MySQL\_for\_Python\_Albert\_c08

eating Users and

Granting Access

There is arguably no part of database programming that requires greater care in

implementation than creating users and granting access. While the careless or

malicious deletion of data is catastrophic, it can only occur at the hands of a trusted

insider (whether a programmer or a database administrator) after a sufficient access

control system is properly put into place.

MySQL for Python allows us to automate user creation and to administer access

controls. In this chapter, we will see:

•How to create and remove users in MySQL

•Why removing a user is not as easy as DROPping (using DROP) them

•The limits of MySQLdb's ability to GRANT access

•Ways to automate user creation and access control

The project for this chapter will build on the web application that we began in the

last chapter.

A word on security

On October 2, 2007, a colocation data centre run by C I Host in Chicago, Illinois

(U.S.A) was robbed for the fourth time in two years. Robbers took $15,000 worth

of servers.

On December 6, 2007, thieves stole $4 million worth of servers from a Verizon data

centre in London, England.Creating Users and Granting Access

On May 5, 2008, Peter Gabriel's official website went offline. Users of his Real World-

Peter Gabriel and WOMAD services lost all access. The reason is that thieves had

stolen the servers from a data centre run by a Carphone Warehouse subsidiary

named Rednet Ltd.

As mentioned previously, if an adequate access control system is in place, rogue

users are trusted insiders by definition. However, it is worth noting that such a

control system is not merely software. Every avenue of attack—from brute force

password cracking to using social engineering on a receptionist, is a vector of

attack against a database. By virtue of its involving human users, an adequate

access control system must provide adequate security policies and practices for that

human component. All the software access controls in the world will not prevent a

snookered staffer from disclosing their login information.

An adequate access control system involves security on every level of access—the

local host, the LAN/WAN, the wider Internet, and any physical access to any

machine located in those areas. If a user from an untrusted area gains physical access

to a machine within an area of greater trust, the chance of damage to the database

rises. Therefore, software security must always be buttressed by physical security.

At the time of writing this book, we live in a world where bootable CDs are easily

obtained, passwords can be read out of RAM, and quantum encryption is available

on an IC chip. Therefore, even full-disk encryption serves to slow and not stop

thieves from accessing data. At the present point of technology, when physical

security is breached, all IT security is increasingly worthless.

Creating users in MySQL

To create a user in MySQL, our user account must have the universal CREATE USER

privilege. In general, no user beyond the database administrator should have this as

it allows the user to create, remove, rename, and revoke the privileges of users on the

database. We will look at the granting of privileges later in this chapter.

Alternatively, if a user has universal INSERT privileges, that user can insert a new

user and relevant data into the user table of the mysql database. This method is

prone to error and can endanger the stability of the entire MySQL installation if

something goes wrong. Therefore, we do not deal with it here.

When creating a user, we need to specify both the user's name or ID and the user's

password. The basic syntax for user creation is as follows:

CREATE USER <userid>;

[ 220 ]Chapter 8

Breaking the statement up by its token, we first tell MySQL that we want to CREATE

something. Then we clarify the object being created as a USER account. That account

should be named with the given userid. An example of this statement is:

CREATE USER exemplar;

Note that if NO\_AUTO\_CREATE\_USER is enabled in your MySQL

configuration, this type of user creation will fail. This is particularly true

in SQL\_MODE.

While this statement will create a user, it is perhaps the least secure way to do

so. MySQL offers two ways of securing user accounts at the time of account

creation—passwords and host restrictions.

Forcing the use of a password

To force the use of a password, we need to declare one at the time of creating

the user. To do this, we append an IDENTIFIED BY clause to the previous user

creation syntax.

CREATE USER <userid> IDENTIFIED BY '<password>';

To use this for our exemplary user, the statement would read as follows:

CREATE USER 'exemplar' IDENTIFIED BY 'MoreSecurity';

This forces the use of a password to log in. In the first user creation statement, if the

user did not offer a password, they would still be allowed access simply by using an

existing user ID.

To reset a password for an account that is already created, use a SET

PASSWORD statement:

SET PASSWORD FOR 'exemplar'@'localhost' = PASSWORD('dogcatcher')

Which hostname you use depends on how the account was created. See the next

section for more.

Restricting the client's host

In addition to requiring a password, MySQL also provides the ability to restrict

the host from which the login may come. This clarification comes immediately

after userid.

CREATE USER <userid>@'<host name>' IDENTIFIED BY '<password>';

[ 221 ]Creating Users and Granting Access

If we want the user exemplar to login only from the localhost, the machine on which

MySQL is running—we would use the following command:

CREATE USER 'exemplar'@'localhost' IDENTIFIED BY 'MoreSecurity';

If we want the user to log in from http://www.sample.com/ only, we would change

the preceding statement to this:

CREATE USER 'exemplar'@'sample.com' IDENTIFIED BY 'MoreSecurity';

In doing so, however, we make it impossible for them to log in from localhost.

To ensure that the user can log in from any host, we need to issue two CREATE

statements that can effectively create one user for the local host and one for the

remote client.

