University of Zürich

Master Basic Module

Data Curation in the Swiss Feed Database

Nathalie TORRENT

supervised by

Prof. Dr. Michael Böhlen
Dept. of Informatics - Datenbanktechnology

January-February 2019

Contents

1 Introduction

The Swiss Feed Database contains detailled information about feeds available in Switzerland. By querying the database, it is possible to retrieve the nutritional value or the exact composition of feeds. Data is provided by the Agroscope which is the Swiss Center for Agricultural Research. The goal of this project is to develop a new solution that can be used by the researcher of the Agroscope in order to import data on the Swiss Feed Database. In the following report, the required steps for the installation of the database will be explained. Then the requirements expressed by Agroscope will be explained. Based on them, an excel import file will be defined and explained. Then the functioning of the import application will be detailled.

2 Installation

In order to install a local version of the database, the following tasks must be executed. Firstly PostgreSQL must be setted up, then Apache must be installed. Finally a local database must be created and the data must be loaded. After that, the local web application can be created.

The following steps are only required if aptitude, emacs, nodejs and npm are not already installed on the computer.

```
sudo apt-get install aptitude
sudo apt-get install emacs
sudo apt install nodejs
sudo apt install npm
```

2.1 PostgreSQL setup

Pggadmin and postgis can be installed using the following commands:

```
sudo aptitude install pgadmin3
sudo apt install postgis
```

Pgadmin is an application used to admin and manage PostgreSQL plattform whereas Postgis is an application that allows Postgres to use geographical objects. Into the postgres command shell, a user and a database are created.

```
sudo su - postgres
createuser -s name_of_user
createdb name_of_db
exit
```

The following command will open the client authentication configuration file in the previously installed text editor emacs. This file has to be modified so that given criteria are matched.

```
sudo emacs /etc/postgresql/10/main/pg_hba.conf:
```

The previouly created user name_of_user is allowed to have a local connection using Unix-domain sockets on all databases. The authentication method is peer, which means that the username in the operating system is used. All other users are allowed to have a local connection with a md5-encrypted password. All users are allowed to have a connection via TCP/IP using a md5-encrypted password.

```
local  all  name_of_user           peer
local  all  all                    md5
host   all  all      127.0.0.1/32  md5
```

At this point, PostgreSQL is totally setted up. The installation continues with the Apache setup.

2.2 Apache setup

The first line of the script installs the HTTP server Apache. The following sudo ufw commands are used to set up the firewall. This command will list the available applications which are Apache, Apache Full, Apache Secure and CUPS.

```
sudo apt install apache2
sudo ufw app list
```

After that, the status of the firewall is tested and ApacheFull is allowed to go through the firewall.

```
sudo ufw status
sudo ufw allow 'Apache Full'
```

Then, the good functioning of the server is tested. The first command verifies if the server is active and launches it. A final test is made by typing the localhost address into a web browser. If everything has been setted up correctly, a "it works" message will appear on the web page.

```
sudo systemctl status apache2
http://localhost
```

Finally, php must be installed in order to execute scripts on the server and to return the corresponding HTML content. As the computer used for this installation was running Ubuntu 18.04, the available php version was 7.2 This installation has been done in this way:

```
sudo apt-get install php libapache2-mod-php
sudo apt-get install php-pgsql
```

2.3 Creating and loading the local database

The local database has been loaded using the PostGres terminal. The subsequent commands are executed in order to connect to postgres, to create the database named tdfb and to create a superuser called php_client.

```
sudo su -postgres
createdb tfdb
createuser -s php_client
psql -d table_name -c "alter user php_client with password 'php_client';"
```

Finally the two last commands will duplicate the database.

```
psql -d tfdb
psql -d tfdb < dump.181207
psql -d tfdb -c "alter database tfdb set search_path to dataschema, public;"
```

At this point, the full database is loaded on the computer. The final step is the installation of the web application.

2.4 Web Application

In order to install the web application, the following commands must be executed in the feedbase directory. RSA public and private key pairs are generated. Then using those keys, a SSL connection is established.

```
cd ~/SW/Feedbase
ssh-keygen -t rsa -b 2048 -f jwtRS256.key
openssl rsa -in jwtRS256.key -pubout -outform PEM -out jwtRS256.key.pub
```

Finally the server is installed which allows to consult the web application of the database in a web browser at http://localhost:3000.

```
node -v
npm install
npm i npm@latest -g
node bin/www # without supervisor
npm install supervisor -g
npm start # if supervisor is installed
http://localhost:3000
```

If the application can't be seen in the web browser, it is possible that postgreSQL doesn't run on the same port as the web server. The postgreSQL port can be found be typing in the psql command shell /conninfo. This port must be the one written in the params.json file. In the params json file, the user php_client, the database tdbd and the port on wich postgreSQL run can be seen:

```
 "db": {
    "user": "php_client",
    "password": "php_client",
    "database": "tfdb",
    "host": "localhost",
    "port": 5432}
```

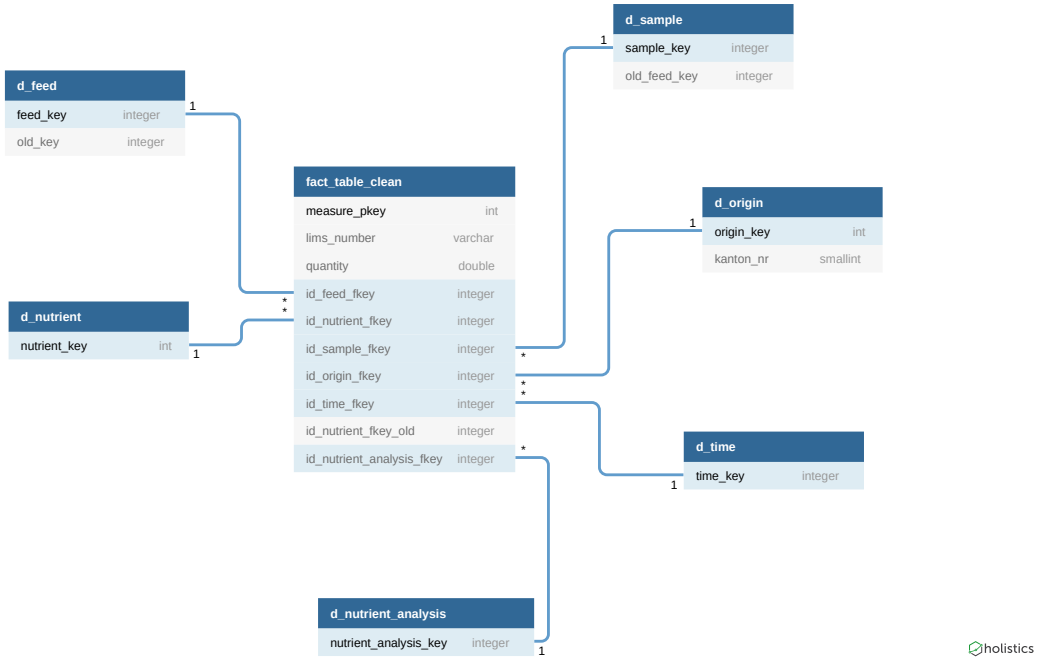


Figure 1: Relevant part of the database for the data import

3 Database Schema

Due to the size and the complexity of the database, only the tables relevant for the data import are represented in this star schema (figure 1). The full schema of the database can be seen in the appendix.

