4Python 操作数据库

像电子表格一样,数据库在商务中很常见。公司使用数据库来贮存客户，库存，员工信息。数据库对于追踪运营，销售，财务等很重要。我们将数据库从电子表格或电子表格的工作簿分开，是因为数据库关联的，即一个表格的一行可以关联另一个或多个表格的一行或多行。
举个标准的例子，客户数据--名称，地址等可以关联（用客户 ID）到“订单”表单的一行，订单中包含订购的项目。这些项目关联到“供应商”表单，你可以追踪和执行订单--而且进行深度分析。尽管 CSV 和 Excel 很常见，你可以用 python 自动的处理重要的数据源并放大，构建技巧来处理这些文件，从学习的角度（学习常见的编程操作）和实践的角度（很多的数据贮存在这些文件中），数据库可以真正的利用计算机的能力执行上百倍，上千倍，上百成倍的任务。

关系数据库

这一章讨论关系数据库和关系数据库管理系统(RDBMSs)，其中表格中的信息按预定的关系相互关联，使用键“order ID number” 来关联客户记录到产品记录,货运记录,等等。有时候(通常是“大数据”情况),为运营定义所有的关系不必要或要花费过多的计算工作。则可以用非关系数据库来贮存和查找数据。不是关联客户记录到订单记录在不同的表中，例如，所有的订单可能被贮存在在个记录，包括客户数据和订单子集信息。 (这种情况，可以省下从第二个表中查找客户信息的工作,但是代价是每次客户下订单时都要贮存客户数据的拷贝)。我们不想在这一章处理非关系数据库,但是你要知道 (a)它们是存在的，而且 (b) 用 Python 模块来访问它们。你需要学习如何用 Pythong 与数据库进行交互，如何用数据填充数据库中的表格。如果你还没有访问过数据库和数据表，这个要求可能是绊脚石。幸运的是，有两个资源可以快速和方便的开始并运地本章的例子。首先，Python 有内置的 sqlite3 模块，便以我们创建内存中的数据库。即我们可以用 python 代码创建数据库和数据表并用数据直接填充而不用下载和安装特定的数据库软件。我们在前半部分使用这个特性来快速的开始和运行，以便关注于数据库、数据表、数据的交互而不是下载和安装数据库。第二，你可能已经操作过 MySQL, PostgreSQL, 或 Oracle 等常见的数据库系统。这些数据系统的供应商已使这些系统便于下载和安装。虽然你不必每天操作数据库系统,但是它们在商务中是很常见的，所以熟悉常见的数据库系统的操作，并用 Python 进行操作是很重要的。因此在后半部分我们会下载和安装一个数据库系统。以便你可以利用前半部分学习的知识来与真实的数据库系统交互。

SQL 是什么?

你会注意到我们在本章使用的许多模块和软件带有名字”SQL” (通常读为 “sequel,” 虽然有人坚持读为“es-queue-el”)，是 Structured Query Language 的缩写,广泛地用于与数据库交互的命令。有很多不同的 SQL，你的数据库可能命使用特定的命令和符号，但是有些操作如 SELECT, JOIN, INSERT, 和 UPDATE 是通用的。这一章教你完全用 Python 来构建数据库，并用 SQL 来获取数据供 python 脚本处理。

Python 内置的 qlite3 模块

前面已提过，我们要从 python 内置的 sqlite3 模块开始快速度的创建内存中的数据库和数据表并用 python 代码来填写数据。像第 2 章和第 3 章，第一个例子的重点是展示如何用 SQL 查询来计算行和列。很多情况下这种能力很重要，当你不确定你的查询会输出多少行时，以便你知道你要处理多少行数据。这个例子也是有用的，因为我们用很多与数据库交互的代码，以创建数据库表，插入数据到表中，获取和计算输出的行数。我们可以看到很多代码在后面的例子中重复。

我们开始吧。要创建数据库，插入数据到表，获取和计算输出的行数，在文本编辑器输入如下代码并保存为 1db_count_rows.py：

```
1 #!/usr/bin/env python3
2 import sqlite3
3
4 # Create an in-memory SQLite3 database
5 # Create a table called sales with four attributes
6 con = sqlite3.connect(':memory:')
7 query = """CREATE TABLE sales
8 (customer VARCHAR(20),
9 product VARCHAR(40),
10 amount FLOAT,
11 date DATE);"""
12 con.execute(query)
13 con.commit()
14
15 # Insert a few rows of data into the table
16 data = [('Richard Lucas', 'Notepad', 2.50, '2014-01-02'),
17 ('Jenny Kim', 'Binder', 4.15, '2014-01-15'),
18 ('Svetlana Crow', 'Printer', 155.75, '2014-02-03'),
19 ('Stephen Randolph', 'Computer', 679.40, '2014-02-20')]
20 statement = "INSERT INTO sales VALUES(?, ?, ?, ?)"
21 con.executemany(statement, data)
22 con.commit()
23
24 # Query the sales table
25 cursor = con.execute("SELECT * FROM sales")
26 rows = cursor.fetchall()
27
28 # Count the number of rows in the output
29 row_counter = 0
30 for row in rows:
31 print(row)
32 row_counter += 1
33 print('Number of rows: %d' % (row_counter))
```

图 4-1， 4-2，和 4-3 是脚本在 Anaconda Spyder，Notepad++（Windows），和 TextWrangler（macOS），中看起来的样子。

