Department of Informatics, University of Zurich

Implementing Grouping Factors in Nutrient Statistics

Self-study project

June 30, 2015

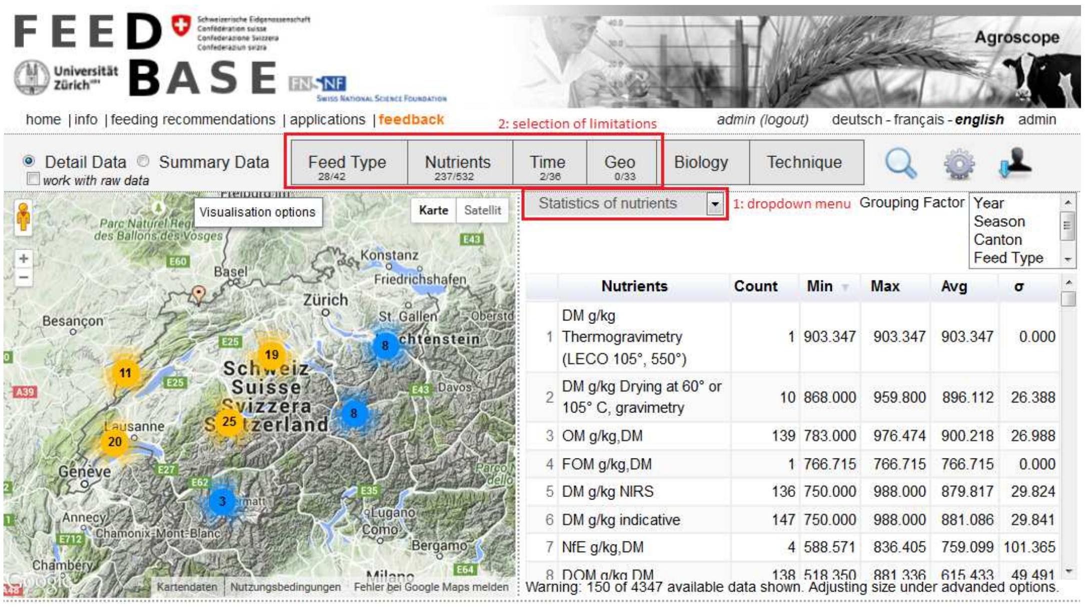
Simon Rüegg, 08-609-844

simon.rueegg@uzh.ch

Supervisor: Francesco Cafagna

1 Introduction

The Swiss Feed Database[1] contains measurements of nutrients in animal feeds. Users can access the database and find out which nutrients are contained in which feed types. If there exist measurements of a certain nutrient in a certain feed type, it is also possible to see how many measurements there are, what the minimum, maximum, and average amount (typically measured in g/kg) of the nutrient are and what the standard deviation of the measurements is. Figure 1 is a screenshot of the website and on the right hand side, the statistics of the nutrients are shown in a table. These statistics can be accessed by clicking on “Statistics of nutrients” in the dropdown menu (red box Nr. 1 in figure 1). Other options are “scatter chart”, “one-to-many regions comparison”, and “correlated nutrients”. The third row in the table basically says that there exist 139 measurements of the nutrient OM (measured in g/kg on a dried matter basis), the minimum value is 783, the maximum value is 976.474 the average is 900.218 and the standard deviation is 26.988. In the red box Nr. 2, the users can select feed types, nutrients, years, seasons, cantons or altitude of the measurements on which to compute the statistics. In the example on figure 1, 28 feed types, 237 nutrients, and 2 years (2013 and 2014) were chosen. There are no further limitations based on season, canton, or altitude. This example will be used throughout this report.

