## Setting up the database

### Creating A Database

When you have logged in and a mysql prompt is displayed, let's then take a look at the databases we currently have. To do so, we use the *SHOW DATABASES* command.

**user $** mysql -uroot -q

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 673

Server version: 5.1.62-log Gentoo Linux mysql-5.1.62-r1

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

**mysql>** SHOW DATABASES;

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

| Database |

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

| mysql |

| test |

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

2 rows in set (0.09 sec)

**Important**

Please remember that MySQL commands should end with a semicolon (;)

Despite the fact that a test database is already created, we are going to create our own. Databases are created using the *CREATE DATABASE* command. We'll create one named "gentoo".

**mysql>** SHOW CREATE DATABASE gentoo;

Query OK, 1 row affected (0.08 sec)

The response lets us know that the command was executed without any errors. In this case, 1 row was modified. This is a reference to the main mysql database, which carries a list of all the databases. We can verify the database was created by running the *SHOW DATABASES* command again.

**mysql>** SHOW DATABASES;

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

| Database |

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

| gentoo |

| mysql |

| test |

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

3 rows in set (0.00 sec)

Indeed our database has been created. In order to work with creating tables for our new gentoo database, we need to select it as our current database. To do so, we use the *USE* command. The *USE* command takes the name of the database you wish to use as your current database.

**mysql>** USE gentoo;

Database changed

And the current database is now our previously created gentoo database. Now that we're using it, we can start to create some tables and put information in them.

## Working With Tables In MySQL

### Creating a Table

In the structure of MySQL, there are databases, tables, records, and fields. Databases hold together tables, tables hold together records, records hold together fields, which contain the actual information. This structure lets users select how they want to access their information. First off, tables can be listed similarly to databases using the *SHOW TABLES* command. Right now there are no tables in our gentoo database, as running the command will show us:

**mysql>** SHOW TABLES;

Empty set (0.00 sec)

Use the *CREATE TABLE* command to create some tables. However, this command is quite different from simple '*CREATE DATABASE* command. This command takes a list of arguments. The form is as follows:

**mysql>** CREATE TABLE [table\_name] ([field\_name] [field\_data\_type]([size]);

table\_name is the name of the table we wish to create. In this case, let's make a table named developers. This table will contain the developer's name, email and job. field\_name will contain the name of the field. We have 3 required names in this case: name, email, and job. The field\_data\_type is what type of information will be stored. VARCHAR comes to working with strings. size is how much of data a single field will store. In this case, we'll use 128. This means that the field can have VARCHAR data that is 128 bytes.

**mysql>** CREATE TABLE developers (name VARCHAR(128), email VARCHAR(128), job VARCHAR(128));

Query OK, 0 rows affected (0.11 sec)

Let's check it with the SHOW TABLES command:

**mysql>** SHOW TABLES;

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

| Tables\_in\_gentoo |

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

| developers |

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

1 row in set (0.00 sec)

It doesn't seem to have any information on the types of fields we setup, we can use the DESCRIBE command (or DESC for short), which takes the name of the table as its argument.

**mysql>** DESCRIBE developers;

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

| Field | Type | Null | Key | Default | Extra |

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

| name | varchar(128) | YES | | NULL | |

| email | varchar(128) | YES | | NULL | |

| job | varchar(128) | YES | | NULL | |

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

3 rows in set (0.00 sec)

This shows the different fields and their types. It also shows a few extra attributes.

### Populating Our MySQL Database

We populate a table (or add data) using the INSERT command. It also has a specific format:

**mysql>** INSERT INTO table (col1, col2, ...) VALUES('value1', 'value2', ...);

This command is used to insert a record into table. table contains the MySQL table we wish to enter the information into. The table name may be followed by the list of columns to insert data into and VALUES() contains the values you wish to insert into the table. You may omit the list of columns if you insert a value into each one and if you write the values in the same order the columns have been defined. In this case, we want to insert data into the developers table. Let's insert sample records:

**mysql>** INSERT INTO developers VALUES('Joe Smith', 'joesmith@gentoo.org', 'toolchain');

Query OK, 1 row affected (0.06 sec)

If you don't know the order of the columns in the table or want to insert an incomplete record:

**mysql>** INSERT INTO developers (job, name) VALUES('outsourced', 'Jane Doe');

Query OK, 1 row affected (0.01 sec)

### Browsing MySQL Tables With Queries

Queries are one of the main features of any SQL database. They help us turn data in our tables into something useful. Most queries are done with the SELECT command.

