MySQL中文版

极客学院出版

前言

MySQL 目前是最流行的开源关系型 SQL 数据库管理系统，是一种用于最适于开发 Web 软件应用的 RDBMS。

本教程将引领你快速了解 MySQL，熟悉 MySQL 编程知识。

适宜的读者对象

本系列教程专为初学者量身打造，能够帮助他们从零开始逐步了解 MySQL 的相关知识。

预备知识

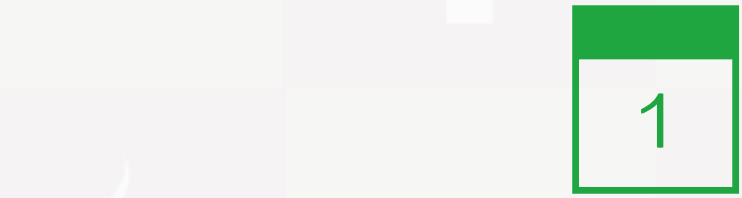
要想实际练习本系列教程中的各种范例，读者应该事先知道什么是数据库（特别是 RDBMS），并且明白何为编程语言。

编译/执行 MySQL 程序

假如你想利用 SQLite DBMS 来编译并执行 SQL 程序，但苦于手头没有安装该系统，不要着急。有一个架设于高端专用服务器上的网站：compileonline.com (http://www.compileonline.com/execute_sql_online.php)，它可以为你提供真实的编程体验，而且具有非常方便的一键执行的功能。它不仅是在线的，而且更棒的是：它是完全免费的！

更新日期	更新内容
2015-04-14	MySQL 教程发布

目录



MySQL – 教程

MySQL 介绍

何谓数据库

数据库是一种用于存储数据集合的独立应用程序。每种数据库都会有一个或多个独特的 API，用来创建、访问、管理、搜索或复制数据库中保存的数据。

除了数据库之外，也可以使用其他一些数据存储方式，比如说利用文件系统来存储文件，或者利用内存中的大型散列表，但这些系统均无法快速便利地提取或写入数据。

因此，现在业界一般采用**关系型数据库管理系统**（RDBMS）来存储并管理海量数据。之所以称其为关系型数据库，是因为所有数据都存储在不同的表中，表之间的关系是建立在主键或其他键（被称为外键）的基础之上的。

关系型数据库管理系统（RDBMS）具有以下特点：

- 能够实现一种具有表、列与索引的数据库。
- 保证不同表的行之间的引用完整性。
- 能自动更新索引。
- 能解释 SQL 查询，组合多张表的信息。

RDBMS 术语

在继续讨论 MySQL 数据库系统之前，先让我们来说明一些关于数据库的术语定义：

- **数据库**（Database）：数据库是带有相关数据的表的集合。
- **表**（Table）：表是带有数据的矩阵。数据库中的表就像一种简单的电子表格。
- **列**（Column）：每一列（数据元素）都包含着同种类型的数据，比如邮编。
- **行**（Row）：行（又被称为元组、项或记录）是一组相关数据，比如有关订阅量的数据。
- **冗余**（Redundancy）：存储两次数据，以便使系统更快速。
- **主键**（Primary Key）：主键是唯一的。同一张表中不允许出现同样两个键值。一个键值只对应着一行。
- **外键**（Foreign Key）：用于连接两张表。
- **复合键**（Compound Key）：复合键（又称组合键）是一种由多列组成的键，因为一列并不足以确定唯一性。

- **索引**（Index）：它在数据库中的作用就像书后的索引一样。
- **引用完整性**（Referential Integrity）：用来确保外键一直指向已存在的一行。

MySQL 数据库

MySQL 是一种快速易用的 RDBMS，很多企业（不分规模大小）都在使用它来构建自己的数据库。MySQL 由一家瑞典公司 MySQL AB 开发、运营并予以支持。它之所以非常流行，原因在于具备以下这些优点：

- 基于开源许可发布，无需付费即可使用。
- 自身的功能非常强大，足以匹敌绝大多数功能强大但却价格昂贵的数据库软件。
- 使用业内所熟悉的标准SQL数据库语言。
- 可运行于多个操作系统，支持多种语言，包括 PHP、PERL、C、C++ 及 Java 等语言。
- 非常迅速，即使面对大型数据集也毫无滞涩。
- 非常适用于 PHP 这种 Web 开发者最喜欢使用的语言。
- 支持大型数据库，最高可在一个表中容纳 5千多万行。每张表的默认文件大小限制为 4GB，不过如果操作系统支持，你可以将其理论限制增加到 800 万 TB。
- 可以自定义。开源 GPL 许可保证了程序员可以自由修改 MySQL，以便适应各自特殊的开发环境。

准备须知

在开始学习本系列教程之前，你应该通过我们的教程简单地了解一下 PHP 和 HTML 方面的相关知识。

本教学重点在于利用 PHP 使用 MySQL，所以很多实例对于 PHP 程序员来说非常实用。

对于不熟悉 PHP 的读者，我们强烈建议您读一读 PHP教学 (http://www.tutorialspoint.com/php/index.htm) 。

MySQL 安装

下载 MySQL

MySQL 的全部下载链接都在这个页面：MySQL 下载 (http://www.mysql.com/downloads/) 。选择所需的 *MySQL Community Server* 版本号，并且尽可能准确地选择所需平台。

在 Linux/Unix 上安装 MySQL

在 Linux 系统上安装 MySQL，建议采用 RPM 形式进行安装。MySQL AB 在其网站上提供了以下几种 RPM 文件包：

- MySQL MySQL 数据库服务器，用于管理数据库与表，控制用户访问，以及处理 SQL 查询。
- MySQL-client MySQL 客户端程序，实现用户与服务器的连接与交互功能。
- MySQL-devel 编译使用 MySQL 的其他程序的过程中会用到的一些库及头文件。
- MySQL-shared MySQL 客户端的共享库。
- MySQL-bench 用于MySQL 服务器的基准测试与性能测试工具。

这里列出的 MySQL RPM 都是基于 Linux 的 SuSE 分发版系统构建的，但它们一般也能轻松地运行在其他 Linux 变种系统上。

接着按照以下步骤完成安装：

- 使用 root 用户登录系统。
- 切换到含有 RPM 文件包的目录中。
- 执行下面命令，安装 MySQL 服务器。记住，用你自己的 RPM 文件名替换命令中斜体标识的文件名：

  ```
  [root@host]# rpm -i MySQL-5.0.9-0.i386.rpm
  ```

  上面的命令安装 MySQL 服务器，创建了一个 MySQL 用户，进行了必要的配置，并开始自动启动 MySQL 服务器。

  在 /usr/bin 与 /usr/sbin 可找到 MySQL 所有的相关库。创建的所有的表和数据库都在 /var/lib/mysql 目录下。

- 安装剩下的RPM，可参照下列命令（但建议采用这种方式）进行：

```
[root@host]# rpm -i MySQL-client-5.0.9-0.i386.rpm [root@host]# rpm -i MySQL-devel-5.0.9-0.i386.rpm [root@host]# rpm -i MySQL-shared-5.0.9-0.i386.rpm [root@host]# rpm -i MySQL-bench-5.0.9-0.i386.rpm
```

在 Windows 下安装 MySQL

现在，Windows 系统下的 MySQL 默认安装方式都比过去容易多了，因为已经利用安装程序将所需的 MySQL 内容精心打包起来。只需下载安装程序包，随便将它解压缩在某个目录，然后运行 setup.exe 就可以了。

默认的安装程序 setup.exe 能帮你打理琐碎的安装过程，同时默认安装在 C:\mysql 目录下。

首次测试服务器，可以采用命令行方式。找到 mysqld 服务器的位置（可能位于 C:\mysql\bin），输入如下命令：

```
mysqld.exe --console
```

> 注意：如果是 NT 系统，就不能使用 mysqld.exe 了，必须使用 mysqld-nt.exe。

不出意外的话，你就会看到一些关于启动和 InnoDB 的信息。如果没有出现这类信息，那么可能是因为你的许可权限有问题。确保所有用户（可能是 mysql）都能访问存储数据的目录。

MySQL 不会自动将其自身添加到开始菜单中，而且目前也没有一些比较好的能够用来停止服务的GUI。因此，假如你喜欢通过双击 mysqld 可执行文件来启动服务器，那么当要关闭服务器时，记得要手动借助 mysqladmin、任务列表、任务管理器或者 Windows 的一些专用方法来进行。

MySQL 安装验证

成功安装完 MySQL 后，就会初始化基表，启动服务器。可以通过一些简单的测试来验证安装是否一切正常。

使用 mysqladmin 工具来获取服务器状态

使用 mysqladmin 工具来查看服务器版本。在Linux下，这一工具位于 /usr/bin；Windows下则在C:\mysql\bin。

```
[root@host]# mysqladmin --version
```

在 Linux 下，上述命令可能会产生如下结果。根据你安装的 Linux 版本的差异，结果也可能会有些许不同。

```
mysqladmin Ver 8.23 Distrib 5.0.9-0, for redhat-linux-gnu on i386
```

如果没有显示类似这样的信息，则说明安装可能出现了一些问题，需要借助一些帮助来修补它们。

使用MySQL客户端来执行简单的SQL命令

你可以通过在MySQL客户端上使用 **mysql** 命令去连接 MySQL 服务器。这时，不需要输入任何密码，因为默认情况下会设置为空白。

所以只需输入如下命令即可：

```
[root@host]# mysql
```

系统应该显示出 mysql> 提示符，这就表明你已经连接上了 MySQL 服务器，可以在提示符后输入一些 SQL 命令了，如下所示：

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
2 rows in set (0.13 sec)
```

安装后的一些步骤

对于根用户，MySQL初始是不需要密码的。一旦成功安装好数据库和客户端后，你就需要设置一个根用户密码，如下所示：

```
[root@host]# mysqladmin -u root password "new_password";
```

接着连接MySQL服务器，就会要求你输入密码了：

```
[root@host]# mysql -u root -p
Enter password:*******
```

对于UNIX用户来说，同样也必须把MySQL目录放入PATH环境变量中，这样在使用命令行客户端时，就不必每次手动输入路径全称了。对于bash shell 来说，应该这样设置：

```
export PATH=$PATH:/usr/bin:/usr/sbin
```

启动时运行 MySQL

如果想让 MySQL 在系统启动时自动运行，则可以 /etc/rc.local 文件中加入下列项：

```
/etc/init.d/mysqld start
```

另外，在 etc/init.d/ 目录中必须存在 mysqld 工具。

MySQL 管理

运行与关闭 MySQL 服务器

首先检查 MySQL 服务器是否正在运行。可以使用下列命令来确认这一点：

```
ps –ef | grep mysqld
```

如果 MySQL 正在运行，在上述命令的运行结果中就能看到 mysqld 进程。如果服务器没有运行，使用下列命令来启动它：

```
root@host# cd /usr/bin
./safe_mysqld &
```

如果想关闭正在运行的 MySQL 服务器，使用如下命令即可：

```
root@host# cd /usr/bin
./mysqladmin –u root –p shutdown
Enter password: ******
```

建立 MySQL 用户账号

添加新的 MySQL 用户，只需在数据库 mysql 的 user 表中添加一个新项即可。

在以下范例中，添加了一个新用户 guest，该用户具有 SELECT、INSERT、UPDATE 权限，密码是 guest123。SQL 查询如下：

```
root@host# mysql –u root –p
Enter password:*******
mysql> use mysql;
Database changed

mysql> INSERT INTO user
        (host, user, password,
         select_priv, insert_priv, update_priv)
         VALUES ('localhost', 'guest',
         PASSWORD('guest123'), 'Y', 'Y', 'Y');
Query OK, 1 row affected (0.20 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 1 row affected (0.01 sec)
```

```
mysql> SELECT host, user, password FROM user WHERE user = 'guest';
+-----------+---------+------------------+
| host      | user    | password         |
+-----------+---------+------------------+
| localhost | guest | 6f8c114b58f2ce9e |
+-----------+---------+------------------+
1 row in set (0.00 sec)
```

在添加新用户时，记住要用 MySQL 提供的 `PASSWORD()` 函数对该用户的密码进行加密处理。如上例所示，密码 mypass 被加密成了 6f8c114b58f2ce9e。

注意这里所用的 FLUSH PRIVILEGES 语句。它让服务器重新加载授权表。如果不使用它，就至少得等到服务器重新启动后，才能使用新用户账号连接 mysql。

你也可以为新用户指定其他权限，在执行 INSERT 查询时，将用户表中的下面这些列的值都设为 ‘Y’，或者使用 UPDATE 查询稍后对它们进行更新。

- Select_priv
- Insert_priv
- Update_priv
- Delete_priv
- Create_priv
- Drop_priv
- Reload_priv
- Shutdown_priv
- Process_priv
- File_priv
- Grant_priv
- References_priv
- Index_priv
- Alter_priv

另外一种添加用户账号的方式是使用 SQL命令 GRANT。下面这个例子将在数据库 TUTORIALS 上添加一个名为 zara 的新用户，其密码为 zara123。如下所示：

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use mysql;
Database changed

mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    -> ON TUTORIALS.*
    -> TO 'zara'@'localhost'
    -> IDENTIFIED BY 'zara123';
```

这会在 mysql 数据库的 user 表中创建一个项。

> 注意：如果 SQL 命令不以分号（ ; ）结束的话，MySQL 就不会终止这个命令。

配置 /etc/my.cnf 文件

大多数情况下，根本用不到这个文件。默认状态下，它应该包含如下项：

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock

[mysql.server]
user=mysql
basedir=/var/lib

[safe_mysqld]
err-log=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

在这里，可以为 error log 更换不同的目录。另外，不要更改这张表中的其他项。

用于管理 MySQL 的一些命令

下面列出了一些重要且经常会用到的MySQL命令：

- USE *Databasename* 用于在MySQL工作区内选择具体某个数据库。
- SHOW DATABASES 列出 MySQL DBMS 所能访问的数据库。

- SHOW TABLES 一旦数据库被 use 命令选中，显示数据库中的表。
- SHOW COLUMNS FROM *tablename* 显示表的属性、属性类型、键信息、是否允许 NULL 值，默认值，以及其他一些信息。
- SHOW INDEX FROM *tablename* 显示表中所有索引的细节信息，包括PRIMARY KEY。
- SHOW TABLE STATUS LIKE *tablename*\G 报告MySQL DBMS的性能及统计的细节信息。

MySQL PHP语法

MySQL 可以很好地适用于多种编程语言，比如PERL、C、C++、JAVA，以及 PHP。由于可以开发 Web 应用程序，PHP 是其中最流行的一门语言。

本教程讲解重点在于PHP 环境中使用 MySQL。如果你对使用 PERL 来操作 MySQL 有兴趣，可以参看这个教程：**PERL 与 MySQL 教程**

PHP 提供了多种能够访问 MySQL 数据库并且操纵数据记录的函数。这些函数的调用方式就跟其他 PHP 函数一样。

PHP 中用于操作 MySQL 的函数一般都采取如下的格式：

```
mysql_function(value,value,...);
```

函数名称的第二部分是函数所专有的，通常是一个描述函数行为的词。下面是教程中将会用到的两个函数：

```
mysqli_connect($connect);
mysqli_query($connect,"SQL statement");
```

下面这个例子展示的是PHP调用MySQL函数的常见语法格式：

```
<html>
<head>
<title>PHP with MySQL</title>
</head>
<body>
<?php
  $retval = mysql_function(value, [value,...]);
  if( !$retval )
  {
     die ( "Error: a related error message" );
  }
  // Otherwise MySQL  or PHP Statements
?>
</body>
</html>
```

从下一节开始，我们将介绍与 PHP 相关的 MySQL 功能。

连接 MySQL 服务器

可以在命令行方式下使用 mysql 命令建立 MySQL数据库。

范例:

下面这个例子显示如何采用命令行方式连接 MySQL 服务器:

```
[root@host]# mysql -u root -p
Enter password:******
```

上述命令将显示 mysql> 命令提示符。在该命令提示符后面，可以执行任何 SQL 命令。下面就是上述命令的显示结果:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2854760 to server version: 5.0.9

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

在上面这个例子中，使用 root 作为用户（你也可以使用其他用户）。任何用户都能执行 root 用户所能执行的全部 SQL 操作。

无论何时，只要在 mysql> 提示符下输入 `exit` ，就能随时中断与 MySQL 的连接。

```
mysql> exit
Bye
```

使用 PHP 脚本连接 MySQL

通过 PHP 的 `mysql_connect()` 函数，可以开启数据库连接。该函数有5个参数。当成功连接后，该函数返回一个 MySQL 连接标识；如连接失败，则返回FALSE。

语法格式

```
connection mysql_connect(server,user,passwd,new_link,client_flag);
```

参数	说明
server	可选参数。运行数据库服务器的主机名。如未指定，则默认值 localhost:3036。

参数	说明
user	可选参数。访问数据库的用户名。如未指定，则默认值为拥有服务器进程的用户名称。
passwd	可选参数。用户访问数据库所用密码。如未指定，则默认没有密码。
new_link	可选参数。如果利用同样的参数第二次调用 mysql_connect() ，则不会建立新的连接，而是返回已打开连接的标识。
client_flags	可选参数。是由下列常量组合而成： • MYSQL_CLIENT_SSL——使用 SSL 加密。 • MYSQL_CLIENT_COMPRESS——使用数据压缩协议。 • MYSQL_CLIENT_IGNORE_SPACE——允许函数名后出现空格。 • MYSQL_CLIENT_INTERACTIVE——关闭连接之前所空闲等候的交互超时秒数。

通过 PHP 的 mysql_close() 函数，随时可以中断与 MySQL 数据库的连接。该函数只有一个参数，是一个由 mysql_connect()函数所返回的连接。

语法格式

```
bool mysql_close ( resource $link_identifier );
```

如果某个资源未被指定，则最后打开的数据库就会被关闭。如果成功中断连接，该函数返回 true，否则返回 false。

范例

下面通过一个具体的范例来实际了解如何连接 MySQL 服务器。

```
<html>
<head>
<title>Connecting MySQL Server</title>
</head>
<body>
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'guest';
  $dbpass = 'guest123';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);
  if(! $conn )
  {
    die('Could not connect: ' . mysql_error());
  }
  echo 'Connected successfully';
  mysql_close($conn);
```

```
?>
</body>
</html>
```

创建 MySQL 数据库

使用 mysqladmin 创建数据库

创建或删除数据库需要拥有特殊的权限。假设你获得了root用户权限，那么利用 mysqladmin 二进制命令可以创建任何数据库。

范例

下面就来创建一个名叫 TUTORIALS 的数据库：

```
[root@host]# mysqladmin -u root -p create TUTORIALS
Enter password:******
```

通过上述命令，就创建好了一个名叫 TUTORIALS 的 MySQL 数据库。

利用PHP脚本创建数据库

PHP利用 mysql_query 函数来创建或删除 MySQL 数据库。该函数有2个参数，成功执行操作则返回TRUE，失败则返回FALSE。

语法

```
bool mysql_query( sql, connection );
```

参数	说明
sql	必需参数。创建或删除 MySQL 数据库所用的 SQL 命令。
connection	可选参数。如未指定，将使用最后一个由 mysql_connect 打开的连接。

范例

通过下面这个范例来了解如何创建数据库。

```
<html>
<head>
```

```
<title>Creating MySQL Database</title>
</head>
<body>
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully<br />';
$sql = 'CREATE DATABASE TUTORIALS';
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not create database: ' . mysql_error());
}
echo "Database TUTORIALS created successfully\n";
mysql_close($conn);
?>
</body>
</html>
```

MySQL 终止数据库

利用 mysqladmin 删除 MySQL 数据库

同上一节的情况完全一样，创建或删除 MySQL 数据库需要特殊的权限。假如有了 root 用户权限，那就可以用 mysqladmin 二进制命令来随意创建数据库了。

删除数据库要非常谨慎，因为这样做会丢失数据库中所保存的全部数据。

在下面这个范例中，删除了上一节所创建的数据库。

```
[root@host]# mysqladmin -u root -p drop TUTORIALS
Enter password:******
```

这时，系统会显示一条警示消息，询问是否确定要删除数据库。

```
Dropping the database is potentially a very bad thing to do.
Any data stored in the database will be destroyed.

Do you really want to drop the 'TUTORIALS' database [y/N] y
Database "TUTORIALS" dropped
```

使用 PHP 脚本删除数据库

PHP 使用 mysql_query 函数来创建或删除 MySQL 数据库。该函数包含两个参数，如果成功执行操作，返回 TRUE，否则返回 FALSE。

语法格式

```
bool mysql_query( sql, connection );
```

范例

下面这个范例展示了如何删除一个数据库。

```
<html>
<head>
```

```
<title>Deleting MySQL Database</title>
</head>
<body>
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully<br />';
$sql = 'DROP DATABASE TUTORIALS';
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not delete database: ' . mysql_error());
}
echo "Database TUTORIALS deleted successfully\n";
mysql_close($conn);
?>
</body>
</html>
```

> **警告**：在利用 PHP 脚本删除数据库时，它不会提供任何确认提示，因此一定要小心。

MySQL 选择数据库

一旦连接上了 MySQL 服务器，就需要选择一个具体的用来运行的数据库。这是因为，有可能会有多个数据库挂接在MySQL服务器上。

利用命令行方式选择 MySQL 数据库

通过 mysql> 提示符来选择数据库是一种非常简单的方法。可以使用 SQL 命令 use 来选择某个数据库。

范例

下面这个范例展示了如何选择一个名为 TUTORIALS 的数据库。

```
[root@host]# mysql -u root -p
Enter password:******
mysql> use TUTORIALS;
Database changed
mysql>
```

这样就能选择TUTORIALS数据库，所有后续操作都将在TUTORIALS数据库上进行。

> 注意：所有的数据库名称、表名、表字段名都是对大小写敏感的，因此使用SQL命令时，必须要使用正确的名称。

使用PHP脚本选择MySQL数据库

PHP 通过 mysql_select_db 函数来选择数据库。如果成功完成操作，返回 TRUE，否则返回 FALSE。

语法格式

```
bool mysql_select_db( db_name, connection );
```

参数	说明
db_name	必需参数。要选择的 MySQL 数据库名称。
connection	可选参数。如未指定，则将使用 mysql_connect 最后打开的一个连接。

范例

下面这个范例展示如何选择数据库。

```
<html>
<head>
<title>Selecting MySQL Database</title>
</head>
<body>
<?php
$dbhost = 'localhost:3036';
$dbuser = 'guest';
$dbpass = 'guest123';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_select_db( 'TUTORIALS' );
mysql_close($conn);
?>
</body>
</html>
```

MySQL 数据类型

对于数据库的整体优化来说，正确定义表中的字段是非常关键的。应该只采用字段》。如果事先知道只会用到2个字符的宽度，就不要把字段定义为10个字符宽。字段（或者说列）的类型也被称为数据类型。

MySQL使用的多种数据类型可分为三类：数字、日期与时间，以及字符串类型。

数字类型

MySQL使用标准的 ANSI SQL 数字类型，所以如果你在学习MySQL之前，接触过其他数据库系统，那么肯定对这些定义不会感到陌生。下面就列举出常见的一些数字类型及其说明：

- INT 正常大小的整数，可以有符号，也可以没有符号。如果是有符号整数，其允许的取值范围是-2147483648~2147483647；无符号整数的取值范围是从0至4294967295。最高可指定11位数字。
- TINYINT 非常小的整数，分为有无符号两种。前有符号时，其允许取值范围是-128~127；无符号时的取值范围为0~255。所以，最高可指定4位数字。
- SMALLINT 较小的整数，分为有无符号两种。前有符号时，其允许取值范围是-32768~32767；无符号时的取值范围为0~65535。所以最高可指定5位数字。
- MEDIUMINT 中型大小的整数，分为有无符号两种。前有符号时，其允许取值范围是-8388608~8388607；无符号时的取值范围为0~16777215。所以，最高可指定9位数字。
- BIGINT 较大型的整数，分为有无符号两种。前有符号时，其允许取值范围为-9223372036854775808~9223372036854775807；无符号时的取值范围为0~18446744073709551615。最高可指定20位数字。
- FLOAT(M,D) 不带符号的浮点数。M 代表显示长度，D 代表小数位数。这两个参数都不是必需参数，它们默认为10, 2，表示小数点后有2位数字，而整个数字的位数为10（包含小数位数）。FLOAT 类型的小数精度可以达到24位。
- DOUBLE(M,D) 不带符号的双精度浮点数。M 代表显示长度，D 代表小数位数。这两个参数都不是必需参数，它们默认为16, 4，表示小数点后有4位数字，而整个数字的位数为 16（包含小数位数）。DOUBLE 类型的小数精度可以达到53位。DOUBLE 与 REAL 同义。
- DECIMAL(M,D) 非压缩的无符号浮点数。 在未压缩十进制中，每一位十进制数都对应一个字节。需要定义显示长度（M）和小数位数（D）。DECIMAL 与 NUMERIC 同义。

日期与时间类型

MySQL 包含以下几种日期与时间类型：

- DATE YYYY-MM-DD（年-月-日）格式显示的日期，取值范围从1000-01-01 到 9999-12-31。比如1973年的12月30日就存为 1973-12-30。
- DATETIME 按照 YYYY-MM-DD HH:MM:SS 格式组合显示的日期与时间，取值范围从1000-01-01 00:00:00 到 9999-12-31 23:59:59。比如说1973年的12月30日下午3 : 30就存为1973-12-30 15 : 30 : 00。
- TIMESTAMP 介于1970年1月1日凌晨与2037年某个时间点之间的一种时间戳。这种格式与之前的 DATETIME 格式相仿，只不过少了数字间的连字符。1973年12月30日下午3 : 30被存为19731230153000（YYYYMMDDHHMMSS）。
- TIME 按照 HH:MM:SS 格式存储的时间。
- YEAR(M) 用2位或4位格式存储的时间。如果把长度定为2，比如说YEAR(2)，那么可以表示从1970年到2069年的这些年份（70-69）。如果把长度定为4，YEAR(4)，则可以表示从1901年到2155年。默认长度为4。

