIKE '%w%';
+----------+----------+---------+------------+------------+------------+
| name     | besitzer | gattung | geschlecht | geburtstag | todestag   |
+----------+----------+---------+------------+------------+------------+
| Claws    | Gwen     | Katze   | m          | 1994-03-17 | NULL       |
| Bowser   | Diane    | Hund    | m          | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen     | Vogel   | NULL       | 1997-12-09 | NULL       |
+----------+----------+---------+------------+------------+------------+
```

Um Namen zu ermitteln, die aus genau fünf Zeichen bestehen, verwenden Sie fünf Instanzen des Musterzeichens '_':

```
mysql> SELECT * FROM haustier WHERE name LIKE '_____';
+-------+----------+---------+------------+------------+----------+
| name  | besitzer | gattung | geschlecht | geburtstag | todestag |
+-------+----------+---------+------------+------------+----------+
| Claws | Gwen     | Katze   | m          | 1994-03-17 | NULL     |
| Buffy | Harold   | Hund    | w          | 1989-05-13 | NULL     |
+-------+----------+---------+------------+------------+----------+
```

Der andere von MySQL unterstützte Mustervergleichstyp nutzt erweiterte reguläre Ausdrücke. Wenn Sie bei diesem Mustertyp auf Übereinstimmung testen, verwenden Sie die Operatoren `REGEXP` und `NOT REGEXP` (bzw. deren Synonyme `RLIKE` und `NOT RLIKE`).

Erweiterte reguläre Ausdrücke weisen u. a. die folgenden Merkmale auf:

- '`.`' stimmt mit genau einem beliebigen Zeichen überein.
- Eine Zeichenklasse '`[...]`' führt zur Übereinstimmung bei jedem Zeichen in den Klammern. Beispielsweise liegt bei '`[abc]`' eine Übereinstimmung mit '`a`', '`b`' oder '`c`' vor. Um einen Zeichenbereich anzugeben, verwenden Sie einen Bindestrich. '`[a-z]`' führt zur Übereinstimmung mit einem beliebigen Buchstaben, wohingegen '`[0-9]`' einer Übereinstimmung mit jeder Ziffer entspricht.

- '*' stimmt mit null oder mehr Instanzen des Elements überein, das ihm vorangeht. Beispielsweise liegt bei 'x*' eine Übereinstimmung mit einer beliebigen Anzahl von 'x'-Zeichen vor. Gleiches gilt bei '[0-9]*' für eine beliebige Anzahl von Ziffern und bei '.*' für eine beliebige Anzahl eines beliebigen Elements.
- Ein REGEXP-Mustervergleich ist erfolgreich, wenn eine Übereinstimmung des Musters an beliebiger Stelle im getesteten Wert vorliegt. (Hier liegt ein Unterschied zum LIKE-Mustervergleich, der nur erfolgreich ist, wenn das Muster mit dem gesamten Wert übereinstimmt.)
- Um einen Anker im Muster zu setzen, sodass es mit dem Anfang oder Ende des getesteten Wertes übereinstimmen muss, verwenden Sie '^' am Anfang bzw. '$' am Ende des Musters.

Um zu demonstrieren, wie erweiterte reguläre Ausdrücke funktionieren, haben wir die oben gezeigten LIKE-Abfragen so umgeschrieben, dass sie REGEXP verwenden.

Um Namen zu finden, die mit 'b' beginnen, verwenden Sie '^' für eine Übereinstimmung mit dem Namensanfang:

```
mysql> SELECT * FROM haustier WHERE name REGEXP '^b';
+--------+----------+---------+------------+------------+------------+
| name   | besitzer | gattung | geschlecht | geburtstag | todestag   |
+--------+----------+---------+------------+------------+------------+
| Buffy  | Harold   | Hund    | w          | 1989-05-13 | NULL       |
| Bowser | Diane    | Hund    | m          | 1989-08-31 | 1995-07-29 |
+--------+----------+---------+------------+------------+------------+
```

Wenn Sie einen REGEXP-Vergleich mit Unterscheidung der Groß-/Kleinschreibung wirklich erzwingen wollen, verwenden Sie das Schlüsselwort BINARY, um aus einem der Strings einen Binär-String zu machen. Bei folgender Abfrage liegt eine Übereinstimmung nur bei einem kleinen 'b' am Anfang des Namens vor:

```
mysql> SELECT * FROM haustier WHERE name REGEXP BINARY '^b';
```

Um Namen zu finden, die auf 'fy' enden, verwenden Sie '$' für eine Übereinstimmung mit dem Namensende:

```
mysql> SELECT * FROM haustier WHERE name REGEXP 'fy$';
+--------+----------+---------+------------+------------+----------+
| name   | besitzer | gattung | geschlecht | geburtstag | todestag |
+--------+----------+---------+------------+------------+----------+
| Fluffy | Harold   | Katze   | w          | 1993-02-04 | NULL     |
| Buffy  | Harold   | Hund    | w          | 1989-05-13 | NULL     |
+--------+----------+---------+------------+------------+----------+
```

Namen schließlich, die 'w' enthalten, finden Sie mit folgender Abfrage:

```
mysql> SELECT * FROM haustier WHERE name REGEXP 'w';
+----------+----------+---------+------------+------------+------------+
| name     | besitzer | gattung | geschlecht | geburtstag | todestag   |
+----------+----------+---------+------------+------------+------------+
| Claws    | Gwen     | Katze   | m          | 1994-03-17 | NULL       |
| Bowser   | Diane    | Hund    | m          | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen     | Vogel   | NULL       | 1997-12-09 | NULL       |
+----------+----------+---------+------------+------------+------------+
```

Da ein auf einem regulären Ausdruck basierendes Muster eine Übereinstimmung liefert, wenn es an beliebiger Stelle im Wert auftaucht, ist es in dieser Abfrage nicht notwendig, ein Jokerzeichen an den beiden Seiten des Musters zu setzen (bei einem SQL-Muster wäre dies hingegen erforderlich, um eine Übereinstimmung zu erzielen).

Um Namen zu ermitteln, die aus genau fünf Zeichen bestehen, verwenden Sie '^' und '$' für Übereinstimmungen für Anfang und Ende des Namens und fünf Instanzen von '.' dazwischen:

```
mysql> SELECT * FROM haustier WHERE name REGEXP '^.....$';
+-------+----------+---------+------------+------------+----------+
| name  | besitzer | gattung | geschlecht | geburtstag | todestag |
+-------+----------+---------+------------+------------+----------+
| Claws | Gwen     | Katze   | m          | 1994-03-17 | NULL     |
| Buffy | Harold   | Hund    | w          | 1989-05-13 | NULL     |
+-------+----------+---------+------------+------------+----------+
```

Sie könnten diese Abfrage auch mithilfe des Operators {*n*} („*n*-mal wiederholen") schreiben:

