Flask Python (5) - Data

Data

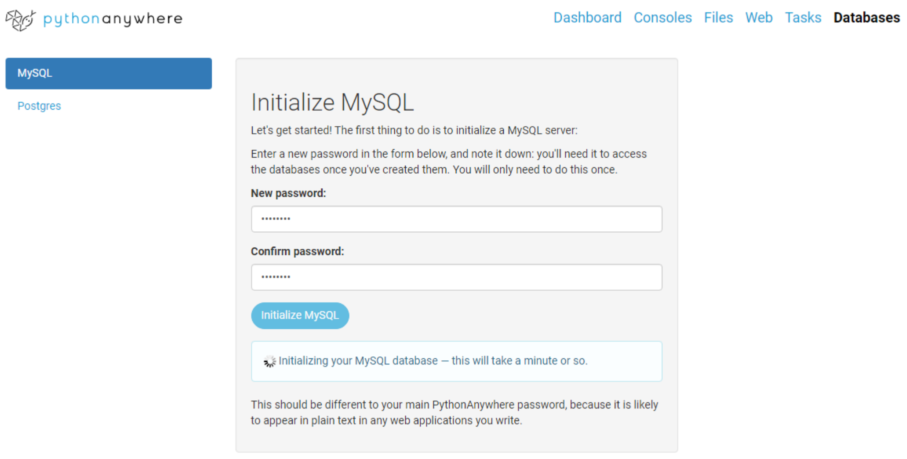
In the previous section we looked at collecting data. We now need to store that user data somewhere or it will disappear. To do this you will need some database knowledge. So we will pause our development and get stuck into database.

You may want to watch this database playlist before you start so you get a feel. This uses Microsoft Access but the fundamentals of databases are nice and clear! The first 8 videos take about 20 minutes. You don't need to go into the Firebase videos (we may get onto that later!)

https://www.youtube.com/watch?v=7Ye_-oYJ1Mc&list=PLqJKbfNydBaxYYaxlKZiLx94xirY_kv7P

MySQL Setup

On the pythonanywhere home page click on **Databases** and put in a password. You will need to remember this password! You have time because it takes minute or so to sort the database out.



Initialize MySQL

Let's get started! The first thing to do is to initialize a MySQL server:

Enter a new password in the form below, and note it down: you'll need it to access the databases once you've created them. You will only need to do this once.

New password: ••••••••

Confirm password: ••••••••

Initializing your MySQL database — this will take a minute or so.

This should be different to your main PythonAnywhere password, because it is likely to appear in plain text in any web applications you write.

You can now create a new database and will call ours library. This will be a library of books and some of them will be about Shackleton. But, only our users will be able to see them listed (eventually!)

That's it a database is all set up and it's time to connect.

Connection

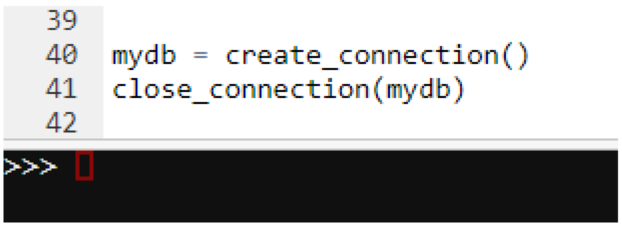
We want to keep our database stuff separate so create a new directory on **mysite** called **library** and in there create a new python file called *models.py* (models is short for data-modelling). This file will hold our functions which mean we can connect to the database when we want!

The following code imports the MySQL connection library and define two functions to open and close our database.

```python
import mysql.connector

def create_connection():
    db = mysql.connector.connect(
        host = "storeyacdulwich.mysql.pythonanywhere-services.com",
        user="storeyacdulwich",
        password="xxxxxx",
        database = "storeyacdulwich$library")
    return db

def close_connection(db):
    db.close()
```

The connection requires you to have to hand your username, database name and password for the database. These can be found in the database link on the pythonanywhere home page.

You can check this works by running the functions at the bottom of the *models.py* file, saving the file and running it.



```python

mydb = create_connection()
close_connection(mydb)

>>> 
```

As you can see, the connection opens and then closes it; the cursor shows me there are no errors!

Creating a Table

We now need to create a table to hold information about our books. Here is the table we are going to create in MySQL and set ISBN as the key field (primary key)

Field Name	Data Type	Description
Title	Text	Title of Book
Author	Text	The surname of the author
ISBN	Text	The ISBN number

Here we define a new function `create_book_table()` and this will create our table to hold the books.

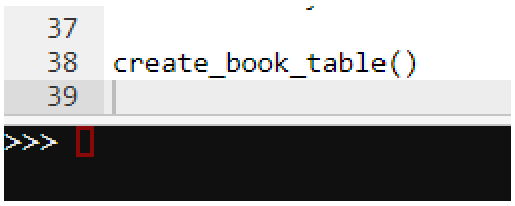
```python
def create_book_table():
    mydb = create_connection()
    mycursor = mydb.cursor()
    sql_txt = "CREATE TABLE T_books (ISBN VARCHAR(25) NOT NULL PRIMARY KEY, title VARCHAR(55), author VARCHAR(55))"
    mycursor.execute(sql_txt)
    close_connection(mydb)
```

Just looking more closely at the SQL text:

```
CREATE TABLE T_books
(
        isbn VARCHAR(25) NOT NULL PRIMARY KEY,
        title VARCHAR(55),
        author VARCHAR(55)
)
```

You will see we have defined the name of the table *T_books*. There are 3 fields *isbn*, *title* and author and *isbn* should not be empty and is a primary key (no other book will have that ISBN number). You can make the other fields not null as well if you like.

