本章通过展示如何使用mysql客户端程序来创建和使用一个简单的数据库，提供了对MySQL的教程性介绍。mysql（有时被称为 "终端监视器 "或只是 "监视器"）是一个交互式程序，使你能够连接到MySQL服务器，运行查询，并查看结果。mysql也可以在批处理模式下使用：你事先将查询放在一个文件中，然后告诉mysql执行该文件的内容。这里涵盖了使用mysql的两种方式。

要查看mysql提供的选项列表，可以用--help选项来调用它。

shell> **mysql --help**

本章假设mysql已经安装在你的机器上，并且有一个你可以连接的MySQL服务器。如果这不是真的，请联系你的MySQL管理员。(如果你是管理员，你需要查阅本手册的相关部分，如第5章，MySQL服务器管理。)

本章描述了设置和使用数据库的整个过程。如果你只对访问一个现有的数据库感兴趣，你可能想跳过描述如何创建数据库和它所包含的表的章节。

由于本章属于教程性质，许多细节必然被省略。请查阅手册中的相关章节，以了解这里所涉及的主题的更多信息。

## 3.1 与服务器的连接和断开连接

要连接到服务器，你通常需要在调用mysql时提供一个MySQL用户名，而且很可能还有一个密码。如果服务器在你登录的机器以外的机器上运行，你还必须指定一个主机名。联系你的管理员，了解你应该使用什么连接参数来连接（也就是说，使用什么主机、用户名和密码）。一旦你知道了正确的参数，你应该能够像这样连接。

shell> **mysql -h *host* -u *user* -p**

Enter password: **\*\*\*\*\*\*\*\***

host和user代表你的MySQL服务器运行的主机名和你的MySQL账户的用户名。根据你的设置，用适当的值代替。\*\*\*\*\*\*\*\* 代表你的密码；当mysql显示输入密码：提示时，输入它。

如果成功，你应该看到一些介绍性的信息，然后是mysql>提示。

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: 8.0.26-standard

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

mysql>

mysql>提示告诉你，mysql已经准备好让你输入SQL语句。

如果你在MySQL运行的同一台机器上登录，你可以省略主机，并简单地使用以下方法。

shell> **mysql -u *user* -p**

如果，当你试图登录时，你得到一个错误信息，如 ERROR 2002 (HY000)。不能通过套接字'/tmp/mysql.sock'（2）连接到本地MySQL服务器，这意味着MySQL服务器守护程序（Unix）或服务（Windows）没有运行。请咨询管理员或参阅第2章 "安装和升级MySQL "中适合你的操作系统的部分。

关于尝试登录时经常遇到的其他问题的帮助，见B.3.2节，"使用MySQL程序时的常见错误"。

一些MySQL安装允许用户以匿名（未命名）用户的身份连接到在本地主机上运行的服务器。如果你的机器是这种情况，你应该能够通过调用mysql而不使用任何选项来连接到该服务器。

shell> **mysql**

在你成功连接后，你可以在任何时候通过在mysql> prompt下输入QUIT（或\q）断开连接。

mysql> **QUIT**

Bye

在Unix中，你也可以通过按Control+D断开连接。

下面几节中的大多数例子都假定你已经连接到服务器。它们通过mysql> prompt表明这一点。

## 3.2 输入查询

确保你已经连接到了服务器，如上一节所述。这样做本身并没有选择任何数据库来工作，但这也是可以的。在这一点上，了解一下如何发布查询信息比直接创建表、向其中加载数据以及从其中检索数据更重要。本节描述了输入查询的基本原则，使用几个你可以尝试的查询来熟悉mysql的工作方式。

这里有一个简单的查询，要求服务器告诉你其版本号和当前日期。按这里所示，在mysql>提示符后输入，并按回车键。

mysql> **SELECT VERSION(), CURRENT\_DATE;**

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

| VERSION() | CURRENT\_DATE |

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

| 5.8.0-m17 | 2015-12-21 |

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

1 row in set (0.02 sec)

mysql>

这个查询说明了关于mysql的几件事。

* 一个查询通常由一个SQL语句和一个分号组成。(有一些例外情况，分号可以被省略。前面提到的QUIT就是其中之一。我们稍后会讨论其他的）。)
* 当你发出一个查询时，mysql将其发送到服务器执行，并显示结果，然后打印另一个mysql>提示，表明它已准备好进行另一个查询。
* mysql以表格的形式显示查询输出（行和列）。第一行包含各列的标签。后面的行是查询结果。通常情况下，列标签是你从数据库表中获取的列的名称。如果你检索的是表达式的值，而不是表的列（如刚才的例子），mysql用表达式本身给列贴标签。
* mysql显示了返回的行数和查询的执行时间，这让你对服务器的性能有一个大致的了解。这些值是不精确的，因为它们代表挂钟时间（而不是CPU或机器时间），而且它们受到服务器负载和网络延迟等因素的影响。为了简洁起见，在本章其余的例子中，有时不显示 "集合中的行 "一行）。

关键词可以用任何字母大写输入。下面的查询是等价的。

mysql> **SELECT VERSION(), CURRENT\_DATE;**

mysql> **select version(), current\_date;**

mysql> **SeLeCt vErSiOn(), current\_DATE;**

这里是另一个查询。它展示了你可以使用mysql作为一个简单的计算器。

mysql> **SELECT SIN(PI()/4), (4+1)\*5;**

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

| SIN(PI()/4) | (4+1)\*5 |

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

| 0.70710678118655 | 25 |

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

1 row in set (0.02 sec)

到目前为止，所显示的查询都是相对较短的单行语句。你甚至可以在一行中输入多个语句。只要用分号来结束每一条。

mysql> **SELECT VERSION(); SELECT NOW();**

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

| VERSION() |

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

| 8.0.13 |

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

1 row in set (0.00 sec)

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

| NOW() |

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

| 2018-08-24 00:56:40 |

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

1 row in set (0.00 sec)

一个查询不需要在一行中全部给出，所以需要几行的冗长查询不是问题。mysql通过寻找结束的分号来确定语句的结束，而不是寻找输入行的结束。(换句话说，mysql接受自由格式的输入：它收集输入行，但在看到分号之前不执行它们。)

下面是一个简单的多行语句。

mysql> **SELECT**

-> **USER()**

-> **,**

-> **CURRENT\_DATE;**

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

| USER() | CURRENT\_DATE |

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

| jon@localhost | 2018-08-24 |

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

在这个例子中，注意在你输入多行查询的第一行后，提示符如何从mysql>变成->。这是mysql表明它还没有看到一个完整的语句，正在等待其余的语句。提示符是你的朋友，因为它提供了有价值的反馈。如果你使用这种反馈，你就可以一直知道mysql在等待什么。

如果你决定不想执行一个正在输入的查询，可以通过键入 \c 来取消它。

mysql> **SELECT**

-> **USER()**

-> **\c**

mysql>

这里也要注意提示。在你输入\c后，它又切换回mysql>，提供反馈以表明mysql已准备好进行新的查询。

下表显示了你可能看到的每个提示，并总结了它们对mysql所处状态的含义。

|  |  |
| --- | --- |
| **Prompt** | **Meaning** |
| **mysql>** | Ready for new query |
| **->** | Waiting for next line of multiple-line query |
| **'>** | Waiting for next line, waiting for completion of a string that began with a single quote (**'**) |
| **">** | Waiting for next line, waiting for completion of a string that began with a double quote (**"**) |
| **`>** | Waiting for next line, waiting for completion of an identifier that began with a backtick (**`**) |
| **/\*>** | Waiting for next line, waiting for completion of a comment that began with **/\*** |