```
mysql> SELECT * FROM haustier WHERE name REGEXP '^.{5}$';
+-------+----------+---------+------------+------------+----------+
| name  | besitzer | gattung | geschlecht | geburtstag | todestag |
+-------+----------+---------+------------+------------+----------+
| Claws | Gwen     | Katze   | m          | 1994-03-17 | NULL     |
| Buffy | Harold   | Hund    | w          | 1989-05-13 | NULL     |
+-------+----------+---------+------------+------------+----------+
```

In Anhang C auf Seite 1755 finden Sie weitere Informationen zur Syntax regulärer Ausdrücke.

3.3.4.8 Zeilen zählen

Datenbanken werden häufig zur Beantwortung der Frage „Wie häufig taucht ein bestimmter Datentyp in einer Tabelle auf?" verwendet. So möchten Sie vielleicht wissen, wie viele Tiere Sie oder ein anderer Besitzer haben, oder verschiedene statistische Erhebungen zu Ihren Tieren machen.

Das Zählen aller vorhandenen Tiere entspricht der Frage „Wie viele Datensätze sind in der Tabelle `haustier` vorhanden?", denn es gibt genau einen Datensatz pro Tier. `COUNT(*)` zählt die Anzahl der Datensätze, d. h., die Abfrage zur Zählung Ihrer Tiere sieht wie folgt aus:

```
mysql> SELECT COUNT(*) FROM haustier;
+----------+
| COUNT(*) |
+----------+
|        9 |
+----------+
```

Weiter oben haben Sie die Namen der Leute abgerufen, die Tiere besitzen. Mit `COUNT()` können Sie etwa ermitteln, wie viele Tiere jeder Besitzer hat:

```
mysql> SELECT besitzer, COUNT(*) FROM haustier GROUP BY besitzer;
+----------+----------+
| besitzer | COUNT(*) |
+----------+----------+
| Benny    |        2 |
| Diane    |        2 |
| Gwen     |        3 |
| Harold   |        2 |
+----------+----------+
```

Beachten Sie, dass `GROUP BY` zur Gruppierung aller Datensätze für jeden Besitzer `besitzer` verwendet wird. Ohne diese Klausel erhalten Sie lediglich eine Fehlermeldung:

```
mysql> SELECT besitzer, COUNT(*) FROM haustier;
ERROR 1140 (42000): Mixing of GROUP columns (MIN(),MAX(),COUNT(),...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

`COUNT()` und `GROUP BY` sind praktisch, wenn Sie Ihre Daten auf verschiedene Arten charakterisieren wollen. Die folgenden Beispiele zeigen verschiedene Wege, statistische Daten zu den Tieren zu erheben.

Anzahl der Exemplare je Tierart:

```
mysql> SELECT gattung, COUNT(*) FROM haustier GROUP BY gattung;
+----------+----------+
| gattung  | COUNT(*) |
+----------+----------+
| Vogel    |        2 |
| Katze    |        2 |
| Hund     |        3 |
| Hamster  |        1 |
| Schlange |        1 |
+----------+----------+
```

Anzahl der Tiere je Geschlecht:

```
mysql> SELECT geschlecht, COUNT(*) FROM haustier GROUP BY geschlecht;
+------------+----------+
| geschlecht | COUNT(*) |
+------------+----------+
| NULL       |        1 |
| w          |        4 |
| m          |        4 |
+------------+----------+
```

(In dieser Ausgabe gibt `NULL` an, dass das Geschlecht unbekannt ist.)

Anzahl der Tiere je Kombination von Tierart und Geschlecht:

```
mysql> SELECT gattung, geschlecht, COUNT(*) FROM haustier GROUP BY gattung,
geschlecht;
+----------+------------+----------+
| gattung  | geschlecht | COUNT(*) |
+----------+------------+----------+
| Vogel    | NULL       |        1 |
| Vogel    | w          |        1 |
| Katze    | w          |        1 |
| Katze    | m          |        1 |
| Hund     | w          |        1 |
| Hund     | m          |        2 |
| Hamster  | w          |        1 |
| Schlange | m          |        1 |
+----------+------------+----------+
```

Wenn Sie `COUNT()` verwenden, müssen Sie nicht die gesamte Tabelle abrufen. Die obige Abfrage beispielsweise sieht wie folgt aus, wenn Sie nur für Hunde und Katzen durchgeführt wird:

```
mysql> SELECT gattung, geschlecht, COUNT(*) FROM haustier
    -> WHERE gattung = 'Hund' OR gattung = 'Katze'
    -> GROUP BY gattung, geschlecht;
+---------+------------+----------+
| gattung | geschlecht | COUNT(*) |
+---------+------------+----------+
| Katze   | w          |        1 |
| Katze   | m          |        1 |
| Hund    | w          |        1 |
| Hund    | m          |        2 |
+---------+------------+----------+
```

Oder, wenn Sie die Anzahl der Tiere je Geschlecht nur für diejenigen Tiere anzeigen wollen, deren Geschlecht auch bekannt ist:

```
mysql> SELECT gattung, geschlecht, COUNT(*) FROM haustier
    -> WHERE geschlecht IS NOT NULL
    -> GROUP BY gattung, geschlecht;
+----------+------------+----------+
| gattung  | geschlecht | COUNT(*) |
+----------+------------+----------+
| Vogel    | w          |        1 |
| Katze    | w          |        1 |
| Katze    | m          |        1 |
| Hund     | w          |        1 |
| Hund     | m          |        2 |
| Hamster  | w          |        1 |
| Schlange | m          |        1 |
+----------+------------+----------+
```

3.3.4.9 Mehr als eine Tabelle benutzen

Die Tabelle `haustier` enthält Daten zu den Tieren, die Sie besitzen. Wollen Sie andere Informationen zu diesen aufzeichnen – z. B. Ereignisse wie Tierarztbesuche oder die Ankunft von Nachwuchs –, dann benötigen Sie eine zweite Tabelle. Wie sollte diese Tabelle aussehen? Folgende Angaben müssen enthalten sein:

- Der Name des Tieres, damit Sie wissen, welches Ereignis zu welchem Tier gehört.
- Das Ereignisdatum.
- Ein Feld, welches das Ereignis beschreibt.
- Ein Ereignistypfeld, das Sie zum Kategorisieren der Ereignisse benötigen.

