MySQL\_for\_Python\_Albert\_c01

Preface

Python is a dynamic programming language, which is completely enterprise ready,

owing largely to the variety of support modules that are available to extend its

capabilities. In order to build productive and feature-rich Python applications, we

need to use MySQL for Python, a module that provides database support to

our applications.

This book demonstrates how to boost the productivity of your Python applications

by integrating them with the MySQL database server, the world's most powerful

open source database. It will teach you to access the data on your MySQL database

server easily with Python's library for MySQL using a practical, hands-on approach.

Leaving theory to the classroom, this book uses real-world code to solve real-world

problems with real-world solutions.

The book starts by exploring the various means of installing MySQL for Python

on different platforms and how to use simple database querying techniques to

improve your programs. It then takes you through data insertion, data retrieval,

and error-handling techniques to create robust programs. The book also covers

automation of both database and user creation, and administration of access

controls. As the book progresses, you will learn to use many more advanced

features of Python for MySQL that facilitate effective administration of your

database through Python. Every chapter is illustrated with a project that you

can deploy in your own situation.

By the end of this book, you will know several techniques for interfacing

your Python applications with MySQL effectively so that powerful database

management through Python becomes easy to achieve and easy to maintain.

**What this book covers**

Chapter 1, Getting Up and Running with MySQL for Python, helps you to install MySQL

for Python specific software, how to import modules into your programs, connecting

to a database, accessing online help, and creating a MySQL cursor proxy within your

Python program. It also covers how to close the database connection from Python

and how to access multiple databases within one program.

Chapter 2, Simple Querying, helps you to form and pass a query to MySQL, to look at

user-defined variables, how to determine characteristics of a database and its tables,

and program a command-line search utility. It also looks at how to change queries

dynamically, without user input.

Chapter 3, Simple Insertion, shows forming and passing an insertion to MySQL, to

look at the user-defined variables in a MySQL insertion, passing metadata between

databases, and changing insertion statements dynamically without user input.

Chapter 4, Exception Handling, discusses ways to handle errors and warnings that

are passed from MySQL for Python and the differences between them. It also

covers several types of errors supported by MySQL for Python, and how to

handle them effectively.

Chapter 5, Results Record-by-Record, shows situations in which record-by-record

retrieval is desirable, to use iteration to retrieve sets of records in smaller blocks

and how to create iterators and generators in Python. It also helps you in using

fetchone() and fetchmany().

Chapter 6, Inserting Multiple Entries, discusses how iteration can help us execute

several individual INSERT statements rapidly, when to use or avoid executemany(),

and throttling how much data is inserted at a time.

Chapter 7, Creating and Dropping, shows to create and delete both databases and tables

in MySQL, to manage database instances with MySQL for Python, and to automate

database and table creation.

Chapter 8, Creating Users and Granting Access, focuses on creating and removing users

in MySQL, managing database privileges with MySQL for Python, automating user

creation and removal, to GRANT and REVOKE privileges, and the conditions under

which that can be done.

Chapter 9, Date and Time Values, discusses what data types MySQL supports for date

and time, when to use which data type and in what format and range, and frequently

used functions for handling matters of date and time.

Chapter 10, Aggregate Functions and Clauses, shows how MySQL saves us time and

effort by pre-processing data, how to perform several calculations using MySQL's

optimized algorithms, and to group and order returned data by column.

Chapter 11, SELECT Alternatives, discusses how to use HAVING clauses, how to

create temporary subtables, subqueries and joins in Python, and the various ways

to join tables.

Chapter 12, String Functions, shows how MySQL allows us to combine strings and

return the single, resulting value, how to extract part of a string or the location of a

part, thus saving on processing, and how to convert cases of results.

Chapter 13, Showing MySQL Metadata, discusses the several pieces of metadata about

a given table that we can access, which system variables we can retrieve, and how to

retrieve user privileges and the grants used to give them.

Chapter 14, Disaster Recovery, focuses on when to implement one of several kinds

of database backup plans, what methods of backup and disaster recovery MySQL

supports, and how to use Python to back up databases

**c01 Getting Up and Running withMySQL for Python**

It may seem rather unnecessary to start a book on MySQL for Python with a chapter on setting it up. There are, in fact, several ways to get MySQL for Python in a place such that your local Python installation can use it. Which one you use will depend as much on your familiarity with your operating system and with Python itself, as it will on which operating system and version of Python you are running.

Puede parecer innecesario comenzar un libro sobre MySQL para Python con un capítulo sobre cómo configurarlo. De hecho, existen varias formas de obtener MySQL para Python en un lugar donde su instalación local de Python pueda usarlo. Cuál utilice dependerá tanto de su familiaridad con su sistema operativo y con el propio Python, como del sistema operativo y la versión de Python que esté ejecutando.

In this chapter we will cover the following:

•Where you can get MySQL for Python

•Installing MySQL for Python

•Importing the module into your programs

•Accessing online help about the MySQL for Python API and its accompanying modules

•How to connect to a database

•How to create a MySQL cursor proxy within your Python program

•How to close the database connection from Python

•How to access multiple databases within one program

**Getting MySQL for Python**

How you get MySQL for Python depends on your operating system and the level of authorization you have on it. In the following subsections, we walk through the common operating systems and see how to get MySQL for Python on each.