字符串类型

虽然数字与日期类型都很有趣，但通常我们存储最多的就是字符串了。下面列出了 MySQL 中常见的字符串类型。

- CHAR(M) 长度固定的字符串，长度范围从1~255个字符，比如CHAR(5)。在存储时，会向右用空格补齐指定长度。长度并非必须参数，默认长度为1。
- VARCHAR(M) 长度不定的字符串，长度范围从1~255个字符。比如：CHAR(25)。在创建VARCHAR字段时，必须定义长度。
- BLOB or TEXT 最大长度为65535个字符的字段。BLOB是Binary Large Objects（二进制大型对象）的缩写，专用于保存大量的二进制数据，比如图片或其他类型的文件。TEXT 类型的文件也能保存大型数据。这两者的区别在于存储数据的排序和对比方面，BLOB类型数据是大小写敏感的，而TEXT类型数据则不是。另外，不能指定它们的长度。
- TINYBLOB or TINYTEXT 最大长度为255个字符的 BLOB 或 TEXT 字段。同样也不能指定它们的长度。
- MEDIUMBLOB or MEDIUMTEXT 最大长度为16777215个字符的 BLOB 或 TEXT 字段。同样也不能指定它们的长度。

- LONGBLOB or LONGTEXT 最大长度为4294967295个字符的 BLOB 或 TEXT 字段。同样也不能指定它们的长度。
- ENUM 枚举类型，是一种很独特的列表类型。ENUM 类型的数据实际是一个包含多个固定值的列表，只能选择这些值（包括 NULL 值）。例如，如果希望某个字段包含 "A"、"B" 和 "C"，必须这样定义：`ENUM ('A', 'B', 'C')`，只有这些值（或 NULL 值）能够填充到该字段中。

创建 MySQL 表

创建表的命令需要：

- 表名
- 字段名
- 每一字段的定义

语法格式

下面就是一种常见的用来创建 MySQL 表的 SQL 语法。

```
CREATE TABLE table_name (column_name column_type);
```

然后在 TUTORIALS 数据库创建如下表：

```
tutorials_tbl(
   tutorial_id INT NOT NULL AUTO_INCREMENT,
   tutorial_title VARCHAR(100) NOT NULL,
   tutorial_author VARCHAR(40) NOT NULL,
   submission_date DATE,
   PRIMARY KEY ( tutorial_id )
);
```

这里需要解释的项目是：

- 使用字段属性 NOT NULL 是因为我们不想让该字段为空值。所以如果用户尝试创建空值记录，MySQL 就会抛出一个错误。
- 字段属性 AUTO_INCREMENT 告诉 MySQL 继续为 id 字段增加下一个可能的数值。
- 关键字 PRIMARY KEY 会将一列定义为主键。也可以使用由逗号分隔的多个列来定义主键。

通过命令行方式创建表

通过命令行来创建 MySQL 表是非常简单的一种方式。使用 SQL命令 CREATE TABLE 即可创建一个表。

范例

在下面这个范例中，创建了表 tutorials_tbl。

```
root@host# mysql -u root -p
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> CREATE TABLE tutorials_tbl(
   -> tutorial_id INT NOT NULL AUTO_INCREMENT,
   -> tutorial_title VARCHAR(100) NOT NULL,
   -> tutorial_author VARCHAR(40) NOT NULL,
   -> submission_date DATE,
   -> PRIMARY KEY ( tutorial_id )
   -> );
Query OK, 0 rows affected (0.16 sec)
mysql>
```

> 注意：只有在SQL命令末尾加上分号（ ; ）才能终止这个命令。

利用 PHP 脚本创建表

在已有的数据库中创建新表，可以使用 PHP 的 mysql_query() 函数。利用正确的SQL命令为其传入第二个参数，就能创建出一张表。

范例

以下范例展示如何利用 PHP 脚本来创建表。

```
<html>
<head>
<title>Creating MySQL Tables</title>
</head>
<body>
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
```

```
  die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully<br />';
$sql = "CREATE TABLE tutorials_tbl( ".
       "tutorial_id INT NOT NULL AUTO_INCREMENT, ".
       "tutorial_title VARCHAR(100) NOT NULL, ".
       "tutorial_author VARCHAR(40) NOT NULL, ".
       "submission_date DATE, ".
       "PRIMARY KEY ( tutorial_id )); ";
mysql_select_db( 'TUTORIALS' );
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not create table: ' . mysql_error());
}
echo "Table created successfully\n";
mysql_close($conn);
?>
</body>
</html>
```

MySQL 下拉表

删除已有的 MySQL 表是很容易的，但你要非常小心，因为删除了表，就无法恢复数据了。

语法

删除 MySQL 表的常用 SQL 命令为：

```
DROP TABLE table_name ;
```

通过命令行方式删除表

只需在命令行中使用 DROP TABLE 这个SQL命令即可。

范例

在以下范例中，删除了表 `tutorials_tbl` 。

```
root@host# mysql -u root -p
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> DROP TABLE tutorials_tbl
Query OK, 0 rows affected (0.8 sec)
mysql>
```

利用 PHP 脚本删除表

要想利用 PHP 脚本删除数据库中的表，需要使用函数 mysql_query()。将正确的 SQL 命令传入该函数的第二个参数中，就能将表删除。

范例

```
<html>
<head>
<title>Creating MySQL Tables</title>
```

```
</head>
<body>
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully<br />';
$sql = "DROP TABLE tutorials_tbl";
mysql_select_db( 'TUTORIALS' );
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not delete table: ' . mysql_error());
}
echo "Table deleted successfully\n";
mysql_close($conn);
?>
</body>
</html>
```

MySQL 插入查询

要想在 MySQL 表中插入数据，需要使用 INSERT INTO 这个 SQL 命令。既可以使用mysql> 提示符方式，也可以使用 PHP 等脚本来完成。

语法格式

利用 INSERT INTO 命令为表插入数据的一般语法如下所示：

```
INSERT INTO table_name ( field1, field2,...fieldN )
                       VALUES
                       ( value1, value2,...valueN );
```

要想插入字符串类型的数据，需要把值用双引号或单引号包括起来，比如：`"value"`。

从命令提示符中插入数据

我们将使用 INSERT INTO 命令为表 tutorials_tbl 插入数据。

范例

在下面的范例中，我们将为表 `tutorials_tbl` 创建3条记录。

```
root@host# mysql -u root -p password;
Enter password:********
mysql> use TUTORIALS;
Database changed
mysql> INSERT INTO tutorials_tbl
    ->(tutorial_title, tutorial_author, submission_date)
    ->VALUES
    ->("Learn PHP", "John Poul", NOW());
Query OK, 1 row affected (0.01 sec)
mysql> INSERT INTO tutorials_tbl
    ->(tutorial_title, tutorial_author, submission_date)
    ->VALUES
    ->("Learn MySQL", "Abdul S", NOW());
Query OK, 1 row affected (0.01 sec)
mysql> INSERT INTO tutorials_tbl
    ->(tutorial_title, tutorial_author, submission_date)
```

```
   ->VALUES
   ->("JAVA Tutorial", "Sanjay", '2007-05-06');
Query OK, 1 row affected (0.01 sec)
mysql>
```

> 注意：代码中的箭头符号（ `->` ）不属于 SQL 命令。它们只是用来表示换行，如果我们在每行命令末尾不添加分号（ `;` ），按下回车键时，MySQL 命令提示符就会自动创建出这个符号。

在上面的范例中，我们没有提供 tutorial_id，因为在创建表时，已为该字段提供了 AUTO_INCREMENT 选项。因此 MySQL 会自动知道插入 ID。 `NOW()` 是一个能够返回当前日期与时间的 MySQL 函数。

利用 PHP 脚本插入数据

同样，也可以在 PHP 的 **mysql_query()**函数中使用 SQL 命令 INSERT INTO 为 MySQL 表插入数据。

范例

在下面这个范例中，从用户那里接收3个参数，然后将这些参数插入到 MySQL 表中。

```
<html>
<head>
<title>Add New Record in MySQL Database</title>
</head>
<body>
<?php
if(isset($_POST['add']))
{
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
    die('Could not connect: ' . mysql_error());
}

if(! get_magic_quotes_gpc() )
{
  $tutorial_title = addslashes ($_POST['tutorial_title']);
  $tutorial_author = addslashes ($_POST['tutorial_author']);
}
else
{
```

```
   $tutorial_title = $_POST['tutorial_title'];
   $tutorial_author = $_POST['tutorial_author'];
}
$submission_date = $_POST['submission_date'];

$sql = "INSERT INTO tutorials_tbl ".
       "(tutorial_title,tutorial_author, submission_date) ".
       "VALUES ".
       "('$tutorial_title','$tutorial_author','$submission_date')";
mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
     die('Could not enter data: ' . mysql_error());
}
echo "Entered data successfully\n";
mysql_close($conn);
}
   else
{
?>
<form method="post" action="<?php $_PHP_SELF ?>">
<table width="600" border="0" cellspacing="1" cellpadding="2">
<tr>
<td width="250">Tutorial Title</td>
<td>
<input name="tutorial_title" type="text" id="tutorial_title">
</td>
</tr>
<tr>
<td width="250">Tutorial Author</td>
<td>
<input name="tutorial_author" type="text" id="tutorial_author">
</td>
</tr>
<tr>
<td width="250">Submission Date [ yyyy-mm-dd ]</td>
<td>
<input name="submission_date" type="text" id="submission_date">
</td>
</tr>
<tr>
<td width="250"> </td>
<td> </td>
</tr>
```

```
<tr>
<td width="250"> </td>
<td>
<input name="add" type="submit" id="add" value="Add Tutorial">
</td>
</tr>
</table>
</form>
<?php
}
?>
</body>
</html>
```

在输入数据时，使用函数 get_magic_quotes_gpc() 检查当前是否配置了魔术引号。如果函数返回 false，则使用 addslashes() 在引号前添加反斜杠。

另外，还可以针对输入数据进行多种验证，确保数据的合法性，并对其采取正确行为。

MySQL 选择查询

SQL 的 SELECT 命令用于从 MySQL 数据库中获取数据。可以在mysql> 提示符中使用这一命令，也可以利用 PHP 等脚本来完成。

语法格式

利用 SELECT 命令从 MySQL 表中获取数据的一般语法格式如下：

```
SELECT field1, field2,...fieldN table_name1, table_name2...
[WHERE Clause]
[OFFSET M ][LIMIT N]
```

- 可以通过逗号分隔一个或多个表，利用 WHERE 子句包含进多种条件，但 WHERE 子句并不是 SELECT 命令的可选部分。
- 可以在一个 SELECT 命令中获取一个或多个字段。
- 可以用星型符号（ * ）代替字段。这时，SELECT 将返回所有字段。
- 可以使用 WHERE 子句指定任何条件。
- 在 SELECT 将要返回记录的位置处，使用 OFFSET 可以指定一个偏移。默认偏移为0。
- 可以使用 LIMIT 属性来限制返回记录的数量。

利用命令行方式获取数据

使用 SELECT 命令从表 tutorials_tbl 中获取数据。

范例

下面这个范例将返回表 tutorials_tbl 中的所有记录。

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> SELECT * from tutorials_tbl
+-------------+----------------+-----------------+-----------------+
```

```
| tutorial_id | tutorial_title | tutorial_author | submission_date |
+-------------+----------------+-----------------+-----------------+
|           1 | Learn PHP      | John Poul       | 2007-05-21      |
|           2 | Learn MySQL    | Abdul S         | 2007-05-21      |
|           3 | JAVA Tutorial  | Sanjay          | 2007-05-21      |
+-------------+----------------+-----------------+-----------------+
3 rows in set (0.01 sec)

mysql>
```

使用 PHP 脚本获取数据

同样，也可以在 mysql_query() 函数中使用 SQL 命令 SELECT。该函数用于执行 SQL 命令。随后另一个 PHP 函数 mysql_fetch_array() 会获取所有选定的数据，该函数会将行以关联数组或数值数组返回，或者还可能同时返回以上两种形式。如果再也没有行，则该函数返回 FALSE。

下面通过一个简单的范例来了解如何获取 tutorials_tbl 表中的记录。

范例

下面这个范例会获取表 tutorials_tbl 中的所有记录。

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT tutorial_id, tutorial_title,
              tutorial_author, submission_date
        FROM tutorials_tbl';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
```

```
{
   echo "Tutorial ID :{$row['tutorial_id']}  <br> ".
        "Title: {$row['tutorial_title']} <br> ".
        "Author: {$row['tutorial_author']} <br> ".
        "Submission Date : {$row['submission_date']} <br> ".
        "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

行的内容被赋给变量 `$row`，随后输出行内所包含的值。

> 注意：一定要记住，在把一个数组值直接插入到字符串中时，一定要加花括号（`{` 与 `}`）。

在上面的例子中，常量 `MYSQL_ASSOC` 被用作 PHP 函数 `mysql_fetch_array()` 的第二个参数，因此才会将行按照关联数组的形式返回。利用关联数组，我们可以使用字段的名字来访问字段，而不需要用到索引。

PHP 还提供了另一个叫做 `mysql_fetch_assoc()` 的函数，也会将行以关联数组的形式返回。

范例

在下面的范例中，利用 `mysql_fetch_assoc()` 函数来显示 tutorials_tbl 表中的所有记录。

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT tutorial_id, tutorial_title,
              tutorial_author, submission_date
        FROM tutorials_tbl';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_assoc($retval))
{
```

```
   echo "Tutorial ID :{$row['tutorial_id']}  <br> ".
        "Title: {$row['tutorial_title']} <br> ".
        "Author: {$row['tutorial_author']} <br> ".
        "Submission Date : {$row['submission_date']} <br> ".
        "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

你可以使用常量 MYSQL_NUM 作为 `mysql_fetch_array()` 函数的第二个参数。这能让函数返回一个带有数字索引的数组。

范例

在下面的范例中，使用参数 MYSQL_NUM 来显示表 tutorials_tbl 中的所有记录。

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT tutorial_id, tutorial_title,
              tutorial_author, submission_date
        FROM tutorials_tbl';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_NUM))
{
   echo "Tutorial ID :{$row[0]}  <br> ".
        "Title: {$row[1]} <br> ".
        "Author: {$row[2]} <br> ".
        "Submission Date : {$row[3]} <br> ".
        "--------------------------------<br>";
}
```

```
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

以上三个范例都会产生同样的结果。

释放内存

在每个 SELECT 语句末尾释放游标内存是一个非常好的做法。使用 PHP 函数 mysql_free_result() 就可以实现这一点，如下例所示。

范例

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT tutorial_id, tutorial_title,
              tutorial_author, submission_date
        FROM tutorials_tbl';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_NUM))
{
    echo "Tutorial ID :{$row[0]}  <br> ".
         "Title: {$row[1]} <br> ".
         "Author: {$row[2]} <br> ".
         "Submission Date : {$row[3]} <br> ".
         "--------------------------------<br>";
}
mysql_free_result($retval);
echo "Fetched data successfully\n";
```

```
mysql_close($conn);
?>
```

在获取数据时，还可以用更复杂的SQL语句。步骤和上面介绍的一样。

MySQL Where Clause

前面介绍了利用 SELECT 命令从表中获取数据。利用一种叫做 WHERE 子句的条件子句可以过滤结果。使用 WHERE 子句可以制定选择规则，从表中选择我们所需的记录。

语法格式

利用 SELECT 命令与 WHERE 子句从表中获取数据的一般语法格式如下所示：

```
SELECT field1, field2,...fieldN table_name1, table_name2...
[WHERE condition1 [AND [OR]] condition2.....
```

- 可以使用逗号分隔一个或多个表，从而在使用 WHERE 子句时，包含多个条件。但是 WHERE 子句并非 SELECT 命令的一个可选部分。
- 可以在使用 WHERE 子句时指定任何条件。
- 可以通过 AND 或 OR 运算符来指定一个或多个条件。
- WHERE 子句可以和 DELETE 或 UPDATE 命令一起使用，同样是用于指定条件。

WHERE 子句的运作方式就像编程语言中的 if 条件语句一样。它会将给定值与MySQL表中的字段值进行对比，如果两值相等，则返回MySQL表中字段值的所在的行。

下面这张表列出了 WHERE 子句中使用的一些运算符。

假设字段 A 保存 10 这个值，字段 B 保存着 20，那么：

运算符	说明	范例
=	检查两个操作数是否相等。如果相等，则条件为真。	(A = B) 非真
!=	检查两个操作数是否相等。如果不相等，则条件为真。	(A != B) 为真
>	检查左侧操作数是否大于右边的操作数。如果大于，则条件为真。	(A > B) 非真
<	检查左侧操作数是否小于右侧操作数。如果小于，则条件为真。	(A < B) 为真
>=	检查左侧操作数是否大于或等于右侧操作数。如果是，则条件为真。	(A >= B) 非真
<=	检查左侧操作数是否小于或等于右侧操作数。如果是，则条件为真	(A <= B) 为真

WHERE 子句可以非常方便地获取表中选定的行，尤其与 MySQL 的 Join 一起使用时。Join 稍后将择章另述。

另外，为了加快搜索，往往使用主键搜索记录。

如果表中记录并不匹配给定条件，查询不会返回任何数据。

采用命令行方式获取数据

使用 SQL 的 SELECT 命令与 WHERE 子句获取 表 tutorials_tbl 中的选定数据。

范例

以下范例将返回表 tutorials_tbl 中作者名称（author name）为 Sanjay 的所有记录。

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> SELECT * from tutorials_tbl WHERE tutorial_author='Sanjay';
+-------------+----------------+-----------------+-----------------+
| tutorial_id | tutorial_title | tutorial_author | submission_date |
+-------------+----------------+-----------------+-----------------+
|           3 | JAVA Tutorial  | Sanjay          | 2007-05-21      |
+-------------+----------------+-----------------+-----------------+
1 rows in set (0.01 sec)

mysql>
```

除非对字符串采用 LIKE 比对，否则默认比对是不区分大小写的。要想让搜索对大小写敏感，可以如下例一般，使用 BINARY 关键字。

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> SELECT * from tutorials_tbl \
       WHERE BINARY tutorial_author='sanjay';
Empty set (0.02 sec)

mysql>
```

使用 PHP 脚本来获取数据

要想获取表中数据，也可以使用 PHP 函数 mysql_query()，同时配合使用 SQL 的 SELECT 命令与 WHERE 子句。先用 `mysql_query()` 执行 SQL 命令，再用另一 PHP 函数 mysql_fetch_array() 来获取所有选定数

据。 mysql_fetch_array() 会将行以关联数组、数值数组，或者两种兼有 》》的形式返回。如果未选定任何数据，则该函数返回 FALSE。

范例

以下范例将返回表 tutorials_tbl 中作者名为 Sanjay 的所有记录。

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT tutorial_id, tutorial_title,
              tutorial_author, submission_date
        FROM tutorials_tbl
        WHERE tutorial_author="Sanjay"';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "Tutorial ID :{$row['tutorial_id']}  <br> ".
         "Title: {$row['tutorial_title']} <br> ".
         "Author: {$row['tutorial_author']} <br> ".
         "Submission Date : {$row['submission_date']} <br> ".
         "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

MySQL 更新查询

有时，MySQL 表中的已有数据可能需要修改，可以使用 SQL 的 UPDATE 命令来处理。它会修改MySQL表中任何字段的值。

语法

利用 UPDATE 命令修改 MySQL 表中数据的一般语法格式如下：

```
UPDATE table_name SET field1=new-value1, field2=new-value2
[WHERE Clause]
```

- 可以一起更新一个或多个字段。
- 可以使用 WHERE 子句指定任意条件。
- 可以每次仅更新一张表中的数值。

在需要更新表中选定行时，WHERE 子句是一个非常有用的工具。

采用命令行方式更新数据

下面我们将使用 SQL 的UPDATE 命令，再配合 WHERE 子句，来更新 MySQL 表 tutorials_tbl 中的选定数据。

范例

下面这个范例将为一个 tutorial_id 为 3 的记录添加一个新的 tutorials_title 字段。

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> UPDATE tutorials_tbl
   -> SET tutorial_title='Learning JAVA'
   -> WHERE tutorial_id=3;
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql>
```

使用 PHP 脚本来更新数据

同样，也可以在 PHP 的 **mysql_query()** 函数中使用 SQL 的 UPDATE 命令和（ 或不用 ）WHERE 子句。该函数执行 SQL 命令的方式与 mysql> 命令行方式相同。

范例

下面这个范例将为一个 tutorial_id 为 3 的记录添加一个新的 **tutorials_title** 字段。

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'UPDATE tutorials_tbl
        SET tutorial_title="Learning JAVA"
        WHERE tutorial_id=3';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not update data: ' . mysql_error());
}
echo "Updated data successfully\n";
mysql_close($conn);
?>
```

MySQL 删除查询

如果想从 MySQL 表中删除记录，就要用到 SQL 命令 DELETE FROM 。可以在命令行中使用该命令，也可以在 PHP 脚本中使用它。

语法格式

下面是利用 DELETE 命令删除 MySQL 表中数据的一般语法格式：

```
DELETE FROM table_name [WHERE Clause]
```

- 如果未指定 WHERE 子句，将删除指定表中的所有记录。
- 可以在 WHERE 子句中指定任意条件。
- 可以一次删除一张表中的所有记录。

在删除表中选定的行时，WHERE 子句是非常有用的。

采用命令行方式删除数据

下面将利用 DELETE 命令配合 WHERE 子句来删除表 tutorials_tbl 中的选定数据。

范例

下例将删除表 tutorials_tbl 中字段 tutorials_id 为 3 的所有记录。

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> DELETE FROM tutorials_tbl WHERE tutorial_id=3;
Query OK, 1 row affected (0.23 sec)

mysql>
```

利用 PHP 脚本来删除数据

可以使用 PHP 的 `mysql_query()` 函数，配合 SQL 的 DELETE 命令以及 WHERE 子句（也可以不用该子句）删除数据。`mysql_query()` 函数执行 SQL 命令的方式类似于上面讲到的命令行方式。

范例

下面这个范例将删除表 tutorials_tbl 中字段tutorial_id为3的记录。

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'DELETE FROM tutorials_tbl
        WHERE tutorial_id=3';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not delete data: ' . mysql_error());
}
echo "Deleted data successfully\n";
mysql_close($conn);
?>
```

MySQL Like Clause

前面几节讲解了如何利用 SQL 的 SELECT 命令来获取 MySQL 表中的数据，以及如何利用 WHERE 子句这种条件子句来选择所需的记录。

当我们需要进行精确匹配时，可以在WHERE子句中加入等号（ = ），就像是 if tutorial_author = 'Sanjay' 这种 if 条件语句一样。但有时我们会想在所有的结果中过滤 tutorial_author 字段包含 "jay" 字符的结果。这时就应该利用 SQL 的 LIKE 子句搭配 WHERE 子句来解决。

如果 SQL 的 LIKE 子句带有 % 字符，则相当于 UNIX 中的元字符（ * ），在命令行中列出所有的文件或目录。

如果 LIKE 子句不带 % 字符，则就相当于 WHERE 子句中带有等号的情况。

语法格式

使用 SQL 的 SELECT 命令，并配合 LIKE 子句，从 MySQL 表中获取数据的一般语法格式如下所示：

```
SELECT field1, field2,...fieldN table_name1, table_name2...
WHERE field1 LIKE condition1 [AND [OR]] filed2 = 'somevalue'
```

- 可以使用 WHERE 子句来指定任何条件。
- 可以搭配使用 LIKE 子句与 WHERE 子句。
- LIKE 子句可以代替WHERE子句中的等号。
- 当 LIKE 子句带有 百分号（ % ）时，会按照元字符搜索那样执行。
- 使用 AND 或 OR 运算符，可以指定一个或多个条件。
- WHERE...LIKE 子句组合还可以搭配 DELETE 或UPDATE 这样的 SQL 命令一起使用。其中，WHERE…LIKE 子句组合的作用同样是指定条件。

在命令行中使用 LIKE 子句

我们将使用 SQL 的SELECT 命令搭配WHERE...LIKE 子句组合，从 MySQL 的表 tutorials_tbl 中获取选定数据。

范例

下面这个范例将返回表 tutorials_tbl 中作者名结尾带有 jay 的所有记录。

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> SELECT * from tutorials_tbl
   -> WHERE tutorial_author LIKE '%jay';
+-------------+----------------+-----------------+-----------------+
| tutorial_id | tutorial_title | tutorial_author | submission_date |
+-------------+----------------+-----------------+-----------------+
|           3 | JAVA Tutorial  | Sanjay          | 2007-05-21      |
+-------------+----------------+-----------------+-----------------+
1 rows in set (0.01 sec)

mysql>
```

在 PHP 脚本中使用 LIKE 子句

在利用PHP 的 `mysql_query()` 函数过程中，我们可以照常使用 WHERE...LIKE 子句组合的语法。如果 WHERE...LIKE 子句组合和 SELECT 命令一起使用，那么先利用 `mysql_query()` 函数执行相关的 SQL 命令，然后再用另一个 PHP 函数 `mysql_fetch_array()` 获取所有的数据。

但如果 WHERE...LIKE 子句组合是和 DELETE 或 UPDATE 命令一起使用的话，就不需要再调用 PHP 函数了。

范例

下面这个范例将返回表 tutorials_tbl 中所有作者名包含 jay 的记录。

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
```

```
}
$sql = 'SELECT tutorial_id, tutorial_title,
              tutorial_author, submission_date
       FROM tutorials_tbl
       WHERE tutorial_author LIKE "%jay%"';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "Tutorial ID :{$row['tutorial_id']}  <br> ".
         "Title: {$row['tutorial_title']} <br> ".
         "Author: {$row['tutorial_author']} <br> ".
         "Submission Date : {$row['submission_date']} <br> ".
         "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

MySQL 排序结果

利用 SQL 的 SELECT 命令可以获取 MySQL 表中的数据。选择行时，如果不指定结果排序方式，MySQL 服务器所返回结果是没有一定的顺序的。指定想要排序的列，通过添加 ORDER BY 子句，就可以对结果集进行排序。

语法格式

利用 SQL 的 SELECT 命令，配合 ORDER BY 子句，对 MySQL 表中的数据进行排序：

