MySQL\_for\_Python\_Albert\_c02

Simple Querying

Record retrieval is without doubt the most common activity employed with regard to MySQL and other relational databases. Like most computer programs, MySQL functions on the basis of being invoked with parameters and returning results in accordance with them. As we seen, Python acts as an intermediary to that process. We can use it to access MySQL, login, and connect to a database of our choice.

La recuperación de registros es sin duda la actividad más común empleada con respecto a MySQL y otras bases de datos relacionales. Como la mayoría de los programas informáticos, MySQL funciona basándose en ser invocado con parámetros y devolver resultados de acuerdo con ellos. Como vimos, Python actúa como intermediario en ese proceso. Podemos usarlo para acceder a MySQL, iniciar sesión y conectarnos a una base de datos de nuestra elección.

In this chapter, we will look at the following:

* Forming a MySQL query directly
* Passing a query to MySQL
* User-defined variables in a MySQL query
* Determining characteristics of a database and its tables
* Changing queries dynamically, without user input

Working through each of these points will help you at the end of the chapter, when we get to the project: a command-line search tool.

Trabajar en cada uno de estos puntos le ayudará al final del capítulo, cuando lleguemos al proyecto: una herramienta de búsqueda de línea de comandos.

A brief introduction to CRUD

The four basic functions of any persistent storage system like MySQL spell CRUD:

* Create
* Read
* Update
* Delete

These are key concepts, which each of the basic MySQL commands reflect. There is nothing technical about the words themselves, but the concepts are very important. They represent the four activities that you can expect to be able to do in every relational database system you use. There are several alternatives to this acronym and keyword series (for example, SCUD for "select, create, update, and delete" or SIDU for "select, insert, delete, and update"). The point of each of these is that database functionality boils down to two sets of opposing activities:

Estos son conceptos clave, que reflejan cada uno de los comandos básicos de MySQL. No hay nada técnico en las palabras en sí, pero los conceptos son muy importantes. Representan las cuatro actividades que puede esperar poder realizar en cada sistema de base de datos relacional que utilice. Existen varias alternativas a esta serie de acrónimos y palabras clave (por ejemplo, SCUD para "seleccionar, crear, actualizar y eliminar" o SIDU para "seleccionar, insertar, eliminar y actualizar"). El objetivo de cada uno de ellos es que la funcionalidad de la base de datos se reduce a dos conjuntos de actividades opuestas:

* Creating and deleting database objects (for example, databases, tables, records)
* Inserting and reading data (that is writing and reading)

Each of these will be addressed in the coming chapters. In this one, we start with reading data using SELECT.

**Forming a query in MySQL**

In order to best understand how to submit a query through MySQL for Python, it is important to ensure you understand how to submit a query in MySQL itself. The similarities between the two outnumber the differences, but the first may seem confusing if you don't properly understand the second.

Para comprender mejor cómo enviar una consulta a través de MySQL para Python, es importante asegurarse de comprender cómo enviar una consulta en MySQL. Las similitudes entre los dos superan en número a las diferencias, pero la primera puede parecer confusa si no se comprende adecuadamente la segunda.

MySQL statements have a basic structure. In following a set structure, they are formed like natural language statements. Being a computer program, it understandably responds very poorly to informational statements and only moderately well to questions. Almost all MySQL statements have an imperatival tone, expressing your command. This is reflective of the client-server relationship. The computer is the servant who exists to do the bidding of yourself as the client or, if you prefer, master.

Las declaraciones de MySQL tienen una estructura básica. Al seguir una estructura establecida, se forman como declaraciones en lenguaje natural. Al ser un programa informático, es comprensible que responda muy mal a declaraciones informativas y sólo moderadamente bien a preguntas. Casi todas las declaraciones de MySQL tienen un tono **imperativo**, expresando su orden. Esto refleja la relación cliente-servidor. La computadora es el sirviente que existe para cumplir sus órdenes como cliente o, si lo prefiere, maestro.

The syntactic structure of a simple MySQL statement is not that different from the language you use every day. Where English would have:

La estructura sintáctica de una declaración MySQL simple no es tan diferente del lenguaje que usas todos los días. Donde el inglés tendría:

* Give me everything from the staff table!

MySQL would need to hear:

* SELECT \* FROM staff;

Let's look at the MySQL statement, comparing it to the English in detail.

**SELECT**

MySQL does not support natural language searching like Give me. Rather, like other programming languages including Python, MySQL has a set of reserved key words. These are largely single synonyms for common, core actions. For data retrieval, the key word is **SELECT**. It could have been GIMME or any of a score of similar ways of saying the same thing, but MySQL is consonant with the Zen of Python:

MySQL no admite búsquedas en lenguaje natural como Dame. Más bien, al igual que otros lenguajes de programación, incluido Python, MySQL tiene un conjunto de palabras clave reservadas. Estos son en gran medida sinónimos únicos de acciones centrales y comunes. Para la recuperación de datos, la palabra clave es SELECCIONAR. Podría haber sido GIMME o cualquiera de una veintena de formas similares de decir lo mismo, pero MySQL está en consonancia con el Zen de Python:

There should be one—and preferably only one—obvious way to do it

Therefore, the MySQL developers settled on a single keyword—one that just happens to be compliant with the SQL standard.

Por lo tanto, los desarrolladores de MySQL se decidieron por una sola palabra clave, una que cumpla con el estándar SQL.

**\* (asterisk)**

Being read up on your regular expressions, I am sure you recognize this universal quantifier. While it is one of the most commonly used, MySQL supports several metacharacters that you can use to nuance your searches.

Al leer sus expresiones regulares, estoy seguro de que reconoce este cuantificador universal. Si bien es uno de los más utilizados, MySQL admite varios metacaracteres que puedes utilizar para matizar tus búsquedas.

MySQL supports different kinds of metacharacters in different contexts. The following is a full list of metacharacters. (Note that not all of them may be supported in a given situation.)

• .: To match any single character

• ?: To match zero or one character

• \*: To match zero or more characters

• +: To match one or more characters

• {n}: To match an expression n times

• {m,n}: To match an expression a minimum of m and a maximum

of n times

• {n,}: To match an expression n or more times

• ^: Indicates the bit-level indicator at the beginning of a line

• $: Indicates the bit-level indicator at the end of a line

• [[:<:]]: To match the beginning of words

• [[:>:]]: To match the ending of words

• [:class:]: To match a character class

• [:alpha:]: For letters

• [:space:]: For whitespace

• [:punct:]: For punctuation

• [:upper:]: For upper case letters

• [abc]: To match one of the enclosed characters

• [^xyz]: To match any character other than those enclosed

• |: Separates alternatives within an expression

In the case of the SELECT command, the asterisk is the only metacharacter supported. In addition to the asterisk, however, SELECT also supports several arguments used to quantify results:

En el caso del comando SELECT, el asterisco es el único metacarácter admitido. Sin embargo, además del asterisco, SELECT también admite varios argumentos utilizados para cuantificar los resultados:

* ALL: All matching rows (synonymous to using an asterisk (\*)
* DISTINCT: Sort the results set into unique values
* DISTINCTROW: Where the entire record is unique

Each of these can be prefaced before the field to be quantified as illustrated here using the database structure from the last chapter:

Cada uno de estos puede ir precedido del campo que se va a cuantificar, como se ilustra aquí utilizando la estructura de la base de datos del último capítulo:

**SELECT DISTINCT id FROM menu;**

This would return the values of the id column from the menu table and remove any duplicates from the results.

**FROM**

As with the English equivalent, MySQL needs some context in order to retrieve anything. In English, if one simply said Give me! without non-verbal cues for the intended context, the listener would rightly be confused. So here we tell MySQL from which table in the current database we want it to extract information.

Al igual que su equivalente en inglés, MySQL necesita algo de contexto para poder recuperar algo. En inglés, si uno simplemente dijera ¡Dame! sin señales no verbales para el contexto deseado, el oyente estaría confundido. Entonces aquí le decimos a MySQL de qué tabla en la base de datos actual queremos que extraiga información.

Note that this is technically optional. Leaving it off, however, typically means that you are using MySQL's built-in datasets. For example, here is a statement using the built-in functions for the current date, user, and version (the \G is the same command as \g, but it tells MySQL to display the results vertically):

Tenga en cuenta que esto es técnicamente opcional. Sin embargo, dejarlo desactivado normalmente significa que está utilizando los conjuntos de datos integrados de MySQL. Por ejemplo, aquí hay una declaración que utiliza las funciones integradas para la fecha, el usuario y la versión actuales (\G es el mismo comando que \g, pero le dice a MySQL que muestre los resultados verticalmente):

**mysql> SELECT NOW(), USER(), VERSION()\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NOW(): 2009-08-29 12:29:23

USER(): skipper@localhost

VERSION(): 5.1.31-1ubuntu2

1 row in set (0.00 sec)

**staff**

This is merely the name of the table to be searched. In English, there are many locations from which one may desire something. That is why we would need to clarify that we want the items from the far table. MySQL, on the other hand, only understands things in terms of databases and tables and so understands as the name of a table whatever immediately follows the FROM keyword.

Este es simplemente el nombre de la tabla que se va a buscar. En inglés, hay muchos lugares desde donde uno puede desear algo. Es por eso que necesitaríamos aclarar que queremos los elementos de la tabla lejana. MySQL, por otro lado, sólo entiende las cosas en términos de bases de datos y tablas, por lo que entiende como el nombre de una tabla todo lo que sigue inmediatamente a la palabra clave FROM.

