# MySQL快速入门

这里的内容主要来自MySQL 5.7手册的第三章。

本章主要介绍通过mysql这个程序链接到MySQL服务器，并对服务器进行使用。mysql程序是一个终端交互程序，用于链接到MySQL服务器，提交查询并查看结果。**[mysql](https://dev.mysql.com/doc/refman/5.7/en/mysql.html" \o "4.5.1 mysql  -  MySQL命令行工具)**也可在批处理模式下使用：先将查询放在一个文件中，然后告诉 [**mysql**](https://dev.mysql.com/doc/refman/5.7/en/mysql.html)执行文件的内容。本章介绍这两种使用方式。本章只是一个简单的入门教程。要查看mysql提供的列表，请使用mysql –help。

## 1、连接和断开服务器

连接时通常需要指定要链接的主机、端口、用户名、密码。

shell> mysql -h *host* -u *user* –P 3306 -p

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

如果客户端与服务器在一台机器上，可以：

shell> mysql -u *user* -p

键入QUIT来退出mysql程序。

## 2、输入查询

MySQL可以包含多个数据库，如上与服务器建立连接后并未选择任何数据库。此时依然可以与mysqld进行交互。例如像MySQL服务器询问版本和当前日期：

mysql> SELECT VERSION(), CURRENT\_DATE;

mysql> select version(), current\_date;

mysql> SeLeCt vErSiOn(), current\_DATE;

如上可见，对于关键字，MySQL不区分大小写。

同一个查询也可以位于多行：

mysql> SELECT

-> USER()

-> ,

-> CURRENT\_DATE;

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

| USER() | CURRENT\_DATE |

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

| jon@localhost | 2010-08-06 |

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

此例中，在第一行提示符为mysql>，后续的为->表示mysql还未看到一个完整的语句。正在等待其余的输入。如果不想执行正在输入的查询，可以输入\c，如下：

mysql> SELECT

-> USER()

-> \c

mysql>

输入\c之后，提示mysql>，表示mysql准备好接受新的查询。

下边是mysql程序会出现的提示符即对应的含义，了解这些是有用的。

| **提示** | **含义** |
| --- | --- |
| mysql> | 准备新的查询 |
| -> | 等待多行查询的下一行 |
| '> | 等待下一行，等待完成一个以单引号（'）开始的字符串 |
| "> | 等待下一行，等待完成一个以双引号（"）开头的字符串 |
| `> | 等待下一行，等待完成一个以反引号（`）开头的标识符 |
| /\*> | 等待下一行，等待完成一个以/\*开始的评论 |

例如你输入如下查询：

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

'>

这时你会感到奇怪，我的查询怎么不执行？这时要注意mysql程序的提示符，mysql客户端程序认为你的字符串缺少单引号。但是你却知道，实际上你是输入错误，那么接下来可以如下输入，从而取消本查询：

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

'> '\c

mysql>

注意要输入'\c，只输入\c不管用哈。

## 3、创建和使用数据库

**1）创建和选择数据库**

先看看当前服务器上存在哪些数据库：

mysql> SHOW DATABASES;

该语句不显示你没有权限的数据库。

选择某个数据库：

mysql> USE test

将某个数据库的所有权限赋予一个用户：

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

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

创建一个数据库：

mysql> CREATE DATABASE menagerie;

创建之后，并不会默认选择该数据库，你必须手动执行USE语句。数据库只需要创建一次，但是每次开始**[mysql](https://dev.mysql.com/doc/refman/5.7/en/mysql.html" \o "4.5.1 mysql  -  MySQL命令行工具)** 会话时都必须选择它。可以通过发出[USE](https://dev.mysql.com/doc/refman/5.7/en/use.html" \o "13.8.4使用语法)语句来执行此操作 。或者，可以在命令行调用**[mysql](https://dev.mysql.com/doc/refman/5.7/en/mysql.html" \o "4.5.1 mysql  -  MySQL命令行工具)**时选择数据库。只需在连接参数最后指定其名称。例如：

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

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

注意：menagerie为数据库，并不是密码，如果要在p之后跟随密码，p和密码之间不可以有空格。但是，不建议将密码放在命令行上，因为这样做会使其被登录在您的计算机上的其他用户所窥探。

您可以随时查看当前使用哪个数据库。 SELECT DATABASE()

**2）创建表**

先看看当前数据库中有哪些表：

mysql> SHOW TABLES;