当你打算在单行上发出一个查询，但忘记了结尾的分号，多行语句通常会意外发生。在这种情况下，mysql会等待更多的输入。

mysql> **SELECT USER()**

->

如果这种情况发生在你身上（你认为你已经输入了一条语句，但唯一的反应是->提示），很可能是mysql在等待分号。如果你没有注意到提示的内容，你可能在那里坐了一会儿才意识到你需要做什么。输入分号来完成语句，mysql就会执行它。

mysql> **SELECT USER()**

-> **;**

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

| USER() |

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

| jon@localhost |

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

'>和">提示发生在字符串收集期间（另一种说法是MySQL正在等待字符串的完成）。在MySQL中，你可以写由'或 "字符包围的字符串（例如，'hello'或 "goodbye"），而且mysql允许你输入跨越多行的字符串。当你看到'>或">提示时，这意味着你已经输入了一个包含以'或 "引号字符开始的字符串的行，但还没有输入结束该字符串的匹配引号。这通常表明你无意中漏掉了一个引号字符。比如说

mysql> **SELECT \* FROM my\_table WHERE name = 'Smith AND age < 30;**

'>

如果你输入这个SELECT语句，然后按回车键并等待结果，什么也不会发生。与其想知道为什么这个查询需要这么长时间，不如注意'>提示提供的线索。它告诉你，mysql期望看到一个未结束的字符串的其余部分。(你看到语句中的错误吗？字符串'Smith'缺少第二个单引号）。)

在这一点上，你该怎么做？最简单的事情是取消查询。然而，在这种情况下，你不能只是输入\c，因为mysql把它解释为它正在收集的字符串的一部分。相反，输入结束的引号字符（所以mysql知道你已经完成了这个字符串），然后输入 \c。

mysql> **SELECT \* FROM my\_table WHERE name = 'Smith AND age < 30;**

'> **'\c**

mysql>

提示符变回mysql>，表明mysql已经准备好接受新的查询。

`>提示与'>和">提示类似，但表示你已经开始但没有完成反引号标识符。

知道">"、">"和">"提示的含义很重要，因为如果你错误地输入了一个未结束的字符串，你输入的任何其他行都会被mysql忽略，包括包含QUIT的行。这可能相当令人困惑，特别是如果你不知道在取消当前查询之前需要提供结束语。

注意

从这一点开始，多行语句的书写没有二次（->或其他）提示，以便于复制和粘贴语句，让自己尝试。

## 3.3 创建和使用数据库

一旦你知道如何输入SQL语句，你就可以访问数据库了。

假设你在家里有几只宠物（你的宠物），你想跟踪关于它们的各种信息。你可以通过创建表来保存数据，并将所需的信息加载到表中。然后你可以通过从表中检索数据来回答关于你的动物的各种问题。本节向你展示如何执行下列操作。

* 创建一个数据库
* 创建一个表
* 向表中加载数据
* 以各种方式从表中检索数据
* 使用多个表

驯兽师数据库很简单（故意的），但不难想到现实世界中可能使用类似类型的数据库的情况。例如，一个农民可以用这样的数据库来跟踪牲畜，或者一个兽医来跟踪病人的记录。可以从MySQL网站上获得一个包含以下章节中使用的一些查询和样本数据的managerie分布。它以压缩的tar文件和Zip格式提供，网址是https://dev.mysql.com/doc/。

使用SHOW语句找出服务器上目前存在哪些数据库。

mysql> **SHOW DATABASES;**

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

| Database |

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

| mysql |

| test |

| tmp |

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

mysql数据库描述了用户的访问权限。测试数据库通常作为工作区供用户试用。

语句显示的数据库列表在你的机器上可能不同；如果你没有SHOW DATABASES权限，SHOW DATABASES不会显示你没有权限的数据库。参见第13.7.7.14节，"SHOW DATABASES语句"。

如果测试数据库存在，尝试访问它。

mysql> **USE test**

Database changed

USE，像QUIT一样，不需要分号。(如果你愿意，你可以用分号来结束这样的语句；这没有什么坏处)。USE 语句在另一个方面也很特别：它必须在单行上给出。

你可以在后面的例子中使用测试数据库(如果你有权限的话)，但你在该数据库中创建的任何东西都可以被其他有权限的人删除。由于这个原因，你也许应该向你的MySQL管理员申请许可，以使用你自己的数据库。假设你想调用你的menagerie。管理员需要执行这样的语句。

mysql> **GRANT ALL ON menagerie.\* TO 'your\_mysql\_name'@'your\_client\_host';**

其中 your\_mysql\_name 是分配给你的 MySQL 用户名， your\_client\_host 是你连接到服务器的主机。

### 3.3.1 创建和选择一个数据库

如果管理员在设置你的权限时为你创建了数据库，你就可以开始使用它。否则，你需要自己创建它。

mysql> **CREATE DATABASE menagerie;**

在Unix下，数据库名称是区分大小写的（与SQL关键字不同），所以你必须总是把你的数据库称为menagerie，而不是Menagerie、MENAGERIE或其他变体。这对表名来说也是如此。(在Windows下，这个限制并不适用，尽管你必须在一个给定的查询中使用相同的大写字母来指代数据库和表。然而，由于各种原因，推荐的最佳做法是始终使用创建数据库时使用的相同字母大小写）。)

注意

如果你得到一个错误，例如 ERROR 1044 (42000): 拒绝用户'micah'@'localhost'对数据库'menagerie'的访问，这意味着你的用户账户没有必要的权限来创建数据库。请与管理员讨论这个问题，或者参见第6.2节，"访问控制和账户管理"。

创建一个数据库并不意味着选择它来使用；你必须明确地这样做。要使menagerie成为当前的数据库，使用这个语句。

mysql> **USE menagerie**

Database changed

你的数据库只需要创建一次，但你必须在每次开始mysql会话时选择它来使用。你可以通过发布USE语句来做到这一点，如例子中所示。或者，当你调用mysql时，你可以在命令行上选择该数据库。只要在你可能需要提供的任何连接参数之后指定它的名称。例如

shell> **mysql -h *host* -u *user* -p menagerie**

Enter password: **\*\*\*\*\*\*\*\***

重要提示

刚才的命令中的menagerie不是你的密码。如果你想在命令行中的-p选项之后提供你的密码，你必须这样做，并且没有中间的空格（例如，作为-ppassword，而不是作为-p password）。然而，不建议将你的密码放在命令行上，因为这样做会使你的密码被登录在你机器上的其他用户窥探到。

注意

你可以在任何时候使用SELECT DATABASE()查看当前选择的数据库。

### 3.3.2 创建一个表

创建数据库是容易的部分，但此时它是空的，正如SHOW TABLES告诉你的。

mysql> **SHOW TABLES;**

Empty set (0.00 sec)

更难的部分是决定你的数据库结构应该是什么：你需要什么表，每个表里应该有什么列。

你想要一个表，其中包含你的每个宠物的记录。这可以被称为宠物表，它至少应该包含每个动物的名字。因为名字本身并不十分有趣，该表应该包含其他信息。例如，如果你家里不止一个人养宠物，你可能想列出每个动物的主人。你可能还想记录一些基本的描述性信息，如物种和性别。

那么年龄呢？这可能是有意义的，但它不是一个好东西，可以存储在数据库中。年龄随着时间的推移而变化，这意味着你必须经常更新你的记录。相反，最好是存储一个固定值，如出生日期。然后，每当你需要年龄时，你可以把它计算为当前日期和出生日期之间的差异。MySQL提供了做日期运算的函数，所以这并不困难。存储出生日期而不是年龄也有其他好处。

