

Department of Informatics, University of Zürich

Facharbeit

Development of a Database System based on geographical information

Kristin Kruse

Zürich, Switzerland

Matrikelnummer: 04-720-983

Email: kristin.kruse@uzh.ch

August 31, 2011

supervised by Prof. Dr. M. Böhlen and F. Cafagna



University of
Zurich ^{UZH}

Department of Informatics



Abstract

These theses document the working process of a database constructive web application with name Swiss Feed Database System, Version 2.0. One major point is hereby the inclusion of geographical information such as address specifications and the procedure to display the resulting locations usefully on a map. Next to a short introduction into the topic of Geographical Information System (GIS), a tutorial in embedding Google Maps will mostly cover that theme. Furthermore, the construction of an interactive web page by using a PostgreSQL database, PHP server scripts and JavaScript along with its Asynchronous JavaScript and XML (Ajax) object is part of this documentation.

Contents

1. Preface	1
2. Introduction	2
2.1. Swiss Feed Database Project	2
2.2. Geographical information system (GIS)	2
3. Analysis	5
3.1. Swiss Feed Database, Version 1.0	5
3.1.1. Application	5
3.1.2. Data	5
3.1.3. Database	5
3.2. Input from AGROSCOPE and AGRIDEA	5
3.2.1. AGRIDEA <i>Dürrfutterenquôte</i>	5
3.2.2. Data input	7
3.2.3. Parameters and values	7
3.2.4. Queries	9
4. Requirement	11
5. Design	14
5.1. Graphical outline in HTML	14
5.2. System architecture	15
5.2.1. PostgreSQL	15
5.2.2. PHP 5	15
5.2.3. Expendable Hypertext Markup Language (XHTML)	16
5.2.4. Javascript	16
5.2.5. Asynchronous JavaScript and XML (AJAX)	17
5.2.6. Google Maps	17
5.2.7. Google Visualization	18
5.3. System application flow	18
5.3.1. Module INIT	20
5.3.2. Module M1	20
5.3.3. Module M2	20
5.3.4. Module M3	20
5.3.5. Module M4	20
5.3.6. Module M5	21
5.3.7. Module M5+	21
5.4. Database design	21
5.4.1. Objects and relations	21
5.4.2. Triggers and Views	23

6. Implementation	26
6.1. Introduction	26
6.2. Database	26
6.2.1. Extraction, Transportation, Loading (ETL) of data	26
6.3. Application	31
6.3.1. Generating dynamic SQL	31
6.3.2. Asynchronous client-server communication	36
6.3.3. Geocode locations with Google Maps service in PHP	40
6.3.4. Embed a map of Google Maps with information from the database	41
6.3.5. Using Google Visualization	44
6.3.6. Interactive result display with JavaScript event listeners	47
7. Testing	52
7.1. Surveillance of implementation	52
7.1.1. Single functionality	52
7.1.2. Module	52
7.1.3. System	53
7.1.4. Testing environment	53
7.2. Application practice	53
8. Maintenance and outlook	54
8.1. Updates	54
8.2. Future data input	54
8.3. Future features and adaptations	54
9. Summary	56
9.1. Database system with geographical information	56
A. Code appendix	59
A.1. Start pages	59
A.2. General JavaScripts	68
A.3. Modular JavaScripts	70
A.4. Ajax triggered PHP scripts	82

List of Figures

2.1.	Schematical outline of the distribution of information into separated layers [Geostat 1994, p.3]	3
2.2.	Architecture and assigned tasks of a GIS [Longley et al. 2011, fig. 10.1]	3
2.3.	Example of a data item stored in two database tables <i>subject</i> and <i>geometry</i>	3
3.1.	Screenshot of the result display of the current Swiss Feed Database online Application [Agroscope 2009, "Berechungsprogramm"]	6
3.2.	A possible database design of the Swiss Feed Database 1.0 [very abstracted and only estimated].	6
3.3.	Two examples of a data delivery spreadsheet and its contents. Above: "Rohdaten_Durrfutter_2005-2009.xls", below: "Zusammenfassung_09_def.xls"	7
3.4.	Simplified input structure, indicating the five categories with additional information about static legends and potential cardinalities, data types and field names	7
3.5.	Developed feed content namespace. The dashed lines indicate combinations with Null-values, the italic names are later translations supplied by the AGRIDEA publications.	9
3.6.	Example of statistical subsets, left the use of quartiles, right use of standard deviation in 2sigma range	10
4.1.	UML use case diagram of the next Swiss Feed Database	11
4.2.	The users activities in the Swiss Feed Database online application as UML activity diagram	13
5.1.	Basic outline of the HTML web page. The icons are pointing to additional architectural components, required for the requested activity	15
5.2.	Design of the Swiss Feed Database system architecture	16
5.3.	UML state chart to illustrate the cooperation of the scripting languages on client and server side. Including also the state condition of the AJAX.	18
5.4.	UML sequence diagram of the complete Swiss Feed Database including all major components	19
5.5.	Illustration of the composition of samples.	21
5.6.	Relational concept of the Swiss Feed. Bold: used attributes so far (the others contain only NULL). Also indicated on top of every table is the distribution of the test data into the tables.	22
5.7.	A close up of the two tables <i>d_origin</i> and <i>d_time</i> with a illustration of the designed attribute hierarchy	23
6.1.	Example of a spreadsheet preprocessing to extract workable information concerning the sample's content.	27
6.2.	Schematical example of the references between the <select> fields	32

6.3.	The outline of the PHP class 'SelectField'	33
6.4.	Modeling of the 'generate SQL WHERE IN part' algorithm	37
6.5.	Modeling of the 'Geocode' algorithm.	42
6.6.	Modeling of the combined 'transform samples into Google Maps marker' (dark gray) and 'fill the visualisation dataTable' (light gray) algorithm	46
6.7.	Illustration to the task to transform an XML input, with minimalistic storing principles into a table with static columns	47
6.8.	Illustration of the statistical definition of a outlier	48
6.9.	Screenshot of the Swiss Feed Database 2.0 surface with activated interactivity 'show outlier for nutrient'	48
7.1.	The technical environment during the implementation and testing phase of the Swiss Feed Database 2.0.	53
9.1.	Final presentation of the selection part in the Swiss Feed Database 2.0.	56

List of Tables

2.1. Overview of the four main data types in a GIS. Each type with it separate definition of geometry [BFS 1994, p. 4, modified]	3
5.1. Summary of the view triggering attributes in the database table d_time.	25
6.1. Illustration of the importance of NULL values for tuple specification. Bold are the attributes, which are covered in the new data delivery.	29
9.1. Composition of the application output.	58

Appendix: code listings

A.1. feddb-start.php	59
A.2. styles-feddb.css	62
A.3. class-selectfield.php	65
A.4. pg-dbinfo.php	67
A.5. jsA_global-variables.js	68
A.6. jsB_ajax-object.js	68
A.7. jsC_loader.js	69
A.8. jsD_sql-from-where.js	69
A.9. js1_update_selectfield.js	70
A.10.js2_result_coordinator.js	72
A.11.js3_result_list-map.js	74
A.12.js4_result_diagram.js	77
A.13.js5_result_aggregate.js	79
A.14.ajax-pg-options.php	82
A.15.ajax-pg-result-google.php	82
A.16.ajax-pg-result-linediagram.php	85
A.17.ajax-pg-result-scatterdiagram.php	86

1. Preface

These present thesis will document the implementation process of the Swiss Feed Database System Version 2.0., a database constructive web application. In order to do so, the structure of content is designed according to the 'waterfall model' of software development.

In this sense, a first part contains the project preproduction, with the outline of the general surroundings and technical possibilities (chapter 'Introduction'), mainly affected by the presentation of the geographical information systems (GIS). Next in line is the analysis of the given input data along with the accessible ongoing research queries in the topic of historical feed development (chapter 'Analysis') which is followed by the formulation of the required functional specifications of the system (chapter 'Requirement').

A second part is dealing with the construction of the application. The chapter 'Design' is covering the appearance of the web application, but about all things, the different required components, its characteristics, its appliance and the communication between them. In the chapter 'Implementation', selected topics of the construction will then be discussed in a manner of tutorials. The presented excerpts are exemplary for the processing of the practical work and thereby, the most attention is concentrated into this part.

In the end, two chapters concerning the testing and the maintenance of the application are building, in combination with an overall summary, a concluding part with an outline for future revisions.

As an endnote in this preface, it has to be mentioned that these thesis are neither a strict software documentation nor a more general project protocol. Although it includes a little bit of both, it is principally a documentation of the learning process of the application's creator. The most basic knowhow were acquired from scratch and many later 'simple' constructions were keeping the author occupied for long times. That's why some features, especially in the implementation chapter, are discussed little bit more detailed.

2. Introduction

2.1. Swiss Feed Database Project

The Swiss Feed Database is a project on federal level hosted by the laboratory AGROSCOPE of the Department of Agriculture. The goal is to collect information about the chemical and nutritional content of the inland production of animal feed, aggregate and display this information for scientific research, and make the results usable for farmers in form of consultable guidelines and interactive forms [Agroscope 2011].

As a result of this project a first version of an online application was established. But a next development should now take place in collaboration with the Department of Informatics of the University of Zurich. One task will be to include geographical information about the feed sample's origin into the Swiss Feed Database.

2.2. Geographical information system (GIS)

The illustration of the sample's origin will be handled in this new application by including the Google Maps service. Since this service is only one of many tools, dealing with the same technical possibilities to embed geographical information, a little excursus will be taken by introducing the *geographical information system*.

A geographical information system is a model in a scientific sense: it displays the world simplified yet specialised on a specific topic.

- The *geographical* in the name reveals the main purpose of a GIS to display a specific matter in topographical space. Therefore, the base of every GIS is a geographically encoded coordinate system in two or three dimensions (graticule and elevation model).
- Based on this grid, *information* referring to any topic with a spatial distribution can be added in form of layers connected to the coordinate system (fig. 2.1). Again, this information is normally simplified and originates either of random enquiries (e.g. population density) or optical surface analysis (e.g. road network).
- In the end, a GIS is a computerized, interactive *system* not a static, analogue publication. The user is allowed to choose the scale of the grid and the combination of the information layers according to his requests.

The technical implementation of this required system is always based on a database (*object-relational DBMS* and *data* in the fig. 2.2). It stores the previously mentioned information as data, where every data unit combines the thematical value (subject) along with its spatial reference (geometry), as the example in figure 2.3 illustrates. According to the geometry, every data item can be further distinguished into four data types (table 2.1).

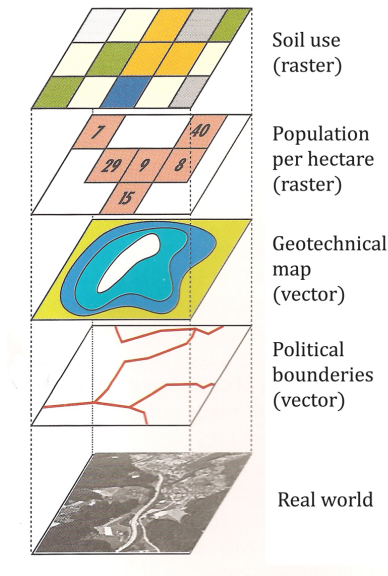


Figure 2.1.: Schematic outline of the distribution of information into separated layers [Geostat 1994, p.3]

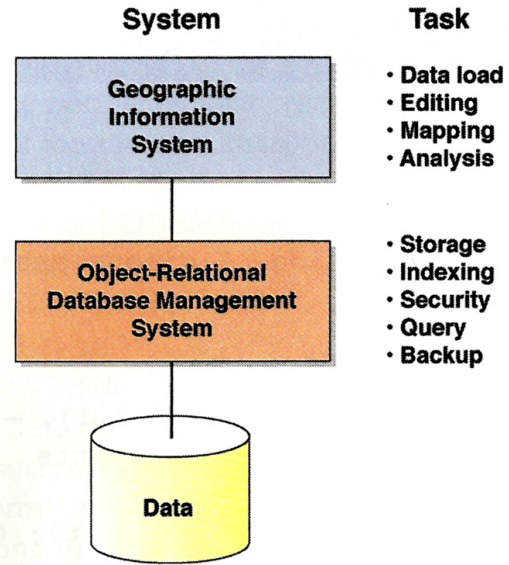


Figure 2.2.: Architecture and assigned tasks of a GIS [Longley et al. 2011, fig. 10.1]

Subject (e.g. Cities)					Geometry		
SID	City_name	City_size	Population	Geometry	GID	Latitude	longitude
1	Zürich	91.88	385'468	A ●	A	47.379022	8.541001

Figure 2.3.: Example of a data item stored in two database tables *subject* and *geometry*.





MATRIX REPRESENTATION	VECTOR REPRESENTATION
 <p><i>Cell</i> geometry, defined by the coordinate of a grid cell, e.g. population per square</p>	 <p><i>Point</i> geometry, defined by the coordinate of a point, e.g. city centers</p>
	 <p><i>Line</i> geometry, defined by the coordinate of start, end and points in between of the line, e.g. rivers</p>
	 <p><i>Polygons</i> geometry, defined by the coordinates of its boundaries, e.g. lakes</p>

Table 2.1.: Overview of the four main data types in a GIS. Each type with it separate definition of geometry [BFS 1994, p. 4, modified]

In addition to the database, the GIS application has to be developed, which will be used as graphical user interface (*geographic information system* in fig. 2.2). Its first and most important job is the so called *mapping*, where selected data items will be visually placed on the geographical reference system to build a map. Next to this the assignment, the application is also implemented to execute predefined algorithms, especially on the geometry values of the database items. Popular algorithms are dealing with the geographical relationship of selected data units, e.g. least cost paths from one point/cell to another or the analysis of the visibility of the surrounding elements of a point/cell. Other algorithms are producing new information layers by calculating with several existing layers, e.g. the combination of the layers about geology, insolation and precipitation generates a map of agricultural suitability. As long as the database is covering all needed information in a good relational organisation, the algorithms know (almost) no bound.

The idea of the geographical information system to build an interactive map, where zoom, excerpt and content can be changed to own will, is interesting for an applying in scientific research as well as in daily use. That's why, GIS is very popular and more and more GIS basing tools will be developed. The largest of it is the very famous Google Maps service, to which we will come again later on in this thesis.

3. Analysis

3.1. Swiss Feed Database, Version 1.0

3.1.1. Application

The first version of the database is online since 2007. It implements an interactive interface in which a user can choose one or several feed types on one hand and scientific parameters like nutrients and animal energetic and digestive parameters on the other hand. The application delivers as a result the value of the chosen nutrients for every chosen feed type or the information concerning the values of a feed's nutrient content for a specific animal type or both (fig. 3.1). The application can be accessed in German and in French [Agroscope 2009].

3.1.2. Data

The values used in this application are static mean values, evaluated and updated manually by AGROSCOPE in advance and stored in a database. The input for the animal dependent parameters originates from several agricultural research projects from 1999 until 2006 which are the official reference for animal feed composition [Agroscope 2009][*"Datenherkunft"*]. The origin of the nutrient values is not declared except of the annotation, that they are aggregated mean values [Agroscope 2009][*"Startseite"*]. But they will most likely originate from the laboratories of AGROSCOPE itself together with collections from the AGRIDEA *Dürrfutterenquête* (see chapter 3.2.1).

3.1.3. Database

Derived from the online application, the database seems to fulfil two purposes (fig. 3.2):

1. It organises the families, groups and subgroups of the scientific parameters, so that these can be chosen by a user without difficulties.
2. It associates a value to every "feed type-parameter" connection whereupon the value can also be derived by formula from other parameters in runtime.

3.2. Input from AGROSCOPE and AGRIDEA

3.2.1. AGRIDEA *Dürrfutterenquête*

Next to AGROSCOPE with its three belonging laboratories, there is also a second source providing data in this research field that is included into the database project. Since thirty years, AGRIDEA - an association of agricultural organisations and institutions - collects measurements of animal feed samples from various producers in Switzerland every year.

The screenshot shows the Agroscope website interface. At the top, there is a header with the Swiss flag and the text 'Schweizerische Eidgenossenschaft', 'Confédération suisse', 'Confederazione Svizzera', and 'Confederaziun svizra'. A navigation bar includes 'Suche' and 'ALP'. The main content area is titled 'Ihre aktuelle Auswahl:' and contains several sections: 'Suche mit Restriktionen' (Suchkriterien: 0), 'Futtermittel Auswahl' (Futtermittel: 3), 'Parameter Auswahl' (Nährstoffe: 3), and 'Ausgabeoptionen'. A table displays feed ingredients and their nutrient content (K, Na, P, NEL). A detailed view for 'Knautgras, Stadium 1, Dürrfutter' is shown on the right, listing parameters like 'Parametername: Kalium', 'Werte-Typ: Fixwert eingegeben', and 'Änderungsdatum: 2007-07-27 10:02:59'.

Futtermittel	K [g/kg TS]	Na [g/kg TS]	P [g/kg TS]	NEL [MJ/kg TS]
Englisches Raigras, Stadium 1, Dürrfutter	37.2	0.20	4.4	5.87
Italienisches Raigras, Stadium 1, Dürrfutter	37.6	0.20	4.2	5.96
Knautgras, Stadium 1, Dürrfutter	40.3	0.20	4.6	5.80

Figure 3.1.: Screenshot of the result display of the current Swiss Feed Database online Application [Agroscope 2009, "Berechnungsprogramm"]

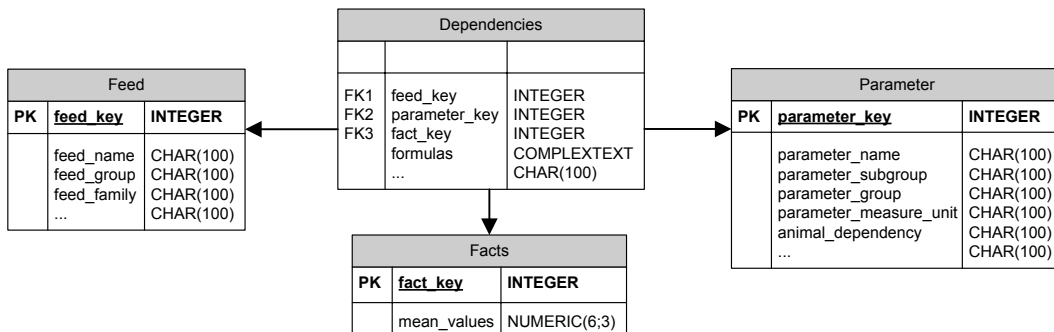


Figure 3.2.: A possible database design of the Swiss Feed Database 1.0 [very abstracted and only estimated].

As a connection between the farmers and the governmental research facilities, such as AGROSCOPE, they are also responsible for the definition of the scientific parameters as well as the quantity, the quality, the temporal and geographical representation of a big partition of the overall data input. This sample testing happens in four associated laboratories [Agridea 2011].

3.2.2. Data input

As mentioned, the data used in the old as well as in the new application are provided by the laboratories of AGROSCOPE and AGRIDEA. As exchange media, spreadsheets have been chosen. Unfortunately, several spreadsheets have been generated and the combination of its columns as well as the column names and the distribution of information to columns are different in several cases, depending on the year, the laboratory or on the language. All in common is that every row corresponds to one unique sample, while one sample consists of several measurements.

Pnr	Jahr	APDE	APDN	Ca	PLZ	Material	Belüftung	Bezeichnung	Bot Zus	Bot Zus Kurz	Schnittdatum	müM	Bemerkungen	Region	Höhenstufe
07-14779-001	7	87.44	78.19		7302	Dürrfutter	belüftet	Heu / Emd	(4) AR ausgewogen (Raigras)	(4)	00.01.1900	800 - 999 m			10
07-14778-001	7	91.53	85.93		7302	Dürrfutter	belüftet	Heu / Emd	(1) unbek. bot. Z-setzung	(1)	00.01.1900	600 - 799 m			10
07-14768-001	7	88.19	78.03		7302	Dürrfutter	belüftet	Heu / Emd	(4) AR ausgewogen (Raigras)	(4)	00.01.1900	> 1000 m			10
07-14761-001	7	85.27	84.81	9.13	9308	Dürrfutter	belüftet	Heu / Emd	(3) G gräserreich (andere)	(3)	00.01.1900	600 - 799 m			8
06-17008558765681	6	91.89911246	89.3840547		3475	Heu / Emd	belüftet		(4) AR ausgewogen (Raigras)	(4)		600 - 799 m	Stock 06		3
06-16835558765681	6	86.39460204	74.9557998		6210	Heu / Emd	belüftet		(4) AR ausgewogen (andere)	(4)		< 600 m		0	6
06-16828558765680	6	94.04778616	90.15480913		6170	Heu / Emd	belüftet		(5) A ausgewogen (andere)	(5)		800 - 999 m		0	3
06-16827557765680	6	96.7473819	100.247782		6170	Emd	belüftet		(5) A ausgewogen (andere)	(5)		800 - 999 m		0	3
06-16824555765680	6	84.03907254	59.64233354		6170	Heu	belüftet		(5) A ausgewogen (andere)	(5)		800 - 999 m		0	3
06-16808548758568	6	85.14281263	78.612329		6212	Heu / Emd	belüftet		(5) A ausgewogen (andere)	(5)		< 600 m	St. Erhard / Knd Nr. 61	0	6
06-16788564771585	6	88.08294582	76.34813266		1816	Foin / Regain	belüftet		(3) G gräserreich (andere)	(3)		< 600 m	1816 Chailly-Montreux	3	1
06-16788562769585	6	90.81255808	85.37545348		1801	Foin / Regain	belüftet		(3) G gräserreich (andere)	(3)		600 - 799 m	1801 Le Mont-Pèlerin	3	2
06-16756548759569	6	90.87281655	84.55211427		9107	Heu / Emd	belüftet		(1) unbek. bot. Z-setzung	(1)		800 - 999 m	Ruppen, 9107 Uräsch	9	3
06-16743554763577	6	86.97423184	80.98923456		6207	Heu / Emd	belüftet		(3) G gräserreich (andere)	(3)		< 600 m		0	6

Labor	PLZ	Altitude	Höhenstufe	Region	TS	RP	RF	APDE	NEL	APDN	Ca	P	Mg	K	Sample Code	Art	Sample Reference
Eurofin	2336	1000m	4	1	138	271	75	5.4	87						107-2009-00815196	3	Ensilage herbe boîte carrée juin
Eurofin	2714	1030m	4	1	150	305	75	5.3	95	6.4	3.6			107-2009-00815188	3	Ensilage herbe, 1er coupe fin mai	
Eurofin	2350	1050m	4	1	154	303	81	5.3	97					107-2009-00815189	3	Ensilage herbe, 1er coupe début juin, 2e 6 semaine, 3e 6 semaine	
Eurofin	8553	460m	1	8	118	279	79	4.9	75					107-2009-00814529	1	Dürrfutter belüftet	
Eurofin	8586	480m	1	8	120	268	62	5.1	76					107-2009-00814513	1	Dürrfutter belüftet	
Eurofin	8242	500m	1	5	159	286	72	5.2	100	10.4	4			107-2009-00812251	3	Grassilage	
Eurofin	8535	500m	1	8	106	318	72	4.4	67	6.3	3.2	1.7	34	107-2009-00814514	1	Dürrfutter belüftet	
Eurofin	8535	500m	1	8	139	271	84	5	89	8.2	4.1	2.4	36.7	107-2009-00814515	1	Dürrfutter belüftet	
Eurofin	8536	500m	1	8	107	268	77	4.8	68	6.2	3.6	1.8	36.1	107-2009-00814522	1	Dürrfutter belüftet	
Eurofin	2824	500m	1	1	77	179	62	6.3	48					107-2009-00815829	4	Ensilage de Maïs	
Eurofin	8564	550m	1	8	137	293	82	4.7	87					107-2009-00814530	1	Dürrfutter belüftet	
Eurofin	8556	580m	1	8	122	275	81	5	77	7.3	3.6	2.1	38.1	107-2009-00814517	1	Dürrfutter belüftet	
Eurofin	1117	580m	1	2	100	365	67	3.9	63					107-2009-00815326	2	Foin balles rondes	
Eurofin	6314	600 - 799m	2	7	127	244	86	5.3	81					107-2009-00807937	1	Dürrfutter belüftet	
Eurofin	4614	600 - 799m	2	6	81	223	64	5.8	53	2.8	1.5			107-2009-00812992	4	Maissilage	

Figure 3.3.: Two examples of a data delivery spreadsheet and its contents. Above: "Rohdaten_-Dürrfutter_2005-2009.xls", below: "Zusammenfassung_09_def.xls"

3.2.3. Parameters and values

In general the usable data from the spreadsheets can be assigned to 1 of 5 categories.

Sample ID	Laboratory	Nutrient 1	Nutrient 2	Nutrient 3	(...)	year	harvest day	postal code	region*	altitude	alt. Level**	content	cont. code	variety
number or empty	name or empty	number, 0 or empty	number, 0 or empty	number, 0 or empty		number	date or empty	number or empty	number or empty	value or interval	number or empty	scattered information	number or empty	drop down
Sample code	Labor	TS	RP	RF	...	> Filename	-	PLZ	Region	Altitude	Höhenstufe	Sample Reference	Art	-
Pnr	-	ADPE	APDN	Ca	...	Jahr	Schnittdatum	PLZ	Region	müM	(müM)	Material	-	Bot. Zus.

- **Legend:**
 1/A = <600
 2/B = 600-799
 3/C = 800-999
 4/D = >1000

- *Legend:**
 1 = JU-NE
 2 = VD
 3 = BE-FR
 4 = Mittelland
 5 = BS-BL
 6 = LU-AG
 7 = 4 cantons
 8 = ZH-TG
 9 = GL-AR-AI
 10 = GR
 11 = TI
 12 = VS



Includes content about

- feed group
- feed name
- ventilation

Figure 3.4.: Simplified input structure, indicating the five categories with additional information about statical legends and potential cardinalities, data types and field names

1. Information concerning the sample itself, e.g. sample key and laboratory name
 - Less than 50 % of the samples have a communicated sample key. The laboratory, where the data is coming from is not always indicated.
2. Measurements of the sample composition (nutrients) and measurements concerning the absorption of the feed, e.g. energy level, grease etc. (nutritive values).
 - Every spreadsheet has its own composition of nutrients and nutritive values.
 - Not taken measurements are stored as empty cells or with the value 0.
3. Information concerning the time, e.g. year or harvest day.
 - A sample can always be associated with a year, mostly because a whole sheet is associated to a specific year (Filename: data_2009.xls).
 - Only a small amount of samples have a higher resolution than the year (for instance: harvest day or season).
 - Sometimes this closer information is mentioned in commentary columns or in the more detailed entries of other categories - *therefore this information cannot be collected*.
4. Information concerning geographical references e.g. postal code or altitude.
 - Almost every sample is associated with a postal code and an altitude.
 - Sometimes not existing postal codes are used (as a result of a spelling mistake) and sometimes it is not clear if the postal code refers to the farm or the laboratory - *as a consequence, the postal codes will be stored as they come (no corrections will be made)*.
 - If the altitude is used it always assigned to one of four altitude levels (see legend in fig. 3.4).
 - In addition, a secondary column with a specific altitude value can be given, but the values differ from absolute to rounded numbers to intervals.
5. Information concerning the material of the sample, like feed name, feed variety or feed treatment.
 - This information is the most scattered one, because there is no real classification of the topic 'feed content'.
 - There is no column with distinct feed names since the information is mixed up with the material description.
 - There is an overall code for the material and its drying conditions, but no assigned overall legend.
 - The two languages German and French are mixed up, whilst depending on the language, the information is associated to different columns.
 - Some entries are more detailed than others.
 - French and German vocabulary doesn't necessarily match.
 - [With the goal to catalogue the material content for later use, a description structure was developed (fig 3.5). This was done by analysing all possible combination of content and content code (by leaving out too detailed information). In addition a French AGRIDEA publication was needed for a proper translation [Python 2010, p.159]]
 - The sample variety (used in 50 % of the samples) has been catalogued by a numerical code with corresponding legend.