Empty set (0.00 sec)

创建一张宠物表：

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

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

该表记录宠物的最为基本的信息：名称，所有者，物种，性别，出生和死亡。

**3）将数据装载到表中**

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

如果在Windows上使用\r\n用作行终止符的编辑器创建该文件 ，则应改用以下语句：

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

如果需要，可以在[LOAD DATA](https://dev.mysql.com/doc/refman/5.7/en/load-data.html" \o "13.2.6 LOAD DATA INFILE语法)语句中明确指定列值之间的分隔符和行尾标记，默认值为制表符和换行符。

也可以插入一条记录：

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

**4）从表中查询信息**

**查询所有数据：**

从表 中检索所有内容的最简单的形式：

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 | +----------+--------+---------+------+------------+------------+

如果你发现表格中某些数据不正确，可以使用update语句：

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

**选择特定的行：**

mysql> SELECT \* FROM pet WHERE name = 'Bowser';

mysql> SELECT \* FROM pet WHERE birth >= '1998-1-1';

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

mysql> SELECT \* FROM pet WHERE species = 'snake' OR species = 'bird';

AND和OR很容易混乱，尽管AND优先级高于OR。因此在使用时最好明确用括号包起来：

mysql> SELECT \* FROM pet WHERE (species = 'cat' AND sex = 'm')

-> OR (species = 'dog' AND sex = 'f');

**选择特定的列：**

mysql> SELECT name, birth FROM pet;

mysql> SELECT owner FROM pet;

mysql> SELECT DISTINCT owner FROM pet;

mysql> SELECT name, species, birth FROM pet WHERE species = 'dog' OR species = 'cat';

**排序行：将查询结果以某种方式排序输出**

按日期排序：

mysql> SELECT name, birth FROM pet ORDER BY birth;

默认为升序，降序排列需要明确指定：

mysql> SELECT name, birth FROM pet ORDER BY birth DESC;

可以按多个列排序，例如：按照动物类型升序（默认）排列，按照出生日期降序排列：

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关键字仅适用于它之前的列名（birth）; 不影响species列的排序顺序。

**日期计算：**

MySQL提供了几种可用于执行日期计算的功能，例如计算年龄或提取部分日期。

要确定你宠物的年龄，请使用 [TIMESTAMPDIFF()](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_timestampdiff)函数。它依据你希望的表达单位和两个日期的差异求的计算结果。以下查询显示每只宠物的出生日期，当前日期和年龄。使用 *别名*（age）来使最终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 |

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

也可以将结果以某种顺序输出：

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 |

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

如果你想知道下个月有哪些动物过生日呢？对于这种类型的计算，年和日是无关紧要的; 你只需要提取birth列的月份部分 。MySQL提供了用于提取日期的部分，如一些函数 [YEAR()](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_year)， [MONTH()](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_month)和 [DAYOFMONTH()](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_dayofmonth)。 [MONTH()](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_month)是现在需要的函数，看看它如何工作：

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 |

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

假设当月是4月。找出在五月（月5）里出生的动物：

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

如果当月为12月，那么总不能写成13月吧。你可以这么写：

mysql> SELECT name, birth FROM pet

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

[DATE\_ADD()](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_date-add)函数可以参考MySQL手册或者网上查查其用法。

或者：

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

**使用NULL值：**

NULL在概念上是一个未知的值。要测试NULL，使用 [IS NULL](https://dev.mysql.com/doc/refman/5.7/en/comparison-operators.html#operator_is-null)和[IS NOT NULL](https://dev.mysql.com/doc/refman/5.7/en/comparison-operators.html" \l "operator_is-not-null)运算符，如下所示：

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

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

| 1 IS NULL | 1 IS NOT NULL |

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

| 0 | 1 |

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

不能使用比较运算符，如：=、<、>、<>等。因为一个不知道的值与其他的值比结果还是不知道即：NULL，一次作比较无任何意义。

GROUP BY中两个NULL值被认为是相等的 。

当执行ORDER BY时，如果执行ORDER BY ... ASC，则首先显示NULL值，如果执行ORDER BY ... DESC，则最后显示NULL值。

**模式匹配：**

MySQL也提供类似UNIX的正则表达式的操作符，即：LIKE, NOT LIKE。

SQL模式匹配中：\_匹配任意单个字符，%匹配任意多个字符（包括0个），在MySQL中，SQL模式默认情况下不区分大小写，这一点下边说明。

找到名字以b开头的数据：

mysql> SELECT \* FROM pet WHERE name LIKE 'b%';