* 你可以使用数据库来完成一些任务，例如为即将到来的宠物生日生成提醒信息。如果你认为这种类型的查询有点傻，那么请注意，在商业数据库中，你可能会问同样的问题，以确定你需要在本周或本月向其发送生日问候的客户，以获得计算机辅助的个人感觉。
* 你可以计算与当前日期以外的日期有关的年龄。例如，如果你在数据库中存储了死亡日期，你可以很容易地计算出一只宠物死亡时的年龄。

你可能还能想到其他类型的信息在宠物表中是有用的，但到目前为止所确定的信息已经足够了：名字、主人、物种、性别、出生和死亡。

使用CREATE TABLE语句来指定你的表的布局。

mysql> **CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),**

**species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);**

VARCHAR是名字、所有者和物种列的一个很好的选择，因为这些列值的长度是不同的。这些列定义中的长度不一定都是一样的，也不一定是20。你通常可以选择从1到65535的任何长度，只要是你认为最合理的。如果你做了一个糟糕的选择，后来发现你需要一个更长的字段，MySQL提供一个ALTER TABLE语句。

可以选择几种类型的值来代表动物记录中的性别，例如 "m "和 "f"，或者是 "male "和 "female"。最简单的方法是使用单个字符'm'和'f'。

为出生和死亡列使用DATE数据类型是一个相当明显的选择。

一旦你创建了一个表，SHOW TABLES应该产生一些输出。

mysql> **SHOW TABLES;**

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

| Tables in menagerie |

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

| pet |

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

为了验证你的表是否按照你预期的方式创建，使用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 | |

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

你可以在任何时候使用DESCRIBE，例如，如果你忘记了你表中的列的名称或它们的类型。

关于MySQL数据类型的更多信息，见第11章，数据类型。

### 3.3.3 向表中加载数据

在创建你的表之后，你需要填充它。LOAD DATA和INSERT语句在这方面很有用。

假设你的宠物记录可以被描述为如下所示。(注意，MySQL期望的日期是 "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 | 1979-08-31 | 1995-07-29 |
| **Chirpy** | Gwen | bird | f | 1998-09-11 |  |
| **Whistler** | Gwen | bird |  | 1997-12-09 |  |
| **Slim** | Benny | snake | m | 1996-04-29 |  |

因为你是从一个空表开始的，所以填充它的一个简单方法是创建一个文本文件，其中包含每个动物的一行，然后用一个语句将文件的内容加载到表中。

你可以创建一个文本文件pet.txt，每行包含一条记录，数值用制表符分隔，并按照CREATE TABLE语句中列的顺序给出。对于缺失的值（比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。要在你的文本文件中表示这些，请使用 \N（反斜杠，大写N）。例如，鸟类Whistler的记录看起来是这样的（其中数值之间的空白是一个单一的制表符）。

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

要将文本文件pet.txt加载到pet表中，使用这个语句。

mysql> **LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pe**

如果你在Windows上用一个使用\r\n作为行结束符的编辑器创建该文件，你应该使用这个语句来代替。

mysql> **LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet**

**LINES TERMINATED BY '\r\n';**

(在运行macOS的苹果机器上，你可能想使用LINES TERMINATED BY '\r')。

如果你愿意，你可以在LOAD DATA语句中明确指定列值分隔符和行尾标记，但默认的是TAB和换行符。这些足以让语句正确读取文件pet.txt。

如果语句失败，可能是你的MySQL安装没有默认启用本地文件功能。参见第6.1.6节，"LOAD DATA LOCAL的安全考虑"，以了解如何改变这种情况。

当你想一次添加新记录时，INSERT语句很有用。在其最简单的形式中，你为每一列提供数值，按照列在CREATE TABLE语句中的顺序排列。假设Diane得到一只新的仓鼠，名叫 "Puffball"。你可以使用像这样的INSERT语句添加一条新记录。

mysql> **INSERT INTO pet**

**VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);**

这里的字符串和日期值被指定为带引号的字符串。另外，通过INSERT，你可以直接插入NULL来表示一个缺失的值。你不像在LOAD DATA中那样使用\N。

从这个例子中，你应该可以看到，最初使用几个INSERT语句而不是一个单一的LOAD DATA语句来加载你的记录，会有更多的打字工作。

### 3.3.4 从表中检索信息

SELECT语句是用来从一个表中提取信息的。该语句的一般形式是

SELECT ***what\_to\_select***

FROM ***which\_table***

WHERE ***conditions\_to\_satisfy***;

what\_to\_select表示你想看到的东西。这可以是一个列的列表，或者用\*来表示 "所有的列"。WHERE子句是可选的。如果它存在，conditions\_to\_satisfy指定了一个或多个条件，这些行必须满足这些条件才有资格被检索。

#### 3.3.4.1 选择所有数据

最简单的SELECT形式可以从一个标签中检索所有数据

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 | 1979-08-31 | 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 |

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

这种形式的SELECT使用\*，它是 "选择所有列 "的缩写。如果你想查看整个表，例如，在你刚刚加载了你的初始数据集之后，这很有用。例如，你可能碰巧觉得Bowser的出生日期似乎不大对劲。咨询你的原始血统文件，你发现正确的出生年份应该是1989年，而不是1979年。

至少有两种方法可以解决这个问题。

* 编辑文件pet.txt以纠正错误，然后清空表并使用DELETE和LOAD DATA重新加载它。

mysql> **DELETE FROM pet;**

mysql> **LOAD DATA LOCAL INFILE 'pet.txt' INTO TABLE pet;**

然而，如果你这样做，你也必须重新输入Puffball的记录。

* 用UPDATE语句只修复错误的记录。

mysql> **UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser';**

UPDATE只改变了有问题的记录，不要求你重新加载表。

SELECT \*选择所有列的原则有一个例外。如果一个表包含不可见的列，\*不包括它们。更多信息请参见章节13.1.20.10, "不可见列"。

#### 3.3.4.2 选择特定行

如上一节所示，检索整个表是很容易的。只要在SELECT语句中省略WHERE子句即可。但是，通常你不想看到整个表，特别是当它变得很大的时候。相反，你通常对回答一个特定的问题更感兴趣，在这种情况下，你会对你想要的信息指定一些约束。让我们从它们所回答的关于你的宠物的问题方面来看看一些选择查询。

你可以只从你的表中选择特定的行。例如，如果你想验证你对Bowser的出生日期所做的改变，可以像这样选择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 |

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

输出结果证实，年份被正确记录为1989年，而不是1979年。

字符串比较通常是不区分大小写的，所以你可以指定名称为'bowser'、'BOWSER'等。查询的结果也是一样的。

你可以在任何列上指定条件，而不仅仅是名字。例如，如果你想知道哪些动物是在1998年期间或之后出生的，可以测试出生列。

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 |

你可以结合条件，例如，找到母狗。

mysql> **SELECT \* FROM pet WHERE species = 'dog' AND sex = 'f';**

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

| name | owner | species | sex | birth | death |

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

| Buffy | Harold | dog | f | 1989-05-13 | NULL |

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

前面的查询使用了AND逻辑运算符。还有一个OR运算符

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可以混合使用，尽管AND的优先级比OR高。如果你同时使用这两个运算符，最好使用圆括号来明确指出条件应该如何分组。