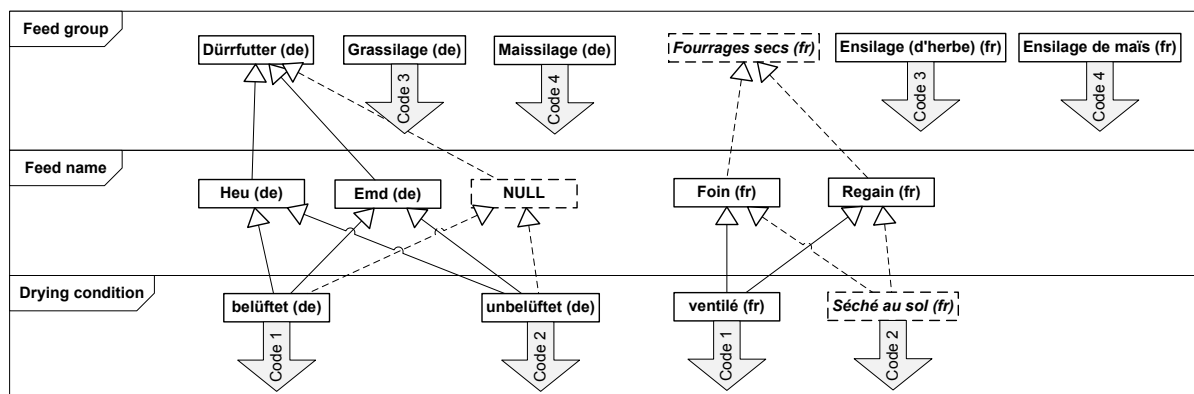


Figure 3.5.: Developed feed content namespace. The dashed lines indicate combinations with Null-values, the italic names are later translations supplied by the AGRIDEA publications.

3.2.4. Queries

AGRIDEA - the agricultural association - is not only collecting measurements, they are also elaborating statistical evaluations on these measurements for agricultural use as well as scientific research. The following part shows the most important queries derived from the ongoing approaches. The enlistment is broken down to the lowest thematical queries. In real life, a combination of these queries is applied.

Agricultural approach queries:

- Provide up-to-date mean values for every parameter (nutrients and nutritive values) of every feed. These mean values are later used as current references in consulting farmers for the next feeding season. (Covered by the Swiss Feed Database Version 1.0)

```

1 SELECT feed_name, parameter_name, avg(parameter_quantity)
2 FROM samples WHERE year = "up-to-date"
3 GROUP BY feed_name, parameter_name; [Estimated SQL query]

```

Scientific research queries:

- Elaborate the **historical** shifting over time of the measurements for a later attempt to correlate the development of feed quality to weather conditions or to agricultural systems. This elaboration calls mostly for the use of mean values.

```

1 SELECT [year||season||day], avg(parameter_quantity)
2 FROM samples WHERE feed_name="x" AND parameter_name="y"
3 GROUP BY [year||season||day];

```

- Elaborate the **geographical** shifting of the measurements. Thematically by region [Boessinger 2010] or canton¹, topographically by altitude or in combination. The results of these queries are used for a spatial determination of different qualities.

¹Oral request of AGROSCOPE made in the meeting June 17, 2011.

Quartiles			
average	avg(upper 25%)	avg(middle 50%)	avg(lower 25%)
nutrient	86	90	85

Standard deviation (2 σ)			
average μ	avg(> $\mu+2\sigma$)	avg(in 2 σ range)	avg(< $\mu-2\sigma$)
nutrient	86	92	84

Figure 3.6.: Example of statistical subsets, left the use of quartiles, right use of standard deviation in 2sigma range

```

1 SELECT [canton||altitude],[...], avg(parameter_quantity)
2 FROM samples WHERE feed_name="x" AND parameter_name="y"
3 GROUP BY [canton||altitude],[...];

```

- Elaborate the statistical composition of certain values (used as a statistical background and as extension to the mean values from above).

- Additional aggregations: count of involved measurements; lowest vs. highest measurement; the standard deviation

```

1 SELECT COUNT(parameter_quantity), MIN(parameter_quantity),
2 MAX(parameter_quantity), STDDEV(parameter_quantity)
3 FROM samples WHERE feed_name="x" AND parameter_name="y";

```

- The creation and comparison of subsets of the measurements, either as quartiles in percentage [Boessinger 2010] or by the standard deviation², mostly used to identify outliers and sharpen the aggregation values (fig. 3.6).

```

1 BEGIN
2   float average, stddev = SELECT AVG(parameter_quantity),
3     STDDEV(parameter_quantity)
4     FROM samples
5     WHERE feed_name = "x" parameter_name = "y";
6   float measurements[] = SELECT parameter_quantity
7     FROM samples
8     WHERE feed_name="x" AND parameter_name="y";
9   foreach (array[] as quantity){
10    case (quantity > (average + stddev)):
11      /* higher then stddev upper bound */
12    case (quantity < (average - stddev)):
13      /* lower then staddev lower bound */
14    else:
15      /* inside the stddev range */
16  }
17 END

```

- Last but not least: display all measurements to make them accessible and verifiable as an ultimate principle in scientific research.

```

1 SELECT * FROM samples;

```

²Oral request of AGROSCOPE made in the meeting July 15, 2011.

4. Requirement

The analysis of the input data together with the ongoing research and the first version of the Swiss Feed Database allows it to briefly outline the future functionalities of the Swiss Feed Database Version 2.0.

(1) *The next Swiss Feed Database should additionally provide the possibilities to be applied in the scientific research concerning temporal and spatial shifting of measurements.*

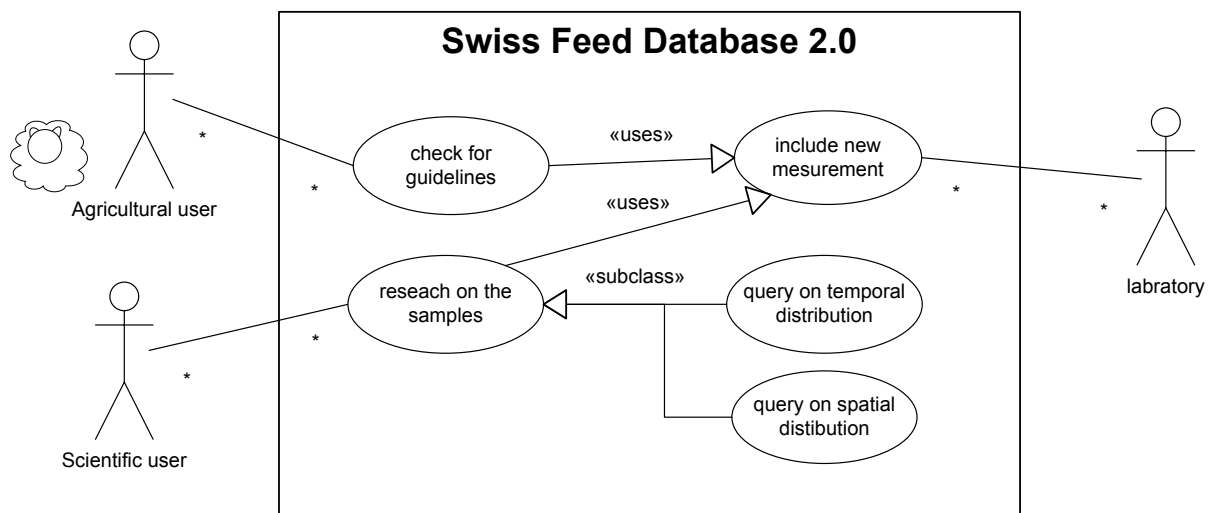


Figure 4.1.: UML use case diagram of the next Swiss Feed Database

For the database, this calls for an extension to store not only up-to-date mean values, but all samples in separate. Additional database relations are necessary to include also the information about the geographical origin of the samples, which were not covered so far. The same needs to be done for the new temporal values.

In the application, this means to establish interactive filter options on parameters like year and harvest day (temporal selection), canton (spatial selection), feed name and nutrient name (result selection) or more, used in a search to allow every user to specify the sample selection for his enquiry himself. By using such interactive filter options, the implementation of generating dynamic SQL queries is absolutely necessary. A second task would be to realise a predefined set of scientific operations with a well-arranged display of results. Especially for the display of special distribution, the geographical information system Google Maps should be included.

Set of covered operations:

1. Computation of the aggregate values count, minimum, maximum, average, standard deviation (of a sample) of the involved measurements.
2. Computation of the aggregate value average in every statistical subset.
3. Computation of the aggregate value average for every distinct temporal value (= development of mean value over time).
4. Enlistment of all involved measurements with temporal information (= distribution of values over time).
5. Enlistment of all involved measurements with information about their origin (= distribution of values in space).
6. Enlistment of all involved measurements with their parameter quantities (= access to original data).

(2) The next Swiss Feed Database should become a tool with a simple and easily adaptive interface.

When it comes to the users activities, the online application should be very linear to guide the user through his inputs. On the other side it should not be too static, so that changes in the user's mind can be turned quickly. An optimal application flow would incorporate the following (see also the activity diagram in fig. 4.2):

- The user is called upon to delimit the sample's quantities in the database to his request by the selection of multiple sample parameters of the dimensions 'feed content', 'measurement content' 'space' and 'time'.
 - **First constraint:**
Every next selection is dependent on the previous one, so that no invalid combination parameters (such with no result) can be made.
- The users input ends with a call for results.
- This result is then the predefined set of operations applied to the previously made selection choice, displayed in four major categories: the result of presented measurements (operation 6), the result of presented locations (operation 5), the result of a presented temporal development of every chosen nutrient(operation 3 & 4) and the result of aggregations for every chosen nutrient (operation 1 & 2).
 - **Second constraint:**
Allow the user to correct his selection choice whenever he wants and on any point he wants it. It includes also the possibility to to send more then one enquiry in his session. If a correction occurs, it means that old results and parameters needs to be properly overwritten.

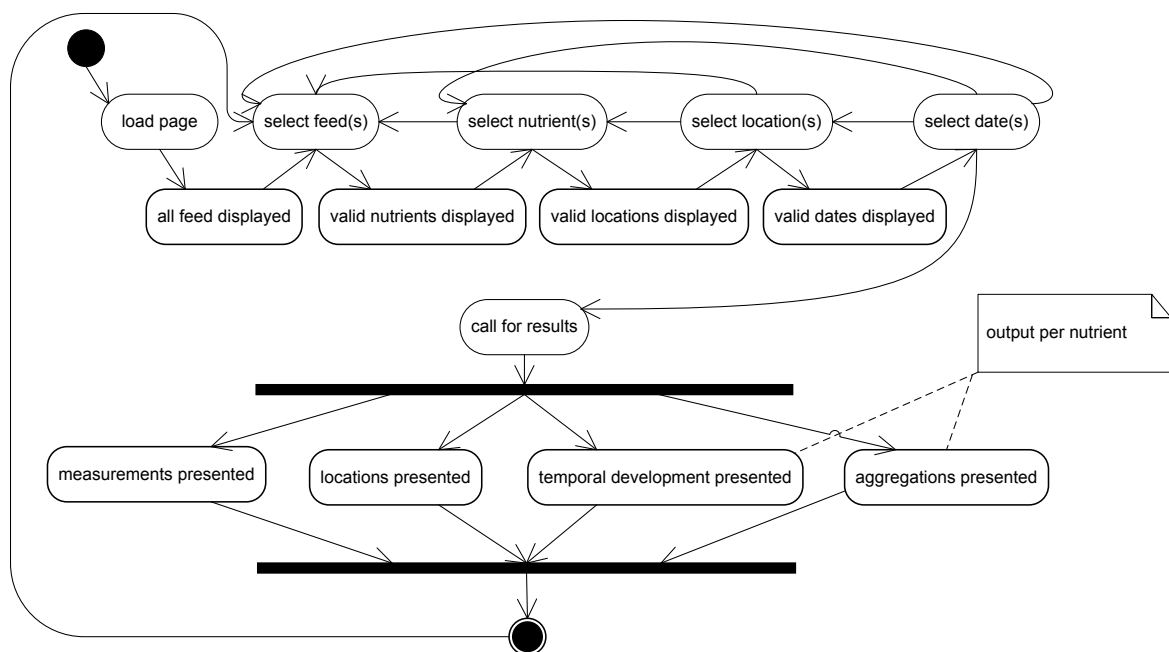


Figure 4.2.: The users activities in the Swiss Feed Database online application as UML activity diagram

5. Design

5.1. Graphical outline in HTML

The requirements formulated in the previous chapter determine the appearance of the online application. Because of the second constraint 'to provide the possibilities to change the selection and load new results all the time' the application needs to be set in one view only, which is then divided into two subparts 'selection' and 'result'.

The decision of building one page only is reducing the usable space enormously and requires a good organisation of the layout.

For the implementation of the basic structure the very common Hypertext Markup Language (HTML) shall be adopted, which is supported by all browsers. In a first step the focus will lie on the nestable block elements (<div>), which will be applied to divide the one web page into several regions and subregions. The basic outline can be seen in figure 5.1. By manipulating the style-attributes of every <div>, for example in a separately stored Cascading Style Sheets (.css), a two dimensional distribution, relative in size to the changeable client's browser window can be reached.

The interior of the blocks is then designed according to its purpose. The selection part will contain a <form>, filled with several <select multiple> elements, so that more then one option of the corresponding filter parameters can be displayed and selected at the time.

Example: HTML code a 'multiple' select field

```
1 <form name="exampleOne" multiple>
2   <select name="selectOne" multiple>
3     <option value=1 >Choose me</option>
4     <option value=2 >Chooose me</option>
5     <option value=3 >Chooooose me</option>
6     <option value=4 >Choooooose me</option>
7   </select>
8 </form>
```

In the subpart 'aggtable', an old fashioned HTML <table> should be generated in runtime to display the aggregations of every selected nutrient, according to the requirements. All other result presenting subparts will be generated in combination with selected web services, which exceeds the normal possibilities of HTML. As already mentioned, a Google map should be embedded to display sample locations. In addition another Google service, called Visualization, shall be used to create and present interactive charts to illustrate rather statistical results, like the temporal development of measurements (line chart) or the enlistment of all involved samples (sortable table chart).

As an concluding note at the end of this section, it should be indicated that an HTML page itself is static as soon as it got loaded. To provide the activity of the requirement, additional scripts on client (browser) and server side are needed. This will be the topic of the next section.

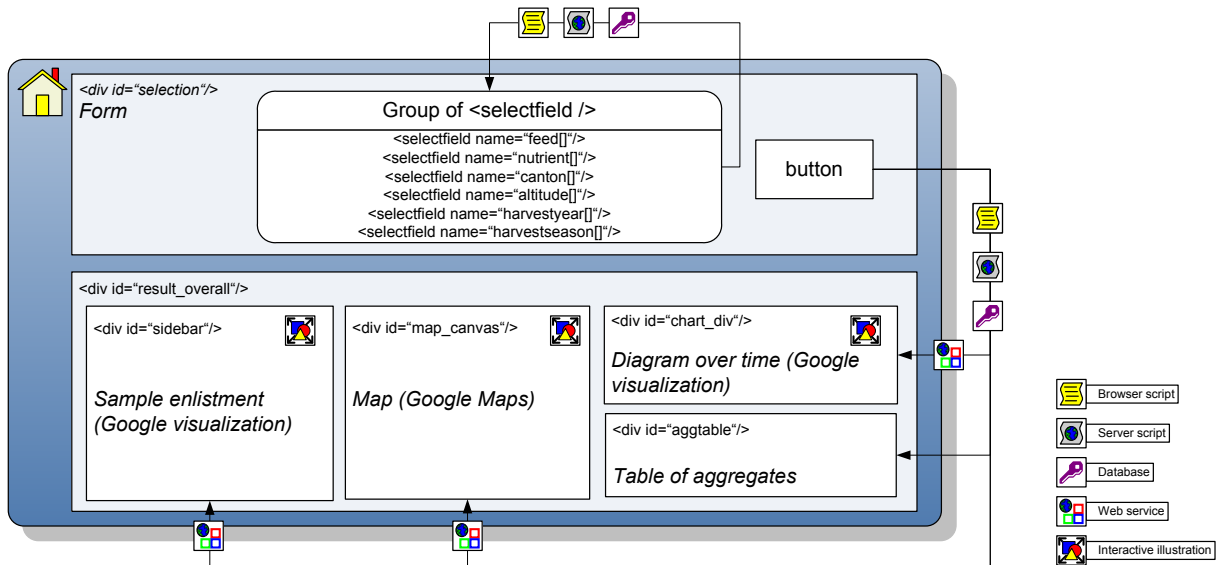


Figure 5.1.: Basic outline of the HTML web page. The icons are pointing to additional architectural components, required for the requested activity

5.2. System architecture

To realise an interactive application, which is set on only one HTML page and includes a database as well as web services, several architectural components on different locations have to be considered. The resulting design of architecture is illustrated in figure 5.2. In following, the major parts will be described in their functionality and discussed in their purpose of this application.

5.2.1. PostgreSQL

The object-relational database system used in this second Swiss Feed Database version consists, like all database system, of two components: there is the actual database where all the measurements along with its further associated information are stored, and then there is the Database Management System, which accepts communication from outside, executes queries and maintains the consistency of the database [PgSQL 2010, "Preface"] [Kemper 2006]. Similar as in the geographical information systems, the database system is the foundation of the complete Swiss Feed Database. The whole build-on application is only an additional tool for easier access and more visualising output.

5.2.2. PHP 5

The shortcut PHP originates from the name *Personal Homepage Tool* and *Hypertext Preprocessor* and indicates the use for the development of web pages. It is a server-side scripting language and a processor module, which provides interfaces to any bigger Database Management Systems, like PostgreSQL (extension 'php_pdo_pgsql.dll') [Achour et al. 2011, "Introduction" and "PostgreSQL"]. PHP operations can be embedded directly into HTML, as well as it itself can generate new HTML expressions. Because the PHP script runs on the web server before being loaded by the client, PHP is used in this application to run the SQL queries on the database and

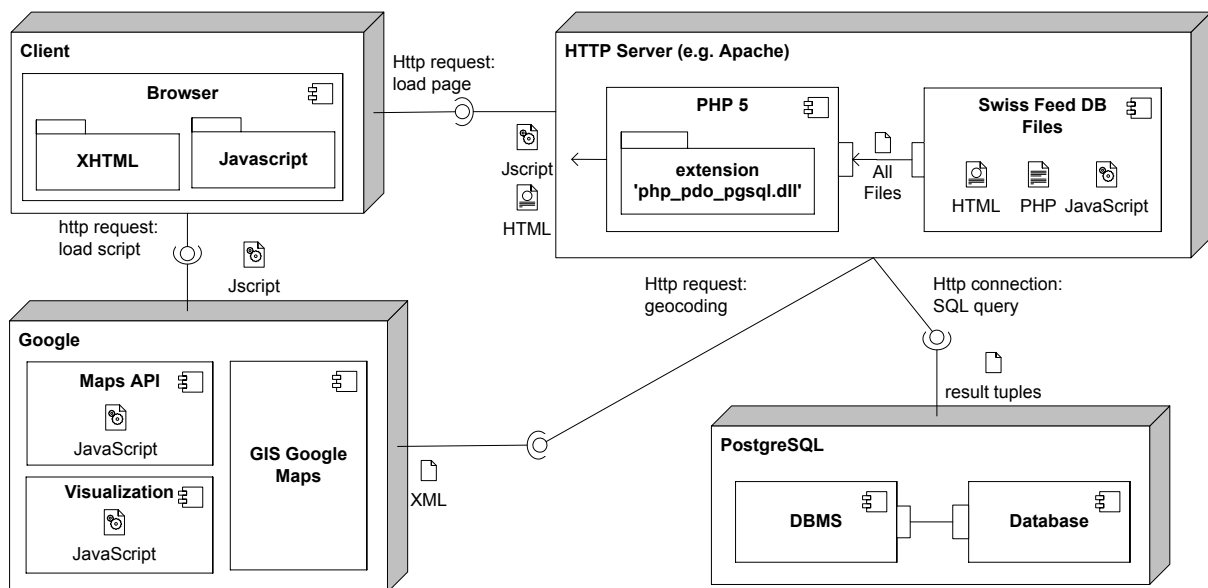


Figure 5.2.: Design of the Swiss Feed Database system architecture

dynamically generate HTML expression out of database result tuples. PHP 5 is the first version being object oriented.

Example: HTML with embedded PHP to print the current weekday

```

1 <html>
2   <?php
3     $dateArray = getDate();
4     echo "<p>Today is " . $dateArray[weekday] . "</p>";
5   ?>
6 </html>

```

5.2.3. Expendable Hypertext Markup Language (XHTML)

HTML is a language used for the basic web programming. It includes a large set of elements (like titles, paragraphs, tables, select fields...) and corresponding attributes (among them also the Internet links), which are used to shape the appearance of a web page similar to a classical typesetting tool. In the Swiss Feed Database, the higher improved language version eXpendable Hypertext Markup Language will be used, which uses a more stricter and cleaner code. But besides this no bigger differences to the non-expandable version exist [Refsnes Data 2011, "XHTML intro"]. The purpose of this component is the optical output design, like already discussed in section 5.1.

5.2.4. Javascript

This client-side scripting language supports the possibilities of *dynamic HTML* or also known as *DOM scripting*, where the complete appearance of a page can be changed even after the page has been loaded. This happens by calling and (re-)setting the values, children or just the formatting attributes of any HTML elements by a script operation in runtime. As triggers for such operations act the so called *event handlers* (onload, onclick, mouseover...), which are

embedded in the HTML code of specific elements. The JavaScript files, containing the associated operations, are loaded together with the HTML page by an script import tag in the header, if a JavaScript capable browser is used (Annotation: all modern browsers are JavaScript enabled)[Refsnes Data 2011, "Javascript" and "Browser statistics"].

In the online application, Javascript is the backbone of all user interactivity, which is required since everything is handled on just one web page with different states.

Example: HTML code with embedded Jscript

```

1 <html>
2   <head>
3     <script type="text/javascript"
4       src="scriptFile.js"></script>
5   </head>
6   <body>
7     <p id="example" onClick="getDate()" >
8       Click on me...</p>
9   </body>
10  </html>

```

Example: scriptFile.js with 'date' operation

```

1 function getDate() {
2
3   DOM-scripting operation: replace text 'click on me' with current date expression
4   document.getElementById("example").innerHTML=Date();
5 }

```

5.2.5. Asynchronous JavaScript and XML (AJAX)

Ajax is the special XMLHttpRequest object of JavaScript and will be provided by the client's browser if asked for. By the use of this object, it can be avoided to reload an HTML page on the client side in order to get new information from the terminated server. Ajax establishes an HTTP-connection from the running web page to the server and sends a request to (re-)call a certain server script. Then Ajax waits until the server delivers back the result (fig. 5.3) [Refsnes Data 2011, "AJAX" (hidden chapter)]. If this request was successful, the result can then be transformed into HTML code and displayed on the screen by using the dynamic HTML functionality as presented above.

In the Swiss Feed Database application, Ajax is used several times to call and receive query results by PHP from the PostgreSQL database whilst the web page is loaded. This is needed for the dynamic output of the select fields as well as for the end results.

5.2.6. Google Maps

The Maps service of Google is a geographical information system (remember chapter 2.2), which covers the whole planet with different granularity of information content. Google provides basically two functionalities, which both are embedded in this application:

1. To use the content of the Google Maps database to cross reference own information [Google 2011b].
Application: sending a postal code and get its coordinates as return.

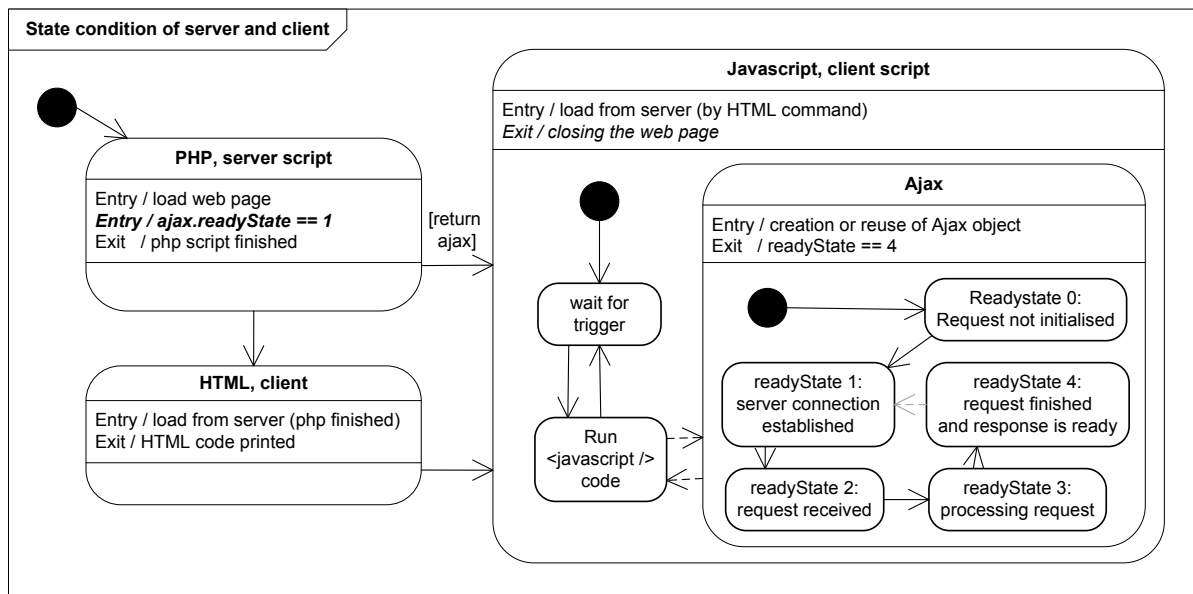


Figure 5.3.: UML state chart to illustrate the cooperation of the scripting languages on client and server side. Including also the state condition of the AJAX.

2. To retrieve an excerpt of the map and display it on some page [Google 2011a].
Application: the spatial distribution of the samples origin is visually presented as pins in a map.

The first functionality has to be done by JavaScript web request (Ajax) while for the second the Google Maps API is offered. This API is an interface based on JavaScript to embed a map to a web page with the benefit of many interactivities as well as the possibilities to display own map layers (called 'overlays') like pins (called 'markers'), background maps and many more visual and functional features.

5.2.7. Google Visualization

This Google service is a collection of loadable JavaScripts with the purpose to visualise higher amounts of values and its statistical evaluation. If the input content can be arranged as a two dimensional table, a sublaying script will transform it into a specific chart. Therefore many different chart types are possible: simple tables, line charts, column charts, even a small version of the Google map can be requested. They all have in common that a very developed interactivity is already included and that the basic is always the input table. Because the complete creation process is predefined by the loaded script only certain amount of own adjustments can be taken. But it is enough for the most cases [Google 2011c].