```
SELECT field1, field2,...fieldN table_name1, table_name2...
ORDER BY field1, [field2...] [ASC [DESC]]
```

- 可以对列出的任何字段的返回结果进行排序。
- 可以对多个字段的返回结果进行排序。
- 可以使用关键字 ASC 或 DESC ，以升降序对结果进行排序。默认是采用升序排序。
- 通常可使用 WHERE...LIKE 子句设置条件。

在命令行中使用 ORDER BY 子句

我们将使用 SQL 的 SELECT 命令与 ORDER BY 子句，从 MySQL 表 tutorials_tbl 中获取数据。

范例

下面这个范例将采用升序的方式对返回结果进行排序。

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> SELECT * from tutorials_tbl ORDER BY tutorial_author ASC
+-------------+----------------+----------------+-----------------+
| tutorial_id | tutorial_title | tutorial_author | submission_date |
+-------------+----------------+----------------+-----------------+
|           2 | Learn MySQL    | Abdul S        | 2007-05-24      |
|           1 | Learn PHP      | John Poul      | 2007-05-24      |
```

```
|         3 | JAVA Tutorial  | Sanjay         | 2007-05-06     |
+-------------+----------------+----------------+-----------------+
3 rows in set (0.42 sec)

mysql>
```

如上所示，作者名称按照升序排列出来。

在 PHP 脚本中使用 ORDER BY 子句

除了在命令行中使用外，我们也可以在 PHP 函数 `mysql_query()` 中使用 ORDER BY 子句，两种情况下的语法都是相同的。先用 `mysql_query()` 执行 SQL 命令，然后再用 PHP 函数 `mysql_fetch_array()` 获取所有选定的数据。

范例

下面这个范例将按升序排列教程作者名称（ `tutorial_author` ）。

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT tutorial_id, tutorial_title,
              tutorial_author, submission_date
        FROM tutorials_tbl
        ORDER BY  tutorial_author DESC';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "Tutorial ID :{$row['tutorial_id']}  <br> ".
         "Title: {$row['tutorial_title']} <br> ".
```

```
        "Author: {$row['tutorial_author']} <br> ".
        "Submission Date : {$row['submission_date']} <br> ".
        "–––––––––––––––––––––––––––––––<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

MySQL Using Join

迄今为止，我们每次只能从一张表里获取数据。这足以应付简单的任务了，但大多数真实的 MySQL 应用场景却经常会需要通过一次查询，从多张表中获取数据。

在一个 SQL 查询中使用多张表，联结（join）行为在 MySQL 数据库中指的就是将2张或更多的表合为一张表。

你可以在 SELECT、UPDATE、DELETE语句中使用 JOIN 来联结 MySQL 表。下面还将介绍一个左联结（LEFT JOIN）的范例，了解一下它与 JOIN 的区别。

在命令行中使用 JOIN

假设数据库 TUTORIALS 中有两张表：tcount_tbl 和 tutorials_tbl。完整的代码清单如下所示。

范例

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> SELECT * FROM tcount_tbl;
+-----------------+----------------+
| tutorial_author | tutorial_count |
+-----------------+----------------+
| mahran          |             20 |
| mahnaz          |           NULL |
| Jen             |           NULL |
| Gill            |             20 |
| John Poul       |              1 |
| Sanjay          |              1 |
+-----------------+----------------+
6 rows in set (0.01 sec)
mysql> SELECT * from tutorials_tbl;
+-------------+----------------+-----------------+-----------------+
| tutorial_id | tutorial_title | tutorial_author | submission_date |
+-------------+----------------+-----------------+-----------------+
|           1 | Learn PHP      | John Poul       | 2007-05-24      |
|           2 | Learn MySQL    | Abdul S         | 2007-05-24      |
|           3 | JAVA Tutorial  | Sanjay          | 2007-05-06      |
+-------------+----------------+-----------------+-----------------+
```

```
3 rows in set (0.00 sec)
mysql>
```

上例通过一个 SQL 查询将两张表联结到一起。这次查询选择了表 tutorials_tbl 中所有的作者，然后获取表 tcount_tbl 中这些作者相应的教程数量。

```
mysql> SELECT a.tutorial_id, a.tutorial_author, b.tutorial_count
    -> FROM tutorials_tbl a, tcount_tbl b
    -> WHERE a.tutorial_author = b.tutorial_author;
+-------------+-----------------+----------------+
| tutorial_id | tutorial_author | tutorial_count |
+-------------+-----------------+----------------+
|           1 | John Poul       |              1 |
|           3 | Sanjay          |              1 |
+-------------+-----------------+----------------+
2 rows in set (0.01 sec)
mysql>
```

在 PHP 脚本中使用JOIN

可以在 PHP 脚本中使用以前学到过的任何一种 SQL 查询。只需要将 SQL 查询传入 PHP 函数 `mysql_query()` 中，就能按照之前的方式获得结果。

范例

相关范例如下：

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT a.tutorial_id, a.tutorial_author, b.tutorial_count
        FROM tutorials_tbl a, tcount_tbl b
        WHERE a.tutorial_author = b.tutorial_author';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
```

```
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "Author:{$row['tutorial_author']}  <br> ".
         "Count: {$row['tutorial_count']} <br> ".
         "Tutorial ID: {$row['tutorial_id']} <br> ".
         "–––––––––––––––––––––––––––––––<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

MySQL的左联结（LEFT JOIN）

MySQL 的左联结（LEFT JOIN）与简单使用 JOIN 的效果不同。左联结侧重考虑左侧的表。

如果进行左联结，除了得到所有跟以上联结同样的匹配记录之外，还会得到左侧表中未曾匹配的记录，从而保证了（在该范例中）照顾到了每一位作者。

范例

下面这个范例可以帮我们更好地理解左联结。

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> SELECT a.tutorial_id, a.tutorial_author, b.tutorial_count
    -> FROM tutorials_tbl a LEFT JOIN tcount_tbl b
    -> ON a.tutorial_author = b.tutorial_author;
+-------------+-----------------+----------------+
| tutorial_id | tutorial_author | tutorial_count |
+-------------+-----------------+----------------+
|           1 | John Poul       |              1 |
|           2 | Abdul S         |           NULL |
|           3 | Sanjay          |              1 |
+-------------+-----------------+----------------+
3 rows in set (0.02 sec)
```

必须多加练习，才能熟悉 JOIN。这是 MySQL/SQL 中的一个比较复杂的概念，必须经过一番真实的案例磨炼才能真正地掌握它。

MySQL NULL Values

前面已经介绍了如何利用 SQL 的 SELECT 命令配合 WHERE 子句来获取 MySQL 表中的数据，但假如尝试给出一个条件，将字段或列值与 NULL 比对，则会出现异常。

为了处理这种情况，MySQL 提供了三种运算符：

- IS NULL：如果列值为 NULL，则该运算符返回 true。
- IS NOT NULL：如果列值不为NULL，则该运算符返回 true。
- <=>：该运算符用于两个值的对比，当两个值相等时（即使这两个值都为 NULL 时，这一点与 `=` 运算符不同）返回 true。

包含 NULL 的条件都是比较特殊的。不能在列中使用 = NULL 或 ! = NULL 来寻找 NULL 值。这样的比对通常都是失败的，因为不可能得知这样的比对是否为真。即使 NULL = NULL 失败。

要想确定列是否为 NULL，要使用 IS NULL 或 IS NOT NULL。

在命令行中使用 NULL 值

假设数据库 TUTORIALS 中包含一张叫做 tcount_tbl 的表，这张表包含两列 tutorial_author 与 tutorial_count，则 tutorial_count 中出现的 NULL 值代表该字段值未知。

范例

请看下面这个范例：

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> create table tcount_tbl
    -> (
    -> tutorial_author varchar(40) NOT NULL,
    -> tutorial_count  INT
    -> );
Query OK, 0 rows affected (0.05 sec)
mysql> INSERT INTO tcount_tbl
    -> (tutorial_author, tutorial_count) values ('mahran', 20);
```

```
mysql> INSERT INTO tcount_tbl
    -> (tutorial_author, tutorial_count) values ('mahnaz', NULL);
mysql> INSERT INTO tcount_tbl
    -> (tutorial_author, tutorial_count) values ('Jen', NULL);
mysql> INSERT INTO tcount_tbl
    -> (tutorial_author, tutorial_count) values ('Gill', 20);

mysql> SELECT * from tcount_tbl;
+-----------------+----------------+
| tutorial_author | tutorial_count |
+-----------------+----------------+
| mahran          |             20 |
| mahnaz          |           NULL |
| Jen             |           NULL |
| Gill            |             20 |
+-----------------+----------------+
4 rows in set (0.00 sec)

mysql>
```

下面，你会发现 = 和 != 并不适用于 NULL 值。

```
mysql> SELECT * FROM tcount_tbl WHERE tutorial_count = NULL;
Empty set (0.00 sec)
mysql> SELECT * FROM tcount_tbl WHERE tutorial_count != NULL;
Empty set (0.01 sec)
```

所以，要想确定 tutorial_count 列中到底何值为 NULL，何值不为 NULL，查询应该这样写：

```
mysql> SELECT * FROM tcount_tbl
    -> WHERE tutorial_count IS NULL;
+-----------------+----------------+
| tutorial_author | tutorial_count |
+-----------------+----------------+
| mahnaz          |           NULL |
| Jen             |           NULL |
+-----------------+----------------+
2 rows in set (0.00 sec)
mysql> SELECT * from tcount_tbl
    -> WHERE tutorial_count IS NOT NULL;
+-----------------+----------------+
| tutorial_author | tutorial_count |
+-----------------+----------------+
| mahran          |             20 |
| Gill            |             20 |
```

```
+----------------+----------------+
2 rows in set (0.00 sec)
```

在 PHP 脚本中处理 NULL 值

可以使用 `if...else` 条件语句来准备一个基于 NULL 值的查询。

范例

下面这个例子获取外部的 tutorial_count，并将其与表中的值进行比对。

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
if( isset($tutorial_count ))
{
   $sql = 'SELECT tutorial_author, tutorial_count
          FROM  tcount_tbl
          WHERE tutorial_count = $tutorial_count';
}
else
{
   $sql = 'SELECT tutorial_author, tutorial_count
          FROM  tcount_tbl
          WHERE tutorial_count IS $tutorial_count';
}

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "Author:{$row['tutorial_author']}  <br> ".
         "Count: {$row['tutorial_count']} <br> ".
```

```
        "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

MySQL 正则表达式

前面介绍过 MySQL 。MySQL 还支持另一种基于正则表达式的模式匹配操作，使用的运算符是 REGEXP。如果你学过 PHP 或 PERL，那么这就很好理解了，因为这里讲的这种匹配方式跟那些脚本语言中的正则表达式很相似。

下面就是一个模式列表，其中结合使用了 REGEXP 运算符。

模式	模式匹配对象
^	字符串的开始位置
$	字符串的结尾
.	单个字符
[...]	一对方括号之间的字符
[^...]	未在一对方括号之间的字符
p1|p2|p3	交替匹配模式1、模式2或模式3
*	匹配前面元素的零个或多个实例
+	匹配前面元素的一个或多个实例
{n}	匹配前面元素的n个实例
{m,n}	匹配前面元素的m~n个实例，m <= n

范例

根据以上这张列表，可以设计出能够满足各种要求的 SQL 查询。下面就来列举一二。假设有一张表 person_tbl，其中包含一个name字段。

寻找以 'st' 开头的名称，查询如下：

```
mysql> SELECT name FROM person_tbl WHERE name REGEXP '^st';
```

寻找以 'ok' 结尾的名称，查询如下：

```
mysql> SELECT name FROM person_tbl WHERE name REGEXP 'ok$';
```

寻找包含 'mar' 的名称，查询如下：

```
mysql> SELECT name FROM person_tbl WHERE name REGEXP 'mar';
```

寻找以元音字母开始并以 'ok' 结尾的名称，查询如下：

```
mysql> SELECT name FROM person_tbl WHERE name REGEXP '^[aeiou]|ok$';
```

MySQL 汇报

事务就是一组连续的数据库操作，执行起来仿佛像是单一的工作单元。换句话说，除非该组内所有操作都成功完成，否则事务永远不会结束。如果事务中的某一个操作失败，则整个事务也将失败。

实际上，将在一个组中结合许多 SQL 查询，你将同时执行所有的事务，作为事务的一部分。

事务的特点

事务一般具有以下4种典型特点，人们通常会用这4种特点的英文首字母缩写组合词 ACID 来表示。

- **原子性**（Atomicity）确保工作单元内的所有操作都能成功完成。如若不然，事务在遭受失败时就会被放弃，之前的种种操作就会被撤销回它们之前的状态。
- **一致性**（Consistency）确保数据库能够在成功提交的事务的基础上正确改变状态。
- **隔离性**（Isolation）使事务能够独立操作，事务之间彼此透明。
- **持久性**（Durability）确保当系统发生失败时，已提交的事务的结果或者说效果能够持续存在。

在 MySQL 中，事务通常以 BEGIN WORK 语句开始，以 COMMIT 或 ROLLBACK（只取其一）语句结束。在开始与结束声明之间的 SQL 命令就构成了事务的主体。

COMMIT 与 ROLLBACK

MySQL事务主要用到两个关键字 **COMMIT** 与 **ROLLBACK**:

- 成功完成一个事务后，就会执行 COMMIT 命令，从而使施加于所涉及的表上的改变生效。
- 如果事务失败，就会执行 ROLLBACK 命令，将事务中所引用的每一个表都回撤到之前的状态。

通过设定会话变量 **AUTOCOMMIT** 可以控制事务行为。如果 AUTOCOMMIT 被设为1（默认值），则每一个 SQL 语句（无论是否在事务中）都会被认为是一个完成的事务，则默认当它结束时予以提交。当 AUTOCOMMIT 被设为0（通过命令 SET AUTOCOMMIT=0）时，后续一系列语句就像是一个事务，直到 COMMIT 语句执行为止，不再提交任何行为。

可以在 PHP 中利用 mysql_query()执行 SQL 命令。

事务的常见范例

这些事件都跟所用的编程语言无关。逻辑路径可以用你所使用的任何语言来创建。

可以在 PHP 中利用 mysql_query()执行 SQL 命令。

- 通过执行 SQL 命令 BEGIN WORK 可开启事务。
- 执行一个或更多的如下 SQL 命令：SELECT、INSERT、UPDATE 或 DELETE。
- 检查是否有错，一切是否符合要求。
- 如果出错，执行 ROLLBACK 命令，否则利用 COMMIT 命令提交。

MySQL 中的事务安全型表类型

不能直接使用事务，如果强行使用，则无法保证它们的安全性。如果打算在 MySQL 编程中使用事务，就需要以特殊的方式来创建表。有很多种支持事务表可供选择，但其中最常见的是 InnoDB。

对 InnoDB 表的支持，需要在编译 MySQL 源码时用到一个特殊的编译参数。如果 MySQL 版本不支持 InnoDB，则需要请你的 ISP 构建一个支持 InnoDB 表类型的 MySQL 版本，或者下载安装一个用于 Windows 或 Linux/UNIX 系统的 MySQL-Max 二进制分发版，在其开发环境中使用这种表类型。

如果你的 MySQL 版本支持 InnoDB 表，则只需在表创建语句中添加一个 TYPE = InnoDB 定义即可。比如，下面这段代码就创建了一个叫做 tcount_tbl 的 InnoDB 表。

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> create table tcount_tbl
    -> (
    -> tutorial_author varchar(40) NOT NULL,
    -> tutorial_count  INT
    -> ) TYPE=InnoDB;
Query OK, 0 rows affected (0.05 sec)
```

有关InnoDB的详细信息，可参看这个链接：

如果你的 MySQL 支持 GEMINI 或 BDB 这两种表类型，也可以使用它们。

MySQL ALTER 命令

利用 MySQL 的 ALTER 命令可以很方便地修改表名与表字段名，以及添加或删除表中已有的列。

为了实践 ALTER 命令，下面先来创建一个名为 testalter_tbl 的表。

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> create table testalter_tbl
    -> (
    -> i INT,
    -> c CHAR(1)
    -> );
Query OK, 0 rows affected (0.05 sec)
mysql> SHOW COLUMNS FROM testalter_tbl;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| i     | int(11) | YES  |     | NULL    |       |
| c     | char(1) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

删除、添加列或对其重新定位

假如要从上面我们创建的这张表中删除 i 这一列，那么应该使用 DROP 子句和 ALTER 命令，如下所示：

```
mysql> ALTER TABLE testalter_tbl DROP i;
```

如果表中只有一列，则 DROP 子句不起作用。

添加一列，使用 ADD 并指定列定义。下面我们再把 i 这一列恢复到 testalter_tbl 中。

```
mysql> ALTER TABLE testalter_tbl ADD i INT;
```

输入该语句之后，这张表将跟之前创建时一样，含有2列。但结构却稍有差异。默认情况下，新增加的列位于表的末尾。在创建表时，i 是第一列，现在却成为最后一列。

```
mysql> SHOW COLUMNS FROM testalter_tbl;
+-------+---------+------+-----+---------+-------+
```

```
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| c     | char(1) | YES  |     | NULL    |       |
| i     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

要想把列放到一个特定位置，可以使用两种方法。第一种方法是使用 FIRST，让指定列成为第一列；第二种则采用 AFTER 后跟给定列名的方式，指示新列应该放到给定列名的后面。下面分别利用 ALTER TABLE 语句对列进行操作，每执行完一行后，我们可以使用 SHOW COLUMNS 来查看一下各自的效果。

```
ALTER TABLE testalter_tbl DROP i;
ALTER TABLE testalter_tbl ADD i INT FIRST;
ALTER TABLE testalter_tbl DROP i;
ALTER TABLE testalter_tbl ADD i INT AFTER c;
```

标识符 FIRST 和 AFTER 只能和 ADD 子句一起使用。这也意味着，如果要重新定位一列，就必须先用 DROP 删除它，然后再用 ADD 将它添加到新的位置。

改变一列的定义或名称

要想改变列的定义，需要使用 MODIFY 或 CHANGE 子句，并配合使用 ALTER 命令。例如，把列 c 从 CHAR(1) 变为 CHAR(10)：

```
mysql> ALTER TABLE testalter_tbl MODIFY c CHAR(10);
```

CHANGE 的语法稍有不同。必须把所要改变的列名放到 CHANGE 关键字的后面，然后指定新的列定义。相关范例如下所示：

```
mysql> ALTER TABLE testalter_tbl CHANGE i j BIGINT;
```

同理，如果想利用 CHANGE 将 j 从 BIGINT 转为 INT，并且不改变列名，则语句如下：

```
mysql> ALTER TABLE testalter_tbl CHANGE j j INT;
```

ALTER TABLE 对 Null 及默认值属性的作用

在利用 MODIFY 或 CHANGE 修改列时，还可以指定该列是否能有 NULL 值，以及它的默认值。事实上，如果我们不这样处理，MySQL 会自动为这些属性指定相关值。

例如，NOT NULL 列默认值为100：

```
mysql> ALTER TABLE testalter_tbl
    -> MODIFY j BIGINT NOT NULL DEFAULT 100;
```

如果不使用上述命令，则MySQL 会在所有这些列中填充 NULL 值。

改变列的默认值

使用 ALTER 命令可以改变任何列的默认值，如下例所示：

```
mysql> ALTER TABLE testalter_tbl ALTER i SET DEFAULT 1000;
mysql> SHOW COLUMNS FROM testalter_tbl;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| c     | char(1) | YES  |     | NULL    |       |
| i     | int(11) | YES  |     | 1000    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

使用 DROP 子句与 ALTER 命令，可以去除任何列中的默认限制。

```
mysql> ALTER TABLE testalter_tbl ALTER i DROP DEFAULT;
mysql> SHOW COLUMNS FROM testalter_tbl;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| c     | char(1) | YES  |     | NULL    |       |
| i     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

改变表类型

结合使用 TYPE 子句与 ALTER 命令，可以使用表类型。在下面范例中，将表 testalter_tbl 的表类型变为 MYISAM。

通过 SHOW TABLE STATUS 语句可以显示当前的表类型。

```
mysql> ALTER TABLE testalter_tbl TYPE = MYISAM;
mysql>  SHOW TABLE STATUS LIKE 'testalter_tbl'\G
*************************** 1. row ****************
           Name: testalter_tbl
           Type: MyISAM
```

```
    Row_format: Fixed
          Rows: 0
 Avg_row_length: 0
   Data_length: 0
Max_data_length: 25769803775
  Index_length: 1024
     Data_free: 0
 Auto_increment: NULL
   Create_time: 2007-06-03 08:04:36
   Update_time: 2007-06-03 08:04:36
    Check_time: NULL
 Create_options:
       Comment:
1 row in set (0.00 sec)
```

对表进行重命名

使用 ALTER TABLE 语句的 RENAME 选项可以对表进行重命名。下面范例将表 testalter_tbl 重新命名为 alter_tbl。

```
mysql> ALTER TABLE testalter_tbl RENAME TO alter_tbl;
```

还可以使用 ALTER 命令来创建并删除 MySQL 文件中的索引。下一节再介绍这种用法。

MySQL 索引

数据库索引是一种能够改善表操作速度的数据结构。索引可以通过一个或多个列来创建，它可以提高随机查询的速度，并在检索记录时实现高效排序。

在创建索引时，需要考虑哪些列会用于 SQL 查询，然后为这些列创建一个或多个索引。

事实上，索引也是一种表，保存着主键或索引字段，以及一个能将每个记录指向实际表的指针。

数据库用户是看不到索引的，它们只是用来加速查询的。数据库搜索引擎使用索引来快速定位记录。

INSERT 与 UPDATE 语句在拥有索引的表中执行会花费更多的时间，而 SELECT 语句却会执行得更快。这是因为，在进行插入或更新时，数据库也需要插入或更新索引值。

简单而唯一的索引

可以为表创建唯一索引。唯一索引要求任意两行的索引值不能相同。下面展示的就是在表中创建索引的语法格式：

```
CREATE UNIQUE INDEX index_name
ON table_name ( column1, column2,...);
```

也可以使用一或多个列来创建索引。比如说，表 tutorials_tbl 中，使用 tutorial_author 来创建索引。

```
CREATE UNIQUE INDEX AUTHOR_INDEX
ON tutorials_tbl (tutorial_author)
```

还可以为表创建简单索引，只需要在查询时不带 UNIQUE 关键字，就可创建简单索引。简单索引允许在表中复制值。

如果打算按照降序在列中索引数值，可以在列名后添加保留字 DESC。

```
mysql> CREATE UNIQUE INDEX AUTHOR_INDEX
ON tutorials_tbl (tutorial_author DESC)
```

添加与删除 INDEX 的ALTER 命令

为表添加索引，可以采用4种语句。

- `ALTER TABLE tbl_name ADD PRIMARY KEY (column_list)` 该语句添加一个主键。意味着索引值必须是唯一的，不能为 NULL。
- `ALTER TABLE tbl_name ADD UNIQUE index_name (column_list)` 该语句为必须唯一的值（除了 NULL 值之外，NULL 值可以多次出现）创建索引。
- `ALTER TABLE tbl_name ADD INDEX index_name (column_list)` 语句为可能多次出现的值创建一般索引。
- `ALTER TABLE tbl_name ADD FULLTEXT index_name (column_list)` 语句创建专用于文本搜索的 FULLTEXT 索引。

下面这个范例将为现有表添加索引。

```
mysql> ALTER TABLE testalter_tbl ADD INDEX (c);
```

可以使用 DROP 子句以及 ALTER 命令删除索引，通过下面这个范例来删除之前创建的索引。

```
mysql> ALTER TABLE testalter_tbl DROP INDEX (c);
```

利用 ALTER 命令来添加与删除主键

添加主键也采用类似方式，但要保证主键一定在列上，是 NOT NULL。

下面这个范例将在现有表中添加主键，先使列为 NOT NULL，然后再将其作为主键。

```
mysql> ALTER TABLE testalter_tbl MODIFY i INT NOT NULL;
mysql> ALTER TABLE testalter_tbl ADD PRIMARY KEY (i);
```

同样，也可以使用 ALTER 命令删除一个主键。

```
mysql> ALTER TABLE testalter_tbl DROP PRIMARY KEY;
```

如果要删除非主键的索引，则必须指定索引名称。

显示索引信息

使用 SHOW INDEX 命令可以列出表的所有索引。以垂直格式输出（标识为 `\G`）会比较便于查看，可避免单行内容过长。语法格式如下：

```
mysql> SHOW INDEX FROM table_name\G
.....
```

MySQL 临时表

当需要保存临时数据时，使用临时表就会很方便。需要注意的一点是：在当前用户会话终止时，临时表会被清除。

MySQL 是从 3.23 版开始引入临时表这一概念。如果使用 3.23 之前的 MySQL版本，则无法使用临时表，但可以使用堆表（HEAP table）。

如上所述，临时表只能在会话生存期内存在。如果在 PHP 脚本中运行代码，那么当脚本结束执行时，就会自动清除临时表。如果通过 MySQL 客户端程序连接 MySQL 数据库服务器，那么临时表就会一直存在，除非关闭客户端或者手动清除该表。

范例

下面范例将展示临时表的用法。同样的代码也可通过 mysql_query() 用于 PHP 脚本中。

```
mysql> CREATE TEMPORARY TABLE SalesSummary (
    -> product_name VARCHAR(50) NOT NULL
    -> , total_sales DECIMAL(12,2) NOT NULL DEFAULT 0.00
    -> , avg_unit_price DECIMAL(7,2) NOT NULL DEFAULT 0.00
    -> , total_units_sold INT UNSIGNED NOT NULL DEFAULT 0
);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO SalesSummary
    -> (product_name, total_sales, avg_unit_price, total_units_sold)
    -> VALUES
    -> ('cucumber', 100.25, 90, 2);

mysql> SELECT * FROM SalesSummary;
+--------------+-------------+----------------+------------------+
| product_name | total_sales | avg_unit_price | total_units_sold |
+--------------+-------------+----------------+------------------+
| cucumber     |      100.25 |          90.00 |                2 |
+--------------+-------------+----------------+------------------+
1 row in set (0.00 sec)
```

利用 SHOW TABLES 命令显示表时，临时表不会出现在结果列表中。如果退出 MySQL 会话，就会执行 SELECT 命令，那么数据库中将没有任何数据，甚至临时表也不存在了。

删除临时表

默认情况下，当与数据库的连接终止时，临时表就不再存在。不过如果想在数据库处于连接时就删除它们，可以用 DROP TABLE 命令来删除。

下面就是一个删除临时表的范例：

