USDA Nutrient Database Editor with Flask Framework

Firstly, a new “usda.sql3” database file is created from the USDA Nutrition data files: https://www.ars.usda.gov/ARSUserFiles/80400535/DATA/SR/sr28/dnload/sr28asc.zip by using the “usda-sqlite” Github repository: https://github.com/alyssaq/usda-sqlite

For more information and documentation of the USDA Nutrition Database:

- Download Links: https://www.ars.usda.gov/northeast-area/beltsville-md-bhnrc/beltsville-human-nutrition-research-center/methods-and-application-of-food-composition-laboratory/mafcl-site-pages/sr11-sr28/
- Documentation: https://www.ars.usda.gov/northeast-area/beltsville-md-bhnrc/beltsville-human-nutrition-research-center/methods-and-application-of-food-composition-laboratory/mafcl-site-pages/sr28-page-reports/

After that, a critical “crud.py” file was written to connect to the usda.db database and execute queries on it via Flask framework:

```
from flask import Flask, request, render_template
import sqlite3

app = Flask(__name__)
```

The next step is to have a homepage file: “index.html” in a “templates” folder:

```
<!DOCTYPE html>
<html>
<head>
    <title>Food Website Home</title>
</head>
<body>
    <h2>Welcome to the Food Website</h2>
    <a href ="/categories/dairy_and_egg_products"> Dairy and Egg Products</a><br><br>
    ...
</body>
</html>
```

In order to show each food-group page as a link, those parameters are used:

Food Group

In the “crud.py” file, “index.html” is set to be the homepage file:

```
@app.route("/")
def index():
    return render_template("index.html")
```

And each “food_group_page.html” file is associated with their related functions:

```
@app.route("/dairy_and_egg_products")
def dairy_and_egg_products():
```

When a link is clicked such as: “Dairy and Egg Products,” those instructions under its related function in “crud.py” are executed; firstly, it connects to the “usda.db” database and ignores any errors while decoding data:

```
con = sqlite3.connect("usda.db")
con.text_factory = lambda b: b.decode(errors='ignore')
```

Then, the sql query which fetches data of the clicked food group is executed:

```
con.row_factory = sqlite3.Row
cur = con.cursor()
cur.execute("SELECT id, short_desc, nitrogen_factor, protein_factor,
fat_factor, calorie_factor FROM food WHERE food_group_id=100")
rows = cur.fetchall()
```

Finally, the fetched data is sent to the related .html file as a “rows” list:

```
return render_template("dairy_and_egg_products.html",rows = rows)
```

A new page is created for both viewing and updating foods called: “viewupdate.html”

```
<table border=3>
    <thead>
        <td>ID</td>
        <td>Food Group ID</td>
        <td>Long Description</td>
        <td>Short Description</td>
        <td>Manufacturer</td>
        <td>Scientific Name</td>
    </thead>
```

```
    <tr>
        <td>{{rows[0]["id"]}}</td>
        <td>{{rows[0]["food_group_id"]}}</td>
        <td>{{rows[0]["long_desc"]}}</td>
        <td>{{rows[0]["short_desc"]}}</td>
        <td>{{rows[0]["manufac_name"]}}</td>
        <td>{{rows[0]["sci_name"]}}</td>
    </tr>
    <tr>
        ...
    </tr>
</table>
```

Here, “rows” contains only one row to be updated, so; rows[0][“column_name”] will give the column values of the returned row data which is fetched from the “crud.py” file:

```
@app.route("/viewupdate", methods=['POST'])
def viewupdate():
    id = request.form["id"]
          ...
    con = sqlite3.connect("usda.db")
    con.text_factory = lambda b: b.decode(errors='ignore')
    con.row_factory = sqlite3.Row
    cur = con.cursor()
    ...
    cur.execute("SELECT id, food_group_id, long_desc, short_desc,
    manufac_name, sci_name FROM food WHERE id = ?",[id])
    rows = cur.fetchall()
        ...
    return render_template("viewupdate.html",
    rows=rows, rwg=rwg, rnt=rnt)
```

