

Chapter 4: GUI

Storing Data in Our MySQL Database via Our GUI

Storing Data in Our MySQL Database via Our GUI

In this chapter, we will learn how to install and use a MySQL database and connect it to our GUI.

MySQL is a full-fledged Structured Query Language (SQL) database server and comes with a very nice GUI of its own so that we can view and work with the data. We will create a database, insert data into our database, and then see how we can modify, read, and delete data.

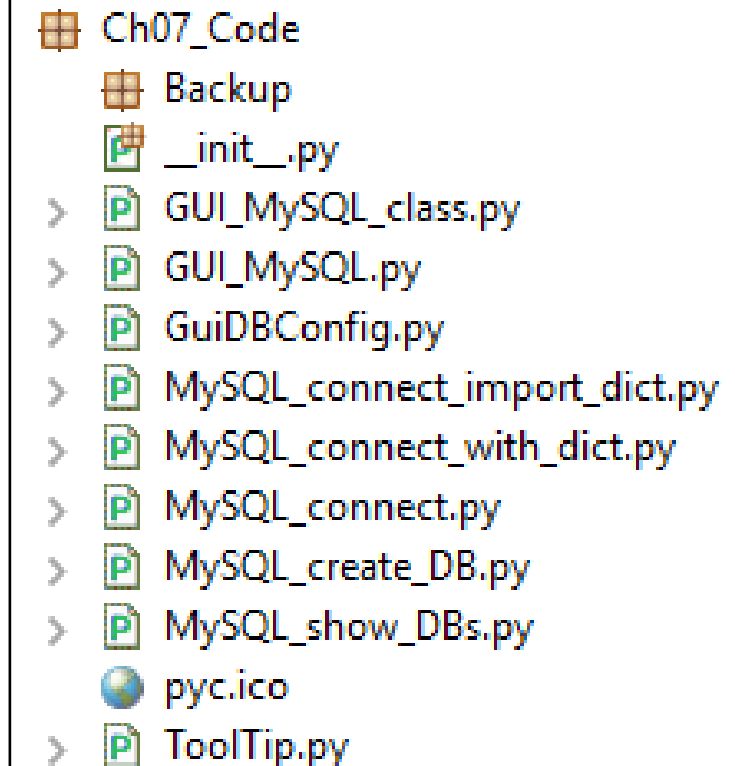
Here, you will learn how to increase your programming skills by adding SQL to your programming toolbox.

The first recipe in this chapter will show you how to install the free MySQL Community Edition.

In this chapter, we will enhance our Python GUI by connecting the GUI to a MySQL database.

Content:

1. Installing and connecting to a MySQL server from Python
2. Configuring the MySQL database connection
3. Designing the Python GUI database
4. Using the SQL INSERT command
5. Using the SQL UPDATE command
6. Using the SQL DELETE command
7. Storing and retrieving data from our MySQL database
8. Using MySQL Workbench



1_ Installing and connecting to a MySQL server from Python

Before we can connect to a MySQL database, we have to connect to the *MySQL server*. In order to do this, we need to know the IP address of the MySQL server as well as the port it is listening on.

We also have to be a registered user with a password in order to be *authenticated* by the MySQL server.

Getting ready

You will need to have access to a running MySQL server instance, as well as have administrator privileges in order to create databases and tables.

How to do it...

Let's look at how to install and connect to a MySQL server from Python:

1. Download the MySQL Installer.



There is a free MySQL Community Edition available from the official MySQL website. You can download and install it on your local PC from <http://dev.mysql.com/downloads/windows/installer/>.

2. Run the installation:

Choosing the right file:

- If you have an online connection while running the MySQL Installer, choose the `mysql-installer-web-community` file.
- If you do NOT have an online connection while running the MySQL Installer, choose the `mysql-installer-community` file.

Note: MySQL Installer is 32 bit, but will install both 32 bit and 64 bit binaries.

Online Documentation

- [MySQL Installer Documentation and Change History](#)

Please report any bugs or inconsistencies you observe to our [Bugs Database](#).

Thank you for your support!

Generally Available (GA) Releases

MySQL Installer 8.0.16

Select Operating System:
Microsoft Windows ▼

[Looking for previous GA versions?](#)

Windows (x86, 32-bit), MSI Installer (mysql-installer-web-community-8.0.16.0.msi)	8.0.16	20.0M	Download
MD5: 08b01313c1f7a7aa26a4b6bc1167c604 Signature			
Windows (x86, 32-bit), MSI Installer	8.0.16	373.4M	Download

3. Choose a password for the **root** user and, optionally, add more users:

Accounts and Roles


Root Account Password
Enter the password for the root account. Please remember to store this password in a secure place.

MySQL Root Password:








Repeat Password:

Password Strength: **Weak**

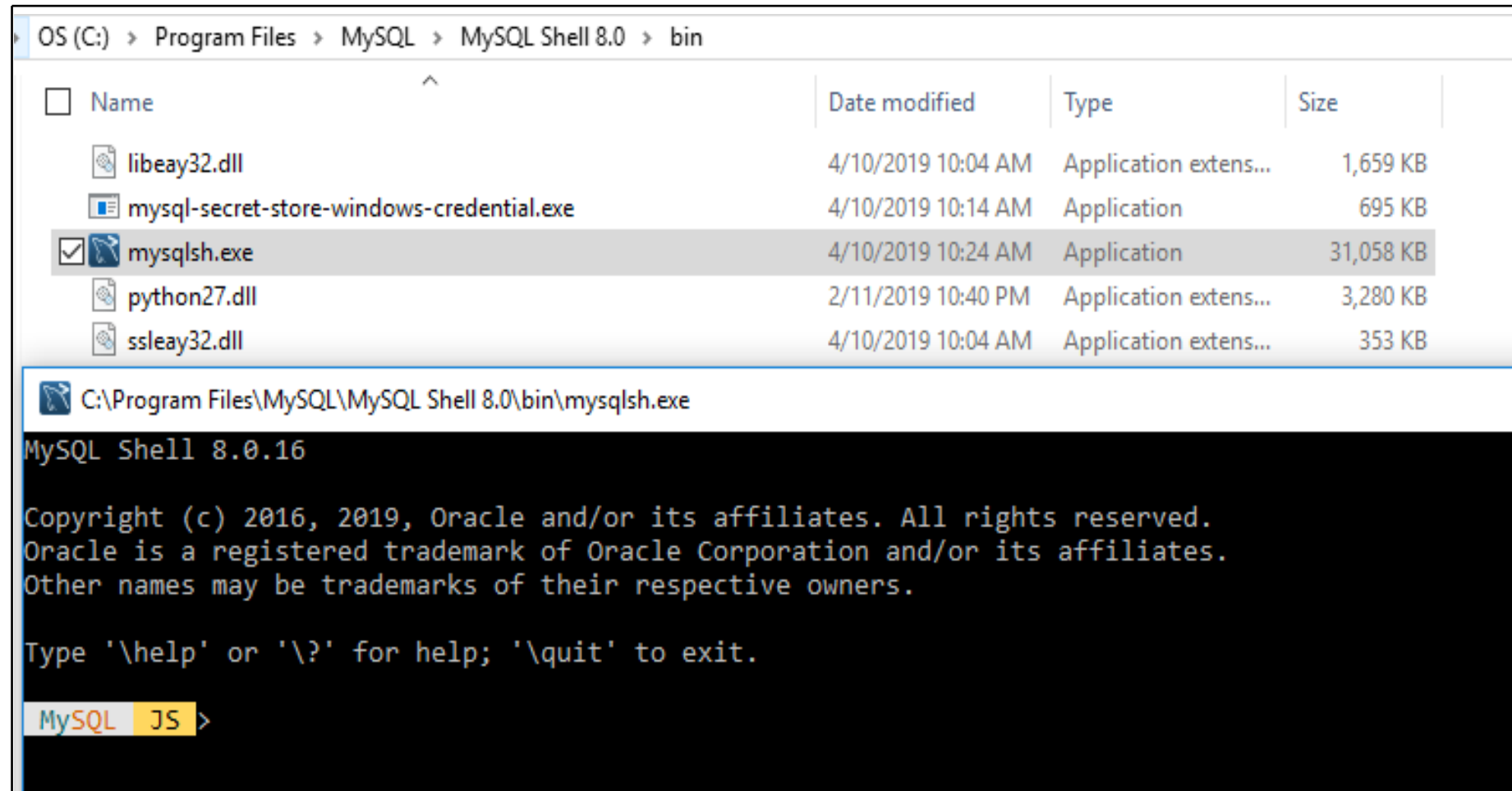
MySQL User Accounts
Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.

	MySQL Username	Host	User Role	
	Burkhard	%	DB Admin	<div>Add User Edit User Delete</div>

4. Verify that you have the `\Python37\Lib\site-packages\mysql\connector` folder:

Python37 > Lib > site-packages > mysql > connector			
<input type="checkbox"/> Name		Date modified	Type
<input type="checkbox"/>  <code>__pycache__</code>		5/29/2019 10:50 AM	File folder
 <code>django</code>		5/29/2019 10:22 AM	File folder
 <code>locales</code>		5/29/2019 10:50 AM	File folder
 <code>__init__.py</code>		3/28/2019 6:40 PM	Python File
 <code>abstracts.py</code>		3/28/2019 6:40 PM	Python File
 <code>authentication.py</code>		3/28/2019 6:40 PM	Python File
 <code>catch23.py</code>		3/28/2019 6:40 PM	Python File

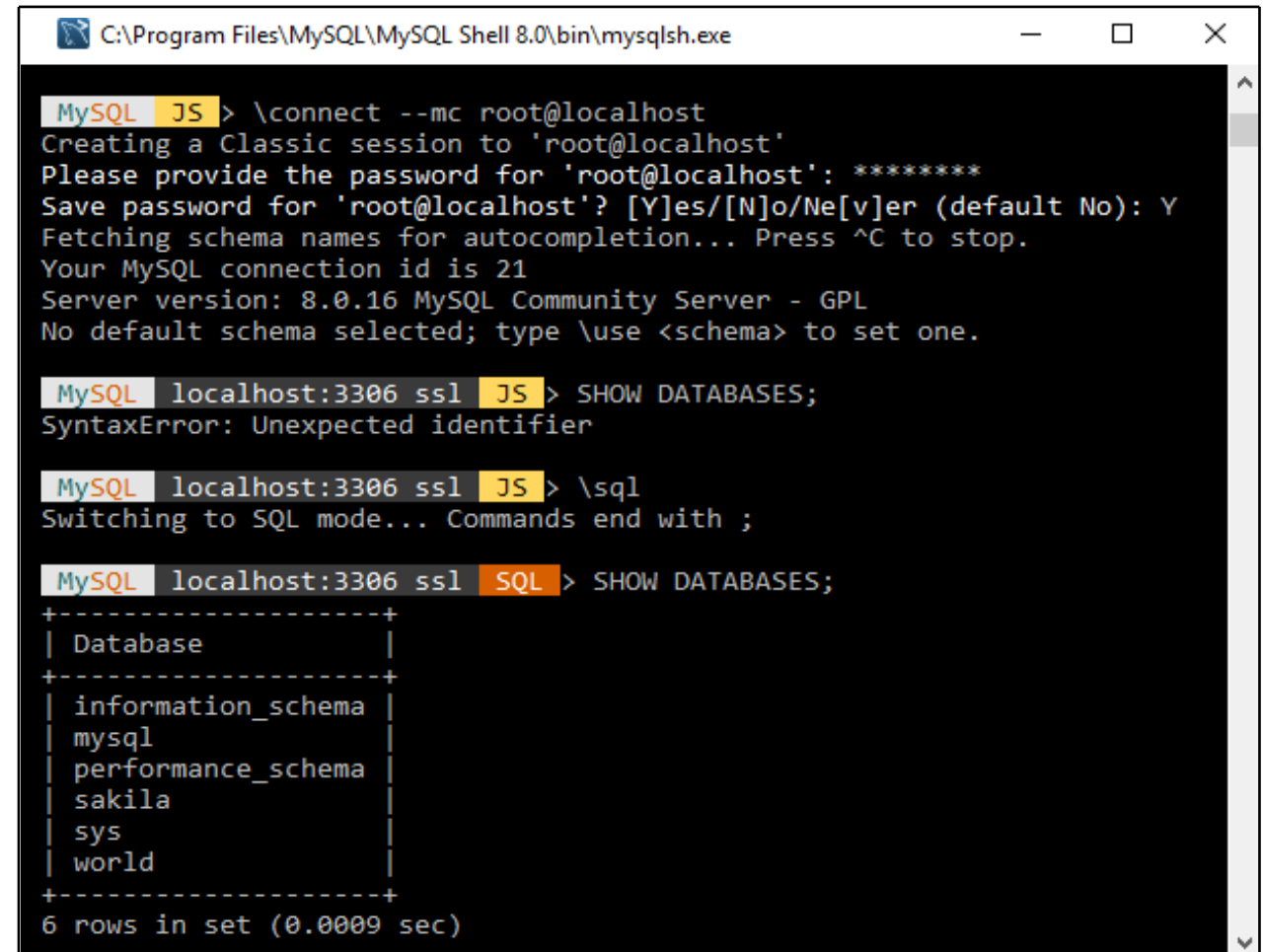
5. Open the `mysqlsh.exe` executable and double-click on it to run it:



6.Type \sql in the prompt to get into SQL mode.

