Flask SQLite database

Python has built-in support for SQLite. The SQlite3 module comes with the Python release. In this article you will learn ho
w the Flask application interacts with SQLite.

SQLite is a relational database system that uses the SQL query language to interact with the database. Each database can have tables and each table can have records.

Related course: Python Flask: Create Web Apps with Flask

Create database and table

The SQLite database storse all data in a single file. You can create an SQLite database from Python code. The program creates a SQLite database 'database.db ' where the student tables are created.

```
import sqlite3

conn = sqlite3.connect('database.db')
print "Opened database successfully";

conn.execute('CREATE TABLE students (name TEXT, addr TEXT, city TEXT
print "Table created successfully";
conn.close()
```

Views

Our Flask application has three View functions.

Submit form

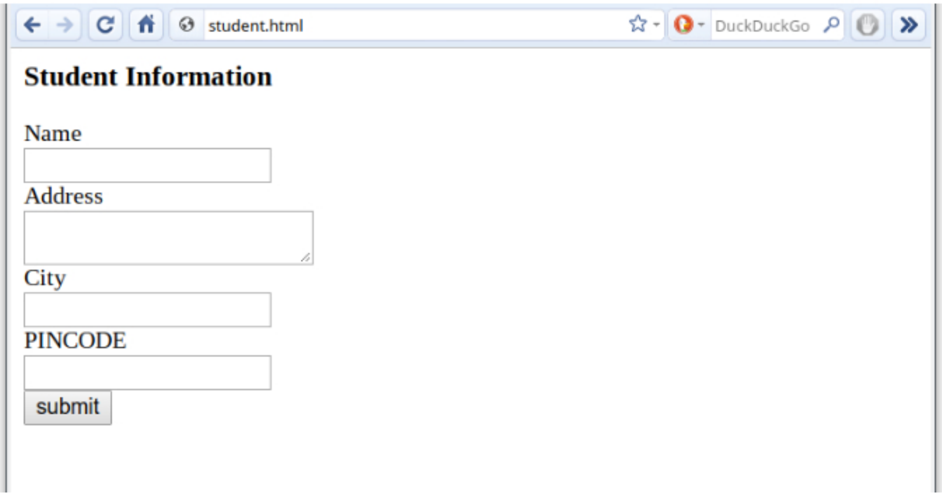
The first `new_student()` function is bound to a URL rule (`'/enternew'`).It presents an HTML file that contains a student information form.

.

```
@app.route('/enternew')
def new_student():
   return render_template('student.html')
```

The file student.html:

```
<html>
   <body>

      <form action = "{{ url_for('addrec') }}" method = "POST">
         <h3>Student Information</h3>
         Name<br>
         <input type = "text" name = "nm" /></br>

         Address<br>
         <textarea name = "add" ></textarea><br>

         City<br>
         <input type = "text" name = "city" /><br>

         PINCODE<br>
         <input type = "text" name = "pin" /><br>
         <input type = "submit" value = "submit" /><br>
      </form>

   </body>
</html>
```



Student Information

Name

Address

City

PINCODE

Add record

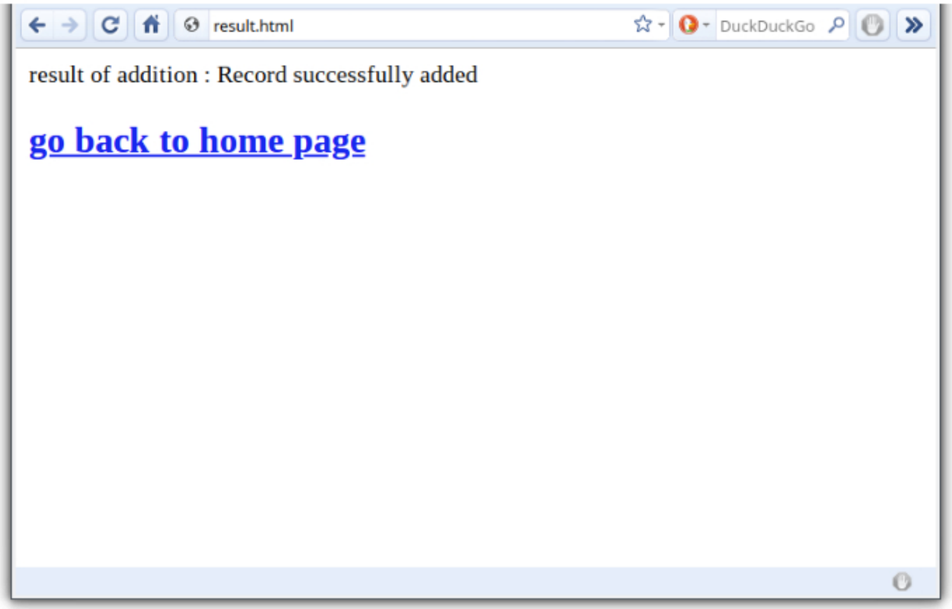
As can be seen, the form data is published to the '/addrec' URL of the binding addrec () function.