WHERE id = request.form[“id”] comes from the form post action in the table pages. After that, the UPDATE query must be run from the viewupdate.html file. There are 6 parameters; the first one is hidden: “id” value of the “food” table, and the others are requested with text boxes to update the database: food_group_id, long_desc, short_desc, manufac_name and sci_name. These parameters are taken from “viewupdate.html” page:

```
id = request.form["id"]
food_group_id = request.form["food_group_id"]
long_desc = request.form["long_desc"]
short_desc = request.form["short_desc"]
manufac_name = request.form["manufac_name"]
sci_name = request.form["sci_name"]
```

After that, each value must be checked whether they contain a new value or not (they’re empty). NOTE: food_group_id.isdigit() checks if the input for this column is integer:

```
if len(food_group_id) != 0:
    if food_group_id.isdigit():
        cur.execute("UPDATE food SET food_group_id = ? WHERE id = ?",
        [food_group_id, id])
```

It is also considered that if a data is requested to be deleted from a row, it will be possible by simply writing a space or spaces in the text box area; strip() will clear the data:

```
if len(long_desc) != 0:
    if long_desc.isspace():
        cur.execute("UPDATE food SET long_desc = ? WHERE id = ?",
        [long_desc.strip(), id])
    else:
        cur.execute("UPDATE food SET long_desc = ? WHERE id = ?",
        [long_desc, id])
...
```

When any UPD is clicked from a table page, it re-sends the id of the related food row:

```
{% for row in rows %}

    <tr>

        <td>{{row["id"]}}</td>
        <td>{{row["food_group_id"]}}</td>
        <td>{{row["short_desc"]}}</td>
        <td>{{row["nitrogen_factor"]}}</td>
        <td>{{row["protein_factor"]}}</td>
        <td>{{row["fat_factor"]}}</td>
        <td>{{row["calorie_factor"]}}</td>


        <td><form action="/viewupdate" method="post">
        <input type="hidden" value={{row["id"]}} name="id">
        <input type="hidden" value="" name="food_group_id"/>
        <input type="hidden" value="" name="long_desc"/>
        <input type="hidden" value="" name="short_desc"/>
        <input type="hidden" value="" name="manufac_name"/>
        <input type="hidden" value="" name="sci_name"/>
        <input type="submit" value="UPD"/></form></td>

        ...
    </tr>

{% endfor %}
```

For the updating process, the non-entered values are sent as hidden and empty data because they must be sent on every call of the viewupdate() function as it is always waiting for those variables with the post method:

```
@app.route("/viewupdate", methods=['POST'])
def viewupdate():
    id = request.form["id"]
    food_group_id = request.form["food_group_id"]
    long_desc = request.form["long_desc"]
    short_desc = request.form["short_desc"]
    manufac_name = request.form["manufac_name"]
    sci_name = request.form["sci_name"]
```

After that, the viewupdate() function starts running. When there is no value in the food_group_id, long_desc, short_desc_manufac_name and sci_name, the if-else conditions will prevent any unnecessary update on the tables of the database:

```
 if len(food_group_id) != 0:
     if food_group_id.isdigit():
         cur.execute("UPDATE food SET food_group_id = ? WHERE id = ?",
         [food_group_id, id])
 if len(long_desc) != 0:
     if long_desc.isspace():
         cur.execute("UPDATE food SET long_desc = ? WHERE id = ?",
         [long_desc.strip(), id])
     else:
         cur.execute("UPDATE food SET long_desc = ? WHERE id = ?",
         [long_desc, id])
 if len(short_desc) != 0:
     if short_desc.isspace():
         cur.execute("UPDATE food SET short_desc = ? WHERE id = ?",
         [short_desc.strip(), id])
     else:
         cur.execute("UPDATE food SET short_desc = ? WHERE id = ?",
         [short_desc, id])


 if len(manufac_name) != 0:
     if manufac_name.isspace():
         cur.execute("UPDATE food SET manufac_name = ? WHERE id = ?",
         [manufac_name.strip(), id])
     else:
         cur.execute("UPDATE food SET manufac_name = ? WHERE id = ?",
         [manufac_name, id])
 if len(sci_name) != 0:
     if sci_name.isspace():
         cur.execute("UPDATE food SET sci_name = ? WHERE id = ?",
         [sci_name.strip(), id])
     else:
         cur.execute("UPDATE food SET sci_name = ? WHERE id = ?",
         [sci_name, id])
```