La forma de obtener MySQL para Python depende de su sistema operativo y del nivel de autorización que tenga. En las siguientes subsecciones, analizamos los sistemas operativos comunes y vemos cómo obtener MySQL para Python en cada uno.

**Using a package manager (only on Linux)**

Package managers are used regularly on Linux, but none come by default with Macintosh and Windows installations. So users of those systems can skip this section.

Los administradores de paquetes se utilizan regularmente en Linux, pero ninguno viene de forma predeterminada con las instalaciones de Macintosh y Windows. Entonces los usuarios de esos sistemas pueden saltarse esta sección.

A package manager takes care of downloading, unpacking, installing, and configuring new software for you. In order to use one to install software on your Linux installation, you will need administrative privileges.

Un administrador de paquetes se encarga de descargar, descomprimir, instalar y configurar el nuevo software por usted. Para utilizar uno para instalar software en su instalación de Linux, necesitará privilegios administrativos.

Administrative privileges on a Linux system can be obtained legitimately in one of the following three ways:

Los privilegios administrativos en un sistema Linux se pueden obtener legítimamente de una de las tres formas siguientes:

•Log into the system as the root user (not recommended)

•Switch user to the root user using su

•Use sudo to execute a single command as the root user

The first two require knowledge of the root user's password. Logging into a system directly as the root user is not recommended due to the fact that there is no indication in the system logs as to who used the root account. Logging in as a normal user and then switching to root using su is better because it keeps an account of who did what on the machine and when. Either way, if you access the root account, you must be very careful because small mistakes can have major consequences. Unlike other operating systems, Linux assumes that you know what you are doing if you access the root account and will not stop you from going so far as deleting every file on the hard drive.

Los dos primeros requieren conocimiento de la contraseña del usuario root. No se recomienda iniciar sesión en un sistema directamente como usuario root debido a que no hay ninguna indicación en los registros del sistema sobre quién usó la cuenta root. Iniciar sesión como usuario normal y luego cambiar a root usando su es mejor porque mantiene una cuenta de quién hizo qué en la máquina y cuándo. De cualquier manera, si accedes a la cuenta root debes tener mucho cuidado porque pequeños errores pueden tener grandes consecuencias. A diferencia de otros sistemas operativos, Linux asume que usted sabe lo que está haciendo si accede a la cuenta raíz y no le impedirá llegar al extremo de eliminar todos los archivos del disco duro.

Unless you are familiar with Linux system administration, it is far better, safer, and more secure to prefix the sudo command to the package manager call. This will give you the benefit of restricting use of administrator-level authority to a single command. The chances of catastrophic mistakes are therefore mitigated to a great degree.

menos que esté familiarizado con la administración del sistema Linux, es mucho mejor y más seguro anteponer el comando sudo a la llamada del administrador de paquetes. Esto le brindará la ventaja de restringir el uso de la autoridad de nivel de administrador a un solo comando. De este modo se mitigan en gran medida las posibilidades de cometer errores catastróficos.

More information on any of these commands is available by prefacing either man or info before any of the preceding commands (su, sudo).

Puede obtener más información sobre cualquiera de estos comandos anteponiendo man o info antes de cualquiera de los comandos anteriores (su, sudo).

Which package manager you use depends on which of the two mainstream package management systems your distribution uses. Users of RedHat or Fedora, SUSE, or Mandriva will use the RPM Package Manager (RPM) system. Users of Debian, Ubuntu, and other Debian-derivatives will use the apt suite of tools available for Debian installations. Each package is discussed in the following:

El administrador de paquetes que utilice depende de cuál de los dos sistemas de administración de paquetes principales utilice su distribución. Los usuarios de RedHat o Fedora, SUSE o Mandriva utilizarán el sistema RPM Package Manager (RPM). Los usuarios de Debian, Ubuntu y otros derivados de Debian utilizarán el conjunto de herramientas adecuado disponible para las instalaciones de Debian. Cada paquete se analiza a continuación:

**Using RPMs and yum**

If you use SUSE, RedHat, or Fedora, the operating system comes with the yum package manager. You can see if MySQLdb is known to the system by running a search (here using sudo):

sudo yum search mysqldb

If yum returns a hit, you can then install MySQL for Python with the following command:

sudo yum install mysqldb

**Using RPMs and urpm**

If you use Mandriva, you will need to use the urpm package manager in a similar fashion. To search use urpmq:

sudo urpmq mysqldb

And to install use urpmi:

sudo urpmi mysqldb

**Using apt tools on Debian-like systems**

Whether you run a version of Ubuntu, Xandros, or Debian, you will have access to aptitude, the default Debian package manager. Using sudo we can search for MySQLdb in the apt sources using the following command:

Ya sea que ejecute una versión de Ubuntu, Xandros o Debian, tendrá acceso a aptitude, el administrador de paquetes predeterminado de Debian. Usando sudo podemos buscar MySQLdb en las fuentes apt usando el siguiente comando:

sudo aptitude search mysqldb

On most Debian-based distributions, MySQL for Python is listed as python-mysqldb.

