本章通过演示如何使用 mysql 客户程序创造和使用一个简单的数据库，提供一个 MySQL 的入门教程。mysql（有时称为“终端监视器”或只是“监视”）是一个交互式程序，允许你连接一个 MySQL 服务器，运行查询并察看结果。mysql 可以用于批模式：你预先把查询放在一个文件中，然后告诉 mysql 执行文件的内容。使用 mysql 的两个方法都在这里涉及。

为了看清由 mysql 提供的一个选择项目表了，用--help 选项调用它：

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

本章假定 mysql 已经被安装在你的机器上，并且有一个 MySQL 服务器你可以连接。如果这不是真的，联络你的 MySQL 管理员。（如果你是管理员，你将需要请教这本手册的其他章节。）

本章描述建立和使用一个数据库的全过程。如果你仅仅对存取一个已经存在数据库感兴趣，你可能想要跳过描述怎样创建数据库及它所包含的表的章节。

既然本章本质上是一个教程，许多细节有必要被省略。对于这里所涉及的话题的更多信息，咨询本手册的相关章节。

8.1 连接与断开服务者

为了连接服务器，当你调用 mysql 时，你通常将需要提供一个 MySQL 用户名和很可能，一个口令。如果服务器运行在不是你登录的一台机器上，你也将需要指定主机名。联系你的管理员以找出你应该使用什么连接参数进行连接(即，那个主机，用户名字和使用的口令)。一旦你知道正确的参数，你应该能象这样连接：

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

********代表你的口令；当 mysql 显示 Enter password:提示时输入它。

如果能工作，你应该看见 mysql>提示后的一些介绍信息：

```
shell> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 459 to server version: 3.22.20a-log

Type 'help' for help.

mysql>
```

提示符告诉你 mysql 准备为你输入命令。

一些 MySQL 安装允许用户以“anoymous”(匿名)用户连接在本地主机上运行的服务器。如果在你的机器是这种情况，你应该能通过没有任何选项地调用 mysql 与该服务器连接：

```
shell> mysql
```

在你成功地连接后，你可以在 mysql>提示下打入 QUIT 随时断开：

```
mysql> QUIT
Bye
```

你也可以键入 control-D 断开。

在下列章节的大多数例子都假设你连接到服务器。由 mysql>提示指明他们。

8.2 输入查询

确保你连接上了服务器，如在先前的章节讨论的。这样做本身将不选择任何数据库来工作，但是那很好。从这点讲，知道关于如何出询问的一点知识，比马上跳至创建表、给他们装载数据并且从他们检索数据要来的重要写。本节描述输入命令的基本原则，使用几个查询，你能尝试让自己 mysql 是如何工作的。

这是一个简单的命令，要求服务器告诉你它的版本号和当前日期。在 mysql>提示打入如下命令并按回车键：

```
mysql> SELECT VERSION(), CURRENT_DATE;
+--------------+--------------+
| version()    | CURRENT_DATE |
+--------------+--------------+
| 3.22.20a-log | 1999-03-19   |
+--------------+--------------+
1 row in set (0.01 sec)
mysql>
```

这询问说明关于 mysql 几件事:

- 一个命令通常由 SQL 语句组成，随后有一个分号。（有一些例外不需要一个分号。早先提到的 QUIT 是他们之一。我们将以后看到其它。）
- 当你发出一个命令时，mysql 发送它给服务器并显示结果，然后打出另外一个 mysql>显示它准备好接受另外的命令。
- mysql 以一张表格(行和列)显示查询输出。第一行包含列的标签，随后的行是询问结果。通常， 列标签是你取自数据库表的列的名字。如果你正在检索一个表达式而非表列的值(如刚才的例子)，mysql 用表达式本身标记列。
- mysql 显示多少行被返回，和查询花了多长执行，它给你提供服务器性能的一个大致概念。因为他们表示时钟时间(不是 CPU 或机器时间)，并且

因为他们受到诸如服务器负载和网络延时的影响，因此这些值是不精确的。（为了简洁，在本章剩下的例子中不再显示“集合中的行”。）

关键词可以以任何大小写字符被输入。下列询问是等价的：

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

这里有另外一个查询，它说明你能将 mysql 用作一个简单的计算器：

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+------------+---------+
| SIN(PI()/4) | (4+1)*5 |
+------------+---------+
|    0.707107 |      25 |
+------------+---------+
```

至今显示的命令是相当短的，单行语句。你甚至能在单行上输入多条语句，只是以一个分号结束每一条：

```
mysql> SELECT VERSION(); SELECT NOW();
+--------------+
| version()    |
+--------------+
| 3.22.20a-log |
+--------------+

+---------------------+
| NOW()               |
+---------------------+
| 1999-03-19 00:15:33 |
+---------------------+
```

一个命令不必全在一个单独行给出，所以需要多行的较长命令不是一个问题。mysql 通过寻找终止的分号而不是寻找输入行的结束来决定你的语句在哪儿结束。（换句话说，mysql 接受自由格式输入：它收集输入行但执行他们直到它看见分号。）

这里是一个简单的多行语句的例子：

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+--------------------+--------------+
| USER()             | CURRENT_DATE |
```

```
+-------------------+-------------+
| joesmith@localhost | 1999-03-18   |
+-------------------+-------------+
```

在这个例子中，在你输入一个多行查询的第一行后，要注意提示符如何从 mysql> 变为->，这正是 mysql 如何指出它没见到完整的语句并且正在等待剩余的部分。提示符是你的朋友，因为它提供有价值的反馈，如果你使用该反馈，你将总是知道 mysql 正在等待什么。

如果你决定，你不想要执行你在输入过程中输入的一个命令，打入\c 取消它：

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

这里也要注意提示符，在你打入\c 以后，它切换回到 mysql>，提供反馈以表明 mysql 准备接受一个新命令。

下表显示出你可以看见的各个提示符并总结他们意味着 mysql 在什么状态下：

提示符	意思
mysql>	准备好接受新命令
->	等待多行命令的下一行
'>	等待下一行，收集以单引号(“’”)开始的字符串
">	等待下一行，收集以双引号(“"”)开始的字符串