And then, the updated row data of the selected food is displayed with that:

```
cur.execute("SELECT id, food_group_id, long_desc, short_desc,
manufac_name, sci_name FROM food WHERE id = ?",[id])
rows = cur.fetchall()
...
```

This SELECT query prints the selected food's information. Under this row, blank text boxes appear to be filled if an UPDATE query is requested to be run by the user:

```
<tr>
    <td><form action="/viewupdate" method="post">
    <td><input type="hidden" value={{rows[0]["id"]}} name="id"/></td>
    <td><input type="text" name="food_group_id"/></td>
    <td><input type="text" name="long_desc"/></td>
    <td><input type="text" name="short_desc"/></td>
    <td><input type="text" name="manufac_name"/></td>
    <td><input type="text" name="sci_name"/></td>
    <td><input type="submit" value="Update"></td>
</tr>
```

Below that, "viewupdate.html" shows Weight & Nutrition data of the updated food:

```
<h4>Weight</h4>
<table border=3>
    <thead>
        <td>Sequence Number</td>
        <td>Amount</td>
        <td>Description</td>
        <td>GM Weight</td>
        <td>Num Data PTS</td>
        <td>STD Dev</td>
    </thead>


    {% for row in rwg %}
        <tr>
            <td>{{row["sequence_num"]}}</td>
            <td>{{row["amount"]}}</td>
            <td>{{row["description"]}}</td>
            <td>{{row["gm_weight"]}}</td>
            <td>{{row["num_data_pts"]}}</td>
            <td>{{row["std_dev"]}}</td>
        </tr>
    {% endfor %}
</table>
```

```
<h4>Nutrition</h4>
<table border=3>
    <thead>
        <td>Nutrient ID</td>
        <td>Amount</td>
        <td>Units</td>
        <td>Tag Name</td>
        <td>Name</td>
        <td>Num Decimal Places</td>
        <td>SR Order</td>
        <td>Num Data Points</td>
        <td>STD Error</td>
        <td>Source Code</td>
        <td>Num Studients</td>
        <td>MIN</td>
        <td>MAX</td>
        <td>Modific.Date</td>
    </thead>

    {% for row in rnt %}
        <tr>
            <td>{{row["nutrient_id"]}}</td>
            <td>{{row["amount"]}}</td>
            <td>{{row["units"]}}</td>
            <td>{{row["tagname"]}}</td>
            <td>{{row["name"]}}</td>
            <td>{{row["num_decimal_places"]}}</td>
            <td>{{row["sr_order"]}}</td>
            <td>{{row["num_data_points"]}}</td>
            <td>{{row["std_error"]}}</td>
            <td>{{row["source_code"]}}</td>
            <td>{{row["num_studients"]}}</td>
            <td>{{row["min"]}}</td>
            <td>{{row["max"]}}</td>
            <td>{{row["modification_date"]}}</td>
        </tr>
    {% endfor %}
</table>
```