En la mayoría de las distribuciones basadas en Debian, MySQL para Python aparece como python-mysqldb.

Once you have found how apt references MySQL for Python, you can install it using the following code:

Una vez que haya descubierto cómo apt hace referencia a MySQL para Python, puede instalarlo usando el siguiente código:

sudo aptitude install python-mysqldb

Using a package manager automates the entire process so you can move to the section Importing MySQL for Python.

El uso de un administrador de paquetes automatiza todo el proceso, por lo que puede pasar a la sección Importación de MySQL para Python.

**Using an installer for Windows**

Windows users will need to use the older 1.2.2 version of MySQL for Python. Using a web browser, go to the following link:

http://sourceforge.net/projects/mysql-python/files/

This page offers a listing of all available files for all platforms. At the end of the file listing, find mysql-python and click on it. The listing will unfold to show folders containing versions of MySQL for Python back to 0.9.1. The version we want is 1.2.2.

Windows binaries do not currently exist for the 1.2.3 version of MySQL for Python. To get them, you would need to install a C compiler on your Windows installation and compile the binary from source.

This is outside the purpose of the present book, but tips for how to do this are contained in the README file that accompanies the 1.2.3 version.

Click on 1.2.2 and unfold the file listing. As you will see, the Windows binaries are differentiated by Python version—both 2.4 and 2.5 are supported. Choose the one that matches your Python installation and download it. Note that all available binaries are for 32-bit Windows installations, not 64-bit.

After downloading the binary, installation is a simple matter of double-clicking the installation EXE file and following the dialogue. Once the installation is complete, the module is ready for use. So go to the section Importing MySQL for Python.

**Using an egg file**

One of the easiest ways to obtain MySQL for Python is as an egg file, and it is best to use one of those files if you can. Several advantages can be gained from working with egg files such as:

•They can include metadata about the package, including its dependencies

•They allow for the use of egg-aware software, a helpful level of abstraction

•Eggs can, technically, be placed on the Python executable path and used without unpacking

•They save the user from installing packages for which they do not have the appropriate version of software

•They are so portable that they can be used to extend the functionality of third-party applications

**Installing egg handling software**

One of the best known egg utilities—Easy Install, is available from the PEAK Developers' Center at <http://peak.telecommunity.com/DevCenter/EasyInstall>. How you install it depends on your operating system and whether you have package management software available. In the following section, we look at several ways to install Easy Install on the most common systems.

**Using a package manager (Linux)**

On Ubuntu you can try the following to install the easy\_install tool (if not available already):

En Ubuntu puedes probar lo siguiente para instalar la herramienta easy\_install (si aún no está disponible):

shell> sudo aptitude install python-setuptools

On RedHat or CentOS you can try using the yum package manager:

shell> sudo yum install python-setuptools

On Mandriva use urpmi:

shell> sudo urpmi python-setuptools

You must have administrator privileges to do the installations just mentioned.

**Without a package manager (Mac, Linux)**

If you do not have access to a Linux package manager, but nonetheless have a Unix variant as your operating system (for example, Mac OS X), you can install Python's setuptools manually. Go to:

http://pypi.python.org/pypi/setuptools#files

Download the relevant egg file for your Python version.

When the file is downloaded, open a terminal and change to the download directory. From there you can run the egg file as a shell script. For Python 2.5, the command would look like this:

sh setuptools-0.6c11-py2.5.egg

This will install several files, but the most important one for our purposes is easy\_install, usually located in /usr/bin.

**On Microsoft Windows**

On Windows, one can download the setuptools suite from the following URL:

<http://pypi.python.org/pypi/setuptools#files>

From the list located there, select the most appropriate Windows executable file.

Once the download is completed, double-click the installation file and proceed through the dialogue. The installation process will set up several programs, but the one important for our purposes is easy\_install.exe. Where this is located will differ by installation and may require using the search function from the Start Menu.

On 64-bit Windows, for example, it may be in the Program Files (x86) directory. If in doubt, do a search. On Windows XP with Python 2.5, it is located here:

C:\Python25\Scripts\easy\_install.exe

Note that you may need administrator privileges to perform this installation. Otherwise, you will need to install the software for your own use. Depending on the setup of your system, this may not always work.

Installing software on Windows for your own use requires the following steps:

1Copy the setuptools installation file to your Desktop.

2Right-click on it and choose the runas option.

3Enter the name of the user who has enough rights to install it

(presumably yourself).

After the software has been installed, ensure that you know the location of the easy\_install.exe file. You will need it to install MySQL for Python.