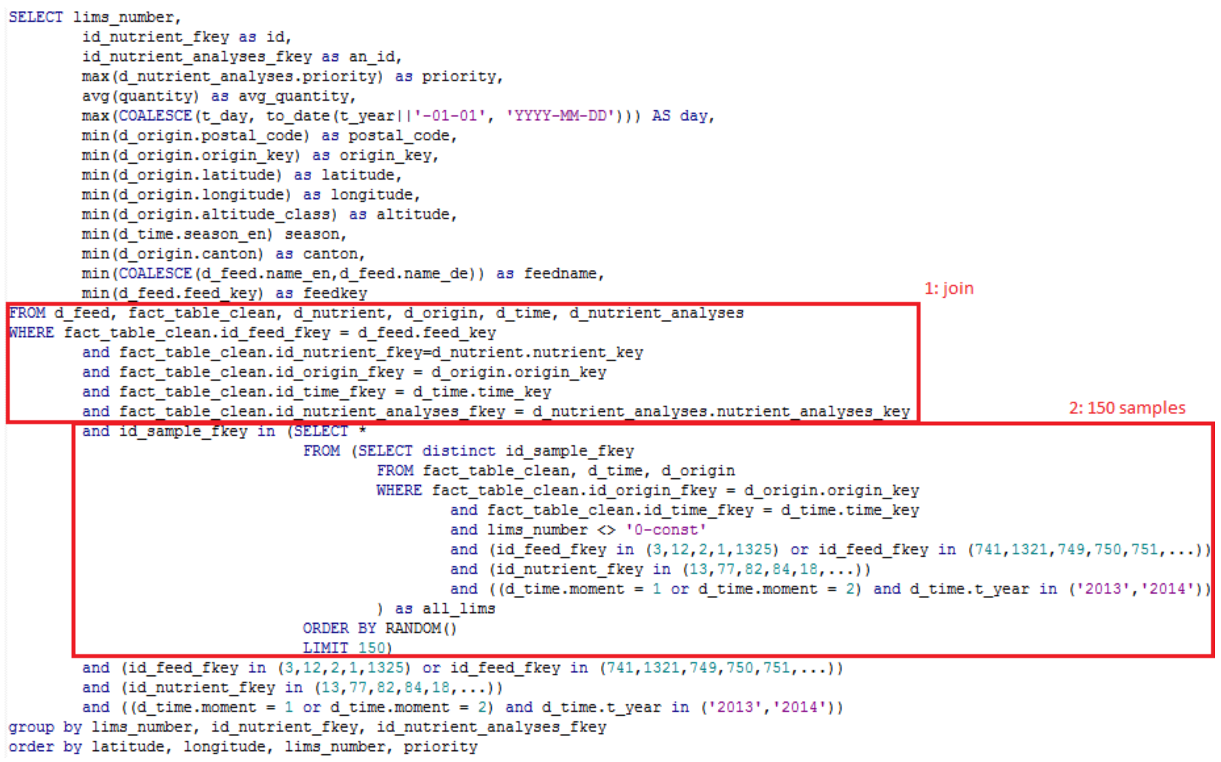


	Nutrients	Count	Min	Max	Avg	σ
1	DM g/kg Thermogravimetry (LECO 105°, 550°)	1	903.347	903.347	903.347	0.000
2	DM g/kg Drying at 60° or 105° C, gravimetry	10	868.000	959.800	896.112	26.388
3	OM g/kg,DM	139	783.000	976.474	900.218	26.988
4	FOM g/kg,DM	1	766.715	766.715	766.715	0.000
5	DM g/kg NIRS	136	750.000	988.000	879.817	29.824
6	DM g/kg indicative	147	750.000	988.000	881.086	29.841
7	NfE g/kg,DM	4	588.571	836.405	759.099	101.365
8	DOM g/kg DM	138	518.350	881.336	615.433	49.491

Warning: 150 of 4347 available data shown. Adjusting size under advanded options.

Figure 1: Website of Feed Base with an example Statistics of nutrients

[1] http://www.feed-alp.admin.ch/index.php (accessed on 2015-06-24)



```sql
SELECT lims_number,
       id_nutrient_fkey as id,
       id_nutrient_analyses_fkey as an_id,
       max(d_nutrient_analyses.priority) as priority,
       avg(quantity) as avg_quantity,
       max(COALESCE(t_day, to_date(t_year||'-01-01', 'YYYY-MM-DD'))) AS day,
       min(d_origin.postal_code) as postal_code,
       min(d_origin.origin_key) as origin_key,
       min(d_origin.latitude) as latitude,
       min(d_origin.longitude) as longitude,
       min(d_origin.altitude_class) as altitude,
       min(d_time.season_en) season,
       min(d_origin.canton) as canton,
       min(COALESCE(d_feed.name_en,d_feed.name_de)) as feedname,
       min(d_feed.feed_key) as feedkey
FROM d_feed, fact_table_clean, d_nutrient, d_origin, d_time, d_nutrient_analyses
WHERE fact_table_clean.id_feed_fkey = d_feed.feed_key
       and fact_table_clean.id_nutrient_fkey=d_nutrient.nutrient_key
       and fact_table_clean.id_origin_fkey = d_origin.origin_key
       and fact_table_clean.id_time_fkey = d_time.time_key
       and fact_table_clean.id_nutrient_analyses_fkey = d_nutrient_analyses.nutrient_analyses_key
       and id_sample_fkey in (SELECT *
                             FROM (SELECT distinct id_sample_fkey
                                   FROM fact_table_clean, d_time, d_origin
                                   WHERE fact_table_clean.id_origin_fkey = d_origin.origin_key
                                         and fact_table_clean.id_time_fkey = d_time.time_key
                                         and lims_number <> '0-const'
                                         and (id_feed_fkey in (3,12,2,1,1325) or id_feed_fkey in (741,1321,749,750,751,...))
                                         and (id_nutrient_fkey in (13,77,82,84,18,...))
                                         and ((d_time.moment = 1 or d_time.moment = 2) and d_time.t_year in ('2013','2014'))
                                   ) as all_lims
                             ORDER BY RANDOM()
                             LIMIT 150)
       and (id_feed_fkey in (3,12,2,1,1325) or id_feed_fkey in (741,1321,749,750,751,...))
       and (id_nutrient_fkey in (13,77,82,84,18,...))
       and ((d_time.moment = 1 or d_time.moment = 2) and d_time.t_year in ('2013','2014'))
group by lims_number, id_nutrient_fkey, id_nutrient_analyses_fkey
order by latitude, longitude, lims_number, priority
```

Figure 2: Query to retrieve the measurements for the selected feed types, nutrients, and years