To quantify the host for all hosts, we use MySQL's pattern matching in lieu of a

hostname. In order to allow the user to truly login from any host then, we need to

issue the following two commands:

CREATE USER 'exemplar'@'localhost' IDENTIFIED BY 'MoreSecurity';

CREATE USER 'exemplar'@'%' IDENTIFIED BY 'MoreSecurity';

You will recall that the first user creation statement did not specify the host to be

used by the user. If the hostname is not specified in the CREATE statement, MySQL

uses % by default.

The user's identity on a MySQL database is determined by their user ID and the

hostname from which they log in. It is possible for these two items to match more

than one row in the MySQL user tables. When that happens, MySQL uses the first

match it finds.

As MySQL uses its own user tables to validate logins, it is worth asking why we

don't simply modify MySQL's own tables instead of issuing CREATE statements. One

can do this, and there are instructions for doing so in the MySQL documentation.

However, doing so leaves one open to errors and possible data corruption.

If one corrupts the data in the user tables, it is possible to lose the ability to contact

the database altogether. This would then require manually restarting MySQL using

mysqld\_safe:

mysqld\_safe --skip-grant-tables &

This method works on Unix-based systems. Windows systems are more complex.

For more information, see the MySQL manual: http://dev.mysql.com/doc/

refman/5.5/en/resetting-permissions.html

[ 222 ]Chapter 8

Creating users from Python

The ability to create users is obviously an administrative task. By default, this means

that one must log in as root, or any other user who has administrative rights, to use

them. If your Python program does not login as root, it will not be able to affect user

creation. Therefore, one's connection credentials must read accordingly:

import MySQLdb

mydb = MySQLdb.connect(host = 'localhost',

user = 'root',

passwd = 'rootsecret')

cursor = mydb.cursor()

From here, one can similarly form the statement to the other CREATE statements that

we have used.

statement = """CREATE USER 'exemplar'@'localhost' IDENTIFIED BY

'MoreSecurity'"""

cursor.execute(statement)

In a Python shell, passing the statement through cursor.execute() will return 0L.

If you execute this code in a Python program file, you will not get such feedback

(unless you overtly tell Python to print it). But for debugging purposes, you have

two choices: check the MySQL users table or try to use the login.

The latter is simply a matter of creating a second connection. This is best placed in a

try...except structure:

try:

mydb2 = MySQLdb.connect(host = 'localhost',

user = 'exemplar',

passwd = 'MoreSecurity')

cursor2 = mydb2.cursor()

except:

raise

We can check the MySQL users table manually or within a program. To affect a

check manually, log into MySQL and select the mysql database for use.

use mysql;

Within mysql, read out from the user table.

SELECT \* FROM users;

[ 223 ]Creating Users and Granting Access

Depending on how many users are registered in the database, this table will

probably run off the screen. However, the last entry should be the account just

created. Alternatively, use a WHERE clause to quantify what you want.

SELECT \* FROM users WHERE User = '<userid>';

So for the user exemplar, we would enter the following statement:

SELECT \* FROM user WHERE User='exemplar';

Within Python, we can issue the last statement as follows:

mycheck = MySQLdb.connect(host = 'localhost',

user = 'root',

passwd = 'rootsecret',

db = 'mysql')

checker = mycheck.cursor()

statement = """SELECT \* FROM user WHERE User='exemplar'"""

results = checker.execute(statement)

if results == 1:

print "Success!"

else:

print "Failure."

Using the user ID in the statement saves us from having to match each record against

that value.

Removing users in MySQL

As with creating databases and tables, the opposite of creating a user is to DROP. As

we shall see, removing a user does not revert any changes that they have made to

the database(s) to which they had access. If a user had the ability to create users,

removing them will not remove the users they created.

Unlike databases and tables, dropping a user requires that you also specify the

hostname of the user's record. Therefore, one cannot always enter:

DROP USER exemplar;

This will only work if the user was created without specifying the hostname.

If it exists, one must include the hostname. For best practice, the basic syntax is:

DROP USER <userid>@<hostname>;

[ 224 ]Chapter 8

Therefore to drop user exemplar, we would pass the following statement:

DROP USER 'exemplar'@@'localhost';

Note that this will not impact that user's ability to log in from another host if that

user had permission to connect from the other host.

DROP, by design, only removes the user's account and its privileges. It does not in any

way affect any database objects that the user created. Therefore, if a user has created

false database structures such as databases, tables, and records, then all of that will

persist after the user is removed from the system.

One very important aspect of DROP that is critical to remember is that DROP does not

impact on existing user sessions. If a user is logged into the server when the DROP

statement is issued, the DROP statement will not take effect until the user has logged

out. The user's subsequent attempts at logging in will then fail.

DROPping users in Python

Dropping a user in Python is as easy as passing the MySQL statement through

Cursor.execute(). So the syntax is:

DROP USER exemplar@localhost;

This previous syntax can be changed to:

mydb = MySQLdb.connect(host = 'localhost',

user = 'root',

passwd = 'rootsecret')

cursor = mydb.cursor()

statement = """DROP USER exemplar@localhost"""

cursor.execute(statement)

However, any part of the statement can be dynamically created through the use of

string formatting conventions.