(Select all entries in a table)

**mysql>** SELECT \* FROM table;

(Select specific entries in a table)

**mysql>** SELECT \* FROM table WHERE field=value;

(Select specific fields)

**mysql>** SELECT field1,field2,field3 FROM table [WHERE field=value];

We'll go ahead and run it to see what data we have so far.

**mysql>** SELECT \* FROM developers;

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

| name | email | job |

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

| Joe Smith | joesmith@gentoo.org | toolchain |

| John Doe | johndoe@gentoo.org | portage |

| Chris White | chriswhite@gentoo.org | documentation |

| Sam Smith | samsmith@gentoo.org | amd64 |

| Jane Doe | NULL | Outsourced job |

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

5 rows in set (0.00 sec)

We see both the data we inserted through INSERT. Now, let's say that we just want to see the record for Chris White. We can do so with the second form of select as shown below.

**mysql>** SELECT \* FROM developers WHERE name = 'Chris White';

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

| name | email | job |

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

| Chris White | chriswhite@gentoo.org | documentation |

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

1 row in set (0.08 sec)

The specific entry that we were looking for has been selected. Now, let's say we only wanted to know the person's job and email address, not their name. We can do so with the third form of SELECT as shown here.

**mysql>** SELECT email,job FROM developers WHERE name = 'Chris White';

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

| email | job |

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

| chriswhite@gentoo.org | documentation |

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

1 row in set (0.04 sec)

Being the root mysql user, we have unlimited permissions to do what we wish with the MySQL database. In order to control who does what with the databases, we setup privileges.

## MySQL Privileges

### Granting Privileges with GRANT

Privileges are what kind of access users have to databases, tables, pretty much anything. Right now in the gentoo database, the MySQL root account is the only account that can access it, given its permissions. The guest account will be a restricted one. All it will be able to do is get information from the database, and that's it. admin will have the same control as root, but only for the gentoo database (not the main mysql databases). Let's have a look at this somewhat simplified format of the GRANT command.

**mysql>** GRANT [privileges] ON database.\* TO '[user]'@'[host]' IDENTIFIED BY '[password]';

First we have the privileges we wish to assign, here are some of the privileges you can set:

Code

ALL - Gives the all privilege control for the database

CREATE - Allows users to create tables

SELECT - Allows users to query tables

INSERT - Allows users to insert data into a table

SHOW DATABASES - Allows users to see a list of databases

USAGE - User has no privileges

GRANT OPTION - Allows users to grant privileges

**Note**

If you're running MySQL to communicate data to a web application, CREATE, SELECT, INSERT, DELETE and UPDATE are the only permissions you will most likely need.

For our admin user, *ALL* will do. For the guest user, *SELECT* will be sufficient for read-only access. *database* is the database we wish the user to have these permissions on. In this example, *gentoo* is the database. The *.\** means all tables. If you wanted to, you could apply per table access. *user* is the name of the user and *host* is the hostname the user will be accessing from. In most cases, this will be localhost. Finally, password is the user's password.

(admin)

**mysql>** GRANT ALL ON gentoo.\* TO 'admin'@'localhost' IDENTIFIED BY 'password';

(guest)

**mysql>** GRANT SELECT ON gentoo.\* TO 'guest'@'localhost' IDENTIFIED BY 'password';

Now that we have the users setup, let's test them out. First we quit mysql:

**mysql>** quit

We're now back at the console. Now that we have our users setup, let's go ahead and see what they can do.

### Testing User Permissions

Now attempt to login as the guest user. Currently, the guest user has SELECT only privileges. This basically comes down to being able to search and nothing more. Go ahead and login with the guest account.

**user $** mysql -u guest -h localhost -p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 6 to server version: 4.0.25

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

mysql>

Now we should test the user restriction(s). Let's switch to the gentoo database:

**mysql>** USE gentoo;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

Now let's try to do something we are not supposed to. We'll attempt to create a table.