```
mysql> CREATE TEMPORARY TABLE SalesSummary (
    -> product_name VARCHAR(50) NOT NULL
    -> , total_sales DECIMAL(12,2) NOT NULL DEFAULT 0.00
    -> , avg_unit_price DECIMAL(7,2) NOT NULL DEFAULT 0.00
    -> , total_units_sold INT UNSIGNED NOT NULL DEFAULT 0
);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO SalesSummary
    -> (product_name, total_sales, avg_unit_price, total_units_sold)
    -> VALUES
    -> ('cucumber', 100.25, 90, 2);

mysql> SELECT * FROM SalesSummary;
+--------------+-------------+----------------+------------------+
| product_name | total_sales | avg_unit_price | total_units_sold |
+--------------+-------------+----------------+------------------+
| cucumber     |      100.25 |          90.00 |                2 |
+--------------+-------------+----------------+------------------+
1 row in set (0.00 sec)
mysql> DROP TABLE SalesSummary;
mysql>  SELECT * FROM SalesSummary;
ERROR 1146: Table 'TUTORIALS.SalesSummary' doesn't exist
```

MySQL 复制表

在某些情况下，你可能需要精确复制一个表。CREATE TABLE ... SELECT 并不适用于此要求，因为这个复制表必须与原有表在索引、默认值等方面都完全相同。

可以采用如下步骤来处理这种情况。

- 使用 SHOW CREATE TABLE 或 CREATE TABLE 语句指定源表的结构、索引以及所有的内容。
- 调整语句，将表名改为克隆表的名称，执行语句。这样就对表进行了克隆。
- 另外，如果想要克隆表的全部内容，也可以使用 INSERT INTO ... SELECT 语句。

范例

通过下面这个范例来创建表 tutorial_tbl 的克隆表：

步骤1

获取表的完整结构。

```
mysql> SHOW CREATE TABLE tutorials_tbl \G;
*************************** 1. row ***************************
       Table: tutorials_tbl
Create Table: CREATE TABLE `tutorials_tbl` (
  `tutorial_id` int(11) NOT NULL auto_increment,
  `tutorial_title` varchar(100) NOT NULL default '',
  `tutorial_author` varchar(40) NOT NULL default '',
  `submission_date` date default NULL,
  PRIMARY KEY  (`tutorial_id`),
  UNIQUE KEY `AUTHOR_INDEX` (`tutorial_author`)
) TYPE=MyISAM
1 row in set (0.00 sec)

ERROR:
No query specified
```

步骤2

重新命名该表，创建另一个表。

```
mysql> CREATE TABLE `clone_tbl` (
  -> `tutorial_id` int(11) NOT NULL auto_increment,
  -> `tutorial_title` varchar(100) NOT NULL default '',
  -> `tutorial_author` varchar(40) NOT NULL default '',
  -> `submission_date` date default NULL,
  -> PRIMARY KEY (`tutorial_id`),
  -> UNIQUE KEY `AUTHOR_INDEX` (`tutorial_author`)
-> ) TYPE=MyISAM;
Query OK, 0 rows affected (1.80 sec)
```

步骤3

执行完步骤2后，就在数据库中创建了一个克隆表。如果想从旧表中复制数据，可以使用 INSERT INTO... SELECT 语句。

```
mysql> INSERT INTO clone_tbl (tutorial_id,
    ->                        tutorial_title,
    ->                        tutorial_author,
    ->                        submission_date)
    -> SELECT tutorial_id,tutorial_title,
    ->        tutorial_author,submission_date,
    -> FROM tutorials_tbl;
Query OK, 3 rows affected (0.07 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

最终，果如所愿，你得到了精确的克隆表。

MySQL 数据库信息

MySQL可为你提供3类有价值的信息：

- **查询结果信息** 受SELECT、UPDATE 或 DELETE 语句影响的记录数量。
- **表与数据库的信息** 表及数据库的相关信息。
- **MySQL 服务器的信息** 数据库服务器的状态以及数据库版本号等信息。

通过命令行方式可以轻松获取以上这些信息，但如果使用 PERL 或 PHP API，就需要显式地调用各种 API 来获取这些信息。下面将介绍具体做法。

获取受查询影响的行数

PERL 范例

在 DBI 脚本中，根据执行查询的方式，通过 do() 或 execute() 返回受查询影响的行数。

```
 # Method 1
 # execute $query using do( )
my $count = $dbh->do ($query);
 # report 0 rows if an error occurred
printf "%d rows were affected\n", (defined ($count) ? $count : 0);

 # Method 2
 # execute query using prepare( ) plus execute( )
my $sth = $dbh->prepare ($query);
my $count = $sth->execute ( );
printf "%d rows were affected\n", (defined ($count) ? $count : 0);
```

PHP 范例

在PHP中，调用 `mysql_affected_rows()` 函数来查找查询所影响的行数。

```
$result_id = mysql_query ($query, $conn_id);
 # report 0 rows if the query failed
$count = ($result_id ? mysql_affected_rows ($conn_id) : 0);
print ("$count rows were affected\n");
```

输出表与数据库的相关信息

很容易就能输出数据库服务器上的数据库及表的相关信息，但如果没有足够权限，则可能所得结果为空。

除了下面将要介绍的方法之外，还可以使用 SHOW TABLES 或 SHOW DATABASES 查询来获取表与数据库的相关信息，这一点对于 PHP 和 PERL 都是适用的。

PERL 范例

```
 # 获取当前数据库中的所有表
my @tables = $dbh->tables ( );
foreach $table (@tables ){
   print "Table Name $table\n";
}
```

PHP 范例

```
<?php
$con = mysql_connect("localhost", "userid", "password");
if (!$con)
{
  die('Could not connect: ' . mysql_error());
}

$db_list = mysql_list_dbs($con);

while ($db = mysql_fetch_object($db_list))
{
  echo $db->Database . "<br />";
}
mysql_close($con);
?>
```

获取服务器元数据

利用下面5种命令可以获取数据库服务器上的各种关键信息。它们既适用于命令行，也适用于 PHP 或 PERL 脚本。

命令	描述
SELECT VERSION()	表明服务器版本的字符串
SELECT DATABASE()	当前数据库名称（如果没有则为空值）
SELECT USER()	当前用户名
SHOW STATUS	服务器状态指示器
SHOW VARIABLES	服务器配置变量

MySQL Using Sequences

序列就是按照要求的顺序产生的一组整数，比如1、2、3……这样。数据库中经常会用到序列，因为很多应用程序都会需要让表中的每行的值唯一，而使用序列就可以轻松地解决这个问题。下面就来介绍 MySQL 中的序列使用。

使用 AUTO_INCREMENT 列

在MySQL中，序列最简单的用法就是将一列定义为 AUTO_INCREMENT ，然后让 MySQL 来处理剩下的任务。

范例

在下面这个范例中，先创建一个表，然后插入一些行。不需要提供记录ID，因为这是由 MySQL 自动增加的。

```
mysql> CREATE TABLE insect
    -> (
    -> id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    -> PRIMARY KEY (id),
    -> name VARCHAR(30) NOT NULL, # type of insect
    -> date DATE NOT NULL, # date collected
    -> origin VARCHAR(30) NOT NULL # where collected
);
Query OK, 0 rows affected (0.02 sec)
mysql> INSERT INTO insect (id,name,date,origin) VALUES
    -> (NULL,'housefly','2001-09-10','kitchen'),
    -> (NULL,'millipede','2001-09-10','driveway'),
    -> (NULL,'grasshopper','2001-09-10','front yard');
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> SELECT * FROM insect ORDER BY id;
+----+-------------+------------+------------+
| id | name        | date       | origin     |
+----+-------------+------------+------------+
|  1 | housefly    | 2001-09-10 | kitchen    |
|  2 | millipede   | 2001-09-10 | driveway   |
|  3 | grasshopper | 2001-09-10 | front yard |
+----+-------------+------------+------------+
3 rows in set (0.00 sec)
```

获取 AUTO_INCREMENT 的值

LAST_INSERT_ID() 是一个 SQL 函数，所以可以把它用在任何能够理解 SQL 语句的客户端中。另外，PERL 和 PHP 还提供了一些专有的函数来获取最后记录的自动增加值。

PERL 范例

使用 mysql_insertid 属性来获取查询所生成的 AUTO_INCREMENT 值。根据查询方式，该属性可通过数据库句柄或语句句柄来访问。下面这个范例是从数据库句柄来引用该属性的。

```
$dbh->do ("INSERT INTO insect (name,date,origin)
VALUES('moth','2001-09-14','windowsill')");
my $seq = $dbh->{mysql_insertid};
```

PHP 范例

当查询生成了 AUTO_INCREMENT 值后，通过调用 mysql_insert_id() 函数来获取该值。

```
mysql_query ("INSERT INTO insect (name,date,origin)
VALUES('moth','2001-09-14','windowsill')", $conn_id);
$seq = mysql_insert_id ($conn_id);
```

对已有序列进行重新编号

假如从表中删除了许多记录，必须对记录再次排序。这时可以使用一个小技巧来解决，但要记住当表还连接着其他表时，一定要非常小心。

如果一定要对 AUTO_INCREMENT 列进行重新排序，那么正确的方式是将该列从表中删除，然后再添加它。下面这个范例中就用了这个技巧，在 insect 表中对 id 值重新排序。

```
mysql> ALTER TABLE insect DROP id;
mysql> ALTER TABLE insect
    -> ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT FIRST,
    -> ADD PRIMARY KEY (id);
```

以特定值作为序列初始值

MySQL 默认以 1 作为序列初始值，但你也可以在创建表时指定其他的数字。下面的范例以 100 作为序列初始值。

```
mysql> CREATE TABLE insect
    -> (
    -> id INT UNSIGNED NOT NULL AUTO_INCREMENT = 100,
    -> PRIMARY KEY (id),
    -> name VARCHAR(30) NOT NULL, # type of insect
    -> date DATE NOT NULL, # date collected
    -> origin VARCHAR(30) NOT NULL # where collected
);
```

另外一种方法是，先创建表，然后再使用 ALTER TABLE 来改变初始序列值。

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

MySQL Handling Duplicates

表或结果集有时会包含重复记录。这种情况一般来说是允许出现的，但有时却需要终止这些重复记录。在某些情况下，需要找出重复记录并将其删除。下面就来介绍一下如何防止表中出现重复记录，如何删除已有的重复记录。

防止表中出现重复记录

可以在表中正确的字段内使用 PRIMARY KEY 或 UNIQUE 索引来终止重复记录。比如下面这张表，由于没有这样的索引或主键，因此 first_name与last_name 就被重复记录了下来。

```
CREATE TABLE person_tbl
(
   first_name CHAR(20),
   last_name CHAR(20),
   sex CHAR(10)
);
```

为了防止表中出现同样姓名的值，为其添加一个 PRIMARY KEY。同时要注意将索引列声明为 NOT NULL，这是因为 PRIMARY KEY 不允许出现空值。

```
CREATE TABLE person_tbl
(
  first_name CHAR(20) NOT NULL,
  last_name CHAR(20) NOT NULL,
  sex CHAR(10),
  PRIMARY KEY (last_name, first_name)
);
```

表中的唯一索引通常会造成错误，如果往表中插入一个记录，复制了定义该索引的一个列（或多个列）中的一个已存记录，问题就会产生。

不要使用 INSERT ，使用 INSERT IGNORE。如果一个记录没有复制一个已存在的记录，MySQL 就会将它照常插入。如果该记录与现存的某个记录重复，IGNORE 关键字就会让 MySQL 默默地将其摒弃，不会产生任何错误。

下面这个范例不会产生任何错误，不会插入会产生重复的记录。

```
mysql> INSERT IGNORE INTO person_tbl (last_name, first_name)
   -> VALUES( 'Jay', 'Thomas');
```

```
Query OK, 1 row affected (0.00 sec)
mysql> INSERT IGNORE INTO person_tbl (last_name, first_name)
    -> VALUES( 'Jay', 'Thomas');
Query OK, 0 rows affected (0.00 sec)
```

使用 REPLACE 而不是 INSERT。如果记录是一个新记录，使用 INSERT 就可以了。如果是一个重复记录，新的记录将会替换旧有记录。

```
mysql> REPLACE INTO person_tbl (last_name, first_name)
    -> VALUES( 'Ajay', 'Kumar');
Query OK, 1 row affected (0.00 sec)
mysql> REPLACE INTO person_tbl (last_name, first_name)
    -> VALUES( 'Ajay', 'Kumar');
Query OK, 2 rows affected (0.00 sec)
```

应该根据想要达到的重复处理行为来选择INSERT IGNORE 和 REPLACE。INSERT IGNORE 会保存重复记录的第一个，抛弃其余的记录；REPLACE 则正好相反，保存最后一个记录，去掉在其之前的所有记录。

强制唯一性的另一种办法是为表添加 UNIQUE 索引而不是主键。

```
CREATE TABLE person_tbl
(
   first_name CHAR(20) NOT NULL,
   last_name CHAR(20) NOT NULL,
   sex CHAR(10)
   UNIQUE (last_name, first_name)
);
```

确认重复记录，并计算重复记录数

下面是计算表中姓名记录重复的查询：

```
mysql> SELECT COUNT(*) as repetitions, last_name, first_name
    -> FROM person_tbl
    -> GROUP BY last_name, first_name
    -> HAVING repetitions > 1;
```

该查询返回表 person_tbl 中所有的重复记录。一般来说，要想确认重复记录，需要采取以下步骤：

- 确定可能产生重复记录的列。
- 在列选择列表中显示所有列，利用 COUNT(*) 。
- 利用 GROUP BY 子句列出列。

- 加入 HAVING 子句排除唯一值。需要让组计数大于1。

从查询结果中消除重复记录

使用DISTINCT 和 SELECT 语句来查找表中的重复记录。

```
mysql> SELECT DISTINCT last_name, first_name
    -> FROM person_tbl
    -> ORDER BY last_name;
```

另一种办法是添加 GROUP BY 子句，命名选择的列。消除重复记录并只选择指定列中的唯一值组合。

```
mysql> SELECT last_name, first_name
    -> FROM person_tbl
    -> GROUP BY (last_name, first_name);
```

使用表替换去除重复记录

下面这种技巧也可以消除表中存在的所有重复记录。

```
mysql> CREATE TABLE tmp SELECT last_name, first_name, sex
    ->              FROM person_tbl;
    ->              GROUP BY (last_name, first_name);
mysql> DROP TABLE person_tbl;
mysql> ALTER TABLE tmp RENAME TO person_tbl;
```

为表加入 INDEX 或 PRIMARY KEY 。即使该表已经存在，你也可以利用这种技巧消除重复记录，这种做法将来也依然保险。

```
mysql> ALTER IGNORE TABLE person_tbl
    -> ADD PRIMARY KEY (last_name, first_name);
```

MySQL SQL Injection

如果通过网页接收用户输入，而后再把这些数据插入到数据库中，那么你可能就会碰到 SQL 注入式攻击。本节简要介绍如何防范这种攻击，确保脚本和 MySQL 语句的安全性。

注入式攻击往往发生在要求用户输入时，比如说要求他们输入自己的名字，但是他们却输入了一段 MySQL 语句，不知不觉地运行在数据库上。

永远不要相信用户所提供的数据，只有在验证无误后，才能去处理数据。通常，利用模式匹配来实现。在下面这个范例中，username（用户名）被限定为字母数字混合编制的字符串，再加上下划线，长度限定为8~20个字符之间。当然，可以按需要修改这些规范。

```
if (preg_match("/^\w{8,20}$/", $_GET['username'], $matches))
{
  $result = mysql_query("SELECT * FROM users
                    WHERE username=$matches[0]");
}
 else
{
  echo "username not accepted";
}
```

为了暴露问题所在，请考虑下面这段代码：

```
// 本应该的输入
$name = "Qadir'; DELETE FROM users;";
mysql_query("SELECT * FROM users WHERE name='{$name}'");
```

函数调用原本会从 users 表中获取一个记录。name 列与用户所指定的名字所匹配。在一般情况下，$name 会包含字母数字混合编制的字符，或许还包含空格，such as the string ilia. 但这里，为 $name 添加了一个全新的查询，数据库调用就变成了灾难：注入的 DELETE 查询会删除 users 表中所有的记录。

幸运的是，如果使用 MySQL，`mysql_query()` 函数不允许堆叠查询，或在一个函数调用中执行多个查询。如果尝试使用堆叠查询，则调用会失败。

然而，有些 PHP 数据库扩展，比如 SQLite 或 PostgreSQL，却能很好地执行堆叠查询，能够执行一个字符串中所提供的所有查询，从而造成严重的安全隐患。

防止 SQL 注入式攻击

使用 PERL 或 PHP 这样的脚本语言，可以很巧妙地处理转义字符。MySQL 针对 PHP 的扩展也提供了 `mysql_real_escape_string()` ，可以将输入的字符转义为MySQL所特有的字符。

```
if (get_magic_quotes_gpc())
{
  $name = stripslashes($name);
}
$name = mysql_real_escape_string($name);
mysql_query("SELECT * FROM users WHERE name='{$name}'");
```

LIKE 窘境

为了解决 LIKE 困境，常用的转义机制必须将 用户所输入的 `%` 和 `_` 字符转义为字面值。使用 `addcslashes()` 能为你指定一个转义字符范围。

```
$sub = addcslashes(mysql_real_escape_string("%something_"), "%_");
// $sub == \%something\_
mysql_query("SELECT * FROM messages WHERE subject LIKE '{$sub}%'");
```

MySQL 数据导出

将表中数据导出为一个文本文件，最简单的方法是用 SELECT...INTO OUTFILE 语句，它会将查询结果直接导出为服务器主机上的一个文件。

利用 SELECT...INTO OUTFILE 语句组合导出数据

该语句组合的语法为：使用正常的 SELECT 语句，后跟 INTO OUTFILE，最后加上要导出的文件名。默认的输出格式和 LOAD DATA 一样，因此下列语句会将表 tutorials_tbl 导出为 /tmp/tutorials.txt，其中的数据以制表符分隔开，以换行符作为每行的终止符。

```
mysql> SELECT * FROM tutorials_tbl
    -> INTO OUTFILE '/tmp/tutorials.txt';
```

你可以通过一些选项来改变输出格式，来指定如何以引用并限定列与记录。下面这个例子将表 tutorials_tbl 以逗号分隔各值，以 CRLF（回车换行符）来作为行的终止符：

```
mysql> SELECT * FROM passwd INTO OUTFILE '/tmp/tutorials.txt'
    -> FIELDS TERMINATED BY ',' ENCLOSED BY '"'
    -> LINES TERMINATED BY '\r\n';
```

SELECT ... INTO OUTFILE 具有下列特点：

- 输出文件直接由 MySQL 服务器创建，因此文件名应该指明其在服务器主机上的保存位置。该语句没有 LOCAL 版，这一点跟 LOAD DATA 不同。
- 必须拥有 MySQL 的 FILE 权限，才能执行 SELECT ... INTO OUTFILE。
- 输出文件不能是已有文件。这一特点保证了 MySQL 不会覆盖掉一些可能是非常重要的文件。
- 你必须有服务器主机的登录账号，或者能够利用其它方式获取主机文件，否则 SELECT ... INTO OUTFILE 对你来说没有任何用处。
- 在 UNIX 系统下，创建的文件是全局可读的，但可写权限却属于 MySQL 服务器。这意味着虽然你可以读取文件，但可能无法删除它。

将表导出为原始数据

`mysqldump` 程序用于复制或备份表与数据库。它能把表输出为一个原始数据文件，或者是一个能重建表中记录的 INSERT 语句集合。

要想把表转储为一个数据文件，必须指定一个 `--tab` 选项，用它来指明 MySQL 服务器写入文件的目录。

例如，把数据库 TUTORIALS 中的表 tutorials_tbl 转储为 /tmp 中的一个文件，需要使用如下命令：

```
$ mysqldump -u root -p --no-create-info \
        --tab=/tmp TUTORIALS tutorials_tbl
password ******
```

将表内容或定义以 SQL 格式导出

以 SQL 格式将表导出为文件，使用类似下列命令：

```
$ mysqldump -u root -p TUTORIALS tutorials_tbl > dump.txt
password ******
```

这样创建的文件将包含如下内容：

```
##
-- MySQL dump 8.23## -- Host: localhost    Database: TUTORIALS-- Server version       3.23.58
##  ## -- Table structure for table `tutorials_tbl`
CREATE TABLE tutorials_tbl (
  tutorial_id int(11) NOT NULL auto_increment,
  tutorial_title varchar(100) NOT NULL default '',
  tutorial_author varchar(40) NOT NULL default '',
  submission_date date default NULL,
  PRIMARY KEY  (tutorial_id),
  UNIQUE KEY AUTHOR_INDEX (tutorial_author)
) TYPE=MyISAM;
##  ## -- Dumping data for table `tutorials_tbl`
INSERT INTO tutorials_tbl
      VALUES (1,'Learn PHP','John Poul','2007-05-24');
INSERT INTO tutorials_tbl
      VALUES (2,'Learn MySQL','Abdul S','2007-05-24');
INSERT INTO tutorials_tbl
      VALUES (3,'JAVA Tutorial','Sanjay','2007-05-06');
```

转储多张表，按照数据库命名》》。转储整个数据库，不需要命名数据库中的任何表：

```
$ mysqldump -u root -p TUTORIALS > database_dump.txt
password ******
```

备份主机上的所有数据库，使用如下命令：

```
$ mysqldump -u root -p --all-databases > database_dump.txt
password ******
```

自MySQL 3.23.12版本开始，可以使用 `--all-databases` 选项。

这种方法可以实现数据库备份。

将一台主机上的表或数据库复制到另一台主机上

如果想把一台 MySQL 服务器上的表或数据库复制到另一台主机上，可以使用 mysqldump 程序，加上数据库名称和表名称。

在源主机上运行以下命令，它会将整个数据库都转储到 dump.txt 文件中。

```
$ mysqldump -u root -p database_name table_name > dump.txt
password *****
```

如前所述，你可以将整个数据库都复制下来，无需使用任何具体的表名称。

接下来，在另一台主机上ftp dump.txt 文件，并运行如下命令。在运行这行命令之前，先要确保已经在目标服务器上创建了 database_name。

```
$ mysql -u root -p database_name < dump.txt
password *****
```

在主机间复制数据库也可以使用另一种方法，它的优点就是无需使用中介文件。将 mysqldump的输出结果直接通过网络传到远端的 MySQL 服务器上。如果你能从源数据库所在的主机上连接到两个服务器上，使用如下命令（一定要确保你能访问两台服务器）：

```
$ mysqldump -u root -p database_name \
    | mysql -h other-host.com database_name
```

以上命令的 mysqldump 部分会连接本地服务器，将转储结果写入管线。剩下的命令连接到另一台主机的远端服务器上，读取管线上传来的转储结果，将每个语句送到目的主机所在的服务器上。

MySQL 数据导入

MySQL 可以采用2种简单的方法将之前备份文件中的数据加载进 MySQL 数据库。

利用 LOAD DATA 导入数据

MySQL 利用 LOAD DATA 语句作为批量数据加载器。下面这个范例将从当前目录中读取 dump.txt 文件，然后把它加载进当前数据库的表 mytbl 中。

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

- 如果不写 LOCAL 关键字，MySQL 会从服务器主机文件系统的根目录开始，以完整指明文件位置的绝对路径名方式开始查找数据文件。MySQL 会从给定位置读取文件。
- 默认情况下，LOAD DATA 假定数据文件中每行都由换行符所终止，每行的数据值由制表符所分隔开。
- 为了明确指定文件格式，使用 FIELDS 子句来描述行内字段特征， LINES 子句指定行末尾序列。下例中的 LOAD DATA 语句表明，数据文件中的值由冒号（ `:` ）分隔，每行由换行符及回车符所终止。

```
mysql> LOAD DATA LOCAL INFILE 'dump.txt' INTO TABLE mytbl
 -> FIELDS TERMINATED BY ':'
 -> LINES TERMINATED BY '\r\n';
```

- LOAD DATA 假定数据文件中的列的顺序与表中列的顺序相同。如果不为真，可以指定一个列表来指示数据文件中具体表列的加载方式。假如表有3个列：a、b和c，但数据文件中对应的是列b、c与a，则可以这样加载。

```
mysql> LOAD DATA LOCAL INFILE 'dump.txt'
   -> INTO TABLE mytbl (b, c, a);
```

利用mysqlimport 导入数据

MySQL 还包含一个工具程序： `mysqlimport` 。它相当于 LOAD DATA 的一个封装器，因而你可以直接从命令行中加载输入文件。

将 dump.txt 中的数据加载进表 mytbl，可以在 UNIX 系统的命令行中使用以下命令：

```
$ mysqlimport -u root -p --local database_name dump.txt
password *****
```

如果使用mysqlimport，命令行选项就会提供格式说明符。mysqlimport 命令作用相当于前面的两个LOAD DATA 语句，语法如下：

```
$ mysqlimport -u root -p --local --fields-terminated-by=":" \
  --lines-terminated-by="\r\n"  database_name dump.txt
password *****
```

对于 mysqlimport 来说，你怎么指定选项的次序并不重要，只要把它们写在数据库名称前面就可以了。

mysqlimport 语句使用 --columns 选项来指定列次序。

```
$ mysqlimport -u root -p --local --columns=b,c,a \
   database_name dump.txt
password *****
```

处理引号与特殊字符

FIELDS 子句能指定除了 TERMINATED BY 之外的其他格式选项。默认情况下，LOAD DATA 会假定值不加引号，并把反斜杠（ \ ）解释为表示特殊意义的转义字符。要想明确指定引号字符，需使用 ENCLOSED BY 。MySQL 会在处理输入时将该字符从数据值末尾清除掉。改变默认的转义字符，需使用 ESCAPED BY。