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 |

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

#### 3.3.4.3 选择特定列

如果你不想从你的表中看到整个行，只需命名你感兴趣的列，用逗号隔开。例如，如果你想知道你的动物是什么时候出生的，就选择名字和出生列。

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 |

要了解谁拥有宠物，请使用这个查询。

mysql> **SELECT owner FROM pet;**

+--------+

| owner |

+--------+

| Harold |

| Gwen |

| Harold |

| Benny |

| Diane |

| Gwen |

| Gwen |

| Benny |

| Diane |

+--------+

请注意，该查询只是从每条记录中检索所有者列，而其中有些记录出现了不止一次。为了尽量减少输出，可以通过添加关键字DISTINCT，只检索每条唯一的输出记录一次。

mysql> **SELECT DISTINCT owner FROM pet;**

+--------+

| owner |

+--------+

| Benny |

| Diane |

| Gwen |

| Harold |

+--------+

你可以使用WHERE子句来结合行选择和列选择。例如，如果只想获得狗和猫的出生日期，可以使用这个查询

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 |

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

#### 3.3.4.4 对行进行排序

你可能已经注意到在前面的例子中，结果行的显示没有特定的顺序。当行以某种有意义的方式进行排序时，通常会更容易检查查询的输出。要对一个结果进行排序，可以使用ORDER BY子句。

下面是动物的生日，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 |

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

在字符类型的列上，排序与所有其他比较操作一样，通常以不区分大小写的方式进行。这意味着，对于除了大小写之外完全相同的列，其顺序是不确定的。你可以通过使用BINARY来强制对一个列进行大小写排序，就像这样。ORDER BY BINARY col\_name。

默认的排序顺序是升序，最小的值在前。要以反向（降序）排序，请在你要排序的列的名称上添加DESC关键字。

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 |

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

你可以对多列进行排序，也可以对不同的列按不同的方向进行排序。例如，要按动物类型以升序排序，然后按动物类型中的出生日期以降序排序（最年轻的动物在前），请使用以下查询。

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 |

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

DESC关键字只适用于紧接在它前面的列名（出生）；它不影响物种列的排序顺序。

#### 3.3.4.5 日期计算

MySQL提供了几个函数，你可以用来对日期进行计算，例如，计算年龄或提取日期的一部分。

要确定你的每个宠物的年龄是多少岁，可以使用TIMESTAMPDIFF()函数。它的参数是你希望结果表达的单位，以及要取差的两个日期。下面的查询显示了每个宠物的出生日期、当前日期和年龄。一个别名（age）被用来使最后的输出列标签更有意义。

mysql> **SELECT name, birth, CURDATE(),**

**TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age**

**FROM pet;**

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

| name | birth | CURDATE() | age |

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

| 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 |

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

这个查询是有效的，但是如果这些行以某种顺序呈现的话，可以更容易地扫描出结果。这可以通过添加一个ORDER BY name子句来实现，以按名字对输出进行排序。

mysql> **SELECT name, birth, CURDATE(),**

**TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age**

**FROM pet ORDER BY name;**

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

| name | birth | CURDATE() | age |

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

| 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 |

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

要按年龄而不是姓名对输出进行排序，只需使用不同的ORDER BY子句。

mysql> **SELECT name, birth, CURDATE(),**

**TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age**

**FROM pet ORDER BY age;**

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

| name | birth | CURDATE() | age |

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

| 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 |

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

一个类似的查询可以用来确定已经死亡的动物的死亡年龄。你可以通过检查死亡值是否为NULL来确定这些动物。然后，对于那些非NULL值的动物，计算死亡值和出生值之间的差异。

mysql> **SELECT name, birth, death,**

**TIMESTAMPDIFF(YEAR,birth,death) AS age**

**FROM pet WHERE death IS NOT NULL ORDER BY age;**

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

| name | birth | death | age |

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

| Bowser | 1989-08-31 | 1995-07-29 | 5 |

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

这个查询使用的是death IS NOT NULL，而不是death <> NULL，因为NULL是一个特殊的值，不能用通常的比较运算符来比较。这一点将在后面讨论。参见第3.3.4.6节，"处理NULL值"。

如果你想知道哪些动物在下个月有生日呢？对于这种类型的计算，年和日是不相关的；你只是想提取出生列的月份部分。MySQL提供了几个函数用于提取日期的一部分，如YEAR()、MONTH()和DAYOFMONTH()。MONTH()是这里合适的函数。为了看看它是如何工作的，运行一个简单的查询，显示出生和MONTH(出生)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 |

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

寻找生日在下个月的动物也很简单。假设当前的月份是四月。那么月份值是4，你可以这样寻找在5月（第5个月）出生的动物。

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

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

| name | birth |

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

| Buffy | 1989-05-13 |

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

如果当前的月份是12月，就会出现一个小的复杂情况。你不能仅仅在月号（12）上加一，然后寻找在13月出生的动物，因为没有这样的月份。相反，你要寻找出生在一月（第1个月）的动物。

你可以编写查询，这样无论当前的月份是什么，它都可以工作，这样你就不必使用某个特定月份的数字。DATE\_ADD()可以使你在一个给定的日期上增加一个时间间隔。如果你在CURDATE()的值中加入一个月份，然后用MONTH()提取月份部分，其结果是产生一个可以寻找生日的月份。

mysql> **SELECT name, birth FROM pet**

**WHERE MONTH(birth) = MONTH(DATE\_ADD(CURDATE(),INTERVAL 1 MONTH));**

完成同样任务的另一种方法是，在使用模数函数（MOD）将月份值包裹为0（如果当前是12）之后，加1来获得当前月份之后的下一个月。

mysql> **SELECT name, birth FROM pet**

**WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;**

MONTH()返回1到12之间的一个数字。而MOD(something,12)返回0到11之间的数字。所以加法必须在MOD()之后，否则我们会从11月（11）到1月（1）。

如果一个计算使用了无效的日期，计算就会失败并产生警告。

mysql> **SELECT '2018-10-31' + INTERVAL 1 DAY;**

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

| '2018-10-31' + INTERVAL 1 DAY |

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

| 2018-11-01 |

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

mysql> **SELECT '2018-10-32' + INTERVAL 1 DAY;**

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

| '2018-10-32' + INTERVAL 1 DAY |

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

| NULL |

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

mysql> **SHOW WARNINGS;**

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

| Level | Code | Message |

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

| Warning | 1292 | Incorrect datetime value: '2018-10-32' |

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

#### 3.3.4.6 与NULL值一起工作

在你习惯它之前，NULL值可能会令人惊讶。从概念上讲，NULL意味着 "一个缺失的未知值"，它的处理方式与其他值有些不同。

为了测试NULL，可以使用IS NULL和IS NOT NULL操作符，如图所示。

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

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

| 1 IS NULL | 1 IS NOT NULL |

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

| 0 | 1 |

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

你不能使用算术比较运算符，如=，<，或<>来测试NULL。为了证明这一点，请尝试下面的查询。

mysql> **SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;**

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

| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |

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

| NULL | NULL | NULL | NULL |

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

因为与NULL进行任何算术比较的结果也是NULL，你不能从这种比较中获得任何有意义的结果。

在MySQL中，0或NULL意味着假，其他的都意味着真。布尔运算的默认真值是1。

对NULL的这种特殊处理是为什么在上一节中，有必要使用死亡不是NULL而不是死亡<>NULL来确定哪些动物已经不在世。

在GROUP BY中，两个NULL值被认为是相等的。

当做ORDER BY时，如果你做ORDER BY ...，NULL值会首先出现，如果你做ORDER BY ...，NULL值会最后出现。ASC，最后一个是ORDER BY ... DESC。