**Installing MySQL for Python from an egg file**

After installing EasyInstall, you still need to install the MySQL for Python egg. The egg files for MySQL for Python can be downloaded from the following URL:

http://sourceforge.net/projects/mysql-python/files/

There you will see a list of all available files relevant to MySQL for Python. Which one you use depends on your operating system and your installed Python version. Currently, the only egg files available for MySQL for Python version 1.2.3c1 are for Linux running either Python 2.5 or 2.6. Mac users should use a tarball (tar. gz) file as discussed in the next section.

To get an egg file for Windows, click on the MySQL-python directory and select the 1.2.2 version. This is the same directory used for the Windows binaries discussed earlier in this chapter. This time, however, you need to select an egg for Windows that fits either Python 2.4 or 2.5. There is no 2.6 version.

Once you have the egg file for MySQL for Python, you simply need to invoke EasyInstall over the newly-downloaded egg file. How you do that will depend on the permissions you have for your operating system.

With administrator permissions, you can simply call the EasyInstall binary. For Linux, it will look like this:

shell> easy\_install <name of egg file>

For Windows, you will use a command similar to this one:

C:\Python25\Scripts\easy\_install.exe <name of egg file>

Note that you must have administrator privileges to do this. Otherwise, Windows users will have to install the software locally. Linux users can use sudo.

EasyInstall will then unpack the archive, install it in your default Python installation folders, and configure it for immediate use.

For Windows users, if you had to install setuptools locally, you may also require a local installation of Python itself in order to install MySQL for Python. See the section On Microsoft Windows under Installing egg-handling software, for help with this. If you need to go through this process, all of your configurations will be local, so you are best to use full path command-line calls.

If your system has MySQL, Python, and setuptools, but you still don't have administrative access, it is advisable to unpack the egg file manually and call it as a local module. To do this, use an archiving program to unzip the file.

The content listing for the Windows egg will look like this:

•Egg-info

•MySQLdb

•\_mysql\_exceptions.py

•\_mysql\_exceptions.pyc

•\_mysql.py

•\_mysql.pyc

•\_mysql.pyd

And the Linux egg unpacks to the following files:

•Egg-info

•MySQLdb

•\_mysql\_exceptions.py

•\_mysql\_exceptions.pyc

•\_mysql.py

•\_mysql.pyc

•\_mysql.so

With the exception of the egg-info directory, the contents are the basic ingredients of a Python module and can be imported locally if one's program resides in the same directory as the files are located.

**Using a tarball (tar.gz file)**

Due to the need for certain programming libraries, this method of installation applies only to users of Unix-derived operating systems. This method involves installing from the source files and so requires the necessary C libraries to compile a binary version. Windows users should therefore use one of the other methods discussed previously.

If you cannot use egg files or if you use an earlier version of Python, you should use the tar.gz file, a tar and gzip archive. The tar.gz archive follows the Linux egg files in the file listing. The current version of MySQL for Python is 1.2.3c1, so the file we want is as following:

MySQL-python-1.2.3c1.tar.gz

This method is by far more complicated than the others. If at all possible, use your operating system's installation method or an egg file.

This version of MySQL for Python is compatible up to Python 2.6. It is worth noting that MySQL for Python has not yet been released for Python 3.0 or later versions. In your deployment of the library, therefore, ensure that you are running Python 2.6 or earlier. As noted, Python 2.5 and 2.6 have version-specific releases. Prior to Python 2.4, you will need to use either a tar.gz version of the latest release or use an older version of MySQL for Python. The latter option is not recommended.

Most Unix-derived operating systems (Linux, Mac) come with the tar and gzip utilities pre-installed. For users of these systems, unpacking the archive is as simple as the following command:

shell> tar xvzf MySQL-python-1.2.3c1.tar.gz

The archive will then unpack into a directory called MySQL-python-1.2.3c1.

Windows users can use any of the following archive programs to unpack the tarball:

•PowerArchiver 6.1

•7-Zip

•WinZip

Once the file is unpacked, you need to ensure that you have the program mysql\_config in your path. For Mac users, this usually comes with the MySQL installation itself. For Linux, if you are using bash or another shell with command-line completion, you can check this by typing the following in a terminal:

shell> mysql\_conf

Then press the tab key. If the command is completed to mysql\_config, there are no issues, otherwise your operating system does not know of any such command, and you need to either find it or install it.

An alternative way of checking is to use the whereis command. Type the following from the command-line:

shell> whereis mysql\_config

If it is installed, the system will return its location. Then echo your current PATH value by typing:

shell> echo $PATH

and compare the results. If the location of mysql\_config is one of the values in your path, there are no issues otherwise, we need to either find it or install it.

The mysql\_config program comes with the MySQL client development libraries. If you have these installed, check the directory that holds the MySQL client binary (use whereis mysql if necessary). If you are unsure, you can check with a package manager using the following commands:

shell> aptitude search mysql | grep client | grep dev

This will work for Debian-based systems. Users of RPM-based systems should substitute either yum search or urpmq for aptitude search. This query will return results for the development files and for the MySQL client, and you can then see if the appropriate package is installed. If it is not, you can install it with the install argument (for either aptitude or yum) or by using urpmi.