The central table is the fact_table_clean. This table, with a primary key named measure_pkey, references six other tables: d_feed , d_nutrient, d_sample, d_origin, d_time and d_nutrient _analysis.
The Agroscope receives feed samples on which they make analyses. In a oversimplified way, it can be admitted that one analysis correspond to one entry in the fact_table_clean. Information about the sample itself, such as biological and technical properties, are saved in the table d_sample. The fact_table_clean maps the sample table using the foreign key *id_sample_fkey*

Each sample is a feed sample, so there is a foreign key *id_feed_fkey* pointing to the table d_feed. A sample has a defined origin which is saved in the table d_origin, therefore the fact_table_clean keeps a reference of it using the foreign key *id_origin_fkey*. A sample also has time information, that is saved in d_time and referenced into the main table with the foreign key *id_time_fkey*. On a single sample, different nutrient analyses are done. So the fact_table_clean keeps track of the analysed nutrient with *id_nutrient_fkey* and of the respective analysis with the foreign key *id_nutrient_analysis_fkey*. A detailled example will be provided in the part 5.2.

4 Requirement Analysis

4.1 Current Solution

At present, the person in charge of the database at the Agroscope imports data by using an application. This person needs to fulfill a excel file consisting of two parts. The first part contains 103 columns in a fix order with information about the type of feed, the origin of the sample, the time and the properties of the sample. In the second part, nutrient analysis are written in a free order. Once the import file is filled out with data respecting the predefined order, the user can lauch an executable jar file in order to import data. This application launches a window in which the user is invited to choose an excel file, indicate his username and password. This java application will read the content of the excel file and put it into the database. The detailled data provided in the excel file will be added to the table fact_table_clean. This application only loads data into the database but not on the web application. After each execution of the import jar file, the user must manually execute a maintenance function so that the newly added data become accessible on the web application.

4.2 Requirements for the new application

4.2.1 Excel Template

Before considering any changes in the template files, it should be remembered that the main user of the future import application mentioned that she has been used to fill in the current excel template and that she is an expert in agronomy and not in informatics. Due to this reasons, the provided solution must be simple to use. The main idea is to continue using the same structure with some simplifications. The detailled new version of the excel file will be presented in the following part.

4.2.2 Import Application

In order to be simple to use, detailled explanations should be provided to the user when the import script goes wrong. As the import application should be able to import approximately 2000 data in a respectable time, it becomes really complicated to detect a mistake. Therefore detailled error reports would be really helpful. They can inform the user about the lines of the excel files that haven't been imported. They should also contain the reason why the excel line wasn't imported.

Below are listed errors that must be detected.

- when the provided data is not complete: no feed key or no lims number is given
- when the given feed key doesn't exist in the database
- when the given nutrient key doesn't exist in the database
- when the given nutrient analysis key doesn't exist in the database

If the lims number is missing, the corresponding data must not be imported, as the lims number is the main identification number into the Agroscope system. If the data was loaded, it would be loaded without it main property. The same occurs for the feed key. It's necessary to

detect when the three above mentioned keys doesn't exist in the database as new values into the corresponding table , d_feed, d_nutrient and d_nutrient_analysis, must be manually added. This manual step is required because the content of those table is complex. If any of those conditions aren't respected, the script won't import the data. At the end of the execution, the user will get a notification about the lines that haven't been imported.

Important remark
If a cell in the excel needs to contain a string, it is not allowed to use the character ' in it as this character is used in the sql statements. If they are used, the will cause mistakes. So if the user wants to write "ensillage d'herbes", he should write "ensillage d'herbes" instead. The ' character must not be used in any of the cells of the excel file.

5 Import File

5.1 Superficial structure of the import file

As mentionned before, the goal of the import application is to fill the table fact_table_clean. This table contains the following entry:

- id_feed_fkey
- id_origin_fkey
- id_time_fkey
- lims_number
- id_sample_fkey
- id_nutrient_fkey
- id_nutrient_fkey_old
- id_nutrient_analyses_fkey
- quantity

The template file should contain all the necessary information to fill the fact_table_clean. The import application won't fill the entry id_nutrient_fkey_old, as this is not used anymore. However this field hasn't been removed from the database as many item into the database are still referencing to the corresponding table.
Based on the entries in the fact_table_clean a template for the excel file has been defined. There are categories such as feed, origin, time, sample and analysis. As each sample can contain up to three time information, there will be three different time category into the excel file: harvest time, sampling time and arrival time. So in total the Excel file contains seven categories all related to the structure of the fact_table_clean. This categories which are feed, origin, harvest time, sampling time, arrival time, sample properties and abbreviations are written in capital letters on the figure 1. This figure shows an overview of the excel file. The first line of the excel file containing information in capital letters is a helper for the user because it recapitulates the main categories. Each category contains different columns header that are written on the line 2 of the excel file. The user will start inserting data at line 3. The detailled functioning of the excel file will be explained in the following part.

1	FEED	ORIGIN	HARVEST TIME	SAMPLING TIME	ARRIVAL TIME	SAMPLE PROPERTIES				ABBREV
2	**feed key**	Various columns about origin	Various columns about harvest time	Various columns about sampling time	Various columns about arrival time	**Lims number**	Main	Bio	Tech	Nutrient key % Analysis key
3	data									

Table 1: Basis structure of the excel file

1	FEED	ORIGIN
2	feed_key	country
3	1500	Switzerland
4		Switzerland
5	12000	China

Figure 2: Excel Template: feed information

5.1.1 Feed

Into the database, feeds are registered and can be retrieved using a feed key of type integer. As an example, if the user has received a sample of chicory, he must indicate it into the excel file. For that, he needs to retrieve the integer key corresponding to the feed with name chicory into the database. With a simple Select SQL statement, the corresponding feed key of 1500 is found. Therefore the user will write 1500 into the column feed key. The lines 4 and 5 on the figure 2 represent unallowed content for the feed key. The user is not allowed to leave the feed key empty or to put a feed key that doesn't exist into the database such as 12000. If one of this case occurs, the corresponding line won't be imported into the database.

5.1.2 Origin

Into the database, the origin can be retrieved using an origin key. However it's not asked to the user to directly provide the origin key. The user can describe the origin of his sample by filling the columns corresponding to the origin which are highlited in blue into the excel template. The columns name and the type of value that must be given are :

- country *string*
- canton *string*
- city *string*
- altitude class *string*
- altitude in meters *int*
- postal code *int*
- latitude *float*
- longitude *float*
- animal density *int*
- region number *int*
- region name *string*

As there could be sample without origin information, it's not mandatory to provide an origin so it implies that all the mentioned columns can be left empty. If the sample has a defined origin, it is possible that only part of the origin information is provided. As an exemple if only the country and the altitude in meters are provided, the user will only fill the corresponding two

1	FEED	ORIGIN										
2	feed_key	country	canton	city	altitude_class	altitude_in_meters	postal_code	latitude	longitude	animal_density	region_number	region_name
3	1500	Switzerland	Luzern	Urswil	< 600		6280	47.15401	8.289826		6	LU-AG
4	1400	Switzerland				1200				15		
5	1450	China										
6	1475											

Figure 3: Excel Template: origin information

columns. So it implies that if a origin is provided, only the columns having information must be filled. On the figure 3, four valid origin line are written. The line 3 to 5 show that it's not mandatory to fill all the columns while the line 6 shows that it's allowed to leave all the origin columns empty.

This origin information will be used by the import application to make a request into the database in order to retrieve the corresponding origin key which is of type integer. If the retrieval fails, because this origin informations are not already into the database, a new origin key will be generated. The origin keys of type int have sequential value. So if the last origin key in the database was 1000, the following key will be 1001. So under the origin key 1001, all the new origin information will be saved.