The underlying data is stored in a PostgreSQL database and accessed via the php-functions *pg_connect* and *pg_query*[2]. The query used for extracting all the relevant measurements for the chosen nutrients (13,77,82,84,18,…), feed types (3,12,2,1,1325,…), years (‘2013’,’2014’), seasons, cantons, and altitudes is shown in figure 2. The FROM and the WHERE clauses (can be seen in red box Nr. 1) are used to join all the tables that are used. Red box Nr. 2 represents the sampling process which limits the number of final results. The standard value for that limitation is 150 and is set in place in order to avoid queries that take too long. In that case, a warning is placed under the table (can be seen on figure 1) to inform the user that not all the available data is shown. The user has the possibility to change the number of samples, but has to be aware that a change could make the completion time of the query longer. Following the 150 samples, the rest of the WHERE clause is used to retrieve the data that the user requested. In this example, that part has the selections of nutrients (13,77,82,84,18,…), feed types (3,12,2,1,1325,…) and years (‘2013’,’2014’). In figure 2, not all ids of the chosen nutrients and feeds are written due to lack of space.

The result of the query is stored in a php variable called *query_result* and then converted into a JSON string which then can be used as parameter to create a *google.visualization.DataTable*, which can be presented in HTML.

[2] http://php.net/manual/en/ref.pgsql.php (accessed on 2015-06-20)

The goal of this self-study project is the addition of grouping factors in nutrient statistics. It should become possible to display nutrient statistics grouped by year, season, canton, feed type, or altitude. This should enable users to learn more from the displayed data. For example, after grouping by year, it becomes possible to observe the development of the measurements over time. Additionally, the choosing of multiple grouping factors together should be implemented, which further expands the possibilities of the users. The algorithm provided should be scalable and fast without any size-constraint.

The second chapter of this report contains descriptions of three solutions that were implemented. It is followed by chapter 3 which contains the detailed changes that were made on the source code and a conclusion in chapter 4.

2 Solutions

In the process of solving the above mentioned problem, three different solutions were implemented: *Nested Loop*, *Sort and Aggregate*, and *Materialized Pre-Aggregates*. These solutions will be described in this chapter. While the first two solutions are applied on the results of the already existing query from figure 2, the third solution uses a newly constructed SQL-query which is directly applied to the database and has no size restriction.

Table 1 shows an excerpt of the above mentioned example (28 feed types, 237 nutrients, and 2 years chosen) after grouping it by year and feed type. Two new columns are introduced, one for year, and the other for feed type. Count, minimum, maximum, average, and standard deviation are newly calculated for every distinct combination of *Nutrients*, *Year*, and *Feed Type*. The white columns represent the groups and the grey columns represent the aggregation functions.

Nutrients	Year	Feed Type	Count	Min	Max	Avg	σ
ADF g/kg,DM indi...	2013	Hay 1. cut	4	272	300.86	287.965	11.027
ADL g/kg,DM indi...	2013	Hay 1. cut	3	20	29	25.667	4.028
Ash g/kg,DM Ther...	2013	Hay 1. cut	1	66.59	66.59	66.59	0
Ash g/kg,DM indi...	2013	Hay 1. cut	12	66.59	136	95.216	20.256
CF g/kg,DM indi...	2013	Hay 1. cut	12	245	312	278.802	20.797

Table 1: Part of statistics table after applying grouping on year and feed type

2.1 Nested Loop

The first solution starts with collecting all the existing values of every grouping factor that has been chosen by the users from the variable *query_result* (the result of the query from figure 2*)*. If the users wish to group the result on year and feed type, all the years and all the feed types that appear in *query_result* are collected and stored in arrays: *years[]* and *feeds[]*.

After that first step, the grouping starts by building all possible combinations of the collected instances and the nutrients. For example, if the user chose grouping by nutrient, year and feed type, all possible combinations of *(nutrient, year, feed type)* are constructed. This is accomplished by running through all the nutrients, *years[]* and *feeds[]* with a nested loop. For every possible combination, *query_result* is then scanned and if an entry matches the current combination, it is added to the calculation process that leads to *Count*, *Min*, *Max*, *Avg*, and *σ.*

With the implementation of this solution, it becomes possible to display the results after applying one or more of the grouping factors (year, season, canton, feed, and altitude) in the style of table 1. However, this solution turned out to be very inefficient. If there are several feed types and nutrients selected and the grouping by all five grouping factors requested, the time to complete the calculations and displaying the results of the grouping procedure was far too long and unlikely to be accepted by the users. That has to do with the fact that the whole *query_result* has to be processed entirely for every possible combination of nutrients and the chosen grouping factors. If any of these increases, the resulting completion time increases quadratically. This solution has been the first to be implemented, but has been replaced by the second solution, called *Sort and Aggregate*.

2.2 Sort and Aggregate

To enable the calculation to be more efficient, a new algorithm had to be implemented. This solution is the one used when the users click on "Statistics of nutrients". The result-table is sorted by all the chosen grouping factors, e.g. by *(nutrient, year, feed type)*. That way, scanning the table several times becomes obsolete. It is just necessary to start the scanning and collect values for the calculation of *Count*, *Min*, *Max*, *Avg*, and *σ* and to proceed until a new combination of nutrient and chosen grouping factors appears. When a new combination is reached, a new row can be added to the resulting table. For the calculation of the standard deviation (σ), all the values of the current group have to be collected, stored in an array, and formula 1 has to be applied.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{X})^2}{N}}$$

Formula 1: Standard Deviation

This process is repeated until all rows of the table are processed. *Query_result* just has to be scanned once and the completion time for large selections of nutrients and feed types can be decreased (n log n for sorting).