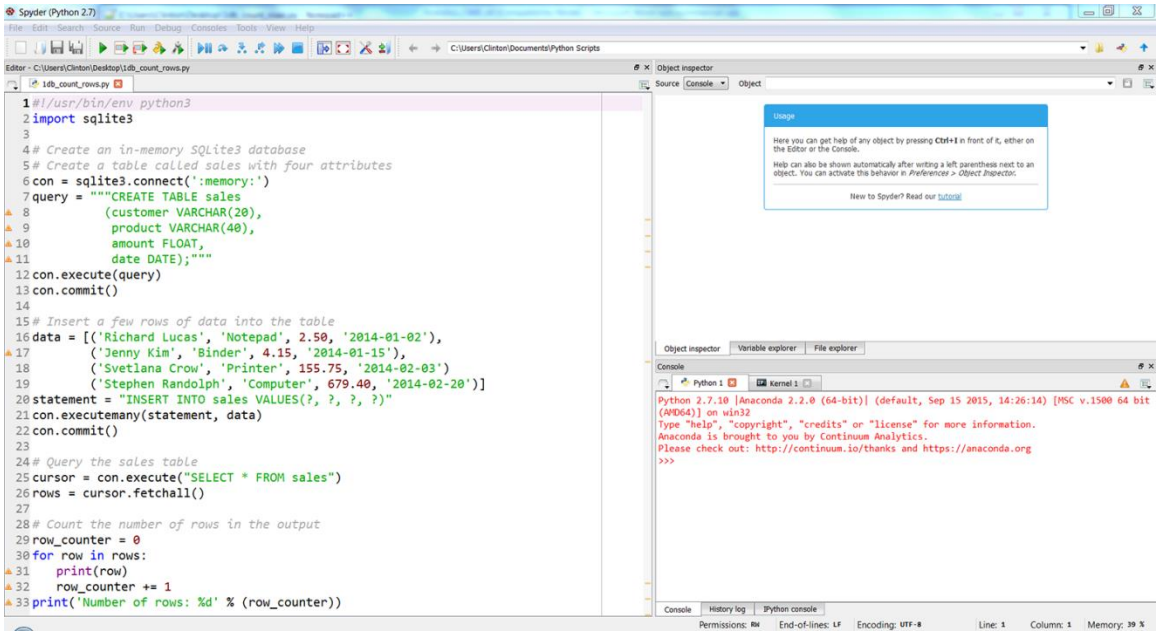


图 4-1. Python 脚本 1db_count_rows.py 在 Anaconda Spyder 中

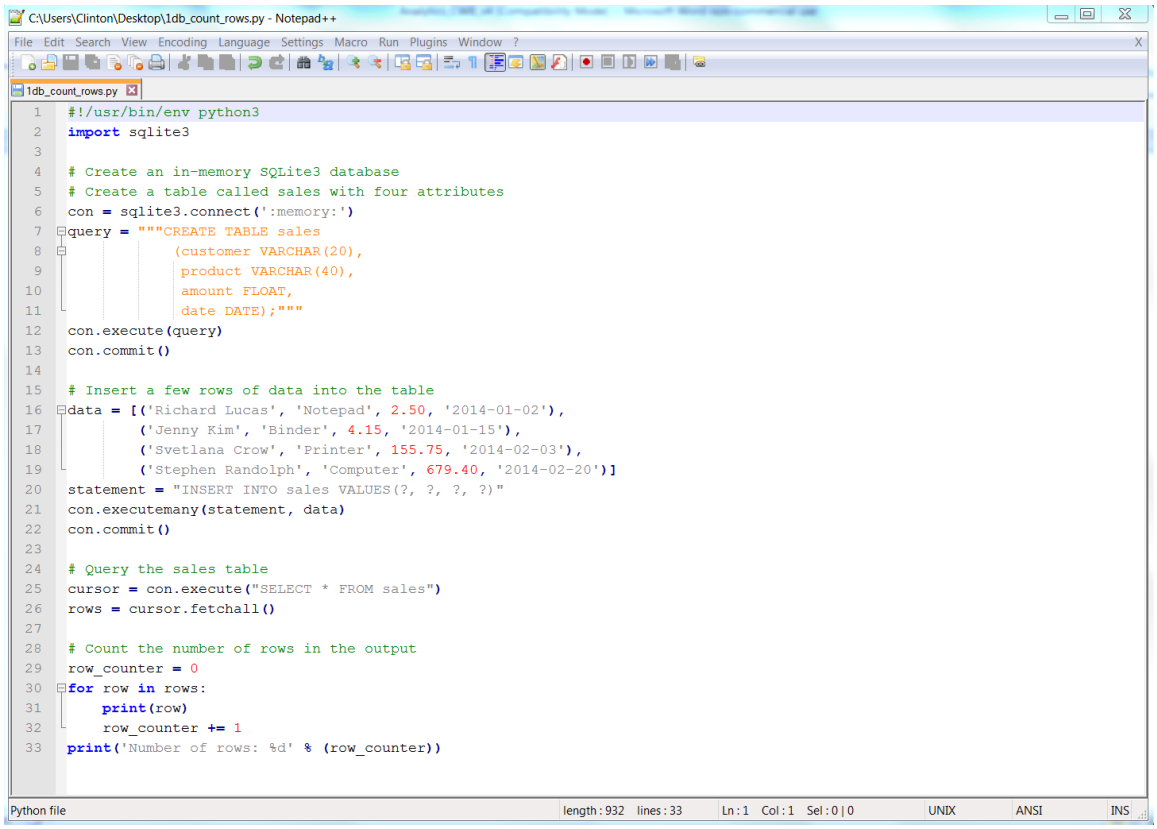


图 4-2. Python 脚本 1db_count_rows.py 在 Notepad++（Windows）中

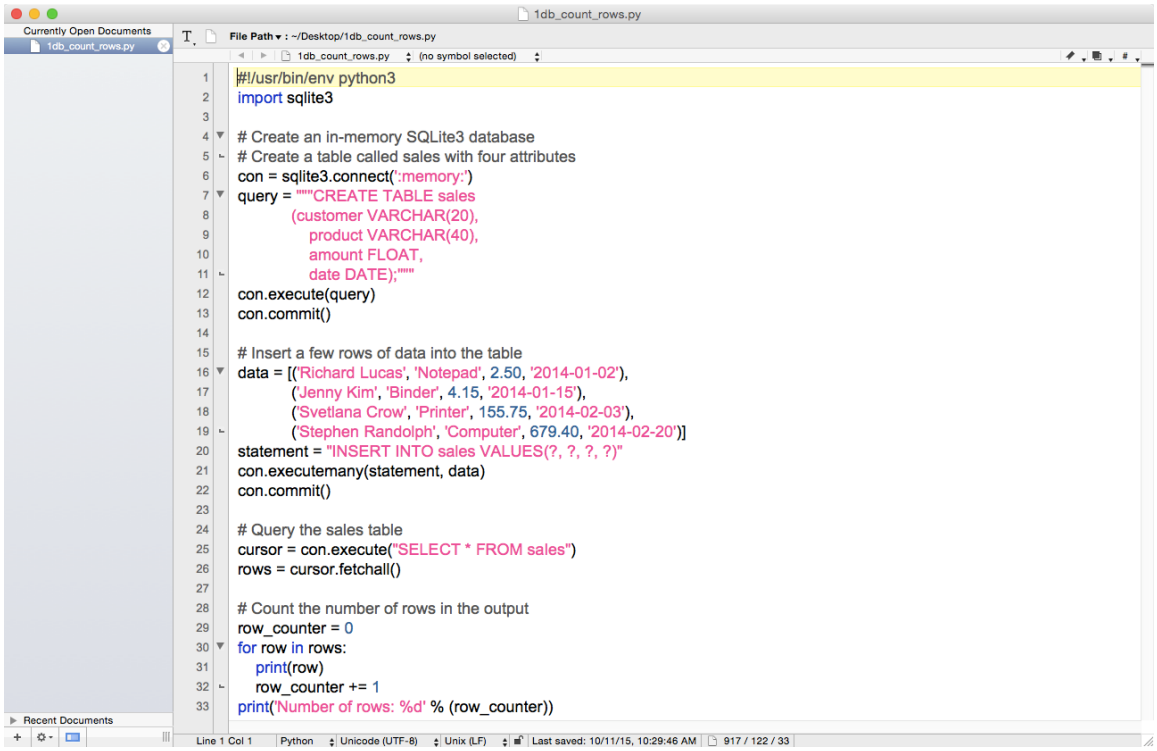


图 4-3. Python 脚本 1db_count_rows.py 在 TextWrangler (macOS)中

在这些图中，你可以看到一些额外的代码需要学习以便与数据库而不是 CSV 或 Excel 文件交互。第 2 行导入 sqlite3 模块，它提供了轻量级的基于磁盘的数据库不需要服务器进程并允许用 SQL 查询语言的变种访问数据库。本例中的 SQL 命令都以大写出现。因为本章是用 python 与数据库交互，本章覆盖常见的 CRUD 的大部分(如 创建,读取,修改，删除）数据库操作。
本例覆盖创建数据库和数据表，插入记录到数据表，修改表中的记录，从表中选择特定的行。这些 SQL 操作在关系数据库中很常见。为了使用这些模块，你首先要创建 connection 对象来代表数据库。第 6 行创建名为 con 的 connection 对象来代表数据库。本例中，我用特别的名字':memory:'来创建内存中的数据库。如果你想数据库持久，你可以提供不同的之串。例如，如果我要用之串'my_database.db'或'C:\Users\Clinton\Desktop\my_database.db' 而不是':memory:'，则数据库对象将持久于我的当前目录或我的桌面。第 7-11 行用三个双引号来创建多行字串并赋值给 query 变量。这个字串是 SQL 命令，它创建名为 sales 的数据表。 sales 表有四个属性: customer, product, amount, 和 date。customer 属性是最长 20 个字符的变量。product 属性最大符号长度为 40。amount 属性是浮点型。date 属性是日期型。第 12 行用 connection 对象的 execute()方法执行 query 变量中的 SQL 命令，在内存数据库中创建 sales 表。第 13 行用 connection 对象的 commit()方法保存对数据库的更改。你总是需要用 commit()方法来保存对数据库的修改,否则你的修改不会保存在数据库中。第 16 行创建元组的列表,将列表赋值给变量 data。每个列表中的元素是包含四个值的元组，三个字串和一个浮点数。这四个值对应于四个表属性（如，表中的四列）。每个元组包含表中的一行。 因为列表包含四个元组，它包含表中的四行数据。

第 20 行与第 7 行相似，它创建字串并将字串赋值给变量 statement。因为字串为一行，它在双引号内而不是三个双引号内，不像第 7 行要管理多行字串。这一行的字串是另一个 SQL 命令，一个 INSERT 语句,我们用来插入 data 中的一行数据到 sales 表。第一次看到这一行你很奇怪使用问号 (?) 的目的。问号是你想要在 SQL 命令中使用的值的容器。然后你提供值的元组给 connection 对象的 execute() 或 executemany()方法，元组中的值被 SQL 命令的位置替代。变种参数替代的方法使你的代码不易被 SQL 注入攻击，它听起来很吓人，比起 SQL 命令字串操作来。

第 21 行用 connection 对象的 executemany()方法为 data 中的每个数据元组执行 statement 中的 SQL 命令。因为 data 中有中个元组数据， executemany()执行 4 次 INSERT 语句，有效的插入 4 行数据到 sales 表。记得讨论第 13 行时我们注意到你总是要用 commit()方法来保存对数据库的修改；否则你的修改不会保存在数据库中。插入 4 行数据到 sales 当然是对数据的修改，所以第 22 行我们再次用 connection 对象的 commit()保存数据库的修改。现在 sales 表在我们的内数据库中，它用 4 行数据，我们来学习一下如何从数据表中提取数据。第 25 行用 connection 对象的 execute()方法运行一行 SQL 命令并将命令结果赋值给名为 cursor 的 cursor 对象。Cursor 对象有多个方法 (如 execute, executemany, fetchone, fetchmany, and fetchall)。但是你通常感兴趣于查看和操作整个 execute()内的 SQL 命令的结果集，你通常想用 fetchall()方法获取结果集中的所有行。第 26 行实现这种代码。它用 cursor 对象的 fetchall()方法返回第 25 行执行的 SQL 命令的结果集，并将行赋值给列表变量 rows。即变量 rows 是列表，包含所有的第 25 行执行的 SQL 命令的结果。每一行是元组，所以行是元组的列表。这种情况，因为我们知道表格 sales 包含四行数据，SQL 命令选择了 sales 表的所有数据，我们知道行是四个元组的列表。最后，第 29-33 行，我们返回基本操作创建 row_counter 变量 rows 变量中行的数目，创建 for 循环遍历 rows 中的每一行，row_counter 为第一行加 1，最后 for 循环完成 rows 中数据行的遍历。打印字串 Number of rows: 和 row_counter 中的数值到命令行。如我所说，我们希望 rows 中有四行数据。

要看这个 Python 脚本的运行结果，在命行中输入如下，如决于你的操作系统，按回车：

Windows 系统:

```
python 1db_count_rows.py
```

macOS 系统:

```
chmod +x 1db_count_rows.py
./1db_count_rows.py
```

你会看到如图 4-4 (on Windows) 或图 4-5 (on macOS) 所示的打印输出到屏幕。输出显示有四个记录在 sales 表。更一般地，输出也显示我们创建内存数据库，创建数据表 sales，用 4 行记录填充表，获取表中的所有记录，计算了输出的行数。现在我们知道了创建内存数据库，创建表，表中加载数据，获取表中的数据，我们扩展能力通过在表中插入数据，修改表中的记录，用 CSV 输入文件。

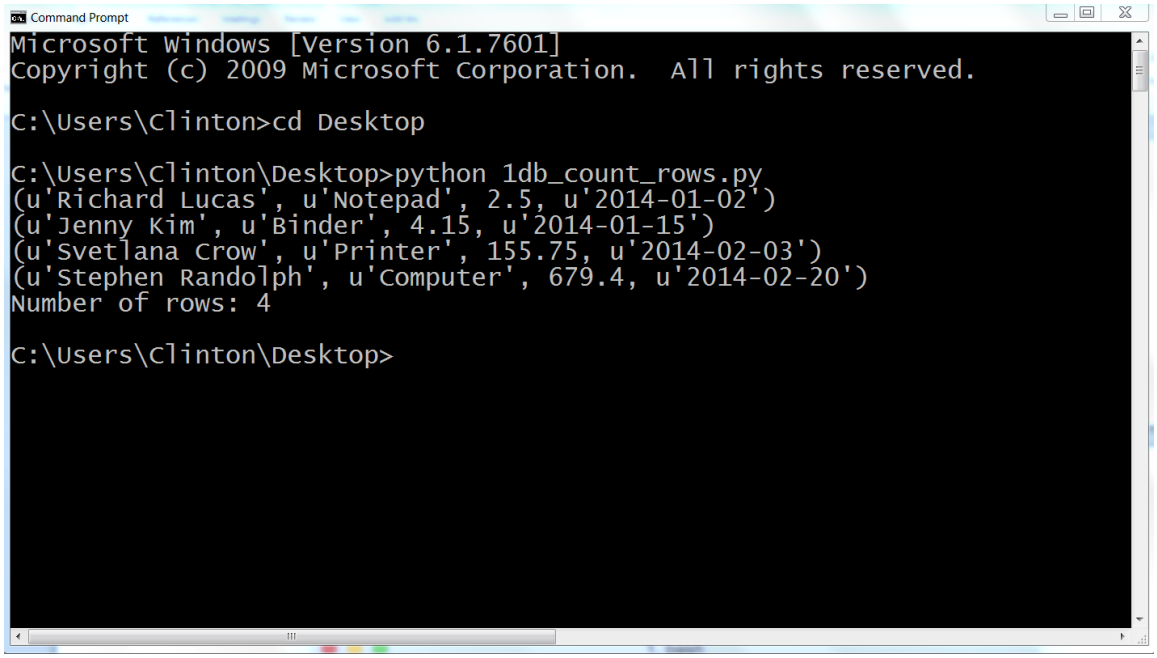


图 4-4. 1db_count_rows.py 输出显示创建 SQLite3 数据库表，插入四行数据到表，查询表中的所有数据，打印结果到 Windows 计算机的屏

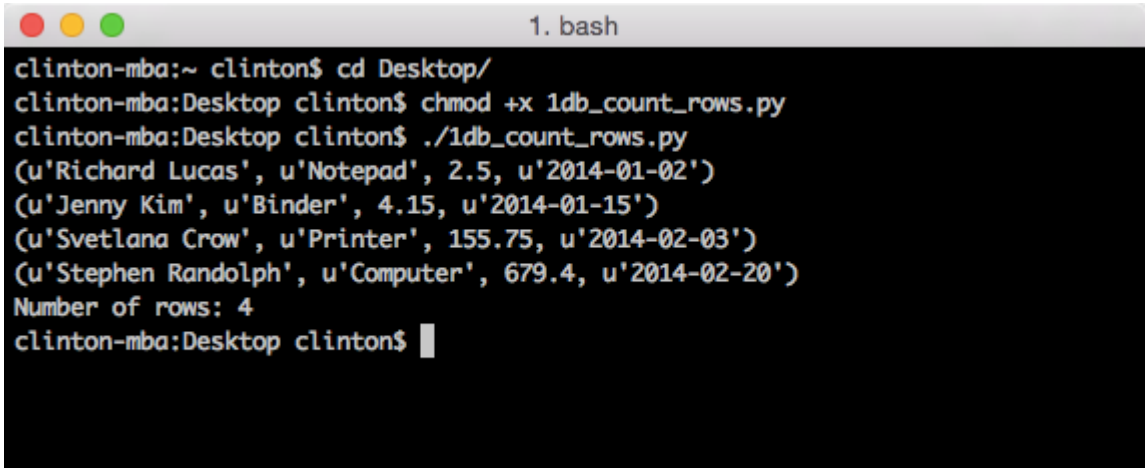


图 4-5. Mac 计算机上 1db_count_rows.py 的输出

在表中插入新的记录

前面的例子解释了加载数据到表的基本操作，但是它有个严重的限制，即我们手工写入我们要加载的数据到表中。如果我们要加载 10000 行记录，有 20 到 30 个属性，到表中，会如何？不用说，手工加载记录不能放大。很多情况下，要加载到数据库表中的数据是数据库查询的结果或者是在一个或多个 Excel 或 CSV 文件里。对于大部分数据库来说，将数据库查询的结果倒入到 CSV 文件相对易容。我们也学习了如何处理 Excel 和 CSV 文件。我们来学习别的方法并将数据加载到数据库，用 CSV 输入文件放大。我们创建新的 python 脚本。脚本创建数据表，自 CSV 文件插入数据到表，然后显示表中的数据。

