Table of Contents

MySQL 5.7 参考手册

这是一个MySQL 5.7 的中文参考手册，翻译自官方文档，需要更详细、准确的信息请查阅官方文档。

此手册目前主要用于本人学习，翻译部分为自己学习的地方，所以章节可能会打乱，但后续会抽空陆续翻译，如有不周到地方，请海涵，请指教，谢谢！

2016/7/29 创建此书，完成“连接和断开服务器”，“查询”部分

2016/7/30 添加“创建和使用数据库”，完成“教程”部分。

2016/8/2 添加“SQL语句语法”

教程

连接和断开服务器

启动mysql服务：

```
sudo service mysql start
```

停止mysql服务：

```
sudo service mysql stop
```

要连接到服务器，我们通常需要提供MySQL的用户名来触发mysql，很可能，还需要密码。如果你的服务器运行在一个其他的机器上，你还需要指定主机名。联系管理员来找到连接参数（例如主机名，用户名和密码），当你知道了正确的参数后，你可以像下面那样连接：

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

host和user表示你的MySQL服务器所运行的主机名和你的mysql用户名。在使用时，请替换成你自己设置的相关值。** 表示你输完上面命令后回车后，需要继续输入密码。

如果工作正常，你应该可以看到mysql提示符的一些介绍信息：

```
shell> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 5.7.15-stan
dard

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

mysql>
```

mysql> 提示符告诉我们，mysql已经准备好了，等待我们输入SQL语句。

如果你正在登录MySQL运行的机器上，你可以忽略host，可以向以下这么简单地来使用：

```
shell> mysql -u user -p
```

当你在尝试登录的时候，你可能会出现错误信息，例如2002（HY000）: Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)，这意味着你的MySQL服务进程没有运行。咨询管理员或查看章节2，给你的操作系统安装和更新MySQL是比较合适的。

For help with other problems often encountered when trying to log in, see Section B.5.2, “Common Errors When Using MySQL Programs”.

有一些MySQL安装允许用来以匿名用户的方式连接到正在本地运行的MySQL服务。如果你是这种情况，你在连接到服务器的时候，不加任何参数。如下。

```
shell> mysql
```

在你连接成功之后，你可以在mysql>命令提示符下输入QUIT（或\q）来断开连接。如下所示。

```
mysql> QUIT
Bye
```

在Unix环境下，你还可以通过按下Control+D的方式来断开连接。

以下章节的例子是在假设你已经成功连接到了服务器上，默认是在mysql>命令提示符下。

查询

像前面章节讨论一样，请确保你已经连接到了服务器。連好之后，不要在里面查询任何数据庫，当然，你也可以查。在这个时候，相比直接跳到创建表，加载数据到表，然后从中检索数据，在先对查询有一定的了解，这是非常重要的，这部分描述了查询的基本原理，使用一些查询，你可以试着了解mysql是如何工作的。

这里有一个简单的查询例子，查询的是版本号和当前日期。在mysql命令提示符中像如下一样输入：

```
mysql> SELECT VERSION(), CURRENT_DATE;
+--------------+--------------+
| VERSION()    | CURRENT_DATE |
+--------------+--------------+
| 5.7.1-m4-log | 2012-12-25   |
+--------------+--------------+
1 row in set (0.01 sec)
mysql>
```

这个查询描述了mysql的一些事情：

一个查询正常情况下包含一个SQL语句，然后跟上一个分号（；）（当然也有不输入分号的例外，如QUIT，还有其他的，随后会了解到），。

当你执行一个查询，mysql会把它发送到服务器执行并显示结果，然后打印另一个mysql>命令提示符，来为你的下一个查询作好准备。

mysql以表格（行和列）的方式显示查询结果。第一行包含列标签。接下来的行是查询结果。列标签是你从数据庫表中提取的列的名字。如果你正在检索的不是一个表的列，而是一个表达式的值，mysql会用表达式本身来标记列。

mysql会显示返回结果的行数和执行查询所消耗的时间，这可以给你一个服务性能的粗略显示。 这些时间值不是非常准确的，因为他们表示的是时钟时间（不是CPU或机器时间），因为他们会受到诸如加载和网络延迟的因子影响。（简单起见，在接下来的例子中，“rows in set"有时没有显示）