Unter Berücksichtigung dieser Aspekte könnte die `CREATE TABLE`-Anweisung für die Tabelle `ereignis` wie folgt aussehen:

```
mysql> CREATE TABLE ereignis (name VARCHAR(20), datum DATE,
    -> typ VARCHAR(15), bemerkung VARCHAR(255));
```

Wie bei der Tabelle `haustier` ist es auch hier am einfachsten, die bereits bekannten Daten über eine tabulatorgetrennte Textdatei in die Tabelle einzuladen. Diese Datei soll die folgenden Informationen enthalten:

name	datum	typ	bemerkung
Fluffy	1995-05-15	Nachwuchs	4 Kätzchen, 3 weiblich, 1 männlich
Buffy	1993-06-23	Nachwuchs	5 Welpen, 2 weiblich, 3 männlich
Buffy	1994-06-19	Nachwuchs	3 Welpen, 3 weiblich
Chirpy	1999-03-21	Tierarzt	Schnabel begradigt
Slim	1997-08-03	Tierarzt	gebrochene Rippe
Bowser	1991-10-12	Hundepension	
Fang	1991-10-12	Hundepension	
Fang	1998-08-28	Geburtstag	neues Kauspielzeug
Claws	1998-03-17	Geburtstag	neues Flohhalsband
Whistler	1998-12-09	Geburtstag	erster Geburtstag

Laden Sie die Datensätze wie folgt:

```
mysql> LOAD DATA LOCAL INFILE 'ereignis.txt' INTO TABLE ereignis;
```

Mit dem, was Sie bereits aus den Abfragen erlernt haben, die Sie zur Tabelle `haustier` abgesetzt haben, sollten Sie auch Daten aus der Tabelle `ereignis` abrufen können; die Prinzipien sind dieselben. Was aber, wenn Sie die Fragen, die Sie stellen, mit der Tabelle `ereignis` allein nicht beantworten können?

Nehmen wir an, Sie wollen herausfinden, in welchem Alter die jeweiligen Tiere Nachwuchs bekommen haben. Wir haben bereits weiter oben gesehen, wie Sie das Alter auf der Grundlage zweiter Datumsangaben berechnen. Das Datum, an dem die Mutter ihren Nachwuchs auf die Welt gebracht hat, ist in der Tabelle `ereignis` gespeichert, aber um das Alter der Mutter am Tag der Niederkunft zu ermitteln, benötigen Sie das Geburtsdatum, welches wiederum in der Tabelle `haustier` abgelegt ist. Das bedeutet, dass wir zwei Tabellen benötigen:

```
mysql> SELECT haustier.name,
    -> (YEAR(datum)-YEAR(geburtstag)) - (RIGHT(datum,5)<RIGHT(geburtstag,5)) AS
alter,
    -> bemerkung
    -> FROM haustier, ereignis
    -> WHERE haustier.name = ereignis.name AND ereignis.typ = 'Nachwuchs';
+--------+-------+-----------------------------------+
| name   | alter | bemerkung                         |
+--------+-------+-----------------------------------+
| Fluffy |     2 | 4 Kätzchen, 3 weiblich, 1 männlich  |
| Buffy  |     4 | 5 Welpen, 2 weiblich, 3 männlich    |
| Buffy  |     5 | 3 Welpen, 3 weiblich                |
+--------+-------+-----------------------------------+
```

Bei dieser Abfrage fällt Verschiedenes auf:

- Die `FROM`-Klausel listet zwei Tabellen auf, da die Abfrage Informationen aus beiden Tabellen abrufen muss.
- Wenn Sie Daten aus mehreren Tabellen kombinieren (verknüpfen), dann müssen Sie angeben, wie die Datensätze der einen Tabelle denen der anderen Tabelle zugeordnet werden können. Dies ist recht einfach, da beide Tabellen eine Spalte `name` haben. Die Abfrage verwendet eine `WHERE`-Klausel zur Zuordnung von Datensätzen der beiden Tabellen basierend auf den Werten in der Spalte `name`.

- Da die Spalte `name` in beiden Tabellen auftritt, müssen Sie bei Bezugnahme auf diese Spalte angeben, welche der beiden Tabellen Sie meinen. Dies erfolgt durch Voranstellung des Tabellennamens vor den Spaltennamen.

Um eine solche Verknüpfung – einen Join – durchzuführen, brauchen Sie keine zwei Tabellen. Manchmal ist es nützlich, eine Tabelle mit sich selbst zu verknüpfen, wenn man etwa Datensätze in einer Tabelle mit anderen Datensätzen in derselben Tabelle vergleichen will. Um beispielsweise Zuchtmöglichkeiten unter Ihren Tieren auszuloten, kann eine Verknüpfung der Tabelle `haustier` mit sich selbst praktisch sein, um passende Männchen und Weibchen einer Tierart zu ermitteln:

```
mysql> SELECT p1.name, p1.geschlecht, p2.name, p2.geschlecht, p1.gattung
    -> FROM haustier AS p1, haustier AS p2
    -> WHERE p1.gattung = p2.gattung AND p1.geschlecht = 'w' AND p2.geschlecht = 'm';
+--------+------------+--------+------------+---------+
| name   | geschlecht | name   | geschlecht | gattung |
+--------+------------+--------+------------+---------+
| Fluffy | w          | Claws  | m          | Katze   |
| Buffy  | w          | Fang   | m          | Hund    |
| Buffy  | w          | Bowser | m          | Hund    |
+--------+------------+--------+------------+---------+
```

In dieser Abfrage geben Sie Aliase für den Tabellennamen an, um auf die Spalten Bezug zu nehmen und klar zu machen, welcher Tabelleninstanz die Bezugnahme auf eine Spalte zuzuordnen ist.

3.4 Informationen über Datenbanken und Tabellen

Was können Sie tun, wenn Sie den Namen einer Datenbank oder Tabelle vergessen haben oder nicht mehr wissen, wie die Struktur einer gegebenen Tabelle aussieht (d. h., wie beispielsweise die Spalten heißen)? MySQL bietet zur Lösung dieses Problems eine Reihe von Anweisungen an, die Informationen zu unterstützten Datenbanken und Tabellen vermitteln.