Figure 3 shows a comparison of the two solutions *Nested Loop* and *Sort and Aggregate*. The chosen nutrients, feeds, and years are the same as before (nutrients (13,77,82,84,18,…), feed types (3,12,2,1,1325,…), years ('2013','2014')). In addition, the sample size, that has a standard value of 150, was increased to get all the 4347 results. After selecting *year* and *feed type* as grouping factors, the completion of *Nested Loop* took 80 seconds, and the completion of *Sort and Aggregate* took 5 seconds. Since both solutions create the same result and *Sort and Aggregate* is much faster, *Nested Loop* was removed from the website and replaced by *Sort and Aggregate*.

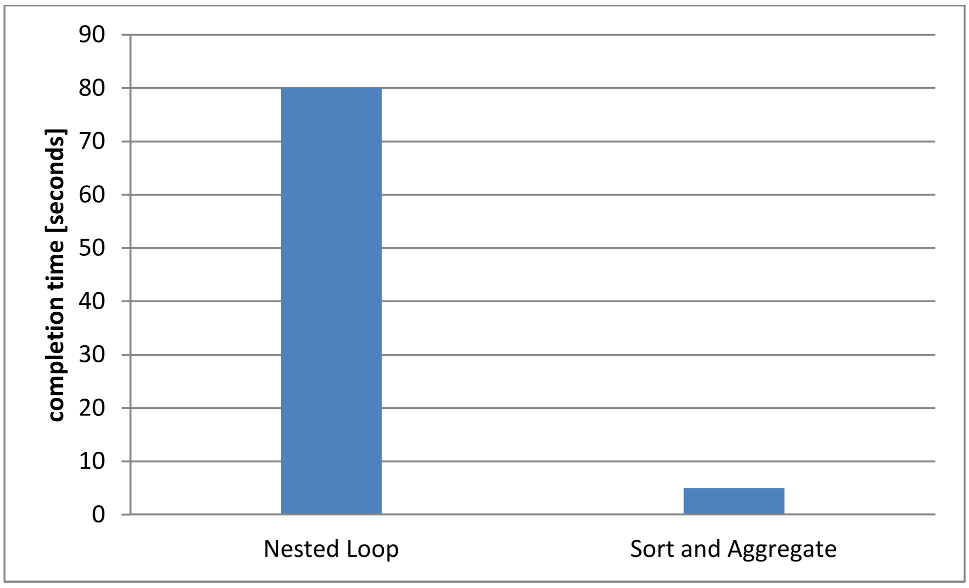
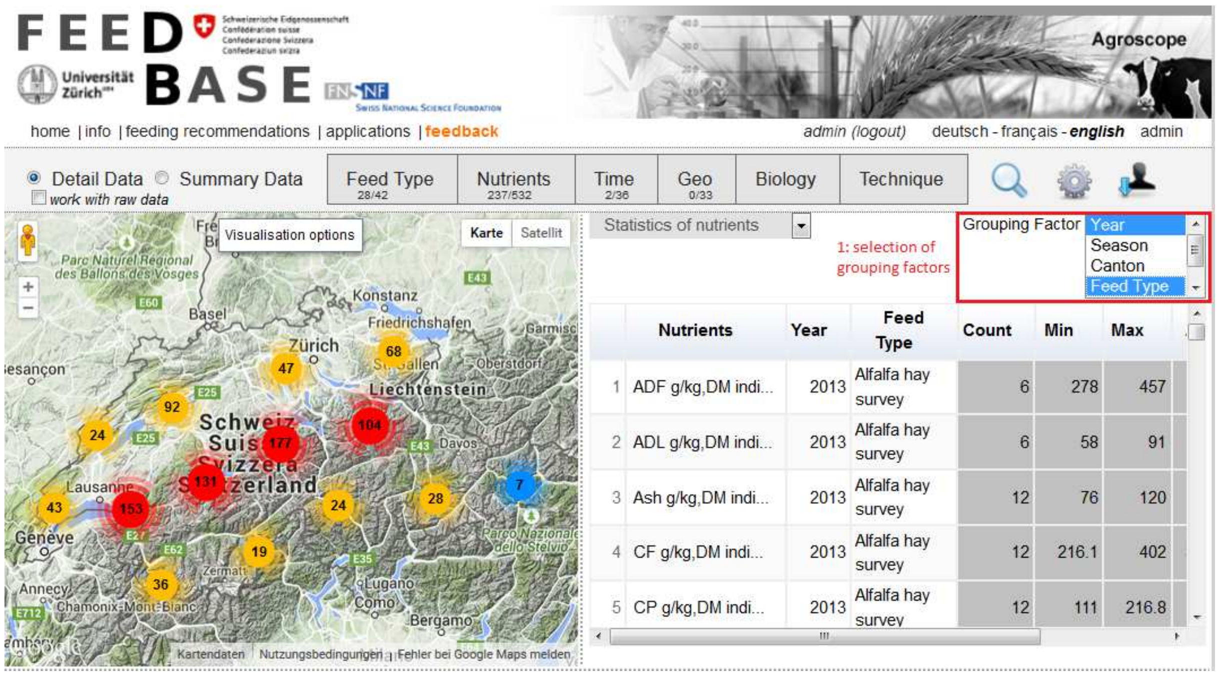


Figure 3: Comparison of Nested Loop and Sort and Aggregate in terms of completion time

Figure 4 represents a screenshot of the website after grouping on year and feed type (the applied solution is *Sort and Aggregate*). Red box Nr. 1 shows a dropdown menu, which was introduced for the selection of the grouping factors. Selections of any combination of the grouping factors are possible. On figure 4, the options year and feed type are enabled, which means that grouping is done on year and feed type.