当你打算在一个单行上发出一个命令时，多行语句通常“偶然”出现，但是忘记终止的分号。在这种情况中，mysql 等待进一步输入：

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

如果这发生在你身上(你认为你输完了语句但是唯一的反应是一个->提示符)，很可能 mysql 正在等待分号。如果你没有注意到提示符正在告诉你什么，在认识到你需要做什么之前，你可能花一会儿时间呆坐在那儿。进入一个分号完成语句，并且 mysql 将执行它：

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

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

'>和">提示符出现在在字符串收集期间。在 MySQL 中，你可以写由“'”或“"”字符括起来的字符串（例如，'hello'或"goodbye"），并且 mysql 让你进入跨越多行的字符串。当你看到一个'>或">提示符时，这意味着你已经输入了包含以“'”或“"”括号字符开始的字符串的一行，但是还没有输入终止字符串的匹配引号。如果你确实正在输入一个多行字符串，很好，但是果真如此吗？不尽然。更常见的，'>和">提示符显示你粗心地省掉了一个引号字符。例如：

```
mysql> SELECT * FROM my_table WHERE name = "Smith AND age < 30;
    ">
```

如果你输入该 SELECT 语句，然后按回车键并等待结果，什么都没有出现。不要惊讶，“为什么该查询这么长呢？”，注意">提示符提供的线索。它告诉你 mysql 期望见到一个未终止字符串的余下部分。（你在语句中看见错误吗？字符串 "Smith 正好丢失第二个引号。）

走到这一步，你该做什么？最简单的是取消命令。然而，在这种情况下，你不能只是打入\c，因为 mysql 作为它正在收集的字符串的一部分来解释它！相反，输入关闭的引号字符(这样 mysql 知道你完成了字符串)，然后打入\c：

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

提示符回到 mysql>，显示 mysql 准备好接受一个新命令了。

知道'>和">提示符意味着什么是很重要的，因为如果你错误地输入一个未终止的字符串，任何比你下一步输入的行好象将要被 mysql 忽略--包括包含 QUIT 的行！这可能相当含糊，特别是在你能取消当前命令前，如果你不知道你需要提出终止引号。

8.3 常用查询的例子

下面是一些学习如何用 MySQL 解决一些常见问题的例子。

一些例子使用数据库表“shop”，包含某个商人的每篇文章(物品号)的价格。假定每个商人的每篇文章有一个单独的固定价格，那么(物品，商人)是记录的主键。

你能这样创建例子数据库表：

```
CREATE TABLE shop (
 article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
 dealer  CHAR(20)                 DEFAULT ''     NOT NULL,
 price   DOUBLE(16,2)             DEFAULT '0.00' NOT NULL,
```

```
 PRIMARY KEY(article, dealer));

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);
```

好了，例子数据是这样的：

```
SELECT * FROM shop

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    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 |
+---------+--------+-------+
```

8.3.1 列的最大值

“最大的物品号是什么？”

```
SELECT MAX(article) AS article FROM shop

+---------+
| article |
+---------+
|       4 |
+---------+
```

8.3.2 拥有某个列的最大值的行

“找出最贵的文章的编号、商人和价格”

在 ANSI-SQL 中这很容易用一个子查询做到：

```
SELECT article, dealer, price
FROM   shop
WHERE  price=(SELECT MAX(price) FROM shop)
```

在 MySQL 中（还没有子查询)就用 2 步做到：

1. 用一个 SELECT 语句从表中得到最大值。
2. 使用该值编出实际的查询：

```
SELECT article, dealer, price
FROM   shop
WHERE  price=19.95
```

另一个解决方案是按价格降序排序所有行并用MySQL特定LIMIT子句只得到的第一行：

```
SELECT article, dealer, price
FROM   shop
ORDER BY price DESC
LIMIT 1
```

注意:如果有多个最贵的文章(例如每个 19.95)，LIMIT 解决方案仅仅显示他们之一!

8.3.3 列的最大值：按组：只有值

“每篇文章的最高的价格是什么？”

```
SELECT article, MAX(price) AS price
FROM   shop
GROUP BY article
```

```
+---------+-------+
| article | price |
+---------+-------+
|    0001 |  3.99 |
|    0002 | 10.99 |
|    0003 |  1.69 |
|    0004 | 19.95 |
+---------+-------+
```

8.3.4 拥有某个字段的组间最大值的行

“对每篇文章，找出有最贵的价格的交易者。”

在 ANSI SQL 中，我可以用这样一个子查询做到：

```
SELECT article, dealer, price
```

```
FROM   shop s1
WHERE  price=(SELECT MAX(s2.price)
              FROM shop s2
              WHERE s1.article = s2.article)
```

在 MySQL 中，最好是分几步做到：

1. 得到一个表(文章，maxprice)。见 8.3.4 拥有某个域的组间最大值的行。
2. 对每篇文章，得到对应于存储最大价格的行。

这可以很容易用一个临时表做到：

```
CREATE TEMPORARY TABLE tmp (
        article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
        price   DOUBLE(16,2)             DEFAULT '0.00' NOT NULL);

LOCK TABLES article read;

INSERT INTO tmp SELECT article, MAX(price) FROM shop GROUP BY article;

SELECT article, dealer, price FROM shop, tmp
WHERE shop.article=tmp.articel AND shop.price=tmp.price;

UNLOCK TABLES;

DROP TABLE tmp;
```

如果你不使用一个 TEMPORARY 表，你也必须锁定“tmp”表。

“它能一个单个查询做到吗？”

是的，但是只有使用我称之为“MAX-CONCAT 诡计”的一个相当低效的诡计：

```
SELECT article,
       SUBSTRING( MAX( CONCAT(LPAD(price,6,'0'),dealer) ), 7) AS dealer,
  0.00+LEFT(      MAX( CONCAT(LPAD(price,6,'0'),dealer) ), 6) AS price
FROM   shop
GROUP BY article;
```

```
+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | C      |  1.69 |
```

```
|     0004 | D        | 19.95 |
+----------+--------+-------+
```

