Temporal Model of the Feed Database

Samuele Zoppi

University of Zurich, Switzerland
samuele.zoppi@uzh.ch

1 Introduction

The Feed Database is designed to store aggregated measures of nutrients which compose different types of animal feed. Aggregated measures are derived in the following way. First, for a given type of animal feed a sample is collected from different places of Switzerland. For example, a sample can be grains from different fields or apples from the different gardens. Next, each sample is investigated in order to find out types and values of nutrients it contains. In the final stage, values of the same nutrients are aggregated and stored in the database. In addition to aggregated values, the database stores information about nutrients groups which are important for specific animal species.

The main focus of the Feed Database is to support analysis of nutrients for various feed types. Currently, the database does not contain historical information and, thus, only simple analysis is possible. For example, one can compare selected nutrients for a group of feeds, find the best combination of feeds which maximize selected nutrients, find the worst and best feeds for a given animals. However, content of nutrients and nutrient groups vary in time and, thus, the results of these analysis is valid only for some period. Moreover, the results are imprecise since the database stores aggregate value of many samples. Our major goal is to support the Feed Database with the full history of changes. In order to do that, the work have been split into three main tasks, which are described below.

2 Tasks

2.1 Deriving Requirements from Current Database Design

The first task is to investigate given SQL schema of the database and from that to derive current data model and entity-relationship diagram. All entities, attributes, keys, relationships, participation and cardinality constraints should be correctly recognize. Based on these models the actually data requirements should be deduced

2.2 Introducing Temporal Information

The second task is to extended ER diagram with temporal information so that the following data requirements are satisfied: instead of storing aggregated values of multiple measures, the database stores history of all single measures. In addition to nutrient’s type, single measures are categorized according to location they come from. Locations are cities, villages or regions in Switzerland.

2.3 Validating the Design

The last task is to validate new design of the database. First, the ER diagram should be translated into SQL schema and next, three SQL queries should be indentified in order that they: *i*) use time domain and *ii*) possibly are the most expensive to execute. The goal is to improve SQL schema with indexes and/or views which benefit most in optimizing given SQL queries. For one of the queries, the querying time will be experimentally evaluated with and without optimization.

3 Current Database Design

3.1 Data Requirements

To derive the data requirements for the Feed database the current SQL-schema was analyzed, as example the SQL-schema for two main tables of the feed database have been reported here below:

```
/*Table structure for table `tbl_components` */
/* !!!!!!!!!! in this table nutrient types are stored !!!!!!!!!!!!!!! */
DROP TABLE IF EXISTS `tbl_components`;
CREATE TABLE `tbl_components` (
 `ID_tbl_Components` int(10) unsigned NOT NULL auto_increment,
 `ref_ComponentGroups` int(10) unsigned NOT NULL,
 `C_Name_D` varchar(255) character set latin1 collate latin1_german1_ci default NULL,
 `C_Name_F` varchar(255) character set latin1 collate latin1_german1_ci default NULL,
 `C_Name_E` varchar(255) character set latin1 collate latin1_german1_ci default NULL,
 `C_Token_D` varchar(45) character set latin1 collate latin1_german1_ci default NULL,
 `C_Token_F` varchar(45) character set latin1 collate latin1_german1_ci default NULL,
 `C_Token_E` varchar(45) character set latin1 collate latin1_german1_ci default NULL,
 `C_std_decimalplace` int(1) default NULL,
 `C_PlausibilityCheckFormula` varchar(255) character set latin1 collate latin1_german1_ci
default NULL,
 `ref_terms` int(10) unsigned default NULL,
 `C_OutputOrder` int(11) default NULL,
 PRIMARY KEY  (`ID_tbl_Components`),
 KEY `tblComponents_FKIndex2` (`ref_ComponentGroups`)
) ENGINE=MyISAM AUTO_INCREMENT=252 DEFAULT CHARSET=latin1
COLLATE=latin1_german2_ci;

/*Table structure for table `tbl_feed` */
DROP TABLE IF EXISTS `tbl_feed`;
CREATE TABLE `tbl_feed` (
 `F_FeedSpecNr` int(10) unsigned NOT NULL,
 `ref_FeedGroup` int(10) unsigned NOT NULL default '0',
 `F_Name_D` varchar(255) character set latin1 collate latin1_german1_ci default NULL,
 `F_Name_F` varchar(255) character set latin1 collate latin1_german1_ci default NULL,
 `F_Name_E` varchar(255) character set latin1 collate latin1_german1_ci default NULL,
 `ref_terms` int(10) unsigned default NULL,
 PRIMARY KEY  (`F_FeedSpecNr`),
 KEY `tblFeed_FKIndex1` (`ref_FeedGroup`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COLLATE=latin1_german2_ci;
```