5.3. System application flow

In this section the sequential development of the application and the communication among its components should be outlined. As figure 5.4 shows, the application flow can be subdivided into six modules.

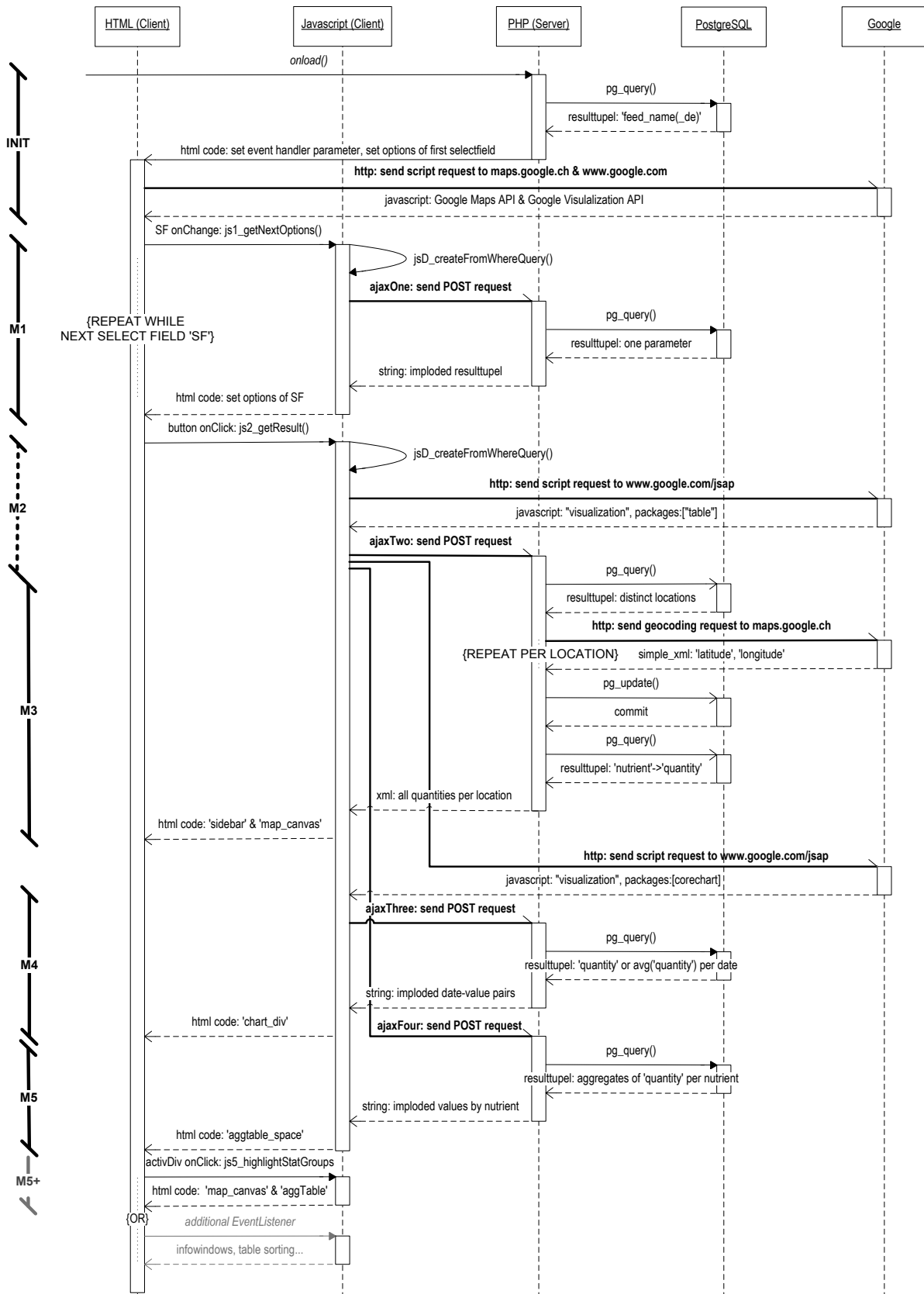


Figure 5.4.: UML sequence diagram of the complete Swiss Feed Database including all major components

5.3.1. Module INIT

The initiation of the application is done by loading the HTML start page. Before the page gets delivered to the client, the PHP server script runs a first database query to set the options of the first select field, which is the parameter *feed name*. Then the page gets loaded by the clients browser together with all the Jscripts embedded in the HTML header. This includes also the Google Maps and the Google Visualization API from the Google server. All this happens in the normal PHP-HTML loading sequence.

5.3.2. Module M1

This module concerns the filling of the next select field with the proper options. It is based on JavaScript and gets reiteratively triggered every time the event handler 'onChange' of the previous field is activated. In order to get the new options, PHP has to send a new SQL query including the previously chosen options as filter to the database to return the corresponding values. This is where Ajax (Asynchronous JavaScript and XML) pitches in the first time and requests for an additional PHP server script to run.

5.3.3. Module M2

The preparation and coordination of the application's output result is covered in this module. Again, it's based on JavaScript and gets triggered by the event handler 'onClick' of the *show result* button. It includes additional loading of Google Visualization scripts and, again, the generation of the input dependent SQL query, in order to retrieve all needed database tuples. But since the application's final output is covering six different operations (chapter 4) and therefore requires a combination of tuple sets, different SQL queries (different in SELECT and ORDER BY) need to be run this time. So, this module splits the one result operation into subparts by creating parallel Ajax requests, one per different SQL query.

5.3.4. Module M3

This module starts with PHP and includes also web requests to the Google server. It deals with the output 'sample enlistment' and 'Google maps' because these two parts are using one and the same tuple set (enlistment of all samples along with its nutrient values and its geographical position). To get this information, the requested PHP script queries the database, sends for every sample location with unknown latitude and longitude a geocoding request to the Google Maps GIS and then sends all information as a XML back to JavaScript and the waiting AjaxTwo. In JavaScript, the XML will be transformed into a Google Visualization table and a Google Maps map with a marker per sample.

5.3.5. Module M4

This module deals with the output 'diagram over time'. It has almost the same sequence as module M3 except that only a simple PHP script with one SQL query to the database is involved. But similar to the previous module, the tuple set of the requested distinct temporal values will then be sent to JavaScript and the waiting AjaxThree (this time as a concatenated string), where they will be transformed into a Google Visualization line chart. This module will be executed in parallel to the last module.

5.3.6. Module M5

This module has the same sequence as the module M4 and it is also executed in parallel to the previous two modules. This time it deals with the output 'table of aggregates' and needs a tuple set of several aggregation function per nutrient. These set will be sent to JavaScript and the waiting AjaxFour, again, as a concatenated string, where JavaScript will then transform it into a HTML table by using dynamic HTML.

5.3.7. Module M5+

This is not an own module, but an additional interactive JavaScript operation, applied on the HTML table of module M5. To every nutrient in the list there is assigned an event handler 'onClick'. The underlying function will then highlight the outliers (the sample which are lying outside of the 2sigma range) of the selected nutrient by setting the markers of the map to a specific colour. To do this, the HTML table with the average and standard deviation, the Google map with the markers and the Google sample table with the sample values have to be used together.

5.4. Database design

5.4.1. Objects and relations

The organisation of the database is developed to deal with the measurements of all nutrients/nutritive values inside of every sample in separate, so that later the quantity of the measurements can be queried, sorted, filtered and aggregated easily (fig. 5.5). This has to be done with less redundancy as possible.

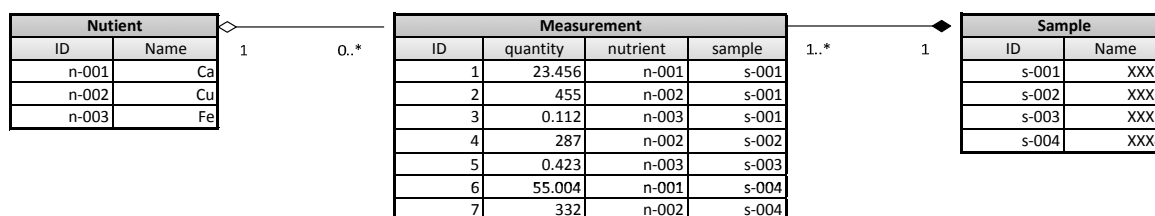


Figure 5.5.: Illustration of the composition of samples.

The presented solution will include the creation of a main table 'fact_table', where only the quantities of the measurements shall be stored (along with a duplication of the sample identifier for faster queries). The specific information of the corresponding nutrient and the sample, as well as all additional information, observed in the data input analysis (like sample treatment, geographical origin, temporal reference and more), will then be stored in separate dimension tables connected to the fact_table by foreign keys. These keys are only used to establish the connection between the tables and will be artificially assigned by the database without deducting them from a specific input value. In order to guarantee the assignment of an unique expression onto every entry, a sequences of auto incrementing numbers for every table should be used.

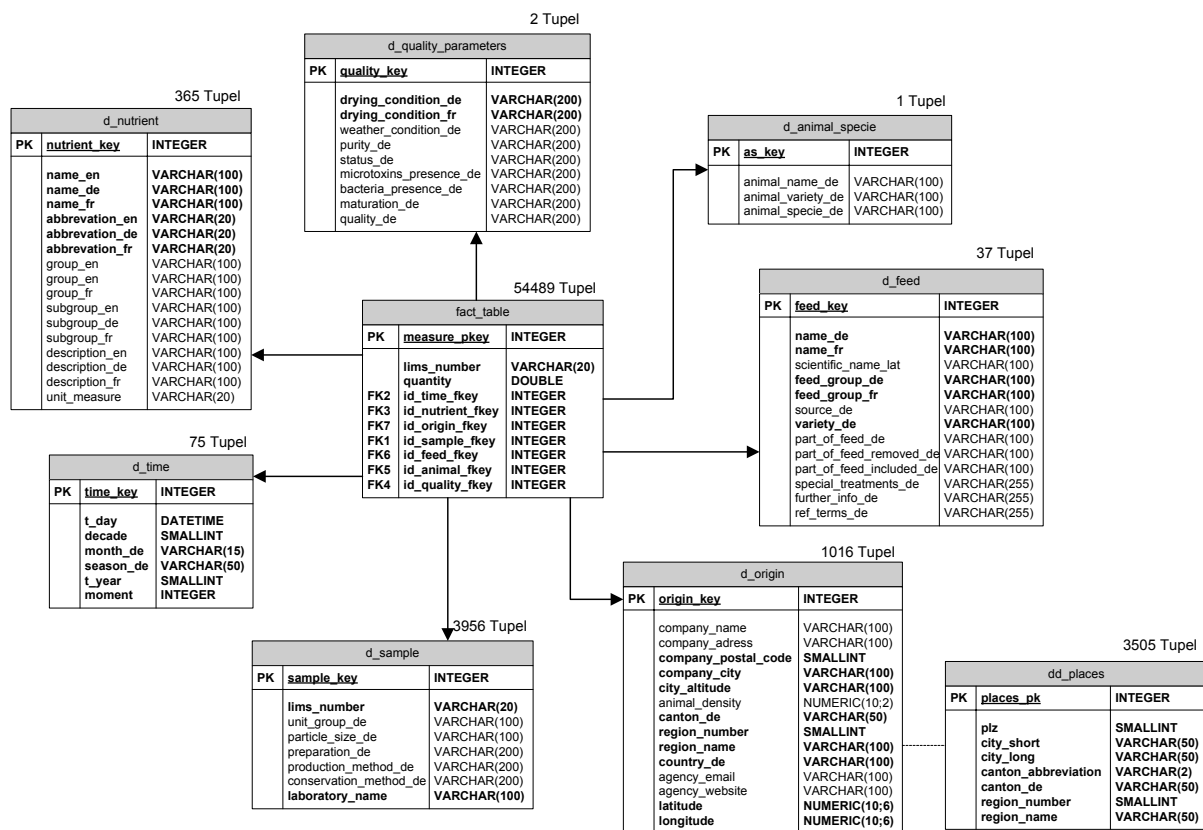


Figure 5.6.: Relational concept of the Swiss Feed. Bold: used attributes so far (the others contain only NULL). Also indicated on top of every table is the distribution of the test data into the tables.

The idea of this design is that the fact_table grows linear when loading new data, while the dimension tables grow only logarithmic because it stores distinct values. Figure 5.6 shows the complete relational concept.

- It can be seen that the fragmentation of the dimension tables is done topically and not with the purpose to erase all redundancy. This is why the model is only in the first normal form.
- It also can be seen, that the database design includes way more tables and attributes as located in the data input analysis. The reason is that the database design has been adapted to fit for an overall application, which is exceeding the bounds of this thesis.
- Finally, it should be mentioned that the online application should actually be available in at least German and French, but the received data input along with further instructions is not sufficient. So the basic concept has been designed for the mostly German expressions, whilst parallel language attributes have been planned only if corresponding inputs have been noticed (for simplification).

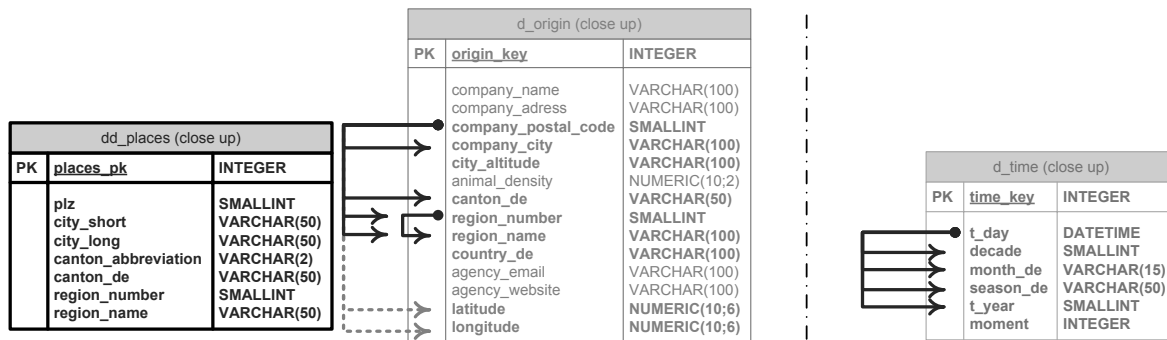


Figure 5.7.: A close up of the two tables `d_origin` and `d_time` with a illustration of the designed attribute hierarchy

5.4.2. Triggers and Views

Two of the dimension tables have been designed with some additional functionalities. Therefore they shall be discussed more detailed.

Case 'd_origin'

All information concerning the geographical origin of a sample is stored in a separate dimension table with name `d_origin`. But, since the data input only delivers some attributes (in most cases the postal code, an altitude level and a region code) a hierarchy can be established to derive additional attributes out of others with help of a static lookup table¹ (fig. 5.7). The best solution to solve this would be to include the trigger functionality of the used database system. Such a trigger would execute a predefined UPDATE SQL query as soon as a specified attribute gets changed. In this sense, if only a postal code gets inserted into the database, the corresponding city, canton and region could be looked up to complete the entry.

```

Example: Trigger function if 'postal code inserted or updated'
1 UPDATE d_origin
2   SET (CASE WHEN company_city IS NULL
3         THEN company_city = dd_places.city_name_long END),
4       (CASE WHEN canton_de IS NULL
5         THEN canton_de = dd_places.canton_de END),
6       (CASE WHEN region_number IS NULL
7         THEN region_number = dd_places.region_number END),
8       (CASE WHEN region_name IS NULL
9         THEN region_name = dd_places.region_name END),
10      country_de = 'Schweiz'
11 FROM dd_places
12 WHERE dd_places.plz = d_origin.company_postal_code;

```

A second triggered UPDATE function shall be established in this table in order to correlate the always indicated region number with the corresponding label.

¹An additional table that includes a complete list of postal codes together with additional static values

Example: Trigger function if 'region code inserted or updated'

```

1 UPDATE d_origin
2   SET  (CASE WHEN region_name IS NULL
3         THEN region_name = regions.region_name END),
4         country_de = 'Schweiz'
5 FROM (SELECT DISTINCT region_number, region_name
6       FROM dd_places) AS regions
7 WHERE regions.region_number = d_origin.region_number;

```

Case 'd_time'

A similar hierarchy should be used in the dimension table d_time (fig. 5.7): if the attribute 't_day', which expects a complete date, gets filled, the other attributes 'decade' (# of ten-day-intervals since the beginning of the year), 'month_de' (month as string), 'season_de' (season as string²) and year can be derived from it by using again a trigger.

Example: Trigger function if 'day inserted or updated'

```

1 UPDATE d_time
2   SET decade = 3*(EXTRACT(month from t_day)-1)+
3     ((CAST(EXTRACT(day from t_day) AS smallint)/10)+1)
4     t_month =
5     (CASE
6       WHEN EXTRACT(month from "day")='01' THEN 'Januar'
7       WHEN EXTRACT(month from "day")='02' THEN 'Februar'
8       (...)
9       ELSE NULL
10    END),
11    season_de=
12    (CASE
13      WHEN t_day < to_date(
14        EXTRACT(year from t_day)||'-06-21', 'YYYY-MM-DD')
15        AND t_day >= to_date(
16          EXTRACT(year from t_day)||'-03-20', 'YYYY-MM-DD')
17      THEN 'Frühling'
18      WHEN t_day < to_date(
19        EXTRACT(year from t_day)||'-09-23', 'YYYY-MM-DD')
20        AND t_day >= to_date(
21          EXTRACT(year from t_day)||'-06-21', 'YYYY-MM-DD')
22      THEN 'Sommer'
23      WHEN t_day < to_date(
24        EXTRACT(year from t_day)||'-12-21', 'YYYY-MM-DD')
25        AND t_day >= to_date(
26          EXTRACT(year from t_day)||'-09-23', 'YYYY-MM-DD')
27      THEN 'Herbst'
28      WHEN t_day < to_date(
29        EXTRACT(year from t_day)||'-03-20', 'YYYY-MM-DD')
30        OR t_day >= to_date(
31          EXTRACT(year from t_day)||'-12-21', 'YYYY-MM-DD')
32      THEN 'Winter'
33      ELSE NULL
34    END),
35    t_year= CAST((EXTRACT(year from t_day), smallint);

```

²Because the static meteorological seasons are not so well know, the common but year-dependent astrological season will be chose. But for easier computation the dates will be generalised.

An additional speciality of the table `d_time` is that all ever appearing timestamps in the measurement process are stored together in this one table relation. To separate them from each other by timestamp type, the attribute 'moment' is included to indicate this by code (tab. 5.1). This code is then used as filter option to create a database View for every timestamp type.

Example: Query to create Harvest day View

```

1 CREATE VIEW d_harvesttime AS
2     SELECT d_time.time_key, d_time.t_day, d_time.decade,
3           d_time.month_de, d_time.season_de, d_time.t_year
4     FROM d_time
5     WHERE d_time.moment = 1;

```

CODE	VIEW	ANNOTATION
1	Harvest time	- used in data input -
2	Sample time	- not use so far -
3	Arrival time	- not use so far -
4	Analyse time	- not use so far -

Table 5.1.: Summary of the view triggering attributes in the database table `d_time`.

6. Implementation

6.1. Introduction

In this chapter, processes and code excerpts to selected topics will be presented with comments and explanations. The topics are aligned chronologically in their execution. When it comes to the application and its final presentation of results, which is done in three parallel modules, the module M2 and M3 (sample enlistment and Google Maps) will be preferred as examples because they are covering all main topics.

6.2. Database

6.2.1. Extraction, Transportation, Loading (ETL) of data

In chapter 3.2.2 the composition of the input data was analysed and according to the detected parameters and subject areas, the database tables and its relations were designed. Now it should be discussed how the data was processed from the spreadsheet delivery into the database relations. Therefore two problematic observations have to be considered:

1. Different internal organisation

The spreadsheet aligns all measurement values from the same sample next to each other in one row, whilst the database is designed to store each measurement as own fact_table entry (remember fig. 5.5).

- This will be solved by loading the complete spreadsheet in his original state into an intermediate database table. Using SQL, the information will then be spread into its final fact or dimension table.

2. Inconsistency

One major point during the analysis was that the spreadsheets are very inconsistent: every new data delivery has own column names and also the composition of the columns can differ; especially the feed description has no real namespace; several mistakes were detected so far.

- This leads to the conclusion: every spreadsheet needs to be treated in separate.

Step 1: Preparing the spreadsheet for data extraction The first thing to do is the inspection of the spreadsheet. If possible, the major inconsistencies should now be detected with the spreadsheet's filter operations and corrected with its formula possibilities, *drag and drop* or *search and replace*. In the best case all processable information is separated to its own column.

Important: Cells containing a "0" but meaning a NULL needs to be emptied. Otherwise additional tuples with value 0 will be inserted into the database.

Example: Divide the column with material description into treatable subparts

In this case, several excel formulas were used to extract the maximum of processable parts out of the very mixed column 'Sample Reference' (fig. 6.1). The ventilation facts were derived from the column 'Art' (= material code) in the same way. The gained result is then available according to the previously developed description structure (fig. 3.5).

Sample Reference	feed group (de)	feed group (fr)	feed name (fr)	feed name (de)	Art	ventilation
Dürrfutter belüftet	Dürrfutter				1	belüftet
Foin-Regain ventilé			Foin / Regain	Heu / Emd	1	belüftet
Dürrfutter belüftet	Dürrfutter				1	belüftet
Foin ventilé			Foin	Heu	1	belüftet
Regain ventilé			Regain	Emd	1	belüftet
Foin de Sol			Foin	Heu	2	unbelüftet
Foin Séchoir balles Rondes			Foin	Heu	2	unbelüftet
Ensilage d'herbe 1 ére Coupe	Grassilage	Ensilage d'herbe			3	
Ensilage d'herbe 2 et 3 éme Coupe					3	
Dürrfutter belüftet	Dürrfutter				1	belüftet
Dürrfutter belüftet	Dürrfutter				1	belüftet
Dürrfutter belüftet	Dürrfutter				1	belüftet
Foin balles rondes			Foin	Heu	2	unbelüftet
Grassilage	Grassilage				3	
Dürrfutter belüftet	Dürrfutter				1	belüftet

Figure 6.1.: Example of a spreadsheet preprocessing to extract workable information concerning the sample's content.

Extraction: Excel formula to extract French feed name

```

1
2 Formula structure: If([condition];[do if true];[do if false])
3 =IF (ISERROR (AND (SEARCH ("foin"; [field]); SEARCH ("regain"; [field]))); FALSE);
4   "Foin / Regain";
5   IF (ISERROR (SEARCH ("foin"; [field])); FALSE);
6     "Foin";
7     IF (ISERROR (SEARCH ("regain"; [field])); FALSE);
8       "Regain";
9       ""
10  )
11 )

```

Step 2: Import spreadsheet as a text file into dummy-table To import the spreadsheet into the PostgreSQL database an additional step by using an intermediate file format is required. The applied solution is the tab-delimited text file, which can be directly chosen as output format in excel ¹. It stores the spreadsheet by dividing every former table row by new-line (\n) whilst its cells are separated by tab (\t). Subsequently, the text file could be imported directly into a database table by the PostgreSQL operation 'COPY FROM'. In order to do this, the required input file needs to be placed on the same server as the database [PgSQL 2010, "SQL COPY"]. In the context of the theses, the database system installed on the server of the Institute of Informatics at the University of Zurich was used with only delimited access rights. Therefore an alternative by writing an interface program in Java, which executes an INSERT query by row, was established.

¹Another solution would have been the CSV (*comma-separated values*), but this format seem not to include special characters.

In this alternative, the text file was used as an input file. After creating a table with all needed columns and data types, one row (equal to one row in the spreadsheet) after each other was read in, formulated into an INSERT query and was then sent by a client-server-connection to the database. The fact that every former cell value is delimited by a tab was used during the formulation activity to apply the proper format: a double tab was replaced by the string `'NULL'`, and the single tab by `'`. After adding some static values in the beginning and at the end, valid INSERT query was ready to send.

```
Input:   Dürrfutter\tbelüftet\tHeu / Emd\t(4)\t0\t\t> 1000
        m\t0\t3\t4\t\t257.52
Output:  INSERT INTO dummytable VALUES ('Dürrfutter',
        'belüftet','Heu / Emd','(4)','0',NULL,'> 1000 m','0',
        '3','4',NULL,'257.52');
```

Important annotation: This solution will generate errors when French expressions containing the apostrophe (Unicode: U+2027) are involved, e.g. "L'Europe". The best solution is to replace every apostrophe in the input data by the single right quotation mark (Unicode: U+2019)². By doing this, not only the current error will be prevented, but also the ones in the application, when working with JavaScript or PHP.

Step 3: Import new data into dimension tables Now that all information is included in the database, the spreadsheet structure can easily be decomposed to add the data to its corresponding final tables. The first relocation will deal with the data to be cataloged inside the "dimension tables". This includes also the headers of the nutrient columns (stored in table `d_nutrient`). But, for later discussed reasons, it will exclude the sample identifier or any other of the future `d_sample` attributes (for instance laboratory name).

Since the dimension tables are all storing distinct tuples, new values cannot simply be appended. The insert query has to check first if the current tuple combination is already stored.

— Example: PostgreSQL INSERT query for `d_origin` with check functionality —

```
1  INSERT INTO d_origin (company_postal_code, city_altitude_level,
2     city_altitude, region_number)
3  SELECT newData.*
4  FROM (SELECT DISTINCT
5     /* All delivered data columns for d_origin: */
6     plz AS postal_code,
7     hoehenstufe AS altitude_level,
8     altitude AS altitude,
9     region AS region_number,
10    FROM excel_table) AS newData
11 WHERE NOT EXISTS (SELECT 1
12    FROM d_origin
13    WHERE /* --1-- Delivered columns: */
14     d_origin.company_postal_code = newData.postal_code
15     AND d_origin.altitude_level = newData.altitude_level
```

²The right single quotation mark is the typographically correct alternative to apostrophe. It is even advised to use instead in German, French and English [e.g. <http://en.wikipedia.org/wiki/Apostrophe#Unicode>].

```

16     AND d_origin.city_altitude = newData.altitude
17     AND d_origin.region_number = newData.region_number
18     /* --2-- Not delivered columns: */
19     AND d_origin.company_name IS NULL
20     AND d_origin.company_address IS NULL
21     AND d_origin.animal_density IS NULL
22     AND d_origin.agency_email IS NULL
23     AND d_origin.agency_website IS NULL
24 );

```

The example shows that the columns, which are not covered in the delivery, need to be referred as well, for the simple reason that they necessarily need to be NULL to find the only correct corresponding match.

COMP. NAME	COMP. ADDRESS	COMP. POSTAL CODE	CITY ALTITUDE
Company A	Somewhere Street 101	8000	< 600
NULL	NULL	8000	< 600

Table 6.1.: Illustration of the importance of NULL values for tuple specification. Bold are the attributes, which are covered in the new data delivery.