最后例子当然能通过在客户程序中分割连结的列使它更有效一点。

8.3.5 使用外键

不需要外键联结 2 个表。

MySQL 唯一不做的事情是 CHECK 以保证你使用的键确实在你正在引用表中存在，并且它不自动从有一个外键定义的表中删除行。如果你象平常那样使用你的键值，它将工作得很好！

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

CREATE TABLE shirts (
    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 persons,
    PRIMARY KEY (id)
);

INSERT INTO persons VALUES (NULL, 'Antonio Paz');

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

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

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

SELECT * FROM persons;
+----+--------------------+
```

```
| id | name                |
+----+---------------------+
|  1 | Antonio Paz         |
|  2 | Lilliana Angelovska |
+----+---------------------+

SELECT * FROM shirts;
+----+---------+--------+-------+
| 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 persons p, shirts s
 WHERE p.name LIKE 'Lilliana%'
   AND s.owner = p.id
   AND s.color &lt;&gt; 'white';

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

8.4 创造并使用一个数据库

既然你知道怎样输入命令，现在是存取一个数据库的时候了。

假定在你的家(你的“动物园”)中有很多宠物，并且你想追踪关于他们各种各样类型的信息。你可以通过创建表来保存你的数据并根据所需要的信息装载他们做到，然后你可以通过从表中检索数据来回答关于你的动物不同种类的问题。本节显示如何做到所有这些事情：

- 怎样创建一个数据库
- 怎样创建一个数据库表
- 怎样装载数据到数据库表

- 怎样以各种方法从表中检索数据
- 怎样使用多个表

动物园数据库将会是简单的(故意的)，但是不难把它想象成可能用到相似类型数据库的真实世界情况。例如，这样的一个数据库能被一个农夫用来追踪家畜，或由一个兽医追踪病畜记录。

使用 SHOW 语句找出在服务器上当前存在什么数据库：

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

数据库列表可能在你的机器上是不同的，但是 mysql 和 test 数据库很可能的在其间。mysql 是必需的，因为它描述用户存取权限，test 数据库经常作为一个工作区提供给用户试试身手。

如果 test 数据库存在，尝试存取它：

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

注意，USE，类似 QUIT，不需要一个分号。（如果你喜欢，你可以用一个分号终止这样的语句；这无碍)USE 语句在使用上也有另外一个特殊的地方：它必须在一个单行上给出。

你可列在后面的例子中使用 test 数据库(如果你能访问它)，但是你在该数据库创建的任何东西可以被与访问它的其他人删除，为了这个原因，你可能应该询问你的 MySQL 管理员许可你自己使用的一个数据库。假定你想要调用你的 menagerie，管理员需要执行一个这样的命令：

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

这里 your_mysql_name 是分配给你的 MySQL 用户名。

8.4.1 创建并选用一个数据库

如果在设置你的权限时，管理员为你创建了数据库，你可以开始使用它。否则，你需要自己创建它：

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

在 Unix 下，数据库名字是区分大小写的(不像 SQL 关键词)，因此你必须总是以 menagerie 引用你的数据库，不是 Menagerie、MENAGERIE 或一些其他变种。对表名也是这样的。（在 Windows 下，该限制不适用，尽管你必须在一个给定的查询中使用同样的大小写来引用数据库和表。）

创建了一个数据库并不选定以使用它，你必须明确地做这件事。为了使 menagerie 称为当前的数据库，使用这个命令：

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

你的数据库只需要创建一次，但是你必须在每次启动一个 mysql 会话时为使用而选择它。你可以由发出上面一个 USE 语句做到。另外，当你调用时 mysql，你可在命令行上选择数据库，就在你可能需要提供的任何连接参数之后指定其名字。例如：

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

注意，menagerie 不是你在刚才所示命令的口令。如果你想要在命令行上在-p 选项后提供你的口令，你必须做到没有多余的空格(例如，如-pmypassword，不是-p mypassword)。然而，不建议把你的口令放在命令行上，因为这样做把它暴露出来，能被在你的机器上登录的其他用户窥探到。

8.4.2 创建一个数据库表

创建数据库是容易的部分，但是在这时它是空的，正如 SHOW TABLES 将告诉你：

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

较难的部分是决定你的数据库结构应该是什么：你将需要什么数据库表，和在他们中有什么样的列。

你将需要一个包含你每个宠物的记录的表。它可称为 pet 表，并且它应该包含，最少，每个动物的名字。因为名字本身不是很有趣，表应该包含另外的信息。例如，如果在你豢养宠物的家庭有超过一个人，你可能想要列出每个动物的主人。你可能也想要记录例如种类和性别的一些基本的描述信息。

年龄呢？那可能有趣，但是在一个数据库中存储不是一件好事情。年龄随着时间流逝而变化，这意味着你将要不断地更新你的记录。相反，存储一个固定值例如生日比较好，那么，无论何时你需要年龄，你可以以当前日期和出生日期之间的

差别来计算它。MySQL 为日期运算提供了函数，因此这并不困难。存储出生日期而非年龄也有其他优点：

- 你可以将数据库用于这样的任务例如生成即将到来的宠物生日的提示。(如果你认为这类查询是点蠢，注意，这与在一个商务数据库来标示你不久要给它发出生日祝贺的客户的环境中是同一个问题，因为计算机帮助私人联络。)
- 你可以相对于日期而不止是当前日期来计算年龄。例如，如果你在数据库存储死亡日期，你能容易计算一只宠物是何时多大死的。

你可能想到 pet 表中其他有用的其他类型信息，但是到目前为止这些现在是足够了：名字、主人、种类，性别、出生和死亡日期。

使用一个 CREATE TABLE 语句指定你的数据库表的布局：

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
    -> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

