Resumen

1. Initial project description

We are going to create a database which use is going to be directly related to aliments. The goal of our database is to provide a determined patient with a diet following its main characteristics (age, height, weight) Therefore, we will need a group of nutritionists. They are the ones who will be in charge of the development of the diets. We will storage data for aliments mainly, and we will use this data in order to create new diets for our patients in the nutritional center.

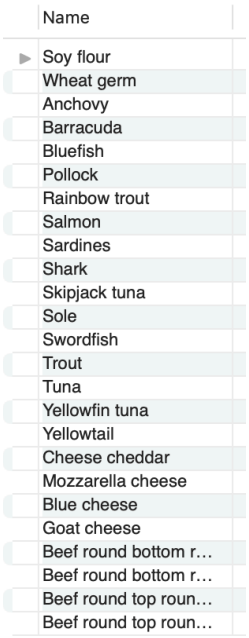
We will use a relational database, a type of database that stores and provides access to data points that are related to one another.

In order to generate data for our database, we will use csv files for some of our tables. For the rest of the tables, we will try to generate data automatically with the tools we will see in class (Python)

2. RDBMS Deployment

In the schema of our database, we will have the following tables: Food, Patient, Nutritionist, Diet and List Of Food. One Diet will consist of different kinds of Food, a Diet will be created by a Nutritionist (although a Nutrionist can create more than one) and a Nutritionist will take care of one or more Patients. The table List Of Food is created as a consequence of the relation between Diet and Food (a relation M:N creates a new table including the primary keys of both tables included in the relation)

Firstly, we have introduced the table Food, which has the food ID as the primary key. The rest of columns of the table are related with information such as the energy in kcal and KJ that provides the food, the quantity of Protein and Fiber in food in every 100 grams... Then the table to register the information about the Patient has been created and fulfilled with relevant information about it in order to organize the new diet. It also stores the current diet that the person is following and the intolerances, information that can be quite useful in order to design a new diet.



Name
Soy flour
Wheat germ
Anchovy
Barracuda
Bluefish
Pollock
Rainbow trout
Salmon
Sardines
Shark
Skipjack tuna
Sole
Swordfish
Trout
Tuna
Yellowfin tuna
Yellowtail
Cheese cheddar
Mozzarella cheese
Blue cheese
Goat cheese
Beef round bottom r…
Beef round bottom r…
Beef round top roun…
Beef round top roun…

Figure 1: First query output

The table of the Nutritionist is created also with an ID as the primary key. It also contains information about the name of the nutritionist and the specialty that he or she has (obesity, for example)

For the diets we have another table. As an identifier, we have and ID for the diet (primary key) Also, a description and a Creator are columns for this table. The Description will contain the different steps to follow in order to do the diet. The data which has been introduced in the table Food has been obtained by a csv file in https://www.fao.org/infoods/infoods/tablas-y-bases-de-datos/es/

3. Query Design

· Food with more than 20 grams of protein (Fig. 1): - SELECT NutritionalDB.Food.Name FROM NutritionalDB.Food WHERE Food.Protein > 20;

· All the names of patients treated by the doctor Pablo Moreno Garcia-Espina (Fig. 2): - SELECT NutritionalDB.Patient.Name FROM NutritionalDB.Patient JOIN NutritionalDB.Nutritionist ON Patient.Doctor = Nutritionist.idNutritionist WHERE Nutritionist.FullName = 'Pablo Moreno Garcia-Espina';

· Description of the diets created by the doctor Susana Rocio Fernandez Giaccomassi (Fig. 3) - SELECT NutritionalDB.Diet.Description FROM Nutrition-

Name
Juan
Ines
Pedro
Ines
Isabel
Antonio
Sofia
Pablo

Figure 2: Second query output

Description
Dieta sin lactosa para veganos
Dieta rica en fibra para mejorar la digestion
Dieta sin sal para mejorar la circulacion
Dieta para mejorar la salud intestinal
Dieta para reducir el estres
Dieta para mejorar la salud ocular
Dieta para mejorar la circulacion sanguinea
Dieta para mejorar la salud del higado
Dieta para mejorar la salud de las articulaciones