第三步，打印数据到命令行或端并不是必须的（我也不推荐打印记录到窗口，如果你加载了上千行记录），但我包含这一步来说明打印每一行的所有列不用指明列索引。要开始，在文本编辑器输入如下代码并保存为 2db_insert_rows.py:

```
#!/usr/bin/env python3
import csv
import sqlite3
import sys
# Path to and name of a CSV input file
input_file = sys.argv[1]
# Create an in-memory SQLite3 database
# Create a table called Suppliers with five attributes
con = sqlite3.connect('Suppliers.db')
c = con.cursor()
create_table = """CREATE TABLE IF NOT EXISTS Suppliers
(Supplier_Name VARCHAR(20),
Invoice_Number VARCHAR(20),
Part_Number VARCHAR(20),
Cost FLOAT,
Purchase_Date DATE);"""
c.execute(create_table)
con.commit()
# Read the CSV file
# Insert the data into the Suppliers table
file_reader = csv.reader(open(input_file, 'r'), delimiter=',')
header = next(file_reader, None)
for row in file_reader:
data = []
for column_index in range(len(header)):
data.append(row[column_index])
print(data)
c.execute("INSERT INTO Suppliers VALUES (?, ?, ?, ?, ?);", data)
con.commit()
print('')
# Query the Suppliers table
output = c.execute("SELECT * FROM Suppliers")
rows = output.fetchall()
for row in rows:
output = []
for column_index in range(len(row)):
output.append(str(row[column_index]))
print(output)
```

这个脚本，像第二章的脚本，依靠 CSV 模块和 SYS 模块。第 2 行导入 CSV 模块以便使用它的方法来读和解析 CSV 输入文件。第 4 行导入 SYS 模块以便我们可以在命令行中提供路径和文件名给脚本。第三行导入 sqlite3 模块以便我们用它的方法来创建简单的本地数据库，数据表，执行 SQL 查询。第 6 行在用 SYS 模块读命令行中的路径和文件名并赋值给变量 input_file。第 9 行创建 connection 到简单的本地数据库称为 Suppliers.db。我提供数据库名而不是特殊的关键字 ':memory:'来说明如何创建可以持久的重启计算机时不会丢失的数据库。因为你要保存脚本到桌面，所以 Suppliers.db 也保存在桌面。如要你相要保存数据库在别的地方，你可以改变路径，如 'C:\Users\<Your Name>\Documents\Suppliers.db'而不是'Suppliers.db'。第 10-18 行创建 cursor 和多行 SQL 语句来创建名为 Suppliers 的表，它有五列属性，执行 SQL 语句，保存对数据库的修改。第 21 到 29 行从 CSV 输入文件读到要加载到数据库表中的数据，为输入文件的每一行执行 SQL 语句以插入数据库表。第 21 行用 CSV 模块创建 file_reader 对象。第 22 行用 next()方法从输入文读第一行，标题行，将它赋值给变量 header。第 23 行创建一个 for 循环遍历输入文件的所有数据行。第 28 行，对于每一行输入，我们用 INSERT 语句所需的行中的数据填充 data。第 25 行创建 for 循环遍历每行的列。第 26 行用列表的 append() 方法填输入文件中那一行的所有值。第 27 行打印添加到 data 的行数据到命令行窗口。注意行首缩进。这一行在 for 循环外行首缩进，而不在 for 循环内，以便它为每一行出现而不是为输入文件的每一行和列。这一行有助于调试，如果代码正确工作，你可以去除它以致不会在屏幕上打印过多的信息。第 28 行是真正加载每一行的数据到数据库表中。这一行用 cursor 对象的 execute() 方法执行 INSERT 语句来插入一行数据到 Suppliers 表。问号是每一个插入值的占位符。问号的个数要与输入文件中列的数目一致。而且，输入文件中列的顺序要与数据库表中列的顺序对应。问号位置取代的值来自 data 中值的列表，它出现在 execute()语句的逗号后。因为 data 用输入文件中每一行的值填充且 INSERT 语句为输入文件的每一行执行，这些代码有效的从输入文件读取行数据并加载到数据库表中。最后，第 29 行是另一个 commit 语句来保存数据库的修改。第 32 行到 38 行展示如何自 Suppliers 选择所有的数据并打印输出到命令行窗口。第 32 行和 33 行执行 SQL 语句来选择 Suppliers 表中的所有数据,获取所有“output”中的行到变量“rows”。第 34 行创建 for 循环遍历“rows”的每一行。第 36 行创建 for 循环遍历每行的列。第 37 行添加每列值到“output”列表。最后第 38 行的打印语句确保每一行输出打印到新的行（注意行首缩进，它在行中而不是在列中）。现在我们所要的是包含所有我们想要加载到数据库表中的数据的 CSV 文件。本例，我们使用第二章用过的 supplier_data.csv 文件。 可能你跳过了第二章或者没有这个文件，supplier_data.csv 中的数据看起来像图 4-6。

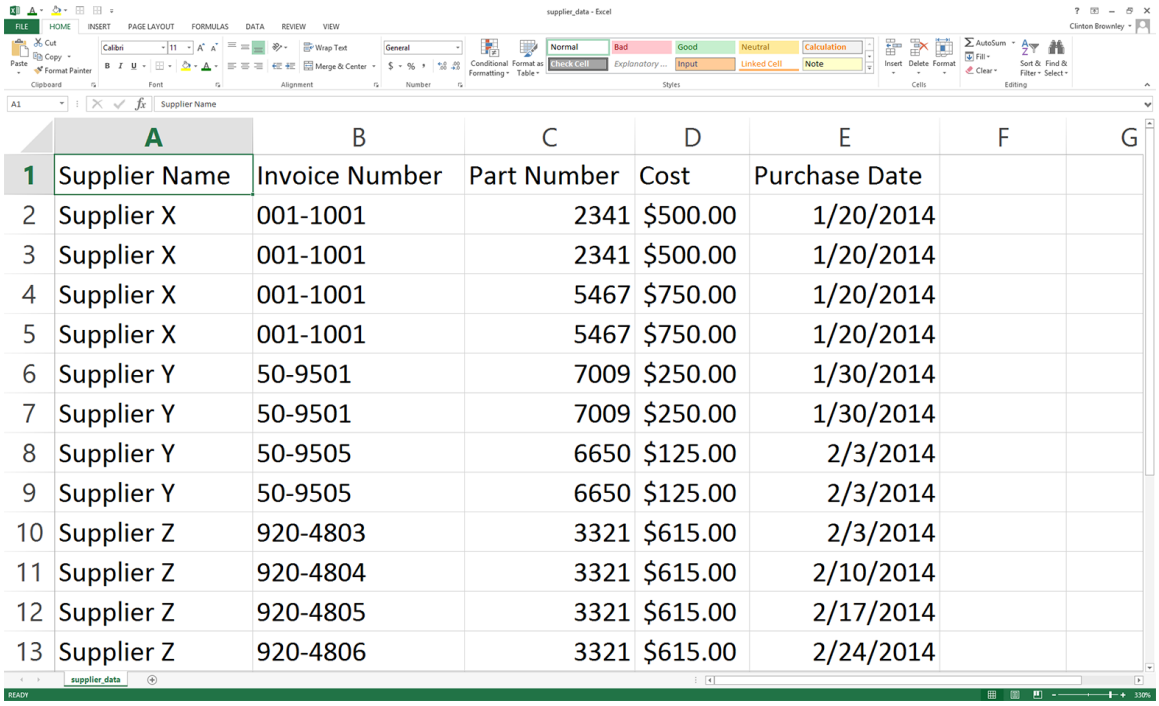
现在我们有脚本和 CSV 输入文件，我们用脚本来加载 CSV 输入文件中的数据到 Suppliers 数据表。

在命令行中输入如下并回车:

```
python 2db_insert_rows.py supplier_data.csv
```

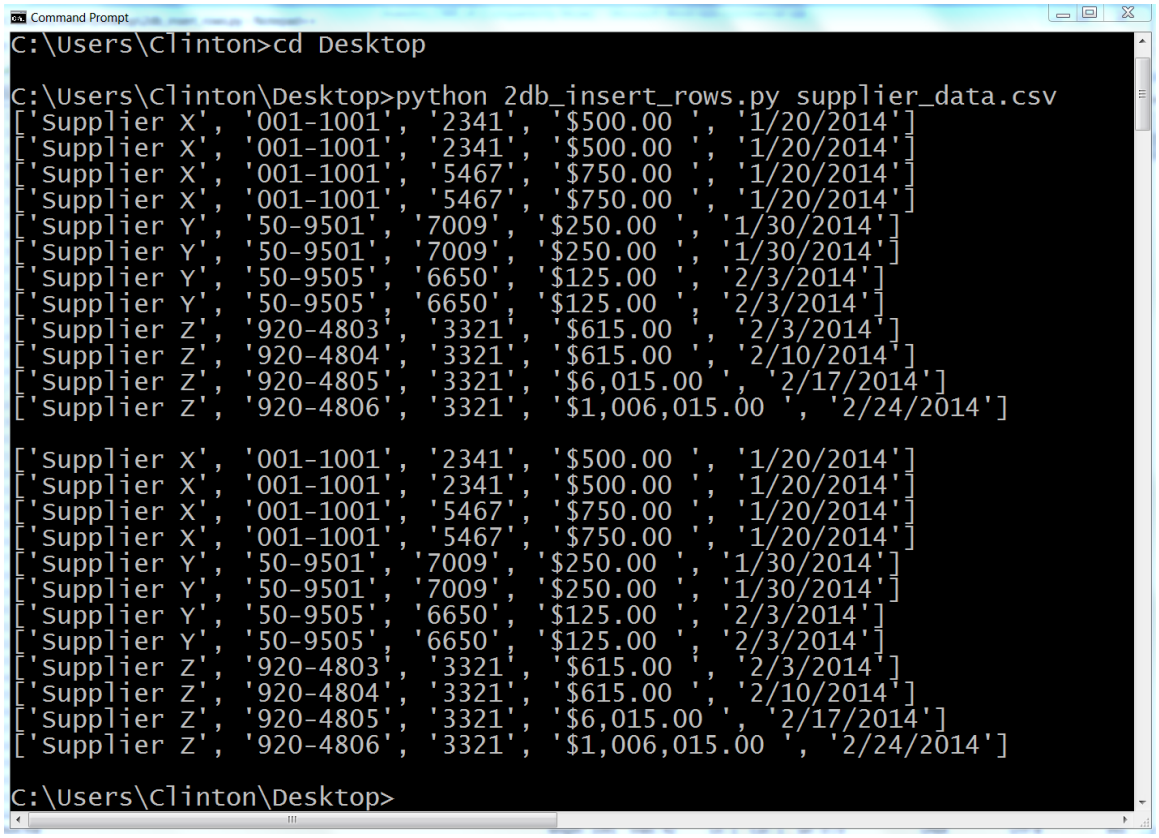
图 4-7 展示输出看起来的样子。

输出的第一块是数据行，因为它们从 CSV 文件解析来，第二块是从 sqlite 表获取的数据。



	A	B	C	D	E	F	G
1	Supplier Name	Invoice Number	Part Number	Cost	Purchase Date		
2	Supplier X	001-1001	2341	$500.00	1/20/2014		
3	Supplier X	001-1001	2341	$500.00	1/20/2014		
4	Supplier X	001-1001	5467	$750.00	1/20/2014		
5	Supplier X	001-1001	5467	$750.00	1/20/2014		
6	Supplier Y	50-9501	7009	$250.00	1/30/2014		
7	Supplier Y	50-9501	7009	$250.00	1/30/2014		
8	Supplier Y	50-9505	6650	$125.00	2/3/2014		
9	Supplier Y	50-9505	6650	$125.00	2/3/2014		
10	Supplier Z	920-4803	3321	$615.00	2/3/2014		
11	Supplier Z	920-4804	3321	$615.00	2/10/2014		
12	Supplier Z	920-4805	3321	$615.00	2/17/2014		
13	Supplier Z	920-4806	3321	$615.00	2/24/2014		