The addrec () function retrieves the form's data through the POST method and inserts the student table.The message corresponding to the success or error in the insert operation will be rendered as 'result.html'.

```
@app.route('/addrec',methods = ['POST', 'GET'])
def addrec():
   if request.method == 'POST':
      try:
         nm = request.form['nm']
         addr = request.form['add']
         city = request.form['city']
         pin = request.form['pin']

         with sql.connect("database.db") as con:
            cur = con.cursor()
            cur.execute("INSERT INTO students (name,addr,city,pin)
               VALUES (?,?,?,?)",(nm,addr,city,pin) )

            con.commit()
            msg = "Record successfully added"
      except:
         con.rollback()
         msg = "error in insert operation"

      finally:
         return render_template("result.html",msg = msg)
         con.close()
```

The HTML script for result.html contains an escape statement , which displays the result of the Insert operation.

```
<!doctype html>
<html>
   <body>
   <body>

      result of addition : {{ msg }}
      <h2><a href = "\">go back to home page</a></h2>

   </body>
</html>
```



result of addition : Record successfully added

go back to home page

List items

The application contains another list () function represented by the ‘/list’ URL.It populates’rows’ as a Multidict object that contains all records in the student table.This object is passed to the list.html template.

```
@app.route('/list')
def list():
   con = sql.connect("database.db")
   con.row_factory = sql.Row

   cur = con.cursor()
   cur.execute("select * from students")

   rows = cur.fetchall();
   return render_template("list.html",rows = rows)
```

The file list.html contains:

```
<!doctype html>
<html>
   <body>

      <table border = 1>
         <thead>
            <td>Name</td>
            <td>Address>/td<
            <td>city</td>
            <td>Pincode</td>
         </thead>

         {% for row in rows %}
            <tr>
               <td>{{row["name"]}}</td>
               <td>{{row["addr"]}}</td>
               <td> {{ row["city"]}}</td>
               <td>{{row['pin']}}</td>
            </tr>
         {% endfor %}
      </table>

      <a href = "/">Go back to home page</a>

   </body>
</html>
```

Finally, the '/' URL rule renders 'home.html', which is the entry point of the application.

```
@app.route('/')
def home():
   return render_template('home.html')
```

SQLite Example

The following is the complete code for the Flask-SQLite application.

```
from flask import Flask, render_template, request
import sqlite3 as sql
app = Flask(__name__)
```

```python
app = Flask(__name__)

@app.route('/')
def home():
   return render_template('home.html')

@app.route('/enternew')
def new_student():
   return render_template('student.html')

@app.route('/addrec',methods = ['POST', 'GET'])
def addrec():
   if request.method == 'POST':
      try:
         nm = request.form['nm']
         addr = request.form['add']
         city = request.form['city']
         pin = request.form['pin']

         with sql.connect("database.db") as con:
            cur = con.cursor()

            cur.execute("INSERT INTO students (name,addr,city,pin)
               VALUES (?,?,?,?)",(nm,addr,city,pin) )

            con.commit()
            msg = "Record successfully added"
      except:
         con.rollback()
         msg = "error in insert operation"

      finally:
         return render_template("result.html",msg = msg)
         con.close()

@app.route('/list')
def list():
   con = sql.connect("database.db")
   con.row_factory = sql.Row

   cur = con.cursor()
   cur.execute("select * from students")

   rows = cur.fetchall();
   return render_template("list.html",rows = rows)

if __name__ == '__main__':
   app.run(debug = True)
```

Related course: Python Flask: Create Web Apps with Flask

Back

Send an E-mail with Python Flask

Next

Flask SQLAlchemy (with Examples)