找到名字以fy结尾的数据：

mysql> SELECT \* FROM pet WHERE name LIKE '%fy';

发现名字中包含w的数据：

mysql> SELECT \* FROM pet WHERE name LIKE '%w%';

发现名字中恰好包含5个字符的数据，使用五个\_：

mysql> SELECT \* FROM pet WHERE name LIKE '\_\_\_\_\_';

MySQL提供的其他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匹配时，请使用 [REGEXP](https://dev.mysql.com/doc/refman/5.7/en/regexp.html#operator_regexp)和[NOT REGEXP](https://dev.mysql.com/doc/refman/5.7/en/regexp.html" \l "operator_not-regexp)操作符（或者 [RLIKE](https://dev.mysql.com/doc/refman/5.7/en/regexp.html#operator_regexp)和 [NOT RLIKE](https://dev.mysql.com/doc/refman/5.7/en/regexp.html#operator_not-regexp)，它们是同义词）。

以下描述了扩展的正则表达式的一些特征：

* **.** 匹配任何单个字符。
* 字符类[...]会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，请使用破折号。[a-z] 匹配任何字母，而[0-9] 匹配任何数字。
* \*匹配其前面的事物的零个或多个实例。例如，x\* 匹配任何数量的x字符， [0-9]\*匹配任意数量的数字，**.\***匹配任何数量的任何东西。
* [REGEXP](https://dev.mysql.com/doc/refman/5.7/en/regexp.html#operator_regexp)的时候，当模式匹配被测值中的任何位置时匹配成功。（这与[LIKE](https://dev.mysql.com/doc/refman/5.7/en/string-comparison-functions.html" \l "operator_like)模式匹配不同 ，只有当模式匹配整个值时才能成功）
* 为了表达必须与正在测试的值的开头或结尾相匹配，需要在模式开始处使用^或在模式结尾处使用$

为了演示扩展正则表达式的工作原理，将使用[REGEXP](https://dev.mysql.com/doc/refman/5.7/en/regexp.html" \l "operator_regexp)重写之前的LIKE查询。

名字以b开头的行：

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

如果你真的要强制 [REGEXP](https://dev.mysql.com/doc/refman/5.7/en/regexp.html#operator_regexp)区分大小写，请使用[BINARY](https://dev.mysql.com/doc/refman/5.7/en/cast-functions.html" \l "operator_binary)关键字使其中一个字符串成为二进制字符串。此查询仅匹配在名称开头为小写b的情况：

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

名字以fy结尾的行：

mysql> SELECT \* FROM pet WHERE name REGEXP 'fy$';

名字中包含w的行：

mysql> SELECT \* FROM pet WHERE name REGEXP 'w';

因为正则表达式模式匹配成功的条件是匹配值的任何位置，因此在上一个查询中不需要在模式的两边放置一个通配符，以使其匹配整个值。

要查找名称正好包含五个字符的行，请使用如下语句 ：

mysql> SELECT \* FROM pet WHERE name REGEXP '^.....$';

也可以：

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

后续会有正则表达式的更多介绍。

**行计数：**

计算宠物的总数：

mysql> SELECT COUNT(\*) FROM pet;

查询，每个宠物所有者分别有多少个宠物:

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

查询每种动物的数量：

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

查询每种性别的宠物数量：

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

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

| sex | COUNT(\*) |

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

| NULL | 1 |

| f | 4 |

| m | 4 |

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

NULL表示性别未知。

查询各个种类和性别相同的宠物的数量：

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 |

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

也可以不对整个表count和group by，这通过where条件完成：

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 |

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

如果启用了SQL模式： [ONLY\_FULL\_GROUP\_BY](https://dev.mysql.com/doc/refman/5.7/en/sql-mode.html#sqlmode_only_full_group_by)，那么如下会出错：

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[ONLY\_FULL\_GROUP\_BY](https://dev.mysql.com/doc/refman/5.7/en/sql-mode.html" \l "sqlmode_only_full_group_by)未启用，则通过将所有行视为单个组来处理查询，但为每个命名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 | 9 |

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

1 row in set (0.00 sec)