During the derivation of the data requirements and the relationship model from the SQL-schema we encountered few challenges. First of all in the SQL-schema all the foreign keys are not specified as foreign keys but just with the word “key” which is much more general and doesn’t strictly mean foreign key. This kind of generalization didn’t allow an easy derivation of all the relations and their cardinality. A second problem was that many tables had variables which actually aren’t part of the data requirements but are just needed for other technical purposes (e.g. the web visualization of the database). A last issue regards tables which are needed to normalize the database but can actually not be reflected into a specific entity-sets (e.g the tables “tbl_FeedAttributes” and “tbl_CompPofiles”).

The following statistics about the SQL-schema and the ER-diagram have been collected:

- Number of tables (SQL-schema): 16
- Number of entity-sets (ER-diagram): 12
- Number of attributes (SQL-schema): 98
- Number of attributes (ER-diagram): 41
- Number of relationships (ER-diagram): 11

The data requirement for the Feed DB have been deduced in the following entity-sets:

- Component: This entity-set describes the nutrients which can be found in the animal Feeds, each component (nutrient) can be found in one or more Feeds and a Feed can have one ore more different components. Each Component has a name which can be stored in different languages (English, German and French), can store many tokens, an output order, the standard decimal place, a plausibility check formula and is uniquely identified by an ID. A Component may belong to a ComponentsGroup and may belong to one or more Profiles.

- Feed: This entity-set describes the animal feeds, a Feed has a name which can be stored in different languages (English, German and French) and is uniquely identified by an ID. Each Feed may belong to a FeedGroup and may have one or more Attributes.
- FeedComponent: This entity-set describes the value of a particular nutrient (Component) in a certain Feed. In order to calculate the value of a nutrient many measurements are done, the value stored by a FeedComponent is the mean value of all the measurements done. Each FeedComponent is uniquely identified by an ID but can also be identified by the Feed and the specific Component which were consider for the measurements. Each FeedComponent have the measurement value and the rounded version of this value. Each FeedComponent specify a Unit and a QualityParameter.
- Species: This entity-set describes species of domestic animals. Each species has a name which can be stored in different languages (English, German and French) and is uniquely identified by an ID. Each Species may belong to one or more ComponentsGroups.
- ComponentsGroup: This entity-set describes the groups of Components, a ComponentsGroup contains one or more Components and may be associated to a particular Species. A ComponentGroup has a name which can be stored in different languages (English, German and French), an output order and is uniquely identified by an ID.
- Profile: This entity-set describes the profiles which every user can build under his restrictions, a Profile contains one or more Components, has a name which can be stored in different languages (English, German and French), a variable which say if the Profile is public or belong just to a certain user and is uniquely identified by an ID.
- FeedGroup: This entity-set describes the groups of feed, a FeedGroup contains one or more Feeds, has a name which can be stored in different languages (English, German and French), an output level, an output line and is uniquely identified by an ID. A FeedGroup may have a parent-FeedGroup, each FeedGroup may be the parent of one or many other FeedGropus.
- Attribute: This entity-set describes the feed attributes, an Attribute of a Feed can be for instance the location where the Feed was taken, the preparation method of the Feed, etc.. Each Attribute has a name

which can be stored in different languages (English, German and French) and is uniquely identified by an ID. An Attribute belong to one or more Feeds and may belong to an AttributeGroup.