关键詞不区别大小写，以下查询是等价的：

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

以下是另一个查询，你可以使用mysql做一个简单的计算：

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

这些查询相对来说，比较短，也是单行语句。你可以在一行输入多条语句，仅仅需要在每一个语句后加上一个分号：

```
mysql> SELECT VERSION(); SELECT NOW();
+------------------+
| VERSION()        |
+------------------+
| 5.7.10-ndb-7.5.1 |
+------------------+
1 row in set (0.00 sec)

+---------------------+
| NOW()               |
+---------------------+
| 2016-01-29 18:02:55 |
+---------------------+
1 row in set (0.00 sec)
```

一个查询不需要把所有的都放一行，如此长的查询，需要放在多行不是问题，mysql决定你的语句是否结束是查找分号，而不是查找输入行的结尾。（换言之，mysql接受自行格式输入：它可以一直接受输入，直到看到分号）

以下是一个简单的多行语句：

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+---------------+--------------+
| USER()        | CURRENT_DATE |
+---------------+--------------+
| jon@localhost | 2010-08-06   |
+---------------+--------------+
```

在这个例子中，注意到在你输入多行的第一行然后回车后，提示符是如何从mysql>改变成->的，这告诉你，mysql还没有看到一个完整的语句，正在等待剩下的。提示符是你的朋友，因为他提供了有价值的反馈。如果你使用这个反馈，你可以一直感知到mysql正在等待。

如果你打算取消正在输入的查询，你可以输入\c，像下面一样：

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

在这，可以注意到，在你输入\c之后，提示符切换回了mysql> 。以此提供了一个反馈，暗示已经为新的查询做好了准备。

以下表格显示了每一个提示符所表示的含义：

提示符	含义
mysql>	已为新的查询作好准备
->	正在等待多行的下一行
'>	正在等待以单引号开头的字符串的下一行
">	正在等待以双引号开头的字符串的下一行
`>	正在等待以（‘）号开头的字符串的下一行
/*>	正在等待以注释符（/*）开头的字符串的下一行

例如：

```
mysql> select user()`
    `> `
    -> /*
   /*> */
    -> '
    '> '
    -> "
    "> "
    -> \c
```

4.3 创建和使用数据库

4.3.1 Creating and Selecting a Database

4.3.2 Creating a Table

4.3.3 Loading Data into a Table

4.3.4 Retrieving Information from a Table

一旦你知道如何输入SQL语句的时候，你就可以准备访问一个数据庫。

假设，在你家（你的menagerie）有一些宠物，并且你想对他们的信息保持一个跟踪。这时候，你可以创建表来存储和加载你渴望的信息。然后你就可以通过检索数据庫里的表来回答各种各样的问题，这部分展示如何执行以下操作：

- 创建数据庫
- 创建表
- 加载数据到表
- 以不同的方式从表中检索数据
- 使用多个表

menagerie数据庫是简单的，但不难想像到，在眞实世界中，这样相似的数据庫会被使用。例如，像这样的数据庫可以被农夫用于跟踪牲畜的信息，兽医跟踪病号记录。

使用SHOW语句来找到服务器上当前存在数据庫：

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

mysql数据库描述的是用户权限。test数据库通常是用于用户作为测试使用。

在你的机器上面，通过这条语句显示出的数据库列表可能是不同的，SHOW DATABASES只会显示当前用户具有权限的数据库，不会显示你没有权限的数据库。查看 14.7.5.14 部分的SHOW DATABASES Syntax 。

如果test数据库存在，尝试访问它：

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

USE,就像QUIT一样，不需要加分号（如果你喜欢，你可以加上分号）。USE语句还有另外一个特殊的地方：它必须出现在单行。

对于接下来的例子，你可以使用ｔｅｓｔ数据库（如果你能访问它），但是你创建的任何东西都可以被能访问它的用户删除。对于这个原因，你可能应该找你的MySQL管理员要属于你的数据库。假如你想使用menagerie，管理员应该执行像下面这个一样的语句：

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

your_mysql_name是MySQL分配给你的用户名，your_client_host是你连接到的服务器的主机名。

创建和选择数据库

创建menagerie数据库：

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

如果想在创建数据库时，指定数据库的字符编码（这里使用ＧＢＫ），可以使用以下方式：