在处理NULL时，一个常见的错误是认为不可能在定义为NOT NULL的列中插入一个零或空字符串，但事实并非如此。这些实际上是值，而NULL意味着 "没有值"。你可以通过使用IS [NOT] NULL来测试这一点，如图所示。

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 |

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

因此，完全可以在NOT NULL列中插入一个零或空的字符串，因为这些实际上是NOT NULL。参见B.3.4.3节，"NULL值的问题"。

#### 3.3.4.7 模式匹配

MySQL提供标准的SQL模式匹配，以及一种基于扩展的正则表达式的模式匹配，类似于Unix工具（如vi、grep和sed）使用的模式。

SQL模式匹配使你能够使用\_来匹配任何单个字符，使用%来匹配任意数量的字符（包括零字符）。在MySQL中，SQL模式默认是不区分大小写的。这里显示了一些例子。当你使用SQL模式时，不要使用=或<>。使用LIKE或NOT LIKE比较运算符来代替。

要查找以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 |

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

要找到以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 |

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

要找到含有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 |

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

要查找正好包含五个字符的名称，请使用\_模式字符的五个实例。

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 |

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

MySQL提供的另一种类型的模式匹配使用扩展的正则表达式。当你测试这种类型的模式的匹配时，使用REGEXP\_LIKE()函数（或REGEXP或RLIKE操作符，它们是REGEXP\_LIKE()的同义词）。

下面的列表描述了扩展正则表达式的一些特征。

* .匹配任何单个字符。
* 字符类 [...] 匹配括号内的任何字符。例如，[abc] 匹配a、b或c。要命名一个字符范围，使用破折号。[a-z] 匹配任何字母，而[0-9] 匹配任何数字。
* \*匹配前面的事物的零或多个实例。例如，x\*匹配任何数量的x字符，[0-9]\*匹配任何数量的数字，而.\*匹配任何数量的东西
* 如果一个正则表达式模式匹配在被测试值的任何地方匹配，那么该模式就会成功。这与LIKE模式匹配不同，后者只在模式匹配整个值时才会成功）。
* 要锚定一个模式，使其必须匹配被测试值的开头或结尾，可以在模式的开头使用^，在结尾使用$。

为了演示扩展的正则表达式是如何工作的，前面显示的LIKE查询在这里被改写为使用REGEXP\_LIKE()。

要查找以b开头的名字，使用^来匹配名字的开头

mysql> **SELECT \* FROM pet WHERE REGEXP\_LIKE(name, '^b');**

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

| name | owner | species | sex | birth | death |

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

| Buffy | Harold | dog | f | 1989-05-13 | NULL |

| Bowser | Diane | dog | m | 1979-08-31 | 1995-07-29 |

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

要强制正则表达式比较区分大小写，请使用区分大小写的整理方式，或使用BINARY关键字使其中一个字符串成为二进制字符串，或指定c匹配控制字符。这些查询中的每一个都只匹配名字开头的小写b。

SELECT \* FROM pet WHERE REGEXP\_LIKE(name, '^b' COLLATE utf8mb4\_0900\_as\_cs);

SELECT \* FROM pet WHERE REGEXP\_LIKE(name, BINARY '^b');

SELECT \* FROM pet WHERE REGEXP\_LIKE(name, '^b', 'c');

要找到以fy结尾的名字，用$来匹配名字的结尾。

mysql> **SELECT \* FROM pet WHERE REGEXP\_LIKE(name, 'fy$');**

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

| name | owner | species | sex | birth | death |

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

| Fluffy | Harold | cat | f | 1993-02-04 | NULL |

| Buffy | Harold | dog | f | 1989-05-13 | NULL |

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

要找到含有w的名字，请使用这个查询。

mysql> **SELECT \* FROM pet WHERE REGEXP\_LIKE(name, '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 |

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

因为正则表达式模式在值的任何地方出现都会匹配，所以在前面的查询中，没有必要像SQL模式那样，在模式的两边加上通配符来让它匹配整个值。

要找到正好包含五个字符的名字，使用^和$来匹配名字的开头和结尾，以及中间的五个.实例。

mysql> **SELECT \* FROM pet WHERE REGEXP\_LIKE(name, '^.....$');**

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

| name | owner | species | sex | birth | death |

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

| Claws | Gwen | cat | m | 1994-03-17 | NULL |

| Buffy | Harold | dog | f | 1989-05-13 | NULL |

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

你也可以用{n}（"重复-n次"）操作符来写前面的查询

mysql> **SELECT \* FROM pet WHERE REGEXP\_LIKE(name, '^.{5}$');**

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

| name | owner | species | sex | birth | death |

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

| Claws | Gwen | cat | m | 1994-03-17 | NULL |

| Buffy | Harold | dog | f | 1989-05-13 | NULL |

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

关于正则表达式语法的更多信息，请参见第12.8.2节，"正则表达式"。

#### 3.3.4.8 计算行数

数据库经常被用来回答这样一个问题："某种类型的数据在表中出现的频率如何？" 例如，你可能想知道你有多少只宠物，或者每个主人有多少只宠物，或者你可能想对你的动物进行各种普查操作。

计算你拥有的动物总数与 "宠物表有多少行？"是同一个问题，因为每个宠物有一条记录。COUNT(\*)计算行数，所以计算动物的查询看起来像这样。

mysql> **SELECT COUNT(\*) FROM pet;**

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

| COUNT(\*) |

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

| 9 |

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

刚才，你检索了拥有宠物的人的名字。如果你想知道每个主人有多少只宠物，你可以使用COUNT()。

mysql> **SELECT owner, COUNT(\*) FROM pet GROUP BY owner;**

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

| owner | COUNT(\*) |

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

| Benny | 2 |

| Diane | 2 |

| Gwen | 3 |

| Harold | 2 |

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

前面的查询使用GROUP BY来分组每个所有者的所有记录。结合GROUP BY使用COUNT()对于在各种分组下描述你的数据特征是非常有用的。下面的例子显示了执行动物普查操作的不同方式。

每个物种的动物数量。

mysql> **SELECT species, COUNT(\*) FROM pet GROUP BY species;**

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

| species | COUNT(\*) |

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

| bird | 2 |

| cat | 2 |

| dog | 3 |

| hamster | 1 |

| snake | 1 |

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

每个性别的动物数量。

mysql> **SELECT sex, COUNT(\*) FROM pet GROUP BY sex;**

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

| sex | COUNT(\*) |

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

| NULL | 1 |

| f | 4 |

| m | 4 |

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

在此输出中，NULL表示性别未知）。

每个物种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，你不需要检索整个表。例如，前面的查询，当只对狗和猫进行查询时，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 |

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

或者，如果你只想知道性别已知的动物的每一性别的数量。

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 |

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

如果你在COUNT()值之外还命名了要选择的列，那么应该有一个GROUP BY子句来命名这些相同的列。否则会出现以下情况。

* 如果ONLY\_FULL\_GROUP\_BY SQL模式被启用，会发生错误。

mysql> **SET sql\_mode = 'ONLY\_FULL\_GROUP\_BY';**

Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT owner, COUNT(\*) FROM pet;**

ERROR 1140 (42000): In aggregated query without GROUP BY, expression

#1 of SELECT list contains nonaggregated column 'menagerie.pet.owner';

this is incompatible with sql\_mode=only\_full\_group\_by

* 如果没有启用ONLY\_FULL\_GROUP\_BY，那么在处理查询时，会把所有的记录作为一个单一的组，但是为每个命名的列选择的值是不确定的。服务器可以自由地从任何行中选择值。

mysql> **SET sql\_mode = '';**

Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT owner, COUNT(\*) FROM pet;**