- AttributeGroup: This entity-set describes the groups of attributes, an Attributegroup contains one or more Attributes, has a name which can be stored in different languages (English, German and French), a source type and has is uniquely identified by an ID.
- QualityParameter: This entity-set stores many parameters which describe how the aggregated measurement (FeedComponent) was done. Each QualityParameter is uniquely identified by an ID, stores the number of measurements, the standard deviation of these values, the value type, the modification date, a formula, the decimal place and the auto-remarks.
- Unit: This entity-set describe the measuring unit used to describe the measurement value. A Unit is uniquely identified by an ID, can store many tokens which actually compose the name of the Unit and has a value which is call kg-converter and serve to convert each Unit into kg. A Unit may belong to a UnitGroup.
- UnitGroup: This entity-set describe the groups of units, a UnitGroup contain one ore more Units, has a name which can be stored in different languages (English, German and French) and is uniquely identified by an ID.

Figure 1 and 2 report a graphical view (ER-models) of the data requirements already described.

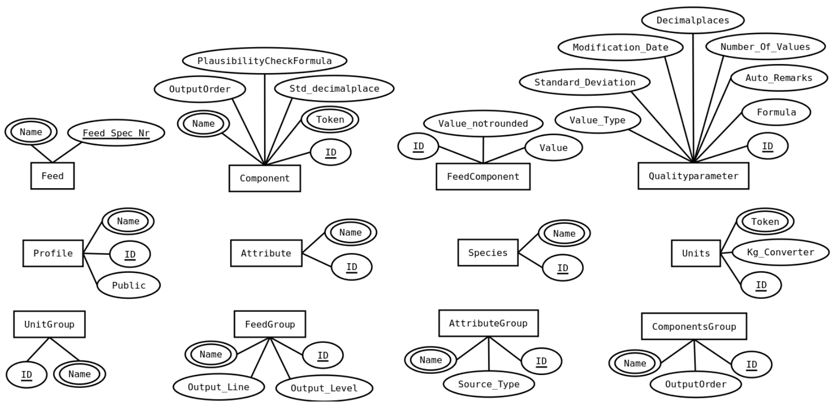


Fig. 1. Entity-sets described in the data requirements

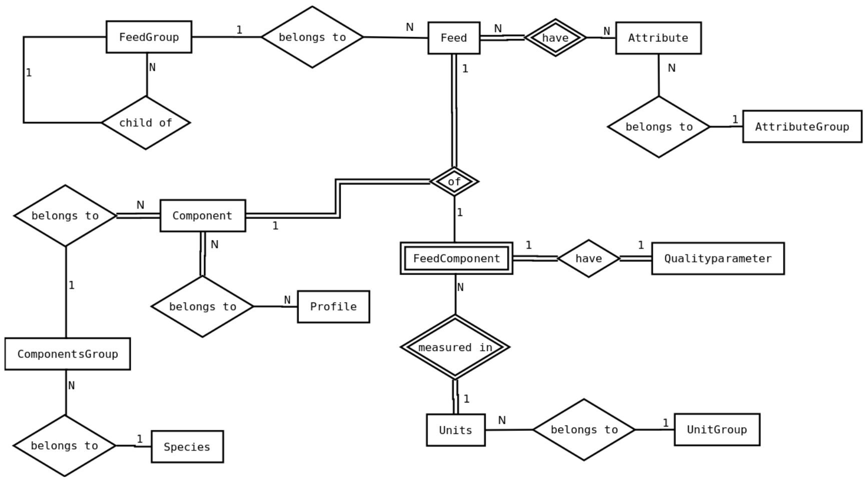


Fig. 2. Relations between the entities described in the data requirements

3.2 Data Input Tables

In order to input data into the database two tables have been implemented, figure 3 shows the ER-model for these tables.