**mysql>** CREATE TABLE test (test VARCHAR(20), foobar VARCHAR(2));

ERROR 1044: Access denied for user: 'guest@localhost' to database 'gentoo'

As you can see, this function fails, as our user does not have the appropriate access. However, one access we did grant is the SELECT statement. Let's go ahead and try that:

**mysql>** SELECT \* FROM developers;

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

| name | email | job |

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

| Joe Smith | joesmith@gentoo.org | toolchain |

| John Doe | johndoe@gentoo.org | portage |

| Sam Smith | samsmith@gentoo.org | amd64 |

| Jane Doe | NULL | Outsourced job |

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

4 rows in set (0.00 sec)

This was created to show that even all permissions granted users can still have limitations. Go ahead and quit MySQL and login as the admin.

**mysql>** quit

**user $** mysql -u admin -h localhost -p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 7 to server version: 4.0.25

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

mysql>

To begin, we'll try creating a new database with our admin user. This admin user will have access similiar to the root MySQL account, and will be able to do any kind of modification to the gentoo database it chooses. This will test the user's access to the main MySQL database.

**mysql>** CREATE DATABASE gentoo2;

ERROR 1044: Access denied for user: 'admin@localhost' to database 'gentoo2'

Indeed, the admin user cannot create databases on the main MySQL database, despite all his permissions on the gentoo database. However, we're still able to use the admin account to modify the gentoo database, as shown here by this example data insertion.

**mysql>** USE gentoo;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

**mysql>** INSERT INTO developers VALUES('Bob Simmons', 'bobsimmons@gentoo.org', 'python');

Query OK, 1 row affected (0.08 sec)

The admin user can access the database as they please. Now sometimes, we need to get rid of user permissions. Let's take a look at how to disable user permissions with the REVOKE command.

### Removing User Access With The REVOKE Command

The REVOKE command lets us deny access to a user. We can either deny full access, or specific access. In fact, the format is very similiar to GRANT.

**mysql>** REVOKE [privileges] ON database.\* FROM '[user]'@'[host]';

Options here are explained in the GRANT command section. In this section however, we're going to deny full access to a user. We login as root and do the needful.

**mysql>** REVOKE ALL ON gentoo.\* FROM 'guest'@'localhost';

Query OK, 0 rows affected (0.00 sec)

**Note**

In this case, user access is simple, so per database revoking is not a problem. However, in larger cases, you would most likely be using \*.\* instead of gentoo.\* to remove user access to all other databases

Now let's quit and attempt to login as a guest user.

**user $** mysql -u guest -h localhost -p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 9 to server version: 4.0.25

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

mysql>

Although we're able to login, our access to gentoo is now gone.

**mysql>** USE gentoo;

ERROR 1044: Access denied for user: 'guest@localhost' to database 'gentoo'

The user is no longer able to access the gentoo database, but the user was still able to login, since they remain in the main MySQL database. Use DELETE to completely remove an account and the MySQL user table.

### Removing Accounts Using DELETE

The MySQL user table is a listing of all users and information about them. Make sure you're logged in as root. Then go ahead and use the main MySQL database.

**mysql>** USE mysql;

**…**

**mysql>** SHOW TABLES;

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

| Tables\_in\_mysql |

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

| columns\_priv |

| db |

| func |

| host |

| tables\_priv |

| user |

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

6 rows in set (0.00 sec)

The user table is the table we're after. In order to make things easier to read, we'll go ahead and use the third version of the SELECT statement. The fields we're after are Host and User.

**mysql>** SELECT Host,User FROM user WHERE User = 'guest';

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

| Host | User |

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

| localhost | guest |

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

1 row in set (0.00 sec)

Now that we have our information, we can get rid of the guest user. This is done with the DELETE command and the syntax is shown below.

**mysql>** DELETE FROM table WHERE field='value';

You may notice that DELETE is somewhat similiar to the SELECT statement in its format. In this case, the field will be User, and the value guest. This will delete the record in the user table where the user is guest, successfully deleting our guest user account. Let's go ahead and do that:

**mysql>** DELETE FROM user WHERE User='guest';

Query OK, 1 row affected (0.07 sec)

(FLUSH PRIVILEGES is needed to update user permissions)

**mysql>** FLUSH PRIVILEGES;

Query OK, 0 rows affected (0.00 sec)

It seems to have worked ok. Let's test by logging out and attempting to login as our guest user.