VARCHAR 对 name、owner 和 species 列是个好的选择，因为列值将会是变长的。这些列的长度都不必是相同的，而且不必是 20。你可以挑选从 1 到 255 的任何长度，无论哪个对你来说好象最合理。(如果你做了较差的选择，以后会变得你需要一个更长的字段，MySQL 提供一个 ALTER TABLE 语句。)

动物性表可以用许多方法表示，例如，"m"和"f"，或也许"male"和"female"。使用单个字符"m"和"f"是最简单的。

为 birth 和 death 列使用 DATE 数据类型是相当明显的选择。

既然你创建了一个表，SHOW TABLES 应该产生一些输出：

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

为了验证你的表是按你期望的方式被创建，使用一个 DESCRIBE 语句：

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
```

```
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

你能随时 DESCRIBE，例如，如果你忘记在你表中的列的名字或他们是什么类型。

8.4.3 将数据装入一个数据库表

在你创建表后，你需要充实它。LOAD DATA 和 INSERT 语句用于此。

假定你的宠物纪录描述如下。（观察到 MySQL 期望日期时以 YYYY-MM-DD 格式；这可能与你习惯的不同。）

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1998-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

因为你是从一张空表开始的，充实它的一个容易方法是创建包含为你的动物各一行一个文本文件，然后用一个单个语句装载文件的内容到表中。

你可以创建一个文本文件“pet.txt”，每行包含一个记录，用定位符(tab)把值分开，并且以在 CREATE TABLE 语句中列出的列次序给出。对于丢失的值(例如未知的性别，或仍然活着的动物的死亡日期)，你可以使用 NULL 值。为了在你的文本文件表示这些，使用\N。例如，对 Whistler 鸟的记录看起来像这样的(这里在值之间的空白是一个单个的定位字符)：

Whistler	Gwen	bird	\N	1997-12-09	\N

为了装载文本文件“pet.txt”到 pet 表中，使用这个命令：

```
mysql> LOAD DATA LOCAL INFILE "pet.txt" INTO TABLE pet;
```

如果你愿意，你能明确地在 LOAD DATA 语句中指出列值的分隔符和行尾标记，但是缺省是定位符和换行符。这些对争取读入文件“pet.txt”的语句是足够的。

当你想要一次增加一个新记录时，INSERT 语句是有用的。在它最简单的形式，你为每一列提供值，以列在 CREATE TABLE 语句被列出的顺序。假定 Diane 把一只新仓鼠命名为 Puffball，你可以使用一个这样 INSERT 语句增加一条新记录：

```
mysql> INSERT INTO pet
    -> VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

注意，这里字符串和日期值被指定为引号扩起来的字符串。另外，用 INSERT，你能直接插入 NULL 代表不存在的值。你不能使用\N，就像你用 LOAD DATA 做的那样。

从这个例子，你应该能看到涉及很多的键入用多个 INSERT 语句而非单个 LOAD DATA 语句装载你的初始记录。

8.4.4 从一个数据库表检索信息

SELECT 语句被用来从一张桌子拉出信息。语句的一般格式是：

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

what_to_select 指出你想要看到的，这可以是列的一张表，或*表明“所有的列”。which_table 指出你想要从其检索数据的表。WHERE 子句是可选的，如果它在，conditions_to_satisfy 指定行必须满足的检索条件。

8.4.4.1 选择所有数据

SELECT 最简单的形式是从一张表中检索每样东西：

```
mysql> SELECT * FROM pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1998-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
```

```
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       |
+----------+--------+---------+------+------------+------------+
```

如果你想要考察整个表，这种形式的 SELECT 是很有用的。例如，在你刚刚给它装载了你的初始数据集装以后。当它发生时，刚才显示的输出揭示了在你的数据文件的一个错误：在 Bowser 死了以后，它好象要出生了！请教你原来的家谱，你发现正确的出生年是 1989，而不是 1998。

至少有一些修正它的方法：

- 编辑文件“pet.txt”改正错误，然后使用 DELETE 和 LOAD DATA 弄空表并且再次装载它:
- mysql> DELETE FROM pet;
- mysql> LOAD DATA LOCAL INFILE "pet.txt" INTO TABLE pet;

  然而，如果你这样做，你必须重新输入 Puffball 记录。

- 用一个 UPDATE 语句仅修正错误记录：
- mysql> UPDATE pet SET birth = "1989-08-31" WHERE name = "Bowser";

如上所示，检索整个表是容易的，但是一般你不想那样做，特别地当表变得很大时。相反，你通常对回答一个特别的问题更感兴趣，在这种情况下你在你想要的信息上指定一些限制。让我们看一些他们回答有关你宠物的问题的选择查询。

8.4.4.2 选择特定行

你能从你的表中只选择特定的行。例如，如果你想要验证你对 Bowser 的出生日期所做的改变，像这样精选 Bowser 的记录：

```
mysql> SELECT * FROM pet WHERE name = "Bowser";
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

输出证实年份现在正确记录为 1989，而不是 1998。

字符串比较通常是大小些无关的，因此你可以指定名字为"bowser"、"BOWSER"等等，查询结果将是相同的。

你能在任何列上指定条件，不只是 name。例如，如果你想要知道哪个动物在 1998 以后出生的，测试 birth 列：

```
mysql> SELECT * FROM pet WHERE birth >= "1998-1-1";
```

```
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Puffball | Diane | hamster | f    | 1999-03-30 | NULL  |
+----------+-------+---------+------+------------+-------+
```

你能组合条件，例如，找出雌性的狗：

```
mysql> SELECT * FROM pet WHERE species = "dog" AND sex = "f";
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

上面的查询使用 AND 逻辑操作符，也有一个 OR 操作符：

```
mysql> SELECT * FROM pet WHERE species = "snake" OR species = "bird";
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL  |
| Slim     | Benny | snake   | m    | 1996-04-29 | NULL  |
+----------+-------+---------+------+------------+-------+
```

AND 和 OR 可以混用。如果你这样做，使用括号指明条件应该如何被分组是一个好主意：