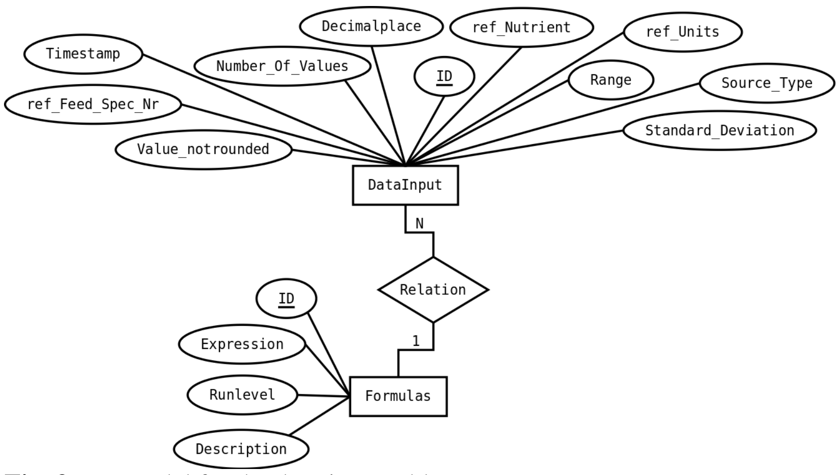
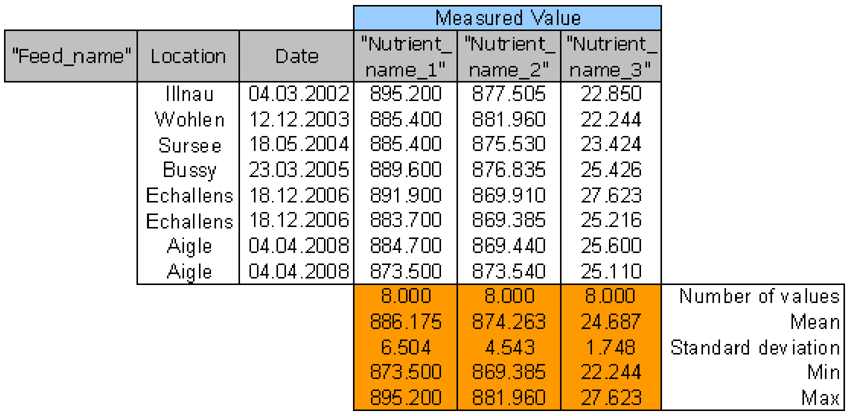


Fig. 3. ER-model for the data-input tables

4 Introducing Temporal Information

4.1 Measurements Data: Actual Data Format

At the moment in the Feed database the value of each component is stored as a mean value of many measures. In order to calculate the mean value, all the measurements about a single feed are stored into an excel file like in figure 4.



			Measured Value			
"Feed_name"	Location	Date	"Nutrient_ name_1"	"Nutrient_ name_2"	"Nutrient_ name_3"	
	Illnau	04.03.2002	895.200	877.505	22.850	
	Wohlen	12.12.2003	885.400	881.960	22.244	
	Sursee	18.05.2004	885.400	875.530	23.424	
	Bussy	23.03.2005	889.600	876.835	25.426	
	Echallens	18.12.2006	891.900	869.910	27.623	
	Echallens	18.12.2006	883.700	869.385	25.216	
	Aigle	04.04.2008	884.700	869.440	25.600	
	Aigle	04.04.2008	873.500	873.540	25.110	
			8.000	8.000	8.000	Number of values
			886.175	874.263	24.687	Mean
			6.504	4.543	1.748	Standard deviation
			873.500	869.385	22.244	Min
			895.200	881.960	27.623	Max

Fig. 4. Excel file for a single feed

In this excel file are reported the measures of three nutrients of a single feed. The feeds name, the location where each single sample come from, the date when each sample was picked up and the value for each component which were analyzed are stored in the excel file. If the Location and the Date are the same for two measured values, then it means that the sample is the same. This bring to the conclusion that for each sample more measurements may be done and stored in the excel file.

Nowadays just the mean value and other parameters marked in orange are stored in the database. More precisely the mean value for each component is stored in his “feedComponent” table, which is related to a “qualityParameter” table that stores the “Number of Values”, the “Standard Deviation” and other parameters about the measurements.

4.2 Measurements Data: Required Format

In the future, we want to be able to store all the single values reported in the excel file (figure 4). For each component of a certain feed one or more samples can be analyzed during the time. This fact should be reported also in the Feed Database storing the analysis done on each sample, so all the measures done on sample should be stored as single elements.

For instance the sample taken at "Echallens" on the "18.12.2006" (figure 4) should be saved as a single element which have two different measures for each component. In the next section the new data requirements which satisfy this new exigencies are described.

4.3 New Data requirements

- Sample: This entity-set describes a feed Sample of a particular Feed. Each Sample stores the Location of derivation and the Date when it was picked-up, an additional artificial ID is used to identify a Sample uniquely because the Date is unfortunately not always given. On each Sample one or more Measures can be done, but at least one should be given.
- Measure: This entity-set replaces the previous entity-set named "FeedComponent". A Measure store the Value of a certain Component (nutrient) in a particular Feed and should be associated to a single Sample. Each Measure also stores the Date when the measurement was done. A single pair of Feed and Component may have one or more Measures, which can belong to the same or to different Samples. A Measure is uniquely identified by an artificial ID because we don't have any variable which can do this.