Figure 3: Third query output

alDB.Diet JOIN NutritionalDB.Nutritionist ON Diet.Creator = Nutritionist.idNutritionist WHERE Nutritionist.FullName = 'Susana Rocio Fernandez Giaccomassi';

· Obtain the description and the name of their creator of all diets with more than 10 foods (Fig. 4): - SELECT NutritionalDB.Diet.Description as 'Diets with more than ten foods', NutritionalDB.Nutritionist.FullName as 'Creator' FROM NutritionalDB.Diet JOIN NutritionalDB.Nutritionist ON Diet.Creator = Nutritionist.idNutritionist WHERE NutritionalDB.Diet.idDiet IN (SELECT NutritionalDB.Diet.idDiet FROM NutritionalDB.ListOfFood GROUP BY idDiet HAVING COUNT(*) > 10);

· Get the names and protein content of foods that have more protein than the average protein of all foods (Fig. 5): - SELECT NutritionalDB.Food.Name, NutritionalDB.Food.Protein FROM NutritionalDB.Food WHERE NutritionalDB.Food.Protein > (SELECT AVG(NutritionalDB.Food.Protein) FROM NutritionalDB.Food);

Diets with more than ten foods	Creator
Dieta rica en proteinas para deportistas	Pablo Moreno Garcia-Espina
Dieta para reducir el colesterol	Pablo Moreno Garcia-Espina
Dieta para aumentar la masa muscular	Pablo Moreno Garcia-Espina
Dieta para reducir la inflamacion	Pablo Moreno Garcia-Espina
Dieta para mejorar la concentracion	Pablo Moreno Garcia-Espina
Dieta para aliviar la artritis	Pablo Moreno Garcia-Espina
Dieta para mejorar la digestion	Pablo Moreno Garcia-Espina
Dieta para reducir la hinchazon abdominal	Pablo Moreno Garcia-Espina
Dieta para reducir la retencion de liquidos	Pablo Moreno Garcia-Espina
Dieta sin lactosa para veganos	Susana Rocio Fernandez Giaccomassi
Dieta rica en fibra para mejorar la digestion	Susana Rocio Fernandez Giaccomassi
Dieta sin sal para mejorar la circulacion	Susana Rocio Fernandez Giaccomassi
Dieta para mejorar la salud intestinal	Susana Rocio Fernandez Giaccomassi
Dieta para reducir el estres	Susana Rocio Fernandez Giaccomassi
Dieta para mejorar la salud ocular	Susana Rocio Fernandez Giaccomassi
Dieta para mejorar la circulacion sanguinea	Susana Rocio Fernandez Giaccomassi
Dieta para mejorar la salud del higado	Susana Rocio Fernandez Giaccomassi
Dieta para mejorar la salud de las articul…	Susana Rocio Fernandez Giaccomassi
Dieta baja en grasas	Ismael Navas Delgado
Dieta equilibrada para mantener el peso	Ismael Navas Delgado
Dieta sin cafeina para mejorar el sueno	Ismael Navas Delgado

Figure 4: Fourth query ouput

Name	Protein
Bagel	11
Buckwheat	13.3
Einkorn wheat	16.5
Kamut	14.3
Millet	11
Quinoa	14.1
Soy flour	39.6
Soybeans cooked	16.6
Spelt raw	14.6
Wheat bran	15.6
Wheat germ	27.2
Wheat whole grain	12.6
Whole-grain bread	10.4
Anchovy	20.3
Barracuda	25.4
Bluefish	20.4
Catfish	18