```
mysql> create database test
    -> default character set gbk;
```

但是，即使这么设置了，在使用程序对MySQL进行插入中文记录操作的时候，仍然会出现中文乱码情况，此时，最为有效的方式，就是修改配置文件

```
hadoop@node1:~/mysql_shell$ sudo vi /etc/mysql/my.cnf
```

然后在文件底部加入以下内容：

```
[mysqld]

character-set-server=utf8
collation-server=utf8_general_ci
```

接着重启MySQL服务：

```
hadoop@node1:~/mysql_shell$ sudo service mysql stop
hadoop@node1:~/mysql_shell$ sudo service mysql start
```

通过 `status` 命令可以查看状态信息：

```
mysql> status
--------------
mysql  Ver 14.14 Distrib 5.7.13, for Linux (x86_64) using  EditL
ine wrapper

Connection id:        3
Current database:    eduCloud
Current user:        root@localhost
SSL:            Not in use
Current pager:        stdout
Using outfile:        ''
Using delimiter:    ;
Server version:        5.7.13-0ubuntu0.16.04.2 (Ubuntu)
Protocol version:    10
Connection:        Localhost via UNIX socket
Server characterset:    utf8
Db     characterset:    utf8
Client characterset:    utf8
Conn.  characterset:    utf8
UNIX socket:        /var/run/mysqld/mysqld.sock
Uptime:            9 min 25 sec

Threads: 1  Questions: 62  Slow queries: 0  Opens: 128  Flush ta
bles: 1  Open tables: 47  Queries per second avg: 0.109
--------------
```

使用(选择）menagerie数据庫：

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

还可以在连接到数据庫服务器的时候，指定要使用的数据庫menagerie：

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

创建表

创建pet表（宠物表），包含列：宠物名字，拥有者，物种，性别，出生日期，死亡日期：

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

一旦你创建了表，你可以查看刚刚创建的表是否是想创建的表：

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

```
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    |       |
+---------+-------------+------+-----+---------+-------+
```

可以看到表pet的字段，类型，等数据。

我们也可以通过脚本的方式进行创建表，在一个文件中写创建表的脚本，然后在mysql>中使用source命令或.运行，如下：

```
mysql> source ./testCreateTable.txt;
Database changed
Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected (0.24 sec)
```

```
mysql> \. ./studentScore.txt
Database changed
Query OK, 0 rows affected (0.14 sec)

Query OK, 0 rows affected (0.24 sec)
```

其中，testCreateTable.txt是脚本文件，内容如下：

```
use test
drop table if exists testTable;
create table testTable(name varchar(50),
address varchar(50),
timestamp timestamp,
primary key(name));
```

*仅限示例，没有考虑表的设计规范。

加载数据到表中

创建表之后，你需要填充数据，你可以通过LOAD DATA和INSERT来实现。

数据格式如下：

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

列之间使用\t间隔（LOAD DATA默认的列间隔符），\N表示NULL。

加载文件pet.txt中的数据到表pet中，使用以下命令：

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

如果pet.txt文件是在Windows下编辑的，则换行符为\r\n，需要指定\r\n为行终止符，如下所示：

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

（如果是在苹果系统中，你可能需要使用\r行终止符。）

在LOAD DATA语句中，你可以显示指定列间隔符和行终止符。默认的列间隔符为TAB，行终止符为换行。

当你想新增一条数据时，可以使用INSERT语句：

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

注意：这里不能像LOAD DATA一样，空值要使用NULL，不能使用\N。

检索数据

SELECT语句用于从表中检索数据，通常格式如下：

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

what_to_select 表示你想看到的数据，可以是具体列，也可以是*(代表所有列)。

which_table 表示你要从哪个表中检索数据。

WHERE 是可选的，如果有的话，conditions_to_satisfy表示指定一个或多个行应该满足的条件。

查询所有数据

```
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       |
+----------+--------+---------+------+------------+------------+
```

查询指定行数据

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