**mysql>** quit

**user $** mysql -u guest -h localhost -p

Enter password:

ERROR 1045: Access denied for user: 'guest@localhost' (Using password: YES)

Our user is now successfully deleted

**MySQL的重要語法**

<http://www.hmes.kh.edu.tw/~jona/redhat/mysqlphp/mysqlsyntax.htm>

**一、****帳號與權限**

設定 root 和其他 user 的密碼

* mysqladmin -u root password '新密碼'
* mysqladmin -u root -p
* Enter password: 　　　此時再輸入密碼(建議採用)
* use mysql;  
  mysql> UPDATE user SET password=password('新密碼') where user='root';　  
  　　只改 root 的密碼，如果沒有用 where ，則表示改全部 user 的密碼
* mysql> FLUSH PRIVILEGES;　在 mysql 資料庫內，一定要用 flush 更新記憶體上的資料

刪除空帳號

* mysql> DELETE FROM user WHERE User = '';
* mysql> FLUSH PRIVILEGES;

建立新帳號

* mysql> GRANT 權限 ON 資料庫或資料表 TO 使用者 IDENTIFIED BY '密碼';   
  　　權限   
  　　　   
  　　資料庫或資料表   
  　　　\*.\*　所有資料庫裡的所有資料表   
  　　　\*　預設資料庫裡的所有資料表   
  　　　資料庫.\*　某一資料庫裡的所有資料表　　　   
  　　　資料庫.資料表　某一資料庫裡的特定資料表   
  　　　資料表　　預設資料庫裡的某一資料表

設定/修改權限

* 用 root 登入 MySQL  
  　 mysqladmin -u root -p  
  　Enter password:
* mysql> GRANT all ON db35.\* TO s35@'localhost' IDENTIFIED BY 's35';   
  　　　把 db35 這個資料庫(含其下的所有資料表)，授權給 s35，從 localhost 上來，密碼為s35
* mysql> GRANT all ON \*.\*　 把所有資料庫及資料表授權給別人，太危險了！
* mysql> GRANT all??? ON www.\* TO '\*'@'\*' IDENTIFIED BY '';  
  　　　 把 www 這個資料庫(含其下的所有資料表)，授權給 任何機器任何人，無密碼（通常給不特定人士使用）
* mysql> FLUSH PRIVILEGES;   （最後一定要強迫更新權限）

**二、****資料庫/資料表/欄位的操作**

建立資料庫　CREATE DATABASE 資料庫名;   
語法：CREATE DATABASE db\_name

使用資料庫　USE 資料庫名;   
語法：USE db\_name

刪除資料庫　DROP DATABASE 資料庫名;   
語法：DROP DATABASE [IF EXISTS] db\_name   
　   
建立資料表   
　CREATE TABLE 資料表名 (欄位1 資料型態, 欄位2 資料型態, ......);   
語法：   
CREATE TABLE [IF NOT EXISTS] tbl\_name (create\_definition,...) [table\_options] [select\_statement]

例：  
craete database basic;  
use basic;  
create table basic(  
　　no char(4)  
　　name char(10)  
　　id char(10));

create\_definition:   
  col\_name type [NOT NULL | NULL] [DEFAULT default\_value] [AUTO\_INCREMENT]   
            [PRIMARY KEY] [reference\_definition]   
  or    PRIMARY KEY (index\_col\_name,...)   
  or    KEY [index\_name] KEY(index\_col\_name,...)   
  or    INDEX [index\_name] (index\_col\_name,...)   
  or    UNIQUE [INDEX] [index\_name] (index\_col\_name,...)   
  or    [CONSTRAINT symbol] FOREIGN KEY index\_name (index\_col\_name,...)   
            [reference\_definition]   
  or    CHECK (expr)