Grouping Factor: [✓ Year] Season Canton [✓ Feed Type]

	Nutrients	Year	Feed Type	Count	Min	Max
1	ADF g/kg,DM indi...	2013	Alfalfa hay survey	6	278	457
2	ADL g/kg,DM indi...	2013	Alfalfa hay survey	6	58	91
3	Ash g/kg,DM indi...	2013	Alfalfa hay survey	12	76	120
4	CF g/kg,DM indi...	2013	Alfalfa hay survey	12	216.1	402
5	CP g/kg,DM indi...	2013	Alfalfa hay survey	12	111	216.8

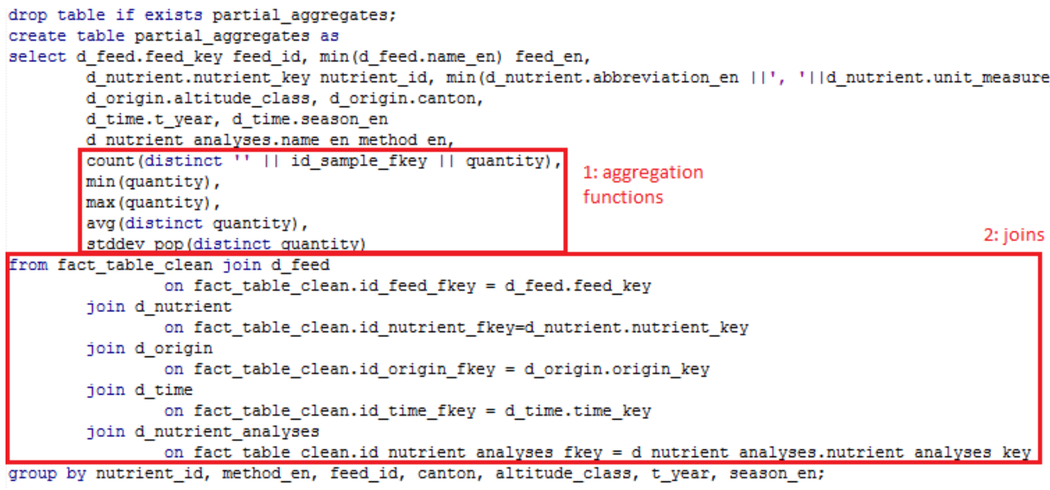
Figure 4: Screenshot of the website after grouping on year and feed type

Sort and Aggregate calculates *Count*, *Min*, *Max*, *Avg*, and *σ* based on the result of the query from figure 2. The limitation of the sample size, that has a standard value of 150, is restricting the significance of the result. That means that if the user does not increase the sample size to the number of available measurements, the results at the end are just based on 150 random samples and not on the whole data that is available.

2.3 Materialized Pre-Aggregates

In the third solution, *Materialized Pre-Aggregates*, it was a goal to come up with a new algorithm that is not dependent on the query from before (figure 2) and therefore does not have the problem of just having sampled entries.

The idea is to pre-compute partial aggregates. Considering the finest possible grouping factor set (*nutrient*, *year*, *season*, *canton*, *feed type*, *altitude*), the aggregates (*Count*, *Min*, *Max*, *Avg*, and *σ*) are calculated and placed in a new table called *partial_aggregates*. In other words, the aggregates of the finest possible grouping procedure are materialized in a table and ready to be fetched without having to be calculated. The query to create that table is shown in figure 5 and a little excerpt of the table can be seen on table 2.



```
drop table if exists partial_aggregates;
create table partial_aggregates as
select d_feed.feed_key feed_id, min(d_feed.name_en) feed_en,
        d_nutrient.nutrient_key nutrient_id, min(d_nutrient.abbreviation_en ||', '||d_nutrient.unit_measure_en) nutrient_en
        d_origin.altitude_class, d_origin.canton,
        d_time.t_year, d_time.season_en
        d_nutrient_analyses.name_en method_en,
        count(distinct '' || id_sample_fkey || quantity),
        min(quantity),
        max(quantity),
        avg(distinct quantity),
        stddev_pop(distinct quantity)
from fact_table_clean join d_feed
                on fact_table_clean.id_feed_fkey = d_feed.feed_key
        join d_nutrient
                on fact_table_clean.id_nutrient_fkey=d_nutrient.nutrient_key
        join d_origin
                on fact_table_clean.id_origin_fkey = d_origin.origin_key
        join d_time
                on fact_table_clean.id_time_fkey = d_time.time_key
        join d_nutrient_analyses
                on fact_table_clean.id_nutrient_analyses_fkey = d_nutrient_analyses.nutrient_analyses_key
group by nutrient_id, method_en, feed_id, canton, altitude_class, t_year, season_en;
```