```
mysql> SELECT * FROM pet WHERE (species = "cat" AND sex = "m")
    -> OR (species = "dog" AND sex = "f");
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

8.4.4.3 选择特定列

如果你不想要看到你的表的整个行，就命名你感兴趣的列，用逗号分开。例如，如果你想要知道你的动物什么时候出生的，精选 name 和 birth 列：

```
mysql> SELECT name, birth FROM pet;
```

```
+----------+------------+
| name     | birth      |
+----------+------------+
| 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 |
+----------+------------+
```

找出谁拥有宠物，使用这个查询：

```
mysql> SELECT owner FROM pet;
+--------+
| owner  |
+--------+
| Harold |
| Gwen   |
| Harold |
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
| Benny  |
| Diane  |
+--------+
```

然而，注意到查询简单地检索每个记录的 owner 字段，并且他们中的一些出现多次。为了使输出减到最少，通过增加关键词 DISTINCT 检索出每个唯一的输出记录：

```
mysql> SELECT DISTINCT owner FROM pet;
+--------+
| owner  |
+--------+
| Benny  |
| Diane  |
| Gwen   |
| Harold |
+--------+
```

你能使用一个 WHERE 子句把行选择与列选择相结合。例如，为了只得到狗和猫的出生日期，使用这个查询：

```
mysql> SELECT name, species, birth FROM pet
    -> WHERE species = "dog" OR species = "cat";
+--------+---------+------------+
| name   | species | birth      |
+--------+---------+------------+
| Fluffy | cat     | 1993-02-04 |
| Claws  | cat     | 1994-03-17 |
| Buffy  | dog     | 1989-05-13 |
| Fang   | dog     | 1990-08-27 |
| Bowser | dog     | 1989-08-31 |
+--------+---------+------------+
```

8.4.4.4 排序行

你可能已经注意到前面的例子中结果行没有以特定的次序被显示。然而，当行以某个有意义的方式排序，检验查询输出通常是更容易的。为了排序结果，使用一个 ORDER BY 子句。

这里是动物生日，按日期排序：

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
+----------+------------+
| name     | birth      |
+----------+------------+
| 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 |
+----------+------------+
```

为了以逆序排序，增加 DESC（下降 ）关键字到你正在排序的列名上：

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
+----------+------------+
| name     | birth      |
+----------+------------+
| 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 |
+----------+------------+
```

你能在多个列上排序。例如，按动物的种类排序，然后按生日，首先是动物种类中最年轻的动物，使用下列查询：

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

注意 DESC 关键词仅适用于紧跟在它之前的列名字(birth)；species 值仍然以升序被排序。

8.4.4.5 日期计算

MySQL 提供几个函数，你能用来执行在日期上的计算，例如，计算年龄或提取日期的部分。

为了决定你的每个宠物有多大，用出生日期和当前日期之间的差别计算年龄。通过变换 2 个日期到天数，取差值，并且用 365 除(在一年里的天数)：

```
mysql> SELECT name, (TO_DAYS(NOW())-TO_DAYS(birth))/365 FROM pet;
+----------+-------------------------------------+
| name     | (TO_DAYS(NOW())-TO_DAYS(birth))/365 |
+----------+-------------------------------------+
| Fluffy   |                                6.15 |
| Claws    |                                5.04 |
```

```
| Buffy    |                                    9.88 |
| Fang     |                                    8.59 |
| Bowser   |                                    9.58 |
| Chirpy   |                                    0.55 |
| Whistler |                                    1.30 |
| Slim     |                                    2.92 |
| Puffball |                                    0.00 |
+----------+-------------------------------------+
```

尽管查询可行，关于它还有能被改进的一些事情。首先，如果行以某个次序表示，其结果能更容易被扫描。第二，年龄列的标题不是很有意义的。

第一个问题通过增加一个 ORDER BY name 子句按名字排序输出来解决。为了处理列标题，为列提供一个名字以便一个不同的标签出现在输出中(这被称为一个列别名)：

```
mysql> SELECT name, (TO_DAYS(NOW())-TO_DAYS(birth))/365 AS age
    -> FROM pet ORDER BY name;
+----------+------+
| name     | age  |
+----------+------+
| Bowser   | 9.58 |
| Buffy    | 9.88 |
| Chirpy   | 0.55 |
| Claws    | 5.04 |
| Fang     | 8.59 |
| Fluffy   | 6.15 |
| Puffball | 0.00 |
| Slim     | 2.92 |
| Whistler | 1.30 |
+----------+------+
```

为了按 age 而非 name 排序输出，只要使用一个不同 ORDER BY 子句：

```
mysql>  SELECT name, (TO_DAYS(NOW())-TO_DAYS(birth))/365 AS age
    ->  FROM pet ORDER BY age;
+----------+------+
| name     | age  |
+----------+------+
| Puffball | 0.00 |
| Chirpy   | 0.55 |
| Whistler | 1.30 |
| Slim     | 2.92 |
| Claws    | 5.04 |
| Fluffy   | 6.15 |
```

```
| Fang     | 8.59 |
| Bowser   | 9.58 |
| Buffy    | 9.88 |
+----------+------+
```

一个类似的查询可以被用来确定已经死亡动物的死亡年龄。你通过检查 death 值是否是 NULL 来决定那些是哪些动物，然后，对于那些有非 NULL 值，计算在 death 和 birth 值之间的差别：

```
mysql>  SELECT name, birth, death, (TO_DAYS(death)-TO_DAYS(birth))/365
AS age
    ->  FROM pet WHERE death IS NOT NULL ORDER BY age;
+--------+------------+------------+------+
| name   | birth      | death      | age  |
+--------+------------+------------+------+
| Bowser | 1989-08-31 | 1995-07-29 | 5.91 |
+--------+------------+------------+------+
```

差询使用 death IS NOT NULL 而非 death != NULL，因为 NULL 是特殊的值，这以后会解释。见 8.4.4.6 用 NULL 值工作。