7.In the MySQL> prompt, type SHOW DATABASES.

Then, press *Enter*:



```
C:\Program Files\MySQL\MySQL Shell 8.0\bin\mysqlsh.exe

MySQL JS > \connect --mc root@localhost
Creating a Classic session to 'root@localhost'
Please provide the password for 'root@localhost': *****
Save password for 'root@localhost'? [Y]es/[N]o/[v]er (default No): Y
Fetching schema names for autocompletion... Press ^C to stop.
Your MySQL connection id is 21
Server version: 8.0.16 MySQL Community Server - GPL
No default schema selected; type \use <schema> to set one.

MySQL localhost:3306 ssl JS > SHOW DATABASES;
SyntaxError: Unexpected identifier

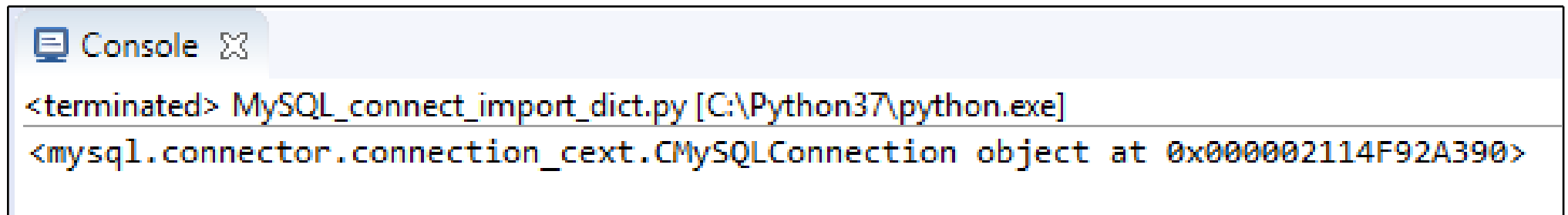
MySQL localhost:3306 ssl JS > \sql
Switching to SQL mode... Commands end with ;

MySQL localhost:3306 ssl SQL > SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sakila |
| sys |
| world |
+-----+
6 rows in set (0.0009 sec)
```

8. Create a new Python module and save it as `MySQL_connect.py`:

```
import mysql
conn = mysql.connector.connect(user=<adminUser>,
password=<adminPwd>, host='127.0.0.1')
print(conn)
conn.close()
```

9. If running the preceding code results in the following output, then we have successfully connected:

A screenshot of a console window with a light blue header bar containing the word "Console" and a close button. The main area is white and displays two lines of text: the first line is "<terminated> MySQL_connect_import_dict.py [C:\Python37\python.exe]" and the second line is "<mysql.connector.connection_cext.CMySQLConnection object at 0x000002114F92A390>".

```
<terminated> MySQL_connect_import_dict.py [C:\Python37\python.exe]
<mysql.connector.connection_cext.CMySQLConnection object at 0x000002114F92A390>
```

2_ Configuring the MySQL database connection

In the previous recipe, we used the shortest way to connect to a MySQL server, that is, by hardcoding the credentials that are required for authentication in the connect method. While this is a fast approach for early development, we definitely do not want to expose our MySQL server credentials to anyone. Instead, we want to *grant* permission to specific users so that they can access databases, tables, views, and related database commands.

A much safer way to be authenticated by a MySQL server is by storing the credentials in a configuration file, which is what we will do in this recipe. We will use our configuration file to connect to the MySQL server and then create our own database on the MySQL server.

Getting ready

Access to a running MySQL server with administrator privileges is required to run the code shown in this recipe.



The previous recipe shows how to install the free *Community Edition* of MySQL server. The administrator privileges will allow you to implement this recipe.

How to do it...

Let's look at how to perform this recipe:

1. First, we will create a dictionary in the same module where the `MySQL_connect.py` code is. Sequentially, we will do the following:

1. Open `MySQL_connect.py`
and save it as `MySQL_connect_with_dict.py`.

2. Add the following code to the module:

```
# create dictionary to hold connection info
dbConfig = {
    'user': <adminName>,           # use your admin name
    'password': <adminPwd>,        # use your real password
    'host': '127.0.0.1',          # IP address of localhost
}
```

2. Write the following code below `dbConfig`:

```
import mysql.connector
# unpack dictionary credentials
conn = mysql.connector.connect(**dbConfig)
print(conn)
```

3. Run the code to make sure it works.

4. Create a new module, `GuiDBConfig.py`, and place the following code in it:

```
# create dictionary to hold connection
info dbConfig = {
    'user': <adminUser>,      # your user name
    'password': <adminPwd>,   # your password
    'host': '127.0.0.1',      # IP address
}
```


5. Now, open `MySQL_connect_with_dict.py` and save it as `MySQL_connect_import_dict.py`.

6. Import `GuiDBConfig` and unpack the dictionary, as shown here:

```
import GuiDBConfig as guiConf
# unpack dictionary credentials
conn = mysql.connector.connect(**guiConf.dbConfig)
print(conn)
```

7. Create a new Python module and save it as `MySQL_create_DB.py`.

Next, add the following code:

```
import mysql.connector
import ch07_Code.GuiDBConfig as guiConf

GUIDB = 'GuiDB'

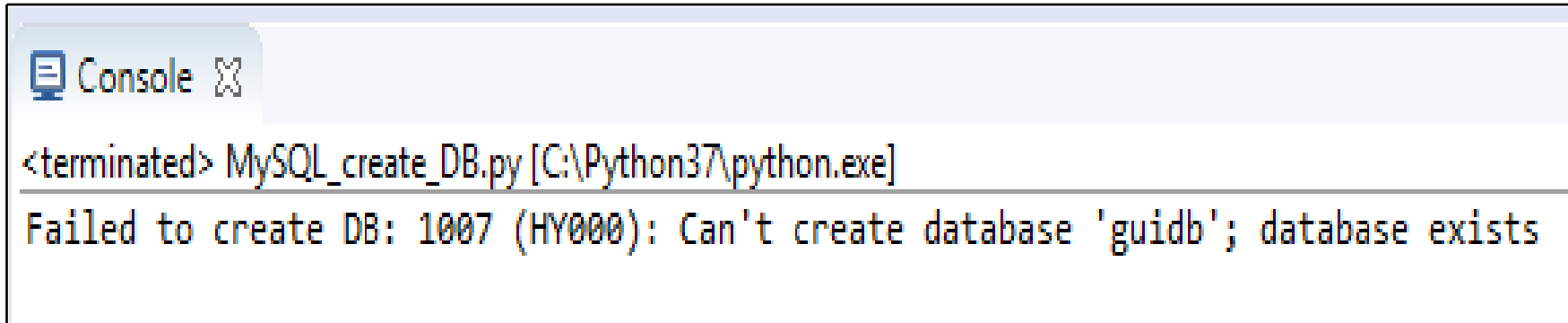
# unpack dictionary credentials
conn = mysql.connector.connect(**guiConf.dbConfig)

cursor = conn.cursor()
```

```
try:
    cursor.execute("CREATE DATABASE {}
                    DEFAULT CHARACTER SET 'utf8'".format(GUI_DB))
except mysql.connector.Error as err:
    print("Failed to create DB: {}".format(err))

conn.close()
```

8. Execute `MySQL_create_DB.py` twice:

A screenshot of a console window titled "Console" with a close button. The window shows the output of a Python script. The first line is a prompt indicating the script has terminated: "<terminated> MySQL_create_DB.py [C:\Python37\python.exe]". The second line is an error message: "Failed to create DB: 1007 (HY000): Can't create database 'guidb'; database exists".

```
<terminated> MySQL_create_DB.py [C:\Python37\python.exe]
Failed to create DB: 1007 (HY000): Can't create database 'guidb'; database exists
```

9. Create a new Python module and save it as `MySQL_show_DBs.py`. Then, add the following code:

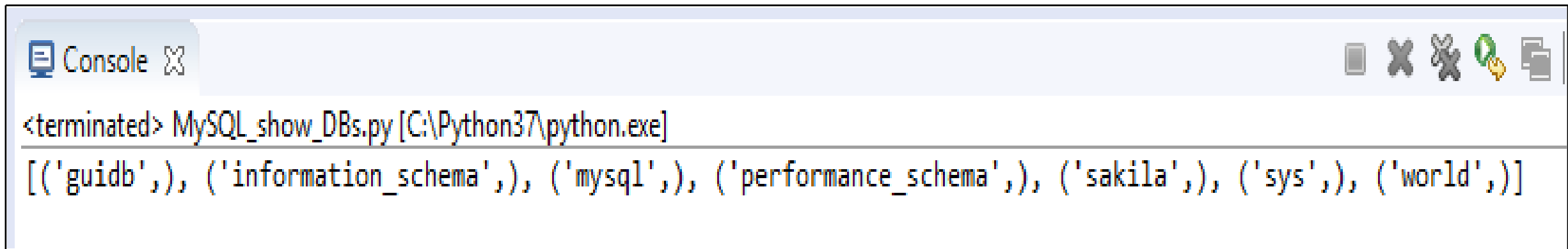
```
import mysql.connector
import GuiDBConfig as guiConf

# unpack dictionary credentials
conn = mysql.connector.connect(**guiConf.dbConfig)
cursor = conn.cursor()

cursor.execute("SHOW DATABASES")
print(cursor.fetchall())

conn.close()
```

10. Running the preceding code gives us the following output:

A screenshot of a console window titled "Console". The window shows the output of a Python script named "MySQL_show_DBs.py" executed from "C:\Python37\python.exe". The output is a list of tuples representing database schemas: [('guidb',), ('information_schema',), ('mysql',), ('performance_schema',), ('sakila',), ('sys',), ('world',)].

```
<terminated> MySQL_show_DBs.py [C:\Python37\python.exe]  
[('guidb',), ('information_schema',), ('mysql',), ('performance_schema',), ('sakila',), ('sys',), ('world',)]
```

3_ Designing the Python GUI database

Before we start creating tables and inserting data into them, we have to design the database. Unlike changing local Python variable names, changing a database **schema** once it has been created and loaded with data is not that easy.

We would have to **DROP** the table, which means we would lose all the data that was in the table. So, before dropping a table, we would have to extract the data, save the data in a temporary table or other data format, and then **DROP** the table, recreate it, and finally **reimport** the original data.

I hope you are getting the picture of how tedious this could be.

Designing our GUI MySQL database means that we need to think about what we want our Python application to do with it and then choose names for our tables that match the intended purpose.