```
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  |
+----------+-------+---------+------+------------+-------+
```

```
mysql> SELECT * FROM pet WHERE species = 'snake' OR species = 'b
ird';
+----------+-------+---------+------+------------+-------+
| 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  |
+----------+-------+---------+------+------------+-------+
```

```
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  |
+-------+--------+---------+------+------------+-------+
```

查询指定列

```
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  |
+-------+--------+---------+------+------------+-------+
```

查询宠物表中的宠物拥有者（不重复）。使用DISTINCT关键詞：

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

```
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 |
+--------+---------+------------+
```

行排序

```
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 |
+--------+---------+------------+
```

默认的排序是升序（ASC），以下是按降序排列：

```
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 |
+----------+------------+
```

日期计算

使用TIMESTAMPDIFF查询宠物年龄：

```
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 |
+----------+------------+------------+------+
```

查询death不为NULL，按年龄升序排列：

```
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 |
+--------+------------+------------+------+
```

查询使用death IS NOT NULL，而不是death <> NULL,因为NULL是一个特殊的值，不能使用常规的方法来比较。

查询出年日期是5月份的宠物：

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

查询下一个月过生日的宠物名称和生日，其中DATE_ADD表示日期加，在这里表示，在当前日期上再加一个月的间隔，也就是，如果当前日期为2016-7-30,则加一个月间隔就是8月份。为什么要这么加呢？大家知道，如果直接加的话，到了12月就变成13月了，明显这不合理，当然，我们也可以通过MOD方法来做到12月到1月的过渡。

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

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

使用NULL

在概念上，NULL是一个丢失的未知的值，它与其他值被不同的对待。

使用IS NULL，IS NOT NULL操作符，如下所示：

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

因为1是一个数字，或者说是一个整型值，所以，对于一个具体的整型值来说，它不是NULL，所以，1 IS NULL为假，所以显示0，而1IS NOT NULL，则表示真，返回1。

对于NULL，你不能使用算术运算符，例如=, <>, < , > 等，如下：

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

在MySQL中，0或NULL表示假，其他任何值表示真，默认的真值为1。

模式匹配

MySQL提供了一个标准的SQL模式匹配，和基于扩展的正则表达式的模式匹配Unix工具（如vi,grep,sed)一样。

SQL模式匹配可以使用“_“来匹配任意单个字符，”%“可以用来匹配任意数量（包含0个字符）的字符。在MySQL中，SQL模式匹配的大小写默认是不敏感的，以下有一些例子，当你在使用SQL模式时，不要使用 = 或 <>，而是使用LIKE 或 NOT LIKE。

要找到以字符“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 |
+--------+--------+---------+------+------------+------------+
```

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  |
+--------+--------+---------+------+------------+-------+
```

找到名字包含字符“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       |
+----------+-------+---------+------+------------+------------+
```

使用5个“_“模式字符，来找到包含5个字符的名字：

```
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提供的其他模式匹配类型是使用扩展的正则表达式，当你使用这个类型来测试一个匹配，要使用REGEXP和NOT REGEXP操作（或者RLIKE和NOT RLIKE，他们是同意詞）。

下面的列表描述了一些扩展正则表达式的特征：

- “.” 匹配任意单个字符.
- 一个字符类“[...]"匹配括号里的任意字符，例如，"[abc]"匹配“a","b",或"c",要指定字符范围，可以使用“-",例如，“[a-z]"匹配任意字母，而“[0-9]"匹配任意数字。
- “*"匹配0个或多个在它前面的东西。例如，“x*“匹配任意个x字符，"[0-9]*"匹配任意个数字，“.*"匹配任意个字符。
- 一个REGEXP模式匹配成功的条件是，模式在测试值中的任意地方匹配即可。（这与LIKE模式匹配不同，LIKE模式匹配成功需要匹配整个值）。
- 想让模式匹配测试值的开头或结尾，可以使用“^"开头作为模式的开头或用“$"作为模式的结尾。

为了验证扩展的正则表达式是如何工作的，在这使用REGEXP对前面使用LIKE查询进行重写。

使用“^“来匹配，找到以“b“开头的名字