Figure 5: Query to create table partial_aggregates

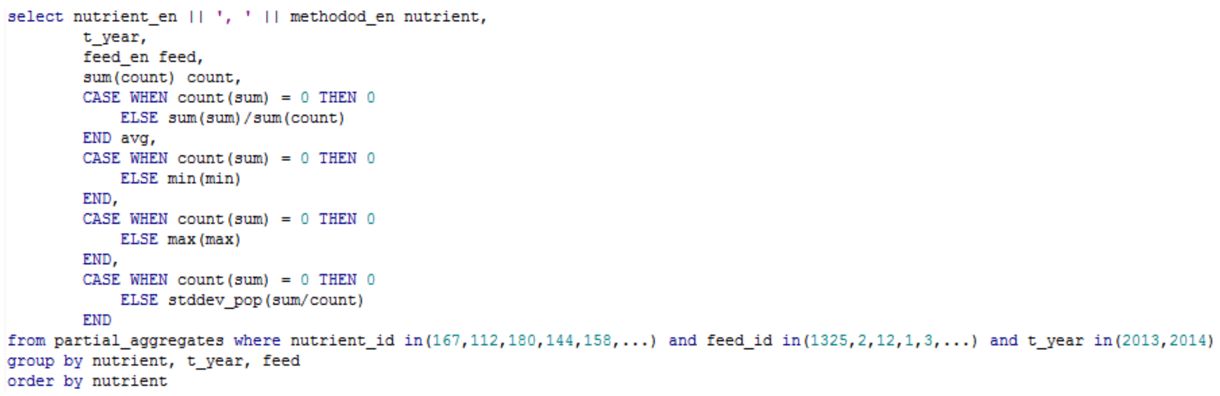
Calculations are made in the *SELECT* clause with the aggregate functions *count(), min(), max(), avg(),* and *stddev_pop()* (red box Nr. 1). The query has the same join functions (red box Nr. 2) as the query from figure 2, but not the sampling procedure. It further contains a *GROUP BY* clause that is used to form the groups.

nutrient	**t_year**	**season**	**canton**	**feed**	**altitude**	**count**	**min**	**Max**	**avg**	**stddev**
ADF, g/kg DM	2003		Genève	Horse bean	< 600	2	111.79493	141.84	126.817465	15.022535
ADF, g/kg DM	2004		Genève	Horse bean	< 600	2	107.72701	142.30052	125.013765	17.286755
ADF, g/kg DM	2008	Autumn	Genève	Horse bean	< 600	4	107.72701	142.30052	125.915615	16.21935762
ADF, g/kg DM	2003		Vaud	Horse bean	< 600	1	153.77757	153.77757	153.77757	0
ADF, g/kg DM	2008	Autumn	Vaud	Horse bean	< 600	1	153.77757	153.77757	153.77757	0

Table 2: Excerpt of the new table partial_aggregates

The new table is called *partial_aggregates* and represents the statistics table showing up on the website when all the possible grouping factors are chosen. While the white columns stand for the groups, the grey columns stand for the aggregation functions. The first row can be read as: “There are 2 measurements for nutrient “ADF” in year “2003”, with unknown season, from canton “Genève”, from feed type “Horse bean” and collected at an altitude that is lower than “600 meters”. The minimum value of those two measurements is 111.79493, the maximum value is 141.84, the average is 126.817465 and the standard deviation is 15.022535”.

To generate other groupings (instead of the grouping by *(nutrient, year, season, canton, feed, altitude)* that is shown in table 2), queries consisting of the chosen grouping factors in the *GROUP BY* clause are applied to the table *partial_aggregates*. An example query that does a grouping by year and feed type is shown on figure 6. The conditional expressions (*CASE*) had to be included to prevent the results from being null. In the *WHERE* clause are the selections of the users in terms of the nutrients, feed types, years, seasons, cantons, and altitudes they are interested in. The selection of the nutrients, feed types, and years are the same as in the other examples throughout this report. The chosen grouping factors are (*nutrient, year, feed type*), that is why these factors appear in the GROUP BY clause. *Method* is a field that is used for building the name of the nutrients. It is added to nutrient by string-concatenation in the SELECT clause.



```sql
select nutrient_en || ', ' || methodod_en nutrient,
        t_year,
        feed_en feed,
        sum(count) count,
        CASE WHEN count(sum) = 0 THEN 0
            ELSE sum(sum)/sum(count)
        END avg,
        CASE WHEN count(sum) = 0 THEN 0
            ELSE min(min)
        END,
        CASE WHEN count(sum) = 0 THEN 0
            ELSE max(max)
        END,
        CASE WHEN count(sum) = 0 THEN 0
            ELSE stddev_pop(sum/count)
        END
from partial_aggregates where nutrient_id in(167,112,180,144,158,...) and feed_id in(1325,2,12,1,3,...) and t_year in(2013,2014)
group by nutrient, t_year, feed
order by nutrient
```

Figure 6: Query on partial_aggregates to group it by year and feed

Count, *Min*, *Max*, and *Avg* are calculated precisely, but σ is computed from averages instead of the real values. Therefore, the solution *Materialized Pre-Aggregates* creates an approximated value for the standard deviation, while it provides an exact result for all other aggregate-functions.