**使用多个表：**

前面我们有一张包含宠物基本信息的表pet，包含如下列：名称，所有者，物种，性别，出生和死亡。

现在需要记录这些宠物的更多信息，即关于宠物的一些活动，为此，需要另外一张表：event。如下：

mysql> CREATE TABLE event (name VARCHAR(20), date DATE,

-> type VARCHAR(15), remark VARCHAR(255));

该表包含字段：名称，时间日期、类型、备注。

假设你要查询宠物的年龄以及它的一些事件，但是这些信息在pet或event中的任何一张表中都无法获取，因此需要查询两张表：

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 |

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

注意：因为name列在两个表中都会出现，所以必须具体说明引用哪个表的该列。这可以通过将表名称添加到列名前来完成。

以上是两个不同的表进行连接，有些时候需要同一个表进行连接，例如：你想给物种配对，即：同一种宠物的不同性别的结合，此时就需要连接pet表：

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 p2.sex = 'm'; +--------+------+--------+------+---------+

| name | sex | name | sex | species |

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

| Fluffy | f | Claws | m | cat |

| Buffy | f | Fang | m | dog |

| Buffy | f | Bowser | m | dog |

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

## 4、获取数据库和表的信息

如果你忘记了当前数据库、数据库名、表名等等该怎么办？

展示服务器管理的各个数据库名字：

mysql> SHOW DATABASE;

查看当前选择的数据库：

mysql> SELECT DATABASE();

查看数据库中有那些表：

mysql> SHOW TABLES;

了解表的结构：

mysql> DESCRIBE pet;

也可以使用DESC作为简写，还可以使用SHOW CREATE TABLE tab\_name来看某个表的定义。

## 5、在批处理模式下使用mysql

在前面的部分中，以交互方式使用mysql输入语句并查看结果。也可以在批处理模式下运行mysql。为此，将要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*

如果需要在命令行中指定连接参数，命令可能如下所示：

shell> mysql -h *host* -u *user* -p < *batch-file*

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

当以这种方式使用**mysql时**，你将要创建一个脚本文件，然后执行该脚本。

如果你希望即使其中的某些语句产生错误脚本也继续执行，则应使用 [--force](https://dev.mysql.com/doc/refman/5.7/en/mysql-command-options.html#option_mysql_force)命令行选项。

为什么要使用脚本？以下是几个原因：

* 反复（例如每天或每周）运行查询，使其成为一个脚本，可以在每次执行它时避免重新输入。
* 可以通过复制和编辑脚本文件来生成与现有查询类似的新查询。
* 在开发查询时，批处理模式也很有用，特别是对于多行语句或多语句序列。如果你犯了一个错误，你不需要重新输入一切。只需编辑你的脚本来纠正错误，然后告诉**mysql**再次执行它。
* 如果有一个产生大量输出的查询，你可以通过如下方式处理，而不是看着它从屏幕顶部滚动：

shell> mysql < *batch-file* | more

* 可以将输出捕获到文件中并进一步处理：

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

* 可以将脚本分发给其他人，以便他们也可以运行语句。
* 某些情况不允许交互式使用，例如，当你从**cron**作业运行查询时。在这种情况下，必须使用批处理模式。

当以批处理模式运行[mysql](https://dev.mysql.com/doc/refman/5.7/en/mysql.html" \o "4.5.1 mysql  -  MySQL命令行工具)时，默认的输出格式与交互式不同，例如，SELECT DISTINCT species FROM pet在[mysql](https://dev.mysql.com/doc/refman/5.7/en/mysql.html" \o "4.5.1 mysql  -  MySQL命令行工具)以交互方式运行时，输出如下所示 ：

+---------+

| species |

+---------+

| bird |

| cat |

| dog |

| hamster |

| snake |

+---------+

在批处理模式下，输出如下所示：

species

bird

cat

dog

hamster

snake

如果要以批处理方式获取交互式的输出格式，请使用**mysql -t**。要回传输出执行的语句，请使用**mysql -v**。

还可以在mysql启动后，在mysql提示符下，使用 source 命令或者**\.** 来执行脚本。

mysql> source *filename*;

mysql> \. *filename*

## 6、常见的查询示例

本节列出一些常用的查询示例。基础数据如下：

一张商品交易表，即：各种经销商（dealer）对文章的交易价格的表，每一篇文章可以被多个经销商销售，同一篇文章不同的经销商可以有不同的价格。因此该表的主键为（article, dealer）。

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);

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 |

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

**1）最大值**

SELECT MAX(article) AS article FROM shop;