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

| owner | COUNT(\*) |

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

| Harold | 8 |

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

1 row in set (0.00 sec)

参见第12.20.3节，"MySQL对GROUP BY的处理"。关于COUNT(expr)行为和相关优化的信息，请参见第12.20.1节 "聚合函数描述"。

#### 3.3.4.9 使用多于一个表

宠物表记录了你有哪些宠物。如果你想记录关于它们的其他信息，比如它们生活中的事件，如去看兽医或什么时候出生的孩子，你需要另一个表。这个表应该是什么样子的？它需要包含以下信息。

* 宠物的名字，以便你知道每个事件与哪个动物有关。
* 一个日期，以便你知道事件发生的时间。
* 一个描述该事件的字段。
* 一个事件类型字段，如果你希望能够对事件进行分类。

考虑到这些因素，事件表的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));**

与宠物表一样，最简单的方法是通过创建一个包含以下信息的带制表符的文本文件来加载初始记录。

|  |  |  |  |
| --- | --- | --- | --- |
| **name** | **date** | **type** | **remark** |
| **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 |

像这样加载记录。

mysql> **LOAD DATA LOCAL INFILE 'event.txt' INTO TABLE event;**

基于你从对宠物表的查询中学到的东西，你应该能够对事件表中的记录进行检索；原理是一样的。但是，什么时候事件表本身不足以回答你可能提出的问题？

假设你想找出每只宠物产仔的年龄。我们前面看到如何从两个日期计算年龄。母亲的产仔日期在事件表中，但要计算她在该日期的年龄，你需要她的出生日期，该日期存储在宠物表中。这意味着该查询需要两个表。

mysql> **SELECT pet.name,**

**TIMESTAMPDIFF(YEAR,birth,date) AS age,**

**remark**

**FROM pet INNER JOIN event**

**ON pet.name = event.name**

**WHERE event.type = 'litter';**

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

| name | age | remark |

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

| Fluffy | 2 | 4 kittens, 3 female, 1 male |

| Buffy | 4 | 5 puppies, 2 female, 3 male |

| Buffy | 5 | 3 puppies, 3 female |

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

关于这个查询，有几件事需要注意。

* FROM子句连接了两个表，因为该查询需要从这两个表中提取信息。
* 当结合（连接）来自多个表的信息时，你需要指定如何将一个表中的记录与另一个表中的记录进行匹配。这很容易，因为它们都有一个名字列。该查询使用一个ON子句，根据名字的值来匹配两个表中的记录。该查询使用一个INNER JOIN来合并两个表。当且仅当两个表都满足ON子句中指定的条件时，INNER JOIN允许任何一个表的记录出现在结果中。在这个例子中，ON子句指定了宠物表中的名字列必须与事件表中的名字列相匹配。如果一个名字出现在一个表中，但是没有出现在另一个表中，那么该行就不会出现在结果中，因为ON子句中的条件失败了。
* 因为名字列同时出现在两个表中，所以在引用该列时，你必须具体说明是指哪个表。这可以通过在列名前加上表名来实现。

你不需要有两个不同的表来执行一个连接。有时，如果你想把一个表中的记录与同一表中的其他记录进行比较，那么把一个表与它本身连接起来是很有用的。例如，为了在你的宠物中寻找繁殖对，你可以将宠物表与它本身连接起来，以产生同类物种的活体雄性和雌性的候选对。

mysql> **SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species**

**FROM pet AS p1 INNER JOIN pet AS p2**

**ON p1.species = p2.species**

**AND p1.sex = 'f' AND p1.death IS NULL**

**AND p2.sex = 'm' AND p2.death IS NULL;**

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

| name | sex | name | sex | species |

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

| Fluffy | f | Claws | m | cat |

| Buffy | f | Fang | m | dog |

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

在这个查询中，我们指定了表名的别名来引用列，并直接保持每列引用与表的哪个实例相关。

## 3.4 获取关于数据库和表的信息

如果你忘记了一个数据库或表的名称，或者一个给定的表的结构是什么（例如，它的列被称为什么），怎么办？MySQL通过提供有关它支持的数据库和表的信息的几个语句来解决这个问题。

你以前看过SHOW DATABASES，它列出了服务器管理的数据库。要想知道当前选择的是哪个数据库，请使用DATABASE()函数。

mysql> **SELECT DATABASE();**

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

| DATABASE() |

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

| menagerie |

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

如果你还没有选择任何数据库，结果是NULL。

要想知道默认数据库包含哪些表（例如，当你不确定某个表的名称时），请使用这个语句。

mysql> **SHOW TABLES;**

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

| Tables\_in\_menagerie |

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

| event |

| pet |

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

这个语句产生的输出中的列名总是Tables\_in\_db\_name，其中db\_name是数据库的名称。更多信息请参见章节13.7.7.39, "SHOW TABLES 语句"。

如果你想了解一个表的结构，DESCRIBE语句是很有用的；它可以显示一个表的每一列的信息。

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表示列的名称，Type是该列的数据类型，NULL表示该列是否可以包含NULL值，Key表示该列是否有索引，Default指定该列的默认值。Extra显示关于列的特殊信息。如果一个列是用AUTO\_INCREMENT选项创建的，其值是自动递增而不是空的。

DESC是DESCRIBE的缩写形式。更多信息请参见第13.8.1节，"DESCRIBE语句"。

你可以使用SHOW CREATE TABLE语句获得创建一个现有表所需的CREATE TABLE语句。参见第13.7.7.10节，"SHOW CREATE TABLE语句"。

如果你在一个表上有索引，SHOW INDEX FROM tbl\_name可以产生关于它们的信息。关于这个语句的更多信息，请参见第13.7.7.22节，"SHOW INDEX语句"。

## 3.5 在批处理模式下使用mysql

在前面的章节中，你以交互方式使用mysql，输入语句并查看结果。你也可以在批处理模式下运行mysql。要做到这一点，把你想运行的语句放在一个文件中，然后告诉mysql从文件中读取其输入。

shell> **mysql < *batch-file***

如果你在Windows下运行mysql，并且文件中有一些导致问题的特殊字符，你可以这样做。

C:\> **mysql -e "source *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，然后执行该脚本。

如果你想让脚本继续下去，即使其中的一些语句产生错误，你应该使用--force命令行选项。

为什么要使用脚本？这里有几个原因。

* 如果你重复运行一个查询（例如，每天或每周），把它变成一个脚本，就可以避免每次执行时都重新输入。
* 你可以通过复制和编辑脚本文件，从现有的类似查询中生成新的查询。
* 批量模式在你开发查询时也很有用，特别是对于多行语句或多语句序列。如果你犯了一个错误，你不需要重新输入所有的内容。只要编辑你的脚本以纠正错误，然后告诉mysql再次执行它。
* 如果你有一个产生大量输出的查询，你可以通过一个寻呼机运行输出，而不是看着它从屏幕顶部滚动。

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

* 你可以在一个文件中捕捉输出，以便进一步处理。

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

* 你可以把你的脚本分发给其他人，这样他们也可以运行这些语句。
* 有些情况不允许交互式使用，例如，当你从一个cron job中运行查询时。在这种情况下，你必须使用批处理模式。