One could say that always the complete tuple needs to be checked. But there are three exceptions:

1. The columns, which are dependent on a hierarchy. They should not be checked, since their content changes after this loading process. But because they are dependent on a master column it is enough to only check the master's content.
2. All columns in the table `d_nutrient` besides the attribute 'abbreviation'. This table has the special purpose of being a glossary, by storing also information about the nutrients real name or its measure unit. That's information, which were delivered separately and is not included, but even so valid in the official deliveries. But again, it is sufficient to check the abbreviation entry only.
3. All attributes in a parallel languages to the applied language. The goal is that the database contains in the end only attribute with according translations. The data delivery will not come with such a requested resolution. If a data entry includes no translations, the identical tuple but with translation should of cause not be despised when checking for an existing entry.

Step 4: Import new data into fact table and dimension table `d_sample` Finally, the import can be completed by loading the last missing pieces: the information concerning the quantities of the measured nutrients. And as mentioned earlier, the goal is to store the measurements of all nutrients equally in the `fact_table`, together with foreign keys referring to all the surrounding (and now up-to-date) dimension tables. But there is one dimension table left: `d_sample`.

The reason for not updating this particular table until now was the following: the analysis of the input data showed that only 50% of all delivered samples posses a sample identifier (LIMS

number). In the case that no such identifier is provided, the primary key of the `d_sample` table (`sample_key`) should be used instead and stored in `d_sample.lims_number` as well as in `fact_table.lims_number`. The only way of covering this without losing the interconnection between sample and measurement, is to load every sample in separate and divide its components into the `d_sample` and the `fact_table`:

Pseudo code

```

1 For every row in dummy_table {
2     If lims_number exists in dummy_table{
3         Insert sample data into d_sample with dummy_table.lims_number as lims_number
4     }else{
5         Insert sample data into d_sample with d_sample.sample_key as lims_number
6     }
7     Treat every nutrient column in the dummy table one by one {
8         For every valid measurement {
9             In all dimension tables{
10                Check for the one matching tuple and get its key;
11            }
12            Insert all keys along with the measurement's quantity and the lims_number
13            into the fact_table;
14        }
15    }
16 }
```

Example: PostgreSQL queries for INSERT in table `d_sample` and `fact_table` The procedural loading according to the declaration above was as well implemented in a Java program. This program expects as input a list of the abbreviations of all nutrients appearing in the dummy table. It's assumed that the names of the nutrient attributes in the dummy tables are identical to the abbreviation. In this case the two variables of query 4, [*dummyNutrientCol-Name*] and [*abbreviation*], can be set by the same input. The Java program will execute the first query once and repeats the other three queries [*sampleAmount*]-times.

Query 1

```
1 [sampleAmount] = SELECT COUNT(excel_key) FROM excel_tbl;
```

Query 2

```

1
2 Annotation: '...seq' is the sequence of the auto-incrementing key number of a table
3 [sampleKey] = SELECT nextval('d_sample_sample_key_seq')
4 AS next_sample_key;
```

Query 3

```

1 INSERT INTO d_sample(sample_key, lims_number,
2     unit_group_de, particle_size_de, preparation_de,
3     production_method_de, conservation_method_de, further_info_de)
4 SELECT
5     [sampleKey],
6     (CASE
7         WHEN excel_tbl.lims_number IS NULL
8         THEN [sampleKey]
9         ELSE excel_tbl.lims_number
10    END) AS lims_number,
11     NULL, NULL, NULL, NULL, NULL, NULL
12 FROM excel_tbl
13 WHERE excel_key = [ascending number];
```

```

----- Query 4 -----
1  INSERT into fact_table (lims_number, quantity, id_time_fkey,
2      id_nutrient_fkey, id_quality_fkey, id_animal_fkey,
3      id_feed_fkey, id_origin_fkey, id_sample_fkey)
4  SELECT lims_number, [dummyNutrientColName], time_key, nutrient_key,
5      quality_key, as_key, feed_key, origin_key, [sampleKey]
6  FROM excel_tbl, d_time, d_nutrient, d_quality_parameters,
7      d_animal_specie, d_feed, d_origin, d_sample
8  WHERE
9      excel_tbl.[dummyNutrientColName] IS NOT NULL
10
11     Join to d_sample: is covered by the sequential number
12     AND d_sample.sample_key = [sampleKey]
13
14     Join to d_nutrient: check the abbreviation only, but in proper language
15     AND d_nutrient.abbreviation_de=' [abbreviation]'
16
17     For joint to all other dimension table:
18     Check every attribute that is not a slave field in a hierarchy
19     AND...
20
21     Argumentation if the field is not available in dummy table (was not delivered)
22     [ [d_table field] IS NULL ||
23     Argumentation if the field is available in both tables and certainly no null value will appear
24     [excel_tbl field] = [d_table field]||
25     Argumentation if the field is available in both tables but null values can appear
26     (CASE
27         WHEN [excel_tbl field] IS NULL
28         THEN [d_table field] IS NULL
29         ELSE [excel_tbl field] = [d_table field]
30     END)) ]*

```

6.3. Application

6.3.1. Generating dynamic SQL

Examples follow the module: INIT, M1

Topic appears also in module: M2

In the application, SQL queries are applied to the database either to get the options of the next select field or to finally get the requested results. Both kinds of queries have in common that their WHERE- and FROM-part needs to be filled dynamically using the previously selected options.

```

----- Example: SQL query for the next select field ('canton[]') -----
1
2  The database attribute of the next select field's parameter:
3  SELECT canton_de
4
5  All involved tables (fact_table is always involved because it is joining all dimension tables):
6  FROM fact_table, d_feed, d_nutrient, d_origin
7
8  The join conditions between every used dimension table and the fact_table:
9  WHERE id_feed_fkey = feed_key

```



```

10     AND id_nutrient_fkey = nutrient_key
11     AND id_origin_fkey = origin_key
12
13 Filter options of selection: case one or more options are selected - one of it requests to display
14 all empty feed names. It could also be only 'IS NULL' or only 'feed_name_de IN (...)'.
15 AND (feed_name_de IN ('Heu','Emd') OR feed_name_de IS NULL)
16
17 Filter options of selection: case all options are selected then no filter condition should be created.
18 This is solved by an artificial first option 'ALL': if(first selected value == 'ALL'){ - do nothing - };
19 /* no filter condition on d_nutrient.name_de... */
20
21 Order by select attribute:
22 ORDER BY canton_de;

```

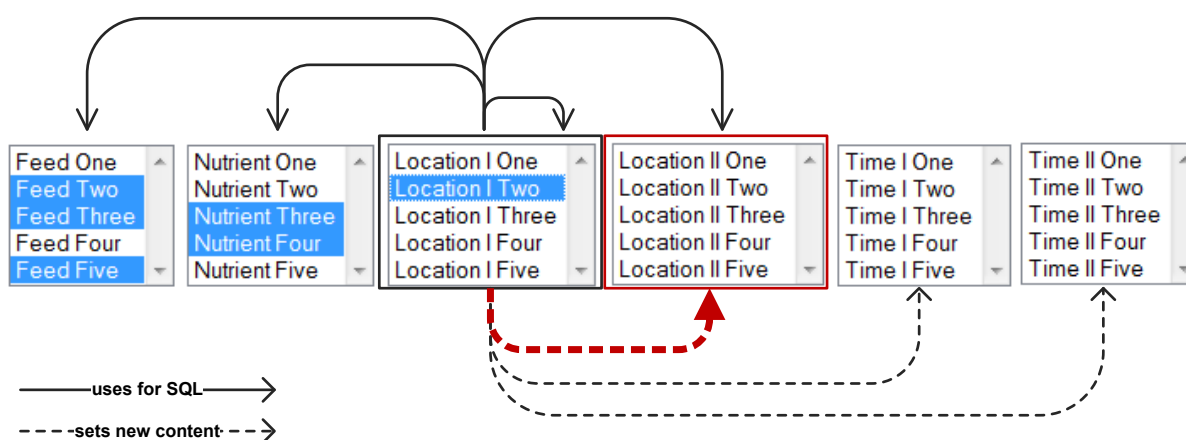


Figure 6.2.: Schematical example of the references between the <select> fields

Figure 6.2 illustrates the example a little: to update its next select field, the current field need the information about the selected options, the corresponding database table, its join to the fact_table and the parameters attribute name of itself and all previous select fields. In addition it also needs the database table, its joint to the fact_table and the parameters attribute name of the next field.

The thin dashed lines are referring to the condition that a user can always go back and restart its selection in a previous field. In such a situation the following select fields need to be set to initiation state (no options) and therefore the current field needs to know, which other select fields, it is supposed to change.

Initiation with PHP 5

In order to generate dynamic SQL queries or delete the following options, every HTML element, which is triggering a corresponding Jscript (= all select fields, the last one executes the script by the 'show result' button), needs to own the previously mentioned information. The information itself can be distinguished into the 'variable' information, which changes in runtime (like the options and its selection) and the 'static' ones that will stay the same at least inside of the user's session (select field names and all database relating specifications). To submit all this in a manageable and ordered way, three thoughts led to a solution:

1. By using DOM-scripting, a <select> element and its selected options can be accessed in JavaScript runtime if the name or the id of the HTML element is known.
2. The distribution of all needed information can be done at the initiation of the application, since the required content of every element can be formulated in 'static' values only, if the first thought is properly applied.
3. The information can be injected, ordered and distributed by PHP before loading it on the client's side, especially with the new object functionality of PHP 5.

Step 1: Manage the information input Every <select> unit will be filled with the information concerning its own database origin only. Therefore a PHP class called 'SelectField' has been created (fig. 6.3), one object per unit is instantiated and the requested information is assigned by setter-methods.

PHP::SelectField
-fieldName -sql_columnName -sql_tableName -sql_joinToFactTable -nextSF -prevSF
+constructor() +setters() +doInitialQuery() +printField() +printLastField() +printQueryButton()

Figure 6.3.: The outline of the PHP class 'SelectField'

```

Example: 1-Inject the requested input information in PHP5
1 <?php
2 require("class-selectfield.php");
3
4 $feed = new selectField("feed[]"a);
5 $feed->setOriginTable("d_feed");
6 $feed->setOriginColumn("d_feed.name_de");
7 $feed->setJoinToFactTable("id_feed_fkey = feed_key");
8 $drying = new selectField("drying[]");
9 $drying->setOriginTable("d_quality_parameters");
10 $drying->setOriginColumn("drying_condition_de");
11 $drying->setJoinToFactTable("id_quality_fkey = quality_key");?>

```

As next, the dependency between the fields is set with the operation setNextField():

```

(Operation setNextField())
1 <?php
2 setNextSF($field) {
3     this->nextSF = $field;
4     $field->prevSF = $this;
5 }?>

```

^aOne can process the HTML <select multiple/> element as an array with its multiple selected values if the name contain the array brackets "[]" [<http://www.php-faq.de/q-formular-select-multiple.html>].

Example: 2-Link select fields in PHP5

```
1 <?php
2 $feed->setNextSF($drying);?>
```

Step 2: Create HTML elements The method `printField()` of the class `SelectField` is producing the final HTML code of a `<select>` element. But before that, it is distributing links and the corresponding database info as discussed above (fig. 6.2). In doing so, the linked `SelectField` objects are used to iterate all previous and all following select fields by using the `$this->next/prevSF` connection.

Since the server scripted `SelectField` object are no longer accessible after loading the page on the client, all required information need to be hand over to the HTML code where most of them are assigned to a corresponding JavaScript function. As a medium between server and client script, a string, if needed with the concatenated values, is used.

Example: Assign the new managed information to HTML and JavaScript

```
1 <?php
2 echo '<select name="'. $this->fieldname.'" id="'. $this->sql_columnName.'"
3 onChange="function('.$this->nextField->fieldname.', '.$concatPrevFields.',
4 '.$concatNeededTables.', '.$concatNeededJoins.',
5 '.$concatFollowFields.')">';?>
```

Annotation: To collect all needed database tables and join conditions to the fact table, an intermediate step was taken by inserting every new entry during the iteration into an associative array³:

Example: Insert two values into a associative PHP array

```
1 <?php
2 $array_tables[$visit->sql_tableName] = $visit->sql_joinToFactTable; ?>
```

Later, the array was iterated itself to concatenate all the entries to a string:

Example: Iterating the associative PHP array

```
1 <?php
2 foreach($array_tables as $key => $value){
3     $concatNeededTables.= "-sep-".$key;
4     $concatNeededJoins.="-sep-".$value;} ?>
```

With this step, tables and joins that are used multiple times (e.g. parameter 'canton_de' and 'altitude' are both stored in the table `d_origin`) are only mentioned once in the string.

Special: 'first' and 'last' select field The specialty of the first select field is that its options are filled at the beginning. Therefore a first SQL query needs to be established and run before the page gets loaded by the client. This is covered in the method `doInitialQuery()` of the class `SelectField`. This operation should be called right after all setters have been used. The last select field has the advantage that no following fields need to be filled and reset. In addition, the JavaScript function is not supposed to start on the event handler `onChange` in the `<select>` element, but `onClick` to a button. That's way there are the separate operations `printLastField()` and `printQueryButton()`, used to print a `<select>` element only with name and id, and a button, which takes over the event handler with the limited Jsript function.

³An array where each ID key is associated with a value [Refsnes Data 2011, "PHP Array"].

Step 3: Setting HTML element at proper place The positioning of the `<select>` elements generated by the PHP `SelectField` objects is done by placing the print operation `printField()`, `printLastField()` and `printQueryButton()` at this point in the HTML code where the return element should lie.

Example: Interaction of HTML and PHP code

```

1 <html><div class="selection">
2     <form name="testForm">
3         <div>
4             <h3>Feed</h3><br>
5             <?php $feed->printField(); ?>
6         </div>
7         <div>
8             <h3>Drying</h3><br>
9             <?php $drying->printLastField(); ?>
10            <?php $drying->printQueryButton(); ?>
11        </div>
12 </form></div></html>

```

By applying these three presented steps, it is possible to create a interface from the HTML to the underlying database, which is easily adaptable in matter of the selection's parameter choice. Only the required information and relationships have to be set once and then the server is producing an HTML document with the following specifications:

- Multiple `<select>` elements and one button are included. The first `<select>` element has additionally several `<option>` already displayed.
- All elements are addressable by specific name.
- These names are distributed to the fellow elements that are in need to address this field.
- Event handlers and Jscript function (with corresponding parameters) are initiated.

SQL generation with JavaScript

When an event handler is triggered and a SQL query has to be assembled, the corresponding Jscript function is processing its input parameters or uses them to collect additional values by addressing the fields with DOM-Scripting.

SELECT

The needed column name is stored in the `id` attribute of the next `<select>` element. The name of this element was handed over as the first input parameter (see previous page).

Example: Using `param1` to get attribute from `<select>` field

```

1 var colName = document.forms["testForm"].elements[param1].id

```

FROM

The needed tables were handed over in form of a concatenated string as the third input parameter.

Example: Prepare and use information in `param3`

```

1 var array_sql_table = param3.split("-sep-");
2 /* then iterate the array with for loop and add every entry
3 to the string "FROM fact_table" divided by a comma.*/

```

WHERE

The needed joins were handed over in form of a concatenated string as the fourth input parameter.

```
Example: Prepare and use information in param4  
1 //same as in FROM part...
```

The needed filter options have to be read from the corresponding <select> elements. Their names were handed over in form of a concatenated string as the second input parameter.

```
Example: Prepare information in param2  
1 var array_inputField = param2.split("-sep-");
```

With this name, the corresponding options can be addressed by DOM-Scripting using the following expressions. The complete algorithm to generate the SQL WHERE IN part is arranged in the fig. 6.4 as a Nassi Shneiderman diagram.

Get amount of options in <select>:

```
1 document.forms[formname].elements[selectname].length;
```

Address one <option> element:

```
1 document.forms[formname].elements[selectname].option[index];
```

Get the value of an <option>:

```
1 document.forms[formname].elements[selectname].option[index].value;
```

Get the option's select status (return value: true or false):

```
1 document.forms[formname].elements[selectname].option[index].selected;
```

6.3.2. Asynchronous client-server communication

Examples follow the module: M2-M3

Topic appears also in module: M1, M2-M4, M2-M5

In the previous chapter, the topic of generating SQL queries during JavaScript runtime was discussed. The next step would be to reactivate the server and let him run the new query on the database. As mentioned in section 5.2.5, the Ajax object can be used to establish such a connection from the client (JavaScript) back to the server, to call a server script (PHP) and to exchange information (in JavaScript runtime).

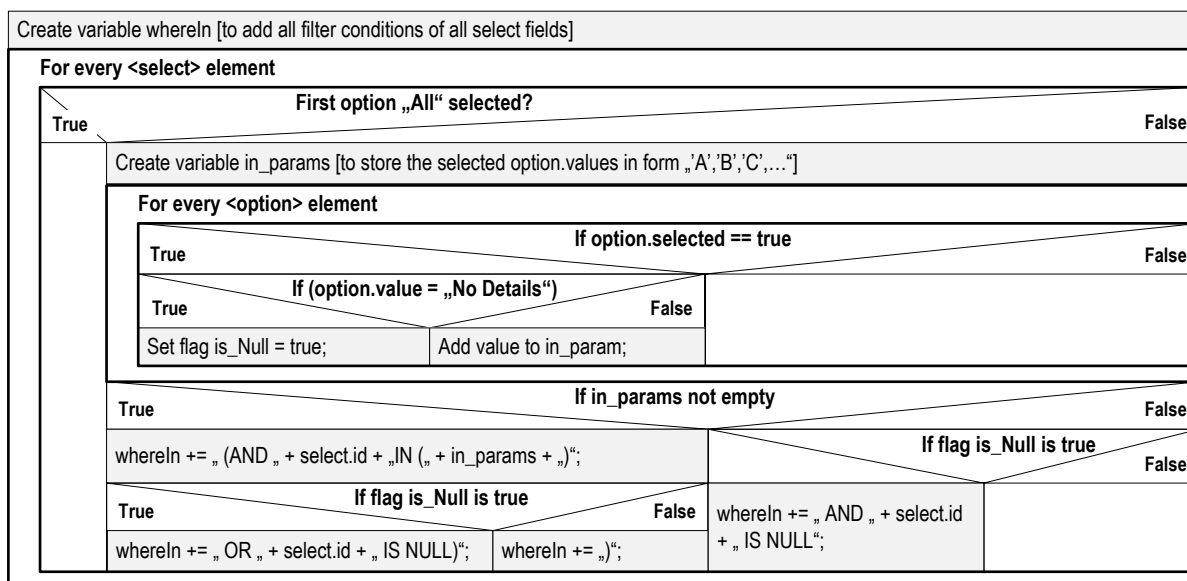


Figure 6.4.: Modeling of the 'generate SQL WHERE IN part' algorithm

Step 1: The allocation of an Ajax object All modern browsers (IE7+, Firefox, Chrome, Opera, Safari) include Ajax, which can be allocated as a new *XMLHttpRequest* object. Yet, IE 5 and 6 support the same functionality but named as *ActiveObject* [Refsnes Data 2011, "AJAX"]. To cover all possibilities depending on the client's browser, the following Jscript function is used to establish a new Ajax:

Example: 'Create Ajax' function

```

1 function jsB_getHTTPObject() {
2   if (window.ActiveXObject) return new ActiveXObject("Microsoft.XMLHTTP");
3   else if (window.XMLHttpRequest) return new XMLHttpRequest();
4   else {
5     alert("Your browser does not support AJAX.");
6     return null;
7   }
8 }

```

Example: 'Create Ajax' call

```

1 var g_ajaxTwo = jsB_getHTTPObject();

```

Step 2: Send request A request will be sent by HTTP communication using a URL. Depending on what information is packed in the URL, there are two different connection types [Refsnes Data 2011, "AJAX"]:

GET: To establish a connection, the server script path and all required variables are concatenated to the URL and send at once. This type is cheaper and faster.

POST: To establish a connection, only the server script path is included in the URL. The additional variables are sent as a concatenated string in separate after the connection stands. This type is more secure and has a limitless variables space.

In the Swiss Feed Database online application, only POST connections are used to guaranty that even the longest query can be communicated completely to the server script.

Example: Create asynchronous connection on client side view

```

1
2 Open connection with parameters 'connection type', 'server script path', 'asynchronicity':
3 g_ajaxTwo.open("POST", "ajax-pg-result-google.php", true);
4 Set the client's request header (for type, using default value):
5 g_ajaxTwo.setRequestHeader("Content-Type",
6     "application/x-www-form-urlencoded");
7 g_ajaxTwo.setRequestHeader("Content-length", sql_from_where.length+6)a;
8 g_ajaxTwo.setRequestHeader("Connection", "close");
9 Send the variables in form varName=Value:
10 g_ajaxTwo.send("query="+sql_from_where);
11 Formulate a callback function, which catches the return values in the
12 asynchronous connection (for the involved Ajax readyStates see fig 5.3 in section 5.2.5):
13 g_ajaxTwo.onreadystatechange = function() {
14     /* so something with result */
15 }
16 Instead of some code, also a Jscript function can be assigned and even parameters can be handed over.
17 g_ajaxTwo.onreadystatechange = function() { js3_setMapList(/*param*/);}

```

With this code, a proper Client-Server connection has been generated and variables have been handed over.

Step 3: Run a Ajax triggered PHP server script The PHP file, called in the ajax.open() function, is a stand-alone script that gets activated by Ajax, runs through completely, returns a result and then terminates. This specific Ajax-called script is dependent on the input of the previously sent variable, which it can address with help the defined *varName*. The result of the server script can then be simply committed back to Javascript by "printing" it out, because the print command *echo* passes a value to the next lower surroundings:

- Case "PHP fragments in HTML code": HTML catches echo output
- Case "PHP script in Ajax surroundings": JavaScript catches echo output

Example: Create asynchronous connection on server side view

```

1 <?php
2 if(isset($_POST['query'])){
3     Read in the value part of varName=value:
4     $sql_from_where = $_POST['query'];
5     Do something with it:
6     /* add some SELECT, GROUP BY, ORDER BY to the $sql_from_where,
7     run query(ies) and organise the result tuple for output */
8     Set the server's request header for back transmission (only necessary in case of $resultXML):
9     header("Content-type: text/xml");
10    Returning the result back to JavaScript and the waiting Ajax:
11    echo [$resultText || $resultXML];
12 }?>

```

Step 4: Receive results on client side Whilst PHP was running its script, Ajax was waiting but constantly checking the progress. Anytime its readyState changed (for Ajax state conditions see fig. 5.3), the callback function `js3_setMapList()` was called, but only if the PHP script is terminated and the readyState changes to 4: *request finished and response is ready*, the code gets executed.

^aThe '+6' is used, because the variable name "query=" will later be added to the string

Example: Create asynchronous connection on client side view (contd.)

```

1 function js3_setMapList() {
2     check if final state was reached:
3     if(g_ajaxTwo.readyState == 4){
4         Load the result:
5         - As a string
6         var resultText = g_ajaxTwo.responseText;
7         - As a simple XML object
8         var resultXML = g_ajaxTwo.responseXML.documentElement;
9         /* do something with it */
10    }
11 }

```

Since we are communicating over HTTP, all information is transmitted as a string. This was the case when sending a request and variables by URL. This is also the case when the server returns its results to the client. This means that the server needs to arrange its results to a string before sending it back to the client. This are the possibilities used in the Swiss Feed Database to pass information between PHP and JavaScript:

A simple concatenated string

Used in module M1: 1dimensional result tuple (one column only).

Example

```

1 var string = "option-sep-option-sep-option";
2 var array = string.split("-sep-");

```

A nested concatenated string

*Used in module M4 & M5: 2dimensional result tuple (multiple columns * multiple rows).*

Example

```

1 var string = "nutrient-s-avg(quantity)-s-stddev(quantity)-sep-
2             nutrient-s-avg(quantity)-s-stddev(quantity)";
3 var array1 = string.split("-sep-");
4 var array2 = array1[i].split("-s-");

```

Use of the Expendable Markup Language (XML)

Used in module M3: 3dimensional result tuple (two queries with 2dimensional result tuples are nested with each other in a relationship of 1:N).

Example

```

1 Var string = "<xml>
2     <sample id=lims_number plz=company_postal_code city=company_city
3     canton=canton_de lat=latitude long=longitude>
4         <value abbreviation=abbreviation_de quantity=quantity/>
5         <value abbreviation=abbreviation_de quantity=quantity/>
6         <value abbreviation=abbreviation_de quantity=quantity/>
7     </sample>
8     <sample id=lims_number plz=company_postal_code city=company_city
9     canton=canton_de lat=latitude long=longitude>
10        <value abbreviation=abbreviation_de quantity=quantity/>
11        <value abbreviation=abbreviation_de quantity=quantity/>
12    </sample>
13 </xml>"

```


Ajax supports the possibility to process XML documents. Therefore the echoed result needs to be read in with `ajax.responseXML.documentElement`, which compiles the previous string into a XML object. This object can then be addressed with DOM-Scripting functionalities.

```

1  XML DOM-scrip.: Create array of specific XML tag
   var resultArray = resultXML.getElementsByTagName("sample");
1  XML DOM-scrip.: Address attribute of secific XML tag
   var postalCode = resultArray[i].getAttribute("plz");

```

6.3.3. Geocode locations with Google Maps service in PHP

Examples follow the module: M2

Topic appears also in module: -

This section discusses how, where and why the geocode service of Google Maps was included into the application. Geocoding describes the process to assign an absolute geographical references, typically latitude and longitude, to a location, which was available only in a descriptive way (e.g. city name). To do this, a database with cross references between the descriptions and the geographical references is needed. The Google Maps geocode service is now offering to use the Google Maps GIS database with its worldwide high resolution for this purpose.

The reason for using this service at all, is that an absolute geographical reference is needed to mark a specific location on a map. At the current state of data input, the most detailed geographical information is the postal code assigned to almost every sample. This information can easily be geocoded to mark the corresponding city by coordinates. A better input would be the address of a farm/laboratory or the latitude and longitude of the agricultural fields, but at this point, the postal code is the highest resolution and the code is aligned to deal with it.

The geocoding operation takes place in the PHP script `ajax-pg-result-google.php`, which was triggered by `ajaxTwo` to run. The reason why applying it on the server and not including it into JavaScript, was:

1. To generate as less communication costs on the client side as possible.
2. To store the geocode result back into the database, so that every sample has to be geocoded only once.

The Google Maps geocode service is handled as a web request, which means that its core is another Ajax request (see section 5.2.5). The descriptive geographical information will be transmitted according to the GET connection type, where all variables are suspended to the URL.

```

1  Example: Establishing a geocode request
   <?php
2   Specify the server according to the country for better hits (first priority: Switzerland):
3   define("MAPS_HOST", "maps.google.ch");
4   Create URL with script path, declaration of return type (xml) and declaration if active GPS is used (sensor).
5   $base_url = "http://" . MAPS_HOST . "/maps/api/geocode/xml?sensor=false";
6   Generate a geocode input without empty space: "8051+Zürich,+Zürich,+Switzerland"
7   $address = $plz + "+" + $city + "," + $canton + ",+Switzerland";

```

```

8   Add varName and variable to URL:
9   $request_url = $base_url . "&address=" . urlencode($address);
10  Send synchronous request (send and receive in one):
11  $xml = simplexml_load_file($request_url) or die("url not loading");?>

```

The response gets delivered as XML because it contains a lot of specifications. Next to a process status, there are the actual result values. But these results are covering not only the latitude and longitude; there are information about alternative writing, neighbourhood, altitude and even political associations. In addition, there can be more than one hit onto the geocode request ⁴, so everything gets wrapped in a deeply nested XML document [Google 2011b, #XML: with example of result XML]. In PHP every XML tag can be addressed by following its internal path from the documents root, one just has to know where it lies.