Die Anweisung `SHOW DATABASES`, die die vom Server verwalteten Datenbanken auflistet, haben Sie bereits gesehen. Um herauszufinden, welche Datenbank gerade gewählt ist, verwenden Sie die Funktion `DATABASE()`:

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| menagerie  |
+------------+
```

Haben Sie noch keine Datenbank ausgewählt, dann lautet das Ergebnis `NULL`.

Um herauszufinden, welche Tabellen die Standarddatenbank enthält (wenn Sie etwa den Namen einer Tabelle nicht mehr genau wissen), verwenden Sie diesen Befehl:

```
mysql> SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| ereignis            |
| haustier            |
+---------------------+
```

Wollen Sie die Struktur einer Tabelle ermitteln, dann ist der Befehl `DESCRIBE` praktisch, denn er zeigt Informationen zu allen Spalten einer Tabelle an:

```
mysql> DESCRIBE haustier;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| name       | varchar(20) | YES  |     | NULL    |       |
| besitzer   | varchar(20) | YES  |     | NULL    |       |
| gattung    | varchar(20) | YES  |     | NULL    |       |
| geschlecht | char(1)     | YES  |     | NULL    |       |
| geburtstag | date        | YES  |     | NULL    |       |
| todestag   | date        | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

`Field` ist dabei der Spaltenname, `Type` der Datentyp der Spalte, `NULL` zeigt an, ob die Spalte `NULL`-Werte enthalten kann, `Key`, ob die Spalte indiziert ist, und `Default` bezeichnet den Standardwert der Spalte.

Wenn Sie Indizes in einer Tabelle verwenden, generiert `SHOW INDEX FROM` *`tbl_name`* Informationen zu diesen.

3.5 mysql im Stapelbetrieb

In obigen Abschnitten haben Sie `mysql` interaktiv zur Eingabe von Abfragen und zur Anzeige der Ergebnisse verwendet. Sie können `mysql` aber auch im Stapelbetrieb (Batch-Modus) ausführen. Zu diesem Zweck legen Sie die gewünschten Befehle in einer Datei ab und weisen `mysql` dann an, diese Datei auszulesen und die Befehle zu verarbeiten:

```
shell> mysql < batch-file
```

Wenn Sie `mysql` unter Windows ausführen und gewisse Sonderzeichen in der Datei Probleme verursachen, dann können Sie Folgendes tun:

```
C:\> mysql -e "source batch-file"
```

Müssen Sie die Verbindungsparameter auf der Befehlszeile angeben, dann könnte der Befehl so aussehen:

```
shell> mysql -h host -u user -p < batch-file
Enter password: ********
```

Wenn Sie `mysql` auf diese Weise verwenden, erstellen Sie eine Skriptdatei und führen das Skript dann aus.

Soll das Skript auch dann vollständig abgearbeitet werden, wenn enthaltene Anweisungen Fehler erzeugen, dann sollten Sie die Befehlszeilenoption `--force` verwenden.

Warum sollte man Skripten verwenden? Dafür gibt es ein paar gute Gründe:

- Führen Sie eine Abfrage häufiger aus (beispielsweise einmal in der Woche), dann können Sie ein Skript erstellen und sich so die Tipparbeit sparen, die bei jeder Neueingabe der Abfrage anfallen würde.

- Sie können neue Abfragen aus bereits vorhandenen ähnlichen erstellen, indem Sie Skriptdateien kopieren und bearbeiten.
- Der Stapelbetrieb kann auch nützlich sein, wenn Sie eine Abfrage entwickeln; dies gilt insbesondere für mehrzeilige Befehle oder aus mehreren Anweisungen bestehende Befehlsabfolgen. Wenn Sie einen Fehler machen, brauchen Sie nicht alles neu einzugeben. Beheben Sie den Fehler einfach in Ihrem Skript und weisen Sie `mysql` dann an, es erneut auszuführen.
- Wenn Ihre Abfrage eine umfangreiche Ausgabe erzeugt, dann können Sie diese seitenweise anzeigen lassen, statt sie am oberen Bildschirmrand verschwinden zu sehen:

  ```
  shell> mysql < batch-file | more
  ```

- Zur weiteren Verarbeitung können Sie die Ausgabe auch in eine Datei leiten:

  ```
  shell> mysql < batch-file > mysql.out
  ```

- Sie können Ihr Skript auch an andere Benutzer weitergeben, damit diese ebenfalls die entsprechenden Befehle ausführen können.
- Manche Situationen erlauben keine interaktive Vorgehensweise. Dies ist etwa der Fall, wenn eine Abfrage aus einem `cron`-Job heraus erfolgt. In diesem Fall müssen Sie den Stapelbetrieb verwenden.

Wenn Sie `mysql` im Stapelbetrieb verwenden, ist das Standardausgabeformat anders (genauer gesagt, knapper) als im interaktiven Modus. Die Ausgabe von `SELECT DISTINCT gattung FROM haustier` etwa sieht wie folgt aus, wenn `mysql` interaktiv läuft:

```
+----------+
| gattung  |
+----------+
| Vogel    |
| Katze    |
| Hund     |
| Hamster  |
| Schlange |
+----------+
```

Im Stapelbetrieb hingegen sieht die Ausgabe so aus:

```
gattung
Vogel
Katze
Hund
Hamster
Schlange
```

Wenn Sie das interaktive Ausgabeformat auch im Stapelbetrieb wünschen, verwenden Sie zum Aufruf `mysql -t`. Damit die Ausgabe der ausgeführten Befehle angezeigt wird, verwenden Sie `mysql -vvv`.

Sie können auch Skripten an der `mysql`-Eingabeaufforderung verwenden. Hierzu dient der Befehl `source` bzw. `\.`:

```
mysql> source filename;
mysql> \. filename
```

3.6 Beispiele gebräuchlicher Abfragen

Es folgen einige Beispiele dafür, wie man häufige Probleme mit MySQL löst.

Einige dieser Beispiele verwenden die Tabelle `shop`, in der die Preise jedes Artikels (bzw. Artikelnummer) für bestimmte Händler aufgeführt sind. Wenn man davon ausgeht, dass jeder Händler genau einen bestimmten Festpreis für einen Artikel hat, dann ist (`artikel`, `haendler`) ein Primärschlüssel für die Datensätze.

Starten Sie das Befehlszeilen-Tool `mysql` und wählen Sie eine Datenbank aus:

```
shell> mysql your-database-name
```

(Bei den meisten MySQL-Installationen können Sie die Datenbank `test` verwenden.)