資料結構(type):

|  |  |
| --- | --- |
| 資料型態 | 說明 |
| TINYINT | 有符號的範圍是-128到127， 無符號的範圍是0到255。 |
| SMALLINT | 有符號的範圍是-32768到32767， 無符號的範圍是0到65535。 |
| MEDIUMINT | 有符號的範圍是-8388608到8388607， 無符號的範圍是0到16777215。 |
| INT | 有符號的範圍是-2147483648到2147483647， 無符號的範圍是0到4294967295。 |
| INTEGER | INT的同義詞。 |
| BIGINT | 有符號的範圍是-9223372036854775808到 9223372036854775807，無符號的範圍是0到18446744073709551615。 |
| FLOAT | 單精密浮點數字。不能無符號。允許的值是-3.402823466E+38到- 1.175494351E-38，0 和1.175494351E-38到3.402823466E+38。 |
| DOUBLE | 雙精密)浮點數字。不能無符號。允許的值是- 1.7976931348623157E+308到-2.2250738585072014E-308、 0和2.2250738585072014E-308到1.7976931348623157E+308。 |
| DOUBLE PRECISION | DOUBLE的同義詞。 |
| REAL | DOUBLE的同義詞。 |
| DECIMAL | DECIMAL值的最大範圍與DOUBLE相 同。 |
| NUMERIC | DECIMAL的同義詞。 |
| DATE | 日期。支援的範圍是'1000-01-01'到'9999-12-31'。 |
| DATETIME | 日期和時間組合。支援的範圍是'1000-01-01 00:00:00'到'9999-12-31 23:59:59' |
| TIMESTAMP | 時間戳記。範圍是'1970-01-01 00:00:00'到2037年的某時。 |
| TIME | 一個時間。範圍是'-838:59:59'到'838:59:59'。 |
| YEAR | 2或4位數字格式的年(內定是4位)。允許的值是1901到2155。 |
| CHAR | 固定長度，1 ～ 255個字元。 |
| VARCHAR | 可變長度，1 ～ 255個字元。 |
| TINYBLOB |  |
| TINYTEXT | 最大長度為255(2^8-1)個字符。 |
| MEDIUMBLOB |  |
| MEDIUMTEXT | 最大長度為16777215(2^24-1)個字符。 |
| LONGBLOB |  |
| LONGTEXT | 最大長度為4294967295(2^32-1)個字符。 |
| ENUM | 一個ENUM最多能有65535不同的值。 |
| SET | 一個SET最多能有64個成員。 |

index\_col\_name:   
        col\_name [(length)]

reference\_definition:   
        REFERENCES tbl\_name [(index\_col\_name,...)]   
                   [MATCH FULL | MATCH PARTIAL]   
                   [ON DELETE reference\_option]   
                   [ON UPDATE reference\_option]

reference\_option:   
        RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

table\_options:   
        type = [ISAM | MYISAM | HEAP]   
or      max\_rows = #   
or      min\_rows = #   
or      avg\_row\_length = #   
or      comment = "string"   
or      auto\_increment = #   
select\_statement:   
        [ | IGNORE | REPLACE] SELECT ...  (Some legal select statement) 

刪除資料表　DROP TABLE 資料表名;   
語法：DROP TABLE [IF EXISTS] tbl\_name [, tbl\_name,...]

改變資料表結構(新增/刪除欄位、建立/取消索引、改變欄位資料型態、欄位重新命 名)   
語法：   
ALTER [IGNORE] TABLE tbl\_name alter\_spec [, alter\_spec ...]

alter\_specification:   
        ADD [COLUMN] create\_definition [FIRST | AFTER column\_name ]   
  or    ADD INDEX [index\_name] (index\_col\_name,...)   
  or    ADD PRIMARY KEY (index\_col\_name,...)   
  or    ADD UNIQUE [index\_name] (index\_col\_name,...)   
  or    ALTER [COLUMN] col\_name {SET DEFAULT literal | DROP DEFAULT}   
  or    CHANGE [COLUMN] old\_col\_name create\_definition   
  or    MODIFY [COLUMN] create\_definition   
  or    DROP [COLUMN] col\_name   
  or    DROP PRIMARY KEY   
  or    DROP INDEX key\_name   
  or    RENAME [AS] new\_tbl\_name   
  or    table\_option   
範例：   
　欄位重新命名   
　mysql> ALTER TABLE t1 CHANGE a b INTEGER;   
　　將資料表 t1 欄位 a 改名為 b (其資料型態是 integer)   
　改變欄位資料型態   
　mysql> ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;   
　mysql> ALTER TABLE t1 MODIFY b BIGINT NOT NULL;   
　　將資料表 t1 欄位 b 的資料型態改為 bigint not null