Example: Address XML elements in PHP

```

1  <?php
2   if ($xml->status == "OK") {
3     $lat = $xml->result->geometry->location->lat;
4     $long = $xml->result->geometry->location->long;}?>

```

This so far presented functionality was applied in the application to geocode the location of every involved sample, if not yet available in the database. Therefore many requests were sent whilst iterating all samples. However, therein lays a dilemma: first, the requests of the Google Maps geocode service are delimited to an upper bound of 2'500 per 24 hours [Google 2011b, #Limits]. This can be solved by storing back the result values in the database so that less and less request will be sent. Secondly, when sending more than one request, they need to have a temporal gap of unknown and variable size between each other or the geocoding will return the status "query over limit". For this case a sample's request will be resent until it is successfully geocoded or officially not geocodable (status: "zero result"). In addition, a break will be made between two requests, which will be enlarged every time a geocoding failed. Fig 6.5 shows the complete geocode algorithm.

6.3.4. Embed a map of Google Maps with information from the database

Examples follow the module: M3

Topic appears also in module: -

Google Maps seems to be the most used geographic information system in the world and its primary functionality is the mapping. In this case Google provides an interactive map, which any web page developer can embed into his code easily.

Step 1: Embed Google Maps mapping functionality The mapping functionality of Google Maps bases completely on JavaScript. Therefore the precondition is to load the Google Maps API (V3), which is a set of all required Jscripts, from the Google server to the client. This will be done by a script import command in the HTML header, just like the Swiss Feed Database Jscripts are loaded. In addition a <div> element needs to be prepared, where the map later will set.

⁴In this case the first hit gets used

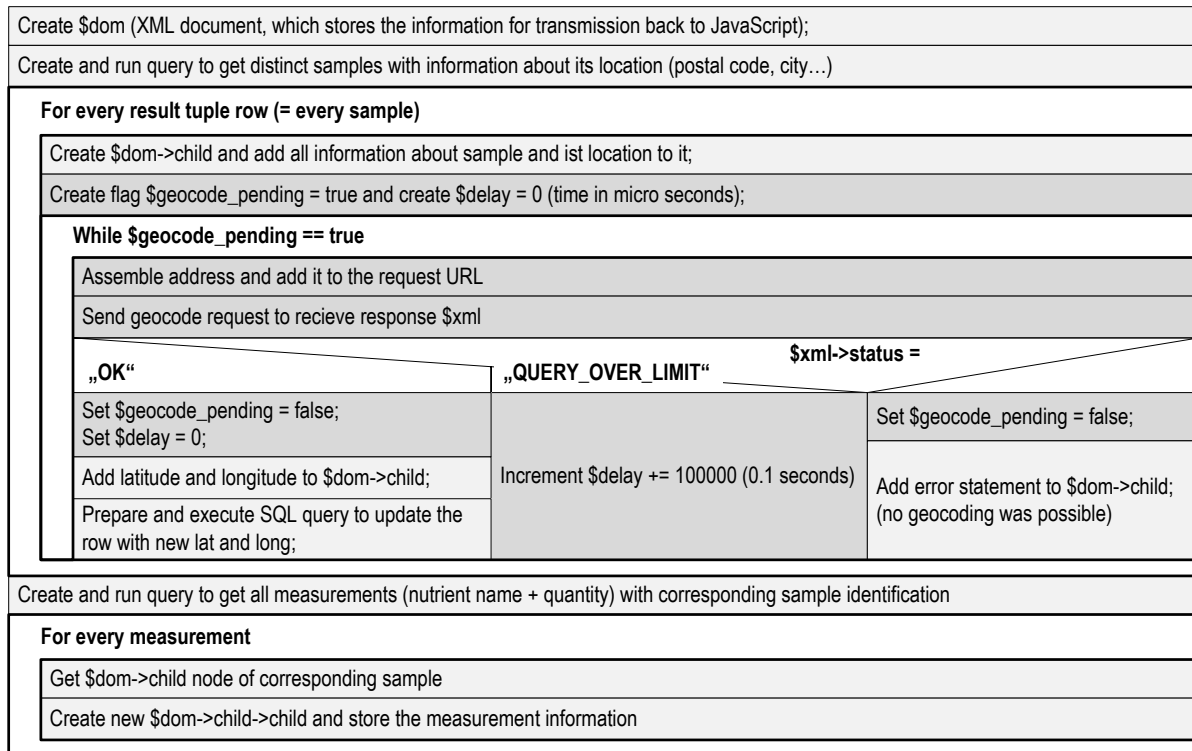


Figure 6.5.: Modeling of the 'Geocode' algorithm.

Example: Required HTML code for mapping

```

1 <html>
2 <header>
3 <script type="text/javascript"
4     src="http://maps.google.ch/maps/api/js?sensor=false"></script>
5 </header>
6 <body>
7     <div id="map_canvas"></div>
8 </body>
9 </html>

```

The only way of visually initiate a map is by JavaScript. In this application the map should only be displayed when showing the results, that's way the required code is embed to the `js3_setMapList()` function. If a map should appear right in the beginning, the corresponding Jscript can be called by adding the event handler *onLoad* to the `<body>` element.

Example: Minimal Jscript to display a map

```

1 Create a map in the prepared <div> element:
2 g_map = new google.maps.Map(document.getElementById("map_canvas"), {
3     Optional: set display options e.g. background map type
4     mapTypeId: google.maps.MapTypeId.TERRAIN});
5 Define the map excerpt object with the coordinates of the SW-, and NE-corner:
6 g_bounds = new google.maps.LatLngBounds(/*param1: LatLng SW,
7     param 2: LatLng NE*/);
8 Set the map to the selected excerpt:
9 g_map.fitBounds(g_bounds);

```

Step 2: Add markers (pins) from the database Constitutive on the background map, it is possible to include all sorts of shapes by using the basic GIS data types (see table 2.1). Every additional shape will then be assigned to the map as a layer, which gives them the name overlay. The simplest overlay is the so called marker (this is the typical Google Maps pin pointer), another is the infowindow (the popup speech bubble).

The goal for the application is to set a marker for every sample with an infowindow containing all information about the address and the measured nutrients. For a better display, only one infowindow should be visible at a time and its visibility should be triggered by hovering over the marker. This constraint can be implemented the following way:

- Only one empty infowindow object will be allocated.
- One marker per sample (data comes from the responseXML) will be allocated.
- The content about address and nutrient will be prepared as HTML output and stored in the corresponding marker: `var html = "LIMS-Nr. " + name + "
..."`
- By the event listener `this.marker.mouseover`, the one infowindow will set visible with the content and the position of the corresponding marker.

Example: Jscript to set markers according to goal

```

1 Create one empty infowindow:
2 g_infowindow = new google.maps.InfoWindow({
3     //content: /*nothing*/,
4     //position: /*nothing*/
5 });
6 for(/*every sample in the responseXML*/) {
7     Create new position with the coordinates from the responseXML:
8     var latlng = new google.maps.LatLng(latitude, longitude);
9     Allocate new marker and set its options (including position and content):
10    var marker = new google.maps.Marker({
11        map: g_map,
12        position: latlng,
13        icon: "images/red-dot.png",
14        shadow: "images/msmarker.shadow.png",
15        content: html,
16        title: "LIMS-Nr. "+name
17    });
18
19    Create event listeners for every marker to open and close the infowindow:
20    google.maps.event.addListener(g_markerArray[i], 'mouseover',
21    function() {
22        g_infowindow.setContent(this.content);
23        g_infowindow.setPosition(this.position);
24        g_infowindow.open(g_map,this);
25    });
26    google.maps.event.addListener(g_markerArray[i], 'mouseout',
27    function() {
28        g_infowindow.close(g_map,this);
29    });
30 }

```

The complete algorithm will be visualised in the next chapter in combination with the set up of the sample enlistment (figure 6.6).

Annotation I: It occurs many times that several samples originate from the same location. To not place the marker all at the exact same spot, a random shifting in the latitude and longitude is used. The following code adds or subtracts max. 100 m in latitude and max. 50 m in longitude to the original coordinates.

```

...add-on to example...
1  var latitude = parseFloat(resultArray[i].getAttribute("lat")) +
2  parseFloat((Math.random()5 - Math.random())/1000);
3  var longitude = parseFloat(resultArray[i].getAttribute("lng")) + -
4  parseFloat((Math.random()- Math.random())/1000);

```

Annotation II: The map excerpt, defined by the object *bounds*, can either be set from the beginning (as presented above), or generated dynamically depending on the overlays in the map so that e.g. all markers will be visible at fist side with maximal possible zoom. This second way will be done by adding every marker's coordinates to the bounds and let the object do the rest.

```

...add-on to example...
1  var g_bounds = new google.maps.LatLngBounds();
2  var latlng = new google.maps.LatLng(latitude, longitude);
3  g_bounds.extend(latlng);

```

Annotation III: One constraint of the application was that new results can be requested in the same session. For this case old markers need to be deleted before adding new ones. This is solved with the help of a global array to which every marker gets assign in his creation process. With this array the current markers can be addressed at any time and if requested erased from the map.

```

...add-on to example...
1  for (i in g_markerArray) {
2      Erase marker from the map:
3      g_markerArray[i].setMap(null);
4  }
5  Delete all old markers:
6  g_markerArray.length = 0;

```

6.3.5. Using Google Visualization

Examples follow the module: M2 (load), M3 (draw table chart)

Topic appears also in module: M4 (draw line chart)

Another service of Google, used in the application, is the collection of Jscripts for supported data visualisation that every web developer can load and use on his page. The embedding of the service corresponds to the one of Google Maps: include the API when loading the application and prepare a <div> element for the later output.

```

Example: Required HTML code for visualisation
1  <html>
2  <header>
3  <script type="text/javascript"
4  src="https://www.google.com/jsapi"></script>

```

```

5 </header>
6 <body>
7     div id="sidebar"></div>
8 </body>
9 </html>

```

This loaded API covers the basic overall functionalities including further loading operations, since every chart type possesses its own script package still laying on the server. This chart dependent, additional loading process will now take place in JavaScript runtime using an asynchronous HTTP request (which is again based on an Ajax object) and needs to be caught by a callback function (see Ajax 5.2.5).

```

Example: Load visualization package for interactive table
1 google.load("visualization", "1", {
2     packages: ["table"],
3     callback: function() {js2_sendMapListAjax(sql_from_where);}
4 });

```

A package will not be loaded twice in one session, even when the same load request gets placed a second time. In this case it jumps directly to the callback function [Google 2011c][*Enhanced Library Loading*]. This is an important fact for the "many result request in one session" constraint.

If the loading was successful and the corresponding callback function has been called, the creation of a chart can be done in two steps.

Step 1: Create and fill a dataTable with all required data The dataTable is the base element of all chart types. All input data required for the chart will be inserted into it. The columns specify the topic (table chart: columns; line chart: series) and the rows get filled by the entries.

```

Example: How to handle a dataTable
1 Create a new dataTable object:
2 g_sampleData = new google.visualization.DataTable();
3 Add columns (type, name) with automatically ascending index values:
4 g_sampleData.addColumn('string', 'LIMS-Nr. ');
5 g_sampleData.addColumn('string', 'Canton');
6 g_sampleData.addColumn('string', 'PLZ');
7 for(/*every sample in responseXML*/){
8     Add a row and fill its first, second and third cell:
9     g_sampleData.addRow(1);
10    g_sampleData.setValue(i, 0, resultArray[i].getAttribute("name"));
11    g_sampleData.setValue(i, 1, resultArray[i].getAttribute("canton"));
12    g_sampleData.setValue(i, 2, resultArray[i].getAttribute("plz"));
13    Assign an additional variable to the entire row, for later identification of row:
14    g_sampleData.setRowProperty(j, "lims",
15        resultArray[i].getAttribute("name"));
16 }

```

In the application, the filling of the dataTable is combined with the creation of Google Maps markers since both are iterating the responseXML with the "nutrient per sample"-results the same way. The complete structural design of this operation is illustrated in fig. 6.6.

Get responseXML and transform it into the resultArray, where every XML sample is one array entry	
If not running the first time (in this session)	
True	False
Delete all markers in g_map and d_markerArray	Create g_map, g_bounds, g_infowindow and g_markerArray
Delete all rows and nutrient columns in g_sampleData	Create g_sampleMap and fill the static columns (Lims-Nr., PLZ, Canton)
For every nutrient in g_nutrientArray (=all selected options in <select name="nutrient[]">)	
Add new column with nutrient name to g_sampleData	
For every sample in resultArray	
Add new row to g_sampleData and fill the cells at index 0, 1 & 2 with name (LIMS-Nr.), plz and canton	
Assign name as row property (row identifier)	
Create var html = "LIMS-Nr. " + masked name + " " + plz + " " + city + " (" + canton + ")"	
Get resultArray[i] and transform it into the nutrientArray, where every child XML nutrient is one array entry	
Create var colCheckNumber = index of first nutrient column (3)	
For every nutrient in nutrientArray	
Append to html as listed expression: „[Nutrientname]: [Quantity]“	
Create var colCheckFlag = true	
While colCheckNumber < number of g_sampleData columns and colCheckFlag is true	
(Var colName = name of column at index colCheckNumber in g_sampleData)	
If colName equals the current nutrient	
True	False
Add quantity to current cell	Set colCheckNumber += 1
Set colCheckFlag = false and colCheckNumber += 1	
If absolute coordinates are available	
True	False
Get coordinates and add a random shift	
Create marker with content: html, position: coordinates, title (id): name	
Set marker on map and add it to the g_markerArray	
Extend the bounds by adding the coordinates	
Create event listener on the current marker for open and close the infowindow	

Figure 6.6.: Modeling of the combined 'transform samples into Google Maps marker' (dark gray) and 'fill the visualisation dataTable' (light gray) algorithm

Annotation: The samples in the responseXML only include valid measurements as assigned children. Empty fields, used in a table, are spared (fig. 6.7). That's why two catches were developed, when transforming the responseXML into an output table:

1. Missing input values: since the nutrient columns in the dataTable cannot be generated with help of the responseXML, all selected nutrients in the select field *nutrient[]* were stored in a global array at the beginning of the 'show result' processing (*js2_result_coordinator.js*). By consulting this array, every requested nutrient gets its column.
2. Empty cells needed: to fill the quantities from the responseXML to the proper dataTable column, sometimes a column needs to be skipped. A corresponding algorithm had to be included in the overall operation (see the While- and For-loop dealing with the parameters *colCheckNumber* and *colCheckFlag* in fig. 6.6). This algorithm presumes that all involved nutrients will always be sorted alphabetically inside of ever sample.

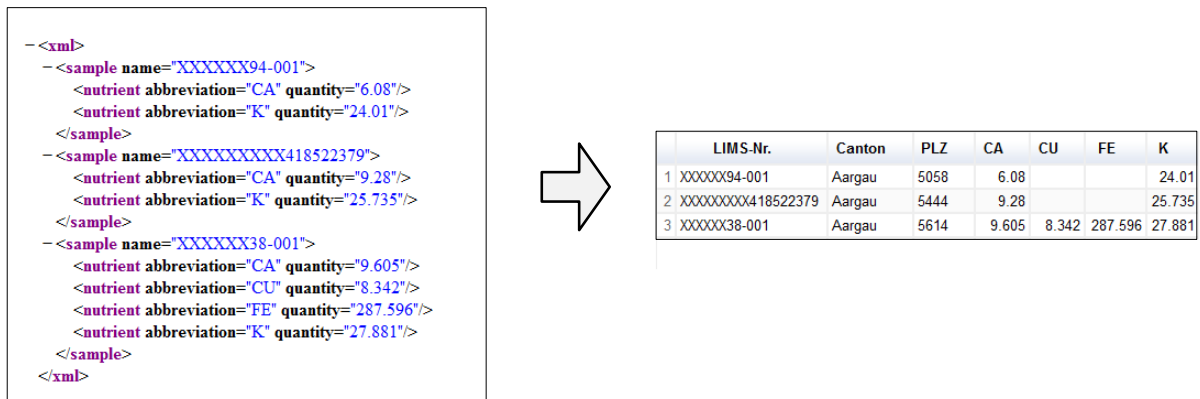


Figure 6.7.: Illustration to the task to transform an XML input, with minimalistic storing principles into a table with static columns

Step 2: Draw the chart When the dataTable has been filled properly, the requested chart will be drawn by the corresponding chart scripts. Own definitions about the appearance can be committed as options into the drawing process.

Example: Finishing the chart

```

1
2 Create new chat object depending on the selected char type and assign it to the prepared <div> element:
3 var sidebarTable = new google.visualization.Table(
4     document.getElementById(' sidebar' ) );
5 sidebarTable.draw(g_sampleData, {
6     Set own parameters (here only one of many possibilities was selected):
7     showRowNumber: true
8 });

```

The result in the examples case is an interactive table, which can be sorted by any column in ascending or descending order and where every entry can be selected with colour highlighting.

6.3.6. Interactive result display with JavaScript event listeners

Examples follow the module: M5+

Topic appears also in module: M3, M4, M5

In the previous two chapters the focus was on creating illustrations by using Google services, whereupon a lot of interactivity for the user was offered inside of the delivered result units:

Google Map: *Hovering over the marker will open its associated infowindow.
*Scale- and scrollable map excerpt.

Visualization Table: *Selecting a row will be indicated by changing its colour.
*The rows can be sorted according to a selected column.

As presented several times so far, the supply of these interactivities is always based on JavaScript. It was also discussed how such interactivity can be applied to the HTML web page code by using DOM scripting. Now, should be shown that it is also possible to include the result units of the external services with all their sub elements (marker, table rows) into superior web page scripts.

In the following example, the establishing of the most complex interaction in this application by combining a HTML element with the map via the visualisation table will be discussed:

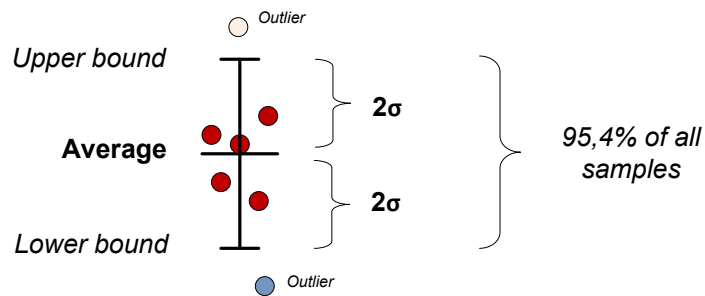


Figure 6.8.: Illustration of the statistical definition of an outlier

The statistical outliers of all samples in relation to a nutrient should be reflected as highlighted Google Maps marker as soon as the corresponding nutrient in the table "Statistical information of nutrients" gets clicked (fig. 6.9 & 6.8).

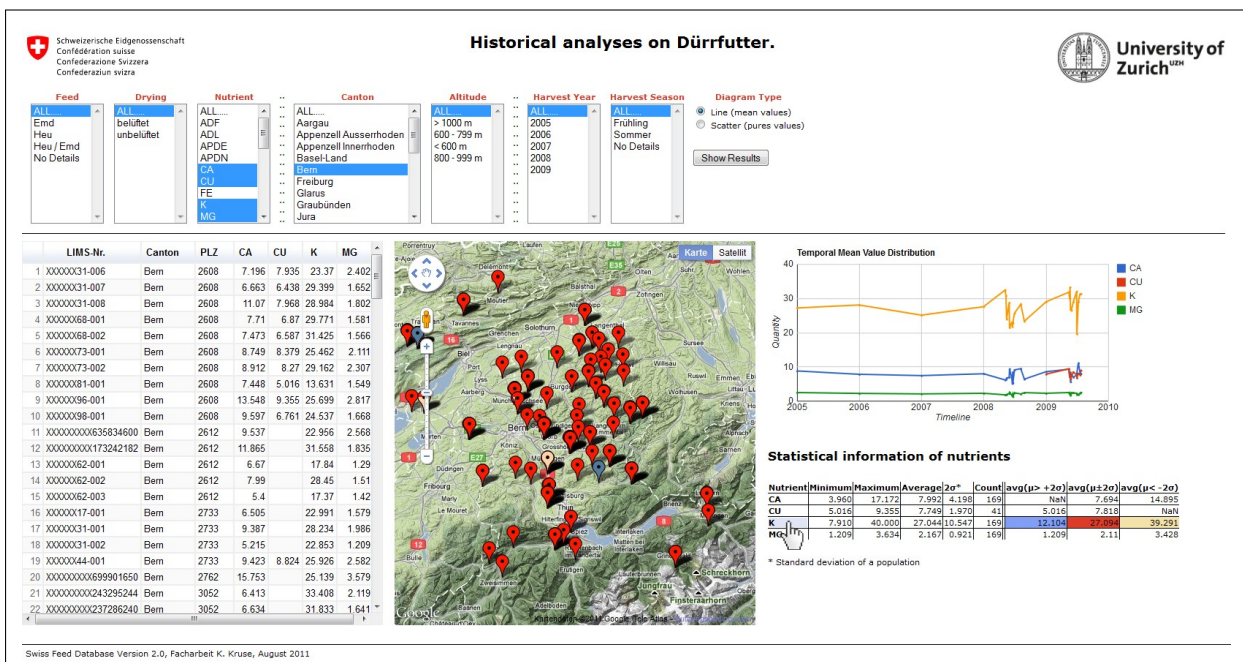


Figure 6.9.: Screenshot of the Swiss Feed Database 2.0 surface with activated interactivity 'show outlier for nutrient'

From a statistical point of view, the computation includes:

- The average value of the selected nutrient
- The standard deviation of a population in 2sigma range of the selected nutrient
- The distinct samples with the quantity of the selected nutrient.

The first two inputs are available in the aggregation table ("Statistical information of nutrients", build as HTML table) were also the click event is supposed to start. But the values of the third requirement are only addressable in the sample table (build as Google Visualisation table).

From a technical point of view, the goal is to link the row selection of the HTML table with corresponding nutrient column of the Google Visualization table. When arriving, all cells of the requested column will be traversed and the outlier samples detected. These samples are then to be forwarded to the Google map where the corresponding markers need to be found. Then finally, the markers will be set to a different colour.

Step 1: Prepare HTML table The base to this interactivity lies in the HTML code. To be able to trigger Javascript inside a table, every nutrient cell includes a <div> element, where the event handler and the Jscript function is implemented.

Example: Required HTML base for the interactivity

```

1 <table>
2 <tr><th>Nutrients:</th><th>Average:</th><th>StdDev:</th></tr>
3 <tr><td><div onClick="function([params1])">Nutrient 1</div></td>
4   <td>Average 1</td><td>StdDev 1</td></tr>
5 <tr><td><div onClick="function([params2])">Nutrient 2</div></td>
6   <td>Average 2</td><td>StdDev 2</td></tr>
7 </table>

```

The parameters inside the Jscript function will be set during the generation of the table itself, and therefore contains information depending on the nutrient.

1. Parameter: column number
2. Parameter: upper bound of this nutrients 2sigma range
3. Parameter: lower bound of this nutrients 2sigma range

The first parameter is storing the required link to the sample list. It is supposed to be the index number of the corresponding nutrient in the Google Visualization table.

Annotation: To assign the right column number, an algorithm has to be written, which allows the HTML table to impersonate the indices of the Google Visualization columns. Therefore the shifting between these two tables has to be respected. Additionally, the global `d_nutrientArray` (= the array which stores all selected nutrients from the <select> field `nutrient[]` as presented in the annotation of chapter 6.3.5) is iterated to coordinate the right number exchange, even if a selected nutrient does appear in the sample table, but not in the HTML table (happens if a nutrient contains no measurements).

Step 2: Prepare JavaScript elements Another trace of breadcrumbs has been laid on the JavaScript level to be able to link the samples from the visualisation table with the markers in the map: Every marker carries the sample identifier (LIMS-Number) as unique key, stored in the parameter "title".

```

1  var marker = new google.maps.Marker({
2      map: g_map,
3      position: latlng,
4      icon: "images/red-dot.png",
5      shadow: "images/msmarker.shadow.png",
6      content: html,
7      title: "LIMS-Nr. "+name
8  });

```

In addition, the same identifier was also assigned to every row in the sample table when the table was filled:

```

1  g_sampleData.setRowProperty(j, "lims",
2      resultArray[i].getAttribute("LIMS-Nr. "+name));

```

Step 3: Connect and query By having all necessary connections, the actual function is the quite simple. The samples in the Visualization table can be addressed and iterated directly by their row and column index number. When finding an outlier sample, the global marker array, which stores all current maps marker, get traversed until a one element gets found with the same sample key as the table row. The last step is then to set the marker to a new colour. Since there are no colour options for markers in Google Maps, this needs to be done by loading another coloured marker symbol.

```

1  function js5_highlightStatGroups(col, highSigmaValue, lowSigmaValue ){
2      var rows = g_sampleData.getNumberOfRows();
3      var g_markerArraylength = g_markerArray.length;
4
5      Traverse the sample table:
6      for(var j = 0; j < rows; j++){
7          Get the row key:
8          var wanted = g_sampleData.getRowProperty(j, "lims");
9          Get the cell content (quantity):
10         var cell = g_sampleData.getValue(j, col);
11
12         Iterate the global marker array to find the corresponding marker:
13         var marker = null;
14         for(var k = 0; k < g_markerArraylength; k++){
15             if(g_markerArray[k] .title == wanted){
16                 marker = g_markerArray[k];
17             }
18         }
19         Sort the quantity according to outlier detection and assign new marker symbol:
20         if(cell != null){
21             if(cell < lowSigmaValue){
22                 marker.setIcon("images/dark-dot.png");
23                 marker.setShadow("images/shadow.png");
24             }else if(cell > highSigmaValue){
25                 marker.setIcon("images/light-dot.png");
26                 marker.setShadow("images/shadow.png");

```

```
27         }else if((cell >= lowSigmaValue)
28             && (cell <= highSigmaValue)){
29             marker.setIcon("images/red-dot.png");
30             marker.setShadow("images/shadow.png");
31         }else{
32             marker.setIcon("images/grey.png");
33             marker.setShadow("images/shadow.png");
34         }
35     }else{
36         marker.setIcon("images/grey.png");
37         marker.setShadow("images/shadow.png");
38     }
39 }
40 }
```

7. Testing

7.1. Surveillance of implementation

The implementation of the application has been monitored constantly during the creation process. Because of the rather developed interaction of the system components on the separated architectural locations (according to chapter 5.2), the testing was done from small to big scale. All testing levels had in common that they used an identical test database on the pg.ifi.uzh.ch server. This database was created according to the design in section 5.4 and filled with 54'489 measurements with corresponding dimension table parameters from a first data delivery 'Rohdaten_Dürrfutter_2005-2009_mit PLZ' handed over as excel spreadsheet.

7.1.1. Single functionality

When it comes to the development of the server (PHP) and client (JavaScript) scripts, the traceable execution required always a browser as environment and a HTML file as a medium. Any new functionality was first implemented and executed in a stand-alone environment according to its purpose. In this sense the pure PHP features, like 'connect and query the database' or 'transform some query result into a XML', could be tested in separate before including them into the online application where their execution is triggered indirectly by JavaScript. This has the advantage that the results or errors, which would later be caught by JavaScript, can be inspected at once by printing them on some intermediate HTML page. Also the JavaScript feature are tested in separate by using an own HTML output file. But next to the possibility to display result values on the screen, this browser script brings also along the *alert* functionality, where popups with variable content can be set and called at anytime during the algorithm's runtime. These *alert* messages can be perfectly used to follow the processes and check its internal values.