Getting ready

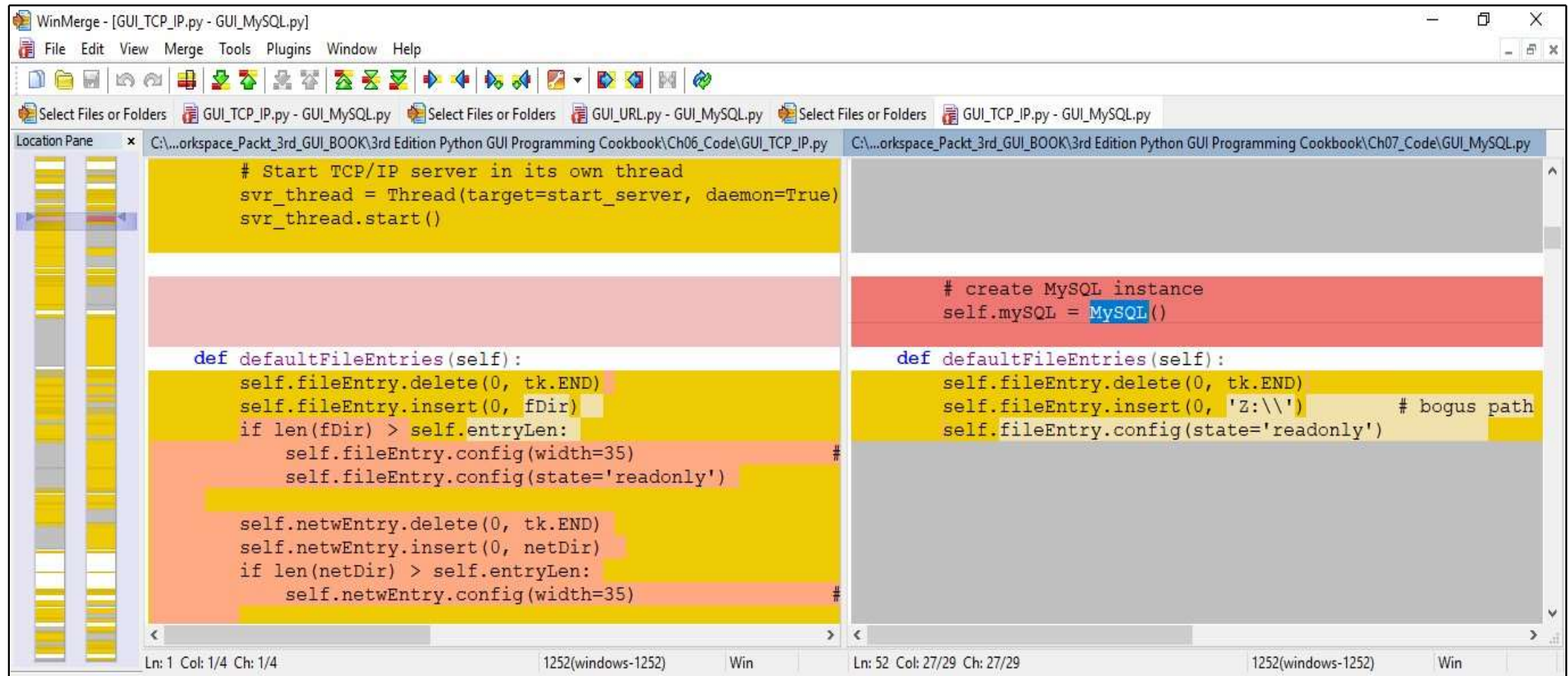
We will be working with the MySQL database we created in the previous recipe, *Configuring the MySQL database connection*. A running instance of MySQL is necessary and the two previous recipes show you how to install MySQL, all the necessary additional drivers, and how to create the database we are using in this chapter.

How to do it...

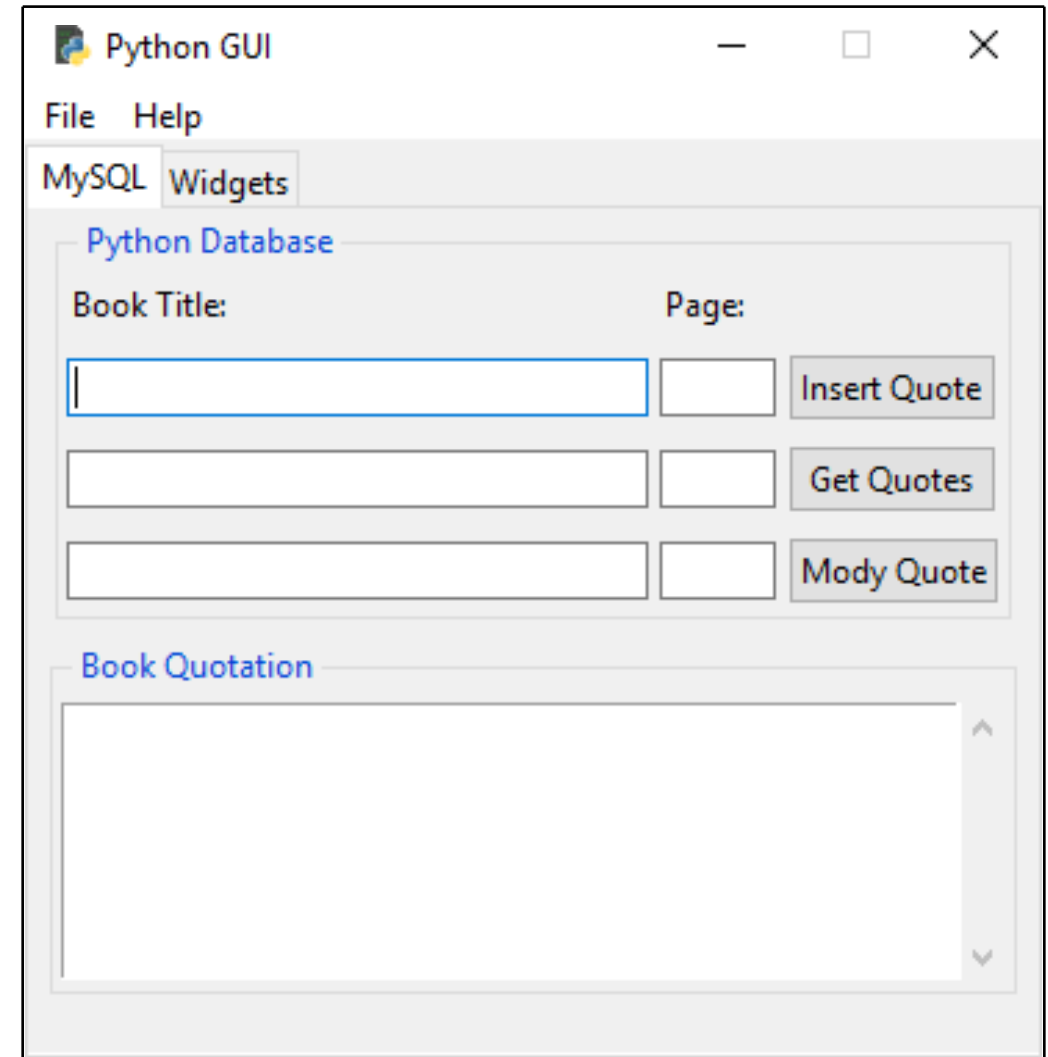
In this recipe, we are starting with the `GUI_TCP_IP.py` file from the previous chapter. We will move the widgets from our Python GUI between the two tabs we created in the previous recipes in order to organize our Python GUI so that it can connect to a MySQL database. Let's take a look at how can we complete this recipe:

- 1. Open `GUI_TCP_IP.py` and save it as `GUI_MySQL.py`.**
- 2. Download the full code from the Packt website.**

3. Use a tool such as WinMerge to compare the two versions of the GUI:



4. Run the code located in **GUI_MySQL.py**. You will observe the following output:



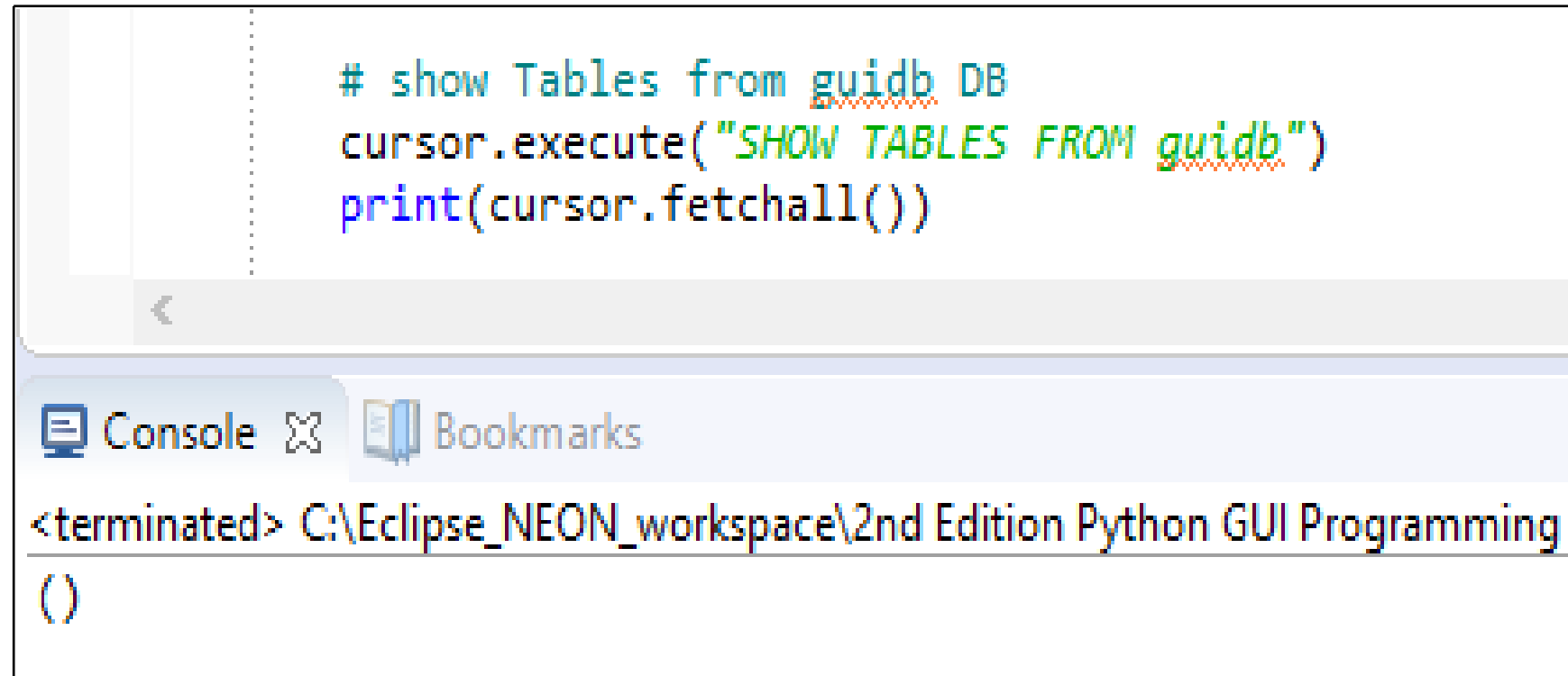
1. Now, open `MySQL_create_DB.py` and save it as `MySQL_show_DB.py`.
2. Replace the `try...catch` block with the following code:

```
# unpack dictionary credentials
conn = mysql.connect(**guiConf.dbConfig)
# create cursor
cursor = conn.cursor()

# execute command
cursor.execute("SHOW TABLES FROM guiddb")
print(cursor.fetchall())

# close connection to MySQL
conn.close()
```

5. Run the code and observe the output:



The screenshot shows an IDE window with a code editor and a console. The code editor contains the following Python code:

```
# show Tables from guidb DB
cursor.execute("SHOW TABLES FROM guidb")
print(cursor.fetchall())
```

Below the code editor, there are two tabs: "Console" and "Bookmarks". The "Console" tab is active, showing the output of the code execution:

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming
()
```

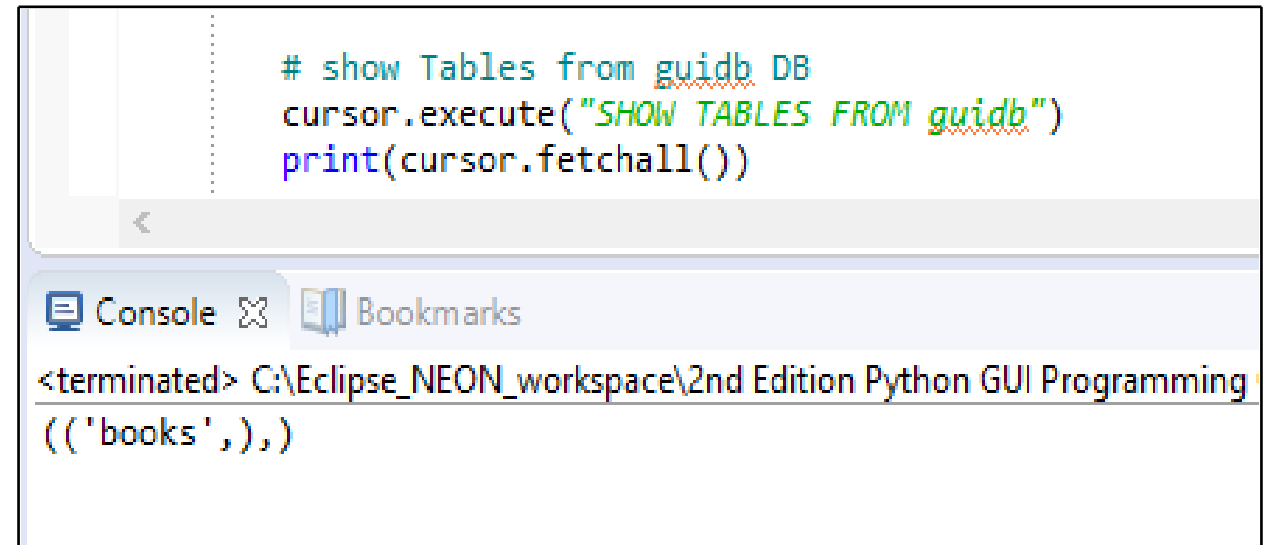
1. Create a module similar to `GUI_MySQL_class.py`.
2. Add and run the following code:

```
# connect by unpacking dictionary credentials
conn = mysql.connect(**guiConf.dbConfig)
# create cursor
cursor = conn.cursor()
# select DB  cursor.execute("USE guidb")

# create Table inside DB
cursor.execute("CREATE TABLE Books (
    Book_ID INT NOT NULL AUTO_INCREMENT,
    Book_Title VARCHAR(25) NOT NULL,
    Book_Page INT NOT NULL,
    PRIMARY KEY (Book_ID)
) ENGINE=InnoDB")

# close connection to MySQL
conn.close()
```

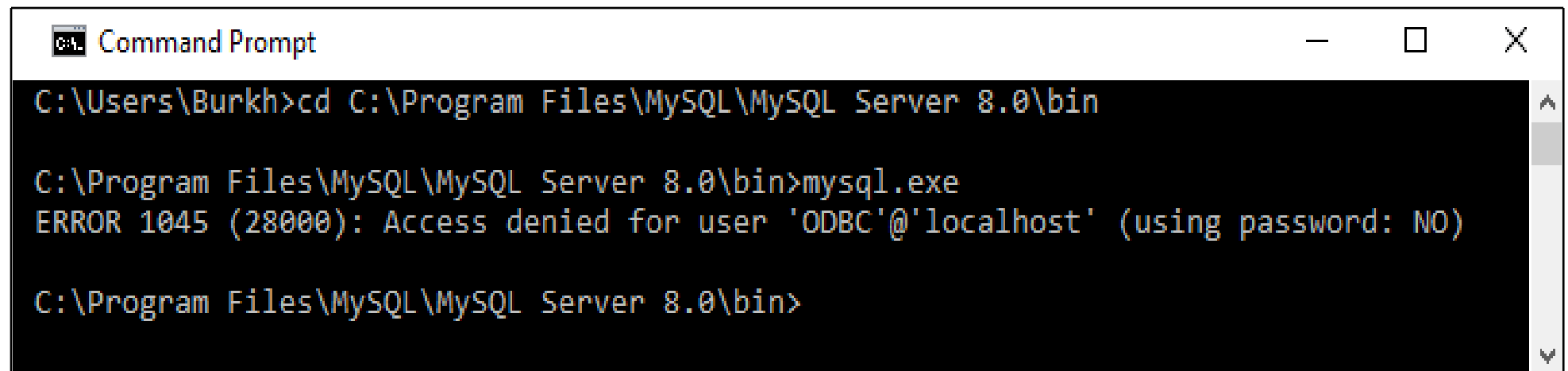
6. Run the following code, which is located in `GUI_MySQL_class.py`:



```
# show Tables from guidb DB
cursor.execute("SHOW TABLES FROM guidb")
print(cursor.fetchall())
```

<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming
 (('books',),),)

7. Open Command Prompt and navigate to `mysql.exe`:



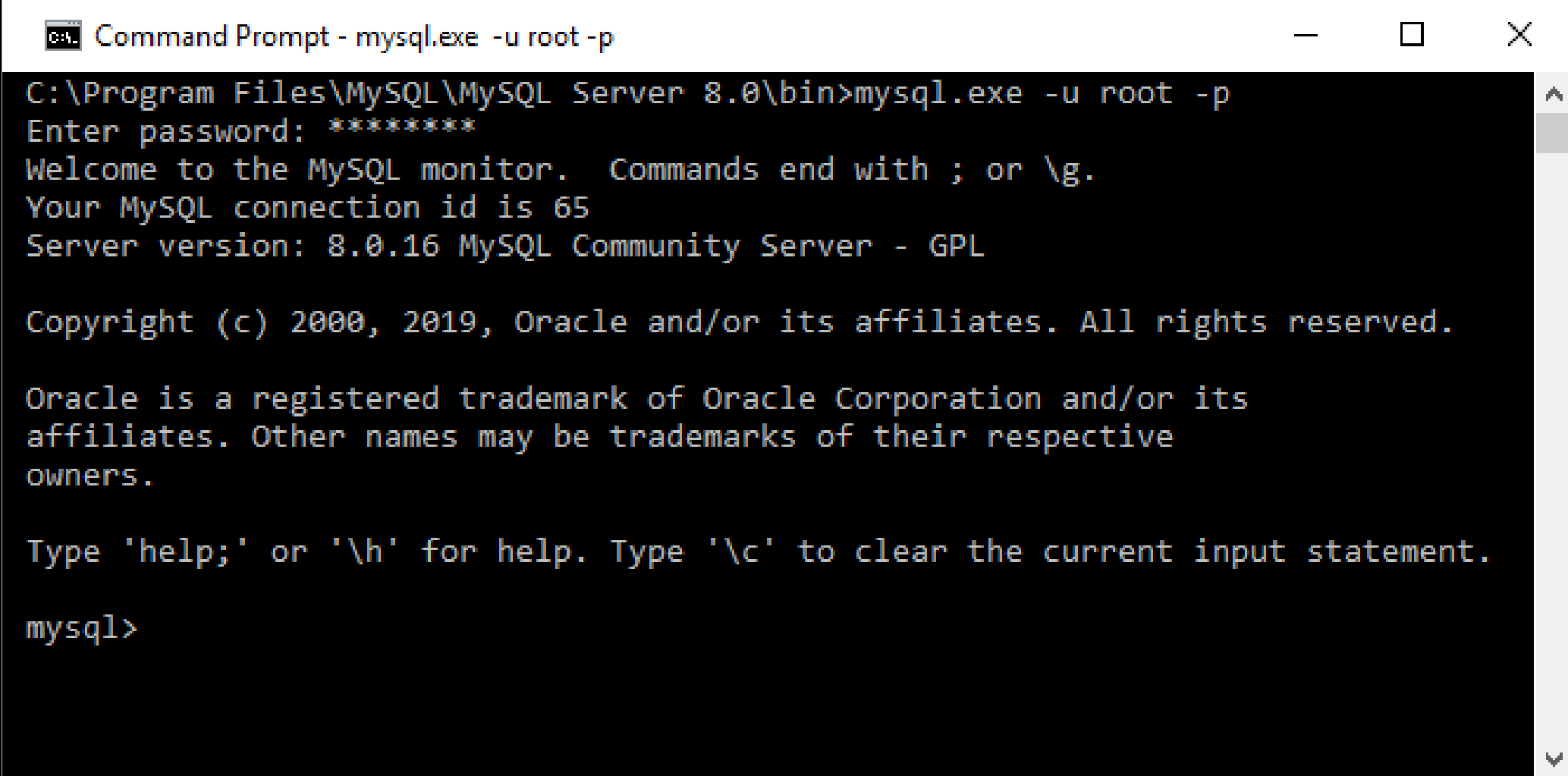
```
Command Prompt

C:\Users\Burkh>cd C:\Program Files\MySQL\MySQL Server 8.0\bin

C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql.exe
ERROR 1045 (28000): Access denied for user 'ODBC'@'localhost' (using password: NO)

C:\Program Files\MySQL\MySQL Server 8.0\bin>
```

8. Run `mysql.exe`:



```
Command Prompt - mysql.exe -u root -p

C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql.exe -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 65
Server version: 8.0.16 MySQL Community Server - GPL

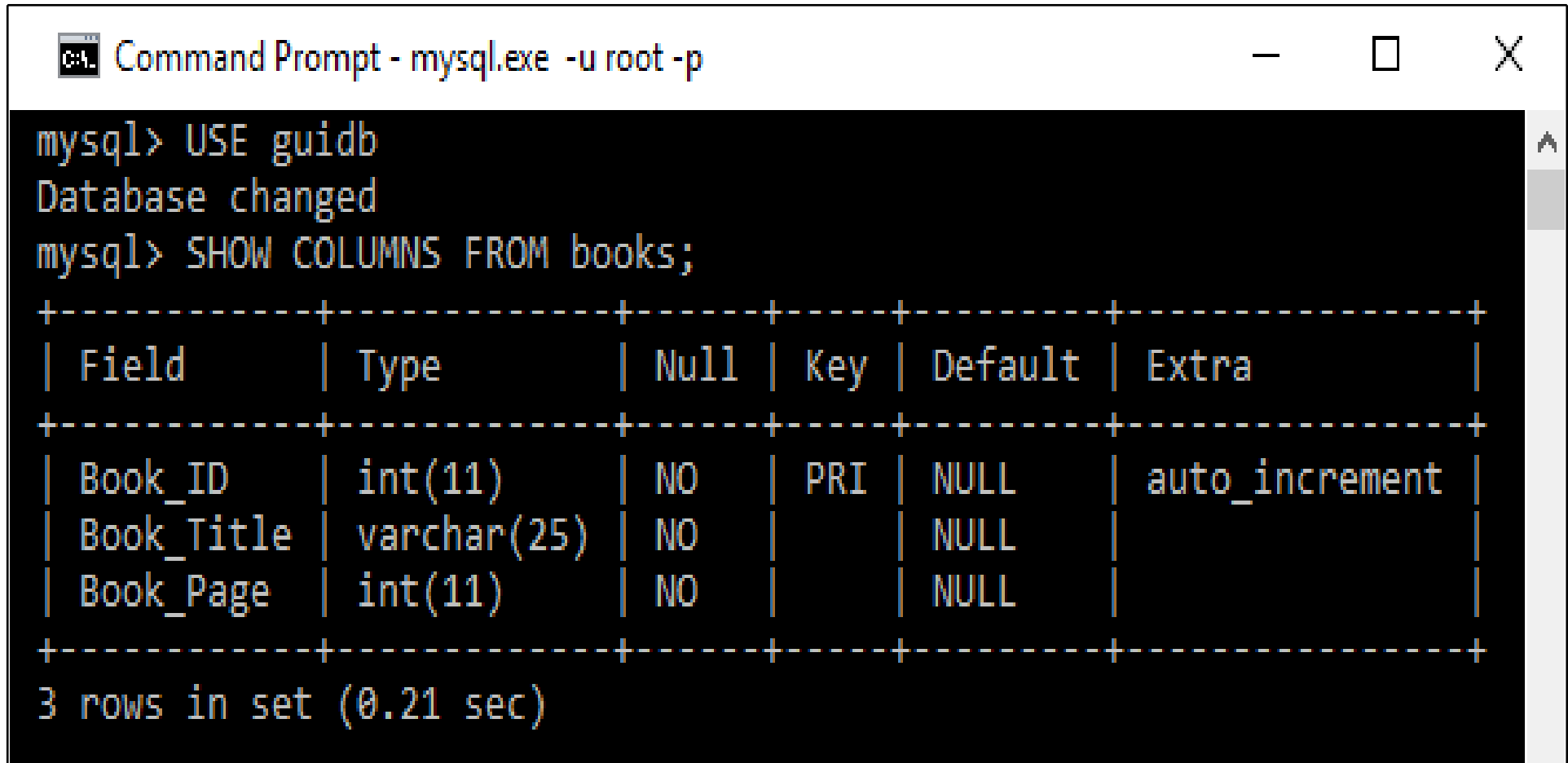
Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql>
```

9. Enter the **SHOW COLUMNS FROM books;** command:



```
Command Prompt - mysql.exe -u root -p

mysql> USE guidb
Database changed
mysql> SHOW COLUMNS FROM books;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra           |
+-----+-----+-----+-----+-----+-----+
| Book_ID    | int(11)       | NO   | PRI | NULL    | auto_increment |
| Book_Title | varchar(25)   | NO   |     | NULL    |                 |
| Book_Page  | int(11)       | NO   |     | NULL    |                 |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.21 sec)
```


10. Create a second table by running the following code:

```
# select DB
```

```
cursor.execute("USE guidb")
```

```
# create second Table inside DB
```

```
cursor.execute("CREATE TABLE Quotations (  
    Quote_ID INT,  
    Quotation VARCHAR(250),  
    Books_Book_ID INT,  
    FOREIGN KEY (Books_Book_ID)  
    REFERENCES Books(Book_ID)  
    ON DELETE CASCADE  
) ENGINE=InnoDB")
```

11. Execute the SHOW TABLES command:

```
# show Tables from guidb DB
cursor.execute("SHOW TABLES FROM guidb")
print(cursor.fetchall())
```

<

Console ✕ Bookmarks

<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming
 (('books',), ('quotations',))

12. Execute the SHOW COLUMNS command:

```
# execute command
cursor.execute("SHOW COLUMNS FROM quotations")
print(cursor.fetchall())
```

Console ✕ Bookmarks

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\GUI_MySQL_class.py
(('Quote_ID', 'int(11)', 'NO', 'PRI', None, 'auto_increment'), ('Quotation', 'varchar(250)', 'YES',
```

13. Execute SHOW COLUMNS again with pprint:

```
from pprint import pprint
# execute command
cursor.execute("SHOW COLUMNS FROM quotations")
pprint(cursor.fetchall())
```