在指定 ENCLOSED BY 来表示引号字符应该从数据值末尾清除时，有可能在数据值中包含引号字符，或在其之前添加转义字符。比如，如果引号和转义字符是 " 和 \ ，那么输入值 "a""b\"c" 就会被解读为 a"b"c 。

对于 mysqlimport 而言，相应的指定引号和转义值的命令行选项是 --fields-enclosed-by 和 --fields-escaped-by 。

2

MySQL 帮助资源

一些有用的 MySQL 函数与子句

下面是一些重要的 MySQL 函数。每一类函数都配有详尽的适用范例。

- MySQL Group By 子句 (页 104) MySQL 的 GROUP BY 语句常与 SUM 这样的 SQL 聚合函数一起使用，可对特定列的结果集采取多种方式进行组合。
- MySQL IN 子句 (页 0) 一个可以与任何 MySQL 查询组合使用，用来指定条件的子句。
- MySQL BETWEEN 子句 (页 99) 一个可以与任何 MySQL 查询组合使用，用来指定条件的子句。
- MySQL UNION 关键字 (页 0) 使用 UNION 操作，将多个结果集合并为一个结果集。
- MySQL COUNT 函数 (页 103) 用来计算数据库表中的行数的聚合函数。
- MySQL MAX 函数 (页 107) 能够选择列中最大值的聚合函数。
- MySQL MIN 函数 (页 109) 能够选择列中最小值的聚合函数。
- MySQL AVG 函数 (页 97) 选择列的平均值的聚合函数。
- MySQL SUM 函数 (页 149) 选择数字型列的总数值的聚合函数。
- MySQL SQRT 函数 (页 127) 计算指定数字的平方根。
- MySQL RAND 函数 (页 125) 用于生成随机数的函数。
- MySQL CONCAT 函数 (页 101) 用于连接 MySQL 命令中任意字符串的函数。
- MySQL 日期与时间方面的函数 (页 151) 与日期和时间相关的一系列 MySQL 函数。
- MySQL 数值函数 (页 111) 用于处理数值的一系列 MySQL 函数。
- MySQL 字符串函数 (页 129) 处理字符串的一系列 MySQL 函数。

一些非常有用的学习资源

如果你想把自己的网站、读过的好书，或者其他有用的学习资源推荐给大家，请联系我们。

一些非常不错的 MySQL 学习站点

- MySQL 与 PERL (http://www.tutorialspoint.com/perl/perl_database.htm) 这是我们出品的一套讲解如何利用 PERL 和 DBI 模块 使用 MySQL 的教程。你从中可了解到所需的 MySQL 操作知识以及应用范例。
- MySQL 官方网站 (http://www.mysql.com/) 可在此下载最新版本的 MySQL，了解最热门的 MySQL 业界新闻。对于想用 MySQL 构建动态网站的开发者来说，它的邮件列表也是一个非常不错的学习资源。
- PHP 官方网站 (http://php.net/) PHP 相关内容的完整汇总，包含了最新的 PHP 更新及函数手册。
- MySQL 维基百科 (http://en.wikipedia.org/wiki/MySQL) 值得一读的小文章。
- MySQL 参考手册 (http://dev.mysql.com/doc/refman/5.6/en/index.html) 有关 MySQL 内容的完整官方文档。
- MySQL-sr-lib (http://www.nongnu.org/mysql-sr-lib/) MySQL通用存储库例程。包含了经常需要的完整例程。这是一款免费软件，根据 GNU 通用公共许可证的相关条款，你可以重新发布并修改它。
- 理解 SQL (http://www.faqs.org/docs/ppbook/c1164.htm) SQL 新手必备教程。SQL（结构化查询语言）是一种成熟、健壮、通用的关系型查询语言。

3 附录

MySQL AVG 函数

AVG 函数用来在不同记录中找出某一字段的平均值。

例如，在表 employee_tbl 中，所有记录如下：

```
mysql> SELECT * FROM employee_tbl;
+------+------+------------+-------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+-------------------+
|   1 | John | 2007-01-24 |              250 |
|   2 | Ram  | 2007-05-27 |              220 |
|   3 | Jack | 2007-05-06 |              170 |
|   3 | Jack | 2007-04-06 |              100 |
|   4 | Jill | 2007-04-06 |              220 |
|   5 | Zara | 2007-06-06 |              300 |
|   5 | Zara | 2007-02-06 |              350 |
+------+------+------------+-------------------+
7 rows in set (0.00 sec)
```

如果想根据该表计算字段 dialy_typing_pages 的平均值，可以使用如下命令：

```
mysql> SELECT AVG(daily_typing_pages)
    -> FROM employee_tbl;
+-------------------------+
| AVG(daily_typing_pages) |
+-------------------------+
|              230.0000 |
+-------------------------+
1 row in set (0.03 sec)
```

还可以使用 GROUP BY 子句来计算多种记录集的平均值。下面这个范例将计算每个人的所有记录的平均值，将得到每个人的平均输入页面。

```
mysql> SELECT name, AVG(daily_typing_pages)
    -> FROM employee_tbl GROUP BY name;
+------+-------------------------+
| name | AVG(daily_typing_pages) |
+------+-------------------------+
| Jack |              135.0000 |
| Jill |              220.0000 |
| John |              250.0000 |
| Ram  |              220.0000 |
```

```
| Zara |              325.0000 |
+------+-----------------------+
5 rows in set (0.20 sec)
```

MySQL BETWEEN 子句

BETWEEN 子句可以代替“大于等于XX，并且小于等于XX”这样的条件。

为了理解 BETWEEN 子句，依旧来看 employee_tbl 这个例子，它的所有记录如下：

```
mysql> SELECT * FROM employee_tbl;
+------+------+------------+-------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+-------------------+
|   1 | John | 2007-01-24 |              250 |
|   2 | Ram  | 2007-05-27 |              220 |
|   3 | Jack | 2007-05-06 |              170 |
|   3 | Jack | 2007-04-06 |              100 |
|   4 | Jill | 2007-04-06 |              220 |
|   5 | Zara | 2007-06-06 |              300 |
|   5 | Zara | 2007-02-06 |              350 |
+------+------+------------+-------------------+
7 rows in set (0.00 sec)
```

假设要从上表中获取 daily_typing_pages 大于或等于170，小于或等于300的数据。可以利用 >= 与 <= 条件来实现。

```
mysql>SELECT * FROM employee_tbl
   ->WHERE daily_typing_pages >= 170 AND
   ->daily_typing_pages <= 300;
+------+------+------------+-------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+-------------------+
|   1 | John | 2007-01-24 |              250 |
|   2 | Ram  | 2007-05-27 |              220 |
|   3 | Jack | 2007-05-06 |              170 |
|   4 | Jill | 2007-04-06 |              220 |
|   5 | Zara | 2007-06-06 |              300 |
+------+------+------------+-------------------+
5 rows in set (0.03 sec)
```

用 BETWEEN 实现起来就简单多了，如下所示：

```
mysql> SELECT * FROM employee_tbl
   -> WHERE daily_typing_pages BETWEEN 170 AND 300;
+------+------+------------+-------------------+
| id   | name | work_date  | daily_typing_pages |
```

```
+------+------+------------+--------------------+
|   1 | John | 2007-01-24 |              250 |
|   2 | Ram  | 2007-05-27 |              220 |
|   3 | Jack | 2007-05-06 |              170 |
|   4 | Jill | 2007-04-06 |              220 |
|   5 | Zara | 2007-06-06 |              300 |
+------+------+------------+--------------------+
5 rows in set (0.03 sec)
```

MySQL CONCAT 函数

CONCAT 函数用于将两个字符串合并为一个字符串。如下例所示：

```
mysql> SELECT CONCAT('FIRST ', 'SECOND');
+---------------------------+
| CONCAT('FIRST ', 'SECOND') |
+---------------------------+
| FIRST SECOND              |
+---------------------------+
1 row in set (0.00 sec)
```

以下面这个表 employee_tbl 为例：

```
mysql> SELECT * FROM employee_tbl;
+------+------+------------+-------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+-------------------+
|   1 | John | 2007-01-24 |              250 |
|   2 | Ram  | 2007-05-27 |               220 |
|   3 | Jack | 2007-05-06 |              170 |
|   3 | Jack | 2007-04-06 |              100 |
|   4 | Jill | 2007-04-06 |             220 |
|   5 | Zara | 2007-06-06 |              300 |
|   5 | Zara | 2007-02-06 |              350 |
+------+------+------------+-------------------+
7 rows in set (0.00 sec)
```

如果需要将id、name以及work_date这三个字段值合并起来，可以使用如下命令：

```
mysql> SELECT CONCAT(id, name, work_date)
    -> FROM employee_tbl;
+----------------------------+
| CONCAT(id, name, work_date) |
+----------------------------+
| 1John2007-01-24            |
| 2Ram2007-05-27             |
| 3Jack2007-05-06            |
| 3Jack2007-04-06            |
| 4Jill2007-04-06            |
| 5Zara2007-06-06            |
| 5Zara2007-02-06            |
```

```
+----------------------------+
7 rows in set (0.00 sec)
```

MySQL COUNT 函数

COUNT 函数的用法很简单，就是为了统计记录数。SELECT 语句所返回的。

为了理解这个函数，让我们再次搬出 employee_tbl 表，它的所有记录如下所示：

```
mysql> SELECT * FROM employee_tbl;
+------+------+------------+--------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+--------------------+
|   1 | John | 2007-01-24 |              250 |
|   2 | Ram  | 2007-05-27 |              220 |
|   3 | Jack | 2007-05-06 |              170 |
|   3 | Jack | 2007-04-06 |              100 |
|   4 | Jill | 2007-04-06 |              220 |
|   5 | Zara | 2007-06-06 |              300 |
|   5 | Zara | 2007-02-06 |              350 |
+------+------+------------+--------------------+
7 rows in set (0.00 sec)
```

假设要根据上表统计行的总数，当然可以这样做：

```
mysql>SELECT COUNT(*) FROM employee_tbl ;
+----------+
| COUNT(*) |
+----------+
|       7 |
+----------+
1 row in set (0.01 sec)
```

同样，如果希望计算 Zara 的记录数，可以这样实现：

```
mysql>SELECT COUNT(*) FROM employee_tbl
    -> WHERE name="Zara";
+----------+
| COUNT(*) |
+----------+
|       2 |
+----------+
1 row in set (0.04 sec)
```

> 注意：由于 SQL 查询对大小写不敏感，所以在 WHERE 条件中，无论是写成 ZARA 还是 Zara，结果都是一样的。

MySQL Group By 子句

可以使用 GROUP BY 子句对列中的值进行分组。如果你愿意，还可以对列实施某种计算。可以对分组的列使用COUNT、SUM以及AVG等函数。

为了理解 GROUP BY 子句，考虑表 employee_tbl，它包含如下记录：

```
mysql> SELECT * FROM employee_tbl;
+------+------+------------+--------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+--------------------+
|   1 | John | 2007-01-24 |              250 |
|   2 | Ram  | 2007-05-27 |              220 |
|   3 | Jack | 2007-05-06 |              170 |
|   3 | Jack | 2007-04-06 |              100 |
|   4 | Jill | 2007-04-06 |              220 |
|   5 | Zara | 2007-06-06 |              300 |
|   5 | Zara | 2007-02-06 |              350 |
+------+------+------------+--------------------+
7 rows in set (0.00 sec)
```

现在，根据上面这张表，我们来计算一下每位员工的工作天数。

如果像下面这样写 SQL 查询，那么将得到下列结果：

```
mysql> SELECT COUNT(*) FROM employee_tbl;
+--------------------------+
| COUNT(*)                 |
+--------------------------+
| 7                        |
+--------------------------+
```

但这并不符合预期，我们希望的是输出每位员工输入的页数。可以结合使用聚合函数与 GROUP BY 子句，如下所示：

```
mysql> SELECT name, COUNT(*)
    -> FROM   employee_tbl
    -> GROUP BY name;
+------+----------+
| name | COUNT(*) |
+------+----------+
| Jack |        2 |
| Jill |        1 |
```

```
| John |      1 |
| Ram  |      1 |
| Zara |      2 |
+------+----------+
5 rows in set (0.04 sec)
```

以后，我们还将介绍更多 GROUP BY 与 SUM、AVG 等函数相结合的用法。

MySQL IN 子句

可以使用 IN 子句代替许多 OR 条件。

要想理解 IN 子句，还以表 employee_tbl 为例，它的所有记录如下所示：

```
mysql> SELECT * FROM employee_tbl;
+------+------+------------+--------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+--------------------+
|   1 | John | 2007-01-24 |              250 |
|   2 | Ram  | 2007-05-27 |              220 |
|   3 | Jack | 2007-05-06 |              170 |
|   3 | Jack | 2007-04-06 |              100 |
|   4 | Jill | 2007-04-06 |              220 |
|   5 | Zara | 2007-06-06 |              300 |
|   5 | Zara | 2007-02-06 |              350 |
+------+------+------------+--------------------+
7 rows in set (0.00 sec)
```

现在，我们希望根据以上表格，显示出 daily_typing_pages 等于250、220和170这三个值的记录。利用 OR 条件实现如下：

```
mysql>SELECT * FROM employee_tbl
   ->WHERE daily_typing_pages= 250 OR
   ->daily_typing_pages= 220 OR  daily_typing_pages= 170;
+------+------+------------+--------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+--------------------+
|   1 | John | 2007-01-24 |              250 |
|   2 | Ram  | 2007-05-27 |              220 |
|   3 | Jack | 2007-05-06 |              170 |
|   4 | Jill | 2007-04-06 |              220 |
+------+------+------------+--------------------+
4 rows in set (0.02 sec)
```

同样的实现也可以用 IN子句 来完成：

```
mysql> SELECT * FROM employee_tbl
    -> WHERE daily_typing_pages IN ( 250, 220, 170 );
+------+------+------------+--------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+--------------------+
|    1 | John | 2007-01-24 |                250 |
|    2 | Ram  | 2007-05-27 |                220 |
|    3 | Jack | 2007-05-06 |                170 |
|    4 | Jill | 2007-04-06 |                220 |
+------+------+------------+--------------------+
4 rows in set (0.02 sec)
```

MySQL MAX 函数

MAX 函数就是用来寻找记录集中的最大值的。

为了理解这个函数，再次搬出 employee_tbl 表，其内容如下所示：

```
mysql> SELECT * FROM employee_tbl;
+------+------+------------+-------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+-------------------+
|   1 | John | 2007-01-24 |              250 |
|   2 | Ram  | 2007-05-27 |              220 |
|   3 | Jack | 2007-05-06 |              170 |
|   3 | Jack | 2007-04-06 |              100 |
|   4 | Jill | 2007-04-06 |              220 |
|   5 | Zara | 2007-06-06 |              300 |
|   5 | Zara | 2007-02-06 |              350 |
+------+------+------------+-------------------+
7 rows in set (0.00 sec)
```

假设要获取上表中 daily_typing_pages 的最大值，可以这样实现：

```
mysql> SELECT MAX(daily_typing_pages)
    -> FROM employee_tbl;
+-------------------------+
| MAX(daily_typing_pages) |
+-------------------------+
|                     350 |
+-------------------------+
1 row in set (0.00 sec)
```

还可以结合利用 GROUP BY 子句，找出每个名字的最大值：

```
mysql> SELECT id, name, MAX(daily_typing_pages)
    -> FROM employee_tbl GROUP BY name;
+------+------+-------------------------+
| id   | name | MAX(daily_typing_pages) |
+------+------+-------------------------+
|   3 | Jack |                     170 |
|   4 | Jill |                     220 |
|   1 | John |                     250 |
|   2 | Ram  |                     220 |
|   5 | Zara |                     350 |
```

```
+------+------+-----------------------+
5 rows in set (0.00 sec)
```

也可以组合使用 MIN 和 MAX 函数找出最小值，如下所示：

```
mysql> SELECT MIN(daily_typing_pages) least, MAX(daily_typing_pages) max
    -> FROM employee_tbl;
+-------+------+
| least | max  |
+-------+------+
|   100 |  350 |
+-------+------+
1 row in set (0.01 sec)
```

MySQL MIN 函数

MIN 函数用于寻找记录集中拥有最小值的记录。

为了理解 MIN 函数，还是以 employee_tbl 表为例，它的所有记录如下所示：

```
mysql> SELECT * FROM employee_tbl;
+------+------+------------+--------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+--------------------+
|   1 | John | 2007-01-24 |              250 |
|   2 | Ram  | 2007-05-27 |              220 |
|   3 | Jack | 2007-05-06 |              170 |
|   3 | Jack | 2007-04-06 |              100 |
|   4 | Jill | 2007-04-06 |              220 |
|   5 | Zara | 2007-06-06 |              300 |
|   5 | Zara | 2007-02-06 |              350 |
+------+------+------------+--------------------+
7 rows in set (0.00 sec)
```

假设需要根据上表获取 daily_typing_pages 的最小值，只需使用如下命令即可：

```
mysql> SELECT MIN(daily_typing_pages)
    -> FROM employee_tbl;
+------------------------+
| MIN(daily_typing_pages) |
+------------------------+
|                    100 |
+------------------------+
1 row in set (0.00 sec)
```

也可以使用 GROUP BY 子句找到每一名字下的最小值记录。

```
mysql>SELECT id, name, MIN(daily_typing_pages)
    -> FROM employee_tbl GROUP BY name;
+------+------+------------------------+
| id   | name | MIN(daily_typing_pages) |
+------+------+------------------------+
|   3 | Jack |                    100 |
|   4 | Jill |                    220 |
|   1 | John |                    250 |
|   2 | Ram  |                    220 |
|   5 | Zara |                    300 |
```

```
+------+------+-----------------------+
5 rows in set (0.00 sec)
```

还可以组合使用 MIN 与 MAX 函数找出最小值，如下所示：

```
mysql> SELECT MIN(daily_typing_pages) least, MAX(daily_typing_pages) max
    -> FROM employee_tbl;
+-------+------+
| least | max  |
+-------+------+
|   100 |  350 |
+-------+------+
1 row in set (0.01 sec)
```

MySQL 数值函数

函数名称	函数说明
ABS() (页 112)	返回数值表达式的绝对值
ACOS() (页 113)	返回数值表达式的反余弦值。如果参数未在[−1, 1]区间内，则返回 NULL
ASIN() (页 113)	返回数值表达式的反正弦值。如果参数未在[−1, 1]区间内，则返回 NULL
ATAN() (页 114)	返回数值表达式的反正切值
ATAN2() (页 114)	返回两个参数的反正切值
BIT_AND() (页 114)	返回表达式参数中的所有二进制位的**按位与**运算结果
BIT_COUNT() (页 115)	返回传入的二进制值的字符串形式
BIT_OR() (页 115)	返回表达式参数中的所有二进制位的**按位或**运算结果
CEIL() (页 116)	返回值为不小于传入数值表达式的最小整数值
CEILING() (页 0)	同 CEIL() 返回值为不小于传入数值表达式的最小整数值
CONV() (页 116)	转换数值表达式的进制
COS() (页 116)	返回所传入数值表达式（以弧度计）的余弦值
COT() (页 117)	返回所传入数值表达式的余切值
DEGREES() (页 117)	将数值表达式参数从弧度值转变为角度值
EXP() (页 117)	返回以e（自然对数的底数）为底，以所传入的数值表达式为指数的幂
FLOOR() (页 118)	返回不大于所传入数值表达式的最大整数
FORMAT() (页 118)	将数值表达式参数四舍五入到一定的小数位
GREATEST() (页 118)	返回传入参数的最大值

函数名称	函数说明
INTERVAL() (页 119)	比较所传入的多个表达式：expr1 、expr2 、expr3 ……，如果 expr1 < expr2 ，则返回0；如果 expr1 < expr3 ，则返回1……以此类推
LEAST() (页 119)	返回传入参数中的最小值
LOG() (页 119)	返回传入数值表达式的自然对数
LOG10() (页 120)	返回传入数值表达式的常用对数（以10为底的对数）
MOD() (页 120)	返回参数相除的余数
OCT() (页 120)	返回传入数值表达式的八进制数值的字符串表现形式。如果传入值为 NULL，则返回 NULL
PI() (页 121)	返回 π 值
POW() (页 121)	返回两个参数的幂运算结果，其中一个参数为底，另一个参数为它的指数。
POWER() (页 0)	返回两个参数的幂运算结果，其中一个参数为底，另一个参数为它的指数。
RADIANS() (页 121)	将参数由角度值转换成弧度值
ROUND() (页 1)	将所传入数值表达式四舍五入为整数。也可以用来将参数四舍五入到一定的小数位
SIN() (页 123)	返回参数（以弧度计）的正弦值
SQRT() (页 123)	返回参数的非负平方根
STD() (页 123)	返回参数的标准方差值
STDDEV() (页 0)	返回参数的标准方差值
TAN() (页 124)	返回参数（以弧度计）的正切值
TRUNCATE() (页 124)	将数值参数 expr1 的小数位截取到 expr2 位如果 expr2 为0，则结果没有小数位。

ABS(X) ()

返回参数 X 的绝对值。如下例所示：

```
mysql> SELECT ABS(2);
+-----------------------------------------------------------+
| ABS(2)                                                    |
+-----------------------------------------------------------+
| 2                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ABS(-2);
+-----------------------------------------------------------+
| ABS(2)                                                    |
+-----------------------------------------------------------+
| 2                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

ACOS(X) ()

返回参数 X 的反余弦值。参数 X 的取值区间为 [-1, 1]，如果不在该区间内，则返回 NULL 值。实例如下：

```
mysql> SELECT ACOS(1);
+-----------------------------------------------------------+
| ACOS(1)                                                   |
+-----------------------------------------------------------+
| 0.000000                                                  |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

ASIN(X) ()

返回参数 X 的反正弦值。参数 X 的取值区间为 [-1, 1]，如果不在该区间内，则返回 NULL 值。实例如下：

```
mysql> SELECT ASIN(1);
+-----------------------------------------------------------+
| ASIN(1)                                                   |
+-----------------------------------------------------------+
| 1.5707963267949                                           |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

ATAN(X) ()

返回参数 X 的反正切值。

```
mysql> SELECT ATAN(1);
+-------------------------------------------------------+
| ATAN(1)                                               |
+-------------------------------------------------------+
| 0.78539816339745                                      |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

ATAN2(Y,X) ()

返回两个参数 X 与 Y 的反正切值，类似于 Y/X 的反正切值，但两个参数的符号是用来确定所得结果的象限的。

```
mysql> SELECT ATAN2(3,6);
+-------------------------------------------------------+
| ATAN2(3,6)                                            |
+-------------------------------------------------------+
| 0.46364760900081                                      |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

BIT_AND(expression) ()

返回表达式参数 expression 中的所有二进制位的**按位与**运算结果。按位与运算的规则是这样的：如果两个同等位都是同样值（0或1），则返回1，否则返回0。函数本身返回的是一个64位的整形值，如果没有匹配项，则返回18446744073709551615。在下面的示例中，对表 CARS 按照Maker 字段进行分组， 然后再对 PRICE（价格）字段执行 BIT_AND() 函数。

```
mysql> SELECT
       MAKER, BIT_AND(PRICE) BITS
       FROM CARS GROUP BY MAKER
+-------------------------------------------------------+
|MAKER        BITS                          |
+-------------------------------------------------------+
|CHRYSLER      512                          |
```

```
|FORD          12488                          |
|HONDA          2144                           |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

BIT_COUNT(numeric_value) ()

该函数会将参数 numeric_value 转化成二进制数，然后再返回这个二进制数中1的个数。下例展示了如何对一些数使用 BIT_COUNT() 函数。

```
mysql> SELECT
       BIT_COUNT(2) AS TWO,
       BIT_COUNT(4) AS FOUR,
       BIT_COUNT(7) AS SEVEN
+-----+------+-------+
| TWO | FOUR | SEVEN |
+-----+------+-------+
|  1 |   1 |    3 |
+-----+------+-------+
1 row in set (0.00 sec)
```

BIT_OR(expression) ()

按位或运算函数。返回表达式参数 expression 中所有位的按位或运算结果。其中的基本原理是：如果相对应的位匹配（就是相同的意思，同为0或1），则返回0，否则返回1。函数本身返回的是一个64位的整形数值，如果记录并不匹配，则返回0。下例将对对 CARS 表按照 Maker 分组，然后对 PRICE 字段执行 BIT_OR() 函数。

```
mysql> SELECT
       MAKER, BIT_OR(PRICE) BITS
       FROM CARS GROUP BY MAKER
+-----------------------------------------------------------+
|MAKER          BITS                           |
+-----------------------------------------------------------+
|CHRYSLER       62293                            |
|FORD           16127                          |
|HONDA          32766                            |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

CEIL(X) 与 CEILING(X) ()

返回不小于 X 的最小整型值。示例如下：

```
mysql> SELECT CEILING(3.46);
+-------------------------------------------------------+
| CEILING(3.46)                                         |
+-------------------------------------------------------+
| 4                                                     |
+-------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CEIL(-6.43);
+-------------------------------------------------------+
| CEIL(-6.43)                                           |
+-------------------------------------------------------+
| -6                                                    |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

CONV(N,from_base,to_base) ()

该函数用于在不同进制间转换数值。将数值 N 从 初始进制参数 from_base 转换为目标进制参数 to_base，然后以字符串的形式返回。该函数可使用的进制范围为2-36。如果函数参数中有一个为 NULL 值，则函数返回 NULL。下例中，将16进制数5转换为了2进制数101。

```
mysql> SELECT CONV(5,16,2);
+-------------------------------------------------------+
| CONV(5,16,2)                                          |
+-------------------------------------------------------+
| 101                                                   |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

COS(X) ()

以弧度值形式返回 X 的余弦值。示例如下：

```
mysql>SELECT COS(90);
+-------------------------------------------------------+
```

```
| COS(90)                                                    |
+------------------------------------------------------------+
| -0.44807361612917                                          |
+------------------------------------------------------------+
1 row in set (0.00 sec)
```

COT(X) ()

返回 X 的余切值。示例如下：