In the case of creating the database or simply verifying database queries or table contents, the PostgreSQL client interface pgAdmin III, Version 1.12.03 was used to address the database server and execute SQL queries. Larger, repetitive SQL transactions, like the loading of the data were simplified by embed the involved queries into a Java program, from where they were send to the server by using the Java-Database interface JDBC.

7.1.2. Module

On the module level, the focus was on the proper interconnection of the components, especially by the use of the Asynchronous JavaScript and XML (AJAX), and the right visual appearance in the online application. Therefore, the output medium was the final HTML web page itself, but for testing reasons in different browsers. In this state of testing, the web developer has no longer direct access to the server results, since they will not be visually displayed until they got processed by JavaScript. So, the JavaScript *alert* functionality has an even bigger scope than before, by catching and displaying the server's responses. In addition, it has to be taken care

of that the server script responses are also covering reasonable error messages in failure cases, which can be alerted by JavaScript as soon as they are loaded on the client.

7.1.3. System

In the end, the complete application system was brought to trial any time a new feature or module was included, to check whether all components were functioning in parallel or in inter-connection as thought. This test was not only applied on different browsers, but also a second test version, online available on a second HTTP server could be used in this case.

7.1.4. Testing environment

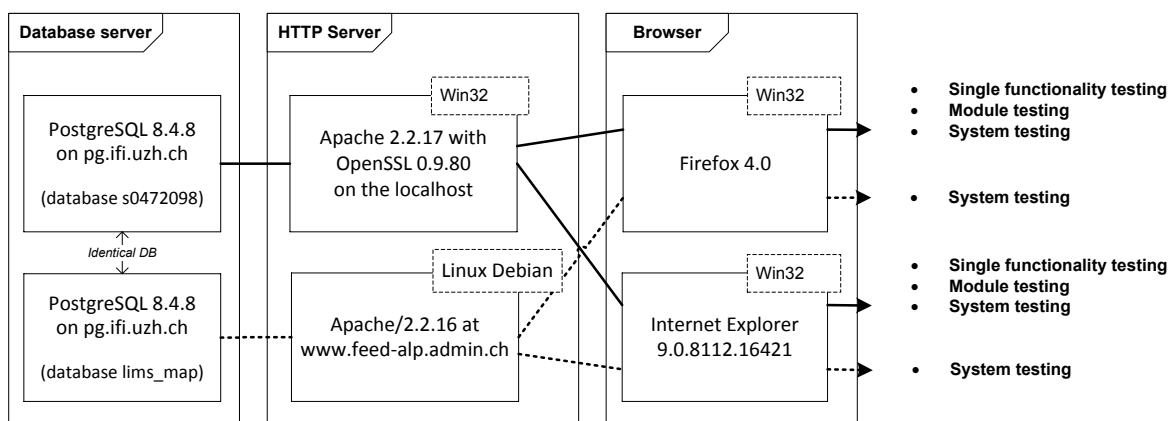


Figure 7.1.: The technical environment during the implementation and testing phase of the Swiss Feed Database 2.0.

7.2. Application practice

Next to the tests to guarantee a correct implementation, a field test, where the final end user is checking the application for practical approach is needed. This test has not yet taken place. Only short demonstrations and brief feedbacks have been exchanged between developer and user. So, this test will have to be caught up later, after this theses are concluded, maybe with the effect that some of the implemented functionalities will be rewritten.

8. Maintenance and outlook

This last technical chapter is about the future handling of the now developed Swiss Feed Database Version 2.0. It includes, next to the pure support of the new existing implementation, also annotation to future developments in or for the application.

8.1. Updates

The maintenance of the current implementation centres mainly on the adjustment of its components to future developments and updates. This starts with the 'internal' components (HTML, PHP, JavaScript), where especially the JavaScripts scripts and the included Ajax have to be held up-to-date, since they are running on some clients, where they are dependent on the surrounding browser technology. But also the more 'static' components like the PHP server script or the basic HTML web page should be adapted to the newest possibilities for a modern exposition. Furthermore, a special focus has to be brought on the 'external' components (Google Maps 'mapping' & 'geocoding', Google Visualization). Google will constantly enhance its products together with the public interfaces, like the APIs that are used in this application. Therefore it is important to always keep on track with this updates and to adapt them to the present implementation.

8.2. Future data input

An additional point will be to elaborate a proper data exchange for future data imports. It starts with a verification of the database design and requests the development of a more restricted sample description schema including a predefined parallel namespace in at least German and French. After this, the data loading process could be implemented into an additional web access, where the involved laboratories can load their data directly.

8.3. Future features and adaptations

Finally, the present application can and should be expanded to new functionalities or different initial states. Some suggestions basing on the current implementation will be listed here:

- The compilation of the parameter's <select> fields in the input form of the web page can be set up at own preferences. The implementation of this part was specifically engineered for being easily adoptable. Any parameter, which is stored in the database as own attribute can be used as source for a new field. Therefore, only the PHP instructions in the HTML start page needs to be changed to the new amount and the new properties of the <select> fields, all constitutive operations like 'creating proper SQL queries' will be handled by the system. Further information including the instructions are collected in section 6.3.1.

- The actual application is held in a mixture of English web surface and German database content. One further development would be to establish a multi-language functionality, not only in the data content (as requested in the previous section 8.2) but also in the web page and in its access to the corresponding database attributes.
- The map has not yet fully tabbed its potential. Especially one functionality, supported by Google Maps could be of interest for an agricultural online application: it is possible to embed one or several custom maps as background to the Google map. In this sense, an own map with agricultural information, originating from the GEOSTAT project of the federal office of statistics [Geostat 1994], could be used instead of a normal earth surface background.

9. Summary

9.1. Database system with geographical information

In this presented theses, the construction of the application system (database plus online application as user interface) was the main topic. By doing so, the actual specialty of implementing a database based on geographical information was implied together with the surrounding tasks. Now, in a brief summary, the different stations in including geographical information will be revisited more explicit.

The first step was to design a database that includes into the stored information also a geographical dimension. Derived from the expected data input, the new database of the Swiss Feed Database project was equipped with the dimension table `d_origin`, which memorises all distinct locations of the sample's origin by saving its address and altitude level information (chapter 5.4).

In order to display this locations later on a map, the so far descriptive information were expanded by the attributes latitude and longitude, which are specifying the exact position of every location. This, mostly unknown, coordinates will be collected later on by the embedded Google Maps 'geocode' service, where descriptive address information can be transformed into corresponding coordinates by web request (chapter 6.3.3).

In a next step, the user input of the online application was developed, with the purpose that the database and its measurements can be filtered, according to an interactive selection, before they will be applied on predefined output algorithms (chapter 5.1). Therefore several HTML `<select multiple>` elements were filled with options originating from selected attributes of the database. In addition, all select fields are linked, so that the options of one field are always dependent on the selection of the previous one (fig. 9.1). The geographical information is represented by two select fields with the content of the attributes `d_origin.canton_de` and `d_origin.city_altitude`, with a linked property so that if a specific canton is chosen, only valid altitude levels will be displayed in the next select field.

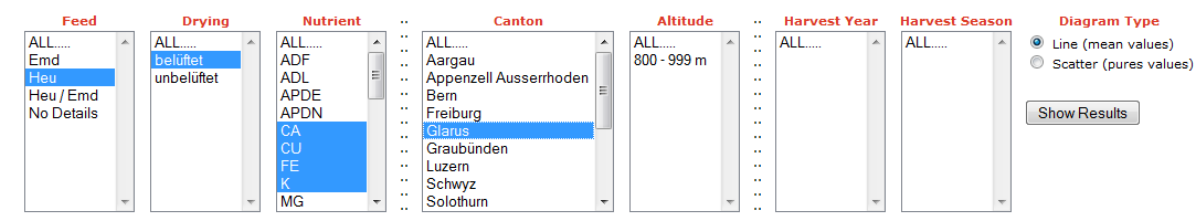


Figure 9.1.: Final presentation of the selection part in the Swiss Feed Database 2.0.

One major point was the implementation of dynamical SQL queries, since the FROM and WHERE parts needs to be generated according to the chosen options. This requires a distribution of information across several select fields (chapter 6.3.1). The applied solution is using the server script on one hand to distribute the information before loading the page and a browser script algorithm on the other hand to generate the SQL queries in runtime. Thereby, the server script is constructed in a way, so that the combination of parameters for the input selection can be changed according to own preferences.

Another point was the construction of an interactive web page architecture with only one HTML file in the centre. In order to gather information from the database even after the page is loaded on the client, the Asynchronous JavaScript and XML object (Ajax) had to be used to establish the required connections (chapter 5.2.5).

The main step was then to prepare the output of the online application. Therefore, a set of six basic operations were carved out of the ongoing scientific research and used to elaborate parallel algorithms for the final output (chapter 4).

As presented in table 9.1, the output of operation 5 specifically covers the geographical information by illustrating the spatial distribution of involved samples on a customised map. In order to gain this result, Google Maps with its map providing service had to be embedded into the application. The empty map structure is then to be filled with the required sample information, among them latitude and longitude, from the database (chapter 6.3.4).

In a final step, additional element-connecting features on the loaded page were developed to allow the user an interactive reading of results (chapter 6.3.6). It is in this interactivity, where the geographical information demonstrate its central role in the Swiss Feed Database. Every selection of the surrounding output units is visually linked to the map, whenever a connection to a single sample can be established:

- Operation 6: When a sample in the list gets selected, the corresponding location will be indicated on the map by showing the associated infowindow.
- Operation 4: When a measurement in the scatter chart gets selected, the corresponding sample will be indicated on the map, again, by showing the associated infowindow.
- Operation 2: The distribution of the samples into the sigma-dependent subsets will be visualised by colouring the locations on the map in a corresponding way as soon as a nutrient in the aggregation table is selected.

Concluding this summary, it can be said that the geographical information are playing a crucial part in many aspects of the presented implementation. Although the embedding of a Google map for an appropriate illustration of locations was the desired intention in the development process, several preliminary arrangements in the database and the application had to be covered first. In doing so, many interesting opportunities opened, regarding for instance the 'geocode' service or the highly linkable interactivity in JavaScript. There is still some margin left to expand the Swiss Feed Database Version 2.0. but the basic installation and functionality is hereby set.

OPERATION	OUTPUT UNIT																																																																																																																																																																								
<p>1: Computation of the aggregate values count, minimum, maximum, average, standard deviation (of a population) of the involved measurements.</p> <p>Implemented as a HTML <table>, which gets created and filled by a PHP script. The data origins directly from the database.</p>	<p>Statistical information of nutrients</p> <table border="1"> <thead> <tr> <th>Nutrient</th> <th>Minimum</th> <th>Maximum</th> <th>Average</th> <th>2σ*</th> <th>Count</th> </tr> </thead> <tbody> <tr> <td>ADL</td> <td>19.000</td> <td>52.000</td> <td>31.820</td> <td>14.811</td> <td>50</td> </tr> <tr> <td>APDN</td> <td>35.971</td> <td>112.724</td> <td>76.221</td> <td>30.660</td> <td>61</td> </tr> <tr> <td>CU</td> <td>7.816</td> <td>8.342</td> <td>8.040</td> <td>0.544</td> <td>3</td> </tr> <tr> <td>K</td> <td>27.881</td> <td>32.227</td> <td>29.996</td> <td>4.350</td> <td>3</td> </tr> </tbody> </table> <p>* Standard deviation of a population</p>	Nutrient	Minimum	Maximum	Average	2σ*	Count	ADL	19.000	52.000	31.820	14.811	50	APDN	35.971	112.724	76.221	30.660	61	CU	7.816	8.342	8.040	0.544	3	K	27.881	32.227	29.996	4.350	3																																																																																																																																										
Nutrient	Minimum	Maximum	Average	2σ*	Count																																																																																																																																																																				
ADL	19.000	52.000	31.820	14.811	50																																																																																																																																																																				
APDN	35.971	112.724	76.221	30.660	61																																																																																																																																																																				
CU	7.816	8.342	8.040	0.544	3																																																																																																																																																																				
K	27.881	32.227	29.996	4.350	3																																																																																																																																																																				
<p>2: Computation of the aggregate value average in sigma-dependent subsets.</p> <p>Implemented as a HTML <table>, which get filled by JavaScript, when clicking on the corresponding nutrient ¹. As data, the sample list in the web page is used.</p>	<p>Statistical information of nutrients</p> <table border="1"> <thead> <tr> <th>Nutrient</th> <th>avg(μ> +2σ)</th> <th>avg(μ±2σ)</th> <th>avg(μ< -2σ)</th> </tr> </thead> <tbody> <tr> <td>ADL</td> <td>NaN</td> <td>30.745</td> <td>48.667</td> </tr> <tr> <td>APDN</td> <td>39.073</td> <td>76.873</td> <td>112.724</td> </tr> <tr> <td>CU</td> <td>NaN</td> <td>8.04</td> <td>NaN</td> </tr> <tr> <td>K</td> <td>NaN</td> <td>29.996</td> <td>NaN</td> </tr> </tbody> </table>	Nutrient	avg(μ> +2σ)	avg(μ±2σ)	avg(μ< -2σ)	ADL	NaN	30.745	48.667	APDN	39.073	76.873	112.724	CU	NaN	8.04	NaN	K	NaN	29.996	NaN																																																																																																																																																				
Nutrient	avg(μ> +2σ)	avg(μ±2σ)	avg(μ< -2σ)																																																																																																																																																																						
ADL	NaN	30.745	48.667																																																																																																																																																																						
APDN	39.073	76.873	112.724																																																																																																																																																																						
CU	NaN	8.04	NaN																																																																																																																																																																						
K	NaN	29.996	NaN																																																																																																																																																																						
<p>3: Computation of the aggregate value average for every distinct temporal value (= development of mean value over time).</p> <p>Implemented as a Google Visualisation line chart, created by JavaScript after PHP returns the corresponding mean values.</p>	<p>Diagram Type</p> <ul style="list-style-type: none"> <input checked="" type="radio"/> Line (mean values) <input type="radio"/> Scatter (pures values) 																																																																																																																																																																								
<p>4: Enlistment of all involved measurements with temporal information (= distribution of values over time).</p> <p>Implemented as a Google Visualisation scatter chart (equals line chart without connection lines), created by JavaScript after PHP returns the corresponding measurement values.</p>	<p>Diagram Type</p> <ul style="list-style-type: none"> <input type="radio"/> Line (mean values) <input checked="" type="radio"/> Scatter (pures values) 																																																																																																																																																																								
<p>5: Enlistment of all involved measurements with information about their origin (= distribution of values in space).</p> <p>Implemented as a Google Maps map, created by JavaScript (in combination with the sample list) after PHP returns the corresponding sample and measurement values.</p>	<table border="1"> <thead> <tr> <th>LIMS-Nr.</th> <th>Canton</th> <th>PLZ</th> <th>ADL</th> <th>APDN</th> <th>CU</th> <th>K</th> </tr> </thead> <tbody> <tr><td>1</td><td>XXXXXX084-015</td><td>Aargau 5722</td><td>29</td><td>76.922</td><td>7.816</td><td>32.227</td></tr> <tr><td>2</td><td>XXXXXX084-014</td><td>Aargau 5722</td><td>31</td><td>82.757</td><td>7.961</td><td>29.888</td></tr> <tr><td>3</td><td>XXXXXX088-001</td><td>Aargau 5614</td><td>29</td><td>83.43</td><td>8.342</td><td>27.881</td></tr> <tr><td>4</td><td>XXXXXX088-001</td><td>Aargau 4805</td><td>27</td><td>72.955</td><td></td><td></td></tr> <tr><td>5</td><td>XXXXXX045-001</td><td>Aargau 5024</td><td>40</td><td>72.911</td><td></td><td></td></tr> <tr><td>6</td><td>XXXXXX088-001</td><td>Aargau 5054</td><td>31</td><td>95.332</td><td></td><td></td></tr> <tr><td>7</td><td>XXXXXX045-001</td><td>Aargau 5306</td><td>31</td><td>51.095</td><td></td><td></td></tr> <tr><td>8</td><td>XXXXXX11-001</td><td>Aargau 5444</td><td>34</td><td>80.076</td><td></td><td></td></tr> <tr><td>9</td><td>XXXXXX03-001</td><td>Aargau 5444</td><td>33</td><td>80.774</td><td></td><td></td></tr> <tr><td>10</td><td>XXXXXX045-001</td><td>Aargau 5610</td><td>33</td><td>91.404</td><td></td><td></td></tr> <tr><td>11</td><td>XXXXXX03-001</td><td>Aargau 5623</td><td></td><td>85.29</td><td></td><td></td></tr> <tr><td>12</td><td>XXXXXX12-001</td><td>Aargau 5623</td><td>24</td><td>88.718</td><td></td><td></td></tr> <tr><td>13</td><td>XXXXXX036-001</td><td>Aargau 5623</td><td>26</td><td>86.101</td><td></td><td></td></tr> <tr><td>14</td><td>XXXXXX029-001</td><td>Aargau 5624</td><td></td><td>79.54</td><td></td><td></td></tr> <tr><td>15</td><td>XXXXXX01-001</td><td>Aargau 5624</td><td></td><td>79.95</td><td></td><td></td></tr> <tr><td>16</td><td>XXXXXX058-001</td><td>Aargau 5624</td><td>21</td><td>88.631</td><td></td><td></td></tr> <tr><td>17</td><td>XXXXXX032-001</td><td>Aargau 5624</td><td>23</td><td>93.515</td><td></td><td></td></tr> <tr><td>18</td><td>XXXXXX039-001</td><td>Aargau 5624</td><td>23</td><td>74.862</td><td></td><td></td></tr> <tr><td>19</td><td>XXXXXX047-001</td><td>Aargau 5630</td><td></td><td>91.19</td><td></td><td></td></tr> <tr><td>20</td><td>XXXXXX15-001</td><td>Aargau 5634</td><td>23</td><td>106.562</td><td></td><td></td></tr> <tr><td>21</td><td>XXXXXX17-001</td><td>Aargau 5647</td><td>25</td><td>95.985</td><td></td><td></td></tr> <tr><td>22</td><td>XXXXXX173-001</td><td>Aargau 5647</td><td>30</td><td>67.15</td><td></td><td></td></tr> <tr><td>23</td><td>XXXXXX193-001</td><td>Aargau 5647</td><td>30</td><td>66.74</td><td></td><td></td></tr> </tbody> </table>	LIMS-Nr.	Canton	PLZ	ADL	APDN	CU	K	1	XXXXXX084-015	Aargau 5722	29	76.922	7.816	32.227	2	XXXXXX084-014	Aargau 5722	31	82.757	7.961	29.888	3	XXXXXX088-001	Aargau 5614	29	83.43	8.342	27.881	4	XXXXXX088-001	Aargau 4805	27	72.955			5	XXXXXX045-001	Aargau 5024	40	72.911			6	XXXXXX088-001	Aargau 5054	31	95.332			7	XXXXXX045-001	Aargau 5306	31	51.095			8	XXXXXX11-001	Aargau 5444	34	80.076			9	XXXXXX03-001	Aargau 5444	33	80.774			10	XXXXXX045-001	Aargau 5610	33	91.404			11	XXXXXX03-001	Aargau 5623		85.29			12	XXXXXX12-001	Aargau 5623	24	88.718			13	XXXXXX036-001	Aargau 5623	26	86.101			14	XXXXXX029-001	Aargau 5624		79.54			15	XXXXXX01-001	Aargau 5624		79.95			16	XXXXXX058-001	Aargau 5624	21	88.631			17	XXXXXX032-001	Aargau 5624	23	93.515			18	XXXXXX039-001	Aargau 5624	23	74.862			19	XXXXXX047-001	Aargau 5630		91.19			20	XXXXXX15-001	Aargau 5634	23	106.562			21	XXXXXX17-001	Aargau 5647	25	95.985			22	XXXXXX173-001	Aargau 5647	30	67.15			23	XXXXXX193-001	Aargau 5647	30	66.74		
LIMS-Nr.	Canton	PLZ	ADL	APDN	CU	K																																																																																																																																																																			
1	XXXXXX084-015	Aargau 5722	29	76.922	7.816	32.227																																																																																																																																																																			
2	XXXXXX084-014	Aargau 5722	31	82.757	7.961	29.888																																																																																																																																																																			
3	XXXXXX088-001	Aargau 5614	29	83.43	8.342	27.881																																																																																																																																																																			
4	XXXXXX088-001	Aargau 4805	27	72.955																																																																																																																																																																					
5	XXXXXX045-001	Aargau 5024	40	72.911																																																																																																																																																																					
6	XXXXXX088-001	Aargau 5054	31	95.332																																																																																																																																																																					
7	XXXXXX045-001	Aargau 5306	31	51.095																																																																																																																																																																					
8	XXXXXX11-001	Aargau 5444	34	80.076																																																																																																																																																																					
9	XXXXXX03-001	Aargau 5444	33	80.774																																																																																																																																																																					
10	XXXXXX045-001	Aargau 5610	33	91.404																																																																																																																																																																					
11	XXXXXX03-001	Aargau 5623		85.29																																																																																																																																																																					
12	XXXXXX12-001	Aargau 5623	24	88.718																																																																																																																																																																					
13	XXXXXX036-001	Aargau 5623	26	86.101																																																																																																																																																																					
14	XXXXXX029-001	Aargau 5624		79.54																																																																																																																																																																					
15	XXXXXX01-001	Aargau 5624		79.95																																																																																																																																																																					
16	XXXXXX058-001	Aargau 5624	21	88.631																																																																																																																																																																					
17	XXXXXX032-001	Aargau 5624	23	93.515																																																																																																																																																																					
18	XXXXXX039-001	Aargau 5624	23	74.862																																																																																																																																																																					
19	XXXXXX047-001	Aargau 5630		91.19																																																																																																																																																																					
20	XXXXXX15-001	Aargau 5634	23	106.562																																																																																																																																																																					
21	XXXXXX17-001	Aargau 5647	25	95.985																																																																																																																																																																					
22	XXXXXX173-001	Aargau 5647	30	67.15																																																																																																																																																																					
23	XXXXXX193-001	Aargau 5647	30	66.74																																																																																																																																																																					
<p>6: Enlistment of all involved measurements with their parameter quantities (= access to original data).</p> <p>Implemented as a Google Visualization table, created by JavaScript (in combination with the map) after PHP returns the corresponding sample and measurement values.</p>																																																																																																																																																																									

Table 9.1.: Composition of the application output.

A. Code appendix

A.1. Start pages

Listing A.1: feeddb-start.php

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml">
  <head>
5 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <meta http-equiv="cache-control" content="no-cache">
7 <meta http-equiv="pragma" content="no-cache">
  <link rel="stylesheet" href="styles-feeddb.css" type="text/css">
9
  <script type="text/javascript" src="http://maps.google.ch/maps/api/js?
    sensor=true"></script>
11 <script type="text/javascript" src="https://www.google.com/jsapi"></script
    >
13 <script type="text/javascript" src="jsA_global-variables.js"></script>
  <script type="text/javascript" src="jsB_ajax-object.js"></script>
15 <script type="text/javascript" src="jsC_loader.js"></script>
  <script type="text/javascript" src="jsD_sql-from-where.js"></script>
17 <script type="text/javascript" src="js1_update_selectfield.js"></script>
  <script type="text/javascript" src="js2_result_coordinator.js"></script>
19 <script type="text/javascript" src="js3_result_list-map.js"></script>
  <script type="text/javascript" src="js4_result_diagram.js"></script>
21 <script type="text/javascript" src="js5_result_aggregate.js"></script>
  <title>Swiss Feed Database 2.0.</title>
23 </head>
25 <body>
27 <!-- Block with settings of included selection parameters -->
  <?php
29   require("class-selectfield.php");
   require("pg-dbinform.php");
31   $conn=pg_connect('host='.$_host.' port='.$_port.' dbname='.$_database.' user
     ='.$_username.' password='.$_password.' sslmode=require');
   if (!$conn) {
33     die();
   }
35
   //--1-- instantiate select Fields:
37   $feed = new selectField("feed[]");
   $feed->setOriginTable("d_feed");
39   $feed->setOriginColumn("d_feed.name_de");
   $feed->setJoinToFactTable("id_feed_fkey = feed_key");
41   $feed->doInitialQuery($conn);
```

```
43 $drying = new selectField("drying[]");
44 $drying->setOriginTable("d_quality_parameters");
45 $drying->setOriginColumn("drying_condition_de");
46 $drying->setJoinToFactTable("id_quality_fkey = quality_key");
47
48 $nutrient = new selectField("nutrient[]");
49 $nutrient->setOriginTable("d_nutrient");
50 $nutrient->setOriginColumn("abbreviation_de");
51 $nutrient->setJoinToFactTable("id_nutrient_fkey = nutrient_key");
52
53 $canton = new selectField("canton[]");
54 $canton->setOriginTable("d_origin");
55 $canton->setOriginColumn("canton_de");
56 $canton->setJoinToFactTable("id_origin_fkey = origin_key");
57
58 $altitude = new selectField("altitude[]");
59 $altitude->setOriginTable("d_origin");
60 $altitude->setOriginColumn("city_altitude");
61 $altitude->setJoinToFactTable("id_origin_fkey = origin_key");
62
63 $time_year = new selectField("timeYear[]");
64 $time_year->setOriginTable("d_harvesttime");
65 $time_year->setOriginColumn("t_year");
66 $time_year->setJoinToFactTable("id_time_fkey = time_key");
67
68 $time_season = new selectField("timeSeason[]");
69 $time_season->setOriginTable("d_harvesttime");
70 $time_season->setOriginColumn("season_de");
71 $time_season->setJoinToFactTable("id_time_fkey = time_key");
72
73 //--2-- set sql and dependency parameter:
74 $feed->setNextSF($drying);
75 $drying->setNextSF($nutrient);
76 $nutrient->setNextSF($canton);
77 $canton->setNextSF($altitude);
78 $altitude->setNextSF($time_year);
79 $time_year->setNextSF($time_season);
80
81 pg_close($conn);
82 ?>
83
84 <!-- HTML structure of the application page -->
85 <div class="page">
86   <div class="title">
87     <h1>
88       <IMG SRC="images/logo_suisse.jpg" align="left"/>
89       Historical analyses on D&uuml;rffutter.
90       <IMG src="images/logo_uzh.jpg" align="right" / width="230" height="70">
91     </h1>
92   </div>
93
94   <!-- ***** SELECTION ***** -->
95   <div class="selection">
96     <form name="testForm">
97       <div>
98         <h3>Feed</h3><br>
```