Erstellung der Beispieltabelle und Ausfüllen mit Daten erfolgen mit den folgenden Anweisungen:

```
mysql> CREATE TABLE shop (
    -> artikel INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
    -> haendler  CHAR(20)                 DEFAULT ''     NOT NULL,
    -> preis   DOUBLE(16,2)             DEFAULT '0.00' NOT NULL,
    -> PRIMARY KEY(artikel, haendler));
mysql> INSERT INTO shop VALUES
    -> (1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),
    -> (3,'C',1.69),(3,'D',1.25),(4,'D',19.95);
```

Nachdem Sie die Anweisungen abgesetzt haben, weist die Tabelle den folgenden Inhalt auf:

```
mysql> SELECT * FROM shop;
+---------+----------+-------+
| artikel | haendler | preis |
+---------+----------+-------+
|    0001 | A        |  3.45 |
|    0001 | B        |  3.99 |
|    0002 | A        | 10.99 |
|    0003 | B        |  1.45 |
|    0003 | C        |  1.69 |
|    0003 | D        |  1.25 |
|    0004 | D        | 19.95 |
+---------+----------+-------+
```

3.6.1 Der höchste Wert einer Spalte

„Welches ist die höchste Artikelnummer?“

```
SELECT MAX(artikel) AS artikel FROM shop;

+---------+
| artikel |
+---------+
|       4 |
+---------+
```

3.6.2 Die Zeile, die den höchsten Wert einer bestimmten Spalte enthält

Aufgabe: Ermitteln Sie Nummer, Händler und Preis des teuersten Artikels.

Dies lässt sich recht einfach mit einer Unterabfrage bewerkstelligen:

```
SELECT artikel, haendler, preis
FROM   shop
WHERE  preis=(SELECT MAX(preis) FROM shop);
```

Eine andere Lösung besteht darin, alle Datensätze nach Preis aufsteigend zu sortieren und nur den ersten Datensatz mithilfe der MySQL-spezifischen `LIMIT`-Klausel abzurufen:

```
SELECT artikel, haendler, preis
FROM shop
ORDER BY preis DESC
LIMIT 1;
```

Hinweis: Wenn mehrere teuerste Artikel zum gleichen Preis vorhanden sind, dann zeigt die `LIMIT`-Lösung nur einen davon an.

3.6.3 Höchster Wert einer Spalte pro Gruppe

Aufgabe: Ermitteln Sie den höchsten Preis je Artikel.

```
SELECT artikel, MAX(preis) AS preis
FROM   shop
GROUP BY artikel

+---------+-------+
| artikel | preis |
+---------+-------+
|    0001 |  3.99 |
|    0002 | 10.99 |
|    0003 |  1.69 |
|    0004 | 19.95 |
+---------+-------+
```

3.6.4 Die Zeilen, die das gruppenweise Maximum eines bestimmten Felds enthalten

Aufgabe: Ermitteln Sie für jeden Artikel den oder die Händler mit dem höchsten Preis.

Dieses Problem lässt sich mit einer Unterabfrage wie der folgenden lösen:

```
SELECT artikel, haendler, preis
FROM   shop s1
WHERE  preis=(SELECT MAX(s2.preis)
              FROM shop s2
              WHERE s1.artikel = s2.artikel);
```

3.6.5 Wie Benutzervariablen verwendet werden

Sie können MySQL-Benutzervariablen verwenden, um die Ergebnisse zu vermerken, ohne sie in Temporärvariablen auf dem Client speichern zu müssen. (Siehe auch Abschnitt 9.3 auf Seite 653.)

Um beispielsweise die Artikel mit dem höchsten und dem niedrigsten Preis zu ermitteln, können Sie Folgendes tun:

```
mysql> SELECT @min_preis:=MIN(preis),@max_preis:=MAX(preis) FROM shop;
mysql> SELECT * FROM shop WHERE preis=@min_preis OR preis=@max_preis;
+---------+----------+-------+
| artikel | haendler | preis |
+---------+----------+-------+
|    0003 | D        |  1.25 |
|    0004 | D        | 19.95 |
+---------+----------+-------+
```

3.6.6 Wie Fremdschlüssel verwendet werden

Bei MySQL unterstützen `InnoDB`-Tabellen die Überprüfung von Fremdschlüsselbeschränkungen. Siehe auch Abschnitt 14.2 auf Seite 984 und Abschnitt 1.9.5.5 auf Seite 52.

Eine Fremdschlüsselbeschränkung ist nicht erforderlich, wenn Sie lediglich zwei Tabellen miteinander verknüpfen wollen. Bei anderen Speicher-Engines als `InnoDB` ist dies möglich, indem eine Spalte zur Verwendung einer `REFERENCES` *`tbl_name`*`(`*`col_name`*`)`-Klausel definiert wird; diese hat keine eigentliche Wirkung und *dient Ihnen lediglich als Erinnerung oder Anmerkung, damit Sie daran denken, dass die derzeit definierte Spalte auf eine Spalte in einer anderen Tabelle verweisen soll*. Es ist bei Verwendung dieser Syntax extrem wichtig, sich zu vergegenwärtigen, dass

- MySQL keinerlei Überprüfung durchführt, um sicherzustellen, dass *`col_name`* tatsächlich in *`tbl_name`* vorhanden ist (bzw. dass *`tbl_name`* selbst existiert),
- MySQL keine Aktionen an *`tbl_name`* – wie etwa das Löschen von Datensätzen – als Reaktion auf Aktionen ausführt, die an Datensätzen in der von Ihnen gerade definierten Tabelle vorgenommen werden (mit anderen Worten, diese Syntax enthält keine Funktionalität wie etwa `ON DELETE` oder `ON UPDATE`; Sie können zwar eine `ON DELETE`- oder `ON UPDATE`-Klausel als Bestandteil der `REFERENCES`-Klausel schreiben, aber diese wird ebenfalls ignoriert),
- diese Syntax eine *Spalte* erstellt (und *keinerlei* Index oder Schlüssel),
- diese Syntax einen Fehler erzeugt, wenn damit eine `InnoDB`-Tabelle zu erzeugen versucht wird.