5.1.3 Harvest, Sampling and Arrival Time

Into the database, the time information can be retrieved by using a time key. However, it's easier for the user to write the time information into separated columns. All the time information must be written into the pink part of the excel file. As each sample can have up to three time information, there are three category into the excel file: the harvest time (moment 1), the sampling time (moment 2) and the arrival time (moment 3). For each moment, the following columns can be filled.

- day_i *Date format DD.MM.YYYY*
- season_en_i *string*
- season_de_i *string*
- season_fr_i *string*
- month_i *int*
- year _i *int*

Again, the user doesn't need to fill all the columns for a given moment. If only the year and the saison are available for the moment 1, he will only fill year_1, season_en_1, season_de_1, season_fr_1. As a side remark, if the user inserts a season, he must write it in the three different languages. If the user fills the columns day_i, he must really be attentive to use the right time date format wich is DD.MM.YYYY. If he doesn't do it, the import will fail. Moreover, if the user doesn't have any information about a moment, he can leave all the columns corresponding to this moment empty.

For readability reason, on the figure 4 , the columns season_de_i and season_fr_i have been left out. On the line 3, the three time information are filled. On the line 4, only the information about the harvest and the arrival time are filled. On the line 5, only the arrival time is provided. The lines 3 to 5 are all valid and will be taken into consideration, whereas the line 6 isn't as the date format is wrong. It is in the responsability of the user to use the right date format.

1	HARVEST TIME				SAMPLING TIME				ARRIVAL TIME			
2	day_1	season_en_1	month_1	year_1	day_2	season_en_2	month_2	year_2	day_3	season_en_3	month_3	year_3
3	02.04.2015		4	2015	02.03.2015	Winter	3	2015	04.05.2018	Spring	5	2018
4		Autumn		2018					04.12.2018	Winter	12	2018
5											5	2018
6	2018-05-02				2018/05/02							

Figure 4: Excel Template: time information

1	main properties of a sample								
2	lims_number	preparation_de	info_1	info_2	provenance	project_code	project_cod	batch_nr	labs_name
3	15-21997-004		Luzerne Heu		AGRIDEA				UFAG
4			Luzerne Heu						
5	15-21997-004				null				AOMC

Figure 5: Excel Template: sample information

For each of the available time information, the corresponding time key must be retrieved. If the key retrieval fails, the corresponding time information must be added into the database. Again the time key are sequentially added. So a new time key will be generated and all the corresponding time information will be added into the table d_time using this new time key.

5.1.4 Sample Information

The information about the sample must be written into the grey part of the excel file. The following columns can be filled:

- lims number *string*
- preparation de *string*
- info 1 *string*
- info 2 *string*
- provenance *string*
- project code *string*
- project code ext *string*
- batch nr *string*
- labs name *string*
- bi properties *string or integer*
- te properties *string or integer*

Summarized into the name bi properties and te properties are actually 64 fields that represent detailled information about the biological and technical properties of a sample. This properties can either be string or integer. The requested format for each bi or te column is indicated directly into the excel file.
Into the Agroscope system, the sample are referenced by a lims number which is one of the most important features. As mentioned before, it is not permitted not to provide a lims number into a excel line. It's common that not all the sample properties such as bi or te properties are known. Therefore, if not information is provided, all the columns, except the lims number can be left empty. On the figure 5, the line 3 and line 5 are valid lines because a lims number is provided. The line 4 is not valid even if there is information in the column info_1 because no lims number is provided.

New sample
On the figure 5, the lims number given in the line 3 is a lims number that doesn't exist in the table d_sample. As a new lims number is provided, a sample key will be sequentially generated

1	TVK	TSO	TSL	TSLM
2	180%2	144%7	158%2	163%2
3	892	901	99	
4		50	null	165

Figure 6: Excel Template: analysis information

and all the sample information will be inserted into the database using this newly generated sample key. On the respecting sample key, the following informations will be saved in the table d_sample: Info_1 will contain Luzerne Heu, the provenance will be Agridea and the labs name will be UFAG.

Overwritting an existing sample
As it can be seen, the lims number provided in the line 5 is identical to the one written in the line 3. This implies that when the import application will consider the line 5, it will detect that this lims number is already contained into the table d_sample. Therefore the corresponding sample key will be retrieved. In this case, the sample properties will be overwritten into the database. As no information is provided in the excel cell info_1, the content of info_1 in the database won't be changed. The cell provenance contains a null value, this means that the user wants to delete the provenance value. So the new content of provenance into the database will be null which corresponds to "no information provided". In the column labs name, AOMC is written. That means that the user want to change the content of the labs name into AOMC. So after the insertion, the database entry labs name will contain AOMC instead of UFAG.

5.1.5 Nutrient Analysis

The last part of the excel file contains information about nutrient analysis made on the sample. The nutrient analysis can be written in a free order. However the user must respect certain conditions. In the line 1 of the excel file, the user can write "help" information such as abbreviations. This line won't be read by the import application. In the line 2, the user must indicate the analysed nutrient and the employed analysis. He must respect the following format: Nutrient_key % Analysis_key. In the line 3 he must add the analysed quantity.
On the figure 6, some examples are shown. On line 1, the user wrote abbreviation such as TVK, TSO, TSL and TSLM in order to help him filling the excel file. The line 2 contains the nutrient analysis that have been made. As an example a quantity of 892g of nutrient 180 has been analysed with the analysis method 2. The Agroscope always provide analysis for a sample. However it's not necessary that all analyses are done, so the line 4 is valid even if not all the nutrient analysis are filled. The user must pay attention to provide valid nutrient and nutrient analyses keys. Valid keys are keys that already exist into the corresponding tables. If unvalid keys are provided, the corresponding nutrient analysis won't be imported.

As the nutrient analysis are made on sample, there is also a case distinction based on the fact that it is a new sample or an existing one.

New sample

	FEED	ORIGIN				HARVEST TIME		
2	feed_key	country	canton	city	altitude_class	day_1	season_en_1	month_1
3	**799**	corresponding information for origin_key **3334**				Info corresponding to time_key **10571**		

	SAMPLING TIME			ARRIVAL TIME				TVK	TSO	TSL
2	day_2	season_en_2	month_2	day_3	season_en_3	month_3	lims_number	**180%2**	**144%7**	**158%2**
3	Info corresponding to time_key **10572**			Info corresponding to time_key **10573**			15-21977		901	99

Figure 7: Excel example: New Sample

Considering that the line 3 of the figure 6 refers to a new sample mean that the nutrient analysis must be added into the database. So each analysis will be introduced into the fact_table_clean. At the end of the import, the analysis 180%2 with quantity 892, 144%7 with quantity 901 and 158%2 with quantity 99 will be contained into the database for the given sample.

Overwritting an existing sample It will be admitted that the line 4 of the figure 6 has the same lims number as the line 3. This means that nutrient analysis must be overwritten. The nutrient analysis 180%2 won't be changed into the database as no information is provided on line 4. The content of the nutrient analysis 144%7 will be changed from a quantity of 901 to a quantity of 50. The analysis 158%2 will be deleted as the corresponding cell contains the null value. Finally, a new analysis 163%2 with a quantity of 165 will be added into the fact_table_clean.

5.2 Detailled exemple of the insertion into the fact table clean

In this part, detailled examples of the insertion into the fact_table_clean will be presented. Again a distinction will be made if the sample is new or if it's an existing one.

New sample
On the figure 7 , the Excel file can be seen. On the line 3, the user wrote the information as explained before. In order to explain how the import works, the information filled by the user have been replaced by the corresponding keys. That's actually what the import application will do. It will either retrieve or create keys based on the information provided by the user.