**; (semicolon)**

The semicolon is the default statement delimiter in MySQL. When creating a MySQL script or interacting with MySQL dynamically through its shell, leaving off a semicolon at the end of a statement will result in either a second prompt or, if you press the matter, an error being thrown. As we will see shortly, the syntax of MySQL for Python and Python itself mandates a different way of showing the end of the line. Therefore when passing MySQL queries in Python, we do not need to end any statements with the semicolon.

El punto y coma es el delimitador de declaraciones predeterminado en MySQL. Al crear un script MySQL o interactuar con MySQL dinámicamente a través de su shell, dejar un punto y coma al final de una declaración dará como resultado un segundo mensaje o, si presiona el asunto, se generará un error. Como veremos en breve, la sintaxis de MySQL para Python y el propio Python exigen una forma diferente de mostrar el final de la línea. Por lo tanto, al pasar consultas MySQL en Python, no es necesario finalizar ninguna declaración con punto y coma.

You may wonder why certain portions of the MySQL query are capitalized. It is a standard presentation format for MySQL statements to present the static or standard elements of a MySQL statement in capitals. Variable parts of the statement, however, are case sensitive and must be called with the same case in which they were created (otherwise, MySQL will throw an error). This matter of capitalization is not significant if you interact with MySQL directly, from a MySQL prompt. There, MySQL will understand your statements whether they be in all caps or lowercase. However, in your code, proper capitalization is critical to making your SQL statements readable to the next developer—both in Python and in MySQL scripts.

Quizás se pregunte por qué ciertas partes de la consulta MySQL están en mayúscula. Es un formato de presentación estándar para declaraciones MySQL para presentar los elementos estáticos o estándar de una declaración MySQL en mayúsculas. Las partes variables de la declaración, sin embargo, distinguen entre mayúsculas y minúsculas y deben llamarse con el mismo caso en el que fueron creadas (de lo contrario, MySQL arrojará un error). Esta cuestión de las mayúsculas no es significativa si interactúa con MySQL directamente, desde un indicador de MySQL. Allí, MySQL entenderá sus declaraciones, ya sea que estén en mayúsculas o minúsculas. Sin embargo, en su código, el uso adecuado de mayúsculas es fundamental para que sus declaraciones SQL sean legibles para el siguiente desarrollador, tanto en scripts de Python como de MySQL.

Where the semicolon is the statement delimiter in the MySQL shell, the backslash (\) is used to delimit lines within a statement. So, if you want to break up a statement but not have it executed when you press return, simply use a backslash at the end of each line of the statement. For example:

Donde el punto y coma es el delimitador de declaraciones en el shell MySQL, la barra invertida (\) se usa para delimitar líneas dentro de una declaración. Por lo tanto, si desea dividir una declaración pero no ejecutarla cuando presiona Intro, simplemente use una barra invertida al final de cada línea de la declaración. Por ejemplo:

**mysql> SELECT \**

**-> \* \**

**-> FROM \**

**-> menu;**

**Other helpful quantifiers**

The previous discussion offers an overview of the SELECT command and its most common arguments. There are many other ways to nuance the data. In addition to FROM, you can also employ SELECT with the following optional arguments.

La discusión anterior ofrece una descripción general del comando SELECT y sus argumentos más comunes. Hay muchas otras formas de matizar los datos. Además de FROM, también puedes emplear SELECT con los siguientes argumentos opcionales.

**WHERE**

WHERE is used to declare a condition under which MySQL is to narrow the results of the search. The basic syntax of the clause is:

WHERE se utiliza para declarar una condición bajo la cual MySQL limitará los resultados de la búsqueda. La sintaxis básica de la cláusula es:

**[WHERE where\_condition]**

For example:

**mysql> SELECT \* FROM menu WHERE id='5';**

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

| id | name

| price |

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

|

5 | trout |

6.00 |

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

1 row in set (0.00 sec)

**GROUP BY**

GROUP BY allows you to group results according to one of the following three parameters:

* col\_name: Is the name of one of the table's columns
* expr: Is a regular expression
* position: Is a position in the table

Once grouped, you can then tell MySQL to list the results in either ASCending or DESCending order through ASC and DESC, respectively. The former is the default. Additionally, MySQL provides for a summative line at the end of the results through the use of WITH ROLLUP.

Una vez agrupados, puede decirle a MySQL que enumere los resultados en orden ASCENDENTE o DESCENDENTE a través de ASC y DESC, respectivamente. El primero es el predeterminado. Además, MySQL proporciona una línea sumativa al final de los resultados mediante el uso de FROM ROLLUP.

The syntax of a GROUP BY clause is:

**GROUP BY {col\_name | expr | position} [ASC | DESC], [WITH ROLLUP]**

To appreciate the effect of GROUP BY, you can retrieve all of the values from a table.

**mysql> SELECT \* FROM menu;**

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

| ID | NAME

| PRICE |

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

| 4 | catfish|5.00 |

| 2 | bass|6.75 |

| 6 | haddock|6.50 |

| 3 | salmon|9.50 |

| 5 | trout|6.00 |

| 1 | tuna|7.50 |

| 7 | yellowfin tuna | 12.00 |

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

7 rows in set (0.00 sec)

Using GROUP BY on just one column can give us the same list in alphabetical order:

**mysql> SELECT \* FROM menu GROUP BY name;**

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

| id | name

| price |

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

|2 | bass|6.75 |

|4 | catfish|5.00 |

|6 | haddock|6.50 |

|3 | salmon|9.50 |

|5 | trout|6.00 |

|1 | tuna|7.50 |

|7 | yellowfin tuna | 12.00 |

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

7 rows in set (0.00 sec)

[ 31 ]Simple Querying

If we had multiple entries for some of the fish (for example, tuna and yellowfin tuna), it could also be used to give a count by field value.

Si tuviéramos varias entradas para algunos de los peces (por ejemplo, atún y atún aleta amarilla), también podría usarse para dar un recuento por valor de campo.

**mysql> SELECT name, count(\*) FROM menu GROUP BY name;**

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

| name

| count(\*) |

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

| bass|1 |

| catfish|1 |

| haddock|1 |

| salmon|1 |

| trout|1 |

| tuna|2 |

| yellowfin tuna |2 |

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

7 rows in set (0.00 sec)

More on how to use the modifiers of GROUP BY can be found in Section 11.12 of the

MySQL manual.

**HAVING**

As the WHERE clause has already been discussed, one might wonder rightly—what is the point of the HAVING clause? The WHERE clause is used for simple facts and does not support aggregate evaluations. The HAVING clause is used for aggregate functions. It can be used to replace WHERE, but to do so is generally viewed as poor coding because it violates the SQL standard.

Como ya se ha discutido la cláusula WHERE, uno podría preguntarse con razón: ¿cuál es el punto de la cláusula HAVING? La cláusula WHERE se utiliza para hechos simples y no admite evaluaciones agregadas. La cláusula HAVING se utiliza para funciones agregadas. Se puede utilizar para reemplazar WHERE, pero hacerlo generalmente se considera una codificación deficiente porque viola el estándar SQL.

The HAVING clause is used to quantify results according to aggregate functions. For this reason, it is usually used in conjunction with the GROUP BY clause.

La cláusula HAVING se utiliza para cuantificar resultados según funciones agregadas. Por este motivo, suele utilizarse junto con la cláusula GROUP BY.

The basic syntax of the HAVING clause is:

**HAVING where\_condition**

Carrying on with the previous menu example, a basic example of this is:

**mysql> SELECT \* FROM menu GROUP BY name HAVING id>'3';**

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

| id | name

| price |

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

|4 | catfish|5.00 |

|6 | haddock|6.50 |

|5 | trout|6.00 |

|7 | yellowfin tuna | 12.00 |

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

4 rows in set (0.00 sec)

For an example closer to real life a video rental store that wants to know which customers rent the most videos might use a query like this one:

Para ver un ejemplo más cercano a la vida real, una tienda de alquiler de vídeos que quiera saber qué clientes alquilan más vídeos podría utilizar una consulta como esta:

**mysql> SELECT customer\_id,count(\*) AS cnt FROM rental GROUP BY customer\_**

**id HAVING cnt> 40;**

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

| customer\_id | cnt |

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

|75 |41 |

|144 |42 |

|148 |46 |

|236 |42 |

|526 |45 |

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

5 rows in set (0.05 sec)

This shows the customer number followed by the number of total rentals in the record of rentals for each customer whose aggregate custom exceeds 40 videos.

Esto muestra el número de cliente seguido del número de alquileres totales en el registro de alquileres para cada cliente cuya costumbre agregada supere los 40 vídeos.

**ORDER BY**

As the name implies, the ORDER BY clause is used to tell MySQL how to order the results of a query. The basic syntactical structure of this clause is as follows:

Como su nombre lo indica, la cláusula ORDER BY se utiliza para indicarle a MySQL cómo ordenar los resultados de una consulta. La estructura sintáctica básica de esta cláusula es la siguiente:

**[ORDER BY {col\_name | expr | position} [ASC | DESC], ...]**

While the ORDER BY clause can be used in conjunction with the GROUP BY modifiers, this is typically not necessary. The following two examples illustrate why:

Si bien la cláusula ORDER BY se puede utilizar junto con los modificadores GROUP BY, normalmente esto no es necesario. Los dos ejemplos siguientes ilustran por qué:

**mysql> SELECT \* FROM menu GROUP BY name ORDER BY id DESC;**

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

| id | name

| price |

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

|7 | yellowfin tuna | 12.00 |

|6 | haddock

|

6.50 |

|5 | trout

|

6.00 |

|4 | catfish

|

5.00 |

|3 | salmon

|

9.50 |

|2 | bass

|

6.75 |

|1 | tuna

|

7.50 |

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

7 rows in set (0.00 sec)

**mysql> SELECT \* FROM menu ORDER BY id DESC;**

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

| id | name

| price |

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

|7 | yellowfin tuna | 12.00 |

|6 | haddock

|

6.50 |

|5 | trout

|

6.00 |

|4 | catfish

|

5.00 |

|3 | salmon

|

9.50 |

|2 | bass

|

6.75 |

|1 | tuna

|

7.50 |

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

7 rows in set (0.00 sec)

Because the ORDER BY is applied after the GROUP BY, it largely abrogates the need for the grouping.