If the mysql\_config program is installed, but is outside your path, you need to indicate its location to the MySQL for Python setup configuration. Navigate to the MySQL-python-1.2.3c1 directory and open the file site.cfg in your favorite text editor. The file is not large, and the following section is easily seen as the second part of the file:

#The path to mysql\_config

#Only use this if mysql\_config is not on your PATH,

or you have some weird setup that requires it

#mysql\_config = /usr/local/bin/mysql\_config

If mysql\_config is outside of your path, uncomment the last line of the part cited here and enter the correct path. So, if mysql\_config is installed to:

/usr/local/bin/mysql/bin/mysql\_config

The last line should read:

mysql\_config = /usr/local/bin/mysql/bin/mysql\_config

Then save the file and close it.

Next, we should build the package using the instructions that came with it in setup.py. Use the following command to attempt a build without installing it:

shell> python setup.py build

If the process goes through without error, which it usually does, the build is successful. If there is an error, it usually involves the lack of a module or software package. In which case, confirm that you have all the prerequisites needed for the task by checking the list in the readme file that comes with the archive.

Be sure to read the readme file that comes with the source code. It contains a lot of help on the installation process.

Once the build is successful, installation can be done with the following command:

shell> python setup.py install

Note that you will need super user access for this. If you do not have administrative access to your system, you need to use one of the other methods.

**Importing MySQL for Python**

The name of the project MySQL for Python is the current version of a project that began under the rubric MySQLdb. Consequently, unlike most Python modules, the MySQL for Python module is not called by its name, but by its historic handle. To import the module, insert the following into a Python program or simply type it in a following Python shell:

**import MySQLdb**

To make working with the module easier, you can also import it with an alias:

**import MySQLdb as mysql**

This allows us to use mysql instead of MySQLdb when we access parts of the module.

When you do this, several things will occur. You need not be concerned about most of them, but you should be aware that MySQLdb depends upon a module called \_mysql. The \_mysql module is largely a Python adaptation of the MySQL C API.

Cuando hagas esto, sucederán varias cosas. No necesita preocuparse por la mayoría de ellos, pero debe tener en cuenta que MySQLdb depende de un módulo llamado \_mysql. El módulo \_mysql es en gran medida una adaptación de Python de la API MySQL C.

This is important to note because it is this API that you will access through MySQL for Python.

Es importante tener esto en cuenta porque es esta API a la que accederá a través de MySQL para Python.

​

MySQL for Python is a wrapper for accessing the \_mysql API. A wrapper is essentially a system of macros, or trusted code, that allows you to do common tasks quickly. It allows you to program without having to repeat commonly used or accessed variables and functions. The \_mysql module is a powerful and proven way of accessing a MySQL database. However, controlling it within a Python program can pose a challenge for some, like driving a Formula 1 car for the first time. So consider MySQL for Python as a system that allows you to harness the power of a Formula 1 racing car even if you're merely driving a Hyundai.

MySQL para Python es un contenedor para acceder a la API \_mysql. Un contenedor es esencialmente un sistema de macros, o código confiable, que le permite realizar tareas comunes rápidamente. Le permite programar sin tener que repetir variables y funciones comúnmente utilizadas o a las que se accede. El módulo \_mysql es una forma potente y comprobada de acceder a una base de datos MySQL. Sin embargo, controlarlo dentro de un programa Python puede suponer un desafío para algunos, como conducir un coche de Fórmula 1 por primera vez. Así que considere MySQL para Python como un sistema que le permite aprovechar la potencia de un coche de carreras de Fórmula 1 incluso si simplemente conduce un Hyundai.

Unlike some systems of macros, MySQL for Python still allows you to access the classes and functions of \_mysql. This is due to the nature of Python's import functionality.

A diferencia de algunos sistemas de macros, MySQL para Python aún le permite acceder a las clases y funciones de \_mysql. Esto se debe a la naturaleza de la funcionalidad de importación de Python.

**Accessing online help when you need it**

As with other modules, Python is able to provide online help about MySQL for Python. In the following sections, we look at the MySQLdb and \_mysql modules in greater depth using Python's built-in help() function.

**MySQLdb**

After importing MySQLdb, you can read over the documentation that accompanies the module. In a Python shell, type:

**help(MySQLdb)**

You will then see a manual page detailing all of the functions and classes of MySQL for Python. It is well worth giving this a cursory read to familiarize yourself with the module. In the course of this book, we will cover most of these items from various angles.

Luego verá una página de manual que detalla todas las funciones y clases de MySQL para Python. Vale la pena leerlo brevemente para familiarizarse con el módulo. A lo largo de este libro, cubriremos la mayoría de estos temas desde varios ángulos.

As the help page indicates, MySQLdb includes the following modules:

* **connections**: Initiating, maintaining, and closing a connection to MySQL
* **cursors**: Managing the execution of queries
* **converters**: For converting between MySQL data types as well as between data types in MySQL and Python
* **times**: Converting date and time values between MySQL and Python

Each of these is abstracted to the point of its own module in the source tree. Without a doubt, the most important part of the module is connections.py, without which we could not interface with MySQL. Where the others are static, the conversion module, convertors.py, allows you to define your own convertor on-the-fly.