As explained before, each sample contains up to three time information. Therefore there are three time key which are 10571, 10572 and 10573. The time keys implies that each nutrient analysis must also contain the three time information. So for each nutrient analysis and provided time information, a new measure_pkey will be generated, those keys are also generated in a sequential order. All the information will be added into the fact_table_clean using this generated key.
So the example contained into the excel file will lead to the fact_table_clean represented on the figure 8. It can be seen that each nutrient analysis has been tripled. So for each nutrient analysis and each time information, a new measure pkey has been generated. If there were only two time information, each nutrient analysis would only be duplicated.

Overwritting an existing sample
On the figure 9 it can be seen that the lims number is the same as in the previous example. Therefore, the nutrient analysis will be overwritten. If a comparison is made between the current excel file and the previous, it can be seen that the analysis 180%2 is new, that the content

Fact_table_clean

Measure pkey	Lims Number	Quantity	Id origin fkey	Id_time_ fkey	Id_feed_fkey	Id_nutrient fkey	Id_nutrient analysis fkey
1	15-21977	901	3334	10571	799	144	7
2	15-21977	901	3334	10572	799	144	7
3	15-21977	901	3334	10573	799	144	7
4	15-21977	99	3334	10571	799	158	2
5	15-21977	99	3334	10572	799	158	2
6	15-21977	99	3334	10573	799	158	2

Figure 8: Result into the fact table clean

1	**FEED**	**ORIGIN**					**HARVEST TIME**					
2	feed_key	country	canton	city	altitude_class		day_1	season_en_1	month_1			
3	**799**	corresponding information for origin_key **3334**					Info corresponding to time_key **10571**					
1	**SAMPLING TIME**			**ARRIVAL TIME**						TVK	TSO	TSL
2	day_2	season_en_2	month_2	day_3	season_en_3	month_3			**lims_number**	**180%2**	**144%7**	**158%2**
3	Info corresponding to time_key **10572**			Info corresponding to time_key **10573**					15-21977	55	null	40

Figure 9: Excel example: Overwritting Sample

of the analysis 144%7 will be deleted and that the quantity of the analysis 158%2 has changed. Therefore the fact_table_clean must be adapted.

As the analysis 158%2 already exists in the table, the corresponding measure pkey are retrieved and based on them the quantity will be changed. The modified quantity is highlighted in red on the figure 10 that represents the content of the fact_table_clean after the overwrite operation. As the user indicates a value of null for the analysis 144%7, the three measure pkey of this analysis will be retrieved. The relative database entry will be deleted. The analysis 180%2 is new analysis, therefore new measure pkey are generated: 12, 13, 14. The new analysis information are saved under those measure pkey.

6 Description of the import application

In order to describe how the import application works, the code will be first explained. Then for each code explanation, a leading example will be provided, this leading examples will show what the code does based on examples. In the examples, the values retrieved for the keys will be artificial values and not real values contained into the database.
The code concerning the windows handling the data import can be found in the views/upload.pug. The window controller is located in public/javascripts/controllers/upload.js. The Upload Controller calls the function /upload located into the file route/api/excel.js. All the code presented below is contained into the last mentioned file.

6.1 Downloading the file

Once the user filled the import excel file, he can go into a webbrowser to access the Swiss Feed Database. He must then connect as an admin and go on the /upload page (http://localhost:3000/upload).

Fact_table_clean

Measure pkey	Lims Number	Quantity	Id origin fkey	Id_time_ fkey	Id_feed_fkey	Id_nutrient fkey	Id_nutrient analysis fkey
~~1~~	~~15-21977~~	~~901~~	~~3334~~	~~10571~~	~~799~~	~~144~~	~~7~~
~~2~~	~~15-21977~~	~~901~~	~~3334~~	~~10572~~	~~799~~	~~144~~	~~7~~
~~3~~	~~15-21977~~	~~901~~	~~3334~~	~~10573~~	~~799~~	~~144~~	~~7~~
4	15-21977	40	3334	10571	799	158	2
5	15-21977	40	3334	10572	799	158	2
6	15-21977	40	3334	10573	799	158	2
12	15-21977	55	3334	10571	799	180	2
13	15-21977	55	3334	10572	799	180	2
14	15-21977	55	3334	10573	799	180	2

Figure 10: Result into the fact table clean after the overwrite

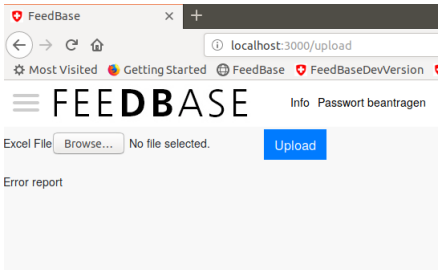


Figure 11: Upload page

This upload page, which is shown on the picture 11, allows the user to import an excel file. When the user clicks on upload, the import application will start.

The import application will handle the excel file using the package Sheetjs wich allows to manipulate spreadsheet file over a server. The import application will convert the excel file to a json file using a defined function of SheetJs. Json files allow a better treatement of data: each line of the excel file will correspond to a json object. A json object can be seen on the figure 12. It can be seen that the data is represented as an array with the column of the excel file being used as key to access the data. As an example, the value 1200 which was written under the column altitude in meters into the excel file is now a json element : altitude in meters : 1200. One big advantage of the convertion of a excel line into a json object is that only the cells containing information have been converted. However, the json object still maintain a reference to the excel line into the field rowNum.

The first part of the import application, which can be seen in the listing 1 converts the excel file into a json file. Then for each json object the function handlerecord is called. In the following part, a record is used as a synonym of a json object.

```
    const workbook = XLSX.read(req.file.buffer, {type: 'buffer'});
    const sheet_name_list = workbook.SheetNames;

```

```
1: Object {country: "Switzerland", altitude_in_meters: "1200", animal_density: "15", …}
  __rowNum__: 3
  144%7: "50"
  altitude_in_meters: "1200"
  animal_density: "15"
  country: "Switzerland"
  day_3: "2018-12-04"
  info_1: "Luzerne Heu "
  season_de_1: "Herbst"
  season_en_1: "Autumn"
  season_fr_1: "Automne"
  year_1: "2018"
```


Figure 12: Json object

```
  try {
    //the option range1 consider the second line of the excel file as the header
    const json = XLSX.utils.sheet_to_json(workbook.Sheets[sheet_name_list[0]],{range:1});
    var error_report = "";

    for (i=0; i<json.length; i++)
    {
      var err = await handlerecord(json[i]);
      error_report = error_report + err;
    }

    res.send(error_report);
  } catch (e) {
    console.log(e);
  }

```

Listing 1: Conversion excel to json

6.2 handlerecord

The handle record function is the main function that will treat each record (json object) and insert it into the database. Due to the length of this function, it has been splited into different listings to explain it easily. The listings combined together form the whole handlerecord function.
The function handlerecord uses many Builder functions. These functions are actually string builder. They will return strings that will be used as SQL statements. As the content of thoses functions consists of concatenating string, they won't be presented in details.