Sie können eine Spalte, die als Join-Spalte erstellt wurde, wie nachfolgend gezeigt verwenden:

```
CREATE TABLE person (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name CHAR(60) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE shirt (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    style ENUM('t-shirt', 'polo', 'dress') NOT NULL,
    color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,
    owner SMALLINT UNSIGNED NOT NULL REFERENCES person(id),
    PRIMARY KEY (id)
);
```

```
INSERT INTO person VALUES (NULL, 'Antonio Paz');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', @last),
(NULL, 'dress', 'white', @last),
(NULL, 't-shirt', 'blue', @last);

INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'dress', 'orange', @last),
(NULL, 'polo', 'red', @last),
(NULL, 'dress', 'blue', @last),
(NULL, 't-shirt', 'white', @last);

SELECT * FROM person;
+----+--------------------+
| id | name               |
+----+--------------------+
|  1 | Antonio Paz        |
|  2 | Lilliana Angelovska |
+----+--------------------+

SELECT * FROM shirt;
+----+---------+--------+-------+
| id | style   | color  | owner |
+----+---------+--------+-------+
|  1 | polo    | blue   |     1 |
|  2 | dress   | white  |     1 |
|  3 | t-shirt | blue   |     1 |
|  4 | dress   | orange |     2 |
|  5 | polo    | red    |     2 |
|  6 | dress   | blue   |     2 |
|  7 | t-shirt | white  |     2 |
+----+---------+--------+-------+


SELECT s.* FROM person p, shirt s
 WHERE p.name LIKE 'Lilliana%'
   AND s.owner = p.id
   AND s.color <> 'white';

+----+-------+--------+-------+
| id | style | color  | owner |
+----+-------+--------+-------+
|  4 | dress | orange |     2 |
|  5 | polo  | red    |     2 |
|  6 | dress | blue   |     2 |
+----+-------+--------+-------+
```

Wenn sie auf diese Weise eingesetzt wird, wird die REFERENCES-Klausel in der Ausgabe von SHOW CREATE TABLE oder DESCRIBE nicht angezeigt:

```
SHOW CREATE TABLE shirt\G
*************************** 1. row ***************************
Table: shirt
Create Table: CREATE TABLE `shirt` (
`id` smallint(5) unsigned NOT NULL auto_increment,
`style` enum('t-shirt','polo','dress') NOT NULL,
`color` enum('red','blue','orange','white','black') NOT NULL,
`owner` smallint(5) unsigned NOT NULL,
PRIMARY KEY  (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

Die Verwendung von REFERENCES als derartigen Kommentar oder „Erinnerung“ in einer Spaltendefinition funktioniert sowohl mit MyISAM- als auch mit BerkeleyDB-Tabellen.

3.6.7 Über zwei Schlüssel suchen

Eine Suche mit OR über einen Schlüssel ist in MySQL ebenso optimal gelöst wie die Verwendung von AND.

Der einzige etwas heikle Fall ist die Suche über zwei verschiedene Schlüssel, die durch OR kombiniert sind:

```
SELECT field1_index, field2_index FROM test_table
WHERE field1_index = '1' OR  field2_index = '1'
```

Dieser Fall ist optimiert. Siehe auch Abschnitt 7.2.6 auf Seite 514.

Sie können das Problem auch effizient mithilfe einer Union lösen, die die Ausgabe zweier getrennter SELECT-Anweisungen zusammenfasst. Siehe auch Abschnitt 13.2.7.2 auf Seite 879.

Jede SELECT-Anweisung sucht nur über einen Schlüssel und kann optimiert werden:

```
SELECT field1_index, field2_index
    FROM test_table WHERE field1_index = '1'
UNION
SELECT field1_index, field2_index
    FROM test_table WHERE field2_index = '1';
```

3.6.8 Besuche pro Tag berechnen

Das folgende Beispiel zeigt, wie Sie die Bitgruppenfunktionen zur Berechnung der Anzahl von Tagen im Monat verwenden können, an denen ein Benutzer eine bestimmte Webseite besucht hat.

```
CREATE TABLE t1 (year YEAR(4), month INT(2) UNSIGNED ZEROFILL,
             day INT(2) UNSIGNED ZEROFILL);
INSERT INTO t1 VALUES(2000,1,1),(2000,1,20),(2000,1,30),(2000,2,2),
            (2000,2,23),(2000,2,23);
```

Die Beispieltabelle enthält Jahr/Monat/Tag-Werte, die die Besuche von Benutzern auf der Seite repräsentieren. Um zu ermitteln, an wie vielen verschiedenen Tagen in den einzelnen Monaten diese Besuche stattfanden, verwenden Sie folgende Abfrage:

```
SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1
       GROUP BY year,month;
```

Das Ergebnis sieht wie folgt aus:

```
+------+-------+------+
| year | month | days |
+------+-------+------+
| 2000 |    01 |    3 |
| 2000 |    02 |    2 |
+------+-------+------+
```

Die Abfrage berechnet, wie viele verschiedene Tage in der Tabelle für jede Jahr/Monat/Tag-Kombination erscheinen, und entfernt doppelte Einträge dabei automatisch.

3.6.9 Verwendung von AUTO_INCREMENT

Das Attribut `AUTO_INCREMENT` kann zur Erzeugung einer eindeutigen Kennung für neue Datensätze verwendet werden:

```
CREATE TABLE animals (
     id MEDIUMINT NOT NULL AUTO_INCREMENT,
     name CHAR(30) NOT NULL,
     PRIMARY KEY (id)
 );

INSERT INTO animals (name) VALUES
    ('dog'),('cat'),('penguin'),
    ('lax'),('whale'),('ostrich');

SELECT * FROM animals;
```

Das Ergebnis sieht wie folgt aus:

```
+----+---------+
| id | name    |
+----+---------+
|  1 | dog     |
|  2 | cat     |
|  3 | penguin |
|  4 | lax     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
```

Sie können den aktuellen `AUTO_INCREMENT`-Wert mit der SQL-Funktion `LAST_INSERT_ID()` oder der C-API-Funktion `mysql_insert_id()` abrufen. Diese Funktionen sind verbindungsspezifisch, d. h., ihre Rückgabewerte werden nicht durch eine andere Verbindung beeinträchtigt, die ebenfalls Einfügeoperationen durchführt.

Hinweis: Bei mehrzeiligen Einfügeoperationen geben `LAST_INSERT_ID()` und `mysql_insert_id()` den `AUTO_INCREMENT`-Schlüssel des *ersten* eingefügten Datensatzes zurück. Auf diese Weise lassen sich mehrzeilige Einfügeoperationen auch von anderen Servern in einer Replikationskonfiguration korrekt nachvollziehen.

Bei MyISAM- und BDB-Tabellen können Sie AUTO_INCREMENT in einer Sekundärspalte eines mehrspaltigen Indexes angeben. In diesem Fall wird der erzeugte Wert der AUTO_INCREMENT-Spalte als MAX(*auto_increment_column*) + 1 WHERE prefix=*given-prefix* berechnet. Dies ist praktisch, wenn Sie Daten in sortierte Gruppen eingeben wollen.