Debido a que ORDER BY se aplica después de GROUP BY, anula en gran medida la necesidad de agrupar.

**LIMIT**

The LIMIT clause is used to restrict the number of rows that are returned in the result set. It takes two positive integers as arguments. The first number indicates the point at which to start counting and counts from zero for that process. The second number indicates how many times to increment the first number by one in order to determine the desired limit.

La cláusula LIMIT se utiliza para restringir el número de filas que se devuelven en el conjunto de resultados. Toma dos números enteros positivos como argumentos. El primer número indica el punto en el que comenzar a contar y cuenta desde cero para ese proceso. El segundo número indica cuántas veces incrementar el primer número en uno para determinar el límite deseado.

The syntax of the LIMIT clause is as follows:

LIMIT {[offset,] row\_count | row\_count OFFSET offset}

The following four examples show how LIMIT may be used to reduce the returned results neatly. Used in an iterative fashion, incrementing the parameters of a LIMIT clause allows you to step through results.

Los siguientes cuatro ejemplos muestran cómo se puede utilizar LIMIT para reducir claramente los resultados devueltos. Usado de forma iterativa, incrementar los parámetros de una cláusula LIMIT le permite avanzar por los resultados.

In this first example, LIMIT is applied to an alphabetic listing of fish names. The table in question is the same one we used previously for GROUP BY. Note that the id numbers are out of sequence.

En este primer ejemplo, LIMIT se aplica a una lista alfabética de nombres de peces. La tabla en cuestión es la misma que usamos anteriormente para GROUP BY. Tenga en cuenta que los números de identificación están fuera de secuencia.

**mysql> SELECT \* FROM menu GROUP BY name LIMIT 3,4;**

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

| id | name

| price |

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

|3 | salmon|9.50 |

|5 | trout|6.00 |

|1 | tuna|7.50 |

|7 | yellowfin tuna | 12.00 |

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

4 rows in set (0.00 sec)

In order to get the id numbers sequenced correctly, we employ an ORDER BY clause to prep the data before applying the terms of the LIMIT clause to it.

Para secuenciar correctamente los números de identificación, empleamos una cláusula ORDER BY para preparar los datos antes de aplicarles los términos de la cláusula LIMIT.

**mysql> SELECT \* FROM menu ORDER BY id LIMIT 3,4;**

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

| id | name

| price |

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

|4 | catfish|5.00 |

|5 | trout|6.00 |

|6 | haddock|6.50 |

|7 | yellowfin tuna | 12.00 |

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

4 rows in set (0.00 sec)

[ 35 ]Simple Querying

These final two examples illustrate how to apply LIMIT to searches that could easily return scores, if not hundreds or thousands, of hits.

Estos dos últimos ejemplos ilustran cómo aplicar LIMIT a búsquedas que fácilmente podrían devolver decenas, si no cientos o miles, de resultados.

**mysql> SELECT \* FROM menu ORDER BY id LIMIT 2,3;**

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

| id | name

| price |

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

|3 | salmon|9.50 |

|4 | catfish |5.00 |

|5 | trout6.00 |

|

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

3 rows in set (0.00 sec)

**mysql> SELECT \* FROM menu LIMIT 2,4;**

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

| id | name

| price |

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

|3 | salmon|9.50 |

|4 | catfish |5.00 |

|5 | trout|6.00 |

|6 | haddock |6.50 |

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

4 rows in set (0.00 sec)

LIMIT and HAVING may seem very similar as they both work to narrow the aggregate. The difference between them lies in the timing of their application by MySQL. HAVING is applied as a parameter of the search before MySQL actions the query. The LIMIT clause, on the other hand, is applied after the search results have been returned.

LÍMITE y TENER pueden parecer muy similares ya que ambos funcionan para reducir el agregado. La diferencia entre ellos radica en el momento de su aplicación por parte de MySQL. HAVING se aplica como parámetro de la búsqueda antes de que MySQL realice la consulta. La cláusula LIMIT, por otro lado, se aplica después de que se hayan devuelto los resultados de la búsqueda.

If you are programming for a web application and your database and web server are located on a single machine, you need to conserve your server resources. Therefore, you almost certainly want to use HAVING instead of LIMIT. If you are trying to reduce your search time, again, use HAVING. However, if your desired hits will comprise a sizable portion of the results otherwise, or your database server, application server, and web server are each discrete systems from each other, then you might consider using LIMIT. In the main, however, LIMIT allows MySQL to use more resources than HAVING because the former is applied after the query is already processed.

Si está programando para una aplicación web y su base de datos y su servidor web están ubicados en una sola máquina, necesita conservar los recursos de su servidor. Por lo tanto, es casi seguro que desee utilizar HAVING en lugar de LIMIT. Si está intentando reducir el tiempo de búsqueda, utilice nuevamente HAVING. Sin embargo, si los resultados deseados comprenderán una parte considerable de los resultados, o si su servidor de base de datos, servidor de aplicaciones y servidor web son sistemas discretos entre sí, entonces podría considerar usar LIMIT. Sin embargo, en general, LIMIT permite a MySQL usar más recursos que HAVING porque el primero se aplica después de que la consulta ya haya sido procesada.

**INTO OUTFILE**

INTO OUTFILE allows for the rapid output of tabular results to a text file on the local host. Its basic syntax is as follows:

INTO OUTFILE permite la salida rápida de resultados tabulares a un archivo de texto en el host local. Su sintaxis básica es la siguiente:

**INTO OUTFILE 'file\_name'**

For example, one could use:

**mysql> SELECT \* FROM menu ORDER BY id LIMIT 3,4 INTO OUTFILE '/tmp/**

**results.txt';**

Query OK, 4 rows affected (0.00 sec)

This would output the results of the query to a file results.txt in the /tmp directory of the server.

Esto generaría los resultados de la consulta en un archivo results.txt en el directorio /tmp del servidor.

More information can be found in the MySQL manual, Section 12.2.8 (URL: http://dev.mysql.com/doc/refman/5.1/en/select.html). It is understood that MySQL for Python allows a program to process the data and output the results using Python's own I/O calls.

**Passing a query to MySQL**

We have just seen how to form a query for a generic MySQL session. While that was not particularly difficult, using MySQL for Python is even easier. For this next section, we will be working against a database fish with a table menu that has the following contents:

Acabamos de ver cómo formular una consulta para una sesión genérica de MySQL. Si bien eso no fue particularmente difícil, usar MySQL para Python es aún más fácil. En la siguiente sección, trabajaremos con un pez de base de datos con un menú de tabla que tiene el siguiente contenido:

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

| id | name

| price |

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

|1 | tuna|7.50 |

|2 | bass|6.75 |

|3 | salmon|9.50 |

|4 | catfish|5.00 |

|5 | trout|6.00 |

|6 | haddock|6.50 |

|7 | yellowfin tuna | 12.00 |

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

[ 37 ]Simple Querying

As discussed in Chapter 1, Python's interface with MySQL requires a cursor. It is through the cursor object that we pass commands to MySQL. So, we import MySQL for Python, log into our database fish and create the cursor as follows:

Como se analizó en el Capítulo 1, la interfaz de Python con MySQL requiere un cursor. Es a través del objeto cursor que pasamos comandos a MySQL. Entonces, importamos MySQL para Python, iniciamos sesión en nuestra base de datos fish y creamos el cursor de la siguiente manera:

**import MySQLdb**

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

**user = 'skipper',**

**passwd = 'mysecret',**

**db = 'fish')**

**cur = mydb.cursor()**

**A simple SELECT statement**

To pass a command to MySQL, we use the execute() method that we briefly covered in the last chapter. The execute() method, as the name implies, expects an argument of what is to be executed by Python. In other words, it takes the MySQL sentence or statement as its argument. Its basic syntax is as follows:

Para pasar un comando a MySQL, usamos el método ejecutar() que cubrimos brevemente en el último capítulo. El método ejecutar(), como su nombre lo indica, espera un argumento de lo que Python ejecutará. En otras palabras, toma la oración o declaración de MySQL como argumento. Su sintaxis básica es la siguiente:

**results\_variable = cursor\_handle.execute('MySQL statement')**

In practice, it looks like this:

**command = cur.execute('SELECT \* FROM menu')**

As you might surmise from the previous discussion on SELECT, this returns all rows of the table menu.

Como puede suponer de la discusión anterior sobre SELECT, esto devuelve todas las filas del menú de la tabla.

You will notice that we did not have to specify the database in the execute() call. This is because it was already specified in the MySQLdb.connect() call. Each connection represents one database being accessed in the name of one user on one host. If any of those dynamics need to change, a new connection object becomes necessary. It is possible to create a connection without declaring a database at the outset, but a database must be specified before a cursor can be created or a query made.

Notarás que no tuvimos que especificar la base de datos en la llamada ejecutar(). Esto se debe a que ya se especificó en la llamada MySQLdb.connect(). Cada conexión representa una base de datos a la que se accede en nombre de un usuario en un host. Si alguna de esas dinámicas necesita cambiar, se hace necesario un nuevo objeto de conexión. Es posible crear una conexión sin declarar una base de datos desde el principio, pero se debe especificar una base de datos antes de poder crear un cursor o realizar una consulta.