```
mysql> SELECT * FROM pet WHERE name REGEXP '^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 |
+--------+--------+---------+------+------------+------------+
```

如果你真的想强制REGEXP比较为大小写敏感，可以使用BINARY关键詞来把字符串转换成二进制字符串。 以下这个查询只匹配以小写字母“b“开头的名字：

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

要找到以"fy"结尾的名字，使用“$“来匹配名字的结尾：

```
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  |
+--------+--------+---------+------+------------+-------+
```

要找到名字包含一个"w"的名字，使用这个查询：

```
mysql> SELECT * FROM pet WHERE name REGEXP '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       |
+----------+-------+---------+------+------------+------------+
```

因为一个正则表达式的模式的匹配，是不管发生在值的哪个位置，在前面查询中，它不需要像使用一个SQL模式,放一个通配符在模式的两端，然后来匹配整个值。

要找到只包含5个字符的名字，使用“^"和“$"来匹配名字的开头和结尾，且放５个“."在中间。如下所示：

```
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）操作符来写前面的查询：

```
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  |
+-------+--------+---------+------+------------+-------+
```

更多的正则表达式的语法信息，请看Section 13.5.2, “Regular Expressions”。

计算行

数据库通常被用于回答问题,“在一个表中，特定的数据有多少条？”，例如，你可能想知道你有多少宠物，或者，每个宠物拥有者有多少只宠物，或者，在普查中，你可能想知道有多少种宠物。

计算你的宠物数量与“在你的宠物表中有多少行”是同样一个问题，因为每个宠物都有一个记录。COUNT(*)可以计算行的数量，所以，统计你宠物数量可以这样：

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

早期，你查询人的名字，及此人所有宠物的数量，你会使用COUNT()来找到拥有者的宠物数量：

```
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来对每个拥有者的记录进行分组。COUNT()和GROUP BY的组合使用对于各种数据分组是很有用的，以下例子显示了执行宠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 |
+---------+----------+
```

每种性别的动物数量：

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

（在这个输出结果中，NULL表示性别未知）

每个物种各性别对应的动物数量:

```
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 |
+---------+------+----------+
```

当你在使用时COUNT()时，你不需要检索整个表。例如，对于前面的查询，当只查询dogs和cats时，可以像这样：

```
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，如果你想知道，sex是已知的那些动物，每个性别所拥有的动物数量：

```
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 |
+---------+------+----------+
```

如果你的列名被用于COUNT()来求值，一个GROUP BY谓詞应该也被用于这些列名，否则，会有如下错误：

如果ONLY_FULL_GROUP_BY模式是启用的，会发生这样的错误：

```
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, expres
sion
#1 of SELECT list contains nonaggregated column 'menagerie.pet.o
wner';
this is incompatible with sql_mode=only_full_group_by
```

如果ONLY_FULL_GROUP_BY没有启用，这个查询会把所有的行都当成一个单一的组来处理，但是，被选择出的列名的值是不确定的。服务器是自动选择了任意行的值：

```
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)
```

See also Section 13.20.3, “MySQL Handling of GROUP BY”.

获取数据库和表信息

如果你忘记了数据库或表的名字，或者表的结构（例如，表的列名叫什么），MySQL通过一些语句，提供了关于数据库和表的信息，解决了这些问题。

你前面看到过SHOW DATABASES,这会列出数据库。要找到当前正选择的数据库，使用DATABASE()方法：

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

如果你还没有选择任何数据库，结果是ＮＵＬＬ。

要找到默认数据库包含的表（例如，当你对一个表的名字不太确定时），使用这个语句：

```
mysql> SHOW TABLES;
+----------------------+
| Tables_in_menagerie |
+----------------------+
| event                |
| pet                  |
+----------------------+
```

在输出中，列的名字为Tables_in_db_name，其中db_name部分为当前数据库名字， 更多信息请看14.7.5.37, “SHOW TABLES Syntax”部分。

如果你想看一个表的结构，DESCRIBE语句是有用的；它显示一个表的每一个列的信息：