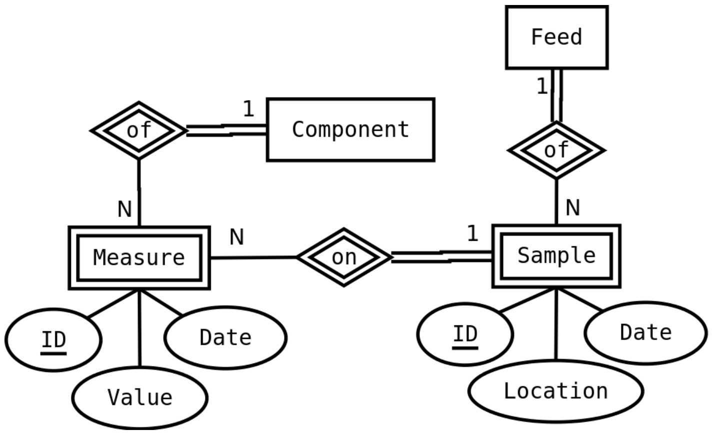


Fig. 5. The temporal ER-model for the new data requirements

With the new data requirements the entity-set FeedComponent has been replaced by the new entity-set named Measure. The entity-set Qualityparameter is no more needed in the new design, the variables stored in this entity-set should and can now be derived by the use of queries. Figure 6 presents the new ER-relations model.

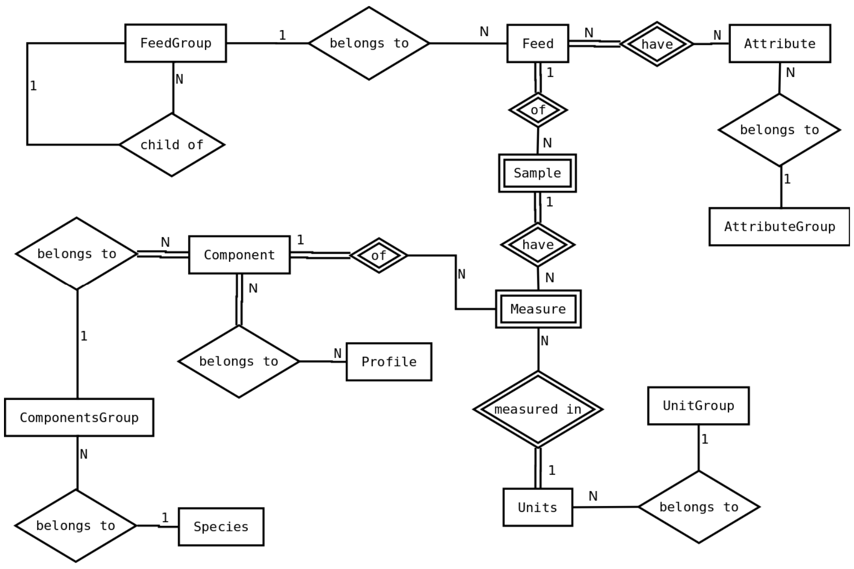


Fig. 6. Relations between the entities described by the new data requirements

5 Validating the Design

In this section the new design of the feed database is going to be tested, the database has been implemented as a real SQL-schema (see Appendix) and has been filled up with real and test data. In order to test the new design we chose to implement and evaluate three important types of queries which should be very important for the temporal model of the database.

5.1 Queries Description

In order to test the new design we are going to implement three different types of queries:

1. Thus the table Qualityparameter and FeedComponent don't exist anymore, we need to implement a query to compute the mean value

of all the measurements value for a specific Component in a given Feed in a given period of time (start date and end date should by entered). Another query compute the standard deviation of this measurements.
2. We want to implement a query which for a given component retrieves the feeds which contain the higher mean value of such a component. The query should also accept a period of time when the sample should have been picked-up.
3. In order to analyze the quality of the data in the database a query return the name of all the feeds which for a given component were not analyzed (no measure on this feeds were taken) between two given dates.