GRANT access in MySQL

After creating a user account, one still needs to tell MySQL what kind of privileges to

assign to it. MySQL supports a wide range of privileges (see the table of privileges on

page 9). A user can only grant any privilege that they have themselves.

As with creating a user, granting access can be done by modifying the mysql tables

directly. However, this method is error-prone and dangerous to the stability of the

system and is, therefore, not recommended.

[ 225 ]Creating Users and Granting Access

Important dynamics of GRANTing access

Where CREATE USER causes MySQL to add a user account, it does not specify that

user's privileges. In order to grant a user privileges, the account of the user granting

the privileges must meet two conditions:

•Be able to exercise those privileges in their account

•Have the GRANT OPTION privilege on their account

Therefore, it is not just users who have a particular privilege or only users with the

GRANT OPTION privilege who can authorize a particular privilege for a user, but only

users who meet both requirements.

Further, privileges that are granted do not take effect until the user's first login after

the command is issued. Therefore, if the user is logged into the server at the time you

grant access, the changes will not take effect immediately.

The GRANT statement in MySQL

The syntax of a GRANT statement is as follows:

GRANT <privileges> ON <database>.<table>

TO '<userid>'@'<hostname>';

Proceeding from the end of the statement, the userid and hostname follow the same

pattern as with the CREATE USER statement. Therefore, if a user is created with a

hostname specified as localhost and you grant access to that user with a hostname

of '%', they will encounter a 1044 error stating access is denied.

The database and table values must be specified individually or collectively. This

allows us to customize access to individual tables as necessary. For example, to

specify access to the city table of the world database, we would use world.city.

In many instances, however, you are likely to grant the same access to a user for all

tables of a database. To do this, we use the universal quantifier ('\*'). So to specify all

tables in the world database, we would use world.\*.

We can apply the asterisk to the database field as well. To specify all databases and

all tables, we can use \*.\*. MySQL also recognizes the shorthand \* for this.

Finally, the privileges can be singular or a series of comma-separated values. If, for

example, you want a user to only be able to read from a database, you would grant

them only the SELECT privilege. For many users and applications, reading and

writing is necessary but no ability to modify the database structure is warranted. In

such cases, we can grant the user account both SELECT and INSERT privileges with

SELECT, INSERT.

[ 226 ]Chapter 8

To learn which privileges have been granted to the user account you are

using, use the statement SHOW GRANTS FOR <user>@hostname>;.

With this in mind, if we wanted to grant a user tempo all access to all tables in

the music database but only when accessing the server locally, we would use

this statement:

GRANT ALL PRIVILEGES ON music.\* TO 'tempo'@'localhost';

Similarly, if we wanted to restrict access to reading and writing when logging in

remotely, we would change the above statement to read:

GRANT SELECT,INSERT ON music.\* TO 'tempo'@'%';

If we wanted user conductor to have complete access to everything when logged in

locally, we would use:

GRANT ALL PRIVILEGES ON \* TO 'conductor'@'localhost';

Building on the second example statement, we can further specify the exact

privileges we want on the columns of a table by including the column numbers

in parentheses after each privilege. Hence, if we want tempo to be able to read

from columns 3 and 4 but only write to column 4 of the sheets table in the music

database, we would use this command:

GRANT SELECT (col3,col4),INSERT (col4) ON music.sheets TO 'tempo'@'%';

Note that specifying columnar privileges is only available when specifying a single

database table—use of the asterisk as a universal quantifier is not allowed. Further,

this syntax is allowed only for three types of privileges: SELECT, INSERT, and UPDATE.

A list of privileges that are available through MySQL is reflected in the

following table:

PrivilegeFunctionContext

ALL

CREATE

DROPGrants all privileges to userDatabases, tables, or indexes

Creates database objectsDatabases, tables, or indexes

Drops database objectsDatabases or tables

GRANT OPTIONGrants privileges to other usersDatabases, tables, or stored

routines

REFERENCESSupported internally but

otherwise unusedDatabases or tables

[ 227 ]Creating Users and Granting Access

PrivilegeFunctionContext

ALTERAllows use of ALTER TABLETables

DELETEAllows use of DELETETables

INDEXEnables creation and dropping

of indexesTables

Allows data insertionTables

Allows reading from a databaseTables

Allows use of UPDATETables

INSERT

SELECT

UPDATE

Allows user to create temporary

CREATE TEMPORARY TABLES tables

Tables

LOCK TABLESEnables the use of LOCK

TABLES for tables on which

SELECT has been grantedTables

TRIGGERAllows the automation of

certain events in a table under

conditions set by the userTables

CREATE VIEWEnables the creation and deletion

Views

of views

SHOW VIEWAllows the showing of viewsViews

ALTER ROUTINEAllows user to alter and delete

stored routinesStored routines

CREATE ROUTINEEnables the creation of stored

routinesStored routines

EXECUTEAllows the execution of stored

routinesStored routines

FILEEnables file access on localhostFile access

CREATE USEREnables the creation of usersServer administration

PROCESSEnables the user to view

all processes with SHOWServer administration

RELOADPROCESSLIST

Enables use of FLUSHServer administration

REPLICATION CLIENTAllows user to query about

master and slave serversServer administration

REPLICATION SLAVEAllows slave servers to read

binary logs from the master

serverServer administration

SHOW DATABASESAllows user to view available

databasesServer administration

SHUTDOWN

Enables the use of

mysqladmin shutdown

[ 228 ]