Unlike in the MySQL shell, the execute() call here does not immediately return the results. They are held in system memory (RAM) until you tell MySQL for Python what you want to do with them. This is another reason why it is important to mind your use of system resources in the use of HAVING and LIMIT, as mentioned previously.

A diferencia del shell MySQL, la llamada a ejecutar() aquí no devuelve los resultados inmediatamente. Se mantienen en la memoria del sistema (RAM) hasta que le dices a MySQL para Python qué quieres hacer con ellos. Esta es otra razón por la que es importante tener en cuenta el uso de los recursos del sistema al utilizar HAVING y LIMIT, como se mencionó anteriormente.

For the purposes of illustration, we can pull down all of the results in one go. To do this, we use the fetchall() method of the cursor object.

A modo de ilustración, podemos desplegar todos los resultados de una sola vez. Para hacer esto, usamos el método fetchall() del objeto cursor.

**results = command.fetchall()**

At this point, the results have now passed from MySQL for Python into the calling program's resource matrix. The fetchall() method returns its results as a series of tuples. Printing the value of results shows the following:

En este punto, los resultados han pasado de MySQL para Python a la matriz de recursos del programa que realiza la llamada. El método fetchall() devuelve sus resultados como una serie de tuplas. Al imprimir el valor de los resultados se muestra lo siguiente:

>>> **print results**

((1L, 'tuna', Decimal('7.50')), (2L, 'bass', Decimal('6.75')), (3L,

'salmon', Decimal('9.50')), (4L, 'catfish', Decimal('5.00')), (5L,

'trout', Decimal('6.00')), (6L, 'haddock', Decimal('6.50')), (7L,

'yellowfin tuna', Decimal('12.00')))

This is obviously far from human-friendly. However, we can now use Python's own data-handling resources to parse it. In programming terms, we now have greater control over our data and can present it as we want. So let's create a loop to iterate through the results and print the results in a formatted way.

Obviamente, esto está lejos de ser amigable para los humanos. Sin embargo, ahora podemos usar los recursos de manejo de datos propios de Python para analizarlos. En términos de programación, ahora tenemos un mayor control sobre nuestros datos y podemos presentarlos como queramos. Entonces, creemos un bucle para iterar a través de los resultados e imprimirlos en forma formateada.

**>>> for record in results:**

**… print record[0] , “-->”, record[1] , “ @”, record[2], “each”**

...

1 --> tuna@ 7.50 each

2 --> bass@ 6.75 each

3 --> salmon@ 9.50 each

4 --> catfish@ 5.00 each

5 --> trout

6 --> haddock

@ 6.00 each

7

@ 6.50 each

--> yellowfin tuna

@ 12.00 each

**Modifying the results**

The last query could easily have returned more results than we could use. As mentioned previously, this is why the SELECT command comes with a comprehensive suite of modifiers to nuance one's query and, hopefully, use a minimal amount of system resources.

La última consulta fácilmente podría haber arrojado más resultados de los que podríamos utilizar. Como se mencionó anteriormente, esta es la razón por la cual el comando SELECT viene con un conjunto completo de modificadores para matizar la consulta y, con suerte, utilizar una cantidad mínima de recursos del sistema.

To use GROUP BY, ORDER BY, or any of the other clauses that one can add to a SELECT statement, one simply adds them to the MySQL statement that is passed to the execute() method.

Para usar GROUP BY, ORDER BY o cualquiera de las otras cláusulas que se pueden agregar a una instrucción SELECT, simplemente se agregan a la instrucción MySQL que se pasa al método ejecutar().

If you wanted to retrieve information only on fish whose price is greater than $7, you would need to sort through the data again and find the record with the matching name. Better to let MySQL do the dirty work. Using the preceding simple query (see Where under Other helpful quantifiers), we can do the following:

Si desea recuperar información solo sobre pescado cuyo precio es superior a $7, deberá ordenar los datos nuevamente y encontrar el registro con el nombre coincidente. Es mejor dejar que MySQL haga el trabajo sucio. Utilizando la consulta simple anterior (consulte Dónde en Otros cuantificadores útiles), podemos hacer lo siguiente:

**command = cur.execute("""SELECT \* FROM menu WHERE price > 7""")**

**results = command.fetchall()**

**for record in results:**

**print record[0], ". ", record[1], "(%s)" %record[2]**

The results would be:

1 .tuna (7.50)

3 .salmon (9.50)

7 .yellowfin tuna (12.00)

Similar statements can be passed for each of the SELECT clauses discussed above.

**Using user-defined variables**

What if you want to specify a different price floor every time you run the search? What if you didn't want to use a floor but specify the price exactly? What if you wanted to reuse part of the statement and automate queries by fish name instead of retrieving all of them at once? Under such circumstances, you need to be able to handle variables in your SELECT statements.

¿Qué sucede si desea especificar un precio mínimo diferente cada vez que ejecuta la búsqueda? ¿Qué pasa si no desea utilizar un suelo pero especifica el precio exactamente? ¿Qué pasaría si quisiera reutilizar parte de la declaración y automatizar las consultas por nombre de pez en lugar de recuperarlas todas a la vez? En tales circunstancias, debe poder manejar variables en sus declaraciones SELECT.

MySQL for Python passes variables to MySQL in the same way that Python formats other kinds of output. If we wanted to specify just the floor of the search, we would assign the variable as any other and pass it to the execute() method as a string. Consider the following snippet from a Python terminal session:

MySQL para Python pasa variables a MySQL de la misma manera que Python formatea otros tipos de resultados. Si quisiéramos especificar solo el piso de la búsqueda, asignaríamos la variable como cualquier otra y la pasaríamos al método ejecutar() como una cadena. Considere el siguiente fragmento de una sesión de terminal Python:

**>>> value = "7.50"**

**>>> command = cur.execute("""SELECT \* FROM menu WHERE price = %s"""**

**%(value))**

**>>> results = cur.fetchall()**

**>>> for record in results:**

**… print record[0], ". ", record[1], "(%s)" %record[2]**

**… 1 . tuna (7.50)**

If we wanted the user to have the option of specifying the price precisely or using comparative expressions, we can add in that option along with making the previous variable user-defined.

Si queremos que el usuario tenga la opción de especificar el precio con precisión o usar expresiones comparativas, podemos agregar esa opción además de hacer que la variable anterior sea definida por el usuario.

**>>> operation = input("operation: ")**

operation: '='

**>>> value = input("value: ")**

value: 7.50

**>>> command = cur.execute("""SELECT \* FROM menu WHERE price %s %s"""**

**%(operation, value))**

**>>> results = cur.fetchall()**

**>>> for record in results:**

**… print record[0], ". ", record[1], "(%s)" %record[2]**

**… 1 . tuna (7.50)**

As you may have surmised by now, the execute() method is simply passing the MySQL statement as a string to \_mysql, which in turn passes it to the C database API, which in turn passes it to MySQL. This being the case, we can define the statement separately and pass it to execute() as a variable. Consider the following replacement for the latter half of the preceding code.

Como ya habrás adivinado, el método ejecutar() simplemente pasa la instrucción MySQL como una cadena a \_mysql, que a su vez la pasa a la API de la base de datos C, que a su vez la pasa a MySQL. Siendo este el caso, podemos definir la declaración por separado y pasarla a ejecutar() como una variable. Considere el siguiente reemplazo para la segunda mitad del código anterior.

**>>> statement = """SELECT \* FROM menu WHERE price %s %s""" %(operation,**

**value)**

**>>> command = cur.execute(statement)**

**>>> results = cur.fetchall()**

**>>> for record in results:**

**… print record[0], ". ", record[1], "(%s)" %record[2]**

**… 1 . tuna (7.50)**

**Determining characteristics of a database and its tables**

For reasons of security, one simply must not rely on the user to know the database structure in order to make a query. Even if the user does, one should never write code that assumes this. You never know who the user is and what nefarious results will be sought by the user. Given that there are more people than you know who want to break your code and exploit your data, it is best practice to restrict the user's knowledge of the database and to verify the integrity of any data the end user inputs into the program.

Por razones de seguridad, simplemente no se debe confiar en que el usuario conozca la estructura de la base de datos para realizar una consulta. Incluso si el usuario lo hace, nunca se debe escribir código que asuma esto. Nunca se sabe quién es el usuario y qué resultados nefastos buscará. Dado que hay más personas de las que usted conoce que quieren descifrar su código y explotar sus datos, es una buena práctica restringir el conocimiento del usuario sobre la base de datos y verificar la integridad de cualquier dato que el usuario final ingrese al programa.

Without doubt, the best way to restrict the user's knowledge of the database is to provide set options for the user in a way that the user cannot edit. In graphical user interfaces (GUIs), this is done most often by drop-down menus and radio buttons. In terminal-based programs, one lists the options. The former keeps the programmer in control of the environment and so funnels the user to the point of either choosing the set options or not using the application. In the case of a terminal-based program or in the case of the text boxes of a GUI, one still has to evaluate the data input by the user. Otherwise, a mishandled error opens the system up for technological vandalism or even burglary.

Sin duda, la mejor manera de restringir el conocimiento del usuario sobre la base de datos es proporcionar opciones establecidas para el usuario de una manera que el usuario no pueda editar. En las interfaces gráficas de usuario (GUI), esto se realiza con mayor frecuencia mediante menús desplegables y botones de opción. En los programas basados en terminal, se enumeran las opciones. El primero mantiene al programador en control del entorno y así canaliza al usuario hasta el punto de elegir las opciones establecidas o no utilizar la aplicación. En el caso de un programa basado en terminal o en el caso de los cuadros de texto de una GUI, todavía hay que evaluar los datos introducidos por el usuario. De lo contrario, un error mal manejado abre el sistema al vandalismo tecnológico o incluso al robo.