图 4-6. supplier_data.csv 文件的示例数据,Excel Worksheet 展示



```
Command Prompt
C:\Users\Clinton>cd Desktop

C:\Users\Clinton\Desktop>python 2db_insert_rows.py supplier_data.csv
['Supplier X', '001-1001', '2341', '$500.00 ', '1/20/2014']
['Supplier X', '001-1001', '2341', '$500.00 ', '1/20/2014']
['Supplier X', '001-1001', '5467', '$750.00 ', '1/20/2014']
['Supplier X', '001-1001', '5467', '$750.00 ', '1/20/2014']
['Supplier Y', '50-9501', '7009', '$250.00 ', '1/30/2014']
['Supplier Y', '50-9501', '7009', '$250.00 ', '1/30/2014']
['Supplier Y', '50-9505', '6650', '$125.00 ', '2/3/2014']
['Supplier Y', '50-9505', '6650', '$125.00 ', '2/3/2014']
['Supplier Z', '920-4803', '3321', '$615.00 ', '2/3/2014']
['Supplier Z', '920-4804', '3321', '$615.00 ', '2/10/2014']
['Supplier Z', '920-4805', '3321', '$6,015.00 ', '2/17/2014']
['Supplier Z', '920-4806', '3321', '$1,006,015.00 ', '2/24/2014']

['Supplier X', '001-1001', '2341', '$500.00 ', '1/20/2014']
['Supplier X', '001-1001', '2341', '$500.00 ', '1/20/2014']
['Supplier X', '001-1001', '5467', '$750.00 ', '1/20/2014']
['Supplier X', '001-1001', '5467', '$750.00 ', '1/20/2014']
['Supplier Y', '50-9501', '7009', '$250.00 ', '1/30/2014']
['Supplier Y', '50-9501', '7009', '$250.00 ', '1/30/2014']
['Supplier Y', '50-9505', '6650', '$125.00 ', '2/3/2014']
['Supplier Y', '50-9505', '6650', '$125.00 ', '2/3/2014']
['Supplier Z', '920-4803', '3321', '$615.00 ', '2/3/2014']
['Supplier Z', '920-4804', '3321', '$615.00 ', '2/10/2014']
['Supplier Z', '920-4805', '3321', '$6,015.00 ', '2/17/2014']
['Supplier Z', '920-4806', '3321', '$1,006,015.00 ', '2/24/2014']

C:\Users\Clinton\Desktop>
```

图 4-7. 2db_insert_rows.py 在 Windows 计算机的输出

这个输出显示 12 个列表值为输入文件中的 12 行数据，不包括标题行。接着是空白，然后是从数据库表获取的 12 行数据。

这个例子展示了如何放大规模通过读取 CSV 输入文件中所有人加载的数据并插入数据到表中来加载变据到数据库表。这个例子覆盖的情形是你要加载新的行到数据表中，但是如何更新现有的表中的行呢？下一个例子覆盖这种情形。

更新表中的记录

前面的例子解释了如何用 CSV 输入文件来增加行到数据库表。它是一种方法，因为你本循环和 glob，你可以放大到任意数目的文件。但是，有时，不是要加载新数据到表格中而是要更新现有表格中的行数据。幸运的是，我们可以重用从 CSV 文件中读取数据的技术来更新现有表中的行。事实上，组装行值组 SQL 语句然后为 CSV 输入文件的每一行执行 SQL 语句的技术与前面的例子一样。需要改变的是 SQL 语句。它从 INSERT 语句改为 UPDATE 语句。

我们已经熟悉了如何使用 CSV 输入文件来加载数据到数据库表，所以我们来看一下如何用 CSV 输入文件来更新数据表中现有的记录。T 在文本编辑器输入如下代码并保存为 3db_update_rows.py:

```
1 #!/usr/bin/env python3
2 import csv
3 import sqlite3
4 import sys
5 # Path to and name of a CSV input file
6 input_file = sys.argv[1]
7 # Create an in-memory SQLite3 database
8 # Create a table called sales with four attributes
9 con = sqlite3.connect(':memory:')
10 query = """CREATE TABLE IF NOT EXISTS sales
11 (customer VARCHAR(20),
12 product VARCHAR(40),
13 amount FLOAT,
14 date DATE);"""
15 con.execute(query)
16 con.commit()
17 # Insert a few rows of data into the table
18 data = [('Richard Lucas', 'Notepad', 2.50, '2014-01-02'),
19 ('Jenny Kim', 'Binder', 4.15, '2014-01-15'),
20 ('Svetlana Crow', 'Printer', 155.75, '2014-02-03'),
21 ('Stephen Randolph', 'Computer', 679.40, '2014-02-20')]
22 for tuple in data:
23 print(tuple)
24 statement = "INSERT INTO sales VALUES(?, ?, ?, ?)"
25 con.executemany(statement, data)
```

```
26 con.commit()
27 # Read the CSV file and update the specific rows
28 file_reader = csv.reader(open(input_file, 'r'), delimiter=',')
29 header = next(file_reader, None)
30 for row in file_reader:
31 data = []
32 for column_index in range(len(header)):
33 data.append(row[column_index])
34 print(data)
35 con.execute("UPDATE sales SET amount=?, date=? WHERE customer=?;", data)
36 con.commit()
37 # Query the sales table
38 cursor = con.execute("SELECT * FROM sales")
39 rows = cursor.fetchall()
40 for row in rows:
41 output = []
42 for column_index in range(len(row)):
43 output.append(str(row[column_index]))
44 print(output)
```

这些代码看起来很熟悉。第 2 到 4 行导入三个 Python 内置模块，以便我们可以用它们的方法来读命令行输入，读 CSV 输入文件，遍历内存数据库存和表。第 6 行赋值 CSV 输入文件给变量 input_file。第 9 到 16 行创建内存数据库和数据表 sales,它有四列属性。第 18 到 24 行为 sales 创建四个记录并插入记录到到中。看一下 Richard Lucas 和 JennyKim 的记录。这两个记录是我们后面的脚本要更新。到此，sales 与你所看到的表相似（可能小一些），当你要更新现有数据表中的记录时。第 28 - 36 与前面例子的代码一样。不同的是第 35 行，用 UPDTE 语句替代前面的 INSERT 语句。语句指出你要更改哪一条记录哪一列属性。本例中，我们要改行定 customers 集的 amount 和 date 值。像前面的例子，查询语句中有很多占位符问号，CSV 输入文件的数据顺序要与模询的属性的顺序相同。本例中，自左到右的顺序为 amount, date, and customer。所以，CSV 文件中从左到右的列应为 amount, date, and customer。最后，第 39-44 的代码，与前面的例之基本一样。这些代码获取放大的表格中的所有行的数据，然后打印每一行到命令行窗口，列之间有一个空格。

现在我们所要的是包含我们要用来更新我们的数据库表中的数据的 CSV 输入文件。

1. 打开电子表格。
2. 如图 4-8 所示增加数据。
3. 将文件保存为 data_for_updating.csv.

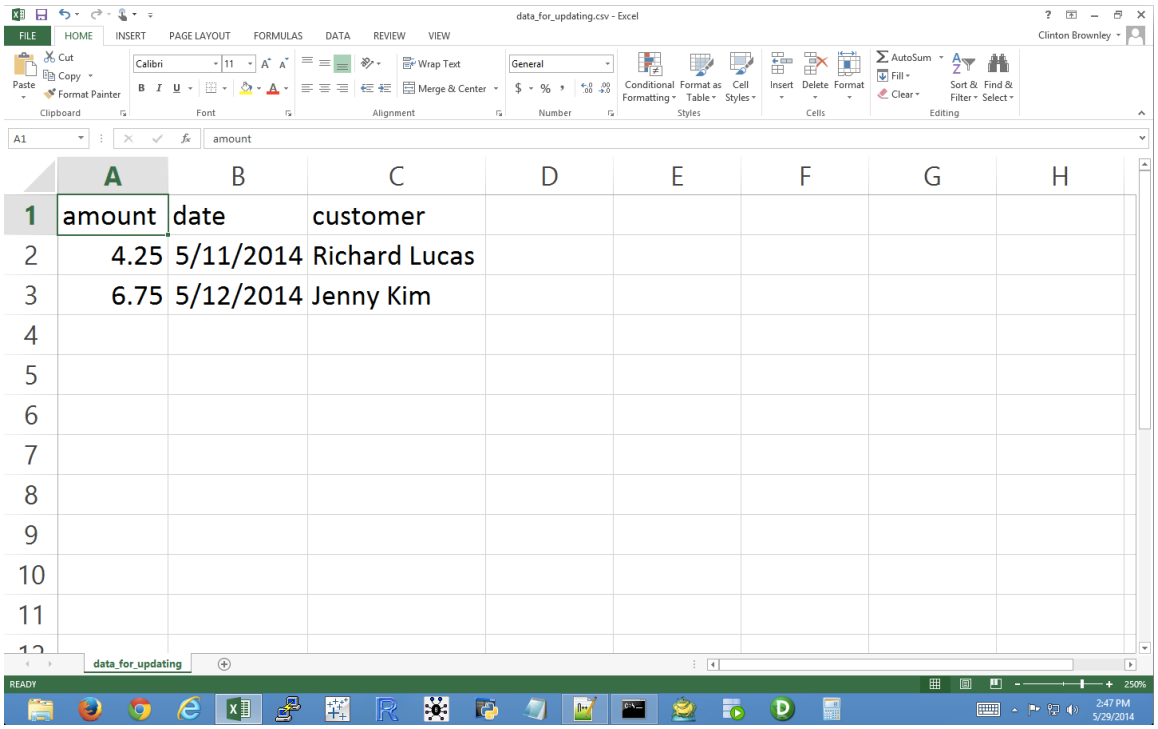


图 4-8. data_for_updating.csv 文件中的示例数据,用 Excel worksheet 显示

显在我们有 4 个脚本和一个 CSV 输入文件，我们用我们的脚本和 CSV 输入文件来更新 sales 数据库表中的记录。

在命令行中输入如下并回车:

```
python 3db_update_rows.py data_for_updating.csv
```

图 4-9 展示输出看起来的样子。前四行是原始数据的行。下面的两行来自 CSV 输入文件。最后四行是数据库表更新以后四行数据。

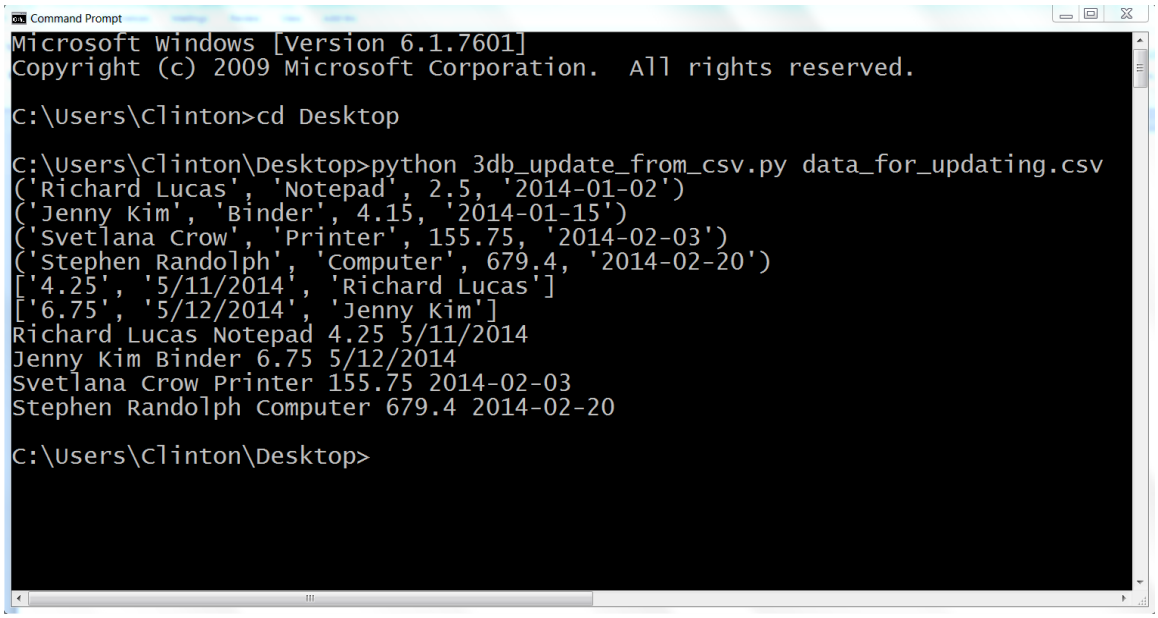


图 4-9. Windows 计算机中 3db_update_rows.py 的输出

输出显示数据库表中的四行初始行，接着两行是需要更改的行。需要更改的两个列表值显示，Richard Lucas 的 amount 将是 4.25，日期间为 5/11/2014。相似的，Jenny Kim 的新的 amount 将会是 6.75，日期将会是 5/12/2014。在更新两行下面，显示了从数据库中获得的执行更新以后的数据。每一行都打印到屏上，每行的值由空格分开。回想一下 Richard Lucas 的原始 amount 和 date 值分别为 2.5 and 2014-01-02。Jenny Kim 原始 amount 和 date 值分别为 4.15 和 2014-01-15。中你从图 4-9 所见，这 Richard Lucas 和 Jenny Kim 的这两个值已更改以反映 CSV 输入文件的值。