Server administrationChapter 8

PrivilegeFunctionContext

SUPEREnables the use of several

superuser privilegesServer administration

ALL [PRIVILEGES]Grants all privileges to the user

that are available to the grantorServer administration

USAGEAllows access to the userServer administration

MySQL does not support the standard SQL UNDER privilege and does not support

the use of TRIGGER until MySQL 5.1.6.

More information on MySQL privileges can be found at

http://dev.mysql.com/doc/refman/5.5/en/privileges-provided.html

Using REQUIREments of access

Using GRANT with a REQUIRE clause causes MySQL to use SSL encryption.

The standard used by MySQL for SSL is the X.509 standard of the International

Telecommunication Union's (ITU) Standardization Sector (ITU-T). It is a commonly used

public-key encryption standard for single sign-on systems. Parts of the standard are

no longer in force. You can read about the parts which still apply on the ITU website

at http://www.itu.int/rec/T-REC-X.509/en

The REQUIRE clause takes the following arguments with their respective meanings

and follows the format of their respective examples:

•

NONE: The user account has no requirement for an SSL connection. This is

the default.

GRANT SELECT (col3,col4),INSERT (col4) ON music.sheets TO

'tempo'@'%';

•

SSL: The client must use an SSL-encrypted connection to log in. In most

MySQL clients, this is satisfied by using the --ssl-ca option at the time of

login. Specifying the key or certificate is optional.

GRANT SELECT (col3,col4),INSERT (col4) ON music.sheets TO

'tempo'@'%' REQUIRE SSL;

[ 229 ]Creating Users and Granting Access

•

X509: The client must use SSL to login. Further, the certificate must be

verifiable with one of the CA vendors. This option further requires the client

to use the --ssl-ca option as well as specifying the key and certificate using

--ssl-key and --ssl-cert, respectively.

GRANT SELECT (col3,col4),INSERT (col4) ON music.sheets TO

'tempo'@'%' REQUIRE X509;

•

CIPHER: Specifies the type and order of ciphers to be used.

GRANT SELECT (col3,col4),INSERT (col4) ON music.sheets TO

'tempo'@'%' REQUIRE CIPHER 'RSA-EDH-CBC3-DES-SHA';

•

ISSUER: Specifies the issuer from whom the certificate used by the client

is to come. The user will not be able to login without a certificate from

that issuer.

GRANT SELECT (col3,col4),INSERT (col4) ON music.sheets TO

'tempo'@'%' REQUIRE ISSUER 'C=ZA, ST=Western Cape, L=Cape

Town, O=Thawte Consulting cc, OU=Certification Services

Division,CN=Thawte Server CA/emailAddress=server-certs@thawte.

com';

•

SUBJECT: Specifies the subject contained in the certificate that is valid for

that user. The use of a certificate containing any other subject is disallowed.

GRANT SELECT (col3,col4),INSERT (col4) ON music.sheets

TO 'tempo'@'%' REQUIRE SUBJECT 'C=US, ST=California,

L=Pasadena, O=Indiana Grones, OU=Raiders, CN=www.lostarks.com/

emailAddress=indy@lostarks.com';

Using a WITH clause

MySQL's WITH clause is helpful in limiting the resources assigned to a user. WITH

takes the following options:

•GRANT OPTION: Allows the user to provide other users of any privilege that

they have been granted

•MAX\_QUERIES\_PER\_HOUR: Caps the number of queries that the account is

•MAX\_UPDATES\_PER\_HOUR: Limits how frequently the user is allowed to issue

UPDATE statements to the database

•MAX\_CONNECTIONS\_PER\_HOUR: Limits the number of logins that a user is

•MAX\_USER\_CONNECTIONS: Caps the number of simultaneous connections that

allowed to request in one hour

allowed to make in one hour

the user can make at one time

[ 230 ]Chapter 8

It is important to note that the GRANT OPTION argument to WITH has a timeless aspect.

It does not statically apply to the privileges that the user has just at the time of

issuance, but if left in effect, applies to any options the user has at any point in time.

So, if the user is granted the GRANT OPTION for a temporary period, but the option

is never removed, then the user grows in responsibilities and privileges, that user

can grant those privileges to any other user. Therefore, one must remove the GRANT

OPTION when it is not longer appropriate.

Note also that if a user with access to a particular MySQL database has

the ALTER privilege and is then granted the GRANT OPTION privilege,

that user can then grant ALTER privileges to a user who has access to

the mysql database, thus circumventing the administrative privileges

otherwise needed.