Figure 5: Fifth query output

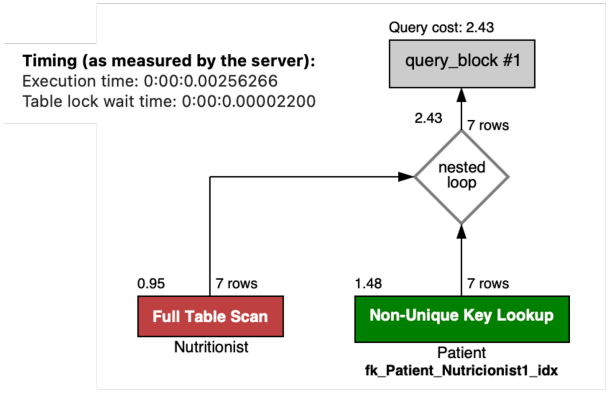


Figure 6: Second query without index

4. RDBMS Optimization

Firstly, we have changed the first query in order to make a deeper search. We can also search in the database according to the type of food, as we made a join between the table of Food and the table of Type of Food.

In order to make the search easier when it comes to diets related with sporty people, we have decided to design two indexes: one with Name and Kcal and another with Protein and Name. This index becomes useful with our first query, as we are looking firstly at the food with protein's value of more than 20 grams.

For the previous index, when we try to execute the first query apparently the index is not used. That could be because the improvement when the index is used is not considerably high when you compare it with the performance without the index.

Furthermore, as we will frequently look for the different diets that one nutritionist has created, we will create an index for the column "Full Name" in the table Nutritionist. We already have an index for the id of a nutritionist, but we will typically get a diet of a nutritionist by his concrete name. With this index we will get an advantage in the second and third query, as they use the full name of the nutritionist in order to make the query.

This picture (Fig. 6) shows the time which lasts the second query without the index.

This picture (Fig. 7) shows the time which lasts the second query with the index.

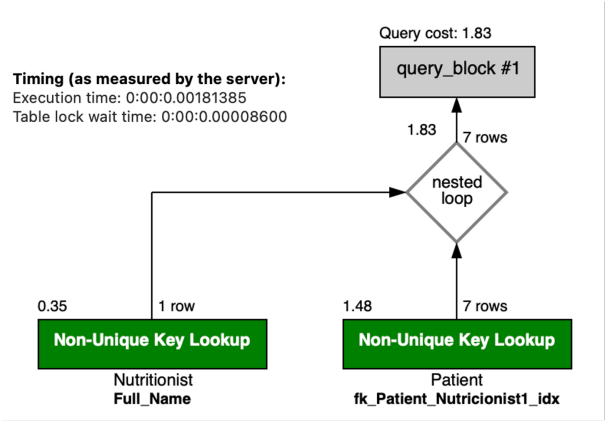


Figure 7: Second query with index

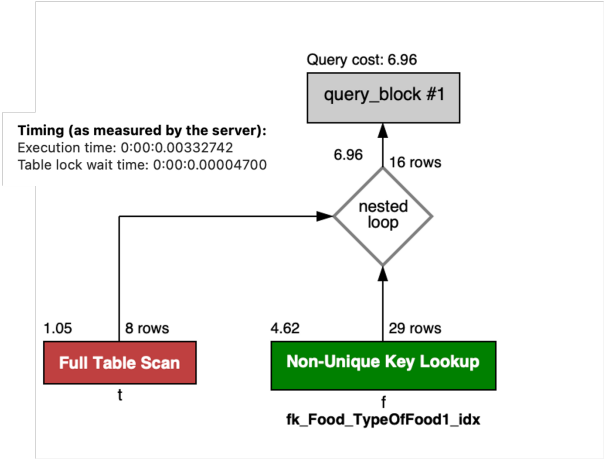


Figure 8: Extra query with index

We see how the time elapsed in the second time is lower than in the first time. The existence of the index makes the performance of the query higher.

We will also do queries which involve the name of the different patients we have in our database. Therefore, it would be suitable too to make an index with the name of the patients. Every time that we do a query in which the name of the patient is involved, we will get an advantage in terms of time, as we can see before.