6.2.1 Prerequesites and feed information

The first part of the handle function can be seen on the listing 2. This first step make sure that the record contains the main required properties. At first, the function checks if the record contains a feed key and a lims number. If one of this information is missing, the record will be ignored and thus won't be imported.
If the record is valid, the potential feed key will be retrieved from the record. It is necessary to make sure that this feed key is valid. In order to do that, a sql statement is generated by calling the function BuilderFeedSQL. This function returns a select statement: "Select feed_key

FEED	
feed_key	
1500	→ saved_feed_key = 1500
	→ missing feed key so the line won't be imported
12000	→ unvalid feed key so the line won't be imported.
1450	→ saved_feed_key = 1450

Figure 13: Code example: Feed

from d_feed where feed key =*potential feed key*". This select statement can be seen as a dummy request because it only makes sure that the given feed key exists in the database. If the execution of this SQL (db.one(sql_feed)) statement provides an error, this will mean that the given feed key doesn't exist in the database. Therefore, the whole record won't be imported. The user will get informed at the end of the execution that the record hasn't been imported.

```
if (record.feed_key == undefined){
  var error_message ="In the excel file a feed key is missing on line " +
 record.__rowNum__ + " so this line won't be imported";
  return error_message;
}
if(record.lims_number == undefined){
  var error_message ="No lims number is provided in the excel line " + record
 .__rowNum__ + " so this line won't be imported");
  return error_message;
}

//FEED INFORMATION
var potential_feed_key = record.feed_key;
var sql_feed = BuilderFeedSQL(potential_feed_key);
try{
  var p1 = await db.one(sql_feed);
  saved_feed_key = p1.feed_key;
} catch (error) {
  //feed key don't exist
  var error_message ="The feed key " + potential_feed_key + " doesn't exist
 in the database, it must be added manually. The excel line" +record.
 __rowNum__  +" won't be imported");
  return error_message;
}
```

Listing 2: Handlerecord part 1

Leading example

In the figure 13, the second column shows what the code does on a real example. It detects if a feed key is missing, if an unvalid feed key is provided. If a valid feed key is provided, it will be stored into the variable saved feed key.

6.2.2 Origin information

As explained in the description of the excel template, it's not mandatory to fulfil the columns about the origin of the sample. Therefore, the handlerecord function must consider the fact where no origin information is provided. For that, the function tests if no data is provided for each origin key. Under the appelation origin keys are all the columns of the excel file related

to origin such as country, city, altitude in meters. This simple function return true if origin information is given and false if not (Listing 3).

```
function originprovided(record){
  if(record.country == undefined && record.canton == undefined && record.city
    == undefined
    && record.altitude_class == undefined && record.altitude_in_meters ==
    undefined
    && record.postal_code == undefined && record.latitude == undefined &&
    record.longitude == undefined
    && record.animal_density == undefined && record.region_number== undefined
    && record.region_name == undefined){
    return false;
  }
  else {
    return true
  }}
```

Listing 3: originprovided

Origin provided

If an origin is provided, the BuilderOriginSQL is called. This function will return an array of string. The first string into the array is a select statement and the second one is an insert statement.

The select statement has this structure: "*SELECT origin_key FROM d_origin where origin_key_1 = value1 AND .... AND origin_key_n IS NULL*". At first, the select statement is executed. If a origin key is retrieved, it will be saved into the variable saved_origin. On the other hand, if the origin key retrieval fails, this implies that a new origin must be inserted into the table d_origin. This is done by executing the second SQL statement which has the following structure: "*INSERT INTO d_origin(origin_key_1, ...,origin_key_n) VALUES(value1,...,null) RETURNING origin_key*". This insert statement also returns the newly created origin key that will be saved into the variable saved_origin.

Origin not provided

If the origin is not provided, the BuilderOriginEmpty is called, this will return this select statement: "*Select origin_key from d_origin where origin_key_1 IS NULL AND .... AND origin_key_n IS NULL*". The maintenance script doesn't allow data without an origin key. Therefore this SQL statement must be executed. It has been decided to execute this statement in order to avoid hard coding the origin key corresponding to an empty origin. Again the retrieved key will be saved into the variable saved_origin (Listing 4).

```
//ORIGIN INFORMATION
  if (originprovided(record) == true){
    var origin_research = BuilderOriginSQL(record);
    try{
      //Checking if the origin is already in the database.
      var p2 = await db.one(origin_research[0]) ;
      saved_origin = p2.origin_key;
    }
    catch(error){
      //origin don't exist it needs to be added
      try{
        var p3= await db.one(origin_research[1]);
```

ORIGIN				
Country	Canton	City	Altitude_class	
Switzerland	Zürich			→ saved_origin = 12
China	Beijing			→ New origin inserted → saved_origin = 150
				→ Case empty origin →saved_origin = 3257

Figure 14: Code example: Origin

```
        saved_origin = p3.origin_key;
      }
      catch(error){
        console.log("The insertion of a new origin key failed");
      }
    }
  }
  else{
    var empty_origin_research = BuilderOriginEmpty () ;
    try{
      //Searching the empty origin into the database
      var p2b = await db.one(empty_origin_research) ;
      saved_origin = p2b.origin_key;
    }
    catch(error){
      console.log(error);
    }
  }
```

Listing 4: Handlerecord part 2

Leading example

In the figure 14, the three mentionned cases are shown. In order to make the sql statement easier to understand, the possible origin keys have been reduced to four.
The first line contains an origin that is already into the database. The statement "*SELECT origin_key FROM d_origin WHERE country = 'Switzerland' AND canton = 'Zürich' AND city IS NULL AND altitude_class IS NULL*" returns the origin key 12.
The second line illustrates the case where the select statement doesn't return any key. Therefore the insert statement "*INSERT INTO d_origin(country, canton) VALUES (China, Beijing) RETURNING origin_key*" is executed. This statement will return the new origin key of 150.
The last line shows an empty origin. In this case the select statement "*SELECT origin_key FROM d_origin WHERE country IS NULL AND canton IS NULL AND altitude_class IS NULL*" is executed and returns the key corresponding to an empty origin which is 3257.

6.2.3 Time information

As there can be up to three time information, it is necessary to check if time information is provided for each moment (moment from 1 to 3). Each time information is considered separately. As there is no guarantee that a time information is provided, the function timeprovided checks if the time columns are filled for the corresponding moment. If time information is provided, the BuilderTimeSQL is called. This function returns an array of string. The first string is a

select statement and the second is an insert statement. Again the select statement that aims to retrieve the time key is executed first. If a time key is retrieved, it will be saved into the array saved time. If the retrieval fails, the insert statement will be executed and the newly generated time key will be saved into the table saved time.

```
// TIME INFORMATION
 var i;
 for (i =1; i<=3; i++){
   if (timeprovided(record, i) == true){
     var time_research = BuilderTimeSQL(record, i);
     try{
       //Searching if the time information i are already in the database
       var p4 = await db.one(time_research[0]) ;
       saved_time[i-1] = p4.time_key;
     }
     catch(error){
       //The time information doesn't exist in the database, need to be added.
       try{
         //generating a new time key
         var p5= await db.one(time_research[1]);
         saved_time[i-1] = p5.time_key;
       }
       catch(error){
         console.log("The insertion of a new time key failed", error);
         var error_format ="ERROR : Wrong date format for the line " + record.
   __rowNum__ + " This line won't be imported \n"
         return error_format;
       }}

    }
    else{
      continue;
    }}
```

Listing 5: Handlerecord part 3

Leading example

In the figure 15, two time information are provided: the sampling and the arrival time.
At first, the handlerecord function will consider the harvest time. The timeprovided function will return false as no information is provided. Therefore the saved time[0] will be undefined.
Secondly, as information is provided about the sampling time, the timeprovided function will return true and SQL statements will be generated. The select statement, "*SELECT time_key FROM d_time WHERE tday IS NULL AND month = 5 AND year = 2015 and moment = 2 and ts = 20150501 and te=20150531*", will return the time key 150, this will be saved into saved time[1]. It is necessary to point out that the SQL statement contains a entry called moment. This entry keeps track of the different time information: a moment of 1 represents the harvest time, a moment of 2 the sampling time and a moment of 3 the arrival time. The entry ts and te represent the time intervals.
Finally the arrival time is considered. The corresponding select statement fails to retrieve a time key, therefore a new time key must be generated. This is done by executing the insert statement "*INSERT INTO d_time(year, moment, ts, te) VALUES (2019,3, 20190101, 20191231) RETURNING time_key*".