本例说明如何用 CSV 输入文件放大更新数据库表中的现有记录。通过 CSV 输入文件提供指定记录需要更新的数据。到本章，例子每依靠 python 的内置 sqlite3 模块。
我们可以用这个模块来写脚本，不需要下载和安装 MySQL, PostgreSQL, 或 Oracle 等数据库。并学习了如何加载数据到数据库，并更新数据库中的记录。以及将查询结果写入到输出 CSV 文件。我们继续开始。

MySQL 数据库

要完成这一节的例子，你要安装 MySQLdb，MySQL-python (Python v2) 或 mysqlclient (Python v3)。这个包使得 PYTHON 可以与数据库及数据表交互，所以我们要用它来与本节创建的 MySQL 数据表交互。如果你安装 Anaconda 的 Python，你已经有主个包了。它邦定到安装中了。如果你从 Python.org 网站上安装 Python，你需要按附录 A 的指示下载和安装这个包。与前面一样，为了与数据库工作，我们首先需创建一个。

1. 按附录 A 下载 MySQL 数据库程序。下载完后就可以访问 MySQL 命令行客户端。
2. 在命令行中输入 mysql 来打开 MySQL 命令行客户端。你想要通过命令行来与 MySQL 数据库程序交互。我们来看一下 MySQL 数据库程序中现有的数据库。
3. 输入如下并按回车。图 4-10 显示了在 Windows 计算机中的结果。

展示数据库 SHOW DATABASES;

注意命令以分号结束。这样 MySQL 才知道你完成了命。如果你按回车而没有分号，MySQL 将然望另一行命令。这个命令的输出结果显示，已经用四个数据库在 MySQL 数据库程序中。这些数据库使得 MySQL 数据库程序运行并包括了访程序的用户的访问权信息。

要创建一个数据，我们首先要创建自己的数据库。

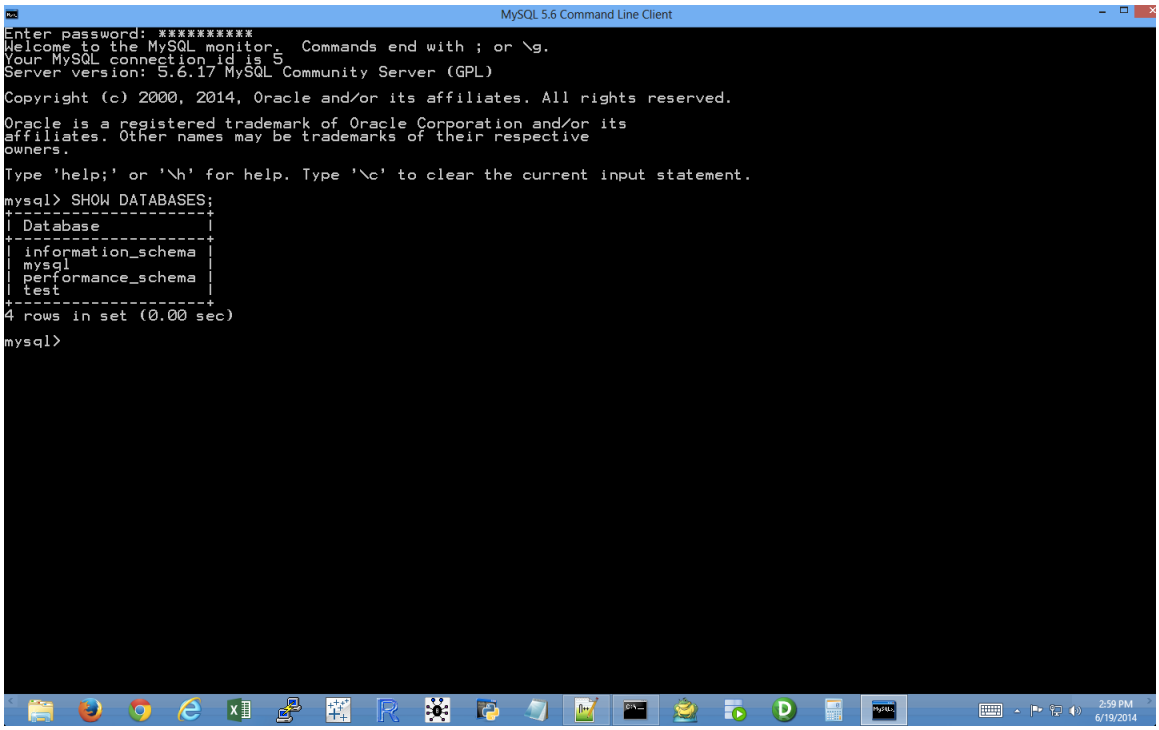


图 4-10. 安装了 MySQL 之后， SHOW DATABASES 显示默认的 MySQL 中的数据库。

4. 要创建数据库，我们输入以下并回车:

```
CREATE DATABASE my_suppliers;
```

回车以后，你可以运行 SHOW DATABASES;命令再一次显示你已经创建了新 的数据库。要在 my_suppliers 数据库中创建数据库表，我们首先要先择处理 my_suppliers 数据库。

5. 要处理 my_suppliers 数据库，输入如下并回车 (见图 4-11):

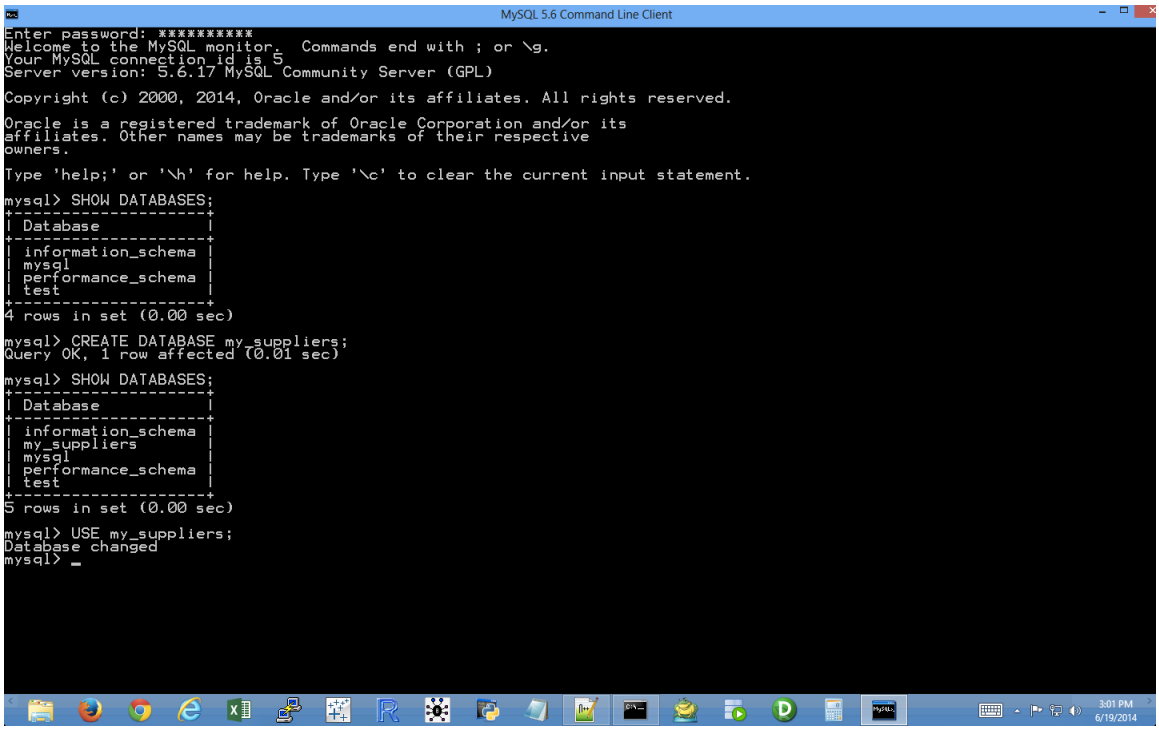


图 4-11. 创建新的数据库 my_suppliers 之后的结查，检查到新的数据库已经包括进现在数据库列表了，切换到这个数据库并使用它。

回车以后，你就在 my_suppliers 数据库中了。

现在我们创建数据库表来贮存供应商信息。

6. 要创建名为 Suppliers 的数据库表,输入如下并回车:

```
CREATE TABLE IF NOT EXISTS Suppliers
(Supplier_Name VARCHAR(20),
Invoice_Number VARCHAR(20),
Part_Number VARCHAR(20),
Cost FLOAT,
Purchase_Date DATE);
```

这个命令创建了名为 Suppliers 的数据库表，如果数据库中原来没有。数据表有 5 列数据，(也称为 fields 或 attributes)：Supplier_Name，Invoice_Number，Part_Number，Cost，和 Purchase_Date。前三列是字符变量 VARCHAR fields.。20 是分配了 20 个字符的意思。如果输入的数据大于 20 个字符，数据会被截掉。如果小于 20 个字符，就会补充一定的空格键。使用 VARCHAR 作字段包含了变量的长度，它很有用，因为表不会浪费空间来存贮更多的不必要的字符。但是你要确保括号中的数值足够以免最长的字串被截了。对于 VARCHAR 的一些补充是 CHAR，ENUM，和 BLOB。你可以考虑这些选项，当你需要指定一定长度的数值时，指明左填充和右填充的长度，允许的值如（最小，最大，中值）。可者允许用大文本文件。第四列是浮点数 FLOAT 字段。浮点字段可以贮存浮点数，近似值。本例中第四列含有钱的值，对 FLOAT 的选项是 NUMERIC，它是固定点的值。.例如，

不是 FLOAT，而用 NUMERIC(11,2)。11 是精确值，2 表示小数点的位数。本例中我们用 FLOAT 而不是 NUMERIC。是后一列是 DATE 字段。 DATE 存贮日期，没有时间部分，格式为 'YYYY-MM-DD'。所以 6/19/2014 在 MySQL 中贮存为'2014-06-19'。无效的日期转换为'0000-00-00 '。

7. 要确保数据库表被正确创建，输入如下并回车：

then hit Enter:

```
DESCRIBE Suppliers;
```

回车后你看到表列出了你的列名和数据类型(如, VARCHAR 或 FLOAT)， 以及列是否允许 NULL。

我们已创建了数据库 my_suppliers 和数据表，Suppliers。我们创建新的用户，给用户与数据库和数据表交互的权力。

8. 要创建用户，输入如下并回车(确保替换 username ;改变密码,secret_password):

```
CREATE USER 'username'@'localhost' IDENTIFIED BY 'secret_password';
```

现在我们创建了新的用户，让我们给用户操作 my_suppliers 的所有权力。

通过给用户所有的权力，我们可以对数据表进行许多少不同的操作。 这些权力很有用，因为本节的脚本包括加载数据到表，修改特定的记录，查询表格。

9.给新用户所有的 权力，输入以下命令，回车 (确保用用户名替代 username):

```
GRANT ALL PRIVILEGES ON my_suppliers.* TO 'username'@'localhost';
FLUSH PRIVILEGES;
```