Cada uno de estos se abstrae hasta el punto de su propio módulo en el árbol de origen. Sin duda, la parte más importante del módulo es conexiones.py, sin el cual no podríamos interactuar con MySQL. Mientras que los demás son estáticos, el módulo de conversión, convertors.py, le permite definir su propio convertidor sobre la marcha.

The MySQLdb module itself has only one operating class that does not pertain to errors—DBAPISet. This is MySQLdb's internal object class for processing data. To interface with MySQL, however, we use functions. Of the several listed at the end of the MySQLdb help page, one uses connect() in every MySQLdb program.

El módulo MySQLdb en sí tiene sólo una clase operativa que no pertenece a errores: DBAPISet. Esta es la clase de objeto interno de MySQLdb para procesar datos. Sin embargo, para interactuar con MySQL utilizamos funciones. De los varios enumerados al final de la página de ayuda de MySQLdb, uno usa connect() en cada programa MySQLdb.

At first glance, it may here be confusing to see that MySQLdb seems to have three ways of connecting with a database. In the list of functions, these are as follows:

A primera vista, puede resultar confuso ver que MySQLdb parece tener tres formas de conectarse con una base de datos. En la lista de funciones, estas son las siguientes:

* connect()
* Connection
* Connect

Knowing the ins and outs of these functions is not necessary. It is, however, important to know that they exist and to recognize that the latter two are simply different ways of transferring data to the first. Connect() then passes the arguments to the connections. Connection() class, MySQLdb's MySQL database connection class, in the connections.py module.

No es necesario conocer los entresijos de estas funciones. Sin embargo, es importante saber que existen y reconocer que los dos últimos son simplemente formas diferentes de transferir datos al primero. Connect() luego pasa los argumentos a las conexiones. Clase Connection(), clase de conexión de base de datos MySQL de MySQLdb, en el módulo links.py.

\_mysql

In looking over the module, you may also note that reference is made to the \_mysql module, but it is not explicitly detailed. This is because it is a dependency and not part of the module itself. However, you can access the documentation for \_mysql without importing it directly by using the MySQLdb namespace:

Al revisar el módulo, también podrá notar que se hace referencia al módulo \_mysql, pero no se detalla explícitamente. Esto se debe a que es una dependencia y no parte del módulo en sí. Sin embargo, puedes acceder a la documentación de \_mysql sin importarla directamente utilizando el espacio de nombres MySQLdb:

**help(MySQLdb.\_mysql)**

In the previous discussion about connections.Connection(), we stopped following the trail of the connection and any ensuing data transmission where MySQLdb stopped. In reality, however, the data does not stop there. When a connection or operational request is received by connections.Connection(), it is processed and passed to \_mysql and subsequently to the MySQL API in C to perform it.

En la discusión anterior sobre conexiones.Connection(), dejamos de seguir el rastro de la conexión y cualquier transmisión de datos posterior donde MySQLdb se detuvo. En realidad, sin embargo, los datos no se detienen ahí. Cuando Connections.Connection() recibe una conexión o solicitud operativa, se procesa y se pasa a \_mysql y posteriormente a la API de MySQL en C para realizarla.

To handle this interface, \_mysql uses two classes:

* connection
* result

The first is used to establish communication with MySQL and thus returns a connection object. The second, as the name implies, returns a set containing the results from a MySQL command that a program sends. These results can be either the query results or an error. \_mysql naturally passes the error to the calling process. In the case of MySQLdb, we then have a comprehensive toolbox to handle the errors that may arise.

El primero se utiliza para establecer comunicación con MySQL y así devuelve un objeto de conexión. El segundo, como su nombre lo indica, devuelve un conjunto que contiene los resultados de un comando MySQL que envía un programa. Estos resultados pueden ser los resultados de la consulta o un error. \_mysql naturalmente pasa el error al proceso de llamada. En el caso de MySQLdb, tenemos una caja de herramientas completa para manejar los errores que puedan surgir.

**Connecting with a database**

In making a phone call, one picks up the handset, dials a number, talks and listens, and then hangs up. Making a database connection through MySQL for Python is nearly as simple. The four stages of database communication in Python are as follows:

Al realizar una llamada telefónica, uno levanta el auricular, marca un número, habla, escucha y luego cuelga. Realizar una conexión a una base de datos a través de MySQL para Python es casi igual de simple. Las cuatro etapas de la comunicación de una base de datos en Python son las siguientes:

* Creating a connection object
* Creating a cursor object
* Interacting with the database
* Closing the connection

**Creating a connection object**

As mentioned previously, we use connect() to create an object for the program's connection to the database. This process automates logging into the database and selecting a database to be used.

Como se mencionó anteriormente, usamos connect() para crear un objeto para la conexión del programa a la base de datos. Este proceso automatiza el inicio de sesión en la base de datos y la selección de una base de datos que se utilizará.