HARVEST TIME			
Day_1	Month_1	Year_1	
			→ timeprovided(record)= false Saved_time[0] = undefined
SAMPLING TIME			
Day_2	Month_2	Year_2	
	05	2015	→ timeprovided(record) = true Time already into the database Saved_time[1] = 150
ARRIVAL TIME			
Day_3	Month_3	Year_3	
		2019	→ timeprovided(record) = true Time not into the database, insertion Saved_time[2] = 300

Figure 15: Code example: Time

6.2.4 Sample information

In the listing 6, the sample information are treated. At first the BuilderSampleSQL is called, this builder returns the string "*SELECT sample_key FROM d_sample WHERE lims_number = 'ValueLimsNumber'*".
The result of the execution of this sql statement will define if a new sample is provided or if a existing sample must be overwritten.

Overwritting an existing sample
If the select statement returns a sample key, this means that the given lims number was already into the database. Therefore the sample information must be overwritten. In order to do that, the BuilderSampleOverwrite is called. This function returns a string in the form of "*UPDATE d_sample SET key1 = VALUE, ..., keyn = null WHERE lims_number = ValueLimsNumber*". If the update statement contains keyn= null , this means that the current keyn value into the database will be deleted.

New sample
On the other hand if the provided lims number is not into the database, the select statement won't return any sample key. If this happens, the BuilderSampleInsert will be called and will return an insert statement. Based on this insert statement, a new entry will be added into the table d_sample. Once the insertion is done, the new sample key will be returned and will be saved into the variable saved sample key.

```
//SAMPLE INFORMATION
try{
  //The lims number is already in the database, it must be overwritten
  var sample_research = BuilderSampleSQL(record);
  var p6 = await db.one(sample_research);
  saved_sample_key = p6.sample_key;
    try{
      var sample_overwrite = BuilderSampleOverwrite(record);
      var p6b= await db.none(sample_overwrite);
      overwrite = true;
    }
    catch(error){
      console.log("ERROR while updating the sample",error);
    }}
```

Sample Properties				
Lims_number	Info_1	Provenance	Batch_nr	
1012-4	Luzerne	AGRIDEA	13	→ new sample inserted into the database Saved_sample_key = 160
1012-5	Null	LABO13		→ existing lims number Saved_sample_key = 10 Update sample information

Figure 16: Code example: Sample

```
catch (error){
  //the lims number doesn't exist, a new sample is inserted into the database;
  try{
    var sample_creation = BuilderSampleInsert(record);
    var p7 = await db.one(sample_creation);
    saved_sample_key = p7.sample_key;
  }
  catch(error){
    console.log("ERROR while trying to insert a new sample", error);
  }}
```

Listing 6: Handlerecord part 4

Leading example

In the figure 16, the first line contains a lims number that is not registered into the database. So the select statement, "*SELECT sample_key FROM d_sample WHERE lims_number = '1012-4'*", won't return a sample key. This implies that the insert statement, "*INSERT INTO d_sample(info_1, provenance, batch_nr) VALUES ('Luzerne','AGRIDEA', 13) RETURNING sample_key*", will be executed. The new sample will be saved under the sample key 160.
The second line of the figure 16 represents the fact where the lims number is already contained into the table d sample. The select statement will return a value of 10 for the sample key. As a sample key has been retrieved, the information must be updated, this is done by executing the following sql statement: "*UPDATE d_sample SET info_1 = null, provenance = 'LABO13' WHERE lims_number ='1012-5'*"

6.2.5 Nutrient and Nutrient Analysis

Finally the handlereport will start filling the fact_table_clean. For that, the function will consider each nutrient analysis that can be identified with a key containing % . An example of a nutrient analysis will be 120%2: 30. The nutrient key is 120, the nutrient anaylsis key is 2 and the analysed quantity is 30. The following will be executed for each key containing %.
At first, the function will split the key into two part in order to retrieve the nutrient key which will be saved in res[0] and the nutrient analysis key that will be saved in res[1]. Then the function will make sure that both keys are existing keys in the database by executing a select statement for each case. The select statement for the test of the existence of the nutrient key is defined by the BuilderNutrientResearch while the existence of the nutrient analysis key is defined by the BuilderAnalysisResearch. If those keys are retrieved, they will be saved into variables in order to be used later. If one of the key retrieval fails, the corresponding analysis won't be imported into the database. This can be seen in the listing 7. The star at the end of the listing 7 indicates that the rest of the function is shown in the listing 8.

```
//NUTRIENT AND NUTRIENT ANALYSIS
for (key in record){
  if (/%/.test(key)){
    saved_quantity = record[key];
    var str =key;
    var res = str.split(/%/);
    //in res[0] the nutrient key is saved, in res[1] the nutrient analysis key
     //Looking if the nutrient and nutrient analysis key are valid
     try{
      var nutrient_research = BuilderNutrientResearch(res[0]);
      var p8 = await db.one(nutrient_research);
      saved_nutrient = p8.nutrient_key;
      //Checking if the nutrient analysis key is valid
        try{
          var analysis_research = BuilderAnalysisResearch(res[1]);
          var p9 = await db.one(analysis_research);
          saved_nutrient_analyses = p9.nutrient_analyses_key
        }
        catch(error){
          error_report_analysis ="ERROR: The nutrient analysis key"+ res[1]+ "
    is not valid, so the related analysis won't be imported, excel line" +
    record.__rowNum__;
          continue;
        }
      }
      catch(error){
          error_report_analysis ="ERROR: The nutrient key"+ res[0]+ "is not
    valid, so the related analyis won't be imported, excel line" + record.
    __rowNum__;
          continue;
      }
    *******
```

Listing 7: Handlerecord part 5

If the nutrient and nutrient analyses keys are retrieved, data can be inserted into the fact_table_clean. Again a distinction must be made if the analysis is made on a sample that was already included into the database or if it was made on a new sample.

New sample

If the sample is new, the handlerecord executes the part of the listing that is represented at the line 29 of the listing 8. In this case, new analyses will be added into the fact table clean. The BuilderInsertFactTable will be called to provide the corresponding insert statement.

Overwrite an existing sample

If the sample has been overwritten, this implies that some analyses may already exist into the fact table clean. Therefore it is necessary to check if the given analysis already exists into the database. This check must be executed for each timestamp. As a quick remember, a single analysis occupies three lines in the fact_table_clean, therefore there will be three different measure pkey.