The completion time of *Materialized Pre-Aggregates* in comparison to *Sort and Aggregate* was analyzed for two examples. The first example is the same that has been used in the whole report, and the second example has the same selection of nutrients and feed types, but considers in addition also the years 2010, 2011, and 2012. The first example has 4743 measurements, and the second example has 10161 measurements. *Materialized Pre-Aggregates* takes 2 seconds to complete in example 1 and 7 seconds in example 2. *Sort and Aggregate* takes 5 seconds in example 1 and 17 seconds in example 2. The comparison is visualized on figure 7.

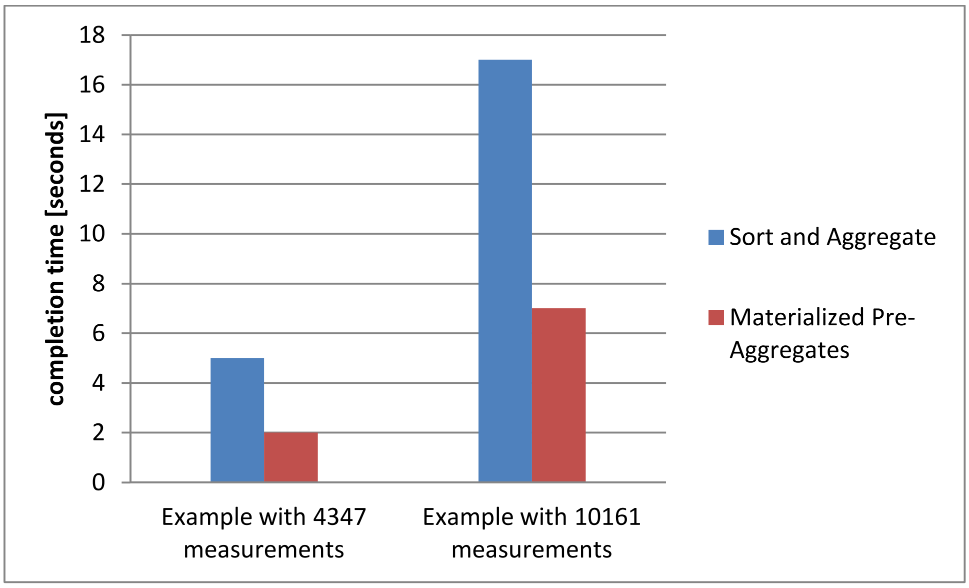
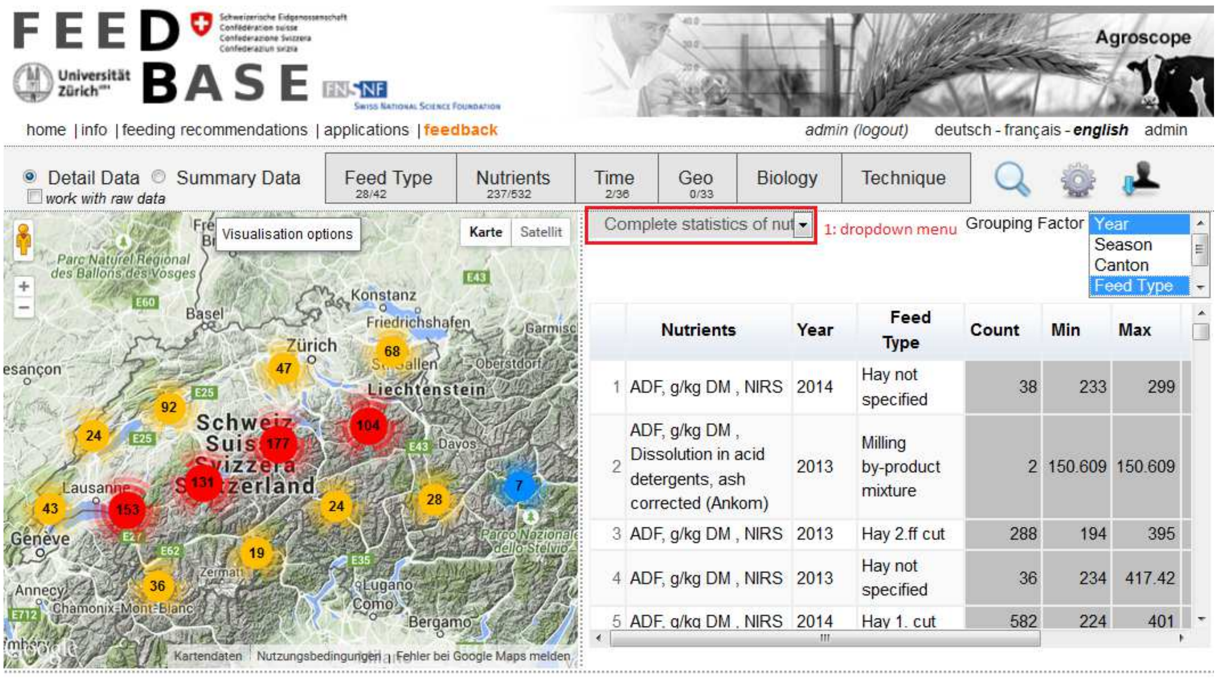


Figure 7: Comparison of S.a.A and M.P.-A. in terms of completion time

Materialized Pre-Aggregates is faster than *Sort and Aggregate* and is, in addition, also not dependent on the query from figure 2. That means that the results are never just calculated from 150 samples. In addition to that, any way of avoiding the query from figure 2 can be seen as an advantage if it is possible to get the same amount of useful pieces of information by fetching a much smaller dataset (the pre-aggregates) from the database.