你现在可以在本地与 my_suppliers 数据库的 Suppliers 表交互了 (如，你的本地计算机）。见图 4-12。

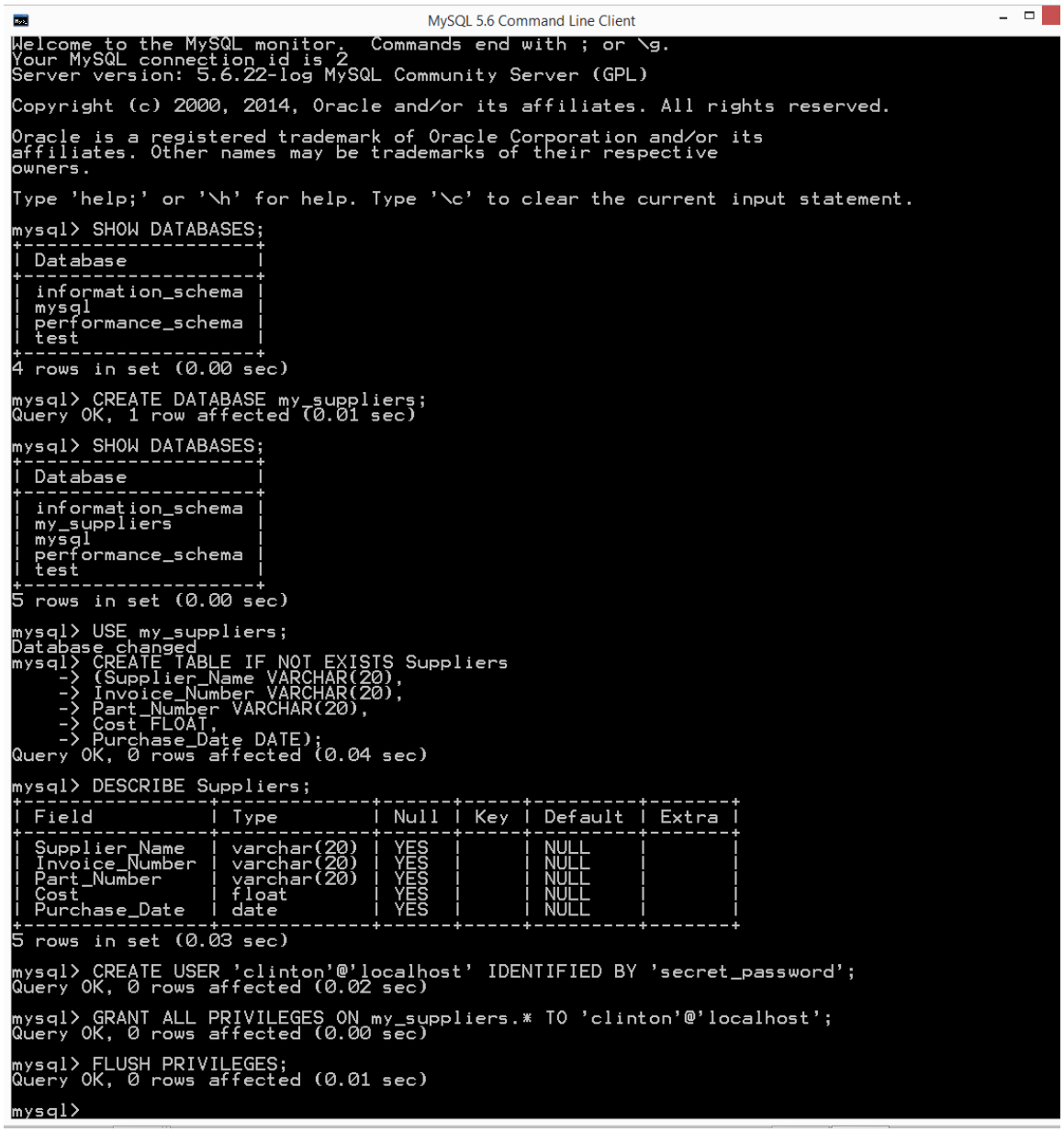


图 4-12. 在 my_suppliers 数据库中创新新的表 Suppliers，创建新用户，给用户操作 my_suppliers 数据库和表的所有权力。

我们已经有数据库和数据表，我们来学习一下如何用 python 加载数据到表中。

在表中插入新的记录

现在我们可以从 CSV 文件加载记录到数据库表了。你已可以从 Pythone 脚本或 Excel 加载记录到 CSV 文件了，这用助于你创建通用的数据管线。

我们来创建新的 python 脚本。脚本自 CSV 文件插入记录到我们的数据库表，然后显示表中的数据。 第二步，打印数据到命令行窗口，不是必须的（我不推荐打印记录，如果你加载了上

千行记录！），但是我包括了这一步，来展示一种打印记录的所有列的方法，不需指明列的索引。(例如，代码对于任意数目的列通用)。

在文本编辑器中输入以下代码并保存为 4db_mysql_load_from_csv.py:

```
#!/usr/bin/env python3
import csv
import MySQLdb
import sys
from datetime import datetime, date

# Path to and name of a CSV input file
input_file = sys.argv[1]
# Connect to a MySQL database
con = MySQLdb.connect(host='localhost', port=3306, db='my_suppliers', \
user='root', passwd='my_password')
c = con.cursor()
# Insert the data into the Suppliers table
file_reader = csv.reader(open(input_file, 'r', newline=''))
header = next(file_reader)
for row in file_reader:
data = []
for column_index in range(len(header)):
if column_index < 4:
data.append(str(row[column_index]).lstrip('$')\
.replace(',', '').strip())
else:
a_date = datetime.date(datetime.strptime(\
str(row[column_index]), '%m/%d/%Y'))
# %Y: year is 2015; %y: year is 15
a_date = a_date.strftime('%Y-%m-%d')
data.append(a_date)
print data
c.execute("""INSERT INTO Suppliers VALUES (%s, %s, %s, %s, %s);""", data)
con.commit()
print("")
# Query the Suppliers table
c.execute("SELECT * FROM Suppliers")
rows = c.fetchall()
for row in rows:
row_list_output =[]
for column_index in range(len(row)):
row_list_output.append(str(row[column_index]))
```

```
39 print(row_list_output)
```

这个脚本，与我们在第 2 章所写的相似。依靠 csv， datetime,string， 和 sys modules。第二行导入 csv 以便我们可以用它的方法来读取和解析 CSV 输入文件。第 4 行导入 sys 模块，以便我们可以在命令行中提供文件路径和文件名给脚本使用。第 5 行导入 datetime 模块的 datetime 和 date 方法以便我们可以操作和格式化输入文件最后一列的日期。我们需要从值中去除美元符号，并却除任何嵌入的逗号以使可以输入到接收浮点数的数据库表。第 3 行导入我们下载和安装的 MySQLdb 模块，以便我们可以用它的方法连接到 MySQL 数据为和表。第 8 行用 sys 模块在命令行中读文件的路径和名称并赋值给变量 input_file。第 10 行用 MySQLdb 模块的 connect()方法连接到我们前面创建的数据库 my_suppliers。不像处理 CSV 或 Excel 文件，你可以现场读，修改，删除，MySQL 把数据库当作另一台计算机（服务器），你可以连接，发送数据，请求数据。连接指定多个参数，包括主机，端口，数据库，用户，密码。主机是贮存数据库的计算机的主机名。MySQL 服务器贮存于你的计算机，所以，主机名是 localhost。如果连接到不同的数据源，服务器是不同的计算机，所以你要改 localhost 为贮存数据说库服务器的计算机的主机名。端口号是 TCP/IP 边接 MySQL 服务器的端口号。默认的端口号为 3306。如主机名参数，如果你不使用本地计算机，且你的 MySQL 服务器的管理员设置了不同的服务器端口，则你要用那个端口来连接到 MySQL 服务器。本例中，我们用默认值，所以 localhost 是有效的主机名，3306 是有效的端口号。db 就是你想要连接的数据库的名称。本例中我们要连接 my_suppliers 数据库，因为它贮存了我们想要加载数据的数据库表。如果你以后在本地计算机中创建了别的数据库，如 contacts，你需要将 my_suppliers 改为 contacts 作为 db 参数来连接数据库。User 是连接数据库的用户的用户名。本例中，我们以“root”用户名连接，password 是我们安装 MySQL 服务器时创建的。当你安装 MySQL （按附录 A）时， MySQL 安装过程向你提供 root 用户的密码。我的 root 用户的密码是’my_password’。当然，脚本中你要替换成你的密码。在数据库，数据库表，新用户设置阶段，我创建的新用户 clinton 和密码 secret_password。因此，我在脚本中也用下面的连接细节：user=’clinton’ and passwd=’secret_password’。 如果你要用 user=’root’ ，则你要用实际密码替代’my_password’。可选地，你可以 CREATE USER 命令创建的用户名和密码。用这 5 个输入，你创建了到 my_suppliers 数据库的本地连接。第 12 行创建 cursor ，我们可以用它来对 Suppliers 执行 SQL 语句并保存对数据库的更改。第 14 到 29 行处理读取需要加载到库据库表中的数据，从 CSV 输入文件读，并为输入文件的每一行执行 SQL 语句以插入记录到数据库表。第 14 行用 CSV 模块创建 file_reader 对象。第 15 行用 next()方法从输入文件读第一行--标题行并赋值给变量 header。第 16 行创建 for 循环遍历输入文件的所有行。第 17 行创建空的列表变量 data。对于每一行输入，我们用需要用于 INSERT 语句的行中的值填充 data。第 18 行用 for 循环遍历每一行的列。第 19 行创建 if-else 语句检查列索引是否小于是 4。因为输入文件有 5 列，日期是最后一列。日期列的索引值是 4。因此，这一行评估我们是否处理日期列之前的列。对于所有前面的列，索引值为 0,1,2,3。第 20 行转换值到字串，从字串左连去除美元符号，如果存在的话，然后添加值到列表变量 data。对于最后一列日期，第 23 行转换值到字串，从字串创建 datetime 对象，基于字串的输入格式，转换 datetime 对象到 date（只保留年，月，日元素），并赋值给变量 a_date。接着，第 26 行转换 date 对象到字串，用我们需要加载日期字串的新的格式(如 YYYY-MM-DD)，重新赋值字串变量到 a_date。最后，第 27 行添加字串到 data。第 28 行打印已添加到 data 的数据行到命令行窗口。注意行 缩进。这一行缩进在 for 循环的外面，而不是在 for 循环的里面，以便它为输入文件的每一行而不是每一行和每一列执行。这一行对于调试有用。但是代码正确工作以后，你可以删除它或注释掉它，以致不用输出过多的信息到命令行窗口。第 29 行是加载行数据到数据库表的。变一行用

Cursor 对象的 execute()方法来执行 INSERT 语句来插入行来 Suppliers 表。 每个%s 是要插入值的占位符。占位符的数目要与输入文件的列的数目一致。而且,输入文件中列的顺序要与数据库表的列的顺序一致。取代 %s 值来自于 data 值列表，出现在 execute()语句的逗号后面。因为 data 是用输入文件的每一行数据填充，INSERT 语句为输入文件的每一行执行，这一行代码有效的从输入文件读行数据并加载行数据到数据库表。再一次，注意行首缩进。这一行在 for 循环的外部行首缩进，以便为输入文件的每一行执行。最后，第 30 行，是另一个保存语句来保存数据库的变更。第 33 行到 39 行展示如何选择 Suppliers 表的所有的数据并打印到命令行窗口。第 33 行和 34 行执行 SQL 语句从 Suppliers 表选择所有的数据并获取所有的行到变量 rows。第 35 行创建 for 循环遍历 rows 的每一行。第 36 行创建空的列表变量 row_list_output，包含自 SQL 查询的每一行的值。第 37 行创建 for 循环遍历行的所有列。第 38 输换每个值到字串，然后添加值到 row_list_output。最后，一旦所有的值在 row_list_output，第 39 行打印行到屏幕。