The WITH clause follows all other options given in a GRANT statement. So, to grant

user tempo the GRANT OPTION, we would use the following statement:

GRANT SELECT (col3,col4),INSERT (col4) ON music.sheets TO 'tempo'@'%'

WITH GRANT OPTION;

If we want to limit the number of queries that the user can have in one hour to five,

as well, we simply add to the argument of the single WITH statement. We do not need

to use WITH a second time.

GRANT SELECT,INSERT ON music.sheets TO 'tempo'@'%' WITH GRANT OPTION

MAX\_QUERIES\_PER\_HOUR 5;

More information on the many uses of WITH can be found at

http://dev.mysql.com/doc/refman/5.1/en/grant.html

Granting access in Python

Using MySQLdb to enable user privileges is not more difficult than doing so in

MySQL itself. As with creating and dropping users, we simply need to form the

statement and pass it to MySQL through the appropriate cursor.

As with the native interface to MySQL, we only have as much authority in

Python as our login allows. Therefore, if the credentials with which a cursor is

created has not been given the GRANT option, an error will be thrown by MySQL

and MySQLdb, subsequently.

[ 231 ]Creating Users and Granting Access

Assuming that user skipper has the GRANT option as well as the other necessary

privileges, we can use the following code to create a new user, set that user's

password, and grant that user privileges:

#!/usr/bin/env python

import MySQLdb

host = 'localhost'

user = 'skipper'

passwd = 'secret'

mydb = MySQLdb.connect(host, user, passwd)

cursor = mydb.cursor()

try:

mkuser = 'symphony'

creation = "CREATE USER %s@'%s'" %(mkuser, host)

results = cursor.execute(creation)

print "User creation returned", results

mkpass = 'n0n3wp4ss'

setpass = "SET PASSWORD FOR '%s'@'%s' = PASSWORD('%s')" %(mkuser,

host, mkpass)

results = cursor.execute(setpass)

print "Setting of password returned", results

granting = "GRANT ALL ON \*.\* TO '%s'@'%s'" %(mkuser, host)

results = cursor.execute(granting)

print "Granting of privileges returned", results

except MySQLdb.Error, e:

print e

If there is an error anywhere along the way, it is printed to screen. Otherwise,

the several print statements are executed. As long as they all return 0, each step

was successful.

[ 232 ]Chapter 8

Removing privileges in MySQL

To remove privileges that have been granted, one uses the REVOKE statement. One

uses the same information to revoke privileges as to grant them:

•The kinds of privileges to be revoked

•The database and table involved

•The user ID

•The hostname used in granting the privilege

As with dropping and creating a user, a pattern matching hostname of % does not

include localhost. That host must be revoked explicitly.

Basic syntax

The REVOKE command has the following basic syntax:

REVOKE <privileges> ON <database>.<table> FROM

'<userid>'@'<hostname>';

So to revoke all access for user tempo to the City table of the world database when

logged in locally, we would use the following statement:

REVOKE ALL PRIVILEGES ON world.City FROM 'tempo'@'localhost';

If we want to revoke only INSERT privileges for remote access, we would adapt the

preceding statement accordingly:

REVOKE INSERT ON world.City FROM 'tempo'@'%';

Again, it is important to remember that the following two lines affect different records in

the MySQL user table:

REVOKE ALL PRIVILEGES ON world.City FROM 'tempo'@'localhost';

REVOKE ALL PRIVILEGES ON world.City FROM 'tempo'@'%';

After using REVOKE, the user still has

access!?

All administrative changes in MySQL are applied to MySQL's internal databases.

Therefore, any change that is effected, only takes effect the next time MySQL needs

to read those administrative tables. Consequently, users can still have access to

databases or tables after the revocation statement has been issued.

[ 233 ]Creating Users and Granting Access

Often, administrative changes can wait until the user logs out. However, even then,

it can take a while for the changes to take effect. Depending on how frequently

MySQL reads the administrative tables, the change may not take effect even if

you manually remove the permissions from the administrative tables that govern

privileges (columns\_priv, procs\_priv, and tables\_priv). Within MySQL, one can

pass the following command:

FLUSH PRIVILEGES;

If one's login has RELOAD privileges.

If time is of the essence, however, and you want to force MySQL to re-read all of the

administrative tables, you may want to restart it. In Linux and other Unix variants,

execute the following with root privileges:

/etc/init.d/mysql restart

From Windows:

1. Click Start | Control Panel | Administrative Controls | Services.

2. Select mysql.

3. Right click then select Restart under Options.

Currently, there is no interface available to restart MySQL from Python

without issuing OS-specific commands (that is, using the os module).

This is not a tenable development strategy as the Python program

would need to run with administrator privileges (an obvious security

problem). However, Mats Kindahl, lead developer at MySQL, has started

MySQL Replicant, a project designed for replicating MySQL servers, but

that incidentally should include administrative tasks such as starting,

restarting, and stopping. For more, including a download link, see this

blog post from Kindahl's blog: http://mysqlmusings.blogspot.

com/2009/12/mysql-replicant-library-for-controlling.

html

[ 234 ]Chapter 8

Using REVOKE in Python

As with GRANT, revoking privileges in Python just depends on forming the statement.