So for each timestamp, the BuilderFactTableResearch will provide a select statement: "*SELECT measure pkey FROM fact table clean WHERE id feed fkey = value AND id time fkey = value_i AND lims number = value and id origin key = value and id nutrient fkey = value AND id nutrient analyses fkey = value*". If a measure key is retrieved, the given analysis must be

overwritten. Therefore the BuilderAnalysisOverwrite is called. This Builder can either provide an upload or a delete sql statement. If the analysed quantity is null, the BuilderAnalysisOverwrite will return a delete statement. "*DELETE FROM fact table clean WHERE measurepkey = Value*". If a analysed quantity is provided, the BuilderAnalysisOverwrite will return an update statement.
It is also possible that new analysis are added on existing samples. In this case, the BuilderInsertFactTable will generate the corresponding insert statement.

```
    *******
     if(overwrite == true){
         var j;
      for (j=0; j< saved_time.length; j++){
        if(saved_time[j]!= undefined){
          try{
            var analysis_research = BuilderFactTableResearch(saved_feed_key, saved_time[j], record.lims_number, saved_origin, saved_nutrient, saved_nutrient_analyses);
            //retrieve the measure pkey
            var p10 = await db.one(analysis_research);
            var measure_pkey_saved = p10.measure_pkey;
              try{
                var analysis_overwrite = BuilderAnalysisOverwrite(measure_pkey_saved, saved_quantity);
                await db.none(analysis_overwrite);
              }catch(error){}
          }
          catch(error){
            //There is a new analysis that must be added.
            try{
              var insertion_final = BuilderInsertFactTable (saved_feed_key, saved_time[j], saved_sample_key, record.lims_number, saved_origin, saved_nutrient, saved_nutrient_analyses, saved_quantity);
              //console.log(insertion_final);
              var p10a = await db.one(insertion_final);
              var measurekey = p10a.measure_pkey;

            }catch(error){console.log(error);}
          }}}}

     else{
      var j;
      for (j=0; j< saved_time.length; j++){
        try{
          if(saved_time[j]!= undefined){
           var insertion_final = BuilderInsertFactTable (saved_feed_key, saved_time[j], saved_sample_key, record.lims_number, saved_origin, saved_nutrient, saved_nutrient_analyses, saved_quantity);
           //console.log(insertion_final);
           var p10 = await db.one(insertion_final);
           var measurekey = p10.measure_pkey;
          }
        }
        catch(error){
          console.log ("An error occured while inserting into fact table ", error);
        }}}}

  else{
```

Sample Properties			TVK	TSO	
Lims_number	Saved_ sample_key		180%2	144%7	
1012-4	160		13		→ insert it for each time key
1012-5	10	overwrite	150	null	→ update 180%2 and delete 144%7 for each time key

Figure 17: Code example: Analysis

```
    continue;
  }}
var success = "Line " + record.__rowNum__ + " has been successfully inserted \n
    ";
var success = error_report_analysis + success;
return success;
```

Listing 8: Handlerecord part 6

Leading example

The first line of the example in the figure 17 assumes that the sample with lims number 1012-4 is new, therefore the insert statement will be executed. Below is provided one insert statement that must be executed for the analysis 180%2. As each analysis can have up to three time information , there must be one insert statement pro time key for each nutrient analsis. A feed key of 1500, an origin key of 12, a time key of 150, a sample key of 160 are considered. The insert statement would be: "*INSERT INTO fact_table_clean(id_feed_fkey, id_origin, id_time_fkey, id_sample_fkey, lims_number, id_nutrient_fkey, id_nutrient_analyses_fkey, quantity') VALUES(' 1500, 12, 150, 160, '1012-4', 180, 2, 13) RETURNING measure_pkey'*".
The second lines aims to show the case where the sample with a sample key of 10 must be overwritten. In this case, the content of the analysis 180%2 was already into the database, therefore the quantity must be updated. Again there will be an update statement for each measure pkey corresponding to the analysis: "*UPDATE fact_table_clean SET quantity = 150 where measure_pkey = 1566;* ".
The last examples aims to delete all the values corresponding to the analysis 144%7. At first the corresponding measure pkey must be retrieved. Then for each of the three measure pkey the following statement will be executed: "*DELETE FROM fact_table_clean WHERE measure_key = value*"

6.3 End of the import

When the handlereport function has been executed for each report, the error report will be shown on the web browser (Figure 18). The goal of this error report is to help the user knows what happened. The idea is that the user copies the excel line that haven't been imported into a new excel file, corrects them and then tries to upload them again.

Following messages can be seen:

1. ERROR: in the excel file a feed key is missing on line X so this line wasn't imported.

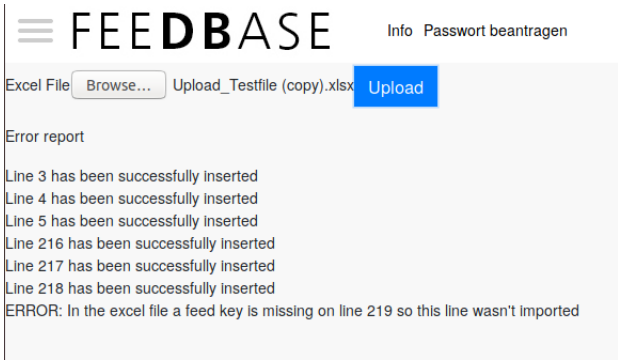


Figure 18: Error report

2. ERROR: No lims number was provided in the excel line X so this line wasn't imported.
3. ERROR: The feed key Y doesn't exist in the database, it must be added manually. The excel line X won't be imported.
4. ERROR: Wrong date format for the line X and the moment i. This line won't be imported.
5. ERROR: impossible to overwrite the sample given on line X. Possible error causes: apostrophe or space contained into a cell that must contain a numerical value. Please remove them. *Overwrite statement causing the issue*
6. ERROR: impossible to insert the sample given on line X. Possible error causes: apostrophe or space contained into a cell that must contain a numerical value. Please remove them. By looking at the following insert statement, you can see where if they are 2 commas in the VALUE part and find the corresponding header *Insert statement causing the issue*
7. ERROR: The nutrient analysis key Y is not valid so the related analysis won't be imported, excel line X
8. ERROR: The nutrient key Y is not valid so the related analysis won't be imported, excel line X
9. Line X inserted into the database.

The errors 1, 2, 3, 7 and 8 are easy to detect. The error 4 is harder to detect. This error can occurs if the user used the wrong date format in excel : he used DD/MM/YYYY instead of DD.MM.YYYY. This error can also happen if the cell is not in a date format or if the cell day_i is not empty e.g it contains a space. Before trying to reimport this line, the user should check the above mentionned cases. The first thing he should do, is select all the empty cells day_i and click on delete to make sure that they are really empty.
The errors 5 and 6 have similar causes. This can occurs if the character ' is contained in one of the excel cells. This character is a special character used into the SQL statements. So if the user writes the character ', this will implies that the corresponding SQL statements will be wrong. So at first the user should open the excel file, click "Ctrl + F" and search for ' and replace it by a space. If this doesn't work, he can delete the content of all the empty cell to make sure that no blank space is contained in them, as this is the second source of errors.
For each line that have been correctly inserted, "Line X inserted into the screen" will be shown. Once the error report appears on the webpage, it implies that the content of the excel file has

been introduced into the database. However the new entries are still not accessible on the web application. In order to push the newly added data on the database, the maintenance step must be executed.