我们已有 python 脚本,我们来加载 supplier_data.csv 的数据到 Suppliers 数据库表。

在命令行窗口中输入如下并回车:

```
python 4db_mysql_load_from_csv.py supplier_data.csv
```

在 Windows，你可以看到如图 4-13 的输出到命令行窗口。

第一块输出是从 CSV 文件解析的数据，第二块输出是从数据库表查询的数据。

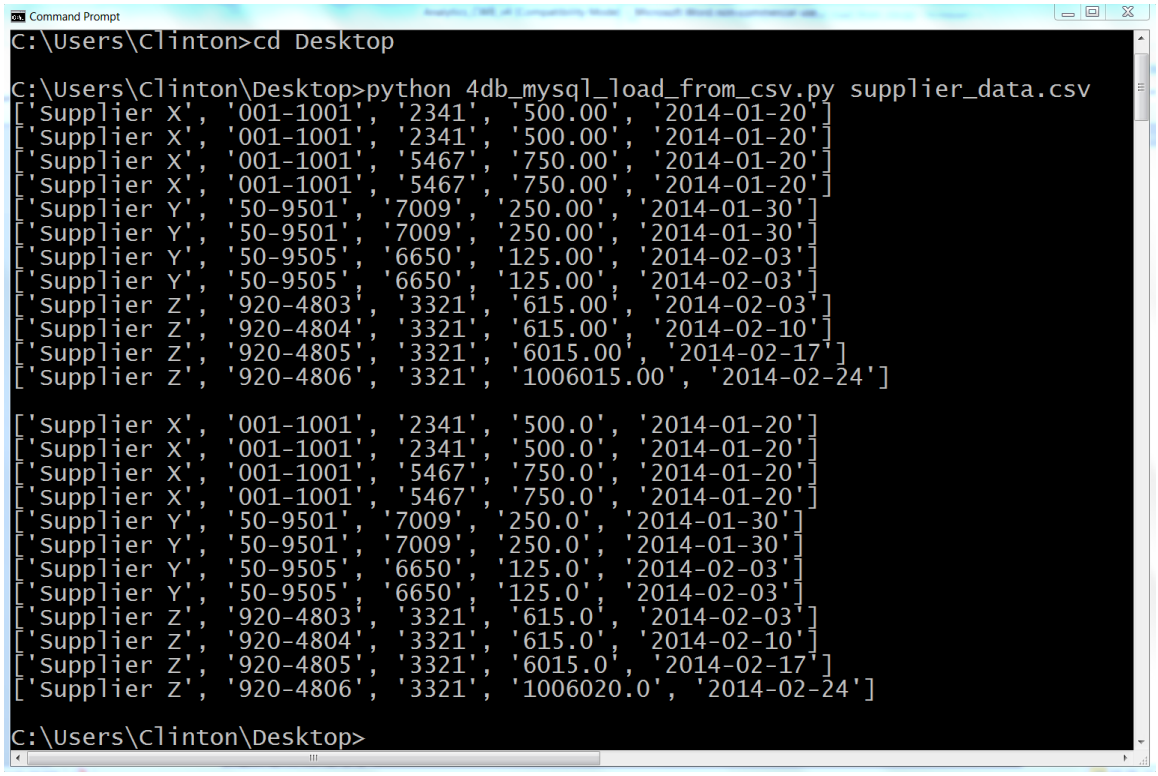


图 4-13. CSV 文件 supplier_data.csv 的输出数据,它插入到 MySQL 表 Suppliers

输出显示 12 个列表的值，代表 CSV 文件的 12 行数据,不包括标题行。你可以看到 12 列表因为每一个列表是用方括号封闭的 ([])且每一个值由逗号分隔。这 12 个列表后是空行，然后是 12 行来自数据库表查询，SELECT * FROM Suppliers，的数据。按行打印，值用逗号分隔。这个输出确认数据成功的加载到数据库然后成功的读取。

要用不同的方法确认，在 MySQL 命令行中输入以下并回车：

```
SELECT * FROM Suppliers;
```

回车后你可以看到表格列出 Suppliers 数据表的每列，12 行数据每列，如图 4-14 所示。

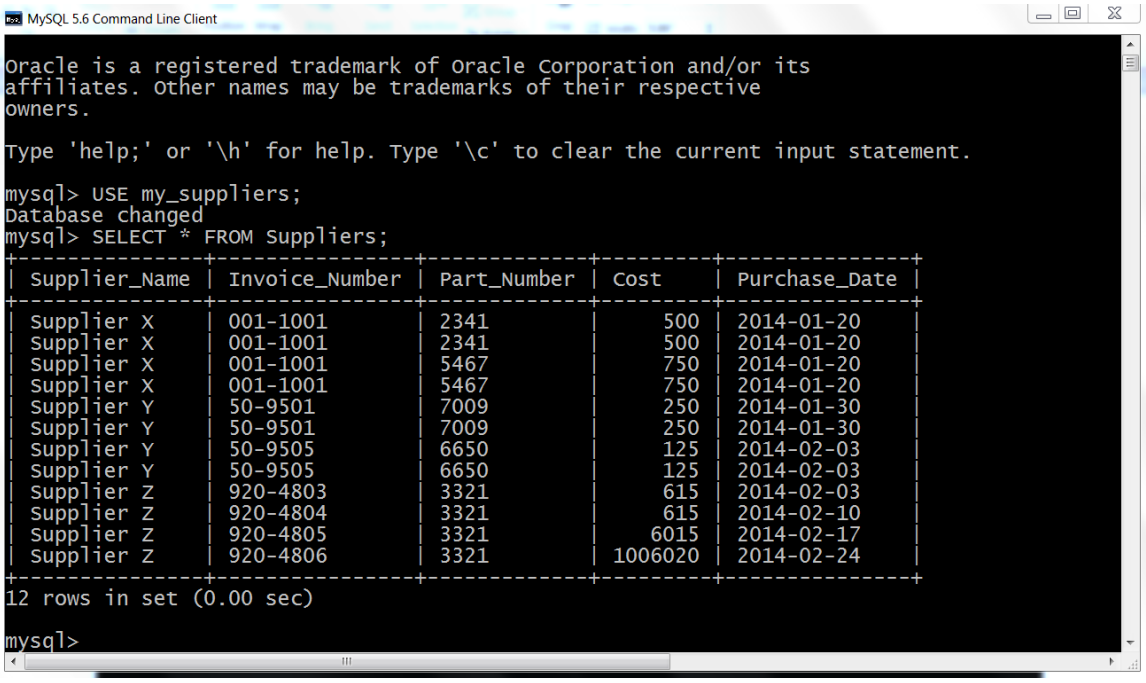


图 4-14. MySQL 客户端的命令行查询结果

我们的数据库表已经有了数据，我们来学习如何用 python 查询数据表并将查询结果写入到 CSV 输出文件而不是打印到屏幕。

查询数据库表并将结果写入 CSV 文件

一旦你的数据库表里有数据，最常见的下一步就是查询表中的数据子集，用于回答或分析商务问题。例如，你可能感兴趣于提供最高利润的客户子集，或感兴趣于花费大于一定阈值的子集。

我们来创建脚本。它将查询 Suppliers 表中的特定记录然后写入到 CSV 文件。这种情况，我们想要输出所有的列，其中 Cost 列大于 1,000.00。要开始，在文本编辑器中输入如下代码并将文件保存为 5db_mysql_write_to_file.py：

```
#!/usr/bin/env python3
import csv
import MySQLdb
import sys
# Path to and name of a CSV output file
output_file = sys.argv[1]
# Connect to a MySQL database
con = MySQLdb.connect(host='localhost', port=3306, db='my_suppliers', \
```

```
user='root', passwd='my_password')
c = con.cursor()
# Create a file writer object and write the header row
filewriter = csv.writer(open(output_file, 'w', newline=''), delimiter=',')
header = ['Supplier Name','Invoice Number','Part Number','Cost','Purchase Date']
filewriter.writerow(header)
# Query the Suppliers table and write the output to a CSV file
c.execute("""SELECT *
FROM Suppliers
WHERE Cost > 700.0;""")
rows = c.fetchall()
for row in rows:
filewriter.writerow(ro ‘w)
```

本例的几行代码几乎是前面例子的子集，所以我只关注新的几行代码。第 2，3，4 行代码导入 CSV，MySQLdb，sys 模块。以便我们可以用它们的方法与 MySQL 数据库交互并将查询输出到 CSV 文件。第 6 行用 sys 模块读文件名和路径并赋值给 output_file。第 8 行用 MySQLdb 模块的 connect()方法连接 my_suppliers，我们在前面创建的 MySQL 数据库。第 10 行创建 cursor，我们用它来对 Suppliers 表执行 SQL 语句然后保存对数据库的修改。第 12 行用 csv 模块的 writer()方法创建 writer 对象 file_writer。第 13 行创建列表变量 header，包含 5 个字串，对应于数据表的列标题。第 14 行用 filewriter 的 writerow()方法，写这字串的列表，用逗号分隔，到 CSV 输出文件。数据库查询只输出数据，不输出列标题，所以这些行的代码确保我们的输出文件有列标题。第 16 到 18 行查询所有列，其中行的 Cost 列的值大于 700.0。查询可以有多行，因为它包含三个双引号。用三个双引号包括你的查询非常有用，你可以格式化你的查询更易于读取。第 19 行到 21 行与前面的例子相似，只是不是打印输出到命令行窗口，第 21 行写输出到 CSV 输出文件。

现在我们有了脚本，我们用脚本来从 Suppliers 数据库查询特定的数据，将输出写入 CSV 文件。在命令行中输入如下命并回车：

```
python 5db_mysql_write_to_file.py output_files\5output.csv
```

你看不到命令行窗口或终端的输出，但是你可以打开输出文件 5output.csv，检查结果。

如你所见，输出文件包括列标题行，显示 5 列的名字，以及数据库表中的 4 行，其中 Cost 列的值大于 700.0。Excel 重新格式化 Purchase Date 列的日期到 M/DD/YYYY，且 Cost 列的值不含逗号或美元符号，但是易于重新格式化这些值，如果有必要的话。

加载数据到数据库表并查询数据库表是两个常用的操作。另一个常用的操作是更新现有的数据库表中的行。下一个例子就是这种情形，解释如何更新现有的表。

更新表中的记录

前面的例子解释了如何增加行到 MySQL 数据库表，以 CSV 文件为输入，并将查询的结果写入 CSV 输出文件。但是有时，不是加载数据到表或查表，而是你要更新表中现有的行。

幸运的，我们可以重用从 CSV 文件读数据来更新表中的行的技术。事实上，这种组装行的值给 SQL 语句然后为 CSV 输入文件的行执行 SQL 语句与前面的例子相似。变化的是 SQL 语句。将 INSERT 语句改为 UPDATE 语句。因为我们已经熟悉如何用 CSV 输入文件来加载数据到表。我们来学习如可用 CSV 输入文件来更新现在的数据库表中的记录。

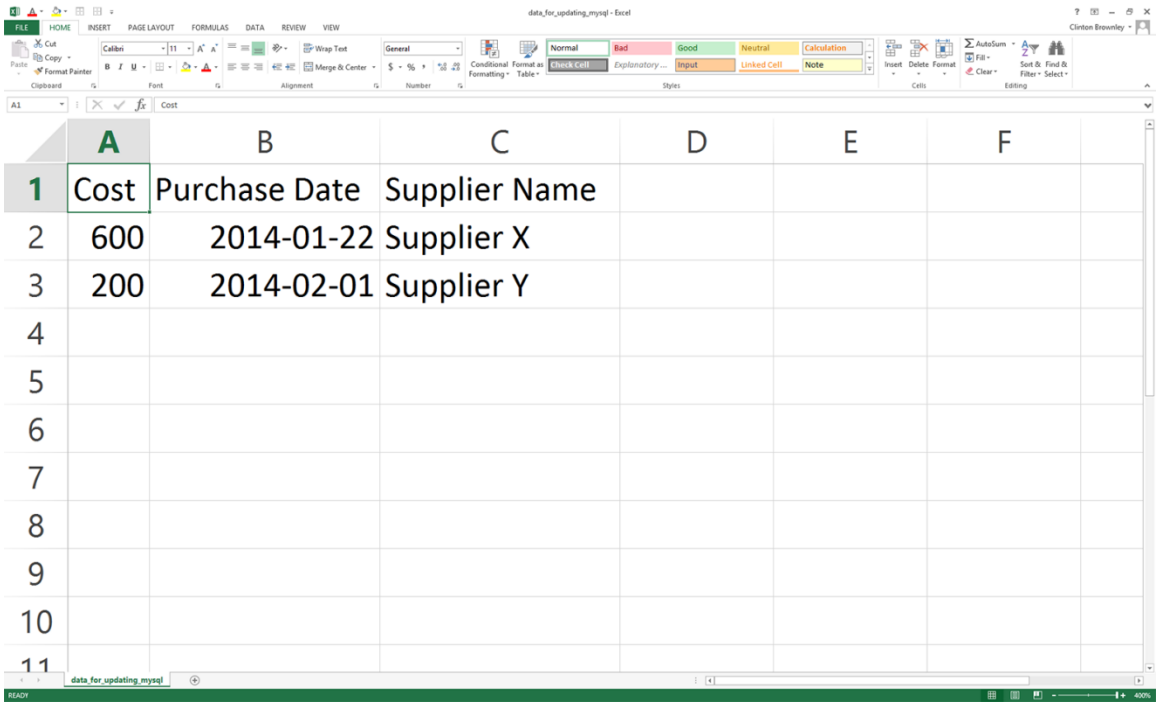
在文本编辑中输入以下并保存为 6db_mysql_update_from_csv.py:

```
1 #!/usr/bin/env python3
2 import csv
3 import MySQLdb
4 import sys
5
6 # Path to and name of a CSV input file
7 input_file = sys.argv[1]
8 # Connect to a MySQL database
9 con = MySQLdb.connect(host='localhost', port=3306, db='my_suppliers', \
10 user='root', passwd='my_password')
11 c = con.cursor()
12
13 # Read the CSV file and update the specific rows
14 file_reader = csv.reader(open(input_file, 'r', newline=''), delimiter=',')
15 header = next(file_reader, None)
16 for row in file_reader:
17 data = []
18 for column_index in range(len(header)):
19 data.append(str(row[column_index]).strip())
20 print(data)
21 c.execute("""UPDATE Suppliers SET Cost=%s, Purchase_Date=%s \
22 WHERE Supplier_Name=%s;""", data)
23 con.commit()
24 # Query the Suppliers table
25 c.execute("SELECT * FROM Suppliers")
26 rows = c.fetchall()
27 for row in rows:
28 output = []
29 for column_index in range(len(row)):
30 output.append(str(row[column_index]))
31 print(output)
```