　mysql> CREATE TABLE t1 (a INTEGER,b CHAR(10));　   
　mysql> ALTER TABLE t1 RENAME t2;   
　　將資料表 t1 改名為 t2   
　mysql> ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);   
　　將資料表 t2 欄位 a 資料型態由 integer 改為 tinyint not null ，欄位 b 改名為 c 資料型態改為 char(20)   
　mysql> ALTER TABLE t2 ADD d TIMESTAMP;   
　　在資料表 t2 增加新欄位 d 資料型態是 timestamp   
　mysql> ALTER TABLE t2 ADD INDEX (d), ADD PRIMARY KEY (a);   
　　在資料表 t2 ，對 d 欄位做索引，並以欄位 a 作為主索引鍵   
　mysql> ALTER TABLE t2 DROP COLUMN c;   
　　刪除欄位 c   
 　mysql> ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO\_INCREMENT, ADD INDEX (c);   
　　新增欄位 c，並做索引(做索引的欄位必須為 not null )

資料表最佳化　OPTIMIZE TABLE 資料表名   
語法：OPTIMIZE TABLE tbl\_name   
　欄位長度有變動、刪除大量資料，都應進行資料表最佳化

**三、****紀錄的操作**

插入一筆或多筆紀錄　INSERT INTO 資料表(欄位1,欄位2,......) VALUES(值1,值2,......), (值1,值2,......), ........   
　　(MySQL 3.22.5 以後可插入多筆記錄)   
語法：   
    INSERT [LOW\_PRIORITY | DELAYED] [IGNORE]   
        [INTO] tbl\_name [(col\_name,...)]   
        VALUES (expression,...),(...),...   
or  INSERT [LOW\_PRIORITY | DELAYED] [IGNORE]   
        [INTO] tbl\_name [(col\_name,...)]   
        SELECT ...   
or  INSERT [LOW\_PRIORITY | DELAYED] [IGNORE]   
        [INTO] tbl\_name   
        SET col\_name=expression, col\_name=expression, ...   
範例：   
　mysql> INSERT INTO tbl\_name (col1,col2) VALUES(15,col1\*2);   
　　不可寫成   
　mysql> INSERT INTO tbl\_name (col1,col2) VALUES(col2\*2,15);   
　　因為：欄位 col1 的值先填入後，才可以計算欄位 col2

從檔案讀入資料   
語法：   
　LOAD DATA [LOCAL] INFILE 'file\_name.txt' [REPLACE | IGNORE]   
    INTO TABLE tbl\_name   
    [FIELDS   
        [TERMINATED BY '\t'] 　　　每一欄位以某字元分開(內定是 tab)   
        [OPTIONALLY] ENCLOSED BY "] 　　每一欄位以某字元括住(內定是不使用括號)   
        [ESCAPED BY '\\' ]] 　　　　   
    [LINES TERMINATED BY '\n']　　設定換行的字元(內 定是 \n)   
    [IGNORE number LINES] 　　　忽略最前面幾行(最前面幾筆記錄不抄進來)   
    [(col\_name,...)]   
範例：   
　mysql> USE db1;   
　mysql> LOAD DATA INFILE "./data.txt" INTO TABLE db2.my\_table;   
　　從目前 MySQL 目錄讀入 data.txt   
　mysql> LOAD DATA INFILE "./88.txt" INTO TABLE TEACHER FIELDS TERMINATED BY ' ' ;   
　　從目前 MySQL 目錄(我的在 /var/lib/mysql )讀入 data.txt ，每一欄位以 空白 分開   
　mysql> LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata (col1,col2,...);   
　　只將 persondata.txt 裡某些欄位的資料抓過來

刪除紀錄　DELETE [LOW-PRIORITY] FROM 資料表名 WHERE 條件 [LIMIT rows]   
語法：   
　DELETE [LOW\_PRIORITY] FROM tbl\_name   
    [WHERE where\_definition] [LIMIT rows]   
　　LOW-PRIORITY　是等到沒有用戶端使用時再刪   
　　LIMIT rows　限制刪除紀錄的筆數   
範例：   
　mysql> DELETE FROM 資料表名;   
　　刪除所有紀錄   
　mysql> DELETE FROM 資料表名 WHERE 1>0;   
　　刪除所有紀錄，但速度較慢，方便在螢幕上看 