6.4 Maintenance

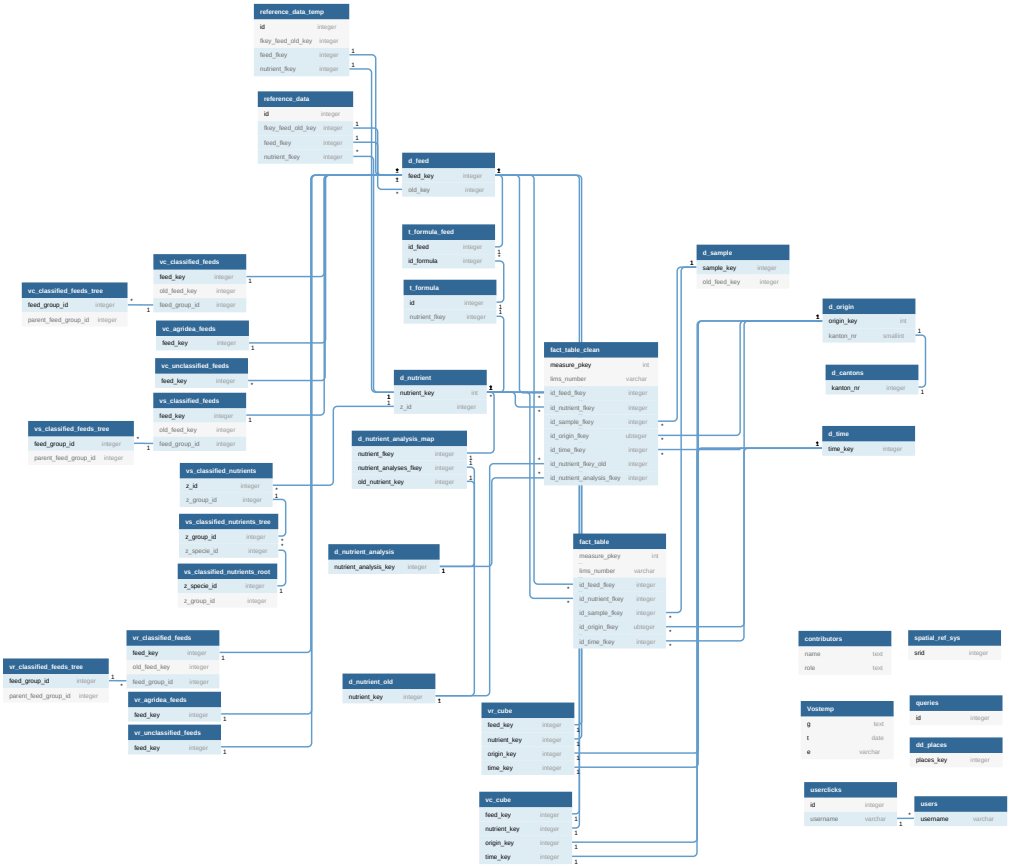
The maintenance step aims to delete old content into the views of the database, insert the newly added data, to reindex the table and to recompute statistics. This step is mainly executed by executing the SQL statement: Select * from maintenance_global().
As this step is time and ressource consuming, it has been decided that the user should execute this sql statement manually after he imported the data. It can be judicious to execute this maintenance step at night or outside of the working hours.

7 Conclusion and remaining issues

The detailled functioning of the application has been explained. At first, the new import file has been described in details using many different example. The goal of this explanation was to make sure that the user knows exactly how to fulfill the file. He must at least provide two information: a feed key and a lims number and must especially pay attention not the use the character '. Then the source code of the application has been explained. Basically the application goes through each line of the excel file. At first it retrieves the feed key, then take care of the origin information. After that, each timestamp is considered and in total three time keys are generated. One advantage in the management of the origin and time information is that the application can handle the fact where no information is provided. Then the sample information were considered. The application is able to handle new and existing samples. If the sample is new, it will be inserted into the database. If on contrario the sample already exists, the sample information will be overwritten. Finally the nutrient analysis are considered. This part is the most complicated part of the import application as there must be a distinction based on the fact that nutrient analysis are made on a new or on a existing sample. If the sample is new, all the nutrient analysis will be inserted into the database. It must be remembered that each nutrient analysis must be inserted into the database up to three times: there is a new entry for each nutrient analysis and for each provided timestamp. On the other hand, if the analysis were made on an existing sample, some analysis may be modified, added or even deleted. After a discussion with Ms. Bracher, it has been found out that the deletion of analysis is a possible case even if it happens rarely. However this special case is still considered by the import application. Once the whole file has been treated the error report will be shown to the user. This is a signal that the execution of the import application is complete. Then the user must manually execute the maintenance step. Regarding the given time, it was more important to provide a working import application as the old one was slowly getting obsolete. Therefore, the focus was put on the import part rather than on the maintenance part.
One of the main programming issue was the fact that Javascript could be considered as an asynchronous language. Trials have been done to create a progress bar in addition to the error report. This comes out to be a way more complicated than expected. At the end, it has been decided to focus only on what was more important for the user. Therefore, only errors were reported. It is actually more helpful for the user to get a feedback about what has functioned and what hasn't instead of only getting a progress bar. However, a further developpment of this import application could include a progress bar in addition to the error report.

Regarding the errors that can occur, the fact that the character ' is not allowed can lead to many errors if the user has forgotten not to use it. Actually all the sql statements are made of string concatenation: so if a character ' is found, the functions will be provide a wrong sql statement especially in the insert statement. This point and the fact that the date format must be really precise is one of the issue that could go against the robustness of the application. Considering the ease of use, the webpage is simple to use as only one action is required from the user: select the excel file to upload. The detailled report are also an interesting way to help him understand what has been going on. Moreover inserting around 200 excel lines into the database will take maximally one minute. So the application works well even if a large amount of data is provided into the excel file.

Appendices


reference_data_temp
id integer
fkey_feed_old_key integer
feed_fkey integer
nutrient_fkey integer
reference_data
id integer
fkey_feed_old_key integer
feed_fkey integer
nutrient_fkey integer
d_feed
feed_key integer
old_key integer
t_formula_feed
id_feed integer
id_formula integer
t_formula
id integer
nutrient_fkey integer
d_sample
sample_key integer
old_feed_key integer
d_origin
origin_key int
kanton_nr smallint
d_cantons
kanton_nr integer
d_time
time_key integer
vc_classified_feeds
feed_key integer
old_feed_key integer
feed_group_id integer
vc_classified_feeds_tree
feed_group_id integer
parent_feed_group_id integer
vc_agridea_feeds
feed_key integer
vc_unclassified_feeds
feed_key integer
vs_classified_feeds
feed_key integer
old_feed_key integer
feed_group_id integer
vs_classified_feeds_tree
feed_group_id integer
parent_feed_group_id integer
d_nutrient
nutrient_key int
z_id integer
d_nutrient_analysis_map
nutrient_fkey integer
nutrient_analyses_fkey integer
old_nutrient_key integer
fact_table_clean
measure_pkey int
lims_number varchar
id_feed_fkey integer
id_nutrient_fkey integer
id_sample_fkey integer
id_origin_fkey ubtiger
id_time_fkey integer
id_nutrient_fkey_old integer
id_nutrient_analysis_fkey integer
vs_classified_nutrients
z_id integer
z_group_id integer
vs_classified_nutrients_tree
z_group_id integer
z_specie_id integer
vs_classified_nutrients_root
z_specie_id integer
z_group_id integer
d_nutrient_analysis
nutrient_analysis_key integer
fact_table
measure_pkey int
lims_number varchar
id_feed_fkey integer
id_nutrient_fkey integer
id_sample_fkey integer
id_origin_fkey ubtiger
id_time_fkey integer
vr_classified_feeds
feed_key integer
old_feed_key integer
feed_group_id integer
vr_classified_feeds_tree
feed_group_id integer
parent_feed_group_id integer
vr_agridea_feeds
feed_key integer
vr_unclassified_feeds
feed_key integer
d_nutrient_old
nutrient_key integer
vr_cube
feed_key integer
nutrient_key integer
origin_key integer
time_key integer
vc_cube
feed_key integer
nutrient_key integer
origin_key integer
time_key integer
contributors
name text
role text
spatial_ref_sys
srid integer
Vostemp
g text
t date
e varchar
queries
id integer
dd_places
places_key integer
userclicks
id integer
username varchar
users
username varchar