```
CREATE TABLE animals (
    grp ENUM('fish','mammal','bird') NOT NULL,
    id MEDIUMINT NOT NULL AUTO_INCREMENT,
    name CHAR(30) NOT NULL,
    PRIMARY KEY (grp,id)
);

INSERT INTO animals (grp,name) VALUES
    ('mammal','dog'),('mammal','cat'),
    ('Vogel','penguin'),('fish','lax'),('mammal','whale'),
    ('Vogel','ostrich');

SELECT * FROM animals ORDER BY grp,id;
```

Das Ergebnis sieht wie folgt aus:

```
+--------+----+---------+
| grp    | id | name    |
+--------+----+---------+
| fish   |  1 | lax     |
| mammal |  1 | dog     |
| mammal |  2 | cat     |
| mammal |  3 | whale   |
| bird   |  1 | penguin |
| bird   |  2 | ostrich |
+--------+----+---------+
```

Beachten Sie, dass in diesem Fall (wenn die AUTO_INCREMENT-Spalte Teil eines mehrspaltigen Index ist) AUTO_INCREMENT-Werte neu verwendet werden, wenn Sie den Datensatz mit dem höchsten AUTO_INCREMENT-Wert in einer beliebigen Gruppe löschen. Dies gilt auch für MyISAM-Tabellen, bei denen AUTO_INCREMENT-Werte normalerweise nicht wiederverwendet werden.

Wenn die AUTO_INCREMENT-Spalte Bestandteil mehrerer Indizes ist, erzeugt MySQL Folgewerte auf der Basis des Indexes, der mit der AUTO_INCREMENT-Spalte beginnt (sofern er vorhanden ist). Wenn also die Tabelle animals die Indizes PRIMARY KEY (grp, id) und INDEX (id) enthielte, würde MySQL den Index PRIMARY KEY bei der Erstellung von Folgewerten ignorieren. Das Ergebnis wäre eine Tabelle, die eine einzelne Folge (und nicht eine Folge pro grp-Wert) enthalten würde.

Wenn Sie bei einem anderen AUTO_INCREMENT-Wert als 1 beginnen wollen, können Sie diesen Wert mit CREATE TABLE oder ALTER TABLE wie folgt festlegen:

```
mysql> ALTER TABLE tbl AUTO_INCREMENT = 100;
```

Weitere Informationen zu AUTO_INCREMENT finden Sie hier:

- Wie Sie das AUTO_INCREMENT-Attribut einer Spalte zuweisen: Abschnitt 13.1.5 auf Seite 825 und Abschnitt 13.1.2 auf Seite 815.
- Wie sich AUTO_INCREMENT je nach SQL-Modus verhält: Abschnitt 5.2.5 auf Seite 312.
- Wie man den Datensatz mit dem aktuellsten AUTO_INCREMENT-Wert findet: Abschnitt 12.1.3 auf Seite 727.

- Wie man den zu verwendenden AUTO_INCREMENT-Wert einstellt: Abschnitt 13.5.3 auf Seite 924.
- AUTO_INCREMENT und Replikation: Abschnitt 6.8 auf Seite 462.
- Wie die auf AUTO_INCREMENT bezogenen Serversystemvariablen (auto_increment_increment und auto_increment_offset) zur Replikation verwendet werden können: Abschnitt 5.2.2 auf Seite 265.

3.7 Anfragen aus dem Zwillingsprojekt

Bei Analytikerna und Lentus haben wir die System- und Feldarbeiten für ein umfangreiches Forschungsprojekt durchgeführt. Dieses Projekt ist eine Zusammenarbeit zwischen dem Institut für Umweltmedizin am Karolinska Institutet in Stockholm und der Abteilung für klinische Forschung bei Altersprozessen und Psychologie an der University of Southern California.

Das Projekt umfasste u. a. ein Screening, bei dem alle in Schweden lebenden Zwillinge, die über 65 Jahre alt waren, telefonisch befragt wurden. Zwillinge, die bestimmte Kriterien erfüllten, wurden dann in eine zweite Untersuchungsphase überführt. In dieser zweiten Phase konnten sich die teilnehmenden Zwillinge auf freiwilliger Basis einer ärztlichen Untersuchung unterziehen. Zu dieser Untersuchung gehörten körperliche und neuropsychologische Tests, Labortests, Neuro-Imaging, eine Bewertung des psychologischen Zustandes und eine Erfassung der Familiengeschichte. Ferner wurden Daten zu medizinischen und umweltbedingten Risikofaktoren ermittelt.

Weitere Informationen zum Zwillingsprojekt finden Sie unter folgender Adresse:
http://www.mep.ki.se/twinreg/index_en.html

Im Verlauf des Projekts kam zu Verwaltungszwecken eine Weboberfläche zum Einsatz, die in Perl und MySQL geschrieben war.

Am Ende jedes Tages wurden die Daten der Befragungen in eine MySQL-Datenbank eingetragen.

3.7.1 Alle nicht verteilten Zwillinge finden

Mit der folgenden Abfrage wurde ermittelt, wer für die zweite Phase des Projekts geeignet war:

```
SELECT
    CONCAT(p1.id, p1.tvab) + 0 AS tvid,
    CONCAT(p1.christian_name, ' ', p1.surname) AS Name,
    p1.postal_code AS Code,
    p1.city AS City,
    pg.abrev AS Area,
    IF(td.participation = 'Aborted', 'A', ' ') AS A,
    p1.dead AS dead1,
    l.ereignis AS event1,
    td.suspect AS tsuspect1,
    id.suspect AS isuspect1,
    td.severe AS tsevere1,
    id.severe AS isevere1,
    p2.dead AS dead2,
    l2.ereignis AS event2,
    h2.nurse AS nurse2,
    h2.doctor AS doctor2,
```