To evaluate data input from the user, one typically identifies parameters for the variable installed and then validates the input through a series of conditionals. Such parameters can include criteria such as string length, variable type, alphabet characters only, alphanumeric characters, or others. If the data fails anywhere along the way, the program prints a customized error message to the user. The error message is not the message thrown by Python or other, ancillary process. Rather, it is the message given when that error message is detected. The user is then directed to do a given action—contacting their administrator, changing their input, and so on.

Para evaluar la entrada de datos del usuario, normalmente se identifican los parámetros para la variable instalada y luego se valida la entrada mediante una serie de condicionales. Dichos parámetros pueden incluir criterios como longitud de cadena, tipo de variable, caracteres alfabéticos únicamente, caracteres alfanuméricos u otros. Si los datos fallan en algún punto del camino, el programa imprime un mensaje de error personalizado para el usuario. El mensaje de error no es el mensaje arrojado por Python u otro proceso auxiliar. Más bien, es el mensaje que se da cuando se detecta ese mensaje de error. Luego se le indica al usuario que realice una acción determinada: contactar a su administrador, cambiar su entrada, etc.

The scenario works well for most cases. There are, however, instances in database-driven applications where one must implement more advanced measures. For example, if you had several tables that could be searched, you would not necessarily want to have a different program for each one. Instead, it makes better sense to abstract the problem of search to where the same search function can be applied to any of the tables at the user's discretion. The problem breaks down as follows:

El escenario funciona bien en la mayoría de los casos. Sin embargo, hay casos en aplicaciones basadas en bases de datos en los que es necesario implementar medidas más avanzadas. Por ejemplo, si tuviera varias tablas en las que se pudieran buscar, no necesariamente querrá tener un programa diferente para cada una. En cambio, tiene más sentido abstraer el problema de la búsqueda hasta el punto en que la misma función de búsqueda pueda aplicarse a cualquiera de las tablas a discreción del usuario. El problema se desglosa de la siguiente manera:

1. Determine what tables exist

2. Assign a number to each one for a terminal-based program

3. Offer the options to the user

4. Allow the user to detail a search query

5. Ensure that the data input for the table is one of the options

6. Run the query

7. Print the results

**Determining what tables exist**

In a MySQL session, the tables of a database are available through the following command:

En una sesión MySQL, las tablas de una base de datos están disponibles a través del siguiente comando:

**SHOW TABLES in <database name>;**

This allows you to specify a different database from that which you are using at the time.

If you specify the database to be used in the argument to MySQLdb.connect(), then you do not need to specify the database name.

Si especifica la base de datos que se utilizará en el argumento de MySQLdb.connect(), entonces no necesita especificar el nombre de la base de datos.

In Python, we pass the SHOW TABLES statement to execute() and process the returned data.

>>> statement = """SHOW TABLES"""

>>> command = cur.execute(statement)

>>> results = cur.fetchall()

Previously, here we would iterate over the results and output the parts we want. Instead, we will initiate a list and append the table names to it.

Anteriormente, aquí iterábamos sobre los resultados y generamos las partes que queremos. En su lugar, iniciaremos una lista y le agregaremos los nombres de las tablas.

**>>> table\_list = []**

**>>> for record in results:**

**… table\_list.append(record[0])**

**Assigning each table a number**

While we detailed a necessary part of the pseudocode in the last section, it is not wholly necessary for us in this process. Using a list, we can access each item discretely without having to give it a number. However, in order to ensure that the process is plain to see, we could process the list into a dictionary, using the item's ordinal place plus one as the key value.

Si bien detallamos una parte necesaria del pseudocódigo en la última sección, no es del todo necesaria para nosotros en este proceso. Mediante una lista podemos acceder a cada elemento de forma discreta sin tener que darle un número. Sin embargo, para garantizar que el proceso sea fácil de ver, podríamos procesar la lista en un diccionario, utilizando el lugar ordinal del elemento más uno como valor clave.

**>>> item\_dict = {}**

**>>> for item in xrange(1,len(table\_list)):**

**… item\_dict[item-1] = table\_list[item-1]**

The effect is that the first item of the list, which normally is accessed with subscript 0, is assigned the key value 1. All other tables are handled similarly, and we are set to scale to any number of tables in the database.

El efecto es que al primer elemento de la lista, al que normalmente se accede con el subíndice 0, se le asigna el valor clave 1. Todas las demás tablas se manejan de manera similar y estamos configurados para escalar a cualquier número de tablas en la base de datos.

**Offering the options to the user**

Offering the options to the user, we simply print out the key value as the indicator for the table name.

Al ofrecer las opciones al usuario, simplemente imprimimos el valor clave como indicador para el nombre de la tabla.

**>>> for key in item\_dict:**

**… print "%s => %s" %(key, item\_dict[key])**

**>>> choice = input("Please enter your choice of table to be queried. ")**

To verify the input, we would then check that the value is an integer within the same range as those offered. As shown below, however, we can also code the program to be self-validating. Note, however, that this should be complementary to proper security checks. One never knows with what haste the next person who edits the code will approach the task.

Para verificar la entrada, verificaríamos que el valor sea un número entero dentro del mismo rango que los ofrecidos. Sin embargo, como se muestra a continuación, también podemos codificar el programa para que sea autovalidante. Sin embargo, tenga en cuenta que esto debería complementar los controles de seguridad adecuados. Nunca se sabe con qué rapidez abordará la tarea la siguiente persona que edite el código.

**Allowing the user to detail a search query**

Now, the user can be allowed to input a value for which he or she would like to search. As shown previously, this can be any value in the database. However, realistically speaking, we need to give structure to the choice-making process. We can do this at the same time that we validate the user's choice of database by requesting of MySQL the names of the columns for the given table.

Ahora, se le puede permitir al usuario ingresar un valor que le gustaría buscar. Como se mostró anteriormente, este puede ser cualquier valor en la base de datos. Sin embargo, siendo realistas, debemos estructurar el proceso de toma de decisiones. Podemos hacer esto al mismo tiempo que validamos la elección de base de datos del usuario solicitando a MySQL los nombres de las columnas de la tabla dada.

**>>> try: table\_choice = item\_dict[choice]**

**... except: print 'Invalid input. Please try again.'**

If the user's choice reconciles with item\_dict, then we get the name of the table to search. We can then ask MySQL for Python to return the column headings of that table.

Si la elección del usuario coincide con item\_dict, obtenemos el nombre de la tabla a buscar. Luego podemos pedirle a MySQL que Python devuelva los encabezados de las columnas de esa tabla.

**>>> statement = """DESCRIBE %s""" %item\_dict[choice]**

**>>> command = cur.execute(statement)**

**>>> results = cur.fetchall()**

**>>> column\_list = []**

**>>> for record in results:**

**...**

**column\_list.append(record[0])**

With the column names in a list, we can offer them to the user in the same way as the table names.

Con los nombres de las columnas en una lista, podemos ofrecerlos al usuario de la misma forma que los nombres de las tablas.

**>>> for i in xrange(0, len(column\_list)):**

**… print "%s.**

**%s" %(i+1, column\_list[i])**

...

1.id

2.name

3.price

>>> table\_choice = input("Please input the number of the table you wish

to query. ")

[ 44 ]Chapter 2

Once again, we would check that the value entered is an integer within the range offered. This can be affected with a try-except-else statement:

Una vez más comprobaríamos que el valor introducido es un número entero dentro del rango ofrecido. Esto puede verse afectado con una declaración try-except-else:

while True:

try:

if column\_choice > 0:

if column\_choice < len(column\_list):

continue

else:

break

else:

break

except:

print "Invalid input. Please try again."

else:

break

From here one would then solicit the search query from the user and submit it to MySQL.

Desde aquí, se solicitaría la consulta de búsqueda al usuario y se la enviaría a MySQL.

**Changing queries dynamically**

But what if the user does not want to submit a precise query but needs a list of the possibilities? There are a couple of ways to clarify the search. We could first keep a list of the common search queries. This is something done often by the likes of Google and Yahoo!. This works very well with large datasets served through web servers because it uses a static list of terms and simply culls them out. For more dedicated applications, one can use MySQL's pattern matching ability to present known options on-the-fly.

Pero ¿qué pasa si el usuario no quiere enviar una consulta precisa pero necesita una lista de posibilidades? Hay un par de formas de aclarar la búsqueda. Primero podríamos mantener una lista de las consultas de búsqueda comunes. Esto es algo que hacen a menudo empresas como Google y Yahoo!. Esto funciona muy bien con grandes conjuntos de datos servidos a través de servidores web porque utiliza una lista estática de términos y simplemente los elimina. Para aplicaciones más dedicadas, se puede utilizar la capacidad de coincidencia de patrones de MySQL para presentar opciones conocidas sobre la marcha.

**Pattern matching in MySQL queries**

Where Python's regular expression engine is very robust, MySQL supports the two following metacharacters for forming regular expressions:

Mientras que el motor de expresiones regulares de Python es muy robusto, MySQL admite los dos metacaracteres siguientes para formar expresiones regulares:

* %: Zero or more characters matched in aggregate
* \_: Any single character matched individually

Pattern matching is always a matter of comparison. Therefore, with either of these, never use operators of equality.

La combinación de patrones es siempre una cuestión de comparación. Por lo tanto, con cualquiera de estos, nunca utilice operadores de igualdad.