```
mysql>SELECT COT(1);
+------------------------------------------------------------+
| COT(1)                                                     |
+------------------------------------------------------------+
| 0.64209261593433                                           |
+------------------------------------------------------------+
1 row in set (0.00 sec)
```

DEGREES(X) ()

将 X 从弧度值转换为角度值。示例如下：

```
mysql>SELECT DEGREES(PI());
+------------------------------------------------------------+
| DEGREES(PI())                                              |
+------------------------------------------------------------+
| 180.000000                                                 |
+------------------------------------------------------------+
1 row in set (0.00 sec)
```

EXP(X) ()

返回 e（自然对数的底数）的 X 次幂。示例如下：

```
mysql>SELECT EXP(3);
+------------------------------------------------------------+
| EXP(3)                                                     |
+------------------------------------------------------------+
| 20.085537                                                  |
+------------------------------------------------------------+
1 row in set (0.00 sec)
```

FLOOR(X) ()

返回不大于 X 的最大整形数值。

```
mysql>SELECT FLOOR(7.55);
+-------------------------------------------------------+
| FLOOR(7.55)                                           |
+-------------------------------------------------------+
| 7                                                     |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

FORMAT(X,D) ()

将数值 X 以下列格式进行格式化：###,###,###.##，并将小数位缩减到 D 位。示例如下：

```
mysql>SELECT FORMAT(423423234.65434453,2);
+-------------------------------------------------------+
| FORMAT(423423234.65434453,2)                          |
+-------------------------------------------------------+
| 423,423,234.65                                        |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

GREATEST(n1,n2,n3,.....) ()

返回 n1 、 n2 、 n3 等一系列参数中的最大值。示例如下：

```
mysql>SELECT GREATEST(3,5,1,8,33,99,34,55,67,43);
+-------------------------------------------------------+
| GREATEST(3,5,1,8,33,99,34,55,67,43)                   |
+-------------------------------------------------------+
| 99                                                    |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

INTERVAL(N,N1,N2,N3,.....) ()

将第一个参数 N 与后续的一列参数 N1 、N2 、N3 等一一进行比对。返回结果规则为：如果 N < N1 ，返回0；如果 N < N2 ，则返回1；如果 N < N3 则返回2……以此类推。如果 N 为NULL，则返回−1。参数列表 N1 、N2 、N3 ……必须满足 N1 < N2 < N3 ……才能正常执行本函数。示例如下：

```
mysql>SELECT INTERVAL(6,1,2,3,4,5,6,7,8,9,10);
+-----------------------------------------------------------+
| INTERVAL(6,1,2,3,4,5,6,7,8,9,10)                          |
+-----------------------------------------------------------+
| 6                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

注意，6是在后续参数列表中，第一个大于 N 的值所对应的索引（参数列表的初始索引为0）。所以，在我们这个例子中，7正是这个大于 N 的值，它的索引却是6。

LEAST(N1,N2,N3,N4,.....) ()

该函数是 GREATEST() 函数的逆向函数，返回 N1、N2、N3、N4 等值的最小值。示例如下：

```
mysql>SELECT LEAST(3,5,1,8,33,99,34,55,67,43);
+-----------------------------------------------------------+
| LEAST(3,5,1,8,33,99,34,55,67,43)                          |
+-----------------------------------------------------------+
| 1                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

LOG(X) 与 LOG(B,X) ()

单参数 X 版本的函数返回 X 的自然对数。双参数版本的函数将返回以 B 为底 X 的对数。示例如下：

```
mysql>SELECT LOG(45);
+-----------------------------------------------------------+
| LOG(45)                                                   |
+-----------------------------------------------------------+
| 3.806662                                                  |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

```
mysql>SELECT LOG(2,65536);
+-------------------------------------------------------+
| LOG(2,65536)                                          |
+-------------------------------------------------------+
| 16.000000                                             |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

LOG10(X) ()

返回以10为底的 X 的对数（即常用对数lg）。

```
mysql>SELECT LOG10(100);
+-------------------------------------------------------+
| LOG10(100)                                            |
+-------------------------------------------------------+
| 2.000000                                              |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

MOD(N,M) ()

返回 N 除以 M 的余数。示例如下：

```
mysql>SELECT MOD(29,3);
+-------------------------------------------------------+
| MOD(29,3)                                             |
+-------------------------------------------------------+
| 2                                                     |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

OCT(N) ()

以字符串形式返回八进制数 N，作用相当于 CONV(N,10,8) 。示例如下：

```
mysql>SELECT OCT(12);
+-------------------------------------------------------+
| OCT(12)                                               |
+-------------------------------------------------------+
```

```
| 14                                                          |
+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

PI() ()

返回 π 值。MySQL 所存储的 π 值为双精度浮点值。

```
mysql>SELECT PI();
+-------------------------------------------------------------+
| PI()                                                        |
+-------------------------------------------------------------+
| 3.141593                                                    |
+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

POW(X,Y) 与POWER(X,Y) ()

这两个函数都能返回 X 的 Y 次幂。示例如下：

```
mysql> SELECT POWER(3,3);
+-------------------------------------------------------------+
| POWER(3,3)                                                  |
+-------------------------------------------------------------+
| 27                                                          |
+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

RADIANS(X) ()

将角度值 X 转换成弧度值返回。示例如下：

```
mysql>SELECT RADIANS(90);
+-------------------------------------------------------------+
| RADIANS(90)                                                 |
+-------------------------------------------------------------+
|1.570796                                                     |
+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

ROUND(X) 与 ROUND(X,D) ()

将 X 四舍五入，返回最接近 X 的整数。如果传入第二个参数 D，则函数会将 X 四舍五入到 小数点后的 D 位。D 必须为正值，否则小数点后所有数值都将被清除。示例如下：

``` mysql>SELECT ROUND(5.693893); +-----------------------------------------
-------------+ | ROUND(5.693893) | +-----------------------------------------
-------------+ | 6 | +-----------------------------------------------------
--+ 1 row in set (0.00 sec)

mysql>SELECT ROUND(5.693893,2); +-----------------------------------------
-------------+ | ROUND(5.693893,2) | +-----------------------------------------
-------------+ | 5.69 | +-----------------------------------------------------
----+ 1 row in set (0.00 sec)

```

SIGN(X) ()

返回 X 的符号（表明 X 究竟是负数、0还是正数的符号）：-1、0、1。

```
mysql>SELECT SIGN(-4.65);
+-------------------------------------------------------+
| SIGN(-4.65)                                           |
+-------------------------------------------------------+
| -1                                                    |
+-------------------------------------------------------+
1 row in set (0.00 sec)

mysql>SELECT SIGN(0);
+-------------------------------------------------------+
| SIGN(0)                                               |
+-------------------------------------------------------+
| 0                                                     |
+-------------------------------------------------------+
1 row in set (0.00 sec)

mysql>SELECT SIGN(4.65);
+-------------------------------------------------------+
| SIGN(4.65)                                            |
```

```
+-----------------------------------------------------------+
| 1                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

SIN(X) ()

返回 X 的正弦值。示例如下：

```
mysql>SELECT SIN(90);
+-----------------------------------------------------------+
| SIN(90)                                                   |
+-----------------------------------------------------------+
| 0.893997                                                  |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

SQRT(X) ()

返回 X 的非负数平方根。示例如下：

```
mysql>SELECT SQRT(49);
+-----------------------------------------------------------+
| SQRT(49)                                                  |
+-----------------------------------------------------------+
| 7                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

STD(expression) 与 STDDEV(expression) ()

返回表达式 expression 的标准偏差值。等于取 VARIANCE(expression) 的平方根。以下范例计算 CARS 表中 PRICE 列的标准偏差。

```
mysql>SELECT STD(PRICE) STD_DEVIATION FROM CARS;
+-----------------------------------------------------------+
| STD_DEVIATION                                             |
+-----------------------------------------------------------+
| 7650.2146                                                 |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

TAN(X) ()

返回参数 X （以弧度表示）的正切值。示例如下：

```
mysql>SELECT TAN(45);
+-------------------------------------------------------------+
| TAN(45)                                                     |
+-------------------------------------------------------------+
| 1.619775                                                    |
+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

TRUNCATE(X,D) ()

X 值的小数位被截取到 D 位。如果 D 为0，则 X 无小数位。如果 D 为负值，则将把 X 整数部分末位 D 位数值清除为0。以上运算均为清除，非四舍五入。示例如下：

```
mysql>SELECT TRUNCATE(7.536432,2);
+-------------------------------------------------------------+
| TRUNCATE(7.536432,2)                                        |
+-------------------------------------------------------------+
| 7.53                                                        |
+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

MySQL RAND 函数

RAND 函数用于产生从0到1之间的随机数。

```
mysql>  SELECT RAND( ), RAND( ), RAND( );
+------------------+------------------+------------------+
| RAND( )          | RAND( )          | RAND( )          |
+------------------+------------------+------------------+
| 0.45464584925645 | 0.1824410643265 | 0.54826780459682 |
+------------------+------------------+------------------+
1 row in set (0.00 sec)
```

如果为RAND()传入整形参数，则该值将成为随机数发生器的种子值。每次 》》》

```
mysql>  SELECT RAND(1), RAND( ), RAND( );
+------------------+------------------+------------------+
| RAND(1 )         | RAND( )          | RAND( )          |
+------------------+------------------+------------------+
| 0.18109050223705 | 0.75023211143001 | 0.20788908117254 |
+------------------+------------------+------------------+
1 row in set (0.00 sec)
```

还可以使用 ORDER BY RAND() 来随机化一组行或值。比如对于下面这个 employee_tbl 表。

```
mysql> SELECT * FROM employee_tbl;
+------+------+------------+--------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+--------------------+
|   1 | John | 2007-01-24 |              250 |
|   2 | Ram  | 2007-05-27 |               220 |
|   3 | Jack | 2007-05-06 |              170 |
|   3 | Jack | 2007-04-06 |              100 |
|   4 | Jill | 2007-04-06 |             220 |
|   5 | Zara | 2007-06-06 |              300 |
|   5 | Zara | 2007-02-06 |              350 |
+------+------+------------+--------------------+
7 rows in set (0.00 sec)
```

现在使用如下命令：

```
mysql> SELECT * FROM employee_tbl ORDER BY RAND();
+------+------+------------+--------------------+
| id   | name | work_date  | daily_typing_pages |
```

```
+------+------+------------+--------------------+
|    5 | Zara | 2007-06-06 |                300 |
|    3 | Jack | 2007-04-06 |                100 |
|    3 | Jack | 2007-05-06 |                170 |
|    2 | Ram  | 2007-05-27 |                 220 |
|    4 | Jill | 2007-04-06 |               220 |
|    5 | Zara | 2007-02-06 |                350 |
|    1 | John | 2007-01-24 |                250 |
+------+------+------------+--------------------+
7 rows in set (0.01 sec)

mysql> SELECT * FROM employee_tbl ORDER BY RAND();
+------+------+------------+--------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+--------------------+
|    5 | Zara | 2007-02-06 |                350 |
|    2 | Ram  | 2007-05-27 |                 220 |
|    3 | Jack | 2007-04-06 |                100 |
|    1 | John | 2007-01-24 |                250 |
|    4 | Jill | 2007-04-06 |               220 |
|    3 | Jack | 2007-05-06 |                170 |
|    5 | Zara | 2007-06-06 |                300 |
+------+------+------------+--------------------+
7 rows in set (0.00 sec)
```

MySQL SQRT 函数

SQRT 函数用来计算任何数字的平方根，可以使用 SELECT 语句来计算任何数字的平方根，如下所示：

```
mysql>  select SQRT(16);
+-----------+
| SQRT(16) |
+-----------+
| 4.000000 |
+-----------+
1 row in set (0.00 sec)
```

这里出现了浮点数，因为 MySQL 会把平方根以浮点数形式给出。

也可以使用 SQRT 计算多个记录的平方根。例如，对于表 employee_tbl，所有记录如下：

```
mysql> SELECT * FROM employee_tbl;
+------+------+------------+--------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+--------------------+
|   1 | John | 2007-01-24 |              250 |
|   2 | Ram  | 2007-05-27 |              220 |
|   3 | Jack | 2007-05-06 |              170 |
|   3 | Jack | 2007-04-06 |              100 |
|   4 | Jill | 2007-04-06 |              220 |
|   5 | Zara | 2007-06-06 |              300 |
|   5 | Zara | 2007-02-06 |              350 |
+------+------+------------+--------------------+
7 rows in set (0.00 sec)
```

假设需要根据上表计算 dialy_typing_pages 字段的平方根，则可以使用如下命令：

```
mysql> SELECT name, SQRT(daily_typing_pages)
    -> FROM employee_tbl;
+------+--------------------------+
| name | SQRT(daily_typing_pages) |
+------+--------------------------+
| John |              15.811388 |
| Ram  |              14.832397 |
| Jack |              13.038405 |
| Jack |              10.000000 |
| Jill |              14.832397 |
| Zara |              17.320508 |
```

```
| Zara |              18.708287 |
+------+------------------------+
7 rows in set (0.00 sec)
```

MySQL 字符串函数

函数名称	函数功能说明
ASCII() (页 130)	返回字符串 str 中最左边字符的 ASCII 代码值
BIN() (页 131)	返回十进制数值 N 的二进制数值的字符串表现形式
BIT_LENGTH() (页 131)	返回字符串 str 所占的位长度
CHAR() (页 132)	返回每一个传入的整数所对应的字符
CHAR_LENGTH() (页 132)	单纯返回 str 的字符串长度
CHARACTER_LENGTH() (页 132)	作用等同于 CHAR_LENGTH()
CONCAT_WS() (页 133)	返回串联并以某种分隔符进行分隔的字符串
CONCAT() (页 132)	返回串联的字符串
CONV() (页 116)	转换数值的进制
ELT() (页 133)	返回一定索引处的字符串
EXPORT_SET() (页 134)	返回一个字符串，其中，对于每个设置在 bits 中的位，得到一个 on 字符串，而对于每个未设定的位，则得到一个 off 字符串。
FIELD() (页 134)	返回第一个参数在随后参数中的索引（下文中有时也称其为位置）
FIND_IN_SET() (页 134)	返回第一个参数在第二个参数中的索引
FORMAT() (页 118)	将数值参数进行一些格式化，并保留指定的小数位数
HEX() (页 135)	返回参数的16进制数的字符串形式
INSERT() (页 136)	在字符串的指定位置处，将指定数目的字符串替换为新字符串
INSTR() (页 136)	返回子字符串第一次出现的索引
LCASE() (页 136)	等同于 LOWER()
LEFT() (页 136)	按指定规则，返回字符串中最左方的一定数目的字符
LENGTH() (页 137)	返回字符串的字节长度
LOAD_FILE() (页 137)	加载指定名称的文件
LOCATE() (页 138)	返回子字符串第一次出现的位置
LOWER() (页 138)	返回小写的参数
LPAD() (页 138)	返回字符串参数，其左侧由指定字符串补齐指定数目
LTRIM() (页 139)	去除前导空格
MAKE_SET() (页 139)	返回一个由逗号分隔的字符串集，其中每个字符串都拥有bits 集中相对应的二进制位

函数名称	函数功能说明
MID() (页 139)	从指定位置返回子字符串
OCT() (页 120)	将参数转变成八进制数，返回这个八进制数的字符串形式
OCTET_LENGTH() (页 140)	等同于 LENGTH()
ORD() (页 140)	如果参数中最左方的字符是个多字节字符，则返回该字符的ASCII代码值
POSITION() (页 140)	等同于 LOCATE()
QUOTE() (页 140)	对参数进行转义，以便用于 SQL 语句
REGEXP (页 141)	使用正则表达式进行模式匹配
REPEAT() (页 142)	按指定次数重复字符串
REPLACE() (页 142)	查找更换指定的字符串
REVERSE() (页 142)	反转字符串参数中的字符
RIGHT() (页 143)	返回字符串参数最右边指定位数的字符
RPAD() (页 143)	将字符串按指定次数重复累加起来
RTRIM() (页 143)	除去字符串参数的拖尾空格
SOUNDEX() (页 144)	返回一个soundex字符串
SOUNDS LIKE (页 144)	对比声音
SPACE() (页 144)	返回指定空格数目的字符串
STRCMP() (页 144)	对比两个字符串
SUBSTRING_INDEX() (页 146)	将字符串参数中在指定序号的分隔符之前的子字符串予以返回
SUBSTRING() 与 SUBSTR() (页 145)	按指定规则返回子字符串
TRIM() (页 146)	清除字符串参数的前导及拖尾空格
UCASE() (页 147)	等同于 UPPER()
UNHEX() (页 147)	将16进制数的每一位都转变为ASCII字符
UPPER() (页 148)	将参数全转变为大写

ASCII(str) ()

返回字符串 str 中最左边字符的 ASCII 代码值。如果该字符串为空字符串，则返回0。如果字符串为 NULL 则返回 NULL。因为ASCII码表能表示的字符为256个，所以 ASCII() 返回值在0-255之间。 示例如下：

```
mysql> SELECT ASCII('2');
+-----------------------------------------------------------+
| ASCII('2')                                                |
```

```
+-------------------------------------------------------+
| 50                                                    |
+-------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ASCII('dx');
+-------------------------------------------------------+
| ASCII('dx')                                           |
+-------------------------------------------------------+
| 100                                                   |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

BIN(N) ()

返回十进制数值 `N` 的二进制数值的字符串表现形式。其中，`N` 是一 BIGINT 型数值。该函数等同于 `CONV(N, 10, 2)` 。如果 `N` 为 NULL，则返回 NULL。示例如下：

```
mysql> SELECT BIN(12);
+-------------------------------------------------------+
| BIN(12)                                               |
+-------------------------------------------------------+
| 1100                                                  |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

BIT_LENGTH(str) ()

返回字符串 `str` 所占的位长度。示例如下：

```
mysql> SELECT BIT_LENGTH('text');
+-------------------------------------------------------+
| BIT_LENGTH('text')                                    |
+-------------------------------------------------------+
| 32                                                    |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

CHAR(N,... [USING charset_name]) ()

会将每一个参数 N 都解释为整数，返回由这些整数在 ASCII 码中所对应字符所组成的字符串。忽略 NULL 值。示例如下：

```
mysql> SELECT CHAR(77,121,83,81,'76');
+-----------------------------------------------------------+
| CHAR(77,121,83,81,'76')                                   |
+-----------------------------------------------------------+
| MySQL                                                     |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

CHAR_LENGTH(str) ()

单纯返回 str 的字符串长度（字符串中到底有几个字符）。多字节字符会被当成单字符对待，所以如果一个字符串包含5个双字节字符， LENGTH() 返回10，而 CHAR_LENGTH() 会返回5。示例如下：

```
mysql> SELECT CHAR_LENGTH("text");
+-----------------------------------------------------------+
| CHAR_LENGTH("text")                                       |
+-----------------------------------------------------------+
| 4                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

CHARACTER_LENGTH(str) ()

与函数 CHAR_LENGTH() 作用相同。

CONCAT(str1,str2,...) ()

将一众字符串参数加以连接，返回结果字符串。可能有1或多个参数。如果参数中都是非二进制字符串，结果也是非二进制字符串。如果参数包含任何二进制字符串，结果也是二进制字符串。数值型参数会被转化成相应的二进制字符串形式。如果想避免这样，可以使用显式的类型转换，如下例所示：

```
mysql> SELECT CONCAT('My', 'S', 'QL');
+-----------------------------------------------------------+
```

```
| CONCAT('My', 'S', 'QL')                                  |
+----------------------------------------------------------+
| MySQL                                                    |
+----------------------------------------------------------+
1 row in set (0.00 sec)
```

CONCAT_WS(separator,str1,str2,...) ()

一种特殊的 CONCAT 函数。利用分隔符 separator 参数来连接后续的参数 str1 、 str2 ……分隔符添加在后续参数之间，与后续参数一样，它也可以是一个字符串。如果该分隔符参数为 NULL，则结果也是 NULL。示例如下：

```
mysql> SELECT CONCAT_WS(',','First name','Last Name' );
+-----------------------------------------------------------+
| CONCAT_WS(',','First name','Last Name' )                  |
+-----------------------------------------------------------+
| First name, Last Name                                     |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

CONV(N,from_base,to_base) ()

将数值在不同进制间转换。将数值型参数 N 由初始进制 from_base 转换为目标进制 to_base 的形式并返回。如果任何参数为 NULL，则返回 NULL。 N 可以是整数，但也可以是字符串。进制取值范围为2-36。如果 to_base 为负值， N 被认为是有符号数值；反之， N 被认为是无符号数值。函数运算精度为64位。示例如下：

```
mysql> SELECT CONV('a',16,2);
+-----------------------------------------------------------+
| CONV('a',16,2)                                            |
+-----------------------------------------------------------+
| 1010                                                      |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

ELT(N,str1,str2,str3,...) ()

如果 N = 1，则返回 str1 ，如果 N = 2 则返回 str2 ，以此类推。如果 N 小于1或大于参数个数，则返回 NULL。 ELT() 是 FIELD() 的功能补充函数。示例如下：

```
mysql> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
+-----------------------------------------------------------+
| ELT(1, 'ej', 'Heja', 'hej', 'foo')                        |
+-----------------------------------------------------------+
| ej                                                        |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

EXPORT_SET(bits,on,off[,separator[,number_of_bits]]) ()

对于 bits 中的每一位，都能得到一个 on 字符串，对于未在 bits 中的每个比特，则得到 off 字符串。bits 中的比特从右向左（从低位到高位比特）排列，而字符串则是按照从左至右的顺序添加到结果上，并由 separator 字符串分隔（默认采用逗号 , ）。bits 中的位数由 number_of_bits 提供，如果不指定，则默认为64。如果大于64，则会自动截取为64，它是一个无符号整形值，因此上-1也和64具有一样的效果。

```
mysql> SELECT EXPORT_SET(5,'Y','N',',',4);
+-----------------------------------------------------------+
| EXPORT_SET(5,'Y','N',',',4)                               |
+-----------------------------------------------------------+
| Y,N,Y,N                                                   |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

FIELD(str,str1,str2,str3,...) ()

返回 str 在后面的参数列（ str1 、 str2 、 str3 …… ）中的索引（起始索引为1）。如果未在参数列中发现 str 则返回0。

```
mysql> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
+-----------------------------------------------------------+
| FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo')            |
+-----------------------------------------------------------+
| 2                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

FIND_IN_SET(str,strlist) ()

如果字符串 str 在由 N 个子字符串组成的字符串列表 strlist 中，则返回其在 strlist 中的索引（字符串列表 strlist 的初始索引为1）。示例如下：

```
mysql> SELECT FIND_IN_SET('b','a,b,c,d');
+-----------------------------------------------------------+
| SELECT FIND_IN_SET('b','a,b,c,d')                         |
+-----------------------------------------------------------+
| 2                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

FORMAT(X,D) ()

将数值参数 X 以'#,###,###.##' 的形式进行格式化，并四舍五入到小数点后 D 位处，然后将格式化结果以字符串形式返回。如果 D 为0，则结果没有小数部分。示例如下：

```
mysql> SELECT FORMAT(12332.123456, 4);
+-----------------------------------------------------------+
| FORMAT(12332.123456, 4)                                   |
+-----------------------------------------------------------+
| 12,332.1235                                               |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

HEX(N_or_S) ()

当 N_or_S 为数值时，以16进制数的字符串形式返回 N 的值， N 为 BIGINT 型值。该函数作用等同于 CONV(N, 10, 16) 。

当 N_or_S 为字符串时，返回 N_or_S 的16进制字符串形式， N_or_S 中的每个字符都被转换为2个16进制数字。示例如下：

```
mysql> SELECT HEX(255);
+-----------------------------------------------------------+
| HEX(255)                                                  |
+-----------------------------------------------------------+
| FF                                                        |
+-----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT 0x616263;
+-----------------------------------------------------------+
| 0x616263                                                  |
+-----------------------------------------------------------+
| abc                                                       |
```

```
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

INSERT(str,pos,len,newstr) ()

在原始字符串 str 中，将自左数第 pos 位开始，长度为 len 个字符的字符串替换为新字符串 newstr，然后返回经过替换后的字符串。如果 pos 未在原始字符串长度内，则返回原始字符串。如果 len 不在原始字符串长度内，则返回原始字符串中自 pos 位起后面剩余的字符串。如果函数中任一参数为 NULL，则返回 NULL。示例如下：

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
+-----------------------------------------------------+
| INSERT('Quadratic', 3, 4, 'What')                   |
+-----------------------------------------------------+
| QuWhattic                                           |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

INSTR(str,substr) ()

返回 substr 在 str 中第一次出现时的位置（也就是索引）。作用类似于双参数版本的 LOCATE() 函数，只不过参数的次序调换了过来。示例如下：

```
mysql> SELECT INSTR('foobarbar', 'bar');
+-----------------------------------------------------+
| INSTR('foobarbar', 'bar')                           |
+-----------------------------------------------------+
| 4                                                   |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

LCASE(str) ()

等同于 LOWER() 。

LEFT(str,len) ()

返回字符串 str 自左数的 len 个字符。如果任一参数为 NULL，则返回 NULL。示例如下：

```
mysql> SELECT LEFT('foobarbar', 5);
+----------------------------------------------------------+
| LEFT('foobarbar', 5)                                     |
+----------------------------------------------------------+
| fooba                                                    |
+----------------------------------------------------------+
1 row in set (0.00 sec)
```

LENGTH(str) ()

返回字符串 `str` 的字节长度。多字节字符被如实计算为多字节。所以，对于包含5个双字节字符（如中文字符）的字符串，`LENGTH()` 返回10，而 `CHAR_LENGTH()` 返回5。示例如下：

```
mysql> SELECT LENGTH('text');
+----------------------------------------------------------+
| LENGTH('text')                                           |
+----------------------------------------------------------+
| 4                                                        |
+----------------------------------------------------------+
1 row in set (0.00 sec)
```

LOAD_FILE(file_name) ()

读取文件并以字符串形式返回文件内容。使用该函数时，文件必须位于服务器主机中，且必须制定该文件的完整路径名。必须拥有 FILE 权限。文件对于所有人都必须是可读状态，而且文件尺寸也必须小于 `max_allowed_packet` 字节。