As seen in this code, a revision of the earlier example, the REVOKE statement uses

similar context to and all the same information as the GRANT statement:

#!/usr/bin/env python

import MySQLdb

host = 'localhost'

user = 'skipper'

passwd = 'secret'

mydb = MySQLdb.connect(host, user, passwd)

cursor = mydb.cursor()

try:

mkuser = 'symphony'

creation = "CREATE USER '%s'@'%s'" %(mkuser, host)

results = cursor.execute(creation)

print "User creation returned", results

mkpass = 'n0n3wp4ss'

setpass = "SET PASSWORD FOR '%s'@'%s' = PASSWORD('%s')" %(mkuser,

host, mkpass)

results = cursor.execute(setpass)

print "Setting of password returned", results

granting = "GRANT ALL ON \*.\* TO '%s'@'%s'" %(mkuser, host)

results = cursor.execute(granting)

print "Granting of privileges returned", results

granting = "REVOKE ALL PRIVILEGES ON \*.\* FROM '%s'@'%s'" %(mkuser,

host)

results = cursor.execute(granting)

print "Revoking of privileges returned", results

except MySQLdb.Error, e:

print e

[ 235 ]Creating Users and Granting Access

Project: Web-based user administration

In this chapter's project, we will add some user administration facilities to the web

administration program that we created in the last chapter. We will therefore discuss

the changes to the program and will not address those functions we have already

discussed in the previous chapter.

As mentioned in the previous chapter, we should get the same results from this

program regardless of whether we call it through CGI or with PHP. The output is

always a HTML file. Using PHP has the advantage of allowing us to test the program

from the command-line where CGI requires hard-wiring of values in the code. It is

only after the program is proven locally that one should move it to a test server.

For reasons of illustration and portability, we will proceed in this project as if we

called the program through PHP. This allows us to list the new options in a way

that should be easier to follow.

New options in the code

The purpose of this project is to add certain user administration facilities to

the program from last chapter, PyMyAdmin.py. The functionality to be added

includes creating and dropping users as well as granting and revoking access

to proscribed accounts.

In addition to login credentials, the CREATE USER and DROP USER statements require

the declaration of a user's name. Therefore, we need to add the following to the

options supported:

opt.add\_option("-n", "--username",

action="store",

type="string",

help="username to be affected",

dest="username")

The account to be affected is therefore identified by opt.username.

Best practice suggests that one should set a user's password at the same time as

creating the account. Therefore, we need support for this:

opt.add\_option("-w", "--passwd",

action="store",

type="string",

help="password to be used in user creation",

dest="passwd")

[ 236 ]Chapter 8

Therefore, the affected account's password will then be contained in opt.passwd.

In granting and revoking privileges, we need three additional pieces of information:

the relevant privileges, the database, and tables to be used.

opt.add\_option("-r", "--privileges",

action="store",

type="string",

help="privileges to be assigned to user",

dest="privileges")

opt.add\_option("-a", "--acldb",

action="store",

type="string",

help="database to be affected with access rules",

dest="acldb")

opt.add\_option("-b", "--acltb",

action="store",

type="string",

help="table to be affected with access rules",

dest="acltb")

This data will thus reconcile to the following variables in the code:

•Privileges: opt.privileges

•Relevant database: opt.acldb

•Relevant tables: opt.acltb

Finally, we need a switch to indicate which of these the user wants to perform. For

this we use uact:

opt.add\_option("-u", "--uact",

action="store",

type="string",

help="act of user administration",

dest="uact")

This will naturally reconcile to opt.uact.

[ 237 ]Creating Users and Granting Access

At this point, the options listed for the program when the help menu is called looks

like this:

Usage: pymyadmin.py [options]

Options:

-h, --help

show this help message and exit

-U USER, --user=USER user account to use for login

-P PASSWORD, --password=PASSWORD

password to use for login

-d DBACT, --dbact=DBACT

kind of db action to be affected

-D DBNAME, --dbname=DBNAME

name of db to be affected

-t TBACT, --tbact=TBACT

kind of table action to be affected

-Q TBDBNAME, --tbdbact=TBDBNAME

name of database containing table to be

affected

-T TBNAME, --tbname=TBNAME

name of table to be affected

-q QACT, --qact=QACT kind of query to affect

-Z QDBNAME, --qdbname=QDBNAME

database to be used for query

-Y QTBNAME, --qtbname=QTBNAME

table to be used for query

-c COLUMNS, --columns=COLUMNS

columns to be used in query

-v VALUES, --values=VALUES

values to be used in query

-u UACT, --uact=UACT act of user administration

-n USERNAME, --username=USERNAME

username to be affected

-w PASSWD, --passwd=PASSWD

password to be used in user creation

-r PRIVILEGES, --privileges=PRIVILEGES

privileges to be assigned to user

-a ACLDB, --acldb=ACLDB

database to be affected with access rules

-b ACLTB, --acltb=ACLTB

table to be affected with access rules

[ 238 ]Chapter 8

Adding the functions: CREATE and DROP

As CREATE and DROP use the same basic data, it behooves us to use one function to

handle both.