```

99     <?php $feed->printField(); ?>
    </div>
101    <div>
    <h3>Drying</h3><br>
103    <?php $drying->printField(); ?>
    </div>
105    <div>
    <h3>Nutrient</h3><br>
107    <?php $nutrient->printField(); ?>
    </div>
109    <div class="SFseparator">
    ..<br>..<br>..<br>..<br>..<br>..<br>..<br>..<br>..<br>..<br>..<br>..
111    </div>
    <div>
113    <h3>Canton</h3><br>
    <?php $canton->printField(); ?>
115    </div>
    <div>
117    <h3>Altitude</h3><br>
    <?php $altitude->printField(); ?>
119    </div>
    <div class="SFseparator">
121    ..<br>..<br>..<br>..<br>..<br>..<br>..<br>..<br>..<br>..<br>..<br>..
    </div>
123    <div>
    <h3>Harvest Year</h3><br>
125    <?php $time_year->printField(); ?>
    </div>
127    <div>
    <h3>Harvest Season</h3><br>
129    <?php $time_season->printLastField(); ?>
    </div>
131    <div>
    <h3>Diagram Type</h3><br>
133    <input type="radio" name="charttype" value="line" checked> Line (mean
        values)<br>
        <input type="radio" name="charttype" value="scatter"> Scatter (pures
        values)<br><br><br>
135    <?php $time_season->printQueryButton(); ?>
    </div>
137    </form>
</div>
139
141 <!-- ***** RESULT ***** -->
<div id="output">
    <div class="result" id="result_list">
143     <div class="background" id="sidebar"></div>
    <div class="overlay" id="loader_sidebar">
145     
    </div>
147 </div>
    <div class="result" id="result_map">
149     <div class="background" id="map_canvas"></div>
    <div class="overlay" id="loader_map">
151     
    </div>
153 </div>

```

```

155 <div class="result" id="result_diagram">
    <div class="background" id="chart_div"></div>
    <div class="overlay" id="loader_chart">
157     
    </div>
159 </div>
    <div class="result" id="result_aggregate">
161     <div class="background" id="aggtable_space"></div>
    <div class="overlay" id="loader_agg">
163     
    </div>
165 </div>
    <div id="result_overall">
167     <br><br><h3>(1) Please select the parameter in the select fields from
        left to right.<br><br>
        (2) Choose the diagram display possibilities:
169     <ul>
        <li>Line: display of the mean values of all involved samples per
            timestamp connected by a line.</li>
171     <li>Scatter: display of all involved samples according to there
            timestamp.</li>
        </ul>
173     <br>(3) And then press the button "Show Results"...</h3>
    </div>
175 </div>
</div>
177
<div class="impressum">
179     Swiss Feed Database Version 2.0, Facharbeit K. Kruse, August 2011
</div>
181
</body>
183 </html>

```

Listing A.2: styles-feeddb.css

```

1 /*Annotation: the size of every <div> element is set in relation to its
   superior element by %. The highest element is the browser window (<html>
   element) and therefore the page will adapted to the browser's size */
3
4 body, html{
5     width: 99%;
6     height: 99%;
7     margin: 0px;
8     padding: 0px;
9     font-family: Verdana, Arial, Helvetica, sans-serif;
10    font-size: 11px;
11    text-align: left;
12 }
13
14 h1{
15     text-align: center;
16 }
17
18 h3{
19     font-weight: bold;

```

```
font-size: 100%;
21 color: #DC3C28;
margin-top: 2px;;
23 margin-bottom: -10px;
text-align: center;
25 }

27 .overlay{
position: absolute;
29 height: 0%;
width: 0%;
31 top: 45%;
margin-top: -25px;
33 left: 45%;
display: none;
35 opacity: 1;
}

37 .background{
position: relative;
39 height:100%;
width: 100%;
41 display: block;
43 overflow: auto;
}

45 .activdiv{
47 cursor: pointer;
background-color: #fff;
49 }

51 .page{
position: relative;
53 width: 100%;
height: 99%;
55 margin: 0px auto;
}

57 .title{
59 height: 80px; /* Dependent on the logo size */
}

61 .selection{
63 height: 23%;
width: 100%;
65 background-color: #FFFFFF;
border-bottom: 1px solid #000;
67 align: center;
overflow: auto;
69 }

71 .selection div{
float: left;
73 margin-left: 10px;

75 }
```



```
77 .selection select{
    min-width: 100px;
79 margin-left: 2px;
    margin-right: 2px;
81 margin-bottom: 2px;
    }
83
84 .SFseparator{
85 height: 100%;
    overflow: hidden;
87
88 }
89
90 #output{
91 width: 100%;
    height: 65%;
93 overflow: auto;
    opacity: 1;
95 }
96
97
98 #result_overall{
99 position: absolute;
    width: 100%;
101 height: 100%;
    text-align: center;
103 display: block;
    opacity: 1;
105 }
106
107 /* the following subfield are forced into a two-dimensional distribution by
    using the float attribute (erases the linebreak after the <div>) in a
    relative position.*/
108
109 #result_list{
110 width: 30%;
111 height: 95%;
    margin-top: 10px;
113 margin-right: 1%;
    margin-bottom: 10px;
115 float: left;
    position: relative;
117 }
118
119 #result_map{
120 width: 30%;
121 height: 95%;
    margin-top: 10px;
123 margin-right: 1%;
    margin-bottom: 10px;
125 float: left;
    position: relative;
127 }
128
129 #result_diagram{
130 width: 37%;
131 height: 45%;
```

```

margin-top: 10px;
133 margin-right: 1%;
margin-bottom: 10px;
135 float: right;
position: relative;
137 }

139 #result_aggregate{
width: 37%;
141 height: 46%;
margin-top: 10px;
143 margin-right: 1%;
margin-bottom: 10px;
145 float: right;
position: relative;
147 }

149 .impressum{
position: absolute;
151 width: 100%;
clear: left; /* clears the 'float: left' from above */
153 bottom: 0;
border-top: 1px solid #000;
155 padding: 5px;
}

```

Listing A.3: class-selectfield.php

```

<!-- Definition of Class Select Field -->
2 <?php
class selectField{
4 public $name=0;
public $sql_columnName;
6 public $sql_tableName;
public $sql_joinToFactTable;
8
public $result=0;
10 public $numrows=0;

12 public $nextSF=null;
public $prevSF=null;
14

16 public function selectField($fieldName){
$this->name=$fieldName;
18 }

20 public function setOriginColumn($columnName){
$this->sql_columnName = $columnName;
22 }

24 public function setOriginTable($tableName){
$this->sql_tableName = $tableName;
26 }

28 public function setJoinToFactTable($joinExpression){
$this->sql_joinToFactTable = $joinExpression;
}

```

```
30 }
31
32 public function setNextSF($nextSelectField){
33     $this->nextSF = $nextSelectField;
34     $this->nextSF->prevSF = $this;
35 }
36
37 public function doInitialQuery($connName){
38     $query = "SELECT DISTINCT ".$this->sql_columnName." FROM fact_table, ".
39         $this->sql_tableName." WHERE ".$this->sql_joinToFactTable." ORDER BY
40         ".$this->sql_columnName;
41     $this->result = pg_query($connName, $query);
42     if(!$this->result){
43         die();
44     }
45     $this->numrows = pg_numrows($this->result);
46 }
47
48 public function printField(){
49     $next = $this->nextSF;
50     $sf_all_prevName = null;
51     $sf_all_nextName = null;
52
53     $array_tables;
54     $sql_all_tableName = null;
55     $sql_all_joinToFactTable = null;
56
57     $sql_select = "SELECT DISTINCT ".$next->sql_columnName;
58     $sql_orderBy = " ORDER BY ".$next->sql_columnName;
59
60     $visit = $next;
61
62     while($visit != null){
63         $sf_all_prevName .= "-sep-".$visit->name;
64         $array_tables[$visit->sql_tableName] = $visit->sql_joinToFactTable;
65
66         $visit = $visit->prevSF;
67     }
68
69     foreach($array_tables as $key => $value){
70         $sql_all_tableName .= "-sep-".$key;
71         $sql_all_joinToFactTable .= "-sep-".$value;
72     }
73
74     $sf_all_prevName = substr($sf_all_prevName, 5);
75     $sql_all_tableName = substr($sql_all_tableName, 5);
76     $sql_all_joinToFactTable = substr($sql_all_joinToFactTable, 5);
77
78     $visit = $next->nextSF;
79
80     while($visit != null){
81         $sf_all_nextName .= "-sep-".$visit->name;
82         $visit = $visit->nextSF;
83     }
84     $sf_all_nextName = substr($sf_all_nextName, 5);
85
86     echo "<select name=' $this->name' id=' $this->sql_columnName' multiple='
```

```

        multiple' size=10 onChange='js1_getNewOptions("\$sf_all_prevName\",
        \"\$sf_all_nextName\", \"\$next->name\", \"\$sql_all_tableName\", \"
        \$sql_all_joinToFactTable\", \"\$sql_select\", \"\$sql_orderBy\")'><br>
86 <option value=\"all\">ALL.....</option><br>";
88 for($ri = 0; $ri < $this->numrows; $ri++) {
    $row = pg_fetch_array($this->result, $ri);
90     if($row[0] != ""){
        echo "<option value=\"\", $row[0], \"\">\", $row[0], "</option><br>";
92     }else{
        echo "<option value=\"empty\">No Details</option><br>";
94     }
    }
    echo "</select>";
}

96 public function printLastField(){
98     echo "<select name='\$this->name' id='\$this->sql_columnName' multiple='
        multiple' size=10><br>
        <option value=\"all\">ALL.....</option><br>";
100     for($ri = 0; $ri < $this->numrows; $ri++) {
        $row = pg_fetch_array($this->result, $ri);
102         echo "<option value=\"\", $row[0], \"\">\", $row[0], "</option><br>";
        }
104     echo "</select>";
}

106 public function printQueryButton(){
108     $sf_all_prevName = $this->name;

110     $array_tables;
    $sql_all_tableName = null;
112     $sql_all_joinToFactTable = null;

114     $visit = $this;

116     while($visit != null){
        $sf_all_prevName .= "-sep-".$visit->name;
118         $array_tables[$visit->sql_tableName] = $visit->sql_joinToFactTable;

120         $visit = $visit->prevSF;
        }

122     foreach($array_tables as $key => $value){
124         $sql_all_tableName .= "-sep-".$key;
        $sql_all_joinToFactTable .= "-sep-".$value;
126     }
    $sql_all_tableName = substr($sql_all_tableName, 5);
128     $sql_all_joinToFactTable = substr($sql_all_joinToFactTable, 5);

130     echo "<input type='button' name='showResults' value='Show Results'
        onClick='js2_getResult(\"\$sf_all_prevName\", \"\$sql_all_tableName
        \", \"\$sql_all_joinToFactTable\")'>";
    }
132 }
?>

```

Listing A.4: pg-dbinform.php

```
1 <?php
2 $host="pg.ifi.uzh.ch";
3 $port="5432";
4 $username=[-insert username-];
5 $password=[-insert password-];
6 $database=[-insert database name-];
7 ?>
```

A.2. General JavaScripts

Listing A.5: jsA_global-variables.js

```
1 // **** ajax-objects ****
2 var g_ajaxOne = null;
3 var g_ajaxTwo = null;
4 var g_ajaxThree = null;
5 var g_ajaxFour = null;
6
7 // **** google table ****
8 var g_sampleData = null;
9 var g_staticColCount = null;
10
11 // **** google maps ****
12 var g_map = null;;
13 var g_markerArray = null;
14 var g_bounds;
15 var g_infowindow;
16
17 // **** google diagram ****
18 var g_chartData = null;
19 var g_chart = null;
20
21 // **** HTML table (for cell highlightning) ****
22 var g_cellOne = null;
23 var g_cellTwo = null;
24 var g_cellThree = null;
25
26 // **** google table and HTML table ****
27 var g_nutrientArray = 0;
```

Listing A.6: jsB_ajax-object.js

```
1 /* AJAX: get the HTTP Object */
2
3 function jsB_getHTTPObject(){
4     if (window.ActiveXObject) return new ActiveXObject("Microsoft.XMLHTTP");
5     else if (window.XMLHttpRequest) return new XMLHttpRequest();
6     else {
7         alert("Your browser does not support AJAX.");
8         return null;
9     }
10 }
```

Listing A.7: jsC_loader.js

```

function jsC_loadLoader(loader, element){
2  var background = document.getElementById(element);
   background.style.opacity = 0.7;
4  var loader = document.getElementById(loader);
   loader.style.display= "block";
6  }

8  function jsC_unloadLoader(loader, element){
   var background = document.getElementById(element);
10  background.style.opacity = 1;
   var loader = document.getElementById(loader);
12  loader.style.display= "none";
   }

```

Listing A.8: jsD_sql-from-where.js

```

1  function jsD_createFromWhereQuery(inputSelectFields, sql_all_tableName,
   sql_all_joinToFactTable){

3  var array_inputField = inputSelectFields.split("-sep-");
   var array_sql_table = sql_all_tableName.split("-sep-");
5  var array_sql_join = sql_all_joinToFactTable.split("-sep-");

7  var table_count = array_sql_table.length;
   var field_count = array_inputField.length;

9

11  var sql_from = " FROM fact_table, " + array_sql_table[0];
   var sql_where = " WHERE " + array_sql_join[0];
   var sql_whereIn = new String();

13

15  for(var i= 1; i < table_count; i++){
   sql_from += "," + array_sql_table[i];
   sql_where += " AND " + array_sql_join[i];
17  }

19  for(var j= 1; j < field_count; j++){

21     var temp = document.forms["testForm"].elements[array_inputField[j]].
       length;
       var in_params = new String();
23     var is_null = false;

25     if(document.forms["testForm"].elements[array_inputField[j]].options[0].
       selected==false){
       for(var i= 1; i < temp; i++){
27         if(document.forms["testForm"].elements[array_inputField[j]].options[i].
           selected==true){
           if(document.forms["testForm"].elements[array_inputField[j]].options[i]
           ].value == "empty"){
29             is_null = true;
           }else{
31             in_params = in_params + "','" + document.forms["testForm"].elements[
               array_inputField[j]].options[i].value;
           }
33         }
       }
   }

```

```

35     if(in_params != 0){
        in_params = in_params+"'";
37     in_params = in_params.substring(2);

39     sql_whereIn += " AND (" + document.forms["testForm"].elements[
        array_inputField[j]].id + " IN (" + in_params + ")";
        if(is_null == true){
41         sql_whereIn += " OR " + document.forms["testForm"].elements[
            array_inputField[j]].id + " IS NULL)";
        }else{
43         sql_whereIn += ")";
        }
45     }else{
        if(is_null == true){
47         sql_whereIn += " AND " + document.forms["testForm"].elements[
            array_inputField[j]].id + " IS NULL";
        }
49     }
51 }
    var sql_from_where = sql_from + sql_where + sql_whereIn;
53 return sql_from_where;
}

```

A.3. Modular JavaScripts

Listing A.9: js1_update_selectfield.js

```

/* AJAX: send asynchronous query request to php */
2
function js1_getNewOptions(inputSelectFields, emptySelectFields,
    outputSelectField, sql_all_tableName, sql_all_joinToFactTable,
    sql_select, sql_orderBy){
4     var sql_from_where = jsD_createFromWhereQuery(inputSelectFields,
        sql_all_tableName, sql_all_joinToFactTable);
        var sql_query = sql_select + sql_from_where + sql_orderBy;
6     var query = "query=" + sql_query;

8     g_ajaxOne = jsB_getHTTPObject();
        if (g_ajaxOne != null) {
10         g_ajaxOne.open("POST", "ajax-pg-options.php", true);
            g_ajaxOne.setRequestHeader("Content-Type", "application/x-www-form-
                urlencoded");
12         g_ajaxOne.setRequestHeader("Content-length", query.length);
            g_ajaxOne.setRequestHeader("Connection", "close");
14         g_ajaxOne.send(query);

16         g_ajaxOne.onreadystatechange = function() { js1_setOptions(
            outputSelectField, emptySelectFields);}
        }
18 }

20

22

```

```
/* AJAX callback: get result and change to new options */
24
function js1_setOptions(outputSelectField, emptySelectFields){
26  /*readyState 4 = successfull finished transmission of return value. */
  if(g_ajaxOne.readyState == 4){
28    var temp = g_ajaxOne.responseText;
    var array = 0;
30    temp = temp.substring(4);
    if(temp != ""){
32      array = temp.split("-sep-");
    }
34
    /* --1-- delete options of SelectField */
36    var s_count = document.forms["testForm"].elements[outputSelectField].
      length;
    document.forms["testForm"].elements[outputSelectField].options[0].
      selected = false;
38    for(var j = s_count-1; j >= 1; j--){
      document.forms["testForm"].elements[outputSelectField].options[j] = null
        ;
40    }

42    /* --2-- fill new options in SelectField */
    var a_count = array.length;
44    if(a_count>0){
      for(var k = 0; k < a_count; k++){
46        if(array[k]!=""){
          document.forms["testForm"].elements[outputSelectField].options[k+1] =
            new Option(array[k],array[k]);
48        }else{
          document.forms["testForm"].elements[outputSelectField].options[k+1] =
            new Option("No Details", "empty");
50        }
      }
52    }

54    /* --3-- empty all following SelectFields */
    var tempTwo = emptySelectFields;
56    var arrayTwo = tempTwo.split("-sep-");
    var a2_count = arrayTwo.length;
58

    for(var l = 0; l < a2_count; l++){
60      var s2_count = document.forms["testForm"].elements[arrayTwo[l]].length;
      document.forms["testForm"].elements[arrayTwo[l]].options[0].selected =
        false;
62      for(var j = s2_count-1; j >= 1; j--){
        document.forms["testForm"].elements[arrayTwo[l]].options[j] = null;
64      }
    }
66  }
}
```


Listing A.10: js2_result_coordinator.js

```
1  /* AJAX: send all asynchronous requests, required for the application
   output: */

3  function js2_getResult(inputSelectFields, sql_all_tableName,
   sql_all_joinToFactTable){
   /* initiate "is loading" status: */
5  var resultdisplay = document.getElementById("output");
   resultdisplay.style.opacity= 1;
7  var dummydisplay = document.getElementById("result_overall");
   dummydisplay.style.display = "none";

9

11 jsC_loadLoader("loader_map", "map_canvas");
   jsC_loadLoader("loader_sidebar", "sidebar");
   jsC_loadLoader("loader_chart", "chart_div");
13 jsC_loadLoader("loader_agg", "aggtable_space");

15 /* get dynamic sql (by checking the select fields): */
   var sql_from_where = jsD_createFromWhereQuery(inputSelectFields,
   sql_all_tableName, sql_all_joinToFactTable);

17

19 /* store all selected nutrients in an array for later use in the tables (
   if="ALL" selected) */
   g_nutrientArray = new Array();
   var n = 0;
21 var selectFieldLength = document.forms["testForm"].elements["nutrient[]"].
   length;

23 if(document.forms["testForm"].elements["nutrient[]"].options[0].selected==
   true){
   for(var i = 1; i < selectFieldLength; i++){
25     g_nutrientArray[n] = document.forms["testForm"].elements["nutrient[]"].
       options[i].value;
       ++n;
27     }
   }else{
29     for(var i = 1; i < selectFieldLength; i++){
       if(document.forms["testForm"].elements["nutrient[]"].options[i].selected
       ==true){
31         g_nutrientArray[n] = document.forms["testForm"].elements["nutrient[]"].
           options[i].value;
           ++n;
33         }
       }
35     }

37 /* Module M3: GOOGLE MAPS & GOOGLE TABLE (external asynchronous function
   call): */
   google.load("visualization", "1", {packages:["table"], callback: function
   () {js2_sendMapListAjax(sql_from_where);}});

39

41 /* Module M4: GOOGLE DIAGRAM (external asynchronous function call): */
   var chartType = "line";
   if(document.forms["testForm"].elements['charttype'][1].checked == true){
43     chartType = "scatter";
   }
45 google.load("visualization", "1", {packages:["corechart"], callback:
```

```
function() {js2_sendChartAjax(sql_from_where, chartType);});

47 /* Module M5: HTML-TABLE (internal asynchronous function call): */
g_ajaxFour = jsB_getHTTPObject();
49 if (g_ajaxFour != null){
    g_ajaxFour.open("POST", "ajax-pg-result-aggregate.php", true);
51 g_ajaxFour.setRequestHeader("Content-Type", "application/x-www-form-
    urlencoded");
    g_ajaxFour.setRequestHeader("Content-length", sql_from_where.length+6);
53 g_ajaxFour.setRequestHeader("Connection", "close");
    g_ajaxFour.send("query="+sql_from_where);
55 g_ajaxFour.onreadystatechange = function() { js5_setAggregate();}
}
57 }

59 /* Module M3: callback with internal asynchronous function call: */

61 function js2_sendMapListAjax(sql_from_where){
    g_ajaxTwo = jsB_getHTTPObject();
63 if (g_ajaxTwo != null) {
    // for working with POST & XML-Response:
65 // (1) here: g_ajaxTwo.setRequestHeader("Content-Type", "application/x-
    www-form-urlencoded")
    // (2) php: header("Content-type: text/xml");
67
    g_ajaxTwo.open("POST", "ajax-pg-result-google.php", true);
69 g_ajaxTwo.setRequestHeader("Content-Type", "application/x-www-form-
    urlencoded");
    g_ajaxTwo.setRequestHeader("Content-length", sql_from_where.length+6);
71 g_ajaxTwo.setRequestHeader("Connection", "close");
    g_ajaxTwo.send("query="+sql_from_where);
73 g_ajaxTwo.onreadystatechange = function() { js3_setMapList();}
}
75 }

77 /* Module M4: callback with internal asynchronous function call: */

79 function js2_sendChartAjax(sql_from_where, chartType){
    g_ajaxThree = jsB_getHTTPObject();
81 if(g_ajaxThree != null){
    if(chartType == "scatter"){
83 g_ajaxThree.open("POST", "ajax-pg-result-scatterdiagram.php", true);
    }else{
85 g_ajaxThree.open("POST", "ajax-pg-result-linediagram.php", true);
    }
87 g_ajaxThree.setRequestHeader("Content-Type", "application/x-www-form-
    urlencoded");
    g_ajaxThree.setRequestHeader("Content-length", sql_from_where.length+6);
89 g_ajaxThree.setRequestHeader("Connection", "close");
    g_ajaxThree.send("query="+sql_from_where);
91 g_ajaxThree.onreadystatechange = function() { js4_setChart(chartType);}
}
93 }
```

Listing A.11: js3_result_list-map.js

```
1 /* Module M3 : callback to set the markers and the legend */
3 function js3_setMapList(){
4     /* readyState 4 = successfully finished transmission of return value. */
5     if(g_ajaxTwo.readyState == 4){
6         var resultXML = g_ajaxTwo.responseXML.documentElement;
7         var resultArray = resultXML.getElementsByTagName("marker");
8         var xml_length = resultXML.getElementsByTagName("marker").length;
9
10        /* delete old markers */
11        if (g_markerArray != null) {
12            for (i in g_markerArray) {
13                g_markerArray[i].setMap(null);
14            }
15            g_markerArray.length = 0;
16        }
17
18        /* delet old sidebar */
19        if (g_sampleData != null){
20            var prevRows = g_sampleData.getNumberOfRows();
21            if(prevRows != 0){
22                g_sampleData.removeRows(0,prevRows);
23            }
24            var prevColumns = g_sampleData.getNumberOfColumns();
25            if(prevColumns > g_staticColCount){
26                g_sampleData.removeColumns(g_staticColCount,prevColumns-1);
27            }
28        }
29
30        /* fill with new information: */
31        /* --1-- create "static" objects (map, bounds of map, infowindow & google
32            .datatable) */
33        g_bounds = new google.maps.LatLngBounds();
34
35        if(g_map == null){
36            g_map = new google.maps.Map(document.getElementById("map_canvas"), {
37                mapTypeId: google.maps.MapTypeId.TERRAIN});
38            g_markerArray = new Array();
39
40            g_infowindow = new google.maps.InfoWindow({
41                //size: new google.maps.Size(50,50),
42                position: latlng
43            });
44
45            g_sampleData = new google.visualization.DataTable();
46            g_sampleData.addColumn('string', 'LIMS-Nr.');
```

```
47            g_sampleData.addColumn('string', 'Canton');
48            g_sampleData.addColumn('string', 'PLZ');
49
50            /* number of static columns (here: 3) for later use stored in global
51                variabel "g_staticColCount"! */
52            g_staticColCount = g_sampleData.getNumberOfColumns();
53        }
54        /* --1.2-- fill add all needed nutrient columns to the datatable: */
55        var tableColCount = g_staticColCount;
```

```

55 for(var n = 0; n < g_nutrientArray.length; n++){
    g_sampleData.addColumn('number', g_nutrientArray[n]);
    ++tableColCount;
57 }

59 /* --2-- transform the samples into g_map markers and datatable row
    entries; */
for(var j= 0; j < xml_length; j++){
61 /* --2.1-- create basic data for every marker/entry: */
    var name = resultArray[j].getAttribute("name");
63     var middle = name.length/2;
    var maskedName = name.substr(middle);
65     var x = new String();
    for(var l= 0; l < middle; l++){
67         x += "X";
    }
69     maskedName = x + maskedName;

71     g_sampleData.addRows(1);
    g_sampleData.setRowProperty(j, "lims", name);
73     g_sampleData.setValue(j, 0, maskedName);
    g_sampleData.setValue(j, 1, resultArray[j].getAttribute("canton"));
75     g_sampleData.setValue(j, 2, resultArray[j].getAttribute("plz"));

77

    /* --2.2.-- create content of marker and fill the datatable row at the
        same time (iterate the nutrient quantities): */
79     var html = "<b>LIMS-Nr. " + maskedName + "</b><br>" + resultArray[j].
        getAttribute("plz") + " " + resultArray[j].getAttribute("city") + "
        (" + resultArray[j].getAttribute("canton") + "<br><br><table frame
        =\"void\">";

81     var nutrientArray = resultArray[j].getElementsByTagName("nutrient");
    var nutrient_length = resultArray[j].getElementsByTagName("nutrient").
        length;

83

    var colCheckNumber = g_staticColCount;
85     for(var k= 0; k < nutrient_length; k++){
        var quantity = parseFloat(nutrientArray[k].getAttribute("quantity"))
87         html += "<tr><td align=\"left\">Value '"+ nutrientArray[k].getAttribute
            ("abbreviation") + "':</td><td align=\"right\">"+ quantity + "</td
            ></tr>";

89         var colCheckFlag = true;
        while((colCheckNumber < tableColCount) && colCheckFlag){
91             if(g_sampleData.getColumnLabel(colCheckNumber) == nutrientArray[k].
                getAttribute("abbreviation")){

93                 g_sampleData.setValue(j, colCheckNumber, quantity);
                colCheckFlag = false;
95                 ++colCheckNumber;
            }else{
97                 ++colCheckNumber;
            }
99         }
    }
    html += "</table>";
101

```

```
103  /* --2.3.-- create marker with proper position (slightly shifted for
      better display) and add name and content: */
105  if(resultArray[j].getAttribute("lat") != "error"){
      var latitude = parseFloat(resultArray[j].getAttribute("lat")) +
          parseFloat((Math.random()- Math.random())/1000);
      var longitude = parseFloat(resultArray[j].getAttribute("lng")) + -
          parseFloat((Math.random()- Math.random())/1000);
107
      var latlng = new google.maps.LatLng(latitude, longitude);
109
111  var marker = new google.maps.Marker({
      map: g_map,
113  position: latlng,
      icon: "images/red-dot.png",
115  shadow: "images/msmarker.shadow.png",
      content: html,
117  title: "LIMS-Nr. "+name
  });
119  /* --2.4-- add position of marker to the bound and the marker to the
      array -> see delet old marker */
  g_bounds.extend(latlng);
121  g_markerArray[j] = marker;