如果因为未满足上述几个条件，从而文件不存在于服务器主机中，或者不可读，则函数返回 NULL。

自 MySQL 5.0.19 起，`character_set_filesystem` 系统变量负责对字符串形式文件名加以解读。 示例如下：

```
mysql> UPDATE table_test
    -> SET blob_col=LOAD_FILE('/tmp/picture')
    -> WHERE id=1;
.....
```

LOCATE(substr,str), LOCATE(substr,str,pos) ()

第一种格式函数的作用如下：返回 substr 在 str 中第一次出现的位置（即索引）。第二种格式函数则返回自 str 指定位置 pos （即索引）起， substr 在 str 中第一次出现的位置。如果在 str 中未找到 substr，则两种函数都返回0。示例如下：

```
mysql> SELECT LOCATE('bar', 'foobarbar');
+----------------------------------------------------------+
| LOCATE('bar', 'foobarbar')                               |
+----------------------------------------------------------+
| 4                                                        |
+----------------------------------------------------------+
1 row in set (0.00 sec)
```

LOWER(str) ()

根据当前所采用的字符集映射关系，将 str 所有字符都转为小写，并返回新字符串。示例如下：

```
mysql> SELECT LOWER('QUADRATICALLY');
+----------------------------------------------------------+
| LOWER('QUADRATICALLY')                                   |
+----------------------------------------------------------+
| quadratically                                            |
+----------------------------------------------------------+
1 row in set (0.00 sec)
```

LPAD(str,len,padstr) ()

左补齐函数。将字符串 str 左侧利用字符串 padstr 补齐为整体长度为 len 的字符串。如果 str 大于 len，则返回值会缩减到 len 个字符。示例如下：

```
mysql> SELECT LPAD('hi',4,'??');
+----------------------------------------------------------+
| LPAD('hi',4,'??')                                        |
+----------------------------------------------------------+
| ??hi                                                     |
+----------------------------------------------------------+
1 row in set (0.00 sec)
```

LTRIM(str) ()

将字符串 str 中前部的空格字符去除，然后返回新的 str 字符串。示例如下：

```
mysql> SELECT LTRIM('  barbar');
+-------------------------------------------------------------+
| LTRIM('  barbar')                                           |
+-------------------------------------------------------------+
| barbar                                                      |
+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

MAKE_SET(bits,str1,str2,...) ()

返回一个集合值（是一个由字符 , 所分隔的众多子字符串所组合而成的字符串），该集合中包含的字符串的比特数等于 bits 集合中的对应比特数。例如， str1 对应着 bit 0 ， str2 对应 bit 1 ，以此类推。 str1 、 str2 ……中的 NULL 值将不会添加到结果中。示例如下：

```
mysql> SELECT MAKE_SET(1,'a','b','c');
+-------------------------------------------------------------+
| MAKE_SET(1,'a','b','c')                                     |
+-------------------------------------------------------------+
| a                                                           |
+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

MID(str,pos,len) ()

MID(str,pos,len) 作用等同于 SUBSTRING(str,pos,len) 。

OCT(N) ()

以字符串形式返回 N 的八进制数， N 是一个BIGINT 型数值。作用相当于 CONV(N,10,8) 。如果 N 为 NULL，则返回 NULL。示例如下：

```
mysql> SELECT OCT(12);
+-------------------------------------------------------------+
| OCT(12)                                                     |
```

```
+-----------------------------------------------------------+
| 14                                                        |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

OCTET_LENGTH(str) ()

OCTET_LENGTH() 作用等同于 LENGTH() 。

ORD(str) ()

如果 str 最左边的字符是一个多字节字符，利用以下公式计算返回该字符的 ASCII 代码值。

> （第一个字节的 ASCII 代码）
>
> +（第1个字节的 ASCII 代码 × 256）
>
> +（第3个字节的 ASCII 代码 × 256^2）……

如果最左边的字符不是多字节字符，则函数按照 ASCII() 方式返回值。示例如下：

```
mysql> SELECT ORD('2');
+-----------------------------------------------------------+
| ORD('2')                                                  |
+-----------------------------------------------------------+
| 50                                                        |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

POSITION(substr IN str) ()

作用等同于 LOCATE(substr,str) 。

QUOTE(str) ()

产生一个在SQL语句中可用作正确转义数据值的结果。将 str 中的每一个单引号（ ' ）、反转杠（ \ ）、 ASCII的NUL值，以及 Control+Z 组合前加上反斜杠，最后再补足左右闭合用的单引号。如果参数为 NULL，则返回 NULL 的字符串形式（不用单引号闭合），示例如下：

```
mysql> SELECT QUOTE('Don\'t!');
+-------------------------------------------------------+
| QUOTE('Don\'t!')                                      |
+-------------------------------------------------------+
| 'Don\'t!'                                             |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

> 注意：检查一下，如果安装的 MySQL 不支持这个函数，则不要使用。

expr REGEXP pattern ()

将 expr 与 pattern 进行模式匹配。如果相匹配则返回1，否则返回0。如果 expr 与 pattern 两个参数中有一个为 NULL，则结果也为 NULL。除了在处理二进制字符串时，REGEXP 对大小写并不敏感。第1个示例如下：

```
mysql> SELECT 'ABCDEF' REGEXP 'A%C%%';
+-------------------------------------------------------+
| 'ABCDEF' REGEXP 'A%C%%'                               |
+-------------------------------------------------------+
| 0                                                     |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

第2个实例：

```
mysql> SELECT 'ABCDE' REGEXP '.*';
+-------------------------------------------------------+
|  'ABCDE' REGEXP '.*'                                  |
+-------------------------------------------------------+
| 1                                                     |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

第3个示例：

```
mysql> SELECT 'new*\n*line' REGEXP 'new\\*.\\*line';
+-------------------------------------------------------+
| 'new*\n*line' REGEXP 'new\\*.\\*line'                 |
+-------------------------------------------------------+
| 1                                                     |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

REPEAT(str,count) ()

将字符串 str 重复 count 次，返回结果字符串。如果 count 小于1，则返回一个空字符串。如果 str 或 count 为 NULL，则返回 NULL。示例如下：

```
mysql> SELECT REPEAT('MySQL', 3);
+-----------------------------------------------------------+
| REPEAT('MySQL', 3)                                        |
+-----------------------------------------------------------+
| MySQLMySQLMySQL                                           |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

REPLACE(str,from_str,to_str) ()

查找字符串 str 中出现的 from_str ，将其都更换为 to_str 。在查找 from_str 时，函数对大小写是敏感的。示例如下：

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
+-----------------------------------------------------------+
| REPLACE('www.mysql.com', 'w', 'Ww')                       |
+-----------------------------------------------------------+
| WwWwWw.mysql.com                                          |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

REVERSE(str) ()

以反向顺序返回 str 所有字符。示例如下：

```
mysql> SELECT REVERSE('abcd');
+-----------------------------------------------------------+
| REVERSE('abcd')                                           |
+-----------------------------------------------------------+
| dcba                                                      |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

RIGHT(str,len) ()

返回 str 右边末 len 位的字符。如果有的参数是 NULL 值，则返回 NULL。

```
mysql> SELECT RIGHT('foobarbar', 4);
+----------------------------------------------------------+
| RIGHT('foobarbar', 4)                                    |
+----------------------------------------------------------+
| rbar                                                     |
+----------------------------------------------------------+
1 row in set (0.00 sec)
```

RPAD(str,len,padstr) ()

在 str 右方补齐 len 位的字符串 padstr，返回新字符串。如果 str 长度大于 len，则返回值的长度将缩减到 len 所指定的长度。示例如下：

```
mysql> SELECT RPAD('hi',5,'?');
+----------------------------------------------------------+
| RPAD('hi',5,'?')                                         |
+----------------------------------------------------------+
| hi???                                                    |
+----------------------------------------------------------+
1 row in set (0.00 sec)
```

RTRIM(str) ()

去除 str 的拖尾空格，返回新字符串。示例如下：

```
mysql> SELECT RTRIM('barbar   ');
+----------------------------------------------------------+
| RTRIM('barbar   ')                                       |
+----------------------------------------------------------+
| barbar                                                   |
+----------------------------------------------------------+
1 row in set (0.00 sec)
```

SOUNDEX(str) ()

一种能够生成判断字符串是否同音的 soundex 字符串的函数。返回 str 的 soundex 字符串。听起来相似的两个字符串应该具有相同的soundex 字符串。标准的soundex 字符串包含4个字符，但 MySQL 的 SOUNDEX() 函数返回的是任意长度的字符串。可以在结果上使用 SUBSTRING() 来获取标准的 soundex 字符串。str 中的非字母字符自动会被忽略。所有在 A-Z 范围之外的国际字母字符会被认为是元音字母。示例如下：

```
mysql> SELECT SOUNDEX('Hello');
+-------------------------------------------------------+
| SOUNDEX('Hello')                                      |
+-------------------------------------------------------+
| H400                                                  |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

expr1 SOUNDS LIKE expr2 ()

作用等同于 SOUNDEX(expr1) = SOUNDEX(expr2) 。

SPACE(N) ()

返回包含 N 个空格的字符串。示例如下：

```
mysql> SELECT SPACE(6);
+-------------------------------------------------------+
| SELECT SPACE(6)                                       |
+-------------------------------------------------------+
|      '                                                |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

STRCMP(str1, str2) ()

对比两个字符串 str1 和 str2 ，如果两字符串相等，返回1；如果当前的排序规则， str1 小于 str2 ，则返回-1，反之则都返回1。第1个示例如下：

```
mysql> SELECT STRCMP('MOHD', 'MOHD');
+-------------------------------------------------------+
```

```
| STRCMP('MOHD', 'MOHD')                                    |
+-----------------------------------------------------------+
| 0                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

第2个示例如下：

```
mysql> SELECT STRCMP('AMOHD', 'MOHD');
+-----------------------------------------------------------+
| STRCMP('AMOHD', 'MOHD')                                   |
+-----------------------------------------------------------+
| -1                                                        |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

第3个示例如下：

```
mysql> SELECT STRCMP('MOHD', 'AMOHD');
+-----------------------------------------------------------+
| STRCMP('MOHD', 'AMOHD')                                   |
+-----------------------------------------------------------+
| 1                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

SUBSTRING(str,pos)、SUBSTRING(str FROM pos)、SUBSTRING(str,pos,len)、SUBSTRING(str FROM pos FOR len) ()

在以上4种函数变种形式中，没有 len 参数的函数形式会返回自 str 中位置 pos 处之后的子字符串；有 len 参数的函数形式会返回自 str 中位置 pos 处之后，长度为 len 的子字符串。使用 FROM 的函数形式则是采用的标准的 SQL 语法。pos 参数也可能取负值，在这种情况下，取字符串的方式是从字符串 str 的末尾向前（而非从前往后），从这种逆向顺序的 pos 处开始取字符串。另外，负值的 pos 参数可用于任何形式的 SUBSTRING() 函数中。示例如下：

```
mysql> SELECT SUBSTRING('Quadratically',5);
+-----------------------------------------------------------+
| SSUBSTRING('Quadratically',5)                             |
+-----------------------------------------------------------+
| ratically                                                 |
+-----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT SUBSTRING('foobarbar' FROM 4);
```

```
+-----------------------------------------------------+
| SUBSTRING('foobarbar' FROM 4)                       |
+-----------------------------------------------------+
| barbar                                              |
+-----------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT SUBSTRING('Quadratically',5,6);
+-----------------------------------------------------+
| SUBSTRING('Quadratically',5,6)                      |
+-----------------------------------------------------+
| ratica                                              |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

SUBSTRING_INDEX(str,delim,count) ()

返回 str 中第 count 次出现的分隔符 delim 之前的子字符串。如果 count 为正数，将最后一个分隔符左边（因为是从左数分隔符）的所有内容作为子字符串返回；如果 count 为负值，返回最后一个分隔符右边（因为是从右数分隔符）的所有内容作为子字符串返回。在寻找分隔符时，函数对大小写是敏感的。示例如下：

```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
+-----------------------------------------------------+
| SUBSTRING_INDEX('www.mysql.com', '.', 2)            |
+-----------------------------------------------------+
| www.mysql                                           |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str) 与 TRIM([remstr FROM] str) ()

将字符串 str 去除 remstr 所指定的前缀或后缀，返回结果字符串。如果没有指定标识符 BOTH 、LEADING ，或 TRAILING ，则默认采用 BOTH ，即将前后缀都删除。remstr 其实是个可选参数，如果没有指定它，则删除的是空格。示例如下：

```
mysql> SELECT TRIM('  bar   ');
+-----------------------------------------------------+
| TRIM('  bar   ')                                    |
+-----------------------------------------------------+
| bar                                                 |
+-----------------------------------------------------+
```

```
1 row in set (0.00 sec)

mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
+-----------------------------------------------------------+
| TRIM(LEADING 'x' FROM 'xxxbarxxx')                        |
+-----------------------------------------------------------+
| barxxx                                                    |
+-----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
+-----------------------------------------------------------+
| TRIM(BOTH 'x' FROM 'xxxbarxxx')                           |
+-----------------------------------------------------------+
| bar                                                       |
+-----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxxyz');
+-----------------------------------------------------------+
| TRIM(TRAILING 'xyz' FROM 'barxxyz')                       |
+-----------------------------------------------------------+
| barx                                                      |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

UCASE(str) ()

作用等同于 `UPPER()` 。

UNHEX(str) ()

它是 `HEX(str)` 的逆向函数。将参数中的每一对16进制数字都转换为10进制数字，然后再转换成 ASCII 码所对应的字符。结果返回的字符是二进制字符串。

```
mysql> SELECT UNHEX('4D7953514C');
+-----------------------------------------------------------+
| UNHEX('4D7953514C')                                       |
+-----------------------------------------------------------+
| MySQL                                                     |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

参数 X 中的字符必须是合法的16进制数字：0-9，A-F或者a-f（因为16进制不区分字母的大小写）。如果参数 X 中包含非16进制数字，则函数返回 NULL。

UPPER(str) ()

根据当前所采用的字符集映射关系，将 str 所有字符都转为大写，并返回新字符串。示例如下：

```
mysql> SELECT UPPER('Allah-hus-samad'); +-----------------------------------------
-------------+ | UPPER('Allah-hus-samad') | +-----------------------------------------
----------------+ | ALLAH-HUS-SAMAD | +-----------------------------------------
--------------+ 1 row in set (0.00 sec)
```

至此，我们概述了关于 MySQL 字符串函数的一般内容，要想深入了解相关内容，请参看 MySQL 官方网站上的字符串函数部分 (http://dev.mysql.com/doc/refman/5.6/en/string-functions.html) 。

MySQL SUM 函数

SUM 函数用来在不同记录中计算某一字段的总和值。

例如，在表 employee_tbl 中，所有记录如下：

```
mysql> SELECT * FROM employee_tbl;
+------+------+------------+--------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+--------------------+
|   1 | John | 2007-01-24 |              250 |
|   2 | Ram  | 2007-05-27 |              220 |
|   3 | Jack | 2007-05-06 |              170 |
|   3 | Jack | 2007-04-06 |              100 |
|   4 | Jill | 2007-04-06 |              220 |
|   5 | Zara | 2007-06-06 |              300 |
|   5 | Zara | 2007-02-06 |              350 |
+------+------+------------+--------------------+
7 rows in set (0.00 sec)
```

如果想根据该表计算字段 dialy_typing_pages 的总值，可以使用如下命令：

```
mysql> SELECT SUM(daily_typing_pages)
    -> FROM employee_tbl;
+-------------------------+
| SUM(daily_typing_pages) |
+-------------------------+
|                    1610 |
+-------------------------+
1 row in set (0.00 sec)
```

还可以使用 GROUP BY 子句来计算多种记录集的平均值。下面这个范例将计算每个人的所有记录的总值，将得到每个人的平均输入页面。

```
mysql> SELECT name, SUM(daily_typing_pages)
    -> FROM employee_tbl GROUP BY name;
+------+-------------------------+
| name | SUM(daily_typing_pages) |
+------+-------------------------+
| Jack |                     270 |
| Jill |                     220 |
| John |                     250 |
| Ram  |                     220 |
```

```
| Zara |                650 |
+------+--------------------------+
5 rows in set (0.17 sec)
```

MySQL 日期与时间方面的函数

函数名称	函数功能说明
ADDDATE() (#ADDDATE)	添加日期
ADDTIME() (#ADDTIME)	添加时间
CONVERT_TZ() (#CONVERT_TZ)	转换不同时区
CURDATE() (#CURDATE)	返回当前日期
CURRENT_DATE() 与 CURRENT_DATE (#CURRENT_DATE)	等同于 CURDATE()
CURRENT_TIME() 与 CURRENT_TIME (#CURRENT_TIME2)	等同于 CURTIME()
CURRENT_TIMESTAMP() 与 CURRENT_TIMESTAMP (#CURRENT_TIMESTAMP)	等同于 NOW()
CURTIME() (#CURTIME)	返回当前时间
DATE_ADD() (#DATE_ADD)	添加两个日期
DATE_FORMAT() (#DATE_FORMAT)	按指定方式格式化日期
DATE_SUB() (#DATE_SUB)	求解两个日期的间隔
DATE() (#DATE)	提取日期或日期时间表达式中的日期部分
DATEDIFF() (#DATEDIFF)	求解两个日期的间隔
DAY() (#DAY)	等同于 DAYOFMONTH()
DAYNAME() (#DAYNAME)	返回星期中某天的名称
DAYOFMONTH() (#DAYOFMONTH)	返回一月中某天的序号（1-31）
DAYOFWEEK() (#DAYOFWEEK)	返回参数所定影的一周中某天的索引值
DAYOFYEAR() (#DAYOFYEAR)	返回一年中某天的序号（1-366）
EXTRACT (#EXTRACT)	提取日期中的相应部分
FROM_DAYS() (#FROM_DAYS)	将一个天数序号转变为日期值
FROM_UNIXTIME() (#FROM_UNIXTIME)	将日期格式化为 UNIX 的时间戳
HOUR() (#HOUR)	提取时间
LAST_DAY (#LAST_DAY)	根据参数，返回月中最后一天
LOCALTIME() 和 LOCALTIME (#LOCALTIME)	等同于 NOW()
LOCALTIMESTAMP 和 LOCALTIMESTAMP() (#LOCALTIMESTAMP)	等同于 NOW()
MAKEDATE() (#MAKEDATE)	基于给定参数年份和所在年中的天数序号，返回一个日期
MAKETIME (#MAKETIME)	MAKETIME()

函数名称	函数功能说明
MICROSECOND() (#MICROSECOND)	返回参数所对应的毫秒数
MINUTE() (#MINUTE)	返回参数对应的分钟数
MONTH() (#MONTH)	返回传入日期所对应的月序数
MONTHNAME() (#MONTHNAME)	返回月的名称
NOW() (#NOW)	返回当前日期与时间
PERIOD_ADD() (#PERIOD_ADD)	为年-月组合日期添加一个时段
PERIOD_DIFF() (#PERIOD_DIFF)	返回两个时段之间的月份差值
QUARTER() (#QUARTER)	返回日期参数所对应的季度序号
SEC_TO_TIME() (#SEC_TO_TIME)	将描述转变成 'HH:MM:SS' 的格式
SECOND() (#SECOND)	返回秒序号（0-59）
STR_TO_DATE() (#STR_TO_DATE)	将字符串转变为日期
SUBDATE() (#SUBDATE)	三个参数的版本相当于 DATE_SUB()
SUBTIME() (#SUBTIME)	计算时间差值
SYSDATE() (#SYSDATE)	返回函数执行时的时间
TIME_FORMAT() (#TIME_FORMAT)	提取参数中的时间部分
TIME_TO_SEC() (#TIME_TO_SEC)	将参数转化为秒数
TIME() (#TIME)	提取传入表达式的时间部分
TIMEDIFF() (#TIMEDIFF)	计算时间差值
TIMESTAMP() (#TIMESTAMP)	单个参数时，函数返回日期或日期时间表达式；有2个参数时，将参数加和
TIMESTAMPADD() (#TIMESTAMPADD)	为日期时间表达式添加一个间隔 INTERVAL
TIMESTAMPDIFF() (#TIMESTAMPDIFF)	从日期时间表达式中减去一个间隔 INTERVAL
TO_DAYS() (#TO_DAYS)	返回转换成天数的日期参数
UNIX_TIMESTAMP() (#UNIX_TIMESTAMP)	返回一个 UNIX 时间戳
UTC_DATE() (#UTC_DATE)	返回当前的 UTC 日期
UTC_TIME() (#UTC_TIME)	返回当前的 UTC 时间
UTC_TIMESTAMP() (#UTC_TIMESTAMP)	返回当前的 UTC 时间与日期
WEEK() (#WEEK)	返回周序号
WEEKDAY() (#WEEKDAY)	返回某天在星期中的索引值
WEEKOFYEAR() (#WEEKOFYEAR)	返回日期所对应的星期在一年当中的序号（1-53）
YEAR() (#YEAR)	返回年份
YEARWEEK() (#YEARWEEK)	返回年份及星期序号

ADDDATE(date,INTERVAL expr unit), ADDDATE(expr,days) ()

在第2个参数使用 INTERVAL 格式时， ADDDATE() 作用就相当于 DATE_ADD() 。相关的函数 SUBDATE() 相当于 DATE_SUB() 。要想了解 INTERVAL 单位参数，参看 DATE_ADD() 相关内容。示例如下：

```
mysql> SELECT DATE_ADD('1998-01-02', INTERVAL 31 DAY);
+------------------------------------------------------+
| DATE_ADD('1998-01-02', INTERVAL 31 DAY)              |
+------------------------------------------------------+
| 1998-02-02                                           |
+------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ADDDATE('1998-01-02', INTERVAL 31 DAY);
+------------------------------------------------------+
| ADDDATE('1998-01-02', INTERVAL 31 DAY)               |
+------------------------------------------------------+
| 1998-02-02                                           |
+------------------------------------------------------+
1 row in set (0.00 sec)
```

当函数的第2个参数采用 days 格式时，MySQL 会认为它是一个表示天数的整数，将它添加到 expr 上。示例如下：

```
mysql> SELECT ADDDATE('1998-01-02', 31);
+------------------------------------------------------+
| DATE_ADD('1998-01-02', INTERVAL 31 DAY)              |
+------------------------------------------------------+
| 1998-02-02                                           |
+------------------------------------------------------+
1 row in set (0.00 sec)
```

ADDTIME(expr1, expr2) ()

ADDTIME() 将 expr2 参数加到 expr1 参数上，返回结果。expr1 是一个时间或日期时间表达式。expr2 是一个时间表达式。

```
mysql> SELECT ADDTIME('1997-12-31 23:59:59.999999','1 1:1:1.000002');
+------------------------------------------------------+
| DATE_ADD('1997-12-31 23:59:59.999999','1 1:1:1.000002') |
+------------------------------------------------------+
```

```
| 1998-01-02 01:01:01.000001                          |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

CONVERT_TZ(dt,from_tz,to_tz) ()

这是一个转换时区的函数，将参数 from_tz 所定时区的日期时间值 dt 转变到参数 to_tz 所定时区，然后返回结果。如果参数无效，则该函数返回 NULL 值。

```
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','GMT','MET');
+-----------------------------------------------------+
| CONVERT_TZ('2004-01-01 12:00:00','GMT','MET')       |
+-----------------------------------------------------+
| 2004-01-01 13:00:00                                 |
+-----------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','+00:00','+10:00');
+-----------------------------------------------------+
| CONVERT_TZ('2004-01-01 12:00:00','+00:00','+10:00') |
+-----------------------------------------------------+
| 2004-01-01 22:00:00                                 |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

CURDATE() ()

返回当前日期的函数。根据函数究竟用于字符串还是数字上下文，选择使用 'YYYY-MM-DD'（'年-月-日'）或 YYYYMMDD（年月日）格式返回当前日期。

```
mysql> SELECT CURDATE();
+-----------------------------------------------------+
| CURDATE()                                           |
+-----------------------------------------------------+
| 1997-12-15                                          |
+-----------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CURDATE() + 0;
+-----------------------------------------------------+
| CURDATE() + 0                                       |
+-----------------------------------------------------+
```

```
| 19971215                                |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

CURRENT_DATE 和 CURRENT_DATE() ()

CURRENT_DATE 和 CURRENT_DATE() 实际上等于 CURDATE() 。

CURTIME() ()

根据函数究竟用于字符串或数字上下文，选择以 'HH:MM:SS' 还是 HHMMSS 格式返回当前时间值（以当前时区来定）。

```
mysql> SELECT CURTIME();
+-----------------------------------------------------------+
| CURTIME()                                 |
+-----------------------------------------------------------+
| 23:50:26                                |
+-----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CURTIME() + 0;
+-----------------------------------------------------------+
| CURTIME() + 0                                 |
+-----------------------------------------------------------+
| 235026                                |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

CURRENT_TIME 和 CURRENT_TIME() ()

CURRENT_TIME 和 CURRENT_TIME() 都相当于 CURTIME() 。

CURRENT_TIMESTAMP 和 CURRENT_TIMESTAMP() ()

CURRENT_TIMESTAMP 和 CURRENT_TIMESTAMP() 实际上相当于 NOW() 。

DATE(expr) ()

提取日期或日期时间表达式 `expr` 中的日期部分。

```
mysql> SELECT DATE('2003-12-31 01:02:03');
+-----------------------------------------------------------+
| DATE('2003-12-31 01:02:03')                               |
+-----------------------------------------------------------+
| 2003-12-31                                                |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

DATEDIFF(expr1,expr2) ()