Weight rows are extracted from the rwg list and Nutrition rows are extracted from the rnt list, which are fetched from the database by using the queries in the “crud.py” file. There are columns from both nutrient and nutrition tables which have two common values: “nutrient_id” of the nutrition table and “id” of the nutrient table. So, INNER JOIN function is used to get those columns from both the tables by checking their equality in the query:

```
...
cur.execute("SELECT sequence_num, amount, description, gm_weight,
num_data_pts, std_dev FROM weight WHERE food_id = ?",[id])
rwg = cur.fetchall()
cur.execute("SELECT nutrient_id, amount, units, tagname, name,
num_decimal_places, sr_order, num_data_points,std_error, source_code,
num_studients, min, max, modification_date FROM nutrient INNER JOIN
nutrition ON nutrition.nutrient_id=nutrient.id WHERE food_id=?",[id])
rnt = cur.fetchall()
return render_template("viewupdate.html", rows=rows, rwg=rwg, rnt=rnt)
```

And finally, each list of row data that are fetched from those three different sql queries are sent by “return render_template(“viewupdate.html”, rows=rows, rwg=rwg, rnt=rnt)” function to the “viewupdate.html” file which requires them to print as rows. “viewupdate” section calls itself each time when an UPDATE operation is processed.

Home Page || Previous Page

Food Update: Spices and Herbs

ID	Food Group ID	Long Description	Short Description	Manufacturer	Scientific Name	
2001	200	Spices, allspice, ground	ALLSPICE,GROUND		Pimenta dioica	
						Update

Weight

Sequence Number	Amount	Description	GM Weight	Num Data PTS	STD Dev
1	1.0	tsp	1.9		
2	1.0	tbsp	6.0		

Nutrition

Nutrient ID	Amount	Units	Tag Name	Name	Num Decimal Places	SR Order	Num Data Points	STD Error	S
203	6.09	g	PROCNT	Protein	2	600	35	0.336	1
204	8.69	g	FAT	Total lipid (fat)	2	800	214	0.12	1
205	72.12	g	CHOCDF	Carbohydrate, by difference	2	1100	0		4

Near the UPD button on each row, there is also a DEL button which deletes the data of the chosen row from the “food”, “weight”, “nutrition” and “nutrient” tables. This button sends only “id” and “food_group_id” values from the selected row to be deleted which are enough to run DELETE queries for the 4 tables: “food”, “weight”, “nutrition” and “nutrient”:

```
        <td><form action="/delete" method="post">
        <input type="hidden" value={{row["id"]}} name="id">
        <input type="hidden" value={{row["food_group_id "]}}
        name="food_group_id"/>
        <input type="submit" value="DEL"/></form></td>
    </tr>
{% endfor %}
```

In the “crud.py” file, there is also a “delete” section for this operation which executes the DELETE queries related to the selected row and all its data is deleted from the 4 tables:

```
cur.execute("DELETE FROM food WHERE id = ?", [id])
cur.execute("DELETE FROM weight WHERE food_id = ?", [id])
cur.execute("DELETE FROM nutrition WHERE food_id = ?", [id])
cur.execute("DELETE FROM nutrient WHERE id = (SELECT nutrient_id
FROM nutrition WHERE food_id = ?)", [id])
con.commit()
```

After the deletion, the same page of the food group must be displayed because the DEL button is clicked in the food group page. To do this, the “delete()” section must return the row data of the current page back to the same food group page:

```
cur.execute("SELECT id, food_group_id, short_desc,
nitrogen_factor, protein_factor, fat_factor, calorie_factor
FROM food WHERE food_group_id = ?", [food_group_id])
rows = cur.fetchall()
if food_group_id == "100":
    return render_template("tables/dairy_and_egg_products.html",
    rows=rows)
elif food_group_id == "200":
    ...
```

Returning back to the same page is provided by if-else conditions, so that each DEL process will result in showing the same page with the requested row deletion.

Home Page

Spices and Herbs

ID	Group ID	Short Description	Nitrogen	Protein	Fat	Calorie		
2001	200	ALLSPICE,GROUND	6.25	3.36	8.37	2.35	UPD	DEL
2002	200	ANISE SEED	6.25	3.36	8.37	2.9	UPD	DEL
2003	200	SPICES,BASIL,DRIED	6.25	2.44	8.37	3.0	UPD	DEL