更新一筆紀錄 (語法與 INSERT 相同)   
　REPLACE INTO 資料表(欄位1,欄位2,......) VALUES(值1,值2,......)   
語法：   
REPLACE [LOW\_PRIORITY | DELAYED]   
        [INTO] tbl\_name [(col\_name,...)]   
        VALUES (expression,...)   
or  REPLACE [LOW\_PRIORITY | DELAYED]   
        [INTO] tbl\_name [(col\_name,...)]   
        SELECT ...   
or  REPLACE [LOW\_PRIORITY | DELAYED]   
        [INTO] tbl\_name   
        SET col\_name=expression, col\_name=expression,...

更新多筆紀錄   
語法：   
UPDATE [LOW\_PRIORITY] tbl\_name SET col\_name1=expr1,col\_name2=expr2,...     [WHERE where\_definition]   
　　如果沒有設定 WHERE 條件，則整個資料表相關的欄位都更新   
範例：   
　mysql> UPDATE persondata SET age=age+1;   
　　將資料表 persondata 中，所有 age 欄位都加 1   
　mysql> UPDATE persondata SET age=age\*2, age=age+1;   
　　將資料表 persondata 中，所有 age 欄位都\*2，再加 1

**四、****資料的輸出**

SELECT

語法：   
　SELECT [STRAIGHT\_JOIN] [SQL\_SMALL\_RESULT] [DISTINCT | ALL]   
    select\_expression,...   
    [INTO OUTFILE 'file\_name' export\_options]   
    [FROM table\_references   
        [WHERE where\_definition]   
        [GROUP BY col\_name,...]   
        [HAVING where\_definition]   
        [ORDER BY {unsigned\_integer | col\_name} [ASC | DESC] ,...]   
        [LIMIT [offset,] rows]   
        [PROCEDURE procedure\_name] ]

範例：  
  
**排序輸出**select \* from 資料表名 order by 欄位名1,欄位名2,欄位名3......

**反向排序輸出**select \* from 資料表名 order by 欄位名1,欄位名2,欄位名3...... desc

mysql> select concat(last\_name,', ',first\_name) AS full\_name   
  
　　　　　　from mytable ORDER BY full\_name;  
  
  
  
　mysql> select t1.name, t2.salary from employee AS t1, info AS t2  
  
           where t1.name = t2.name;  
  
　　顯示資料庫 employee(別名 t1) 裡，資料表 t1 的欄位 name 和 資料表 t2 的欄位 salary 當.....  
  
　mysql> select t1.name, t2.salary from employee t1, info t2           where t1.name = t2.name;  
  
  
  
　mysql> select college, region, seed from tournament  
  
           ORDER BY region, seed;  
  
　mysql> select college, region AS r, seed AS s from tournament  
  
           ORDER BY r, s;  
  
　mysql> select college, region, seed from tournament  
  
           ORDER BY 2, 3;  
  
  
  
　mysql> select col\_name from tbl\_name HAVING col\_name > 0;  
  
  
  
　mysql> select col\_name from tbl\_name WHERE col\_name > 0;  
  
  
  
　mysql> select user,max(salary) from users  
  
           group by user HAVING max(salary)>10;  
  
  
  
 mysql> select user,max(salary) AS sum from users  
  
           group by user HAVING sum>10;  
  
  
  
　mysql> select \* from table LIMIT 5,10;  # Retrieve rows 6-15  
  
  
  
　mysql> select \* from table LIMIT 5;     # Retrieve first 5 rows



在命令列下進行批次處理：  
shell> mysql -h host -u user -p < batch-file  
Enter password: \*\*\*\*\*\*\*\*