`DATEDIFF()` 将返回 `expr1 - expr2` 的值，用来表示两个日期相差的天数。`expr1` 和 `expr2` 都是日期或日期时间表达式。运算中只用到了这些值的日期部分。

```
mysql> SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');
+-----------------------------------------------------------+
| DATEDIFF('1997-12-31 23:59:59','1997-12-30')              |
+-----------------------------------------------------------+
| 1                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

DATE_ADD(date,INTERVAL expr unit) 与 DATE_SUB(date,INTERVAL expr unit) ()

执行日期计算的两种函数。`date` 是一个用来指定**开始日期**的 DATETIME 或 DATE 值。`expr` 是一种以字符串形式呈现的表达式，用来指定从开始日期增加或减少的间隔值。如果是负的间隔值，则 `expr` 值的第一个字符是 `-` 号。`unit` 是一个单位关键字，用来指定expr表达式应该采取的单位。

INTERVAL 关键字与单位说明符都不区分大小写。

下表列出了每个单位数值所对应的 expr 参数的期望格式。

单位所能取的值	期望的expr格式
MICROSECOND	毫秒
SECOND	秒

单位所能取的值	期望的expr格式
MINUTE	分
HOUR	小时
DAY	日
WEEK	周
MONTH	月
QUARTER	季度
YEAR	年
SECOND_MICROSECOND	'秒.毫秒'
MINUTE_MICROSECOND	'分.毫秒'
MINUTE_SECOND	'分:秒'
HOUR_MICROSECOND	'小时.毫秒'
HOUR_SECOND	'小时:分:秒'
HOUR_MINUTE	'小时:分'
DAY_MICROSECOND	'日.毫秒'
DAY_SECOND	'日 小时:分:秒'
DAY_MINUTE	'日 小时:分'
DAY_HOUR	'日 小时'
YEAR_MONTH	'年-月'

QUARTER 和 WEEK 都是 MySQL 5.0.0 才开始引入的单位值。

```
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
    -> INTERVAL '1:1' MINUTE_SECOND);
+-----------------------------------------------------+
| DATE_ADD('1997-12-31 23:59:59', INTERVAL...         |
+-----------------------------------------------------+
| 1998-01-01 00:01:00                                 |
+-----------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 HOUR);
+-----------------------------------------------------+
| DATE_ADD('1999-01-01', INTERVAL 1 HOUR)             |
+-----------------------------------------------------+
| 1999-01-01 01:00:00                                 |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

DATE_FORMAT(date,format) ()

该函数会根据 format 字符串来格式化 date 值。

下表中列出了一些可用于 format 字符串的标识符。格式标识符第一个字符必须是 % 字符。

格式标识符	说明
%a	一星期中每天名称的缩写（Sun...Sat）
%b	月份的缩写（Jan...Dec）
%c	月份的数字表现形式（0...12）
%D	带有英语后缀的一个月中的每一天的名称（0th、1st、2nd、3rd）
%d	用数字形式表现的每月中的每一天（00...31）
%e	用数字形式表现的每月中的每一天（0...31）
%f	毫秒（000000...999999）
%H	24时制显示的小时（00...23）
%h	12时制显示的小时（01...12）
%I	12时制显示的小时（01...12）
%i	以数字形式表现的分钟数（00...59）
%j	一年中的每一天（001...366）
%k	24时制小时的另一种表现格式（0...23）
%l	12时制小时的另一种表现格式（1...12）
%M	用完整英文名称表示的月份（January...December）
%m	用数字表现的月份（00...12）
%p	上午（AM）或下午（PM）
%r	12时制的时间值（hh:mm:ss，后跟 AM 或 PM）
%S	秒（00...59）
%s	秒（00...59）
%T	24时制的小时（hh:mm:ss）
%U	星期（00...53），其中星期天是每星期的开始日
%u	星期（00...53），其中星期一是每星期的开始日
%V	星期（01...53），其中星期天是每星期的开始日，和 %X 一起使用
%v	星期（01...53），其中星期一是每星期的开始日，和 %x 一起使用
%W	一星期中各日名称（Sunday...Saturday）
%w	一星期中各日名称（0代表星期日，6代表星期六，以此类推）

格式标识符	说明
%X	某星期所处年份。其中，星期天是每星期的开始日，采用4位数字形式表现，和 %V 一起使用
%x	某星期所处年份。其中，星期一是每星期的开始日，采用4位数字形式表现，和 %V 一起使用
%Y	4位数字表示的年份
%y	2位数字表示的年份
%%	符号 % 的字面值
%x（x为斜体）	字符x的字面值，x指以上未列出的任何字符

```
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');
+------------------------------------------------------------+
| DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y')         |
+------------------------------------------------------------+
| Saturday October 1997                                      |
+------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00'
    -> '%H %k %I %r %T %S %w');
+------------------------------------------------------------+
| DATE_FORMAT('1997-10-04 22:23:00.....                  |
+------------------------------------------------------------+
|  22 22 10 10:23:00 PM 22:23:00 00 6                        |
+------------------------------------------------------------+
1 row in set (0.00 sec)
```

DATE_SUB(date,INTERVAL expr unit) ()

类似 DATE_ADD() 函数。

DAY(date) ()

DAY() 等同于 DAYOFMONTH() 。

DAYNAME(date) ()

返回 date 参数所对应的星期几。

```
mysql> SELECT DAYNAME('1998-02-05');
+-----------------------------------------------------------+
| DAYNAME('1998-02-05')                                     |
+-----------------------------------------------------------+
| Thursday                                                  |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

DAYOFMONTH(date) ()

返回 date 参数所对应的一月中的第几天，取值范围从0到31。

```
mysql> SELECT DAYOFMONTH('1998-02-03');
+-----------------------------------------------------------+
| DAYOFMONTH('1998-02-03')                                  |
+-----------------------------------------------------------+
| 3                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

DAYOFWEEK(date) ()

返回 date 参数所对应的每周中的某一天的索引值（1 = Sunday，2 = Monday……7 = Saturday）。这些索引值对应着 ODBC 标准。

```
mysql> SELECT DAYOFWEEK('1998-02-03');
+-----------------------------------------------------------+
|DAYOFWEEK('1998-02-03')                                    |
+-----------------------------------------------------------+
| 3                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

DAYOFYEAR(date) ()

返回 date 参数所对应的一年中的某一天，取值范围从1到366。

```
mysql> SELECT DAYOFYEAR('1998-02-03');
+-----------------------------------------------------------+
| DAYOFYEAR('1998-02-03')                                   |
+-----------------------------------------------------------+
```

```
| 34                                                          |
+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

EXTRACT(unit FROM date) ()

EXTRACT() 函数 使用同样的单位标识符 DATE_ADD() 或 DATE_SUB()，但是只从 date 中提取相应部分，而不执行日期运算。

```
mysql> SELECT EXTRACT(YEAR FROM '1999-07-02');
+-------------------------------------------------------------+
| EXTRACT(YEAR FROM '1999-07-02')                             |
+-------------------------------------------------------------+
| 1999                                                        |
+-------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT EXTRACT(YEAR_MONTH FROM '1999-07-02 01:02:03');
+-------------------------------------------------------------+
| EXTRACT(YEAR_MONTH FROM '1999-07-02 01:02:03')              |
+-------------------------------------------------------------+
| 199907                                                      |
+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

FROM_DAYS(N) ()

给定某日 N，返回一个 DATE 值。

```
mysql> SELECT FROM_DAYS(729669);
+-------------------------------------------------------------+
| FROM_DAYS(729669)                                           |
+-------------------------------------------------------------+
| 1997-10-07                                                  |
+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

使用 FROM_DAYS() 时，要特别注意古代的日期。该函数不适用于格里高里历（即公历）颁布（公元1582年）之前的日期。

FROM_UNIXTIME(unix_timestamp) FROM_UNIXTIME(unix_timestamp,format) ()

```
mysql> SELECT FROM_UNIXTIME(875996580);
+---------------------------------------------------------+
| FROM_UNIXTIME(875996580)                                |
+---------------------------------------------------------+
| 1997-10-04 22:23:00                                     |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

HOUR(time) ()

```
mysql> SELECT HOUR('10:05:03');
+---------------------------------------------------------+
| HOUR('10:05:03')                                        |
+---------------------------------------------------------+
| 10                                                      |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

LAST_DAY(date) ()

```
mysql> SELECT LAST_DAY('2003-02-05');
+---------------------------------------------------------+
| LAST_DAY('2003-02-05')                                  |
+---------------------------------------------------------+
| 2003-02-28                                              |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

LOCALTIME 和 LOCALTIME() ()

`LOCALTIME` 和 `LOCALTIME()` 与 `NOW()` 具有相同意义。

LOCALTIMESTAMP 和 LOCALTIMESTAMP() ()

`LOCALTIMESTAMP` 和 `LOCALTIMESTAMP()` 与 `NOW()` 具有相同意义。

MAKEDATE(year,dayofyear) ()

基于给定参数年份（ year ）和一年中的某一天（ dayofyear ），返回一个日期值。 dayofyear 必须大于0，否则结果为空。

dayofyear 与 DAYOFYEAR() (#DAYOFYEAR) 函数取值类似，取值范围为1-366。

```
mysql> SELECT MAKEDATE(2001,31), MAKEDATE(2001,32);
+-------------------------------------------------------+
| MAKEDATE(2001,31), MAKEDATE(2001,32)                  |
+-------------------------------------------------------+
| '2001-01-31', '2001-02-01'                            |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

MAKETIME(hour,minute,second) ()

基于给定的 hour 、 minute 以及 second 这3个参数，计算出一个时间值。

```
mysql> SELECT MAKETIME(12,15,30);
+-----------------------------------------------------+
| MAKETIME(12,15,30)                                  |
+-----------------------------------------------------+
| '12:15:30'                                          |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

MICROSECOND(expr) ()

基于时间或“日期+时间”的表达式 expr，返回一个以毫秒计的时间值，取值范围为0-99999。

```
mysql> SELECT MICROSECOND('12:00:00.123456');
+-----------------------------------------------------+
| MICROSECOND('12:00:00.123456')                      |
+-----------------------------------------------------+
| 123456                                              |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

MINUTE(time) ()

基于 time 参数，返回分钟数，取值范围为0-59。

```
mysql> SELECT MINUTE('98-02-03 10:05:03');
+-----------------------------------------------------------+
| MINUTE('98-02-03 10:05:03')                               |
+-----------------------------------------------------------+
| 5                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

MONTH(date) ()

基于 date 参数，返回月份值，取值范围为0-12。

```
mysql> SELECT MONTH('1998-02-03')
+-----------------------------------------------------------+
| MONTH('1998-02-03')                                       |
+-----------------------------------------------------------+
| 2                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

MONTHNAME(date) ()

基于 date 参数，返回月份的完整英文名称。

```
mysql> SELECT MONTHNAME('1998-02-05');
+-----------------------------------------------------------+
| MONTHNAME('1998-02-05')                                   |
+-----------------------------------------------------------+
| February                                                  |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

NOW() ()

返回一定格式的日期+时间值。根据函数是否用于字符串或数字内容，格式为 'YYYY-MM-DD HH:MM:SS' 或 YYYYMMDDHHMMSS。

```
mysql> SELECT NOW();
+---------------------------------------------------------+
| NOW()                                                   |
+---------------------------------------------------------+
| 1997-12-15 23:50:26                                     |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

PERIOD_ADD(P,N) ()

将 N 个月添加到时段 P（格式为 YYMM 或 YYYYMM）上，返回值格式为 YYYYMM。注意：时段参数 P 不是一个日期值。

```
mysql> SELECT PERIOD_ADD(9801,2);
+---------------------------------------------------------+
| PERIOD_ADD(9801,2)                                      |
+---------------------------------------------------------+
| 199803                                                  |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

PERIOD_DIFF(P1,P2) ()

时段 `P1` 和 `P2` 之间的月份差值。`P1` 与 `P2` 的格式应为 YYMM 或 YYYYMM。注意，时段参数 `P1` 和 `P2` 都不是日期值。

```
mysql> SELECT PERIOD_DIFF(9802,199703);
+---------------------------------------------------------+
| PERIOD_DIFF(9802,199703)                                |
+---------------------------------------------------------+
| 11                                                      |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

QUARTER(date) ()

返回参数 date 所对应的年中某季度，取值范围为1-4。

```
mysql> SELECT QUARTER('98-04-01');
+-----------------------------------------------------------+
| QUARTER('98-04-01')                                       |
+-----------------------------------------------------------+
| 2                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

SECOND(time) ()

返回参数 time 所对应的秒数，取值范围为0-59。

```
mysql> SELECT SECOND('10:05:03');
+-----------------------------------------------------------+
| SECOND('10:05:03')                                        |
+-----------------------------------------------------------+
| 3                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

SEC_TO_TIME(seconds) ()

将参数 seconds 转换成以'HH:MM:SS' 或 HHMMSS 格式（根据函数应用上下文是字符串还是数字）输出的时间值。

```
mysql> SELECT SEC_TO_TIME(2378);
+-----------------------------------------------------------+
| SEC_TO_TIME(2378)                                         |
+-----------------------------------------------------------+
| 00:39:38                                                  |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

STR_TO_DATE(str,format) ()

DATE_FORMAT() 函数的逆向函数。包含2个参数，字符串类型参数 str 和格式字符串参数 format 。返回值有2种可能性：如果格式字符串既包含日期又包含时间，则返回一个 DATETIME 值；如果格式字符串只包含日期或时间部分，则函数也相应返回 DATE 或 TIME 类型的值。

```
mysql> SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');
+---------------------------------------------------+
| STR_TO_DATE('04/31/2004', '%m/%d/%Y')             |
+---------------------------------------------------+
| 2004-04-31                                        |
+---------------------------------------------------+
1 row in set (0.00 sec)
```

SUBDATE(date,INTERVAL expr unit) 与 SUBDATE(expr,days) ()

当第二个参数采用 INTERVAL 格式时， SUBDATE() 等同于 DATE_SUB() 。要想详细了解 INTERVAL 单元参数，请参考 DATE_ADD() 。

```
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
+-------------------------------------------------+
| DATE_SUB('1998-01-02', INTERVAL 31 DAY)         |
+-------------------------------------------------+
| 1997-12-02                                      |
+-------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT SUBDATE('1998-01-02', INTERVAL 31 DAY);
+-------------------------------------------------+
| SUBDATE('1998-01-02', INTERVAL 31 DAY)          |
+-------------------------------------------------+
| 1997-12-02                                      |
+-------------------------------------------------+
1 row in set (0.00 sec)
```

SUBTIME(expr1,expr2) ()

返回值为 expr1 - expr2 ，格式与 expr1 相同。 expr1 是一个时间或日期时间表达式，而 expr2 是一个时间表达式。

```
mysql> SELECT SUBTIME('1997-12-31 23:59:59.999999',
    -> '1 1:1:1.000002');
+-----------------------------------------------------------+
| SUBTIME('1997-12-31 23:59:59.999999'...                   |
+-----------------------------------------------------------+
| 1997-12-30 22:58:58.999997                                |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

SYSDATE() ()

根据函数所应用的上下文究竟是字符串还是数字，以 'YYYY-MM-DD HH:MM:SS' 或 YYYYMMDDHHMMSS 格式返回当前日期与时间值。

```
mysql> SELECT SYSDATE();
+-----------------------------------------------------------+
| SYSDATE()                                                 |
+-----------------------------------------------------------+
| 2006-04-12 13:47:44                                       |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

TIME(expr) ()

提取时间或日期时间表达式 `expr` 的时间部分，将其作为字符串返回。

```
mysql> SELECT TIME('2003-12-31 01:02:03');
+-----------------------------------------------------------+
| TIME('2003-12-31 01:02:03')                               |
+-----------------------------------------------------------+
| 01:02:03                                                  |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

TIMEDIFF(expr1,expr2) ()

返回表示为时间值的 `expr1 - expr2`， `expr1` 和 `expr2` 都是时间或日期与时间表达式，但两者必须类型相同。

```
mysql> SELECT TIMEDIFF('1997-12-31 23:59:59.000001',
    -> '1997-12-30 01:01:01.000002');
```

```
+----------------------------------------------------------+
| TIMEDIFF('1997-12-31 23:59:59.000001'.....               |
+----------------------------------------------------------+
| 46:58:57.999999                                          |
+----------------------------------------------------------+
1 row in set (0.00 sec)
```

TIMESTAMP(expr), TIMESTAMP(expr1,expr2) ()

当只接受一个参数 `expr`（日期或日期时间类型）时，函数将这个参数以日期时间的形式返回；若接受两个参数，函数则会将时间参数 `expr2` 添加到日期或日期时间参数 `expr1` 上，以日期时间形式返回这个组合值。

```
mysql> SELECT TIMESTAMP('2003-12-31');
+---------------------------------------------------------+
| TIMESTAMP('2003-12-31')                                 |
+---------------------------------------------------------+
| 2003-12-31 00:00:00                                     |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

TIMESTAMPADD(unit,interval,datetime_expr) ()

函数将表示间隔值的整形参数 `interval` 添加到日期或日期时间参数 `datetime_expr` 上。`interval` 所采用的单位由 `unit` 参数指定。`unit` 参数的取值范围是：FRAC_SECOND、SECOND、MINUTE、HOUR、DAY、WEEK、MONTH、QUARTER 或 YEAR。

`unit` 值也可以通过一个前面介绍过的关键字来标识，或者说需要加上前缀 `SQL_TSI_`。例如：DAY 和 SQL_TSI_DAY。这两种形式都是合法的。

```
mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
+---------------------------------------------------------+
| TIMESTAMPADD(MINUTE,1,'2003-01-02')                     |
+---------------------------------------------------------+
| 2003-01-02 00:01:00                                     |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2) ()

返回两个日期或日期时间类型参数 datetime_expr1 与 datetime_epr2 之间的整数差值。返回值所采用的单位由 unit 参数指定。有关 unit 的合法值，可参看 TIMESTAMPADD() 函数介绍。

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
+-----------------------------------------------------------+
| TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01')        |
+-----------------------------------------------------------+
| 3                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

TIME_FORMAT(time,format) ()

该函数和 DATE_FORMAT() 函数用法类似，但 format 字符串中只含有与小时、分钟、秒相关的格式标识符。

如果 time 值包含一个大于23的小时数，%H 与 %k 小时格式标识符就会生成一个超出平时所用范围（0-23）的值。其他与小时相关的格式标识符会生成以12取模的值。

```
mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
+-----------------------------------------------------------+
| TIME_FORMAT('100:00:00', '%H %k %h %I %l')            |
+-----------------------------------------------------------+
| 100 100 04 04 4                                           |
+-----------------------------------------------------------+
1 row in set (0.00 sec)

```

TIME_TO_SEC(time)

将 `time` 参数转换成秒数返回。

```
mysql> SELECT TIME_TO_SEC('22:23:00');
+-------------------------------------+
| TIME_TO_SEC('22:23:00') |
+-------------------------------------+
| 80580 |
+-------------------------------------+
1 row in set (0.00 sec)
```

TO_DAYS(date)

基于日期参数 `date`，返回一个天数（自年份0开始的天数）。

mysql> SELECT TO_DAYS(950501); +---+ | TO_DAYS(950501) | +---+ | 728779 | +---+ 1 row in set (0.00 sec)

UNIX_TIMESTAMP(), UNIX_TIMESTAMP(date)

如果不传入参数调用该函数，返回一个 UNIX 时间戳，它是一个自 '1970-01-01 00:00:00' UTC（世界统一时间）起计算的秒数，无

mysql> SELECT UNIX_TIMESTAMP(); +---+ | UNIX_TIMESTAMP() | +---+ | 882226357 | +---+ 1 row in set (0.00 sec)

mysql> SELECT UNIX_TIMESTAMP('1997-10-04 22:23:00'); +---+ | UNIX_TIMESTAMP('1997-10-04 22:23:00') | +---+ | 875996580 | +---+ 1 row in set (0.00 sec)

UTC_DATE, UTC_DATE()

根据函数应用的上下文究竟是字符串还是数字，相应地以 'YYYY-MM-DD' 或 YYYYMMDD 格式返回当前的 UTC 日期值。

mysql> SELECT UTC_DATE(), UTC_DATE() + 0; +--------------------------------------+ | UTC_DATE(), UTC_DATE() + 0 | +--------------------------------------+ | 2003-08-14, 20030814 | +--------------------------------------+ 1 row in set (0.00 sec)

UTC_TIME, UTC_TIME()

根据函数应用的上下文究竟是字符串还是数字，相应地以 'HH:MM:SS' 或 HHMMSS 格式返回当前的 UTC 时间值。

mysql> SELECT UTC_TIME(), UTC_TIME() + 0; +--------------------------------------+ | UTC_TIME(), UTC_TIME() + 0 | +--------------------------------------+

----------+ | 18:07:53, 180753 | +----------
----------+ 1 row in set (0.00 sec)

UTC_TIMESTAMP, UTC_TIMESTAMP()

根据函数应用的上下文究竟是字符串还是数字，相应地以 'YYYY-MM-DD HH:MM:SS' 或 YYYYMMDDHHMMSS 格式返回当前的

mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0; +----------
----------+ | UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0 | +-----
----------+ | 2003-08-14 18:08:04, 2003
0814180804 | +----------+ 1 row in
set (0.00 sec)

WEEK(date[,mode])

该函数返回日期参数 `date` 所对应的星期序号。如果传入两个参数，则可以指定每星期起始日究竟是星期天还是星期一，以及返回值范

模式	每星期的起始天	范围	当 Week 1 是第一个星期时
0	星期日	0-53	本年有一个周日
1	星期一	0-53	本年有3天以上
2	星期日	1-53	本年有一个周日
3	星期一	1-53	本年有3天以上
4	星期日	0-53	本年有3天以上
5	星期一	0-53	本年有一个周一
6	星期日	1-53	本年有3天以上
7	星期一	1-53	本年有一个周日

mysql> SELECT WEEK('1998-02-20'); +----------
----------+ | WEEK('1998-02-20') | +----------
----------+ | 7 | +----------
---+ 1 row in set (0.00 sec)

WEEKDAY(date)

返回日期参数 `date` 所对应的星期中每天的索引值（例如，0=星期一，1=星期二，6=星期天）。

mysql> SELECT WEEKDAY('1998-02-03 22:23:00'); +----------
----------+ | WEEKDAY('1998-02-03 22:23:00') | +----------

----------+ | 1 | +----------
----------+ 1 row in set (0.00 sec)

WEEKOFYEAR(date)

返回日期参数 `date` 所对应的一年中的星期序号（范围1-53）。`WEEKOFYEAR()` 是一个兼容函数，与 WEEK(date,3)等同。

mysql> SELECT WEEKOFYEAR('1998-02-20'); +----------
----------+ | WEEKOFYEAR('1998-02-20') | +----------
----------+ | 8 | +----------
----------+ 1 row in set (0.00 sec)

YEAR(date)

返回 `date` 的年份，范围为1000-9999。当 `date` 为0时，返回0。

mysql> SELECT YEAR('98-02-03'); +----------
----------+ | YEAR('98-02-03') | +----------
----------+ | 1998 | +----------
-+ 1 row in set (0.00 sec)

YEARWEEK(date) 与 YEARWEEK(date,mode)

返回 `date` 的年份及星期序号。`mode` 参数等同于 `WEEK()` 中的 `mode` 参数。结果中的年份可能会和 `date` 参数中的年份有所

mysql> SELECT YEARWEEK('1987-01-01'); +----------
----------+ | YEAR('98-02-03')YEARWEEK('1987-01-01') | +----------
----------+ | 198653 | +----------
----------+ 1 row in set (0.00 sec)

注意，当可选参数为0或1时，`WEEK()` 函数返回的是0，和这里返回的有所不同，因为 `WEEK()` 返回的是指定年份的星期序号。

要想更深入了解有关 MySQL 日期与时间函数的相关信息，请参看[MySQL官方网站——日期与时间函数](http://dev.mysql.com/doc
MySQL UNION 关键字

可以使用 **UNION** 从一些表中相继选择行，或从 》》

MySQL 是从 4.0 版本起开始加入的UNION 这个关键字，下面就来介绍一下它的用法。

假设有3张表，它们分别列出了潜在顾客、实际顾客，以及你进货的供货商。现在你想建立一个邮件列表，将这3张表中的名字与地址合

mysql> SELECT * FROM prospect; +---------+-------+----------------------+ | fname | lname | addr | +---------+-------+----------------------+ | Peter | Jones | 482 Rush St., Apt. 402 | | Bernice | Smith | 916 Maple Dr. | +---------+-------+----------------------+ mysql> SELECT * FROM customer; +-----------+------------+-----------------------+ | last_name | first_name | address | +-----------+------------+-----------------------+ | Peterson | Grace | 16055 Seminole Ave. | | Smith | Bernice | 916 Maple Dr. | | Brown | Walter | 8602 1st St. | +-----------+------------+-----------------------+ mysql> SELECT * FROM vendor; +------------------+---------------------+ | company | street | +------------------+---------------------+ | ReddyParts, Inc. | 38 Industrial Blvd. | | Parts-to-go, Ltd. | 213B Commerce Park. | +------------------+---------------------+

3张表的列名称不同也没有关系。下面这个查询展示了如何一起选择3张表里的名字和地址。

mysql> SELECT fname, lname, addr FROM prospect -> UNION -> SELECT first_name, last_name, address FROM customer -> UNION -> SELECT company, '', street FROM vendor; +-------------------+----------+-----------------------+ | fname | lname | addr | +-------------------+----------+-----------------------+ | Peter | Jones | 482 Rush St., Apt. 402 | | Bernice | Smith | 916 Maple Dr. | | Grace | Peterson | 16055 Seminole Ave. | | Walter | Brown | 8602 1st St. | | ReddyParts, Inc. | | 38 Industrial Blvd. | | Parts-to-go, Ltd. | | 213B Commerce Park. | +-------------------+----------+-----------------------+

如果想选择所有记录，包括那些重复记录，可以使用 UNION ALL 命令。

mysql> SELECT fname, lname, addr FROM prospect -> UNION ALL -> SELECT first_name, last_name, address FROM customer -> UNION -> SELECT company, '', street FROM vendor; +-------------------+----------+-----------------------+ | fname | lname | addr | +-------------------+----------+-----------------------+ | Peter | Jones | 482 Rush St., Apt. 402 | | Bernice | Smith | 916 Maple Dr. | | Grace | Peterson | 16055 Seminole Ave. | | Bernice | Smith | 916 Maple Dr. | | Walter | Brown | 8602 1st St. | | ReddyParts, Inc. | | 38 Industrial Blvd. | | Parts-to-go, Ltd. | | 213B Commerce Park. | +-------------------+----------+-----------------------+

```
```

极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台

更多信息请访问

http://wiki.jikexueyuan.com/project/mysql/