We think that nutritionists may want to filter the aliments by the nutritional information and origin, so the query would be (Fig.8):

- SELECT f.Name FROM Food f JOIN TypeOfFood t ON f.TypeOfFood = t.Type WHERE t.Origin = 'animal' AND (f.Protein > 15 OR f.Iron > 5);

```python
def __init__(self, host, user, password, database):
    self.connection = pymysql.connect(host=host,
                                      user=user,
                                      password=password,
                                      db=database,
                                      charset='utf8mb4',
                                      cursorclass=pymysql.cursors.DictCursor)

    print("Successfully Connected!")

def close(self):
    self.connection.close()
    print("Connection Closed!")

def print_query(self, query):
    cursor = self.exec_query(query)
    print()
    for row in cursor:
        print(row)
```

Figure 9: Three first Python definitions

In this case, we have created an index for the table Food including the columns Protein and Iron, but it did not used it, neither the protein index created before.

5. Python

In order to develop the Python application to test the queries from an user interface, we have created a new project in PyCharm named NutritionalDB.

Specifically in the class NutritionalDB.py we have created three definitions (Fig. 9): one to initiate the connection to the database in SQL, one close the connection and another one to print the result of the queries we will implement next.

However, if we analyze the code for the ”print query” definition we see that another definition called ”exec query” is called. This new definition is used in order to send the query to the cursor created with the connection (Fig. 10)

After these first definitions, we have the definitions for each of the queries. When we made the queries in SQL we asked for concrete values of the columns (for example 20 grams of protein), however here we have parameterized the functions in order to make the query according to the user's goal.

When the user in Python calls the definition in order to ask for a food whose protein content is greater than X (Fig. 11), he has to specify this X value in the

```python
def exec_query(self, query):
    try:
        with self.connection.cursor() as cursor:
            cursor.execute(query)
            return cursor
    except DatabaseError as e:
        if e.args[1] == '#42000Unknown Database':
            print(e)
        else:
            print(e)
```

Figure 10: execute query function

```python
def gtProtein(self, protein):
    sql = 'SELECT NutritionalDB.Food.Name FROM Food WHERE Food.Protein > ' + str(protein) + ';'
    print()
    print('Food with more than ' + str(protein) + ' grams of protein:')
    self.print_query(sql)
```

Figure 11: Extra query with index

call to the method.

As we can see (Fig. 12), all of the definitions we have implemented in Python have the same structure. Firstly we create a string called sql in which we write the proper query. We have to insert here the different parameters we have in the header of the definition. Secondly, we print an empty line in order to make a separation in the terminal between queries. Then we print a text in order to make the user understand about the meaning of the query, and finally we execute the method ”print query” in order to show in the terminal the results of the query.

```
def moreThanAvg(self, column):
    sql = "SELECT NutritionalDB.Food.Name, NutritionalDB.Food." + column + \
          " FROM NutritionalDB.Food" \
          " WHERE NutritionalDB.Food." + column + " > (SELECT AVG(NutritionalDB.Food." \
          + column + ") FROM NutritionalDB.Food);"
    print()
    print('Food with more ' + str(column) + ' than the average:')
    self.print_query(sql)

def filterByInfoAndOrigin(self, info1, info2, value1, value2, origin):
    sql = "SELECT NutritionalDB.Food.Name FROM NutritionalDB.Food" \
          " JOIN NutritionalDB.TypeOfFood ON Food.TypeOfFood = TypeOfFood.Type" \
          " WHERE TypeOfFood.Origin = '" + origin + \
          "' AND (Food." + info1 + " > " + str(value1) + " OR Food." + info2 + \
          " > " + str(value2) + ");"
    print()
    print('Food with more than ' + str(value1) + ' grams of ' + str(info1) + ' or more than '
          + str(value2) + ' grams of ' + str(info2) + ' from ' + str(origin) + ' origin :')
    self.print_query(sql)
```

Figure 12: Examples of definitions in Python