Of course this is a restricted choice of queries types, other queries could be implemented for example using the Location where a Sample was picked-up. The database and the three types of query have been implemented using PostgreSql.

5.2 Queries Implementation and Testing

The four queries described in the section above have been implemented this way:

Query 1:

```
/*Calculate the mean value for a given feed and a specific component.
The samples tacked in consideration are just the ones between the given dates.*/
SELECT avg(M_Value)
FROM Measure, Sample
WHERE Measure.ref_Component = 17 and Measure.ref_Sample = Sample.ID_sample and
Sample.ref_Feed= 1 and Sample.S_Date >= '2000-01-01' and Sample.S_Date <=
'2020-12-31';
```

Query 2:

```
/*Calculate the standard deviation for a given feed and a specific component.
The samples tacked in consideration are just the ones between the given dates.*/
SELECT stddev_pop(M_Value)
FROM Measure, Sample
WHERE Measure.ref_Component = 17 and Measure.ref_Sample = Sample.ID_sample and
Sample.ref_Feed= 1 and Sample.S_Date >= '2000-01-01' and Sample.S_Date <=
'2020-12-31';
```

Query 3:

```
/*Return the first 10 feeds which contain the higher value of a given component.
The results are sorted in descending order and the samples tacked in
consideration are just the ones between the given dates.*/
SELECT F_Name_E, avg(M_Value)
FROM Measure, Sample, Feed
WHERE Measure.ref_Component = 17 and Measure.ref_Sample = Sample.ID_sample and
Sample.ref_Feed = Feed.F_FeedSpecNr and Sample.S_Date >= '2000-01-01' and
Sample.S_Date <= '2020-12-31'
GROUP BY F_Name_E
ORDER BY avg(M_Value) DESC;
```

Query 4:

```
/*Return the name of all the feeds for that a given component is not been
measured between two given dates.*/
SELECT f.F_Name_E
FROM Feed f
WHERE f.F_FeedSpecNr != all
        (SELECT F_FeedSpecNr FROM Feed, Sample, Measure WHERE
        Measure.ref_component = 17 and Measure.ref_Sample = Sample.ID_sample and
        Sample.ref_Feed = f.F_FeedSpecNr and Measure.M_Date >= '2000-01-01' and
        Measure.M_Date <= '2020-01-02')
GROUP BY f.F_Name_E;
```

In order to test the new design the querying-time has been evaluated for each query. After a first test ran without any optimization a second test with several index optimization (described later in this section) was ran. The different querying-times can be found in the table below. The database was filled-up with test data like this way:

- 150 types of Components
- 500 types of Feeds
- 40 Sample per Feed (2 per year)
- 30 random Components per Sample
- 2-20 Measures for every Component

This data give us a database with 20000 Samples and about 6 millions of Measures (about 280 Mb of data) and should represent the state of the database in about 20 years.

Query number	Without optimization	With optimization
1 (two join)	1500 ms	2 ms
2 (two join)	1500 ms	2 ms
3 (three join + group by)	1700 ms	170 ms
4 (three join + group by)	> 5 min	60 ms

Table 1. Querying-time for the four queries described above

Like showed in the table above the querying-time without any optimization was for every query too long to be acceptable. In order to optimize the queries we took a look to the query evaluation method of all queries and we tried to build up some indexes that could help to improve the querying-time.

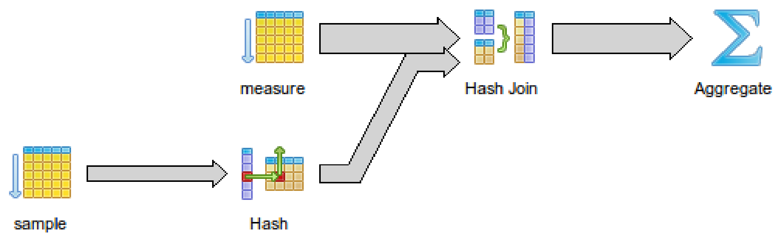


Fig. 7. Query evaluation method for queries 1 and 2 without optimization

After several tests we decided to build an index on the field “ref_Component“ which is a foreign-key of the Measure’s table, in addition we build-up a multiple index on the fields “ref_Component” and “ref_Sample” which are again foreign-keys of the Measure’s table, finally we build-up an index on the field “ref_Feed” which is a foreign-key of the Sample’s table.