Figure 8 shows a screenshot of the website when *Materialized Pre-Aggregates* is active and the same example of nutrient, feed type, and year selection, as well as grouping by *(nutrient, year, feed type)* is chosen. It is accessible by choosing "Complete nutrient statistics" in the dropdown menu (red box Nr. 1).



Complete statistics of nut

	Nutrients	Year	Feed Type	Count	Min	Max
1	ADF, g/kg DM , NIRS	2014	Hay not specified	38	233	299
2	ADF, g/kg DM , Dissolution in acid detergents, ash corrected (Ankom)	2013	Milling by-product mixture	2	150.609	150.609
3	ADF, g/kg DM , NIRS	2013	Hay 2.ff cut	288	194	395
4	ADF, g/kg DM , NIRS	2013	Hay not specified	36	234	417.42
5	ADF, g/kg DM , NIRS	2014	Hay 1. cut	582	224	401

Figure 8: Screenshot of the website after grouping on year and feed type with M.P.-A.

3 Changes to the Source Code

In the process of implementing the grouping factors, some files of the source code were edited. This chapter represents a short description of all the changes made.

index.php

Description: Contains the HTML content of the website.

Change 1: Introduction of a new <option> in the div “dd_stat_type”. That is the dropdown menu that allows the users to choose the type of functionality (“scatter chart”, “Statistics of nutrients”, “one-to-many regions comparison”, “Correlated nutrients”) they want to be displayed. The new option has the name “Complete statistics of nutrients” in the English version of the website and shows the statistic table that is generated by *Materialized Pre-Aggregates*. It is called “complete” because the statistics are always generated on all the available data, and not on a sample, which is the case on “Statistics of nutrients”.

Change 2: A new div with the name “dd_grouping_factor_select” was added. It is the dropdown menu that enables the user to choose which grouping factors they want to be applied on the statistics.

Change 3: The height of the div with the name “dd_stats_nutrients” had to be decreased that there is enough space for the newly created dropdown menu (“dd_grouping_factor_select”) to be placed above it.

Change 4: The variables “message_year”, “message_season”, “message_canton”, “message_feed”, and “message_altitude” were added to enable the new column names of the statistics table to be displayed in English, German, or French according to the language currently selected.

output_results_12.js

Description: All the processing steps for the different functionalities are completed here.

Change 1: For all the possible selections of the div “dd_stat_type” from index.php, it had to be determined whether the newly created div, “dd_grouping_factor_select” is displayed or not.

Change 2: *Sort and Aggregate* was implemented and it had to be defined that it is called whenever “Statistics of nutrients” was chosen on the dropdown menu “dd_stat_type”.

Change 3: *Materialized Pre-Aggregates* was implemented and it had to be defined that it is called whenever “Complete statistics of nutrients” was chosen on “dd_stat_type”.

Change 4: The function *clearSelected()* was implemented to reset the current selection of grouping factors if the users select a new option on “dd_stat_type”.

ajax_samples.php

Description: In this file, the query from figure 2 is created according to the selections from the users. Then, a connection to the database is created and the query is sent via the php function *pg_query()*. The returned data of that query is converted into a JSON string and sent to the caller (inside of output_results-12.js). It can then be used to create a *google.visualization.DataTable*, which can be shown on the website.

Change: Season and altitude of the measurements was added to the *SELECT* clause, because they are later needed by *Sort and Aggregate*.

partial_aggregates.php

Description: This file was newly created and is used to create a query, connect to the database, apply the query to the database, and convert its result into a JSON string and send it back. The query concerned here is the one from figure 6 (the query that is applied to the new table *partial_aggregates* which is used for *Materialized Pre-Aggregates*).

Change: Newly created file.

terms_en.php, terms_de.php, terms_fr.php

Description: All the used words on the website are saved for the corresponding language. The website calls the file according to the language selected by the user.

Change: The terms “Grouping Factor” and “Complete statistics of nutrients” were added in English, German, and French.

4 Conclusions

Three solutions to the problem of implementing grouping factors to an already existing application of nutrient statistics were implemented and described. The first solution, *Nested Loop* needed the results to be processed multiple times and therefore performed really slowly. It was replaced by a second solution, *Sort and Aggregate*, which used sorting and made it possible to reach the same results by just having to process the results once. The third solution, *Materialized Pre-Aggregates*, is not processing its calculations locally, but does it with a PostgreSQL-query and its *GROUP BY* clause. Therefore, it is not dependent on another query, is always based on all the available data (not just a sample), and performs faster if many nutrients and feed types are selected by the user.

List of Figures

List of Tables

List of Formulas