+---------+

| article |

+---------+

| 4 |

+---------+

**2）查询最大列的行**

任务：找出价格最贵的文章对应的信息：

即：该表中哪一行数据的price最大，因此先要求出该表中price的最大值，然后以他为条件查询：

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

其他方案：

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：**如果有几个最贵的文章，每个价格为19.95，LIMIT解决方案将只显示其中之一。即：这个不通用。

**3）每个分组中的最大值**

任务：找到每篇文章最高的价格。

先按照文章分组，再求每组的最大值：

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

**4）找出每组中包含特定列最大值的行**

任务：对于每篇文章，找到价格最昂贵的经销商。

即：以文章分组，找出每组中价格最大的行。可以如下完成：

SELECT article, dealer, price

FROM shop s1

WHERE price=(SELECT MAX(s2.price)

FROM shop s2

WHERE s1.article = s2.article);

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

| article | dealer | price |

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

| 0001 | B | 3.99 |

| 0002 | A | 10.99 |

| 0003 | C | 1.69 |

| 0004 | D | 19.95 |

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

即：对于每一行，处理where条件，首先计算出对于这一行数据中这一个文章ID的价格的最大值。然后选择出对应的列。

上述示例使用了相关子查询，MySQL可能不支持，也可以如下完成：

思路：首先找出每篇文章最大的价格对应的行：

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

然后与自己链接，目的是为了查询到其他的列数据。但是链接条件是文章ID和价格都相等。

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;

或者：

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;

**5）使用用户定义的变量**

您可以使用MySQL用户变量记住结果，而不必将其存储在客户端中的临时变量中。

例如，要找到最高和最低价格的文章，您可以这样做：

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 |

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

**6）使用外键**

在MySQL中，InnoDB表支持检查外键约束。

外键约束涉及到两个表，但是，在定义时，对于除InnoDB之外的存储引擎而言，外键约束没有任何实际意义，仅仅作为备注或者注释。MySQL本身不会对外键进行任何检查，也不会进行任何操作，即：对于REFERENCES ***tbl\_name***(***col\_name***)，即使你在定义时写了ON DELETE或者ON UPDATE，但是却不会有任何实际行为，即对外键无感知。后边会详细学习，这里只是先说明一下，要使用外键功能还是要使用InnoDB。

**7）在两个字段上进行搜索**

使用单个字段的OR很容易优化，但是对于两个不同字段的OR就不好优化：

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

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

这就不好优化，其实你可以这么做：每个SELECT搜索只有一个字段，可以优化查询：

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';

注意：UNION会去除结果集中重复的行，除非是有UNION ALL。

**8）计算每天的访问量**

计算一个Web页面，在各个月之中有几天被访问到：

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);

示例表中包含用户访问页面的年、月、日。要确定每月有几天被访问到，请如下查询：

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

将返回如下信息：

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

| year | month | days |

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

| 2000 | 01 | 3 |

| 2000 | 02 | 2 |

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

该查询可以计算每年/月组合中表格中显示的不同天数，自动删除重复的条目。

**9）使用AUTO\_INCREMENT**

AUTO\_INCREMENT属性可用于为新行生成唯一标识：

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;

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

| id | name |

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

| 1 | dog |

| 2 | cat |

| 3 | penguin |

| 4 | lax |

| 5 | whale |

| 6 | ostrich |

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

可以看到默认从1开始，如果没有启用SQL模式NO\_AUTO\_VALUE\_ON\_ZER，那么可以明确地将0分配给列以生成序列号。可以使用LAST\_INSERT\_ID（）SQL函数或mysql\_insert\_id（）C API函数检索最近自动生成的AUTO\_INCREMENT值。这些功能是特定于连接的，所以它们的返回值不受另外一个也执行插入的连接的影响。

对于自增列，要合理的选用数据类型以保证可以容纳你的行。如果超过了范围，那么下一次自动生成序号时将失败。为了不让序号从1开始，你可以在CREATE TABLE或者ALTER TABLE中指定：

mysql> ALTER TABLE tbl AUTO\_INCREMENT = 100;

作者：许富博

版权所有，文章以学习和交流为主，切勿用于商业用途。

限于本人水平有限，欢迎大家随时指正，联系方式：

[xufubobo@gmail.com](mailto:xufubobo@gmail.com)

[xufubobo@163.com](mailto:xufubobo@163.com)

[1332841493@qq.com](mailto:1332841493@qq.com)