本例中的所有代码看起来相似，第 2-4 行导入三个 Python 模块，我们用它们的方法来读 CSV 输入文件，与 MySQL 数据库交互，读命令行输入。第 7 行将 CSV 输入文件赋值给变量 input_file。第 10 行连接 my_suppliers 数据库，用我们前面例子中相同的连接参数，第 12 行创建 cursor 对象，它可以用来执行 SQL 查询并保存数据库的变更。第 15 到 24 行与本章的第一个例子几乎相同。不同的是第 22 行，其中 UPDATE 语句替代前面的 INSERT 语句。UPDATE 语句是你要指定哪条记录哪列属性你想更新。这种情况，我们想要为特定的 Supplier Names 的子集更新 Cost 和 Purchase Date 的值。像前面的例子，有许多的占位符 %s 作为查询的值，CSV 输入文件的数据顺序应与查询属性的顺序一致。本例，从左到右，查询的属性为 Cost, Purchase_Date, and Supplier_Name，所以 CSV 输入文件的列从左到右应为 Cost, Purchase Date, 和 Supplier Name。最后，第 27 到 32 行代码基本上与早期的代码一样。这些行的代码获得 Suppliers 表的所有行并打印每一行到命令行或终端，列之间用逗号分开。

现在我们所要的是含有我们想要更新到数据库表的的数据的 CSV 输入文件。

1. 打开 Excel.
2. 如图 4-15 添加数据。
3. 保存文件为 data_for_updating_mysql.csv.



	A	B	C	D	E	F
1	Cost	Purchase Date	Supplier Name			
2	600	2014-01-22	Supplier X			
3	200	2014-02-01	Supplier Y			
4						
5						
6						
7						
8						
9						
10						

图 4-15. data_for_updating_mysql.csv 文件的示例数据,用 Excel worksheet 显示

我们已经有了脚本和 CSV 输入文件，我们用脚本和输入文件来更新 Suppliers 表中的特定记录。

在命令行中输入以下并回车：

```
python 6db_mysql_update_from_csv.py data_for_updating_mysql.csv
```

在 windows 系统,你可以看到图 4-16 所示的打印输出到命令行窗口。前两行是 CSV 输入文件中的数据，后面的行是更新记录后的表数据。

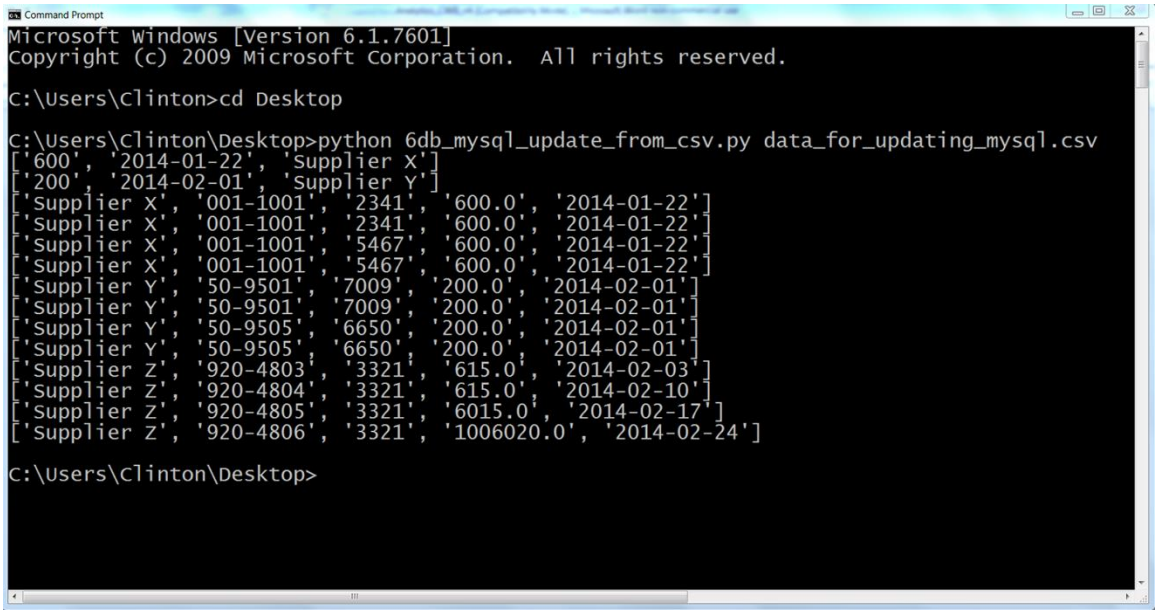


图 4-16. 用 CSV 文件的数据更新 MySQL 数据表的记录

输出显示两个列表的值由 CSV 输入文件的两行数据创建，不包括标题行。你看到两个列表表是因为每一个列表由方括号（[]）封闭且值由逗号分隔。对于 Supplier X，Cost 为 600 且 Purchase Date 为 2014-01-22。对于 Supplier Y，Cost 值为 200 且 Purchase Date 为 2014-02-01。接着输出显示 12 行从数据库表获得的执行更新后的值。各行单独打印，每行的值由空格分开。记得原来的 Supplier X 的 Cost 和 Purchase Date 分别为 500 和 750 和 2014-01-20。相似的原来的 Supplier Y 的 Cost 和 Purchase Date 分别为 250 和 125 和 2014-01-30 和 2013-02-03。如你从打印输出所见，Supplier X 和 Supplier Y 的这些值已经由 CSV 输入文件提供的值更新。

要确认与 Supplier X 和 Supplier Y 相关的 8 行数据已经在数据库表中更新，返回 MySQL 命令行客户端，输入如下并回车：

```
SELECT * FROM Suppliers;
```

回车以后你可以看到表列出了 Suppliers 数据表中的列和各列中 12 行数据，如图 4-17 所示。你可以看到 8 行与 Supplier X 和 Supplier Y 相关的数据已被 CSV 输入文件所更新。

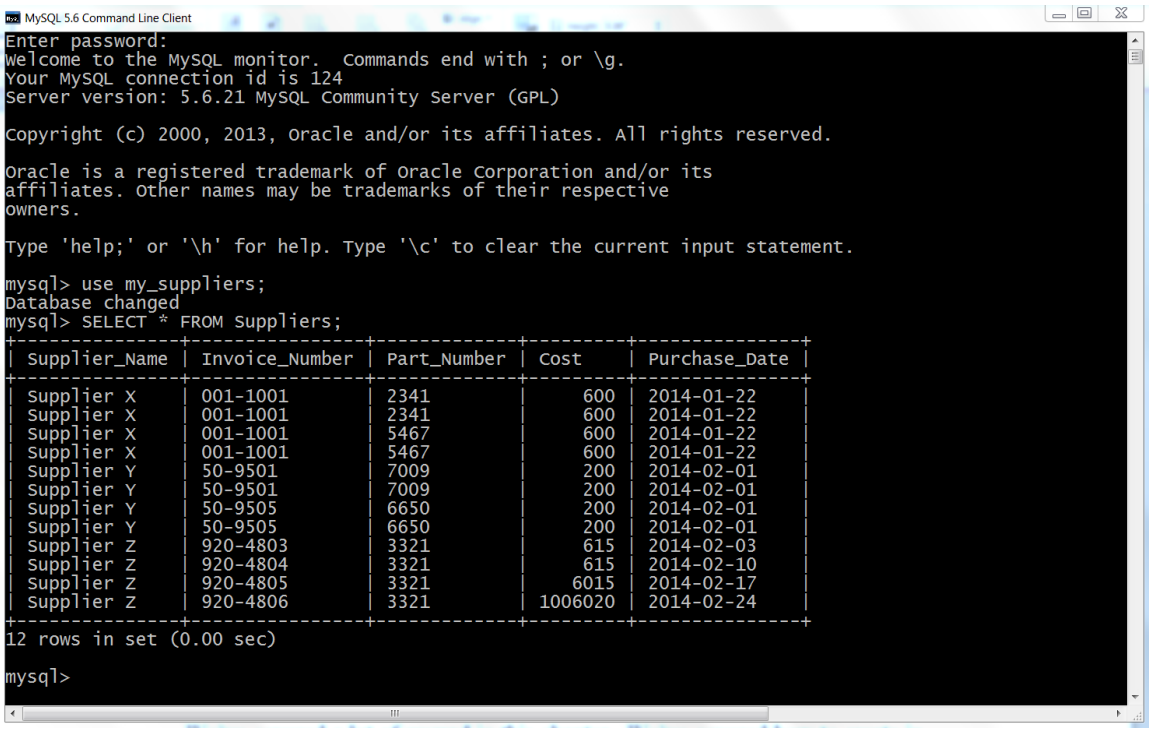


图 4-17. 记录更新后查询 Suppliers 数据表的结果

这一章我们学习了很多。我们讨论了如何用 sqlite3 创建在内存中的和持久化的数据库。并与这些数据库的数据表交互，我们也看到了如何用 python 创建 MySQL 数据库和表，访问 MySQL 数据库和表，加载 CSV 文件的数据到 MySQL 数据库表，用 CSV 输入文件的数据更新 MySQL 数据库记录，将查询结果写入 CSV 输出文件。如果你按照本章的例子，你已经有 6 个 python 脚本了。本章的例子中最好的部分是你可以访问数据中的数据，商务中最常见的操作。本章关注于 MySQL 数据库系统，但如我们在本章开始所讨论的，现在的商务应用中有许多的数据库系统。如，你可以学习 PostgreSQL 数据库系统，你可以看到流行的 python 连接 PostgreSQL 的包，在 Psycopg 和 PyPI 网站。相似的，你可以学习 Oracle 数据库系统，在 SourceForge 和 PyPI 上有连接 Oracle 数据库系统的包的信息。另外，有一个流行的 Python SQL 工具叫做 SQLAlchemy 提供了与 SQLite, MySQL, PostgreSQL, Oracle 以及其它数据库的连接，用 python2 和 python3 的版本。

到此，你已学习了如何访问，导航，处理 CSV 文件，Excel workbooks 和数据库，商务中最常见的数据源。下一步探索一些应用，看如何组合这些技能来完成特定的任务。首先，我们讨论如何从多个文件中找到项目的子集。第二个应用是计算输入文件中的统计量。最后第三个应用是展示如何解析文本文件和计算统计量。学习了这些例子，你就能理解如何组合这些技能来完成特定的任务。