**SELECT \* FROM menu WHERE name = 's%';**

**SELECT \* FROM menu WHERE name <> 's%';**

Instead, use the keywords LIKE and NOT LIKE.

**SELECT \* FROM menu WHERE name LIKE 's%';**

**SELECT \* FROM menu WHERE name NOT LIKE 's%';**

**RIGHT**

**RIGHT**

Using metacharacters, one can match records using very irregular terms. Some of the possible combinations follow below:

Usando metacaracteres, se pueden hacer coincidir registros usando términos muy irregulares. Algunas de las posibles combinaciones son las siguientes:

* s%: A value that begins with the letter s
* %s: A value that ends with the letter s
* %s%: A value that contains the letter s
* s%l: A value that begins with s and ends with l
* s%l%: A value that begins with s and contains at least one instance of
* s\_l%: A value that begins with s and whose third letter is l
* \_\_\_\_\_: A five letter value (that is five underscore characters in succession)
* \_\_%: A value with at least two characters

the letter l

**Putting it into practice**

For a smaller dataset or even larger datasets served over low-contest or no-contest connections (for example local servers or dedicated LAN connections), there is the option of running a live query to present the user with the possible options. If the user has specified the database and table to be used, as in the example seen previously, then it is a small matter to match patterns in a column using LIKE and a regular expression.

Para un conjunto de datos más pequeño o incluso conjuntos de datos más grandes servidos a través de conexiones de bajo concurso o sin concurso (por ejemplo, servidores locales o conexiones LAN dedicadas), existe la opción de ejecutar una consulta en vivo para presentar al usuario las opciones posibles. Si el usuario ha especificado la base de datos y la tabla que se utilizarán, como en el ejemplo visto anteriormente, entonces es un asunto menor hacer coincidir patrones en una columna usando LIKE y una expresión regular.

The MySQL sentence for what we are doing, along with its results, is as follows:

La sentencia de MySQL para lo que estamos haciendo, junto con sus resultados, es la siguiente:

mysql> SELECT name FROM menu WHERE name LIKE 's%';

+--------+

| name

|

+--------+

| salmon |

| sole

|

+--------+

2 rows in set (0.00 sec)

[ 46 ]Chapter 2

It is important to phrase the query in such a way as to narrow the returned values as much as possible. Here, instead of returning whole records, we tell MySQL to return only the namecolumn. This natural reduction in the data reduces processing time for both MySQL and Python. This saving is then passed on to your server in the form of more sessions able

to be run at one time.

Es importante formular la consulta de tal manera que se reduzcan los valores devueltos tanto como sea posible. Aquí, en lugar de devolver registros completos, le decimos a MySQL que devuelva sólo la columna de nombre. Esta reducción natural de los datos reduce el tiempo de procesamiento tanto para MySQL como para Python. Este ahorro luego se transmite a su servidor en forma de más sesiones que se pueden ejecutar al mismo tiempo.

In Python, the preceding statement would look like this:

column = 'name'

term = 's%'

statement = """select %s from menu where name like '%s'""" %(column,

term)

Using the conversion specifier (%s), this code can easily be adapted for more dynamic uses.

Having restricted the parameters of the search, we are in greater control of the results and can therefore anticipate the number of fields in each record returned. We then have to execute the query and tell the cursor to fetch all of the records. To process the records, we iterate over them using a pattern similar to what we used previously:

Al restringir los parámetros de la búsqueda, tenemos un mayor control de los resultados y, por lo tanto, podemos anticipar el número de campos en cada registro devuelto. Luego tenemos que ejecutar la consulta y decirle al cursor que busque todos los registros. Para procesar los registros, los iteramos usando un patrón similar al que usamos anteriormente:

**command = cur.execute(statement)**

**results = cur.fetchall()**

**column\_list = []**

**for record in results:**

**column\_list.append(record[0])**

**print "Did you mean:"**

**for i in xrange(0, len(column\_list)):**

**print "%s. %s" %(i+1, column\_list[i])**

**option = raw\_input ('Number:')**

**intoption = int(option)**

The results for this code are:

Did you mean:

1.salmon

2.sole

Number:

Naturally, we must then test the user input. After that, we can process the query and return the results.

Naturalmente, luego debemos probar la entrada del usuario. Después de eso, podemos procesar la consulta y devolver los resultados.

This example is shown using terminal options so we do not use any JavaScript to transfer the options. However, in modern day reality, any application that relies on a web browser—either for background processing or for a primary interface, can use this code with minor modifications.

Este ejemplo se muestra usando opciones de terminal, por lo que no usamos JavaScript para transferir las opciones. Sin embargo, en la realidad moderna, cualquier aplicación que dependa de un navegador web, ya sea para procesamiento en segundo plano o para una interfaz principal, puede utilizar este código con modificaciones menores.

**Project: A command-line search utility**

This chapter has been about querying MySQL from Python. As a project to finish it out, we will build a command-line search utility. Rather than ask the user for the search term, we will expect the user to state the term as an argument for the command-line invocation.

Este capítulo trata sobre cómo consultar MySQL desde Python. Como proyecto para finalizarlo, crearemos una utilidad de búsqueda de línea de comandos. En lugar de preguntarle al usuario el término de búsqueda, esperaremos que el usuario indique el término como argumento para la invocación de la línea de comandos.

With a bit more code for this project, we could create a GUI for this program. GUI programming increases the complexity of the code. How much more complex it gets depends on the library being used, but it is nonetheless unnecessary for what we need to illustrate in this project. Until we are certain that we have a database and can connect to it, it is

best to keep it simple.

Con un poco más de código para este proyecto, podríamos crear una GUI para este programa. La programación GUI aumenta la complejidad del código. Cuánto más complejo se vuelve depende de la biblioteca que se utilice, pero, no obstante, es innecesario para lo que necesitamos ilustrar en este proyecto. Hasta que estemos seguros de que tenemos una base de datos y podemos conectarnos a ella, es Lo mejor es mantenerlo simple.

Now, it is true that we could simply take input and feed it through MySQL for Python as a generic SELECT \* statement. The logic for this bare bones implementation has been illustrated previously to a great extent. We can create something a bit more sophisticated.

Ahora bien, es cierto que podríamos simplemente tomar información y enviarla a través de MySQL para Python como una declaración SELECT \* genérica. La lógica de esta implementación básica se ha ilustrado en gran medida anteriormente. Podemos crear algo un poco más sofisticado.

The following characteristics should apply to our search utility:

* The user calls it from the command line (that is, shell prompt)
* The search term is defined at the time of calling
* If the -t flag is issued, the following term is the table to be used; default is to search all tables
* If the -f flag is issued, the output is formatted by table
* If the -o flag is issued, the output is written to the given file

To illustrate, calling the application searchme.py, one should be able to call the application as follows:

Para ilustrar, al llamar a la aplicación searchme.py, uno debería poder llamar a la aplicación de la siguiente manera:

**./searchme.py -t menu -f -o output.txt query**

This should search the table menu from the database fish for the term query, format the output, and write it to a file called output.txt. You may need to nuance this statement depending on your operating system's requirements. For example, Windows users should not include ./ before the program call. For more help with calling Python on Windows, see the Python Windows FAQ at the following URL: http://www.python.org/doc/faq/windows/.

Esto debería buscar el término consulta en el menú de la tabla de la base de datos, formatear la salida y escribirla en un archivo llamado salida.txt. Es posible que deba matizar esta afirmación según los requisitos de su sistema operativo. Por ejemplo, los usuarios de Windows no deben incluir ./ antes de la llamada del programa. Para obtener más ayuda sobre cómo llamar a Python en Windows, consulte las preguntas frecuentes sobre Python para Windows en la siguiente URL: <http://www.python.org/doc/faq/windows/>.

**Preparing a database for searching**

For this project, however, we will leave behind the fish database for the moment and use the world sample database available from MySQL. For users of Linux, Unix, and Mac OS X, download the database from the following URL:

Para este proyecto, sin embargo, dejaremos de lado la base de datos de peces por el momento y usaremos la base de datos de muestra mundial disponible en MySQL. Para usuarios de Linux, Unix y Mac OS X, descargue la base de datos desde la siguiente URL:

http://downloads.mysql.com/docs/world.sql.gz

To unpack this archive, simply issue the appropriate unpacking command:

gunzip world.sql.gz

Windows users, or users without the utility gunzip, should use the ZIP file:

http://downloads.mysql.com/docs/world.sql.zip

Then use an archive manager to unpack the ZIP file.

Regardless of your platform, you should then be left with a file world.sql. From the directory in which that file resides, log into MySQL. You first need to create a database world and then import the file.

Independientemente de su plataforma, debería quedarse con un archivo world.sql. Desde el directorio en el que reside ese archivo, inicie sesión en MySQL. Primero debe crear un mundo de base de datos y luego importar el archivo.

1 To create the database, type the following at the prompt:

**CREATE world;**

2 Then tell MySQL to use that database:

**USE world;**

3 Import the file with the following MySQL command:

**SOURCE world.sql;**

MySQL will then populate the database world with three tables of data: City, Country, and CountryLanguage.

Luego, MySQL poblará el mundo de la base de datos con tres tablas de datos: ciudad, país y idioma del país.

**Planning your work, then working your plan**

All of the flags mentioned as characteristics previously are nothing more than added functionality to the core function of searching. Our first order of business should therefore be to create well-abstracted search functionality. Then we can build on it to allow for the functionality of each of the given flags. So our plan of development may be summarized as follows:

Todas las banderas mencionadas anteriormente como características no son más que funcionalidades agregadas a la función principal de búsqueda. Por lo tanto, nuestra primera tarea debería ser crear una funcionalidad de búsqueda bien abstracta. Luego podemos aprovecharlo para permitir la funcionalidad de cada una de las banderas dadas. Así, nuestro plan de desarrollo se puede resumir de la siguiente manera:

* Develop a well-abstracted search functionality.
* Implement specification of the search term from the command-line.
* Implement and incorporate the functionality of each flag in turn:

-t, -f, and –o.