Remove the database connection code at the bottom of *models.py* and instead you can now run the function `create_table_books()` and as you can see below there are no errors.



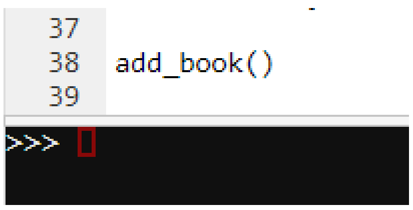
```
create_book_table()

>>>
```

Adding Data to tables

Let's add some data to our table. We can do this with a new function as follows:

```
def add_book():
    mydb = create_connection()
    mycursor = mydb.cursor()
    sql_txt = "INSERT INTO books (ISBN,title,author) VALUES ('9780140861495','A Tale of two cities','Charles Dickins')"
    mycursor.execute(sql_txt)
    mydb.commit()
    close_connection(mydb)
```

This code adds the books "A Tale of Two Cities" to our database.



```
add_book()

>>>
```

The syntax of the INSERT query in SQL is:

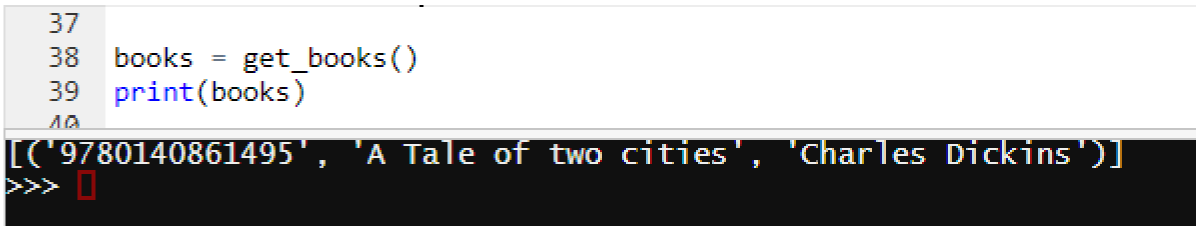
```
INSERT INTO <tablename> (field1,field2, ..) VALUES (val1, val2 ..)
```

Retrieving Data

We can now retrieve our data as follows:

```
def get_books():
    mydb = create_connection()
    mycursor = mydb.cursor()
    sql_txt = "SELECT * FROM T_books"
    mycursor.execute(sql_txt)
    myresult = mycursor.fetchall()
    close_connection(mydb)
    return myresult
```

This function connects to the database and retrieves all the records in the *T_Books* table. It returns a list of tuples which contain the ibsn, title and author of the book. Again, we can test this by running it and printing the results in our *models.py* file:



```

books = get_books()
print(books)
```

```
[('9780140861495', 'A Tale of two cities', 'Charles Dickins')]
>>> 
```

Not the result is in the form `[(a,b,c)]` which is just one tuple in a list.

Adding more books

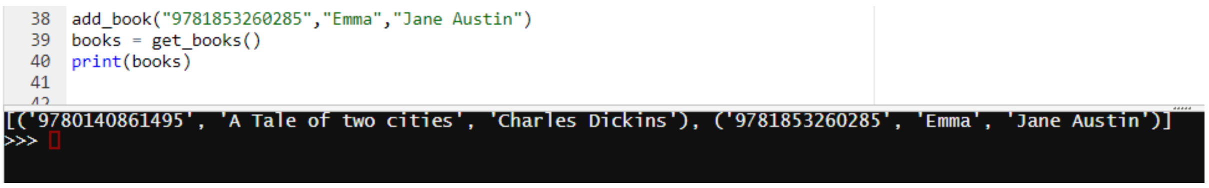
Our add-book() is pretty useless at the moment because we have "hard-coded" the values of into the SQL. We wnmat to run a function like this:

```
add_book("9781853260285","Emma","Jane Austin")
```

To do this we must parameterise our add-book function as follows:

```
def add_book(isbn,title, author):
    mydb = create_connection()
    mycursor = mydb.cursor()
    sql_txt = "INSERT INTO T_books (ISBN,title,author) VALUES ('" + isbn + "','" + title + "','" + author + "')"
    mycursor.execute(sql_txt)
    mydb.commit()
    close_connection(mydb)
```

Now we can add different books using this function. Here's the result when I add a new book and return the list of books in the database.



```
add_book("9781853260285","Emma","Jane Austin")
books = get_books()
print(books)

```

```
[('9780140861495', 'A Tale of two cities', 'Charles Dickins'), ('9781853260285', 'Emma', 'Jane Austin')]
>>> 
```

Our book has been added and we have a new tuple in our database results.