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Bytes** | **From** | **To** |
| TINYINT | 1 | -128 | 127 |
| SMALLINT | 2 | -32768 | 32767 |
| MEDIUMINT | 3 | -8388608 | 8388607 |
| INT | 4 | -2147483648 | 2147483647 |
| BIGINT | 8 | -9223372036854775808 | 9223372036854775807 |

|  |  |
| --- | --- |
| **Column type** | **``Zero'' value** |
| DATETIME | '0000-00-00 00:00:00' |
| DATE | '0000-00-00' |
| TIMESTAMP | 00000000000000 (length depends on display size) |
| TIME | '00:00:00' |
| YEAR | 0000 |

|  |  |
| --- | --- |
| **Column type** | **Display format** |
| TIMESTAMP(14) | YYYYMMDDHHMMSS |
| TIMESTAMP(12) | YYMMDDHHMMSS |
| TIMESTAMP(10) | YYMMDDHHMM |
| TIMESTAMP(8) | YYYYMMDD |
| TIMESTAMP(6) | YYMMDD |
| TIMESTAMP(4) | YYMM |
| TIMESTAMP(2) | YY |

|  |  |  |
| --- | --- | --- |
| **Type** | **Max.size** | **Bytes** |
| TINYTEXT or TINYBLOB | 2^8-1 | 255 |
| TEXT or BLOB | 2^16-1 (64K-1) | 65535 |
| MEDIUMTEXT or MEDIUMBLOB | 2^24-1 (16M-1) | 16777215 |
| LONGBLOB | 2^32-1 (4G-1) | 4294967295 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Value** | CHAR(4) | **Storage required** | VARCHAR(4) | **Storage required** |
| '' | ' ' | 4 bytes | '' | 1 byte |
| 'ab' | 'ab ' | 4 bytes | 'ab' | 3 bytes |
| 'abcd' | 'abcd' | 4 bytes | 'abcd' | 5 bytes |
| 'abcdefgh' | 'abcd' | 4 bytes | 'abcd' | 5 bytes |

|  |  |
| --- | --- |
| **Value** | **Index** |
| NULL | NULL |
| "" | 0 |
| "one" | 1 |
| "two" | 2 |
| "three" | 3 |

|  |  |
| --- | --- |
| **Other vendor type** | **MySQL type** |
| BINARY(NUM) | CHAR(NUM) BINARY |
| CHAR VARYING(NUM) | VARCHAR(NUM) |
| FLOAT4 | FLOAT |
| FLOAT8 | DOUBLE |
| INT1 | TINYINT |
| INT2 | SMALLINT |
| INT3 | MEDIUMINT |
| INT4 | INT |
| INT8 | BIGINT |
| LONG VARBINARY | MEDIUMBLOB |
| LONG VARCHAR | MEDIUMTEXT |
| MIDDLEINT | MEDIUMINT |
| VARBINARY(NUM) | VARCHAR(NUM) BINARY |

|  |  |
| --- | --- |
| **Column type** | **Storage required** |
| TINYINT | 1 byte |
| SMALLINT | 2 bytes |
| MEDIUMINT | 3 bytes |
| INT | 4 bytes |
| INTEGER | 4 bytes |
| BIGINT | 8 bytes |
| FLOAT(X) | 4 if X <= 24 or 8 if 25 <= X <= 53 |
| FLOAT | 4 bytes |
| DOUBLE | 8 bytes |
| DOUBLE PRECISION | 8 bytes |
| REAL | 8 bytes |
| DECIMAL(M,D) | M+2 bytes if D > 0, M+1 bytes if D = 0 (D+2, if M < D) |
| NUMERIC(M,D) | M+2 bytes if D > 0, M+1 bytes if D = 0 (D+2, if M < D) |

|  |  |
| --- | --- |
| **Column type** | **Storage required** |
| DATE | 3 bytes |
| DATETIME | 8 bytes |
| TIMESTAMP | 4 bytes |
| TIME | 3 bytes |
| YEAR | 1 byte |

|  |  |
| --- | --- |
| **Column type** | **Storage required** |
| CHAR(M) | M bytes, 1 <= M <= 255 |
| VARCHAR(M) | L+1 bytes, where L <= M and 1 <= M <= 255 |
| TINYBLOB, TINYTEXT | L+1 bytes, where L < 2^8 |
| BLOB, TEXT | L+2 bytes, where L < 2^16 |
| MEDIUMBLOB, MEDIUMTEXT | L+3 bytes, where L < 2^24 |
| LONGBLOB, LONGTEXT | L+4 bytes, where L < 2^32 |
| ENUM('value1','value2',...) | 1 or 2 bytes, depending on the number of enumeration values (65535 values maximum) |
| SET('value1','value2',...) | 1, 2, 3, 4 or 8 bytes, depending on the number of set members (64 members maximum) |