当你以批处理模式运行mysql时，默认的输出格式与你交互式使用时不同（更简洁）。例如，当mysql以交互方式运行时，SELECT DISTINCT species FROM pet的输出看起来像这样。

+---------+

| species |

+---------+

| bird |

| cat |

| dog |

| hamster |

| snake |

+---------+

在批处理模式下，输出结果看起来像这样，而不是。

species

bird

cat

dog

hamster

snake

如果你想在批处理模式下获得交互式输出格式，使用mysql -t。要将执行的语句回显到输出，使用mysql -v。

你也可以通过使用source命令或\.命令从mysql提示符中使用脚本。

mysql> **source *filename*;**

mysql> **\. *filename***

更多信息见第4.5.1.5节 "从文本文件执行SQL语句"。

## 3.6 常见查询的例子

下面是如何解决MySQL的一些常见问题的例子。

一些例子使用表shop来保存某些交易者（经销商）的每件物品（物品编号）的价格。假设每个商人对每件物品有一个固定的价格，那么（物品，经销商）就是记录的主键。

启动命令行工具mysql并选择一个数据库。

shell> **mysql *your-database-name***

要创建和填充示例表，使用这些语句。

CREATE TABLE shop (

article INT UNSIGNED DEFAULT '0000' NOT NULL,

dealer CHAR(20) DEFAULT '' NOT NULL,

price DECIMAL(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);

发出这些语句后，表应该有以下内容。

SELECT \* FROM shop ORDER BY article;

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

| article | dealer | price |

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

| 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 |

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

### 3.6.1 一列的最高值

"最高的项目编号是多少？"

SELECT MAX(article) AS article FROM shop;

+---------+

| article |

+---------+

| 4 |

+---------+

### 3.6.2 持有某列最大值的行

任务。找到最贵的物品的数量、经销商和价格。

这很容易通过一个子查询完成。

SELECT article, dealer, price

FROM shop

WHERE price=(SELECT MAX(price) FROM shop);

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

| article | dealer | price |

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

| 0004 | D | 19.95 |

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

其他的解决方案是使用LEFT JOIN或按价格降序排序所有的记录，并使用MySQL特有的LIMIT子句只获得第一行。

SELECT s1.article, s1.dealer, s1.price

FROM shop s1

LEFT JOIN shop s2 ON s1.price < s2.price

WHERE s2.article IS NULL;

SELECT article, dealer, price

FROM shop

ORDER BY price DESC

LIMIT 1;

注意

如果有几件最贵的物品，每件的价格都是19.95，那么LIMIT方案将只显示其中一件。

### 3.6.3 每组列的最大值

任务。找到每篇文章的最高价格。

SELECT article, MAX(price) AS price

FROM shop

GROUP BY article

ORDER BY article;

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

| article | price |

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

| 0001 | 3.99 |

| 0002 | 10.99 |

| 0003 | 1.69 |

| 0004 | 19.95 |

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

### 3.6.4 持有某一列的组别最大值的行

任务。对于每件物品，找出价格最贵的一个或多个经销商。

这个问题可以用一个类似这样的子查询来解决。

SELECT article, dealer, price

FROM shop s1

WHERE price=(SELECT MAX(s2.price)

FROM shop s2

WHERE s1.article = s2.article)

ORDER BY article;

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

| article | dealer | price |

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

| 0001 | B | 3.99 |

| 0002 | A | 10.99 |

| 0003 | C | 1.69 |

| 0004 | D | 19.95 |

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

前面的例子使用了一个相关的子查询，这可能是低效的（参见章节13.2.11.7，"相关子查询"）。解决这个问题的其他可能性是在FROM子句中使用一个不相关的子查询，一个LEFT JOIN，或者一个带有窗口函数的公共表表达。

不相关的子查询。

SELECT s1.article, dealer, s1.price

FROM shop s1

JOIN (

SELECT article, MAX(price) AS price

FROM shop

GROUP BY article) AS s2

ON s1.article = s2.article AND s1.price = s2.price

ORDER BY article;

**LEFT JOIN:**

SELECT s1.article, s1.dealer, s1.price

FROM shop s1

LEFT JOIN shop s2 ON s1.article = s2.article AND s1.price < s2.price

WHERE s2.article IS NULL

ORDER BY s1.article;

LEFT JOIN的工作原理是：当s1.price达到最大值时，没有s2.price的值更大，因此对应的s2.article的值为NULL。参见章节13.2.10.2, "JOIN条款"。

带有窗口函数的普通表表达式。

WITH s1 AS (

SELECT article, dealer, price,

RANK() OVER (PARTITION BY article

ORDER BY price DESC

) AS `Rank`

FROM shop

)

SELECT article, dealer, price

FROM s1

WHERE `Rank` = 1

ORDER BY article;

### 3.6.5 使用用户定义的变量

你可以使用MySQL用户变量来记住结果，而不必将它们存储在客户端的临时变量中。参见第9.4节，"用户定义的变量"）。

例如，要找到具有最高和最低价格的文章，你可以这样做。

mysql> **SELECT @min\_price:=MIN(price),@max\_price:=MAX(price) FROM shop;**

mysql> **SELECT \* FROM shop WHERE price=@min\_price OR price=@max\_price;**

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

| article | dealer | price |

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

| 0003 | D | 1.25 |

| 0004 | D | 19.95 |

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

注意

也可以在用户变量中存储数据库对象的名称，如表或列，然后在SQL语句中使用这个变量；但是，这需要使用预处理语句。更多信息请参见章节13.5, "预处理语句"。

### 3.6.6 使用外键

在MySQL中，InnoDB表支持外键约束的检查。参见第15章，InnoDB存储引擎，以及第1.7.2.3节，"FOREIGN KEY约束的差异"。

外键约束并不是仅仅为了连接两个表而需要的。对于InnoDB以外的存储引擎，在定义列的时候可以使用REFERENCES tbl\_name(col\_name)子句，这个子句没有实际作用，只是作为一个备忘或注释，告诉你当前定义的列是为了引用另一个表中的列。在使用这种语法时，认识到以下几点是极其重要的。

* MySQL不执行任何类型的检查以确保col\_name实际存在于tbl\_name中（甚至tbl\_name本身存在）。
* MySQL不对tbl\_name执行任何类型的操作，例如在你定义的表中的行上采取的行动中删除行；换句话说，这个语法没有引起任何ON DELETE或ON UPDATE行为。(尽管你可以在REFERENCES子句中写一个ON DELETE或ON UPDATE子句，但它也会被忽略。)

这个语法创建了一个列；它并没有创建任何类型的索引或键。

你可以使用这样创建的列作为一个连接列，如下面所示。

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 INNER JOIN shirt s

ON s.owner = p.id

WHERE p.name LIKE 'Lilliana%'

AND s.color <> 'white';

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

| id | style | color | owner |

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

| 4 | dress | orange | 2 |

| 5 | polo | red | 2 |

| 6 | dress | blue | 2 |

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

当以这种方式使用时，REFERENCES子句不会显示在SHOW CREATE TABLE或DESCRIBE的输出中。

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=utf8mb4

以这种方式使用REFERENCES作为列定义中的注释或 "提醒"，对MyISAM表是有效的。

### 3.6.7 在两个键上进行搜索

使用单键的OR是很好的优化，和AND的处理一样。

一个棘手的情况是，在两个不同的键上搜索，并结合OR。

SELECT field1\_index, field2\_index FROM test\_table

WHERE field1\_index = '1' OR field2\_index = '1'