 Console  Bookmarks

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook
(('Quote_ID', 'int(11)', 'NO', 'PRI', None, 'auto_increment'),
 ('Quotation', 'varchar(250)', 'YES', '', None, ''),
 ('Books_Book_ID', 'int(11)', 'YES', 'MUL', None, ''))
```

4_ Using the SQL INSERT command

This recipe presents the entire Python code that shows you how to create and drop MySQL databases and tables, as well as how to display the existing databases, tables, columns, and data of our MySQL instance.

After creating the database and tables, we will insert data into the two tables we will create in this recipe.

We are using a primary key to foreign key relationship to connect the data of the two tables.

We will go into detail about how this works in the following two recipes, where we will modify and delete the data in our MySQL database.

Getting ready

This recipe builds on the MySQL database we created in the previous recipe, *Designing the Python GUI database*, and also shows you how to drop and recreate the GuiDB.

Dropping the database, of course, deletes all the data the database has in its tables, so we'll show you how to reinsert that data as well.

How to do it...

The entire code in the `GUI_MySQL_class.py` module is present in the code folder for this chapter, which you can download from <https://github.com/PacktPublishing/Python-GUI-Programming-Cookbook-Third-Edition>.

1. Download the code for this chapter.
2. Open `GUI_MySQL_class.py` and look at the class methods:

```
import mysql.connector
import Ch07_Code.GuiDBConfig as guiConf
```

```
class MySQL():    # class variable GUIDB = 'GuiDB'
    #-----
    def connect(self):
        # connect by unpacking dictionary credentials
        # create cursor
    #-----
    def close(self, cursor, conn): # close cursor
    #-----
    def showDBs(self): # connect to MySQL
    #-----
    def createGuiDB(self): # connect to MySQL
    #-----
    def dropGuiDB(self): # connect to MySQL
    #-----
```



```
#-----  
def useGuiDB(self, cursor): '''Expects open connection.'''  
    # select DB  
#-----  
def createTables(self): # connect to MySQL  
    # create Table inside DB  
#-----  
def dropTables(self): # connect to MySQL  
#-----  
def showTables(self): # connect to MySQL  
#-----
```

```
def insertBooks(self,title, page, bookQuote):    # connect to MySQL
    # insert data
#-----
def insertBooksExample(self): # connect to MySQL
    # insert hard-coded data
#-----
def showBooks(self): # connect to MySQL
#-----
def showColumns(self): # connect to MySQL
#-----
def showData(self): # connect to MySQL
#-----
if __name__ == '__main__':    # Create class instance
    mySQL = MySQL()
```

3. Running the preceding code (including the full implementation of the code) creates the following tables and data in the database we created.
4. Open Command Prompt and execute the two SELECT * statements:

```
mysql> USE guidb
Database changed
mysql> SELECT * FROM books;
+-----+-----+-----+
| Book_ID | Book_Title          | Book_Page |
+-----+-----+-----+
|      1 | Design Patterns     |      7    |
|      2 | xUnit Test Patterns |     31    |
+-----+-----+-----+
2 rows in set (0.10 sec)

mysql> SELECT * FROM quotations;
+-----+-----+-----+
| Quote_ID | Quotation                                                    | Books_Book_ID |
+-----+-----+-----+
|      1 | Programming to an Interface, not an Implementation         |      1        |
|      2 | Philosophy of Test Automation                               |      2        |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

5_ Using the SQL UPDATE command

This recipe will use the code from the previous recipe, *Using the SQL INSERT command*, explain it in more detail, and then extend the code to update the data.

In order to update the data that we previously inserted into our MySQL database tables, we need to use the SQL UPDATE command.

Getting ready

This recipe builds on the previous recipe, *Using the SQL INSERT command*, so read and study the previous recipe in order to follow the code in this recipe, where we will modify the existing data.

How to do it...

Let's take a look at how we can use the SQL UPDATE command:

1. First, we will display the data to be modified by running the following Python to the MySQL command. Sequentially, we perform the following steps:

1. Open GUI_MySQL_class.py.

2. Look at the showData method:

```
import mysql.connector  
import Ch07_Code.GuiDBConfig as guiConf
```

```
class MySQL():
    # class variable GUIDB = 'GuiDB'
    #-----
    def showData(self):
        # connect to MySQL
        conn, cursor = self.connect()

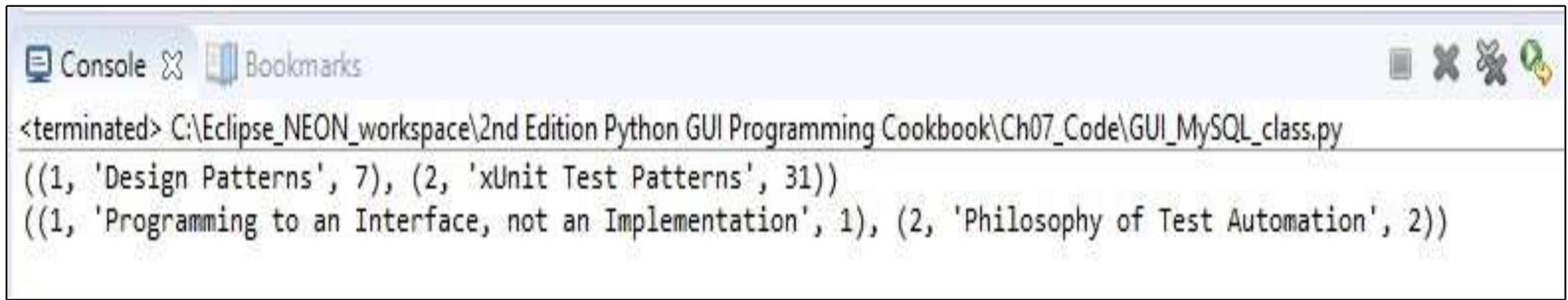
        self.useGuiDB(cursor) # execute command
        cursor.execute("SELECT * FROM books")
        print(cursor.fetchall())

        cursor.execute("SELECT * FROM quotations")
        print(cursor.fetchall())

        # close cursor and connection self.close(cursor, conn)
    #=====
```

```
#=====
if __name__ == '__main__': # Create class instance
    mySQL = MySQL()
    mySQL.showData()
```

2. Running the preceding code gives us the following output:

A screenshot of a Python IDE's console window. The window has a title bar with 'Console' and 'Bookmarks' tabs. The console output shows the execution of a Python script. The first line is a file path: <terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\GUI_MySQL_class.py. The second line is a tuple: ((1, 'Design Patterns', 7), (2, 'xUnit Test Patterns', 31)). The third line is another tuple: ((1, 'Programming to an Interface, not an Implementation', 1), (2, 'Philosophy of Test Automation', 2)).

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\GUI_MySQL_class.py
((1, 'Design Patterns', 7), (2, 'xUnit Test Patterns', 31))
((1, 'Programming to an Interface, not an Implementation', 1), (2, 'Philosophy of Test Automation', 2))
```

3.Look at the updateGOF method:

```
#-----
def updateGOF(self):
    # connect to MySQL
    conn, cursor = self.connect()
    self.useGuiDB(cursor)
    # execute command
    cursor.execute("SELECT Book_ID FROM books WHERE Book_Title =
'Design Patterns'")
    primKey = cursor.fetchall()[0][0]
    print("Primary key=" + str(primKey))
    cursor.execute("SELECT*FROM quotations WHERE Books_Book_ID =(%)",
        (primKey,))
    print(cursor.fetchall())
    # close cursor and connection
    self.close(cursor, conn)
#=====
if __name__ == '__main__':
    mySQL = MySQL()      # Create class instance mySQL.updateGOF()
```


4. Run the method located in `GUI_MySQL_class.py`:

```
# execute command
cursor.execute("SELECT Book_ID FROM books WHERE Book_Title = 'Design Patterns'")
primKey = cursor.fetchall()[0][0]
print("Primary key=" + str(primKey))

cursor.execute("SELECT * FROM quotations WHERE Books_Book_ID = (%s)", (primKey,))
print(cursor.fetchall())
```

 Console   Bookmarks

<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\GUI_MySQL_class.py

```
Primary key=1
((1, 'Programming to an Interface, not an Implementation', 1),)
```

5. Add the following code and run it:

```
#-----  
def showDataWithReturn(self): # connect to MySQL  
    conn, cursor = self.connect()  
  
    self.useGuiDB(cursor) # execute command  
    cursor.execute("SELECT Book_ID FROM books WHERE  
Book_Title = 'Design Patterns'")  
    primKey = cursor.fetchall()[0][0] print(primKey)  
  
    cursor.execute("SELECT * FROM quotations WHERE  
Books_Book_ID = (%s)", (primKey,))  
    print(cursor.fetchall())
```

```
cursor.execute("UPDATE quotations SET Quotation=(%s) WHERE  
Books_Book_ID = (%s)", ("Pythonic Duck Typing: If it walks  
like a duck and talks like a duck it probably is a duck...",  
    primKey))
```

```
# commit transaction conn.commit ()
```

```
cursor.execute("SELECT * FROM quotations WHERE  
Books_Book_ID = (%s)", (primKey,))  
print(cursor.fetchall())
```

```
# close cursor and connection self.close(cursor, conn)
```

```
#=====
```

```
#=====
if __name__ == '__main__':
    # Create class instance
    mySQL = MySQL()
    #-----
    mySQL.updateGOF()
    book, quote = mySQL.showDataWithReturn()
    print(book, quote)
```

6. Open a MySQL client window and run the `SELECT *` statements:

```
mysql> USE guidb
Database changed
mysql> SELECT * FROM books;
```

Book_ID	Book_Title	Book_Page
1	Design Patterns	7
2	xUnit Test Patterns	31

```
2 rows in set (0.10 sec)
```



```
mysql> SELECT * FROM quotations;
```

Quote_ID	Quotation	Books_Book_ID
1	Programming to an Interface, not an Implementation	1
2	Philosophy of Test Automation	2

```
mysql> SELECT * FROM books;
```

Book_ID	Book_Title	Book_Page
1	Design Patterns	7
2	xUnit Test Patterns	31

```
2 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM quotations;
```

Quote_ID	Quotation
1	Pythonic Duck Typing: If it walks like a duck and talks like a duck it probably is a duck...
2	Philosophy of Test Automation

6_ Using the SQL DELETE command

In this recipe, we will use the SQL DELETE command to delete the data we created in the previous recipe, *Using the SQL UPDATE command*.

While deleting data might sound trivial at first, once we get a rather large database design in production, things might not be that easy any more.

Because we have designed our GUI database by *relating* two tables via a *primary to foreign key relation*, when we delete certain data, we do not end up with *orphan records* because this database design takes care of *cascading* deletes.

Getting ready

This recipe uses the MySQL database, tables, and the data that was inserted into those tables from the previous recipe, *Using the SQL UPDATE command*. In order to demonstrate how to create orphan records, we will have to change the design of one of our database tables.



Changing the design to intentionally create a poor design is for demonstration purposes only and is not the recommended way of designing a database.

How to do it...

If we create our `quotations` table without a *foreign key relationship* to the `books` table, we can end up with orphan records. Take a look at the following steps:

1. Open `GUI_MySQL_class.py` and look at `def createTablesNoFK(self):`
...:

```
# create second Table inside DB --  
# No FOREIGN KEY relation to Books Table  
cursor.execute("CREATE TABLE Quotations (  
    Quote_ID INT AUTO_INCREMENT,  
    Quotation VARCHAR(250), Books_Book_ID INT,    PRIMARY  
    KEY (Quote_ID)  
) ENGINE=InnoDB")
```

2.Run the SQL command:

```
cursor.execute("DELETE FROM books WHERE Book_ID = 1")
```

3. Run the two `SELECT *` commands:

```
mysql> SELECT * FROM books;
+-----+-----+-----+
| Book_ID | Book_Title          | Book_Page |
+-----+-----+-----+
|      2 | xUnit Test Patterns |      31 |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM quotations;
+-----+-----+-----+
| Quote_ID | Quotation                                                    | Books_Book_ID |
+-----+-----+-----+
|      1 | Programming to an Interface, not an Implementation          |      1 |
|      2 | Philosophy of Test Automation                                |      2 |
+-----+-----+-----+
2 rows in set (0.00 sec)