def uaction(user, password, uact, username, \*passwd):

cursor = connectNoDB(user, password)

if uact == "create-user":

passwd = passwd[0]

create = "CREATE USER '%s'@'localhost'" %(username)

rescreate = execute(create, cursor, 'create-user')

setpass = "SET PASSWORD FOR '%s'@'localhost' = PASSWORD('%s')"

%(username, passwd)

respass = execute(setpass, cursor, 'set-pass')

results = rescreate + respass

else:

drop = "DROP USER '%s'@'localhost'" %(username)

resdrop = execute(drop, cursor, 'drop-user')

results = resdrop

return results

The main difference in the flow of the program is that we will expect to set a

password whenever the CREATE option is called. Therefore, we will fork the program

flow within the function according to whether we are creating or dropping a user.

When the CREATE option is used, this function expects a fifth value in \*passwd.

Otherwise, it is never used. The program will thus execute fine without it.

If you get errors in relation to the execution of this function, it is best to insert a

print statement to show what Python is passing to MySQL and, if necessary,

the variable types being used. An example of how to do this is as follows:

def uaction(user, password, uact, username, \*passwd):

cursor = connectNoDB(user, password)

if uact == "create-user":

create = "CREATE USER '%s'@'localhost'" %(username)

print "passwd is of type", type(passwd)

rescreate = execute(create, cursor, 'create-user')

setpass = "SET PASSWORD FOR '%s'@'localhost' = PASSWORD('%s')"

%(username, passwd)

respass = execute(setpass, cursor, 'set-pass')

results = rescreate + respass

else:

drop = "DROP USER '%s'@'localhost'" %(username)

resdrop = execute(drop, cursor, 'drop-user')

results = resdrop

return results

[ 239 ]Creating Users and Granting Access

If MySQL complains of problems in your statement, simply print it out before you

pass it to execute(). This will ensure that it is printed before the next program call,

where the statement is passed to MySQL and the problem arises. Alternatively, one

could edit the execute() function to print the statement it receives.

Here, we are also not handling the value of passwd correctly. While passwd is a

string in the main program, it is not passed as a whole. To find out how it is passed

and how to handle it correctly, we need to insert such statements as the second

print statement.

Adding CREATE and DROP to main()

Having created the functionality in uaction(), we can now handle the CREATE and

DROP options in main(). To do this, we simply add another elif clause to the series

that follows if authenticate equates to 1. The new if...elif series looks like this:

if authenticate == 1:

errmsg = "You have not specified the information necessary

for the action you chose. Please check your information and specify

it correctly in the dialogue."

if opt.dbact is not None:

output = dbaction(opt.dbact, opt.dbname, cursor)

elif opt.tbact is not None:

output = tbaction(opt.tbact, opt.tbdbname, opt.tbname,

opt.columns, opt.values, opt.user, opt.password)

elif opt.qact is not None:

output = qaction(opt.qact, opt.qdbname, opt.qtbname,

opt.columns, opt.values, opt.user, opt.password)

elif opt.uact is not None:

if opt.uact == "create":

act = "create-user"

output = uaction(opt.user, opt.password, act, opt.

username, opt.passwd)

elif opt.uact == "drop":

act = "drop-user"

output = uaction(opt.user, opt.password, act, opt.

username)

else:

output = errmsg

Note that we create a new variable act to reflect the import of opt.uact holding

a value and what that value is. This saves us from unnecessarily complicated

code elsewhere.

[ 240 ]Chapter 8

Adding the functions: GRANT and REVOKE

Next, we need to add the functionality for GRANT and REVOKE. For each of these

statements, the values involved are precisely the same. Therefore, we do not need

to use optional arguments. The function looks like this:

def uadmin(user, password, uact, username, privileges, acldb, acltb):

cursor = connectNoDB(user, password)

if uact == "grant":

grant = "GRANT %s ON %s.%s TO '%s'@'localhost'" %(privileges,

acldb, acltb, username)

results = execute(grant, cursor, 'grant')

else:

revoke = "REVOKE %s ON %s.%s FROM '%s'@'localhost'"

%(privileges, acldb, acltb, username)

results = execute(revoke, cursor, 'revoke')

return results

The syntax of each statement is straightforward. As we will remind the user on the

HTML page, the list of privileges should be comma delimited. If it is not, we will get

an error from MySQL. For real-world deployment, one would do well to check for

this or put the operational parts of the function into a try...else clause.