**Develop a well-abstracted search functionality**

Abstraction is the secret of all computer programming. It is what enables computer programs to be run for more than one single task. In the early days of computing, programs were written with very narrow applications. This was usually due to limits in the technology involved. In the modern day, the obtuseness of some languages still keeps them from being applied beyond certain domains (for example, BIOS systems). Even then, languages such as Forth are so difficult to follow that they are largely viewed as write-once-and-forget-about-it. Fortunately, Python offers us significant flexibility, and this flexibility is carried through in MySQL for Python. This allows us to create the infrastructure of a search while allowing us to specify

select parts of it.

La abstracción es el secreto de toda programación informática. Es lo que permite ejecutar programas informáticos para más de una única tarea. En los primeros días de la informática, los programas se escribían con aplicaciones muy limitadas. Esto generalmente se debía a limitaciones en la tecnología involucrada. En la actualidad, la obtusidad de algunos lenguajes todavía impide que se apliquen más allá de ciertos dominios (por ejemplo, los sistemas BIOS). Incluso entonces, lenguajes como Forth son tan difíciles de seguir que en gran medida se los considera como escribir una vez y olvidarse. Afortunadamente, Python nos ofrece una flexibilidad significativa, y esta flexibilidad se aplica en MySQL para Python. Esto nos permite crear la infraestructura de una búsqueda y al mismo tiempo especificar partes seleccionadas de la misma.

In this project, we will specify the host, database, and user information for the database connection. The rest of the query information, however, will be open to the user. First, however, we begin our program with a shebang (line):

En este proyecto, especificaremos el host, la base de datos y la información del usuario para la conexión de la base de datos. El resto de la información de la consulta, sin embargo, estará abierta al usuario. Sin embargo, primero comenzamos nuestro programa con un shebang (línea):

**#!/usr/bin/env python**

This is a Linux shebang line that calls whichever Python interpreter is set for general use in the environmental variables of the shell. If we want a specific interpreter, we can naturally edit this and call the other interpreter directly. This format will also work on newer versions of Mac OS. If you are on Windows, you will naturally have to modify this line according to the directory structure of your Windows installation.

Esta es una línea shebang de Linux que llama a cualquier intérprete de Python configurado para uso general en las variables ambientales del shell. Si queremos un intérprete específico, naturalmente podemos editarlo y llamar al otro intérprete directamente. Este formato también funcionará en versiones más recientes de Mac OS. Si está en Windows, naturalmente tendrá que modificar esta línea de acuerdo con la estructura de directorios de su instalación de Windows.

A common Windows shebang line is:

Una línea común de Windows Shebang es:

**#!c:/Python/python.exe -u**

Ensure that you do not forget the trailing -u flag. This puts Python into an unbuffered mode on Windows systems.

Asegúrese de no olvidar el indicador -u final. Esto pone a Python en modo sin búfer en sistemas Windows.

Next, we import MySQL for Python. Until we are ready to add more functionality beyond a hard-coded search query, we should hold off on importing more modules.

A continuación, importamos MySQL para Python. Hasta que estemos listos para agregar más funciones más allá de una consulta de búsqueda codificada, debemos posponer la importación de más módulos.

**import MySQLdb**

Now we are ready to create a database connection. You will recall from the first chapter that the format for creating a database connection is:

Ahora estamos listos para crear una conexión a la base de datos. Recordará del primer capítulo que el formato para crear una conexión de base de datos es:

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

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

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

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

For the world database, using user user and password p4ssw0rd on the localhost, the invocation for the connection is:

Para la base de datos mundial, utilizando usuario usuario y contraseña p4ssw0rd en el host local, la invocación para la conexión es:

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

**user = 'user',**

**passwd = 'p4ssw0rd',**

**db = 'world')**

We then must create the cursor:

**cur = mydb.cursor()**

We are then ready to construct a query infrastructure. To ensure as much flexibility as possible, we simply pull all of the variables out of the MySQL SELECT syntax and define them separately.

Entonces estamos listos para construir una infraestructura de consulta. Para garantizar la mayor flexibilidad posible, simplemente extraemos todas las variables de la sintaxis SELECT de MySQL y las definimos por separado.

**table =**

**'City'**

**column = 'Name'**

**term = 's%'**

**statement = """select \* from %s where %s like '%s'""" %(table, column,**

**term**

This hardwiring of the search query allows us to test the connection before coding the rest of the function. By defining the variables discretely, we make it easier to change them to user-determined variables later.

Este cableado de la consulta de búsqueda nos permite probar la conexión antes de codificar el resto de la función. Al definir las variables de forma discreta, facilitamos su cambio posterior a variables determinadas por el usuario.

There may be a tendency here to insert user-determined variables immediately. With experience, it is possible to do this. However, if there are any doubts about the availability of the database, your best fallback position is to keep it simple and hardwired. This reduces the number of variables in making a connection and helps one to blackbox the situation, making troubleshooting much easier.

Puede haber una tendencia aquí a insertar inmediatamente variables determinadas por el usuario. Con experiencia es posible hacerlo. Sin embargo, si hay dudas sobre la disponibilidad de la base de datos, la mejor opción alternativa es mantenerla simple y estructurada. Esto reduce la cantidad de variables al establecer una conexión y ayuda a bloquear la situación, lo que facilita mucho la resolución de problemas.

With the query constructed, we can execute it and get the results:

**command = cur.execute(statement)**

**results = cur.fetchall()**

You can then test the connection with the following code:

**record\_list = []**

**for record in results:**

**record\_list.append(record[0])**

**for i in xrange(0, len(record\_list)):**

**print "%s.**

**%s" %(i+1, record\_list[i])**

The logic of this code is discussed previously in this chapter so we will pass over it here as understood.

La lógica de este código se analizó anteriormente en este capítulo, por lo que aquí la pasaremos por alto tal como se entiende.

If you execute the program, the output should scroll off the screen and you should get a list of 431 cities. The last entry should read as follows:

Si ejecuta el programa, el resultado debería desplazarse fuera de la pantalla y debería obtener una lista de 431 ciudades. La última entrada debería decir lo siguiente:

431. Santa Monica

Knowing that we can interact with the database, we can now go to the next step, specifying the search term from the command-line.

Sabiendo que podemos interactuar con la base de datos, ahora podemos pasar al siguiente paso, especificando el término de búsqueda desde la línea de comandos.

**Specifying the search term from the command-line**

Python allows you to receive a command-line argument using the sys module. If the only argument expected on the command-line is the name of the query, we could get by with code like this:

Python le permite recibir un argumento de línea de comandos usando el módulo sys. Si el único argumento esperado en la línea de comandos es el nombre de la consulta, podríamos arreglárnoslas con un código como este:

**import sys**

**...**

**query = sys.argv[1]**

Alas, life is seldom so simple. If we were to follow this route of development, all the flags mentioned previously for this program would be mandatory and have to be submitted every time the program ran. So, for the sample call on page 24, we would have to program for six fixed arguments everytime the program is called. Recall that the command read:

Por desgracia, la vida rara vez es tan sencilla. Si siguiéramos esta ruta de desarrollo, todas las banderas mencionadas anteriormente para este programa serían obligatorias y tendrían que enviarse cada vez que se ejecutara el programa. Entonces, para la llamada de ejemplo en la página 24, tendríamos que programar para seis argumentos fijos cada vez que se llama al programa. Recuerde que el comando decía:

**./searchme.py -t menu -f -o output.txt query**

The arguments for sys.argv that would be required are:

•0 the command itself, naturally

•1 the flag -t

•2 the table name

•3 the -f flag

•4 the flag -o

•5 the output file name

•6 the search string for the query

If we pulled all the flags and left only the arguments in a set order, we would still have three arguments. This makes the program cumbersome and makes calls to it error-prone.

Si quitáramos todas las banderas y dejáramos solo los argumentos en un orden establecido, todavía tendríamos tres argumentos. Esto hace que el programa sea engorroso y que las llamadas sean propensas a errores.

By far, the better way forward is to use the module optparse. Importing optparse incorporates high-level support for processing arguments at runtime. Another alternative is the getopt module, but this is not as robust or as easy to use.

Con diferencia, la mejor manera de avanzar es utilizar el módulo optparse. La importación de optparse incorpora soporte de alto nivel para procesar argumentos en tiempo de ejecución. Otra alternativa es el módulo getopt, pero no es tan robusto ni tan fácil de usar.

For consistency's sake, let's edit the preamble of our Python program to import the module.

En aras de la coherencia, editemos el preámbulo de nuestro programa Python para importar el módulo.

**import optparse**

The optparse module provides an OptionParser class that handles both option processing and error handling. It also provides a standardized interface similar to other programs (for example, -h for help). All for the meager cost of a module import.

El módulo optparse proporciona una clase OptionParser que maneja tanto el procesamiento de opciones como el manejo de errores. También proporciona una interfaz estandarizada similar a otros programas (por ejemplo, -h para ayuda). Todo por el escaso coste de la importación de un módulo.

To access all this option parsing goodness, we need to instantiate an object of the OptionParser class.

Para acceder a todas estas bondades del análisis de opciones, necesitamos crear una instancia de un objeto de la clase OptionParser.