```
    td2.suspect AS tsuspect2,
    id2.suspect AS isuspect2,
    td2.severe AS tsevere2,
    id2.severe AS isevere2,
    l.finish_datum
FROM
    twin_project AS tp
    /* For Twin 1 */
    LEFT JOIN twin_data AS td ON tp.id = td.id
              AND tp.tvab = td.tvab
    LEFT JOIN informant_data AS id ON tp.id = id.id
              AND tp.tvab = id.tvab
    LEFT JOIN harmony AS h ON tp.id = h.id
              AND tp.tvab = h.tvab
    LEFT JOIN lentus AS l ON tp.id = l.id
              AND tp.tvab = l.tvab
    /* For Twin 2 */
    LEFT JOIN twin_data AS td2 ON p2.id = td2.id
              AND p2.tvab = td2.tvab
    LEFT JOIN informant_data AS id2 ON p2.id = id2.id
              AND p2.tvab = id2.tvab
    LEFT JOIN harmony AS h2 ON p2.id = h2.id
              AND p2.tvab = h2.tvab
    LEFT JOIN lentus AS l2 ON p2.id = l2.id
              AND p2.tvab = l2.tvab,
    person_data AS p1,
    person_data AS p2,
    postal_groups AS pg
WHERE
    /* p1 gets main twin and p2 gets his/her twin. */
    /* ptvab is a field inverted from tvab */
    p1.id = tp.id AND p1.tvab = tp.tvab AND
    p2.id = p1.id AND p2.ptvab = p1.tvab AND
    /* Just the screening survey */
    tp.survey_no = 5 AND
    /* Skip if partner died before 65 but allow emigration (dead=9) */
    (p2.dead = 0 OR p2.dead = 9 OR
     (p2.dead = 1 AND
      (p2.todestag_datum = 0 OR
       (((TO_DAYS(p2.todestag_datum) - TO_DAYS(p2.Geburtstag)) / 365)
        >= 65))))
    AND
    (
    /* Twin is suspect */
    (td.future_contact = 'Yes' AND td.suspect = 2) OR
    /* Twin is suspect - Informant is Blessed */
    (td.future_contact = 'Yes' AND td.suspect = 1
                               AND id.suspect = 1) OR
    /* No twin - Informant is Blessed */
    (ISNULL(td.suspect) AND id.suspect = 1
                        AND id.future_contact = 'Yes') OR
    /* Twin broken off - Informant is Blessed */
    (td.participation = 'Aborted'
     AND id.suspect = 1 AND id.future_contact = 'Yes') OR
    /* Twin broken off - No inform - Have partner */
    (td.participation = 'Aborted' AND ISNULL(id.suspect)
                                  AND p2.dead = 0))
```

```
    AND
    l.ereignis = 'Finished'
    /* Get at area code */
    AND SUBSTRING(p1.postal_code, 1, 2) = pg.code
    /* Not already distributed */
    AND (h.nurse IS NULL OR h.nurse=00 OR h.doctor=00)
    /* Has not refused or been aborted */
    AND NOT (h.status = 'Refused' OR h.status = 'Aborted'
    OR h.status = 'Died' OR h.status = 'Other')
ORDER BY
    tvid;
```

Hierzu ein paar Anmerkungen:

- `CONCAT(p1.id, p1.tvab) + 0 AS tvid`

  Die verketteten Werte `id` und `tvab` sollen in numerischer Reihenfolge sortiert werden. Durch Hinzufügen von `0` zum Ergebnis wird MySQL dazu veranlasst, dieses Ergebnis als Zahl zu betrachten.

- Spalte `id`

  Bezeichnet ein Zwillingspaar. Die Spalte wird als Schlüssel in allen Tabellen verwendet.

- Spalte `tvab`

  Bezeichnet einen Zwilling in einem Paar. Der Wert ist entweder `1` oder `2`.

- Spalte `ptvab`

  Dies ist das Gegenstück zu `tvab`. Wenn `tvab` `1` ist, dann ist dieser Wert `2` und umgekehrt. Zweck ist die Einsparung von Tipparbeit und eine vereinfachte Optimierung der Abfrage durch MySQL.

Diese Abfrage veranschaulicht unter anderem, wie man Daten in einer Tabelle mithilfe eines Joins von derselben Tabelle aus durchführt (`p1` und `p2`). In diesem Beispiel wird diese Methode verwendet, um zu ermitteln, ob einer von zwei Zwillingen eines Paares vor dem 65. Lebensjahr starb. Ist dies der Fall, dann wird der Datensatz nicht zurückgegeben.

Alle genannten Elemente sind in allen Tabellen mit Informationen zu den Zwillingen vorhanden. Um Abfragen zu beschleunigen, wurde ein Schlüssel sowohl auf `id,tvab` (alle Tabellen) als auch auf `id,ptvab` (`person_data`) erstellt.

Auf unserem Produktionssystem (UltraSPARC, 200 MHz) gibt diese Abfrage etwa 150 bis 200 Datensätze zurück und benötigt dafür weniger als eine Sekunde.

Die aktuelle Anzahl der Datensätze in den in dieser Abfrage verwendeten Tabellen sieht wie folgt aus:

Tabelle	Datensätze
`person_data`	71074
`lentus`	5291
`twin_project`	5286
`twin_data`	2012
`informant_data`	663
`harmony`	381
`postal_groups`	100

3.7.2 Eine Tabelle über den Zustand von Zwillingspaaren zeigen

Jede Befragung resultiert in einem Statuscode, der `ereignis` heißt. Die nachfolgend gezeigte Abfrage wird zur Anzeige einer Tabelle über alle Zwillingspaare mit zugeordnetem Statuscode verwendet. Sie zeigt an, bei wie vielen Paaren beide Zwillinge zugelassen wurden, bei wie vielen Paaren nur ein Zwilling zugelassen (und der andere abgelehnt) wurde usw.

```
SELECT
        t1.ereignis,
        t2.ereignis,
        COUNT(*)
FROM
        lentus AS t1,
        lentus AS t2,
        twin_project AS tp
WHERE
        /* We are looking at one pair at a time */
        t1.id = tp.id
        AND t1.tvab=tp.tvab
        AND t1.id = t2.id
        /* Just the screening survey */
        AND tp.survey_no = 5
        /* This makes each pair only appear once */
        AND t1.tvab='1' AND t2.tvab='2'
GROUP BY
        t1.ereignis, t2.ereignis;
```

3.8 MySQL mit Apache verwenden

Es gibt Programme, die Ihnen die Authentifizierung Ihrer Benutzer aus einer MySQL-Datenbank gestatten und gleichzeitig das Schreiben der Logdateien in eine MySQL-Tabelle ermöglichen.

Sie können das Apache-Logformat so ändern, dass es leichter von MySQL gelesen werden kann. Hierzu fügen Sie Folgendes in Ihre Apache-Konfigurationsdatei ein:

```
LogFormat \
        "\"%h\",%{%Y%m%d%H%M%S}t,%>s,\"%b\",\"%{Content-Type}o\",  \
        \"%U\",\"%{Referer}i\",\"%{User-Agent}i\""
```

Um eine Logdatei in diesem Format in MySQL einzuladen, können Sie etwa folgende Anweisung verwenden:

```
LOAD DATA INFILE '/local/access_log' INTO TABLE tbl_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

Die benannte Tabelle sollte so erstellt sein, dass sie Spalten aufweist, die denen entsprechen, welche die Zeile `LogFormat` in die Logdatei schreibt.