如果你想要知道哪个动物下个月过生日，怎么办？对于这类计算，年和天是无关的，你简单地想要提取 birth 列的月份部分。**MySQL** 提供几个日期部分的提取函数，例如 YEAR()、MONTH()和 DAYOFMONTH()。在这里 MONTH()是适合的函数。为了看它怎样工作，运行一个简单的查询，显示 birth 和 MONTH(birth)的值：

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
+----------+------------+--------------+
| name     | birth      | MONTH(birth) |
+----------+------------+--------------+
| 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 |
+----------+------------+--------------+
```

用下个月的生日找出动物也是容易的。假定当前月是 4 月，那么月值是 4 并且你寻找在 5 月出生的动物 (5 月)， 象这样：

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
```

```
+-------+------------+
| name  | birth      |
+-------+------------+
| Buffy | 1989-05-13 |
+-------+------------+
```

当然如果当前月份是 12 月，就有点复杂了。你不是只把加 1 到月份数(12)上并且寻找在 13 月出生的动物，因为没有这样的月份。相反，你寻找在 1 月出生的动物(1 月) 。

你甚至可以编写查询以便不管当前月份是什么它都能工作。这种方法你不必在查询中使用一个特定的月份数字，DATE_ADD()允许你把时间间隔加到一个给定的日期。如果你把一个月加到 NOW()值上，然后用 MONTH()提取月份部分，结果产生寻找生日的月份：

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MONTH(DATE_ADD(NOW(), INTERVAL 1 MONTH));
```

完成同样任务的一个不同方法是加 1 以得出当前月份的下一个月(在使用取模函数(MOD)后，如果它当前是 12，则“绕回”月份到值 0)：

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MOD(MONTH(NOW()), 12) + 1;
```

注意，MONTH 返回在 1 和 12 之间的一个数字，且 MOD(something,12)返回在 0 和 11 之间的一个数字，因此必须在 MOD()以后加 1，否则我们将从 11 月(11) 跳到 1 月(1)。

8.4.4.6 NULL 值操作

NULL 值可能很奇怪直到你习惯于它。概念上，NULL 意味着“没有值”或“未知值”，且它被看作有点与众不同的值。为了测试 NULL，你不能使用算术比较运算符例如=、<或!=。为了说明它，试试下列查询：

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

很清楚你从这些比较中得到毫无意义的结果。相反使用 IS NULL 和 IS NOT NULL 操作符：

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
```

```
+-----------+--------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+--------------+
|         0 |              1 |
+-----------+--------------+
```

在 MySQL 中，0 意味着假而 1 意味着真。

NULL 这样特殊的处理是为什么，在前面的章节中，为了决定哪个动物不再是活着的，使用 death IS NOT NULL 而不是 death != NULL 是必要的。

8.4.4.7 模式匹配

MySQL 提供标准的 SQL 模式匹配，以及一种基于象 Unix 实用程序如 vi、grep 和 sed 的扩展正则表达式模式匹配的格式。

SQL 的模式匹配允许你使用“_”匹配任何单个字符，而“%”匹配任意数目字符(包括零个字符)。在 MySQL 中，SQL 的模式缺省是忽略大小写的。下面显示一些例子。注意在你使用 SQL 模式时，你不能使用=或!=；而使用 LIKE 或 NOT LIKE 比较操作符。

为了找出以“b”开头的名字：

```
mysql> SELECT * FROM pet WHERE name LIKE "b%";
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

为了找出以“fy”结尾的名字：

```
mysql> SELECT * FROM pet WHERE name LIKE "%fy";
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

为了找出包含一个“w”的名字：

```
mysql> SELECT * FROM pet WHERE name LIKE "%w%";
+----------+-------+---------+------+------------+------------+
```

```
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

为了找出包含正好 5 个字符的名字，使用“_”模式字符：

```
mysql> SELECT * FROM pet WHERE name LIKE "_____";
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

由 **MySQL** 提供的模式匹配的其他类型是使用扩展正则表达式。当你对这类模式进行匹配测试时，使用 REGEXP 和 NOT REGEXP 操作符(或 RLIKE 和 NOT RLIKE，它们是同义词)。

扩展正则表达式的一些字符是：

- “.”匹配任何单个的字符。
- 一个字符类“[...]”匹配在方括号内的任何字符。例如，“[abc]”匹配“a”、“b”或“c”。为了命名字符的一个范围，使用一个“-”。“[a-z]”匹配任何小写字母，而“[0-9]”匹配任何数字。
- “ * ”匹配零个或多个在它前面的东西。例如，“x*”匹配任何数量的“x”字符，“[0-9]*”匹配的任何数量的数字，而“.*”匹配任何数量的任何东西。
- 正则表达式是区分大小写的，但是如果你希望，你能使用一个字符类匹配两种写法。例如，“[aA]”匹配小写或大写的“a”而“[a-zA-Z]”匹配两种写法的任何字母。
- 如果它出现在被测试值的任何地方，模式就匹配(只要他们匹配整个值，SQL 模式匹配)。
- 为了定位一个模式以便它必须匹配被测试值的开始或结尾，在模式开始处使用“^”或在模式的结尾用“$”。

为了说明扩展正则表达式如何工作，上面所示的 LIKE 查询在下面使用 REGEXP 重写：

为了找出以“b”开头的名字，使用“^”匹配名字的开始并且“[bB]”匹配小写或大写的“b”：

```
mysql> SELECT * FROM pet WHERE name REGEXP "^[bB]";
```

```
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

为了找出以“fy”结尾的名字，使用“$”匹配名字的结尾：

```
mysql> SELECT * FROM pet WHERE name REGEXP "fy$";
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

为了找出包含一个“w”的名字，使用“[wW]”匹配小写或大写的“w”：

```
mysql> SELECT * FROM pet WHERE name REGEXP "[wW]";
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

既然如果一个正规表达式出现在值的任何地方，其模式匹配了，就不必再先前的查询中在模式的两方面放置一个通配符以使得它匹配整个值，就像如果你使用了一个 SQL 模式那样。

