

TePSIT



## Costruire un sito web con Flask+SQLAlchemy

Giulio Angiani  
I.I.S. "Blaise Pascal" - Reggio Emilia



# Web app con Flask+SQLAlchemy

seconda parte



# Flask e database

Sempre in **virtual environment** installiamo l'interfaccia python-mysql : package **mysql-connector-python**



```
$ pip install mysql-connector-python

Collecting mysql-connector-python
  Downloading mysql_connector_python-8.0.23-cp38-cp38-manylinux1_x86_64.whl (18.0 MB)
    |████████| 18.0 MB 100 kB/s
Collecting protobuf>=3.0.0
  Downloading protobuf-3.14.0-cp38-cp38-manylinux1_x86_64.whl (1.0 MB)
    |████████| 1.0 MB 2.0 MB/s
Collecting six>=1.9
  Using cached six-1.15.0-py2.py3-none-any.whl (10 kB)
Installing collected packages: six, protobuf, mysql-connector-python
Successfully installed mysql-connector-python-8.0.23 protobuf-3.14.0 six-1.15.0
```

```
$ python

Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import mysql.connector
>>> cnx = mysql.connector.connect(user='scuola', password='scuola',
   host='localhost', database='scuola',
   auth_plugin='mysql_native_password')
>>> >>> cnx
<mysql.connector.connection.MySQLConnection object at 0x7efeb0fb2730>
```



# Eseguire una query con mysql-connector

PYTHON

```
# creiamo la connessione
>>> cnx = mysql.connector.connect(user='scuola', password='scuola',
                                   host='localhost', database='scuola',
                                   auth_plugin='mysql_native_password')
>>> cursor = cnx.cursor(dictionary=True)      # creiamo il cursore al resultset in modalità dictionary
>>> results = cursor.execute("SELECT * FROM studenti") # eseguo la query
>>> rows = cursor.fetchall() # recupero tutte le righe
>>> import pprint
>>> pprint.pprint(rows[0])
{'cognome': 'ROSSI',
 'datanascita': None,
 'fk_classe_id': 53,
 'id': 128,
 'matricola': '004407',
 'nome': 'FRANCESCA',
 'sesso': 'F'}
>>> cursor.close() # chiudo il cursore
>>> cnx.close() # chiudo la connessione
```



# Esercizi - 3

Sia data la tabella **nazioni** in un database

```
mysql> select * from nazioni;
+----+-----+-----+
| id | nazione | codice |
+----+-----+-----+
| 1  | Italia   | IT    |
| 2  | Francia  | FR    |
| 3  | Regno Unito | UK   |
+----+-----+-----+
```



Si scriva un metodo mappato sulla route **/nazioni** che legga la lista delle nazioni dal database e la rappresenti in tabella HTML

Utente connesso: Marco		
Lista delle nazioni presenti		
ID	Codice	Nazione
1	IT	Italia
2	FR	Francia
3	UK	Regno Unito

10 minuti



# Esercizi - 4

Sia data la tabella **classi** in un database

```
mysql> select * from classi limit 3;
+----+-----+-----+-----+
| id | classe | sezione | indirizzo |
+----+-----+-----+-----+
| 31 |      5 |      B |     INF |
| 51 |      1 |      C |      SA |
| 52 |      1 |      B |     INF |
+----+-----+-----+-----+
```



Si scriva un metodo mappato sulla route **/classi** che legga la lista delle classi dal database e la rappresenti in tabella HTML  
Aggiungere un link del tipo **/classi/<ID>** in ogni riga per visualizzare la lista degli studenti

Utente connesso: Marco			
Lista delle classi presenti			
Classe	Sezione	Indirizzo	
1	B	INF	<a href="#">lista degli studenti</a>
1	C	INF	<a href="#">lista degli studenti</a>
1	C	SA	<a href="#">lista degli studenti</a>
1	D	GR	<a href="#">lista degli studenti</a>
1	E	GR	<a href="#">lista degli studenti</a>

10 minuti



# Passaggio dei parametri con le routes

Le **routes** possono essere **parametrizzate** scrivendole in questo modo

```
@app.route('/classi/<idclasse>')
```

PYTHON

Questa route sarà agganciata quando l'URL avrà un formato compatibile  
es:

- /classi/1
- /classi/100

ma anche

- /classi/qualsiasi

Possiamo gestire il tipo del dato in input nella route

```
@app.route('/classi/<int:idclasse>') # il parametro della route deve essere utilizzato come
```

PYTHON



# Recupero dei parametri con le routes (GET)

I parametri della route **devono** essere in input al metodo con lo stesso nome

```
@app.route('/classi/<int:idclasse>') # il parametro della route deve essere utilizzato come  
def classe_singola(idclasse): # parametro in ingresso al metodo  
    # posso usare il parametro per fare la query giusta...
```

PYTHON

## ESERCIZIO

Completare il metodo **classe\_singola** mappato alla route **/classi/<int:idclasse>** per ottenere quanto in figura sapendo che esiste la tabella studenti così fatta  
Aggiungere anche un **template** corretto.