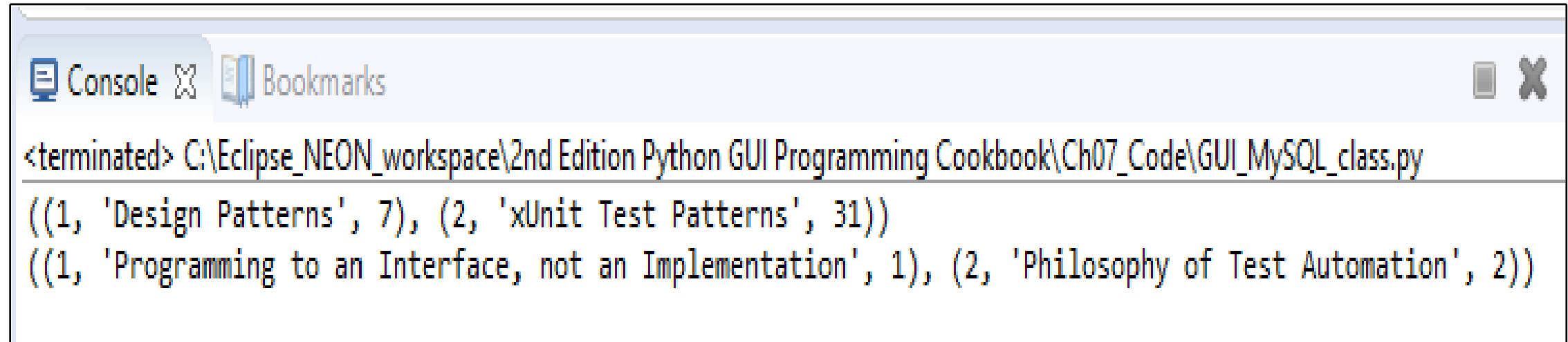
mysql>
```

4. Open GUI_MySQL_class.py and look at def createTables(self): ...:

```
# create second Table inside DB  cursor.execute("CREATE TABLE
Quotations (
    Quote_ID INT AUTO_INCREMENT,
    Quotation VARCHAR(250), Books_Book_ID INT,  PRIMARY KEY
    (Quote_ID),
    FOREIGN KEY (Books_Book_ID) REFERENCES Books(Book_ID)
    ON DELETE CASCADE
) ENGINE=InnoDB")

#=====
if __name__ == '__main__': # Create class instance mySQL =
    mySQL()
    mySQL.showData()
```

5. Run the `showData()` method:



The screenshot shows a console window with a tab labeled 'Console' and a 'Bookmarks' icon. The text in the console is as follows:

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\GUI_MySQL_class.py  
((1, 'Design Patterns', 7), (2, 'xUnit Test Patterns', 31))  
((1, 'Programming to an Interface, not an Implementation', 1), (2, 'Philosophy of Test Automation', 2))
```

6. Run the `deleteRecord()` method, followed by the `showData()` method:

```
import mysql.connector
import Ch07_Code.GuiDBConfig as guiConf

class MySQL():
    #-----
    def deleteRecord(self): # connect to MySQL
        conn, cursor = self.connect()

        self.useGuiDB(cursor) # execute command
        cursor.execute("SELECT Book_ID FROM books WHERE Book_Title
=
        'Design Patterns'")
        primKey = cursor.fetchall()[0][0]
        # print(primKey)
```

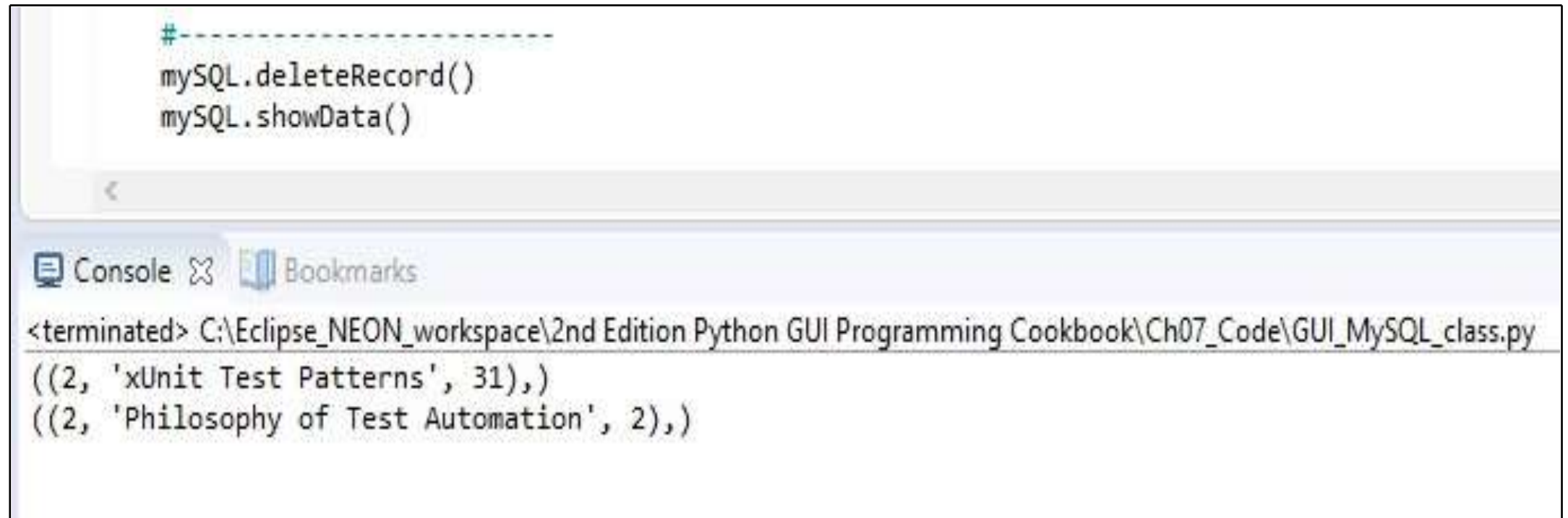
```
cursor.execute("DELETE FROM books WHERE Book_ID =(%s)",
               (primKey,))

# commit transaction conn.commit ()

# close cursor and connection self.close(cursor, conn)

#=====
if __name__ == '__main__': # Create class instance
    mySQL = MySQL()
    #-----
    mySQL.deleteRecord() mySQL.showData()
```

7. The preceding code results in the following output:



The screenshot shows an IDE window with a code editor and a console. The code editor contains the following Python code:

```
#-----  
mysql.deleteRecord()  
mysql.showData()
```

The console output shows the execution of the code, including a terminated message and the resulting data:

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\GUI_MySQL_class.py  
((2, 'xUnit Test Patterns', 31),)  
((2, 'Philosophy of Test Automation', 2),)
```


7_ Storing and retrieving data from our MySQL database

We will use our Python GUI to insert data into our MySQL database tables. We already refactored the GUI we built in the previous recipes in preparation for connecting and using a database.

We will use two textbox Entry widgets, into which we can type the book or journal title and the page number. We will also use a ScrolledText widget to type our favorite book quotations into, which we will then store in our MySQL database.

Getting ready

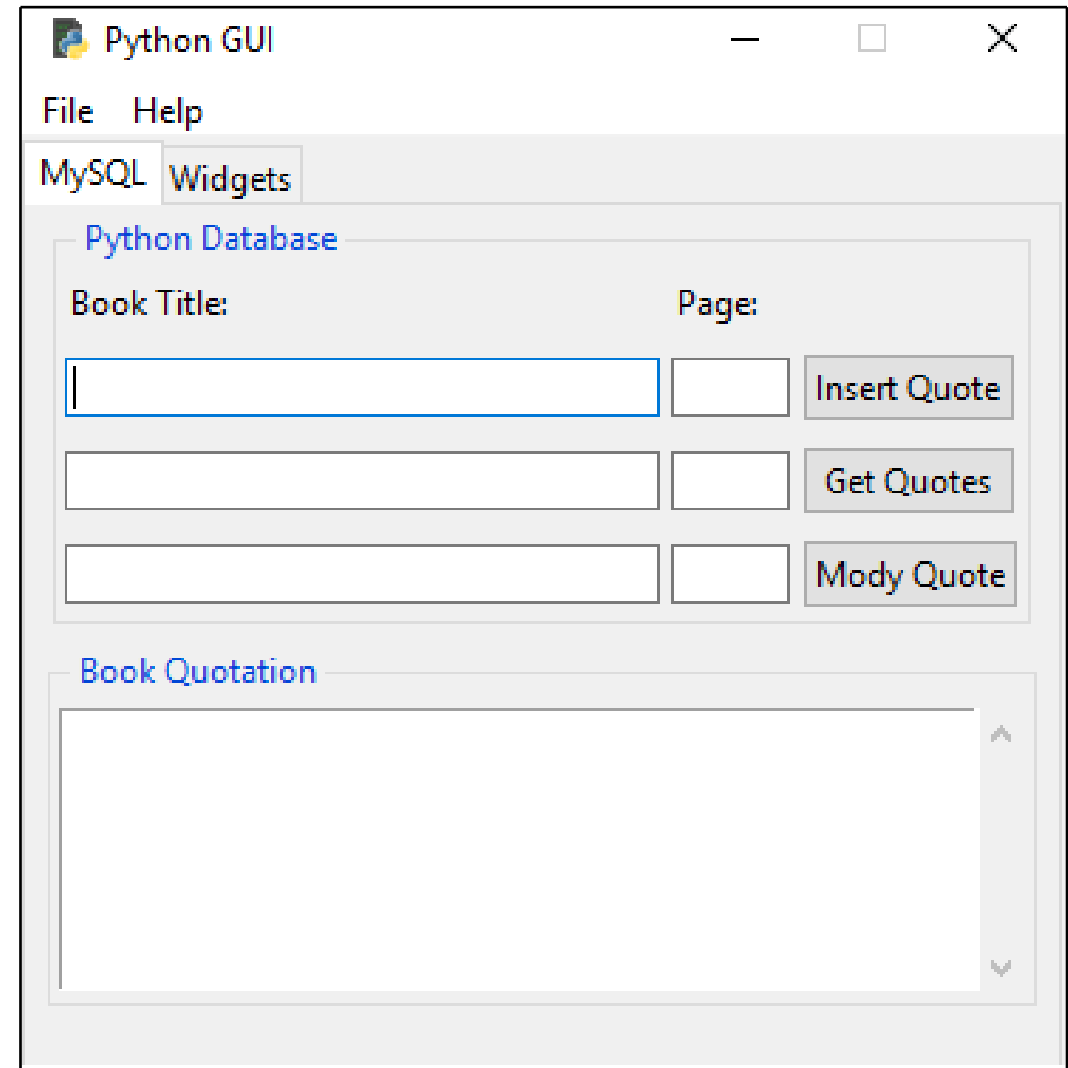
This recipe will build on the MySQL database and tables we created in the previous recipes of this chapter.

How to do it...

We will insert, retrieve, and modify our favorite quotations using our Python GUI. We refactored the MySQL tab of our GUI in preparation for this. Let's look at how we can deal with this:

1. Open GUI_MySQL.py.
2. Running the code in this file shows us our GUI:

2. Running the code in this file shows us our GUI:



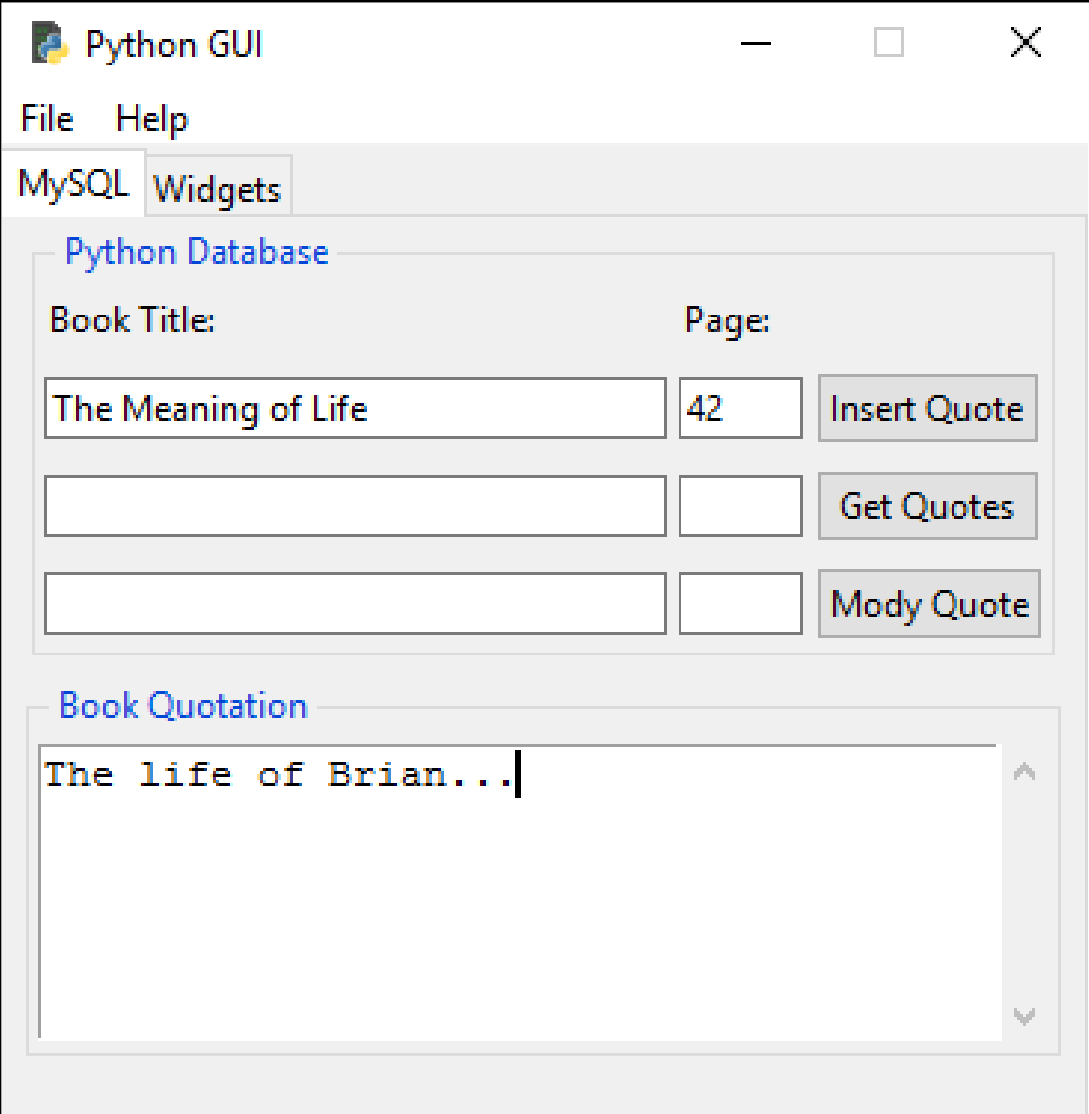
3. Open `GUI_MySQL.py`.

4. Notice the `insertQuote()` method, as shown here:

```
# Adding a Button
self.action = ttk.Button(self.mySQL, text="Insert Quote",
                           command=self.insertQuote)
self.action.grid(column=2, row=1)

# Button callback
def insertQuote(self):
    title = self.bookTitle.get()
    page = self.pageNumber.get()
    quote = self.quote.get(1.0, tk.END)
    print(title)
    print(quote)
    self.mySQL.insertBooks(title, page, quote)
```

5. Run `GUI_MySQL.py`,
enter a quotation, and click
the **Insert Quote** button:



The screenshot shows a window titled "Python GUI" with a menu bar containing "File" and "Help". Below the menu bar are two tabs: "MySQL" (selected) and "Widgets". The main content area is divided into two sections:

- Python Database**: This section contains two input fields labeled "Book Title:" and "Page:". The "Book Title:" field contains the text "The Meaning of Life", and the "Page:" field contains the number "42". To the right of these fields are three buttons: "Insert Quote", "Get Quotes", and "Mody Quote".
- Book Quotation**: This section contains a large text area with the text "The life of Brian..." and a vertical scrollbar on the right.

6. Click Get Quotes:

Python GUI

File Help

MySQL Widgets

Python Database

Book Title: Page:

The Meaning of Life 42 Insert Quote

Get Quotes

Mody Quote

Book Quotation

The Life of Brian...{1 {The Meaning of Life} 42}

7. Open `GUI_MySQL.py` and look at the `getQuote` method and button:

```
# Adding a Button
self.action1 = ttk.Button(self.mySQL, text="Get Quotes",
                           command=self.getQuote)
self.action1.grid(column=2, row=2)

# Button callback def getQuote(self):
allBooks = self.mySQL.showBooks()
print(allBooks)
self.quote.insert(tk.INSERT, allBooks)
```

8. Open `GUI_MySQL.py` and look at `self.mysql` and `showBooks()`:

```
from ch07_Code.GUI_MySQL_class import MySQL

class OOP():
    def __init__(self):
        # create MySQL instance
        self.mysql = MySQL()
```