为了找出包含正好 5 个字符的名字，使用“^”和“$”匹配名字的开始和结尾，和 5 个“.”实例在两者之间：

```
mysql> SELECT * FROM pet WHERE name REGEXP "^.....$";
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

你也可以使用“{n}”“重复 n 次”操作符重写先前的查询：

```
mysql> SELECT * FROM pet WHERE name REGEXP "^.{5}$";
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

8.4.4.8 行计数

数据库经常用于回答这个问题，“某个类型的数据在一张表中出现的频度?”例如，你可能想要知道你有多少宠物，或每位主人有多少宠物，或你可能想要在你的动物上施行各种类型的普查。

计算你拥有动物的总数字与“在 pet 表中有多少行?”是同样的问题，因为每个宠物有一个记录。COUNT()函数计数非 NULL 结果的数目，所以数你的动物的查询看起来像这样：

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

在前面，你检索了拥有宠物的人的名字。如果你想要知道每个主人有多少宠物，你可以使用 COUNT()函数：

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

注意，使用 GROUP BY 对每个 owner 分组所有记录，没有它，你得到的一切是一条错误消息：

```
mysql> SELECT owner, COUNT(owner) FROM pet;
ERROR 1140 at line 1: Mixing of GROUP columns (MIN(),MAX(),COUNT()...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

COUNT()和 GROUP BY 对以各种方式分类你的数据很有用。下列例子显示出实施动物普查操作的不同方式。

每种动物数量：

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+---------+----------+
| species | COUNT(*) |
+---------+----------+
| bird    |        2 |
| cat     |        2 |
| dog     |        3 |
| hamster |        1 |
| snake   |        1 |
+---------+----------+
```

每中性别的动物数量：

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+------+----------+
| sex  | COUNT(*) |
+------+----------+
| NULL |        1 |
| f    |        4 |
| m    |        4 |
+------+----------+
```

（在这个输出中，NULL 表示“未知性别”。）

按种类和性别组合的动物数量：

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | NULL |        1 |
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

当你使用 COUNT()时，你不必检索整个一张表。例如，先前的查询，当只在狗和猫上施行时，看起来像这样：

```
mysql> SELECT species, sex, COUNT(*) FROM pet
    -> WHERE species = "dog" OR species = "cat"
    -> GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
+---------+------+----------+
```

或，如果你仅需要知道已知性别的按性别的动物数目：

```
mysql> SELECT species, sex, COUNT(*) FROM pet
    -> WHERE sex IS NOT NULL
    -> GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

8.4.5 使用多个数据库表

pet 表追踪你有哪个宠物。如果你想要记录他们的其他信息，例如在他们一生中事件看兽医或何时后代出生，你需要另外的表。这张表应该像什么呢？

- 它需要包含宠物名字因此你知道每个事件属于此动物。
- 它需要一个日期因此你知道事件什么时候发生的。
- 需要一个字段描述事件。
- 如果你想要可分类事件，有一个事件类型字段将是有用的。

给出了这些考虑，为 event 表的 CREATE TABLE 语句可能看起来像这样：

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
    -> type VARCHAR(15), remark VARCHAR(255));
```

就象 pet 表，最容易的示通过创建包含信息的一个定位符分隔的文本文件装载初始记录：

Fluffy	1995-05-15	litter	4 kittens, 3 female, 1 male
Buffy	1993-06-23	litter	5 puppies, 2 female, 3 male
Buffy	1994-06-19	litter	3 puppies, 3 female
Chirpy	1999-03-21	vet	needed beak straightened
Slim	1997-08-03	vet	broken rib
Bowser	1991-10-12	kennel	
Fang	1991-10-12	kennel	
Fang	1998-08-28	birthday	Gave him a new chew toy
Claws	1998-03-17	birthday	Gave him a new flea collar
Whistler	1998-12-09	birthday	First birthday

象这样装载记录：

```
mysql> LOAD DATA LOCAL INFILE "event.txt" INTO TABLE event;
```

基于你从已经运行在 pet 表上的查询中学到的，你应该能执行在 event 表中记录的检索；原则是一样的。但是什么时候是 event 表本身不足以回答你可能问的问题呢？

当他们有了一窝小动物时，假定你想要找出每只宠物的年龄。 event 表指出何时发生，但是为了计算母亲的年龄，你需要她的出生日期。既然它被存储在 pet 表中，为了查询你需要两张表：

```
mysql> SELECT pet.name, (TO_DAYS(date) - TO_DAYS(birth))/365 AS age,
remark
    -> FROM pet, event
    -> WHERE pet.name = event.name AND type = "litter";
+--------+------+-----------------------------+
| name   | age  | remark                      |
+--------+------+-----------------------------+
| Fluffy | 2.27 | 4 kittens, 3 female, 1 male |
| Buffy  | 4.12 | 5 puppies, 2 female, 3 male |
| Buffy  | 5.10 | 3 puppies, 3 female         |
+--------+------+-----------------------------+
```

关于该查询要注意的几件事情：

- FROM 子句列出两个表，因为查询需要从他们两个拉出信息。
- 当组合(联结-join)来自多个表的信息时，你需要指定在一个表中的记录怎样能匹配其它表的记录。这很简单，因为它们都有一个 name 列。查询使用 WHERE 子句基于 name 值来匹配 2 个表中的记录。
- 因为 name 列出现在两个表中，当引用列时，你一定要指定哪个表。这通过把表名附在列名前做到。

你不必有 2 个不同的表来执行一个联结。如果你想要将一个表的记录与同一个表的其他记录进行比较，联结一个表到自身有时是有用的。例如，为了在你的宠物之中繁殖配偶，你可以用 pet 联结自身来进行相似种类的雄雌配对：

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
    -> FROM pet AS p1, pet AS p2
    -> WHERE p1.species = p2.species AND p1.sex = "f" AND p2.sex = "m";
+--------+------+--------+------+---------+
| name   | sex  | name   | sex  | species |
+--------+------+--------+------+---------+
| Fluffy | f    | Claws  | m    | cat     |
| Buffy  | f    | Fang   | m    | dog     |
| Buffy  | f    | Bowser | m    | dog     |
+--------+------+--------+------+---------+
```