**opt = optparse.OptionParser()**

Adding arguments then is simply a matter of adding options using the add\_option() method of the object. For our purposes, we need to follow a straightforward syntactic formula:

Entonces, agregar argumentos es simplemente una cuestión de agregar opciones usando el método add\_option() del objeto. Para nuestros propósitos, debemos seguir una fórmula sintáctica sencilla:

**object.add\_option("-[short flag option]", "--[long flag option]",**

**action="store", type="string", dest = "[variable name under which to**

**store the option]"**

We will keep to this formula for the sake of simplicity even when other options might seem more logical.

Mantendremos esta fórmula en aras de la simplicidad incluso cuando otras opciones puedan parecer más lógicas.

The optparse module is a very robust kit for option parsing, and the full syntax of it is beyond the scope of this book. A fuller discussion can be found in the online help() function or the Python documentation.

El módulo optparse es un kit muy sólido para el análisis de opciones y su sintaxis completa está más allá del alcance de este libro. Puede encontrar una discusión más completa en la función de ayuda en línea () o en la documentación de Python.

So to add an option for the query, we include this:

Entonces, para agregar una opción para la consulta, incluimos esto:

**opt.add\_option("-q", "--query", action="store", type="string",**

**dest="term")**

After all the options are assigned in the code, we then have to compile them.

Una vez asignadas todas las opciones en el código, tenemos que compilarlas.

opt, args = opt.parse\_args()

The parse\_args() method returns a tuple. The first of the two values, opt, is an object containing the values of all the options passed to the program. The second value, args, is a list of any remaining arguments.

El método parse\_args() devuelve una tupla. El primero de los dos valores, opt, es un objeto que contiene los valores de todas las opciones pasadas al programa. El segundo valor, args, es una lista de los argumentos restantes.

These last three lines should precede the MySQLdb call to connect to the database. You may then change the assignment of the variable term.

Estas últimas tres líneas deben preceder a la llamada a MySQLdb para conectarse a la base de datos. Luego podrá cambiar la asignación del término variable.

**term = opt.term**

Whatever value is given in the dest argument of the add\_option() method becomes an attribute of the OptionParser object. Therefore, you could also delete the term assignment line and edit the statement value to reflect opt.term. However, this makes the code more difficult to read by someone else or even by yourself six months down the line. Remember the second line of the Zen of Python:

Cualquier valor que se proporcione en el argumento de destino del método add\_option() se convierte en un atributo del objeto OptionParser. Por lo tanto, también puede eliminar la línea de asignación de términos y editar el valor de la declaración para reflejar opt.term. Sin embargo, esto hace que el código sea más difícil de leer para otra persona o incluso para usted mismo seis meses después. Recuerda la segunda línea del Zen de Python:

Explicit is better than implicit

With that code written, you should be able to call the program with a -q flag and set the query from the command-line. If the program is called searchme.py, a sample query would be:

Con ese código escrito, debería poder llamar al programa con un indicador -q y configurar la consulta desde la línea de comandos. Si el programa se llama searchme.py, una consulta de ejemplo sería:

**./searchme.py -q 'dubai'**

The results should be singular:

1. Dubai

Knowing that it is a MySQL database, you could also pass MySQL metacharacters.

Sabiendo que es una base de datos MySQL, también puedes pasar metacaracteres MySQL.

**./project-ch2.py -q 'm%i'**

and get multiple hits in return:

1.Mallawi

2.Makati

3.Marawi

4.Malasiqui

5.Mati

6.Madurai

7.Malkajgiri

8.Morvi

9.Miyazaki

10.Maebashi

11.Moriguchi

12.Manzhouli

13.Moroni

14.Mbuji-Mayi

15.Matadi

16.Mexicali

17.Maradi

18.Maiduguri

19.Makurdi

20.Miaoli

21.Moshi

22.Mytiti

23.Miami

Note that your search will complete and return the same regardless of whether you use -q or --query.

Tenga en cuenta que su búsqueda se completará y devolverá lo mismo independientemente de si usa -q o --query.

**Implementing and incorporating the other functions: -t, -f, and -o**

Now we can add the other options to the program. Like the -q flag for preceding query, we can use -t to assign values for the table.

Ahora podemos agregar las otras opciones al programa. Al igual que el indicador -q para la consulta anterior, podemos usar -t para asignar valores a la tabla.

opt.add\_option("-t", "--table", action="store", type="string",

dest="table")

Unless you want to support multiple formats for the output, the format flag should be a Boolean value.

A menos que desee admitir múltiples formatos para la salida, el indicador de formato debe ser un valor booleano.

**opt.add\_option("-f", "--format", action="store\_true", dest="format")**

In order to store a Boolean variable, we set the default action as either store\_true or store\_false. The former is for setting the value to True when the flag is present; the converse is true for the latter. Because of the type of action applied to the option, the Boolean type of the variable is understood by Python.

Para almacenar una variable booleana, configuramos la acción predeterminada como store\_true o store\_false. El primero sirve para establecer el valor en Verdadero cuando la bandera está presente; lo contrario es cierto para este último. Debido al tipo de acción aplicada a la opción, Python entiende el tipo booleano de la variable.

We then assign the value of opt.table to table and opt.format to the variable format.

For formatting, we then set up a conditional clause that runs if the format flag is present. Given that we are formatting the output on one level, we can also revise the code so that the default is raw. The following code snippet should follow from where the value of record[0] is appended to column\_list, taking the place of that loop as well as the for loop discussed in the section Planning your work, then working your plan (earlier in this chapter).

Para formatear, luego configuramos una cláusula condicional que se ejecuta si el indicador de formato está presente. Dado que estamos formateando la salida en un nivel, también podemos revisar el código para que el valor predeterminado sea sin formato. El siguiente fragmento de código debe seguir desde donde se agrega el valor de registro[0] a column\_list, reemplazando ese bucle así como el bucle for discutido en la sección Planificar su trabajo y luego trabajar su plan (anteriormente en este capítulo). .

**column\_list = []**

**for record in results:**

**column\_list.append(record[0:])**

**if form is True:**

**columns\_query = """DESCRIBE %s""" %(table)**

**columns\_command = cur.execute(columns\_query)**

**headers = cur.fetchall()**

**column\_list = []**

**for record in headers:**

**column\_list.append(record[0])**

**output=""**

**for record in results:**

**output = output + "========================\n\n"**

**for field\_no in xrange(0, len(column\_list)):**

**output = output + column\_list[field\_no]+ ": " +**

**str(record[field\_no]) + "\n"**

**output = output + "\n"**

**else:**

**output=[]**

**for record in xrange(0, len(results)):**

**output.append(results[record])**

**output = ''.join(output)**

Note that the subscript for record in the third line has been broadened to include the rest of the record.

Tenga en cuenta que el subíndice de registro en la tercera línea se ha ampliado para incluir el resto del registro.

At this stage, you can append an output statement at the end of the program to see its results.

En esta etapa, puede agregar una declaración de salida al final del programa para ver sus resultados.

**print output**

We are not yet at the end for this program as we also need to include facility for writing the output to a file of the user's choice.

Aún no hemos llegado al final de este programa ya que también necesitamos incluir la posibilidad de escribir el resultado en un archivo elegido por el usuario.

**Including an option for an output file**

At this point, including an option for an output file is simply a matter of inserting the option into the list at the beginning of the program and then testing for it once output is finally saved. To add the option to the list managed by optparse, we simply insert this line after the last option:

En este punto, incluir una opción para un archivo de salida es simplemente una cuestión de insertar la opción en la lista al comienzo del programa y luego probarla una vez que finalmente se guarde la salida. Para agregar la opción a la lista administrada por optparse, simplemente insertamos esta línea después de la última opción:

**opt.add\_option("-o", "--output", action="store", type="string",**

**dest="outfile")**

Note that the output variable is not a requirement in our program. Therefore, one cannot assign it as automatically as the other variables were assigned. Instead, one must test for it, prefixing the object handle to its attribute. If a value has been assigned, then write the file. Otherwise, default output is STDOUT.

Tenga en cuenta que la variable de salida no es un requisito en nuestro programa. Por lo tanto, no se puede asignarla tan automáticamente como se asignaron las otras variables. En su lugar, se debe probarlo, anteponiendo el identificador del objeto a su atributo. Si se ha asignado un valor, escriba el archivo. De lo contrario, la salida predeterminada es STDOUT.

if opt.outfile:

outfile = opt.outfile

out = open(outfile, w)

out.write(output)

else:

print output

**Room to grow**

While the projection specification that we set for ourselves is fulfilled, there is more that can be done on this program to make it more serviceable in a production environment. Some areas that you might look at for further practice are:

Si bien se cumple la especificación de proyección que nos fijamos, se puede hacer más en este programa para hacerlo más útil en un entorno de producción. Algunas áreas que podrías considerar para practicar más son:

* Set the host, database, username, and password from command-line options. You will naturally need to error-check each of them. You will probably want to use the getpass module to accept the password in production environments.
* Set an option for the column variable and testing whether that column exists in the chosen database.
* Set up error-checking for the results itself.

The world database has a few other tables besides City. Can you aggregate their ecords in Python without using a JOIN statement?

La base de datos mundial tiene algunas otras tabletas además de City. ¿Puedes agregar sus registros en Python sin utilizar una declaración JOIN?

**Summary**

It almost goes without saying that querying and data retrieval is the bread and butter of database programming. In this chapter, we have covered the formation of a MySQL query and how to pass it from Python. We also saw how to use user-defined variables to allow dynamic formation of statements rather than pre-configured queries. In order to adjust our program flow, we also saw how to determine the characteristics of a database and its tables from within Python. Finally, we programmed a command-line search utility that returns data of the user's choice in two different formats.

In the next chapter, we will look at the flipside of the SELECT command, INSERT.