Adding GRANT and REVOKE to main()

Having added facilities to handle GRANT and REVOKE, we need to tell main() how to

handle those options. Once again, we are simply inserting another elif clause into

the previously mentioned series. The new main() function then looks like this:

def main():

"""The main function creates and controls the MySQLStatement

instance in accordance with the user's input."""

output = ""

while 1:

try:

cursor = connectNoDB(opt.user, opt.password)

authenticate = 1

except:

output = "Bad login information. Please verify the

username and password that you are using before trying to login

again."

authenticate = 0

[ 241 ]Creating Users and Granting Access

if authenticate == 1:

errmsg = "You have not specified the information necessary

for the action you chose. Please check your information and specify

it correctly in the dialogue."

if opt.dbact is not None:

output = dbaction(opt.dbact, opt.dbname, cursor)

elif opt.tbact is not None:

output = tbaction(opt.tbact, opt.tbdbname, opt.tbname,

opt.columns, opt.values, opt.user, opt.password)

elif opt.qact is not None:

output = qaction(opt.qact, opt.qdbname, opt.qtbname,

opt.columns, opt.values, opt.user, opt.password)

elif opt.uact is not None:

if opt.uact == "create":

act = "create-user"

output = uaction(opt.user, opt.password, act, opt.

username, opt.passwd)

elif opt.uact == "drop":

act = "drop-user"

output = uaction(opt.user, opt.password, act, opt.

username)

elif opt.uact == "grant" or opt.uact == "revoke":

output = uadmin(opt.user, opt.password, opt.uact,

opt.username, opt.privileges, opt.acldb, opt.acltb)

else:

output = errmsg

printout = HTMLPage()

printout.message(output)

output = printout.page()

print output

break

Note that, instead of using two elif options for each of GRANT and REVOKE, we

combine the two with a disjunctive or. All that is left is to add support for each

of the four options on the HTML page.

[ 242 ]Chapter 8

Test the program

Before adding support on the HTML page, however, it is good practice to test

what you have written. If you used the CGI method, you will need to hardwire

values into the code for testing. Using PHP is the same as calling the program from

the command line. Testing the program means trying all four new options and

validating them in MySQL and from a MySQL login. If the output is expected,

a valid HTML file, then the program executes as expected.

To test the program from the command-line, you will necessarily need to

access the terminal of your operating system and call the program with

flags. If you do not know how to do this, consult your operating system's

documentation.

To test user creation and dropping, open a MySQL shell as the root user. Using

the mysql database, you can verify CREATE and DROP statements against the user

database. The easiest way to do this is to select all from it by using:

SELECT \* FROM user;

Alternatively, nuance the query with a WHERE clause.

To test the granting and revocation of access in real terms, open a MySQL shell in the

name of the user to be affected. You as administrator, can then test the access granted

to that user.

SHOW GRANTS FOR <username>@<hostname>;

This will show you all available permissions for the relevant account.

If you are debugging your code and start receiving errors, remember to blackbox the

process. Don't simply try to edit the Python code and re-run it if you are getting a

MySQL error. First, use print commands to show what statement Python is handing

off to MySQL. Then ensure that those commands actually work in MySQL. Once you

are confident that you have the right MySQL command and syntax, you can look at

how your code passes information to MySQLdb. Finally, you can revisit your Python

code to ensure that it is working as planned.

Remember that optional arguments (for example, \*passwd) are passed as tuples.

So, even if it is a string in the main function, it becomes a tuple when passed. You,

therefore, have to handle it appropriately. Once you are satisfied that the program

will behave as intended, it is time to implement the options in the HTML form.

[ 243 ]Creating Users and Granting Access

New options on the page

To avail these new options to the user, we obviously need to adapt the HTML form.

Insert the following just before the closing </form> tag:

<div>USER ADMINISTRATION</div>

<input type="radio" name="uact" value="create"> CREATE <br>

<input type="radio" name="uact" value="drop"> DROP <br>

<input type="radio" name="uact" value="grant"> GRANT <br>

<input type="radio" name="uact" value="revoke"> REVOKE <br>

User name: <input type="text" name="username" value=""> <br>

Password: <input type="password" name="passwd" value=""> <br>

Privileges (comma-separated): <input type="text"

name="privileges"value=""> <br>

Database and Table:

<input type="text" name="acldb" value="">.

<input type="text" name="acltb" value=""> <br>

Note that, as a matter of good practice, one should not implement options in the

form that are not yet implemented and tested in the code. To do so is a security risk.

The relevant part of the HTML page thus looks like this when rendered in a browser:

Room to grow

Where the above implementation works, several limitations exist in it. One of the

primary ways that an error can arise is if the user does not enter the privileges

separated by commas but by, say, semi-colons. Additional functionality that can be

added includes:

•Allowing user administration for non-local hostnames

•Validating the database and table names before passing them to execute()

•Supporting SHOW GRANTS in order to provide a meaningful error message if a

GRANT or REVOKE statement fails

[ 244 ]Chapter 8

We will look at other ways to augment the user-friendliness of the HTML menu in

later chapters. But can you think of ways to make it easier to use?

Summary

In this chapter, we have covered how to create and remove users and privileges with

MySQL for Python. We have seen:

•How to use MySQL and MySQLdb to CREATE and DROP users

•How to set user passwords

•How we can manage database privileges with MySQL for Python

•Ways to automate user creation and removal

•How to GRANT and REVOKE privileges and the conditions under which that

can be done

In the next chapter, we will look at working with dates and time in MySQL

for Python.

[ 245 ]Date a