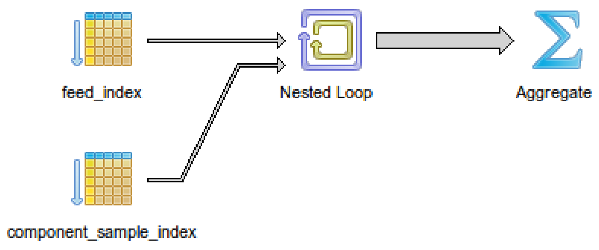


Fig. 8. Query evaluation method for queries 1 and 2 after the optimization

All the graphical representation of the query evaluation methods before and after the optimization can be found in the appendix.

The querying-times after this optimizations has become acceptable for every query, which it means that the temporal design of the database could probably be used to store a good amount of data.

6 Conclusions

After implementing and trying out the new design of the feed-database, we can for sure say that the suggested temporal model cover all the requirements described in the introduction section. The tests we ran also said that with the right optimizations the new design seams to be scalable and can be probably used with no major modifications. Since we don't spend so much time optimizing the database we also think that there is probably still place for even better improvement and so lower querying-times.

The next steps consist into implementing the temporal design on the actual database and begin to fill it with the right format of data. In order to do that the input tables should be adapted to the new format and all the needed queries should be described and implemented.

7 Appendix

```
CREATE DATABASE IF NOT EXISTS FeedDatabase;
USE FeedDatabase;

/*Table structure for table component */
/* !!!!!!!!!! in this table nutrient types are stored !!!!!!!!!!!!!!!! */
DROP TABLE IF EXISTS component CASCADE;
CREATE TABLE component (
  ID_Component serial NOT NULL,
  C_Name_D varchar(255) default NULL,
  C_Name_F varchar(255) default NULL,
  C_Name_E varchar(255) default NULL,
  C_Token_D varchar(45) default NULL,
  C_Token_F varchar(45) default NULL,
  C_Token_E varchar(45) default NULL,
  PRIMARY KEY  (ID_Component));

/*Table structure for table feed */
DROP TABLE IF EXISTS feed CASCADE;
CREATE TABLE feed (
  F_FeedSpecNr int NOT NULL,
  F_Name_D varchar(255) default NULL,
  F_Name_F varchar(255) default NULL,
  F_Name_E varchar(255) default NULL,
  PRIMARY KEY  (F_FeedSpecNr));

/*Table structure for table sample */
/*!!!!!!!!!! in this table feeds samples are stored  !!!!!!!!!!!*/
DROP TABLE IF EXISTS sample CASCADE;
CREATE TABLE sample (
  ID_sample serial NOT NULL,
  ref_Feed int NOT NULL,
  S_Location varchar(128) default NULL,
  S_Date date default NULL,
  PRIMARY KEY  (ID_sample),
  FOREIGN KEY (ref_Feed) REFERENCES feed(F_FeedSpecNr));

/*Table structure for table measure */
/*!!!!!!!!!! in this table single measures on a sample are stored  !!!!!!!!!!!*/
DROP TABLE IF EXISTS measure CASCADE;
CREATE TABLE measure (
  ID_measure serial NOT NULL,
  ref_Component int NOT NULL,
  ref_Sample int NOT NULL,
  M_Value float default NULL,
  M_Date date default NULL,
  PRIMARY KEY  (ID_measure),
  FOREIGN KEY (ref_Component) REFERENCES component(ID_Component),
       FOREIGN KEY (ref_Sample) REFERENCES sample(ID_Sample));
```