。

这种情况被优化。参见章节8.2.1.3, "索引合并优化"。

你也可以通过使用合并两个独立的SELECT语句的输出的UNION来有效地解决这个问题。参见章节13.2.10.3, "UNION子句"。

每个SELECT只搜索一个键，可以进行优化。

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 计算每天的访问量

下面的例子显示了如何使用位组函数来计算用户每月访问一个网页的天数。

CREATE TABLE t1 (year YEAR, month INT UNSIGNED,

day INT UNSIGNED);

INSERT INTO t1 VALUES(2000,1,1),(2000,1,20),(2000,1,30),(2000,2,2),

(2000,2,23),(2000,2,23);

该示例表包含年-月-日的数值，代表用户对该页面的访问。为了确定这些访问在每个月有多少个不同的日子发生，请使用这个查询。

SELECT year,month,BIT\_COUNT(BIT\_OR(1<<day)) AS days FROM t1

GROUP BY year,month;

这将返回。

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

| year | month | days |

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

| 2000 | 1 | 3 |

| 2000 | 2 | 2 |

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

该查询计算出每个年/月的组合有多少个不同的日子出现在表中，并自动删除重复的条目。

### 3.6.9 使用AUTO\_INCREMENT

AUTO\_INCREMENT属性可以用来为新的记录生成一个唯一的标识。

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;

SELECT \* FROM animals;

其中返回。

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

| id | name |

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

| 1 | dog |

| 2 | cat |

| 3 | penguin |

| 4 | lax |

| 5 | whale |

| 6 | ostrich |

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

没有为AUTO\_INCREMENT列指定值，所以MySQL自动分配序列号。你也可以显式地给该列分配0来生成序列号，除非启用NO\_AUTO\_VALUE\_ON\_ZERO SQL模式。例如

INSERT INTO animals (id,name) VALUES(0,'groundhog');

如果列被声明为NOT NULL，也可以将NULL赋值给该列，以生成序列号。例如。

INSERT INTO animals (id,name) VALUES(NULL,'squirrel');

当你在AUTO\_INCREMENT列中插入任何其他值时，该列被设置为该值，并且序列被重置，以便下一个自动生成的值从最大的列值开始依次出现。例如。

**INSERT INTO animals (id,name) VALUES(100,'rabbit');**

**INSERT INTO animals (id,name) VALUES(NULL,'mouse');**

**SELECT \* FROM animals;**

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

| id | name |

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

| 1 | dog |

| 2 | cat |

| 3 | penguin |

| 4 | lax |

| 5 | whale |

| 6 | ostrich |

| 7 | groundhog |

| 8 | squirrel |

| 100 | rabbit |

| 101 | mouse |

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

更新现有的AUTO\_INCREMENT列值也会重置AUTO\_INCREMENT序列。

你可以用LAST\_INSERT\_ID()SQL函数或mysql\_insert\_id()C API函数检索最近的自动生成的AUTO\_INCREMENT值。这些函数是特定于连接的，所以它们的返回值不会受到另一个也在执行插入的连接的影响。

为AUTO\_INCREMENT列使用最小的整数数据类型，其大小足以容纳你需要的最大序列值。当该列达到数据类型的上限时，下一次生成序列号的尝试会失败。如果可能的话，使用UNSIGNED属性以允许更大的范围。例如，如果你使用TINYINT，允许的最大序列号是127。对于TINYINT UNSIGNED，最大是255。关于所有整数类型的范围，请参阅第11.1.2节 "整数类型（精确值）--INTEGER、INT、SMALLINT、TINYINT、MEDIUMINT、BIGINT"。

注意

对于多行插入，LAST\_INSERT\_ID()和mysql\_insert\_id()实际上返回插入行中第一行的AUTO\_INCREMENT键。这使得多行插入可以在复制设置中的其他服务器上正确重现。

要以1以外的AUTO\_INCREMENT值开始，用CREATE TABLE或ALTER TABLE设置该值，像这样。

mysql> **ALTER TABLE tbl AUTO\_INCREMENT = 100;**

**InnoDB注意事项**

关于InnoDB特定的AUTO\_INCREMENT用法的信息，请参见15.6.1.6节 "InnoDB中的AUTO\_INCREMENT处理"。

**MyISAM注意事项**

对于MyISAM表，你可以在一个多列索引中的二级列上指定AUTO\_INCREMENT。在这种情况下，AUTO\_INCREMENT列的生成值被计算为MAX(auto\_increment\_column) + 1 WHERE prefix=given-preix。当你想把数据放入有序的组时，这很有用。

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)

) ENGINE=MyISAM;

INSERT INTO animals (grp,name) VALUES

('mammal','dog'),('mammal','cat'),

('bird','penguin'),('fish','lax'),('mammal','whale'),

('bird','ostrich');

SELECT \* FROM animals ORDER BY grp,id;

其中返回。

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

| grp | id | name |

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

| fish | 1 | lax |

| mammal | 1 | dog |

| mammal | 2 | cat |

| mammal | 3 | whale |

| bird | 1 | penguin |

| bird | 2 | ostrich |

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

在这种情况下（当AUTO\_INCREMENT列是多列索引的一部分时），如果你删除任何一组中具有最大AUTO\_INCREMENT值的行，AUTO\_INCREMENT值会被重新使用。这种情况甚至发生在MyISAM表中，对于MyISAM表来说，AUTO\_INCREMENT值通常不会被重复使用。

如果AUTO\_INCREMENT列是多个索引的一部分，MySQL使用以AUTO\_INCREMENT列开始的索引生成序列值，如果有一个的话。例如，如果动物表包含索引PRIMARY KEY（grp，id）和INDEX（id），MySQL在生成序列值时将忽略PRIMARY KEY。结果是，该表将包含一个单一的序列，而不是每个grp值的序列。

**进一步阅读**

关于AUTO\_INCREMENT的更多信息可以在这里找到。

* 如何为一个列分配AUTO\_INCREMENT属性。第13.1.20节，"CREATE TABLE语句"，以及第13.1.9节，"ALTER TABLE语句"。
* 在NO\_AUTO\_VALUE\_ON\_ZERO SQL模式下，AUTO\_INCREMENT是如何表现的。第5.1.11节，"服务器SQL模式"。
* 如何使用LAST\_INSERT\_ID()函数来查找包含最新AUTO\_INCREMENT值的记录。第12.16节，"信息函数"。
* 设置要使用的AUTO\_INCREMENT值。第5.1.8节，"服务器系统变量"。
* 第15.6.1.6节，"InnoDB中的AUTO\_INCREMENT处理"
* AUTO\_INCREMENT和复制。第17.5.1.1节，"复制和AUTO\_INCREMENT"。
* 与AUTO\_INCREMENT有关的服务器系统变量（auto\_increment\_increment和auto\_increment\_offset），可以用于复制。第5.1.8节，"服务器系统变量"

## 3.7 在Apache中使用MySQL

有一些程序可以让你从MySQL数据库中验证你的用户，也可以让你把你的日志文件写到MySQL表中。

你可以改变Apache的日志格式，使其容易被MySQL读取，方法是将以下内容放入Apache的配置文件中。

LogFormat \

"\"%h\",%{%Y%m%d%H%M%S}t,%>s,\"%b\",\"%{Content-Type}o\", \

\"%U\",\"%{Referer}i\",\"%{User-Agent}i\""

要把这种格式的日志文件加载到MySQL中，你可以使用类似这样的语句。

LOAD DATA INFILE '***/local/access\_log***' INTO TABLE ***tbl\_name***

FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'

命名的表应该被创建为具有与LogFormat行写到日志文件中的列相对应的列。