```
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表示列名，Type表示对应列的数据类型，NULL代表此列是否能包含NULL值，Key代表此列是否为索引，Default指定列的默认值。Extra显示关于列的指定信息。如果一个列是被创建为AUTO_INCREMENT，则这个值将为auto_increment，而不是空的。

常用查询例子

在命令行中，选择要操作的数据库：

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

创建数据库表，并往里添加数据：

```
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);
```

上述语句执行后，有如下结果：

```
mysql> 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 |
+---------+--------+-------+
7 rows in set (0.01 sec)
```

查询列特定列中的最大值

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

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

简单的子查询：

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

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

也可以通过以下方式来达到同样的效果：

```
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;
```

每个分组里的最大值

```
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 |
+---------+-------+
```

使用用户自定义变量

找到价格最高和最低的信息：

```
mysql> SELECT @min_price:=MIN(price),@max_price:=MAX(price) FROM
 shop;
mysql> SELECT * FROM shop WHERE price=@min_price OR price=@max_p
rice;
+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

计算每天的访问量

准备数据：

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

表格中的数据是用户访问网站的年，月，日，现在要计算每个月访问的天数，可以用以下语句：

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

结果是，1月份有3天，2月份有2天，如下表所示：

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

其中，相同的天（23日）只算一次。

使用AUTO_INCREMENT

使用AUTO_INCREMENT可以为每个新行自动产生一个唯一标识。

```
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;
```

返回以下数据：

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

注意：在插入值的时候，需要在表名之后，指定列名，否则会报错

SQL语句语法

MySQL混合语句语法

流程控制语句

CASE语法

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

或者

```
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

对于存储程序，CASE语句实现了一个复杂的条件结构。

对于第一个语法，case_value是一个表达式，这个值与when_value相比较，如果相等，则进入相应的statement_list，statement_list就是一个语句列表，可以包含多条语句。如果在when_value中没有和case_value相等的值，则进入ELSE里面的statement_list语句列表。

对于第二个语法，每一个WHEN后面的search_condition表达式都会被判断，直到一个表达式的结果为真，然后执行相应的THEN后面的语句序列。如果没有匹配的，则执行ELSE后面的语句列表。

如果when_value或search_condition都没有匹配到相应的值，并且没有写ELSE的时候，会出现a Case not found for CASE statement error results错误。

每个statement_list应该包含一个或多个SQL语句，一个空的statement_list是不允许的。

要处理这种没有匹配，也不想在ELSE里写任何SQL语句的时候，可以在ELSE里写一个空的BEGIN ... END 块，如下所示：

```
DELIMITER |

CREATE PROCEDURE p()
  BEGIN
    DECLARE v INT DEFAULT 1;

    CASE v
      WHEN 2 THEN SELECT v;
      WHEN 3 THEN SELECT 0;
      ELSE
        BEGIN
        END;
    END CASE;
  END;
  |
```

IF语法

IF和END IF是成对出现

使用*ELSEIF*的时候，要特别注意，由于平时用习惯*ELSE IF*（即中间多个空格），所以这里很容易出错，而且出错信息很奇怪，记住：中间没有空格。

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

```
DELIMITER //

CREATE FUNCTION SimpleCompare(n INT, m INT)
  RETURNS VARCHAR(20)

  BEGIN
    DECLARE s VARCHAR(20);

    IF n > m THEN SET s = '>';
    ELSEIF n = m THEN SET s = '=';
    ELSE SET s = '<';
    END IF;

    SET s = CONCAT(n, ' ', s, ' ', m);

    RETURN s;
  END //

DELIMITER ;
```

```
DELIMITER //

CREATE FUNCTION VerboseCompare (n INT, m INT)
  RETURNS VARCHAR(50)

  BEGIN
    DECLARE s VARCHAR(50);

    IF n = m THEN SET s = 'equals';
    ELSE
      IF n > m THEN SET s = 'greater';
      ELSE SET s = 'less';
      END IF;

      SET s = CONCAT('is ', s, ' than');
    END IF;

    SET s = CONCAT(n, ' ', s, ' ', m, '.');

    RETURN s;
  END //

DELIMITER ;
```

MySQL程序

目录：

这章给出了一个对Oracle公司提供的MySQL命令行程序的简要概述，它也讨论了当你在运行这些程序时，对于指定参数(options)的通用语法。大多数程序可以指定参数到自己的操作，但是他们的语法相似，最后，本章给出了个别程序的详细描述，包括他们支持的参数。

MySQL程序概述

There are many different programs in a MySQL installation. This section provides a brief overview of them. Later sections provide a more detailed description of each one, with the exception of MySQL Cluster programs. Each program's description indicates its invocation syntax and the options that it supports. Section 19.4, “MySQL Cluster Programs”, describes programs specific to MySQL Cluster.