在这个查询中，我们为表名指定别名以便能引用列并且使得每一个列引用关联于哪个表实例更直观。

8.5 获得数据库和表的信息

如果你忘记一个数据库或表的名字，或一个给定的表的结构是什么(例如，它的列叫什么)，怎么办？ MySQL 通过提供数据库及其支持的表的信息的几个语句解决这个问题。

你已经见到了 SHOW DATABASES，它列出由服务器管理的数据库。为了找出当前选择了哪个数据库，使用 DATABASE()函数：

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

如果你还没选择任何数据库，结果是空的。

为了找出当前的数据库包含什么表(例如，当你不能确定一个表的名字)，使用这个命令：

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

如果你想要知道一个表的结构，DESCRIBE 命令是有很用的；它显示有关一个表的每个列的信息：

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

Field 显示列名字，Type 是为列的数据类型，Null 表示列是否能包含 NULL 值，Key 显示列是否被索引而 Default 指定列的缺省值。

如果你在一个表上有索引，SHOW INDEX FROM tbl_name 生成有关它们的信息。

8.6 以批处理模式使用 mysql

在前面的章节中，你交互式地使用 mysql 输入查询并且查看结果。你也可以以批模式运行 mysql。为了做到这些，把你想要运行的命令放在一个文件中，然后告诉 mysql 从文件读取它的输入：

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

如果你需要在命令行上指定连接参数，命令可能看起来像这样：

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

当你这样使用 mysql 时，你正在创建一个脚本文件，然后执行脚本。

为什么要使用一个脚本？有很多原因：

- 如果你重复地运行查询(比如说，每天或每周)，把它做成一个脚本使得你在每次执行它时避免重新键入。
- 你能通过拷贝并编辑脚本文件从类似的现有的查询生成一个新查询。
- 当你正在开发查询时，批模式也是很有用的，特别对多行命令或多行语句序列。如果你犯了一个错误，你不必重新打入所有一切，只要编辑你的脚本来改正错误，然后告诉 mysql 再次执行它。
- 如果你有一个产生很多输出的查询，你可以通过一个分页器而不是盯着它翻屏到你屏幕的顶端来运行输出：
- shell> mysql < batch-file | more

- 你能捕捉输出到一个文件中进行更一步的处理：
- shell> mysql < batch-file > mysql.out

- 你可以散发脚本给另外的人，因此他们也能运行命令。
- 一些情况不允许交互地使用，例如，当你从一个 cron 任务中运行查询时。在这种情况下，你必须使用批模式。

当你以批模式运行 mysql 时，比起你交互地使用它时，其缺省输出格式是不同的(更简明些)。例如，当交互式运行 SELECT DISTINCT species FROM pet 时，输出看起来像这样：

```
+---------+
| species |
+---------+
| bird    |
| cat     |
| dog     |
| hamster |
| snake   |
+---------+
```

但是当以批模式运行时，像这样：

```
species
bird
cat
dog
hamster
snake
```

如果你想要在批模式中得到交互的输出格式，使用 mysql -t。为了回显以输出被执行的命令，使用 mysql -vvv。

8.7 双胞胎项目的查询（实例）

在 Analytikerna 和 Lentus，我们为一个大的研究项目工程一直在做系统和现场工作。这个项目是 Institute of Environmental Medicine at Karolinska Institutet Stockholm 和 the Section on Clinical Research in Aging and Psychology at the University of Southern California 的合作项目。

双胞胎研究的更多信息可在下列链接找到：

http://www.imm.ki.se/TWIN/TWINUKW.HTM

项目的后面部分是用一个用 Perl 和 **MySQL** 编写的 web 接口来管理。

每天晚上所有会谈的数据被移入一个 **MySQL** 数据库。

8.7.1 找出所有非独处的双胞胎

下列查询用来决定谁进入项目的第二部分：

```
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.event as event1,
        td.suspect as tsuspect1,
        id.suspect as isuspect1,
        td.severe as tsevere1,
        id.severe as isevere1,
        p2.dead as dead2,
        l2.event 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_date
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 sceening 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.death_date = 0 or
           (((to_days(p2.death_date) - to_days(p2.birthday)) / 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.event = '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;
```

一些解释：

concat(p1.id, p1.tvab) + 0 as tvid
: 我们想要在 id 和 tvab 的连接上以数字序排序。结果加 0 使得 MySQL 把结果当作一个数字。

列 id
: 这标识一对双胞胎。它是所有表中的一个键。

列 tvab
: 这标识双胞胎中的一个。它有值 1 或 2。

列 ptvab
: 这是 tvab 一个逆。当 tvab 是 1，它是 2，并且反过来也如此。它存在以保存键入并且使它更容易为 MySQL 优化查询。

这个查询表明，怎样用联结(p1 和 p2)从同一个表中查找表。在例子中，这被用来检查双胞胎的一个是否在 65 岁前死了。如果因此，行不返回。

上述所有双胞胎信息存在于所有表中。我们在 id,tvab 两者上的键值（所有表）和在 id,ptvab 上的键(person_data)以使查询更快。

在我们的生产机器上(一台 200MHz UltraSPARC)，这个查询返回大约 150-200 行并且不超过一秒的时间。

上面所用的表的当前记录数是：

表	行数
person_data	71074
lentus	5291
twin_project	5286

twin_data	2012
informant_data	663
harmony	381
postal_groups	100

8.7.2 显示关于双胞胎近况的表

每一次会面以一个称为 event 的状态码结束。下面显示的查询被用来显示按事件组合的所有双胞胎的表。这表明多少对双胞胎已经完成，多少对的其中之一已完成而另一个拒绝了，等等。

```
select
        t1.event,
        t2.event,
        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 sceening survey */
        and tp.survey_no = 5
        /* This makes each pair only appear once */
        and t1.tvab='1' and t2.tvab='2'
group by
        t1.event, t2.event;
```