123  /* --2.5-- create Listener to evey marker (infowindow-popup) */
  google.maps.event.addListener(g_markerArray[j], 'mouseover', function()
      {
125  g_infowindow.setContent(this.content);
      g_infowindow.open(g_map,this);
127  });
  google.maps.event.addListener(g_markerArray[j], 'mouseout', function()
      {
129  g_infowindow.close(g_map,this);
      });
131  }
  }
133
  jsC_unloadLoader("loader_map", "map_canvas");
135  jsC_unloadLoader("loader_sidebar", "sidebar");

137  /* --3-- print sidebar (sample enlistment) and create and popup-event-
      listener */
  sidebarTable = new google.visualization.Table(document.getElementById('
      sidebar'));
139  sidebarTable.draw(g_sampleData, {showRowNumber: true});

141  google.visualization.events.addListener(sidebarTable, 'select',
      function() {
143
          var selection = sidebarTable.getSelection();
145          for (var i = 0; i < selection.length; i++) {
              var item = selection[i];
147          }
          var wanted = "LIMS-Nr. "+g_sampleData.getRowProperty(item.row, "
              lims");
149          var g_markerArraylength = g_markerArray.length;
```

```

151     var marker = null;
        for(var j = 0; j < g_markerArraylength; j++){
153     if(g_markerArray[j].title == wanted){
        marker = g_markerArray[j];
155     }
        }
157     google.maps.event.trigger(marker, 'mouseover');
        });
159
    g_map.fitBounds(g_bounds);
161 }
}

```

Listing A.12: js4_result_diagram.js

```

/* Module M4: callback to create the line or the scatter chart. */
2
function js4_setChart(chartType){
4 /* readyState 4 = successfully finished transmission of return value. */
    if(g_ajaxThree.readyState == 4){
6     var resultstring = g_ajaxThree.responseText;
        if(resultstring != null){
8         resultstring = resultstring.substring(5);
        }
10     var array = resultstring.split("-sep-");
        var length = array.length;
12
        /* either create new DataTable or empty the previous one: */
14     if(g_chartData == null){
        g_chartData = new google.visualization.DataTable();
16     g_chartData.addColumn('date', 'Age');
        }else{
18     var prevRows = g_chartData.getNumberOfRows();
        if(prevRows != 0){
20     g_chartData.removeRows(0,prevRows);
        }
22     var prevColumns = g_chartData.getNumberOfColumns();
        if(prevColumns > 1){
24     g_chartData.removeColumns(1,prevColumns-1);
        }
26     }

28     /* traverse array with resultstupels and insert columns and rows: */
        var nutrient = null;
30     var nullCount = 0;
        var minDate = "3000-12-31";
32     var maxDate = "1000-12-31";

34     for(var i = 0; i < length; i++){
        var secondArray = array[i].split("-s-");
36     if(secondArray[0] != nutrient){
        g_chartData.addColumn('number', secondArray[0]);
38     nutrient = secondArray[0];
        ++nullCount;
40     }
        if(secondArray[1] < minDate){

```

```
42     minDate = secondArray[1];
43     }
44     if(secondArray[1] > maxDate){
45         maxDate = secondArray[1];
46     }
47     var tempDate = new Date(secondArray[1].substring(0,4), (parseInt (
48         secondArray[1].substring(5,7),10)-1), secondArray[1].substring(8,10));
49
50     g_chartData.addRow(1);
51     g_chartData.setValue(i,0,tempDate);
52     g_chartData.setValue(i,nullCount,parseFloat(secondArray[2]));
53     if(chartType == "scatter"){
54         g_chartData.setRowProperty(i, "lims", secondArray[3]);
55     }
56     }
57
58     /* calculate the margins for the chart-hAxis (maxValue + 1 day, minValue
59     - 1 day) */
60     var hAxisMin = new Date(minDate.substring(0,4), (parseInt(minDate.
61         substring(5,7),10)-1), (parseInt(minDate.substring(8,10),10)-1));
62     var hAxisMax = new Date(maxDate.substring(0,4), (parseInt(maxDate.
63         substring(5,7),10)+1), (parseInt(maxDate.substring(8,10),10)+1));
64
65     jsC_unloadLoader("loader_chart", "chart_div");
66
67     if(chartType == "scatter"){
68         /* ---- draw scatter chart with event "click on scatter pont - open maps
69         marker infowindow" */
70         var g_chart = new google.visualization.ScatterChart(document.
71             getElementById('chart_div'));
72         g_chart.draw(g_chartData, {chartArea: {width: "70%", height:"75%", left
73             :40},
74             title: 'Temporal Value Distribution [click on the dots to connect to
75             the map]',
76             hAxis: {title: 'Timeline', minValue: hAxisMin, maxValue: hAxisMax},
77             vAxis: {title: 'Quantity'},
78             //lineWidth: 0, /* scatter chart = line chart without line */
79             pointSize: 4
80         });
81
82         google.visualization.events.addListener(g_chart, 'select',
83             function() {
84
85                 var selection = g_chart.getSelection();
86                 for (var i = 0; i < selection.length; i++) {
87                     var item = selection[i];
88                 }
89                 var wanted = "LIMS-Nr. "+ g_chartData.getRowProperty(item.row, "lims")
90                     ;
91                 var markerArraylength = g_markerArray.length;
92                 var marker = null;
93                 for(var j = 0; j < markerArraylength; j++){
94                     if(g_markerArray[j].title == wanted){
95                         marker = g_markerArray[j];
96                     }
97                 }
98                 google.maps.event.trigger(marker, 'mouseover');
```

```

90     });
91   }else{
92     /* ---- draw line chart without any additional events */
93     var g_chart = new google.visualization.ScatterChart(document.
94       getElementById('chart_div'));
95     g_chart.draw(g_chartData, {chartArea: {width: "70%", height:"75%", left
96       :40},
97       title: 'Temporal Mean Value Distribution',
98       hAxis: {title: 'Timeline'},
99       vAxis: {title: 'Quantity'},
100      curveType: 'none',
101      lineWidth: 2,
102      pointSize: 1
103    });
104  }

```

Listing A.13: js5_result_aggregate.js

```

/* Module M5: callback to create the aggregation table */
2
function js5_setAggregate(){
4  /* readyState 4 = successfully finished transmission of return value.*/
5  if(g_ajaxFour.readyState == 4){
6    var temp = g_ajaxFour.responseText;
7    var array = 0;
8    var html = new String();
9
10   /* --1-- delete existing Aggregate Table */
11   var aggTableSpace = document.getElementById('aggtable_space');
12   while(aggTableSpace.firstChild){
13     aggTableSpace.removeChild(aggTableSpace.firstChild);
14   }
15
16   /* --2-- create new html output */
17   temp = temp.substring(5);
18   if(temp != ""){
19     array = temp.split("-sep-");
20
21     html+="<h2>Statistical information of nutrients</h2><br>";
22     html+="<table border=\"2\" frame=\"void\" rules=\"all\">";
23     html+="<tr>";
24     html+="<th>Nutrient</th>";
25     html+="<th>Minimum</th>";
26     html+="<th>Maximum</th>";
27     html+="<th>Average</th>";
28     html+="<th> 2&sigma;* </th>";
29     html+="<th>Count</th>";
30     html+="<th></th>";
31     html+="<th>avg (&mu;> +2&sigma;) </th>";
32     html+="<th>avg (&mu;&plusmn;2&sigma;) </th>";
33     html+="<th>avg (&mu;< -2&sigma;) </th>";
34     html+="</tr>";
35
36     var secondArray;
37     var nutrientNumber = 0

```



```

38   for(var i = 0; i < array.length; i++){
        secondArray = array[i].split("-s-");
40
        /*by use of global nutrientNumber the html table (row) and the google
           visualization table (col) are linkable */
42   while((secondArray[0] != g_nutrientArray[nutrientNumber]) && (
           nutrientNumber < g_nutrientArray.length)){
           nutrientNumber++;
44   }

46   html+="<tr>\n";
       html+=" <th><div class=\"activdiv\" onclick=\"js5_highlightStatGroups
           (+ (nutrientNumber+g_staticColCount) +\", \"+(parseFloat(secondArray
           [3])+parseFloat(secondArray[4])) + \", \" + (parseFloat(secondArray[3])
           -parseFloat(secondArray[4])) + \")\", \" onmouseover=\"this.style.
           backgroundColor='#E0E6F8';\" onmouseout=\"this.style.backgroundColor
           ='#fff';\">"+ secondArray[0]+</div></th><td align='right'>"
48   + secondArray[1]+</td><td align='right'>"
       + secondArray[2]+</td><td align='right'>"
50   + secondArray[3]+</td><td align='right'>"
       + secondArray[4]+</td><td align='right'>"
52   + secondArray[5]+</td>";
       html+="<td></td><td align='right' id=\""+nutrientNumber+"-under2Sigma
           \"></td><td align='right' id=\""+nutrientNumber+"-2sigmaRange\"></td
           ><td align='right' id=\""+nutrientNumber+"-over2Sigma\"></td>";
54   html+="</tr>";

56   }
       html+="</table>";
58   html+="<br><p>* Standard deviation of a population</p>";

60   /* --3-- create new Aggregate Table */
       jsC_unloadLoader("loader_agg", "aggtable_space");
62
       var aggTable = document.createElement('div');
64   aggTable.innerHTML = html;
       aggTableSpace.appendChild(aggTable);
66   }
       }
68 }

70 /* Module M5+: additional feature triggered by click on nutrient */

72 function js5_highlightStatGroups(col, highSigmaValue, lowSigmaValue ){
       jsC_loadLoader("loader_agg", "aggtable_space");
74
       var rows = g_sampleData.getNumberOfRows();
76   var g_markerArraylength = g_markerArray.length;

78   var sumUnderLowSigma = 0;
       var sumOverHighSigma = 0;
80   var sumSigmaInterval = 0;
       var countUnderLowSigma = 0;
82   var countOverHighSigma = 0;
       var countSigmaInterval = 0;
84
       for(var j = 0; j < rows; j++){

```

```

86  var wanted = "LIMS-Nr. " + g_sampleData.getRowProperty(j, "lims");
    var cell = g_sampleData.getValue(j, col);
88
    var marker = null;
90  for(var k = 0; k < g_markerArraylength; k++){
    if(g_markerArray[k].title == wanted){
92    marker = g_markerArray[k];
    }
94  }
    if(cell != null){
96    if(cell < lowSigmaValue){
    marker.setIcon("images/dark-high-dot.png");
98    marker.setShadow("images/msmarker.shadow.png");
    sumUnderLowSigma += cell;
100   ++countUnderLowSigma;
    }else if(cell > highSigmaValue){
102    marker.setIcon("images/light-high-dot.png");
    marker.setShadow("images/msmarker.shadow.png");
104    sumOverHighSigma += cell;
    ++countOverHighSigma;
106    }else if((cell >= lowSigmaValue) && (cell <= highSigmaValue)){
    marker.setIcon("images/red-dot.png");
108    marker.setShadow("images/msmarker.shadow.png");
    sumSigmaInterval += cell;
110    ++countSigmaInterval;
    }else{
112    marker.setIcon("images/grey.png");
    }
114    }else{
    marker.setIcon("images/grey.png");
116    }
    }
118  var avgUnderLowSigma = sumUnderLowSigma/countUnderLowSigma;
    var avgOverHighSigma = sumOverHighSigma/countOverHighSigma;
120  var avgSigmaInterval = sumSigmaInterval/countSigmaInterval;

122  /* delet old highlightning */
    if(g_cellOne != null){
124    g_cellOne.style.backgroundColor= 'transparent';
    g_cellTwo.style.backgroundColor= 'transparent';
126    g_cellThree.style.backgroundColor= 'transparent';
    }
128
    /* set value */
130  g_cellOne = document.getElementById((col - g_staticColCount)+"-under2Sigma
    ");
    g_cellOne.innerHTML = Math.round(avgUnderLowSigma * 1000)/1000;
132
    g_cellTwo = document.getElementById((col - g_staticColCount)+"-2sigmaRange
    ");
134  g_cellTwo.innerHTML = Math.round(avgSigmaInterval * 1000)/1000;

136  g_cellThree = document.getElementById((col - g_staticColCount)+"-
    over2Sigma");
    g_cellThree.innerHTML = Math.round(avgOverHighSigma * 1000)/1000;
138
    /* set current highlightning */

```

```

140 g_cellOne.style.backgroundColor= '#819FF7';
    g_cellTwo.style.backgroundColor= '#DC3C28';
142 g_cellThree.style.backgroundColor= '#F3E2A9';

144 jsC_unloadLoader("loader_agg", "aggtable_space");
    }

```

A.4. Ajax triggered PHP scripts

Listing A.14: ajax-pg-options.php

```

1 <?php
  if (isset($_POST['query'])) {
3     require("pg-dbinfop.php");

5     $c = stripslashes($_POST['query']);

7     $conn=pg_connect('host='.$host.' port='.$port.' dbname='.$database.' user
        ='.$username.' password='.$password.' sslmode=require');
    if (!$conn) {
9         echo "Error: no connection to database";
        die();
11    }
    $result = pg_query($conn, $c);
13    if (!$result) {
        echo "Error: query wasn't successful (no result)";
15    } else {
        $numrows = pg_numrows($result);
17        $array = pg_fetch_all_columns($result, 0);
        $string = implode("-sep-", $array);
19
        echo $string;
21    }
    pg_close($conn);
23 } else {
    echo "Error: AJAX didn't transmit (POST-transport failed)";
25 }
?>

```

Listing A.15: ajax-pg-result-google.php

```

<?php
2 if(isset($_POST['query'])) {
    $query_one = "SELECT DISTINCT lims_number, company_postal_code,
        company_city, canton_de, latitude, longitude ".stripslashes($_POST['
        query'])." ORDER BY canton_de, company_postal_code, lims_number";
4    $query_two = "SELECT DISTINCT lims_number, abbreviation_de, quantity ".
        stripslashes($_POST['query'])." ORDER BY lims_number";

6    require("pg-dbinfop.php");

8    define("MAPS_HOST", "maps.google.ch");

10   define("MAPS_STATUS_OK",           "OK");
    define("MAPS_STATUS_ZERO",        "ZERO_RESULTS");

```

```

12  define("MAPS_STATUS_OVER_QUERY_LIMIT", "OVER_QUERY_LIMIT");
    define("MAPS_STATUS_REQUEST_DENIED", "REQUEST_DENIED");
14  define("MAPS_STATUS_INVALID_REQUEST", "INVALID_REQUEST");

16  function parseToXML($htmlStr) {
    $xmlStr=str_replace('<','&lt;',$htmlStr);
18  $xmlStr=str_replace('>','&gt;',$xmlStr);
    $xmlStr=str_replace('"','&quot;',$xmlStr);
20  $xmlStr=str_replace("'",'&#39;',$xmlStr);
    $xmlStr=str_replace("&","&amp;",$xmlStr);
22  return $xmlStr;
    }

24
    /* Start XML file, create parent node */
26  $dom = new DOMDocument("1.0");
    $node = $dom->createElement("markers");
28  $parnode = $dom->appendChild($node);

30  /* Connection to PostgreSQL-DB */
    $conn=pg_connect('host='.$host.' port='.$port.' dbname='.$database.' user
        ='$username.' password='.$password.' sslmode=require');
32  if (!$conn) {
    echo "Error: no connection to database";
34  die();
    }

36
    $result = pg_query($conn, $query_one);
38  $numrows = pg_numrows($result);

40  if (!$result) {
    echo "Error: query wasn't successful (no result)";
42  die();
    }

44

46  /* Initialize URL and delay in geocode speed */
    /* Example of possible URL-Request: http://maps.google.com/maps/api/
        geocode/json?address=1600+Amphitheatre+Parkway,+Mountain+View,+CA&
        sensor=true_or_false; */
48  $delay = 0;
    $base_url = "http://" .MAPS_HOST. "/maps/api/geocode/xml?sensor=false";
50
    header("Content-type: text/xml");
52

    /* Iterate through the rows, geocoding each address and update values in
        Database */
54  for($ri = 0; $ri < $numrows; $ri++) {
    $row = pg_fetch_array($result, $ri);

56
        $latitude = $row['latitude'];
58  $longitude = $row['longitude'];

60  /* add to xml: node for marker with attributes */
    $node = $dom->createElement('marker');
62  $newnode = $parnode->appendChild($node);
    $newnode->setAttribute("name",$row['lims_number']);
64  $newnode->setAttribute("plz", $row['company_postal_code']);

```

```
66 $newnode->setAttribute("city", $row['company_city']);
67 $newnode->setAttribute("canton", $row['canton_de']);
68 $newnode->setIdAttribute("name", true);
69
70 if(($latitude != null) && ($longitude != null)){
71     $newnode->setAttribute("lat", $latitude);
72     $newnode->setAttribute("lng", $longitude);
73
74 }else{
75
76     $geocode_pending = true;
77
78     while ($geocode_pending) {
79         $plz = $row["company_postal_code"];
80         $city = $row["company_city"];
81         $canton = $row["canton_de"];
82         $address = $plz + "+" + $city + ",+" + $canton + ",+Switzerland";
83
84         $request_url = $base_url . "&address=" . urlencode($address);
85         $xml = simplexml_load_file($request_url) or die("url not loading");
86
87         $status = $xml->status;
88
89         if (strcmp($status, MAPS_STATUS_OK) == 0) {
90             /* Successful geocode */
91             $geocode_pending = false;
92             $delay = 0;
93             $coordinates = $xml->result->geometry->location;
94
95             /* result format: Longitude, Latitude, (Altitude) */
96             $lat = $coordinates->lat;
97             $lng = $coordinates->lng;
98
99             $newnode->setAttribute("lat", $lat);
100            $newnode->setAttribute("lng", $lng);
101
102            /* $lat & $lng can be stored as float(10,6) aka numeric(10,6) with no
103               conversion: */
104            $update = "UPDATE d_origin SET latitude= ".$lat.", longitude= ".$lng
105                ." WHERE company_postal_code= ".$address;
106            $update_result = pg_query($conn, $update);
107
108            if (!$update_result) {
109                /*...ceep on running, update it next time. */
110            }
111
112            } else if (strcmp($status, MAPS_STATUS_OVER_QUERY_LIMIT) == 0) {
113                /* sent geocodes too fast, set delay plus 0.1 seconds */
114                $delay += 100000;
115            } else {
116                /* failure to geocode */
117                $geocode_pending = false;
118                $newnode->setAttribute("lat", "error");
119                $newnode->setAttribute("error", $status);
120            }
121            usleep($delay);
122        }
123    }
```

```

120     }
121     }
122
123     $result2 = pg_query($conn, $query_two);
124     $numrows2 = pg_numrows($result2);
125
126     if (!$result2) {
127         echo "Error: query wasn't successful (no result)";
128         die();
129     }else{
130         for($rj = 0; $rj < $numrows2; $rj++){
131             $row = pg_fetch_array($result2, $rj);
132             $nodeToAdd = $dom->getElementById($row['lims_number']);
133             if($nodeToAdd != null){
134                 $node_two = $dom->createElement('nutrient');
135                 $nutrient = $nodeToAdd->appendChild($node_two);
136                 $nutrient->setAttribute("abbreviation", $row['abbreviation_de']);
137                 $nutrient->setAttribute("quantity", number_format(floatval($row['
138                     quantity']),3));
139             }
140         }
141     }
142     pg_close($conn);
143
144     echo $dom->saveXML();
145 }else{
146     echo "Error, AJAX didn't transmit (POST-transport failed)";
147 }
148 ?>

```

Listing A.16: ajax-pg-result-linediagram.php

```

<?php
2 require("pg-dbinform.php");
3 if (isset($_POST['query'])){
4     $c = stripslashes($_POST['query']);
5
6     /* checks if db-relation d_time and d_nutrient are in the query, else
7     insert them: */
8     if(strpos($c,"d_time") === false){
9         $c = str_replace("FROM", "FROM d_time", $c);
10        $c = str_replace("WHERE", "WHERE id_time_fkey = time_key", $c);
11    }
12    if(strpos($c,"d_nutrient") === false){
13        $c = str_replace("FROM", "FROM d_nutrient", $c);
14        $c = str_replace("WHERE", "WHERE id_nutrient_fkey = nutrient_key", $c);
15    }
16
17    $c = "SELECT abbreviation_de, (CASE WHEN t_day IS NULL THEN to_date(
18        t_year||'-01-01', 'YYYY-MM-DD') ELSE t_day END) AS day, t_year, avg(
19        quantity)".$c." GROUP BY abbreviation_de, t_day, t_year ORDER BY
20        abbreviation_de, day";
21
22    $conn=pg_connect('host='.$host.' port='.$port.' dbname='.$database.' user
23       ='.$username.' password='.$password.' sslmode=require');
24    if (!$conn) {
25        echo "Error: no connection to database";
26    }

```

```

    die();
22 }

24 $result = pg_query($conn, $c);
    if(!$result){
26     echo "Error: query wasn't successful (no result)";
    }else{
28     $numrows = pg_numrows($result);

30     $string = null;
    for($i = 0; $i < $numrows; $i++){
32         $row = pg_fetch_array($result, $i);
        $string .= '-sep-'. $row[0];
34         if($row[1] != null){
            $string .= '-s-'. $row[1];
36         }else{
            $string .= '-s-'. $row[2].'-01-01';
38         }
        $string .= '-s-'. $row[3];
40     }
    echo $string;
42 }

44 pg_close($conn);
}else{
46     echo "Error, AJAX didn't transmit (POST-transport failed)";
}
48 ?>

```

Listing A.17: ajax-pg-result-scatterdiagram.php

```

<?php
2 require("pg-dbinform.php");
    if (isset($_POST['query'])){
4     $c = stripslashes($_POST['query']);

6     // checks if db-relation d_time and d_nutrient are in the query (needed),
        else insert them:
    if(strpos($c,"d_time") === false){
8         $c = str_replace("FROM", "FROM d_time", $c);
        $c = str_replace("WHERE", "WHERE id_time_fkey = time_key", $c);
10    }
    if(strpos($c,"d_nutrient") === false){
12        $c = str_replace("FROM", "FROM d_nutrient", $c);
        $c = str_replace("WHERE", "WHERE id_nutrient_fkey = nutrient_key", $c);
14    }

16    $c = "SELECT abbreviation_de, (CASE WHEN t_day IS NULL THEN to_date(
        t_year||'-01-01', 'YYYY-MM-DD') ELSE t_day END) AS day, t_year,
        quantity, lims_number".$c." ORDER BY abbreviation_de, day";

18    $conn=pg_connect('host='.$host.' port='.$port.' dbname='.$database.' user
       ='.$username.' password='.$password.' sslmode=require');
    if (!$conn) {
20        echo "Error: no connection to database";
        die();
22    }

```

```
24 $result = pg_query($conn, $c);
    if(!$result){
26     echo "Error: query wasn't successful (no result)";
    }else{
28     $numrows = pg_numrows($result);

30     $string = null;
    for($i = 0; $i < $numrows; $i++){
32     $row = pg_fetch_array($result, $i);
        $string .= '-sep-'. $row[0];
34     if($row[1] != null){
        $string .= '-s-'. $row[1];
36     }else{
        $string .= '-s-'. $row[2].'-01-01';
38     }
        $string .= '-s-'. $row[3];
40     $string .= '-s-'. $row[4];
    }

42     echo $string;
44 }

46 pg_close($conn);
    }else{
48     echo "Error, AJAX didn't transmit (POST-transport failed)";
    }
50 ?>
```


Bibliography

- [Achour et al. 2011] Achour, M. et al. (2011) PHP Manual. URL: <http://www.php.net/manual/de/index.php> [Last access: 2011-08-13].
- [Agridea 2011] Agridea (2011) AGRIDEA, die Schweizerische Vereinigung für die Entwicklung der Landwirtschaft und des ländlichen Raums. URL: <http://www.agridea.ch/> [Last access: 2011-08-29].
- [Agroscope 2009] Agroscope Liebefeld-Posieux (2009) Schweizerische Futtermitteldatenbank. URL: <http://www.agroscope.admin.ch/futtermitteldatenbank/index.html?lang=de> [Last access: 2011-08-13].
- [Agroscope 2011] Agroscope (2011) Agroscope : Schweizer Forschung für Landwirtschaft, Ernährung und Umwelt. URL: <http://www.agroscope.admin.ch/org/index.html?lang=de> [Last access: 2011-08-29].
- [Boessinger 2010] Boessinger, M. / Buchmann, M. / Python, P. (2010) Dürrfutterproduktion : von den Besten kann noch gelernt werden. In: Kreuzer, M. et al (Ed.) ETH-Schriftenreihe zur Tierernährung Vol. 33, p. 141-143.
- [Fox 2008] Fox, P. (2008) Creating a Store Locator with PHP, MySQL & Google Maps. URL: <http://code.google.com/intl/en/apis/maps/articles/phpsqlsearch.html> [Last access: 2011-08-13].
Still in Google Maps V2, but good tutorial for the first settings in the architecture as well as the proper embedding of the map, sidebar, marker and event listeners
- [Fox/Stucker 2007] Fox, P. / Stucker, L. (2007) Using PHP/MySQL with Google Maps. URL: <http://code.google.com/intl/en/apis/maps/articles/phpsqlajax.html> [Last access: 2011-08-13].
Still in Google Maps V2, but good tutorial for the use of XML in PHP and JavaScript
- [Fox/Manshreck 2007] Fox, P. / Manshreck, T. (2007) Geocoding addresses with PHP/MySQL. URL: <http://code.google.com/intl/en/apis/maps/articles/phpsqlgeocode.html> [Last access: 2011-08-16].

The PHP script part was used as guideline, even if the database system was not the corresponding one

- [Geostat 1994] Bundesamt für Statistik (1994) *GEOSTAT : die Servicestelle des Bundes für raumbezogene Daten.*
- [Google 2011a] Google (2011) Google Maps JavaScript API V3. URL: <http://code.google.com/intl/en/apis/maps/documentation/javascript/> [Last access: 2011-08-13]. *Collection of tutorial, examples and API library*
- [Google 2011b] Google (2011) The Google geocoding API. URL: <http://code.google.com/intl/en/apis/maps/documentation/geocoding/> [Last access: 2011-08-13]. *Short introduction into the geocoding service with Google Maps V3*
- [Google 2011c] Google (2011) Google chart tools. URL: <http://code.google.com/intl/en/apis/chart/interactive/docs/index.html> [Last access: 2011-08-22]. *Collection of tutorial, examples and API library*
- [Kemper 2006] Kemper, A. / Eickler, A. (2006) *Datenbanksysteme : eine Einführung.* 6. Auflage.
- [PgSQL 2010] The PostgreSQL Global Development Group (2010) PostgreSQL 9.0.4 Documentation. URL: <http://www.postgresql.org/docs/9.0/interactive/index.html> [2011-08-13].
- [Longley et al. 2011] Longley, P.A. et al. (2011) *Geographic Information Systems & science.*
- [Python 2010] Python, P. / Boessinger, M. / Buchmann, M. (2010) Teneur moyenne en minéraux majeurs des fourrages secs ventilés selon l'altitude et la situation géographique. In : *ETH-Schriftenreihe zur Tierernährung* Vol. 33, p. 159-162.
- [Refsnes Data 2011] Refsnes Data (2011) W3schools : the world's largest web development site. URL: <http://www.w3schools.com> [Last access: 2011-08-13].
- [SELFHTML 2007] SELFHTML e.V. (2007) *SELFHTML Dokumentation : Version 8.1.2.* URL: <http://de.selfhtml.org/> [Last access: 2011-08-13].