The syntax for calling the connect() function and assigning the results to a variable is as follows:

**[variable] = MySQLdb.connect([hostname], [username], [password],**

**[database name])**

Naming these variables as you assign the values is not required, but it is good practice until you get used to the format of the function call. So for the first few chapters of this book, we will use the following format to call the

No es necesario nombrar estas variables a medida que asigna los valores, pero es una buena práctica hasta que se acostumbre al formato de la llamada a la función. Entonces, durante los primeros capítulos de este libro, usaremos el siguiente formato para llamar al

**connect() function:**

**[variable] = MySQLdb.connect(host="[hostname]",**

**user="[username]",**

**passwd="[password]",**

**db="[database name]")**

Let's say we have a database-driven application that creates the menu for a seafood restaurant. We need to query all of the fish from the menu database in order to input them into a new menu. The database is named menu.

Digamos que tenemos una aplicación basada en una base de datos que crea el menú de un restaurante de mariscos. Necesitamos consultar todos los pescados de la base de datos del menú para poder ingresarlos en un nuevo menú. La base de datos se llama menú.

If you do not have a database called **menu**, you will obviously not be able to connect to it with these examples. To create the database that we are using in this example, put the following code into a text file with the name **menu.sql**:

Si no tienes una base de datos llamada menú, obviamente no podrás conectarte a ella con estos ejemplos. Para crear la base de datos que estamos usando en este ejemplo, coloque el siguiente código en un archivo de texto con el nombre menu.sql:

CREATE DATABASE `menu`;

USE menu;

DROP TABLE IF EXISTS `fish`;

SET @saved\_cs\_client = @@character\_set\_client;

SET character\_set\_client = utf8;

CREATE TABLE `fish` (

`ID` int(11) NOT NULL auto\_increment,

`NAME` varchar(30) NOT NULL default ‘’,

`PRICE` decimal(5,2) NOT NULL default ‘0.00’,

PRIMARY KEY (`ID`)

) ENGINE=MyISAM AUTO\_INCREMENT=27 DEFAULT

CHARSET=latin1;

SET character\_set\_client = @saved\_cs\_client;

LOCK TABLES `fish` WRITE;

INSERT INTO `fish` VALUES (1,’catfish’,’8.50’),(2,’catf

ish’,’8.50’),(3,’tuna’,’8.00’),(4,’catfish’,’5.00’),(5

,’bass’,’6.75’),(6,’haddock’,’6.50’),(7,’salmon’,’9.50

’),(8,’trout’,’6.00’),(9,’tuna’,’7.50’),(10,’yellowfin

tuna’,’12.00’),(11,’yellowfin tuna’,’13.00’),(12,’tuna’

,’7.50’);

UNLOCK TABLES;

Then log into your MySQL session from the directory in which the file menu.sql is located and type the following:

Luego inicie sesión en su sesión MySQL desde el directorio en el que se encuentra el archivo menu.sql y escriba lo siguiente:

**source menu.sql**

This will cause MySQL to create and populate our example database.

For this example, the database and program reside on the same host, so we can use localhost. The user for the database is skipper with password mysecret. After importing the MySQL for Python module, we would call the connect() function as follows:

Para este ejemplo, la base de datos y el programa residen en el mismo host, por lo que podemos usar **localhost**. El usuario de la base de datos es **skipper** con contraseña **mysecret**. Después de importar el módulo MySQL para Python, llamaríamos a la función connect() de la siguiente manera:

**mydb = MySQLdb.connect(host="localhost",**

**user="skipper",**

**passwd="mysecret",**

**db="menu")**

The **connect()** function acts as a foil for the connection class in **connections.py** and returns an object to the calling process. So in this example, assigning the value of MySQLdb.connect() to mydb renders mydb as a connection object. To illustrate this, you can create the necessary database in MySQL, connect to it as shown previously, then **type help(mydb)** at the Python shell prompt. You will then be presented with large amounts of information pertinent to MySQLdb.connections objects.

La función **connect**() actúa como complemento para la clase de conexión en links.py y devuelve un objeto al proceso de llamada. Entonces, en este ejemplo, asignar el valor de MySQLdb.connect() a mydb representa a mydb como un objeto de conexión. Para ilustrar esto, puede crear la base de datos necesaria en MySQL, conectarse a ella como se mostró anteriormente y luego escribir ayuda (mydb) en el símbolo del shell de Python. Luego se le presentarán grandes cantidades de información pertinente a los objetos MySQLdb.connections.

**Creating a cursor object**

After the connection object is created, you cannot interact with the database until you create a **cursor object**. The name cursor belies the purpose of this object. Cursors exist in any productivity application and have been a part of computing since the beginning. The point of a cursor is to mark your place and to allow you to issue commands to the computer. A cursor in MySQL for Python serves as a Python-based proxy for the cursor in a MySQL shell session, where MySQL would create the real cursor for us if we logged into a MySQL database. We must here create the proxy ourselves.