Most MySQL distributions include all of these programs, except for those programs that are platform-specific. (For example, the server startup scripts are not used on Windows.) The exception is that RPM distributions are more specialized. There is one RPM for the server, another for client programs, and so forth. If you appear to be missing one or more programs, see Chapter 2, Installing and Upgrading MySQL, for information on types of distributions and what they contain. It may be that you have a distribution that does not include all programs and you need to install an additional package.

Each MySQL program takes many different options. Most programs provide a --help option that you can use to get a description of the program's different options. For example, try mysql --help.

You can override default option values for MySQL programs by specifying options on the command line or in an option file. See Section 5.2, “Using MySQL Programs”, for general information on invoking programs and specifying program options.

The MySQL server, mysqld, is the main program that does most of the work in a MySQL installation. The server is accompanied by several related scripts that assist you in starting and stopping the server:

使用JDBC的CallableStatements来执行存储过程

创建存储过程demoSp：

```
CREATE PROCEDURE demoSp(IN inputParam VARCHAR(255), \
                        INOUT inOutParam INT)
BEGIN
    DECLARE z INT;
    SET z = inOutParam + 1;
    SET inOutParam = z;

    SELECT inputParam;

    SELECT CONCAT('zyxw', inputParam);
END
```

其中，我们可以这样理解，IN表示后面跟的是输入参数，INOUT后面跟的是输出参数。

要使用Connector/J来调用demoSp存储过程，有以下步骤：

1、使用 `Connection.prepareCall()` 来准备调用语句。如下：

```
CallableStatement cStmt = conn.prepareCall("{call demoSp(?, ?)}"
);
//下标是从1开始,第一个参数是输入参数
//这句是给第一个参数赋初值
cStmt.setString(1, "abcdefg");
```

2、注册输出参数（如果有）

输出参数是在创建存储过程时，用OUT或INOUT修饰过的变量

```
// 以下两种方式均可

// 1、注册输出参数，第二个参数是用INOUT修饰过
cStmt.registerOutParameter(2, Types.INTEGER);

// 2、也可以使用这种方式进行注册参数，直接用参数变量名，而不是使用索引。
cStmt.registerOutParameter("inOutParam", Types.INTEGER);
```

3、注册输入参数

```
    //
    // 通过索引，设置参数
    //

    cStmt.setString(1, "abcdefg");

    //
    // 也可以使用名字来设置一个参数
    //

    cStmt.setString("inputParam", "abcdefg");

    //
    // 使用索引设置'in/out'参数
    //

    cStmt.setInt(2, 1);

    //
    // 也可以使用名字来设置'in/out'参数
    //

    cStmt.setInt("inOutParam", 1);
```

4、执行 `CallableStatement` ，检索任何结果集或输出参数。

尽管 `CallableStatement` 支持调用任意的执行方法（ `executeUpdate()` ， `executeQuery()` 或 `execute()` ），最灵活的方法是 `execute()` ，因为你不需要提前知道存储过程是否返回结果集。

```
...

    boolean hadResults = cStmt.execute();

    //
    // 处理所有的返回结果集
    //

    while (hadResults) {
        ResultSet rs = cStmt.getResultSet();

        // 处理结果集
        ...

        hadResults = cStmt.getMoreResults();
    }

    //
    // 检索输出参数
    //
    // Connector/J支持基本索引和基于名字来检索
    //

    int outputValue = cStmt.getInt(2); // 基于索引

    outputValue = cStmt.getInt("inOutParam"); // 基于名字

...
```

如果你想在MySQL>里调用存储过程，请参考：Call Syntax