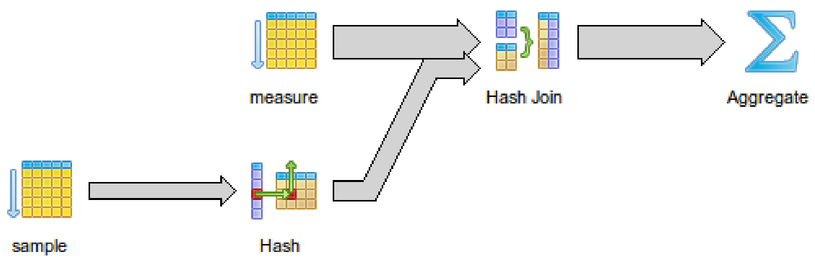


Fig. 9. Query evaluation method for queries 1 and 2 without optimization

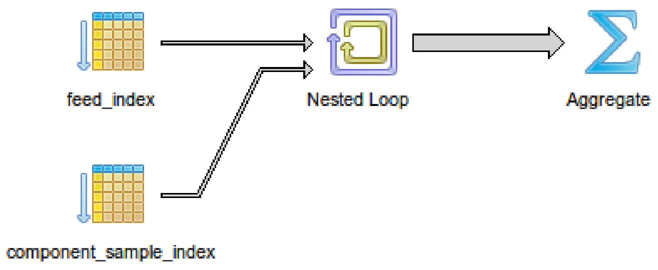


Fig. 10. Query evaluation method for queries 1 and 2 after the optimization

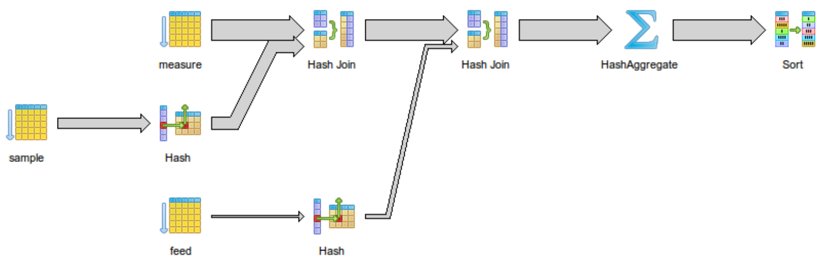


Fig. 11. Query evaluation method for query 3 without optimization

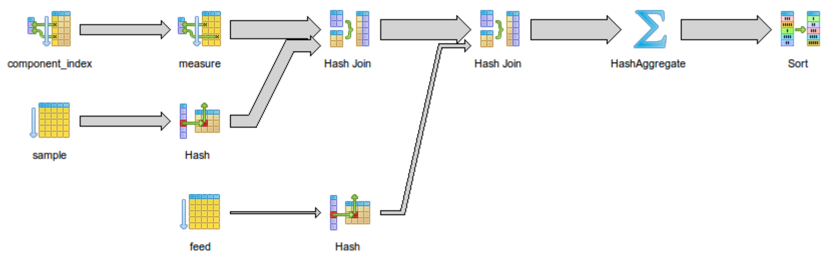


Fig. 12. Query evaluation method for query 3 after the optimization

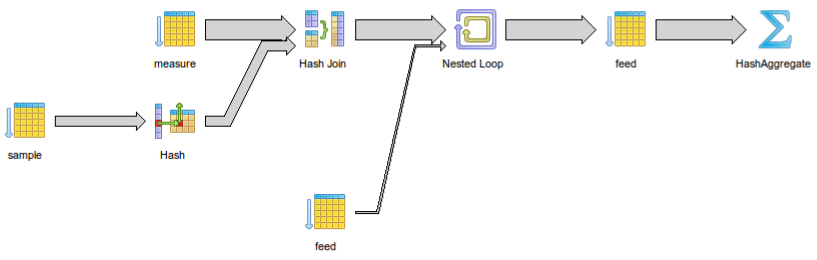


Fig. 13. Query evaluation method for query 4 without optimization

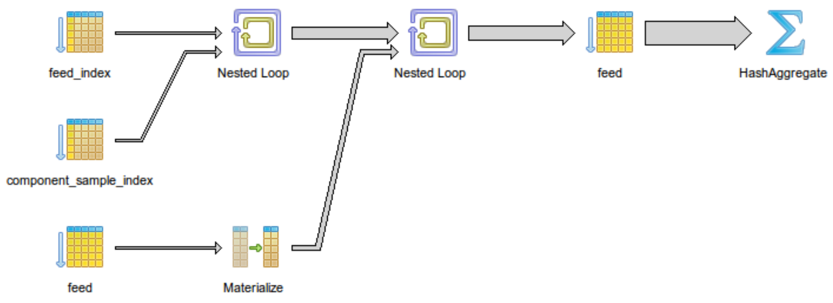


Fig. 14. Query evaluation method for query 4 after the optimization