Una vez creado el objeto de conexión, no puede interactuar con la base de datos hasta que cree un objeto de cursor. El nombre del cursor contradice el propósito de este objeto. Los cursores existen en cualquier aplicación de productividad y han sido parte de la informática desde el principio. El objetivo de un cursor es marcar su lugar y permitirle enviar comandos a la computadora. Un cursor en MySQL para Python sirve como un proxy basado en Python para el cursor en una sesión de shell de MySQL, donde MySQL crearía el cursor real para nosotros si iniciamos sesión en una base de datos MySQL. Aquí debemos crear el proxy nosotros mismos.

To create the cursor, we use the **cursor() method** of the MySQLdb.connections object we created for the connection. The syntax is as follows:

**[cursor name] = [connection object name].cursor()**

Using our example of the menu database above, we can use a generic name cursor for the database cursor and create it in this way:

**cursor = mydb.cursor()**

Now, we are ready to issue commands.

**Interacting with the database**

Many SQL commands can be issued using a single function as:

**cursor.execute()**

There are other ways to issue commands to MySQL depending on the results one wants back, but this is one of the most common. Its use will be addressed in greater detail in future chapters.

Hay otras formas de emitir comandos a MySQL dependiendo de los resultados que uno quiera obtener, pero esta es una de las más comunes. Su uso se abordará con mayor detalle en capítulos futuros.

**Closing the connection**

In MySQL, you are expected to close the databases and end the session by issuing either quit or exit.

To do this in Python, we use the **close() method** of the database object. Whether you close a database outright depends on what actions you have performed and whether MySQL's auto-commit feature is turned on. By default, MySQL has autocommit switched on. Your database administrator will be able to confirm whether auto-commit is switched on. If it is not, you will need to commit any changes you have made. We do this by calling the commit method of the database object. For mydb, it would look like this:

Para hacer esto en Python, usamos el método close() del objeto de base de datos. El hecho de cerrar una base de datos directamente depende de las acciones que haya realizado y de si la función de confirmación automática de MySQL está activada. De forma predeterminada, MySQL tiene activada la confirmación automática. El administrador de su base de datos podrá confirmar si la confirmación automática está activada. Si no es así, deberá confirmar los cambios que haya realizado. Hacemos esto llamando al método de confirmación del objeto de la base de datos. Para mydb, se vería así:

**mydb.commit()**

After all changes have been committed, we can then close the database:

**mydb.close()**

**Multiple database connections**

In MySQL for Python, all database objects are discrete. All you need do is to connect with each under a different name. Consider the following:

En MySQL para Python, todos los objetos de la base de datos son discretos. Todo lo que necesitas hacer es conectarte con cada uno con un nombre diferente. Considera lo siguiente:

**mydb1 = MySQLdb.connect(host="localhost",**

**user="skipper",**

**passwd="mysecret",**

**db="fish")**

**mydb2 = MySQLdb.connect(host="localhost",**

**user="skipper",**

**passwd="mysecret",**

**db="fruit")**

**cursor1 = mydb1.cursor()**

**cursor2 = mydb2.cursor()**

The objects then function like any other variable or object. By calling their methods and attributes separately, you can interact with either or even copy from one to the other.

Luego, los objetos funcionan como cualquier otra variable u objeto. Al llamar a sus métodos y atributos por separado, puede interactuar con cualquiera de ellos o incluso copiarlos de uno a otro.

**Summary**

In this chapter we have looked at where to find MySQL for Python, as it is not part of Python by default. We have also seen how to install it on both Windows and non-Windows systems—UNIX-like and Linux distributions. The authors of MySQL for Python have taken the pain out of this by providing a very easy way to install through an egg utility like EasyInstall.

Like most modules, MySQL for Python must be imported before you can use it in Python. So we then looked at how to import it. Unlike most modules, we saw that MySQL for Python needs to be imported by its earlier moniker, MySQLdb.

After that, we took a peek at what is waiting for us under the MySQL for Python covers using help(). We saw that MySQL for Python is not an interface to MySQL itself but to a MySQL Database API that is built into Python. It has a large number of classes for handling errors, but only one for processing data (There are different kinds of cursors). Further, it does not even use classes to access MySQL, but uses functions to process and pass information to \_mysql, which then passes it to the C MySQL database interface.

Following this trail, we also saw that \_mysql does not have a robust facility for handling errors, but only passes them to the calling process. That is why MySQL for Python has such a robust error handling facility.

Next, we saw how to connect to a MySQL database. As with most parts of Python, this is easy for beginners. But the function used is also sufficiently robust to handle the more complex needs of advanced solutions.

After connecting, we created a MySQLdb cursor and prepared to interact with the database. This showed that, while there are many things that MySQLdb will take care of for us (like connection closure), there are some things we need to do manually. In this instance, it is creating the cursor object that represents the MySQL cursor.

Finally, we saw that one can connect to multiple databases by simply using different object names for each connection. This has the consequence of necessitating different namespaces as we refer to the methods and attributes of each object. But it also allows one to bridge between databases across multiple hosts seamlessly and to present a unified interface for a user.

In the next chapter, we will see how to form a MySQL query and pass it from Python using variables from the system, MySQL, and the user.