Mappare il link **scheda** alla route **/studente/<int:idstudente>**

```
mysql> select * from studenti limit 3;  
+----+-----+-----+-----+-----+-----+  
| id | matricola | cognome | nome | datanascita | sesso | fk_classe_id |  
+----+-----+-----+-----+-----+-----+  
| 128 | 004407 | MENOZZINO | GIOVANNI | 0000-00-00 | F | 53 |  
| 129 | 004408 | SESSI | VALERIA | 0000-00-00 | F | 53 |  
| 130 | 004409 | VERONI | ALEXIA | 0000-00-00 | F | 53 |  
+----+-----+-----+-----+-----+-----+
```

MYSQL

Lista studenti			
Classe 2 C INF			
Matricola	Nome	Cognome	
004562	ANGOTTI	MATTEO	<a href="#">scheda</a>
004563	BECHI	SIMONE	<a href="#">scheda</a>
004564	BELVEDERE	LORENZO	<a href="#">scheda</a>
004565	BRAGLIA	TOMMASO	<a href="#">scheda</a>

15 minuti



# Soluzione

Nella parte **controller** ci occupiamo solo di recuperare l'**id** dello studente selezionato e della query al database

Il resto è delegato al template dove è presente la form per la modifica dei dati

```
@app.route('/studente/<int:idstudente>')
def studente_form(idstudente):
    studente = make_query("select * from studenti where id = '{}' ".format(idstudente))[0]
    return render_template("schedastudente.html", **vars())
```

PYTHON

```
<h3>Scheda dello studente {{ studente.cognome }} {{ studente.nome }} </h3>
<form name='frm_studente' method='POST' action='/updatestudente'>
    <input type='hidden' name='id' value='{{ studente.id }}'>
    <table class='table table-striped w-25 m-auto'>
        <tr>    <td class='align-right'>Matricola</td>
                <td class='left'><input type='text' name='matricola' value='{{ studente.matricola }}'></td> </tr>
        <tr>    <td class='align-right'>Cognome</td>
                <td class='left'><input type='text' name='cognome' value='{{ studente.cognome }}'></td> </tr>
        <tr> ... </tr>
        <tr>    <td class='align-right'><input type='submit' name='submit' value='Modifica'></td>
                <td class='left'></td> </tr>
    </table>
</form>
```

JINJA TEMPLATE

# Soluzione

Interessante la modalità per gestire i campi **select** e **checkbox**

Nel caso specifico il campo **sesso** che viene realizzato con una tendina a due opzioni

```
<tr>
  <td class='align-right'>Sex</td>
  <td class='left'>
    <select name='sesso'>
      <option value='M'
        {%- if studente.sesso == 'M' %} SELECTED {%- endif %}>Maschio</option>
      <option value='F'
        {%- if studente.sesso == 'F' %} SELECTED {%- endif %}>Femmina</option>
    </select>
  </td>
</tr>
```

JINJA TEMPLATE

l'attributo **SELECTED** è generato con un **IF** direttamente lato **Jinja**



# Recupero dei parametri con le routes (POST)

Vediamo ora come recuperare i dati passati via POST

Nelle routes è possibile anche specificare i metodi accettati

```
@app.route('/updatestudente', methods=['POST'])
def updatestudente():
    ...
```

PYTHON

I valori passati per POST finiscono nell'oggetto **Request**  
Per usarlo è necessario importare il modulo opportuno

```
from flask import Flask, render_template, request
```

PYTHON



# Recupero dei parametri con le routes (POST)

Il metodo mappato sulla route **/updatestudente** diventa

```
@app.route('/updatestudente', methods=['POST'])
def updatestudente():
    idstudente = request.form.get("id")      # se non esiste la chiave restituisce None
    cognome = request.form.get("cognome")
    nome = request.form.get("nome")
    sesso = request.form.get("sesso")
    matricola = request.form.get("matricola")
    datanascita = request.form.get("datanascita")
    sql = """
        update studenti set matricola = '{matricola}', cognome = '{cognome}', nome = '{nome}',
        sesso = '{sesso}', datanascita = '{datanascita}'
        where id = '{idstudente}' """.format(**vars())
    # eseguo la query di update/insert/delete e torno alla scheda
    update_delete_insert_query(sql)
    return studente_form(idstudente)
```

PYTHON

NOTA: Le query di update/insert/delete non restituiscono dati ma necessitano di **commit**

```
def update_delete_insert_query(sql):
    cursor = cnx.cursor(dictionary=True)
    cursor.execute(sql)
    cnx.commit()
```

PYTHON



# Esercizi - 5

Creare un metodo **nuovostudente()** mappato sulla route  
**/nuovostudente** per gestire l'inserimento di una nuova riga nella  
tabella **studenti**

**suggerimento:** riutilizzare i template **schedastudente.html** che  
contiene già una buona parte di quello che ci serve....

magari basta cambiare qualcosa...

20 minuti





Giulio Angiani  
I.I.S. "Blaise Pascal" - Reggio Emilia