```
class MySQL():
```

```
    #-----
```

```
    def showBooks(self):
```

```
        # connect to MySQL
```

```
        conn, cursor = self.connect()
```

```
        self.useGuiDB(cursor)
```

```
        # print results
```

```
        cursor.execute("SELECT * FROM Books")
```

```
        allBooks = cursor.fetchall()
```

```
        print(allBooks)
```

```
        # close cursor and connection
```

```
        self.close(cursor, conn)
```

```
        return allBooks
```

8_ Using MySQL Workbench

MySQL has a very nice GUI that we can download for free. It's called **MySQL Workbench**.

In this recipe, we will successfully install Workbench and then use it to run SQL queries against the GuiDB we created in the previous recipes.

Getting ready

In order to use this recipe, you will need MySQL database we developed in the previous recipes. You will also need a running MySQL server.

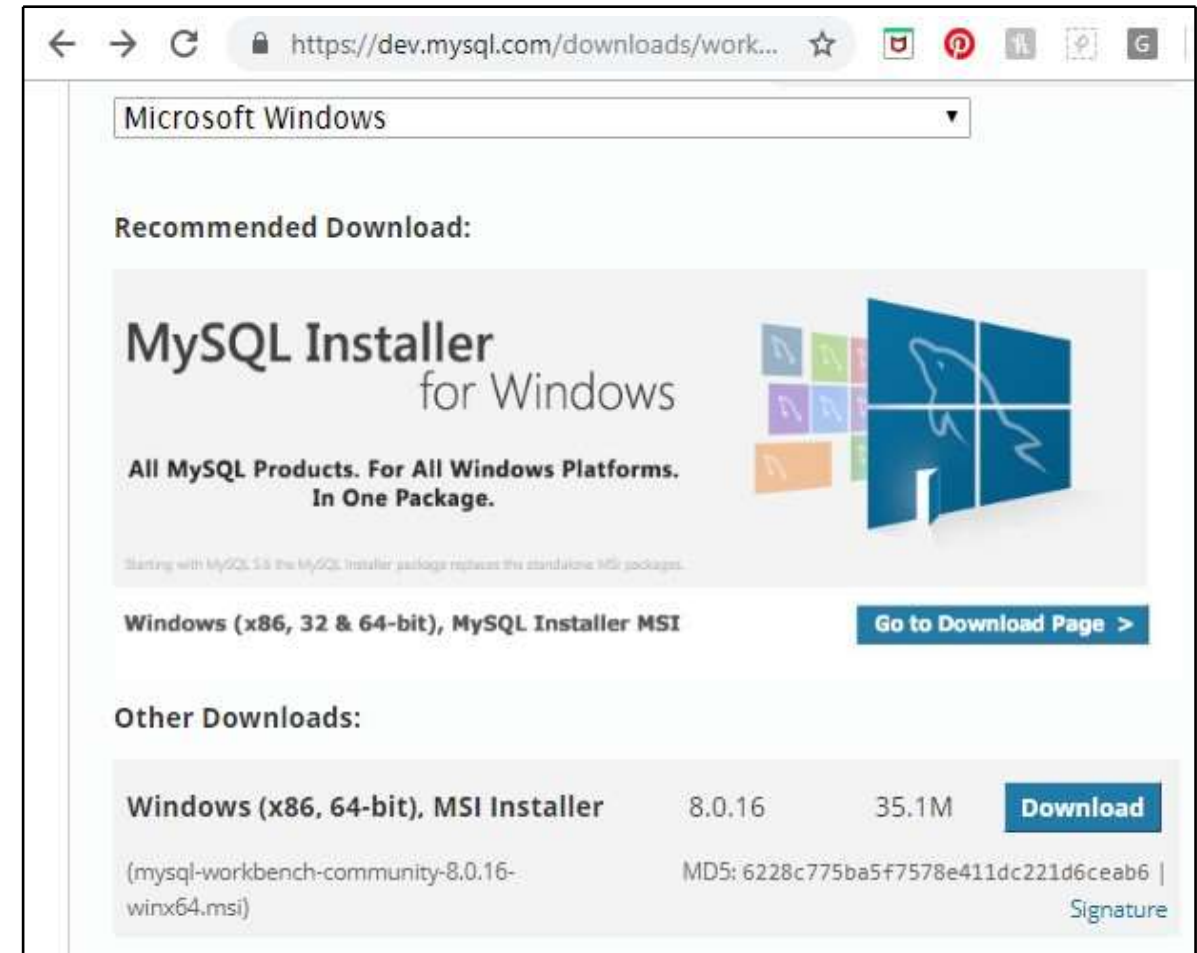
How to do it...

We can download MySQL Workbench from the official MySQL website:

<https://dev.mysql.com/downloads/workbench/>.

Let's look at how we can perform this recipe:

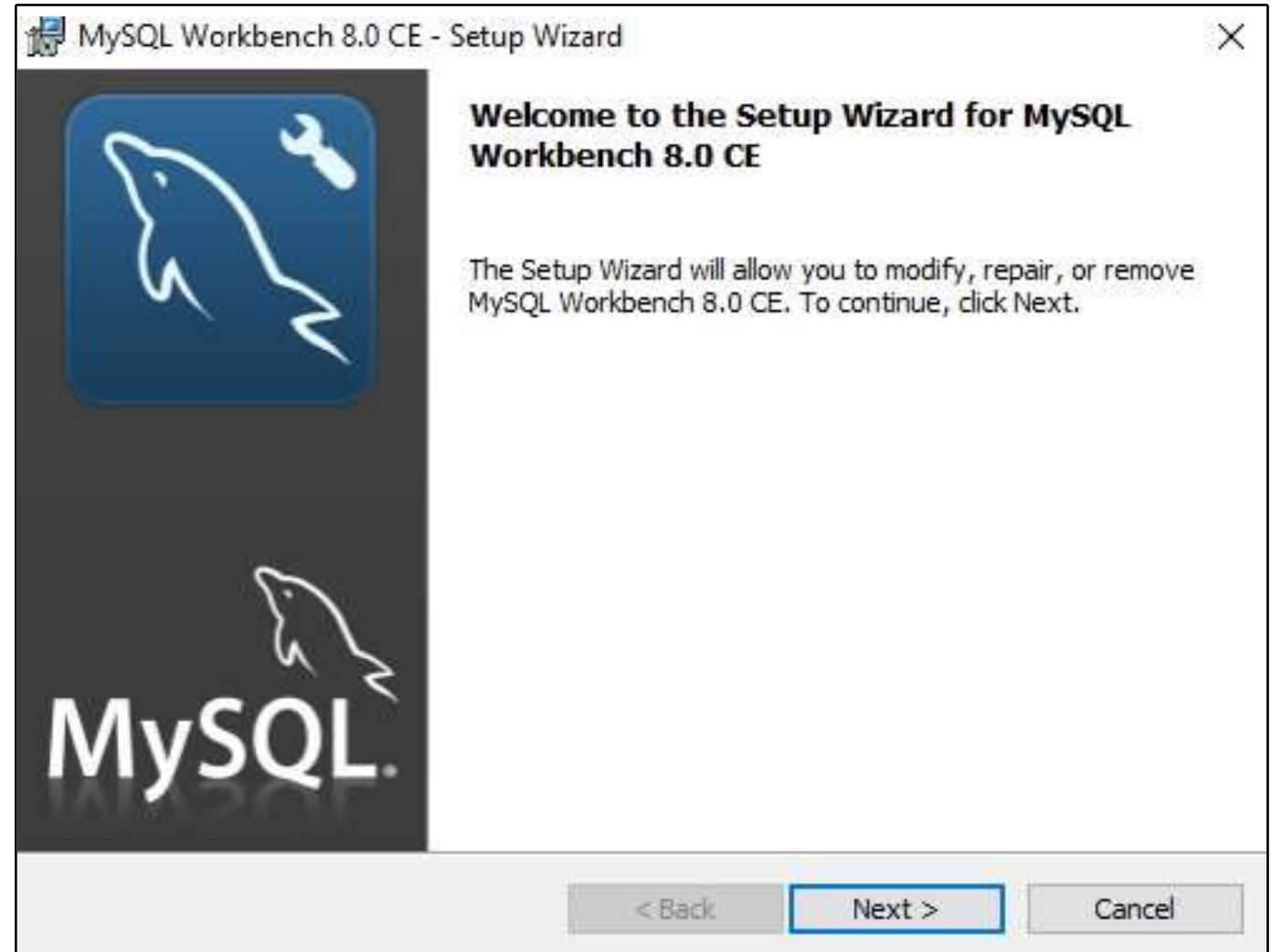
1. Download the MySQL Workbench installer.
2. Click the **Download** button:



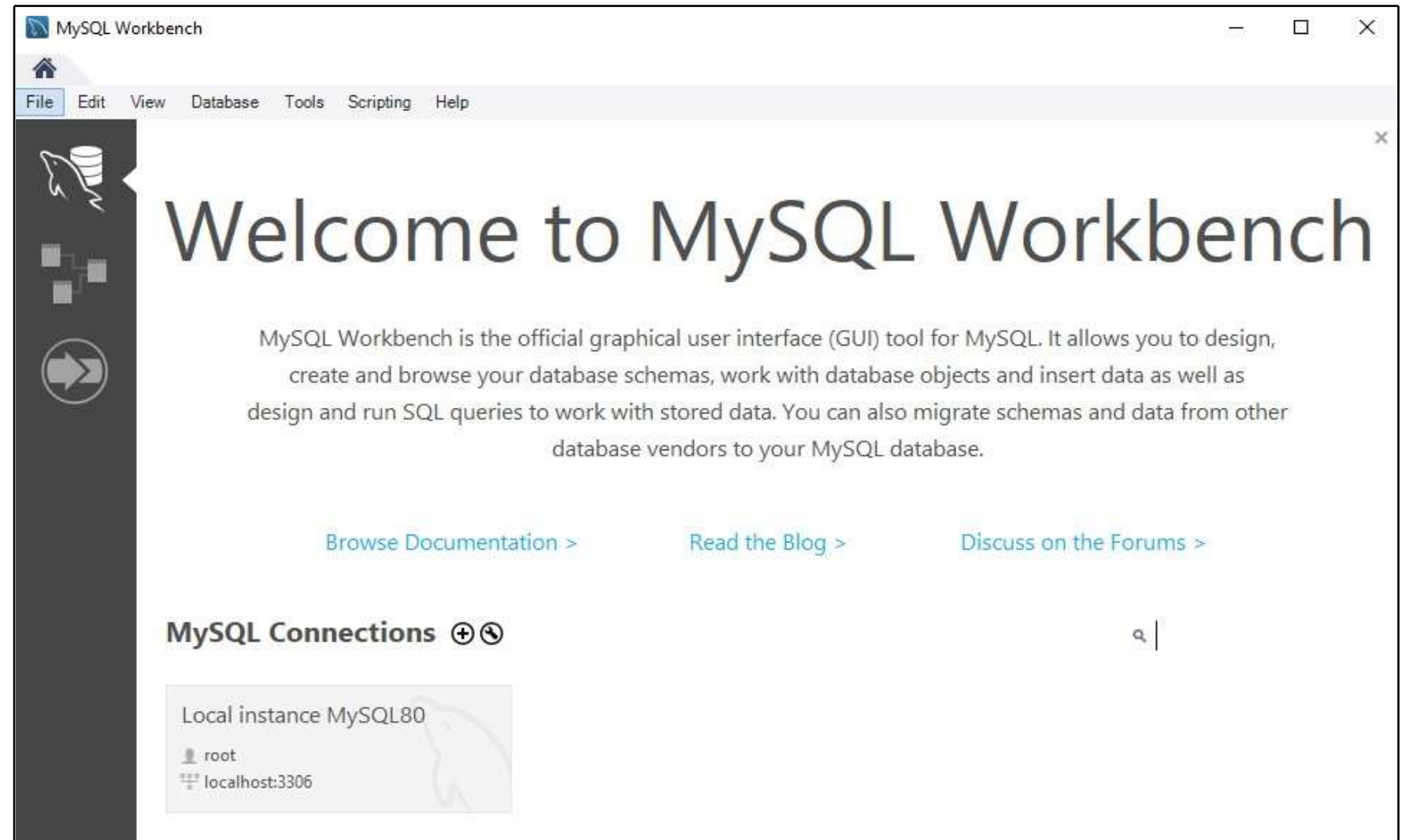
3. Run the installation:



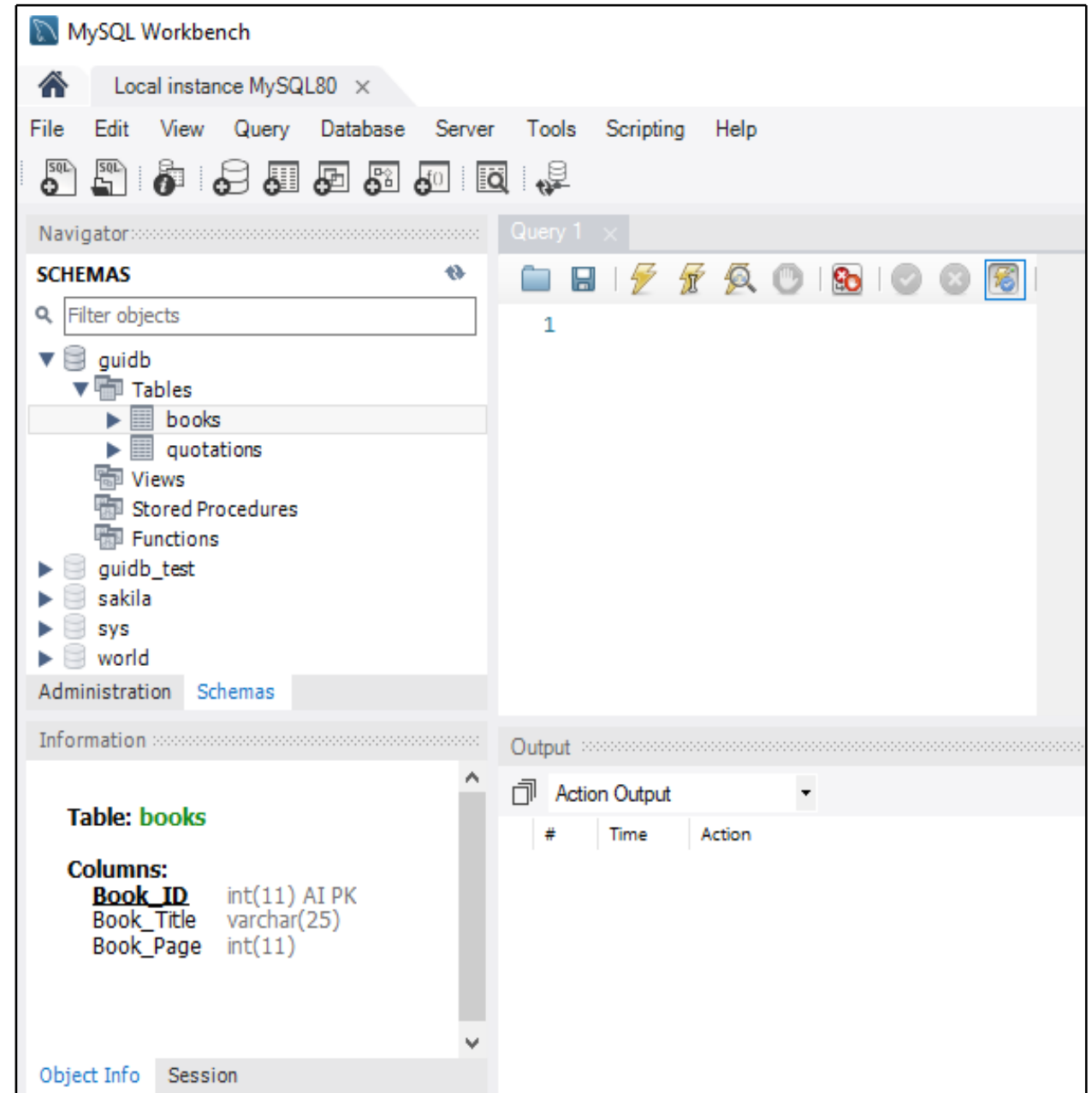
4. Click **Next >** until the installation is complete:



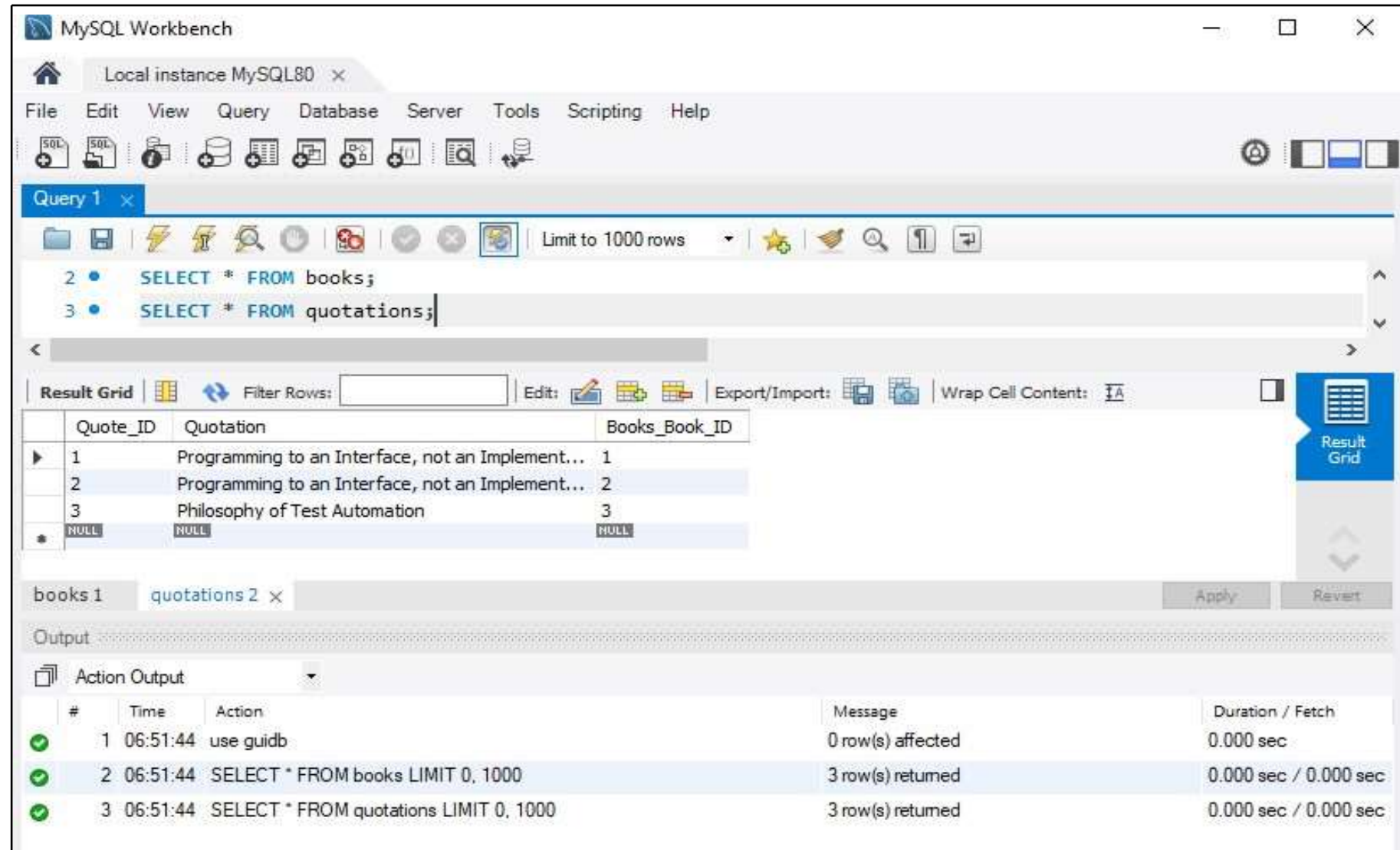
5. Open MySQL Workbench:



6. Select our guidb:



7. Write and execute some SQL commands:



The screenshot displays the MySQL Workbench interface. The top menu bar includes File, Edit, View, Query, Database, Server, Tools, Scripting, and Help. The toolbar contains various icons for file operations, query execution, and navigation. The main query editor shows two SQL queries:

```
2 • SELECT * FROM books;  
3 • SELECT * FROM quotations;
```

The 'Result Grid' tab is active, showing the results of the queries. The first query, 'SELECT * FROM books', returned 3 rows. The second query, 'SELECT * FROM quotations', returned 3 rows. The results are displayed in a table with columns: Quote_ID, Quotation, and Books_Book_ID.

Quote_ID	Quotation	Books_Book_ID
1	Programming to an Interface, not an Implement...	1
2	Programming to an Interface, not an Implement...	2
3	Philosophy of Test Automation	3
NULL	NULL	NULL

The bottom section of the interface shows the 'Output' tab, which displays the 'Action Output' log. The log contains three entries, each with a status icon, a sequence number, a timestamp, an action description, a message, and a duration/fetch time.

#	Time	Action	Message	Duration / Fetch
✓ 1	06:51:44	use guidb	0 row(s) affected	0.000 sec
✓ 2	06:51:44	SELECT * FROM books LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
✓ 3	06:51:44	SELECT * FROM quotations LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec



We can type SQL commands into the **Query Editor** and execute our commands by clicking the lightning bolt icon. It is the button toward the top right, as shown in the following screenshot:

