Relational Scaling and Databases

To be published in the Springer-Verlag LNCS Series
http://www.springer.de/comp/lncs/index.html

Joachim Hereth

Darmstadt University of Technology, Department of Mathematics,
Schlossgartenstr. 7, D-64289 Darmstadt, Germany
hereth@mathematik.tu-darmstadt.de

Abstract More than 20 years of theoretical development and practical experience in the field of Conceptual Information Systems have made available a wide variety of structure and procedures to gain new knowledge from data or to present it in a user-friendly way, by restructuring the data in a conceptual way to help the user interpret and understand the meaning. Even longer, Database Theory has helped develop highly efficient database systems, processing daily huge amounts of data. However, both theories can profit from a cooperation: on the one hand, data and database modeling methodologies could be applied to the building of Conceptual Information System, the connection between the presented conceptual structures and the original data can be clarified. On the other hand, database theory may profit from the experience and ideas for more user-centered interfaces to the stored data, as well as profit from the translation of theoretical results.
In this paper, we present the first necessary steps to perform a translation between the languages used in both domains. For this purpose, we introduce basic notions from Database Theory with a focus on the operations, which are basic for a first application: a more formal way to describe the process of Relational Scaling [PW99] and the transformation of data for Conceptual Information Systems in general. Conversely, we present an approach for a standard problem of database theory by using methods from Formal Concept Analysis. Finally, we discuss the next steps needed for the integration of these two theories.

1 Introduction

Conceptual Information Systems are tools to help the user create new knowledge from the data he wants to explore and analyze. The computational power of database systems has grown considerably in the recent decades and will probably grow more in the future Therefore, we are today able to let those systems process huge amounts of data. Database Theory has helped to develop highly efficient database systems, performing many transactions and transformations on data. Databases of banks, or telephone and travel companies show the power of the techniques developed.

Nevertheless, the power to process those amounts of data gives not automatically the power to analyze it. This problem is, for instance, approached by Kimball in [Ki96], but, as others, he considers merely numerical results for his analysis. In [St00, HSWW00, HS01] different approaches based on the conceptualization of the data were presented. Following the principles of Conceptual

Knowledge Processing presented there, a data analysis system should activate the background knowledge of the user, help him to derive information based on the notion of concepts from daily use. It has already been mentioned that often interesting analysis results can only be obtained by clarifying and highlighting the relation between the calculated results and their meaning in the domain of the analyst (cf. [HSWW00]).

Several approaches have been presented to model the data in the form it is used in the conceptually accessible information systems: the application of multi-contexts for the modeling of databases has been used in [He00], the derivation of a power context family from a database in [PW99, EGSW00]. The former also introduced the idea of *Relational Scaling*, the transformation of a relational database into a conceptual information system. While the resulting systems show that this modeling is a fruitful approach, we still lack the language to formally describe the transformation from the data level, where form and implementation are highly influenced by technical and efficiency concerns, to the conceptual level of the information systems. A formal description of this connection will help us in the engineering of Conceptual Information Systems, as we are then able to adapt those systems more easily to changing data sources. Still more important, a bridge between those theories enables us to use concepts from one theory in the other, as will be shown later.

In this paper, we present a pragmatic approach by providing a direct translation from relational databases into the language of power context families. This enables us on the one hand to describe the process of relational scaling by already established notations, on the other hand we will investigate how some basic concepts from database theory translate into the language of Formal Concept Analysis. The latter will be fruitful for more technical work when a human expert has to actually build a Conceptual Information System using relational scaling and related methods. Then, relational scaling will be explained as composed of this direct translation and following transformations between power context families, called *intenstional enrichment*.

In Section 2 we introduce basic terminology from relational database theory and a more formal definition of the relational algebra and its operators as they are used in this domain. In the following section we briefly recall how concept graphs and power context families are used to model information. The sections 4 and 5 finally introduce the translation from the relational database into a power context family and the process of relational scaling using the language presented before. We conclude this paper with a discussion of further possibilities and research topics initiated by the integration approach presented.

2 The Relational Database Model

A database model provides the means for specifying the data structures and for operating on the data. One of the best studied and most often applied database models is the *relational model*. When E. F. Codd introduced it in [Co70] as a model for databases, this term refered to a specific data model with relations as

data structures and an algebra for specifying queries. With [Co72] he additionally introduced data integrity constraints – namely functional dependencies. Since then, ongoing research has produced languages and new operations based on this model, and also some variations on the algebra and calculus with varying expressive power. Similarly, a rich theory of constraints has been developed. Thus, the term relational model today refers to a whole class of database models, but they all have relations as basic data structures and provide at least some of the basic algebraic operations.

Basic Notations

In the following, we introduce the basic notions for data tables and the basic operations as used in database theory. Two traditions have been developed to describe data tables. The first is nearer to the mathematical notion of relations, considering a data table to be a set of tuples, which is called the *unnamed perspective*. The second way involves the notion of attributes of a data table, thus allowing the specification of columns by names instead of only numbers. This is called the *named perspective* and is used in most implemented relational databases management systems and also by the standard data modeling techniques. As pointed out in [AHV95], the differences are mainly syntactical, while the expressivity is the same. While more simple to read, the named perspective adds unnecessary complexity to formal treatment. For this reason, we will use the unnamed perspective in this paper.

When discussing the design of a relational database, one usually starts on an abstract level, sometimes called the *conceptual level* (see [MR91] for instance), using some kind of data modeling method. Starting from this conceptual level, an iterative process of normalization and decomposition takes place, to make the implemented database as efficient as possible. The resulting database schema may combine facts about different conceptual entities in one data table and separate facts about one entity into several tables. While this helps the efficiency of the database (usually the update efficiency), it makes it hard to understand the structure of the database without having information about the conceptual model.

In practice however, information about the conceptual model is usually not available. When applying data analysis techniques to already existing databases, one of the hardest parts is to re-engineer the conceptual model based on the informations about the implemented database. In more complex cases, this involves the consultation of domain experts to understand the relationship of the data in the database with the objects of the domain. Technically, this process is called *Relational Scaling*, as exemplified in [PW99]. Now, we introduce some basic notions from the theory of relational databases by example.

Example 1. According to [VG01], wine-growing has a long-standing tradition in Bulgaria. Based on archeological findings, one supposes that already more than 3000 years ago grapes have been cultivated. When the romans entered the province of Thrakia, they found a well established wine-growing culture.

Figure 1. The principal regions and districts of wine-growing in Bulgaria

Geographically, the country is divided in five principal regions: North, East, South, Southwest, and the South-Balkan region, as can be seen in Fig. 1. For our example, we sampled some information about these regions and the wine districts from [VG01].

Southbalkandistricts

District	Region
Sungurlare	South-Balkan
Rozova Dolina	South-Balkan

Southbalkangrapes

Grape	Region
Misket	South-Balkan
Gamza	South-Balkan
Riesling	South-Balkan
Rkatsiteli	South-Balkan
Cabernet Sauvignon	South-Balkan

Southdistricts

District	Region
Assenovgrad	South
Brestnik	South
Oriachowitz	South
Strandja	South
Sakar	South
Stambolovo	South

Southgrapes

Grape	Region
Mavrud	South
Cabernet Sauvignon	South
Merlot	South
Pamid	South
Misket red	South
Pinot Noir	South

Figure 2. Data tables showing the wine districts in the South and South-Balkan region and the grapes growing there

Fig. 2 shows some simple data tables: "Southdistricts" and "Southbalkandistricts" with the main wine districts in the South and South-Balkan region, and the tables "Southgrapes" and "Southbalkangrapes" with the grapes growing there. The descriptors of the table columns ("District", "Region", and "Grape") are called (*table*) *attributes*. They are used in the named perspective when treating the the data tables and are provided in our examples for convenience.

Each line of a table represents a *tuple*. The entries of the tuple are taken from a set of constants called *domain*, that may include, for example, the set of integers, strings, and Boolean values. Usually, we have some data tables for

reference in a database, which have a *name* – as "Southdistricts", "Southbalkandistricts" and so on in our example. However, not every data table has to have a name of its own (e. g. when creating new tables on the fly by some relational operations)

The following definition of a relational database can only be considered to be very basic (due to space limitations), e. g. we do not address the question of different value domains for different attributes, neither do we use the named perspective.

Definition 1. *Formally, we define a* (relational) database *to be a tuple* $\mathcal{D} := (\mathbf{dom}, N)$ *with* **dom** *being the* domain *of the database and* N *being the set of* named (data) tables *in the database.*

In general, a data table is any element $D \in \bigcup_{i\in\mathbb{N}_0} \mathfrak{P}(\mathbf{dom}^i)$. *The* arity *of* D *is the (smallest*[1]*)* $i \in \mathbb{N}_0$ *such that* $D \in \mathfrak{P}(\mathbf{dom}^i)$ *and is written* $\mathrm{arity}(D)$. *For a tuple* $t{\in}D$ *we write* $t[j]$ *with* $1{\leq}j{\leq}\mathrm{arity}(D)$ *to denote the* j*th value of the tuple.*

Example 2. The data tables shown in Fig. 2 belong to a database we formally define as $\mathcal{V} := (\mathbf{dom}, N)$, where **dom** includes (at least) the set of all geographical regions, wine growing districts and grapes of Bulgaria. The set N of named tables consists of the tables "Northgrapes", "Northdistricts", "Southgrapes", "Southdistricts", "Eastgrapes", "Eastdistricts", "Southbalkangrapes", "Southbalkandistricts", "Southwestgrapes", and "Southwestdistricts". They either contain pairs of a sort of grape and a region to indicate that this grape grows in this region (the "... grapes" tables), or a wine district and a region, if the district lies in this region (the "... districts" tables). As you can see by the description of the tables, they all have arity 2.

The Relational Algebra

Of course, a static representation of data is not sufficient for a database model, we also need the ability to operate on the data, to retrieve specific informations from the tables. For this purpose Codd introduced in [Co70] the first relational query language, a named algebra and showed it to be essentially equivalent to first-order predicate calculus in [Co72]. The algebra presented here is based on the unnamed version of the relational algebra in [AHV95], which is called the *SPCU*$^-$*-Algebra*. The elements of this algebra are – of course – the data tables, i. e. all elements of $\mathfrak{D}$. The operations of the algebra are defined as follows (we will use D, E to denote arbitrary data tables):

Definition 2. *There are two kinds of* selection operators. *The first is written in the form* $\sigma_{i=\mathrm{c}}$ *with* $i \in \mathbb{N}_0$ *and* $\mathrm{c} \in \mathbf{dom}$. *It removes from a relation all tuples that do not have the specified value* c *as* i*-th value.*

$$D^{\sigma_{i=\mathrm{c}}} := \begin{cases} \{t \in D \mid D[i] = \mathrm{c}\} \text{ if } i \leq \mathrm{arity}(D) \\ \emptyset \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}$$

[1] For every non-empty relation there is exactly one i. For the empty relation the arity is 0.

The second kind is of the form $\sigma_{i=j}$ with $i, j \in \mathbb{N}_0$. This form selects only those tuples, that have the same value on the i-th and the j-th place:

$$D^{\sigma_{i=j}} := \begin{cases} \{t \in D \mid D[i] = D[j]\} \text{ if } i, j \leq \text{arity}(D) \\ \emptyset \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}$$

Definition 3. *The* projection operators *reduce a relation by removing whole columns. For this, we select a set $\{x_1, x_2, \ldots, x_k\} \subset \mathbb{N}_0$, and retain only the columns specified by X. The arity of the resulting data table is $|X|$ if $X \subseteq \{1, 2, \ldots, \text{arity}(D)\}$.*

We suppose $x_1 \leq x_2 \leq \ldots \leq x_k$.

$$D^{\pi_X} := \begin{cases} \{(t[x_1], t[x_2], \ldots, t[x_k]) \mid t \in D\} \text{ if } x_k \leq \text{arity}(D) \\ \emptyset \qquad\qquad\qquad\qquad\qquad\quad \text{otherwise} \end{cases}$$

Definition 4. *The* cross-product *is the first operation of the algebra to operate on two data tables. The arity of the resulting data table is the sum of the arities of the two constituting tables.*

$$D \times E := \{(t[1], t[2], \ldots, t[\text{arity}(D)], s[1], s[2], \ldots, s[\text{arity}(E)]) \mid t \in D,\ s \in E\}$$

Definition 5. *The* union operator *merges data tables of the same arity.*

$$D \cup E := \begin{cases} \{(t \mid t \in D \text{ or } t \in E\} \text{ if } \text{arity}(D) = \text{arity}(E) \\ E \qquad\qquad\qquad\qquad \text{if } D = \emptyset \\ D \qquad\qquad\qquad\qquad \text{if } E = \emptyset \\ \emptyset \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}$$

Definition 6. *The* difference operator *is the set theoretic minus operation. If the arity of the data tables are different, the result is the first data table.*

$$D - E := \{t \in D \mid t \notin E\}$$

Functional Dependencies

Another important topic of database theory is the question how to avoid several kind of anomalies. "Anomaly" here means that changing something that is supposed to be a single bit of information implies the manipulation of many tuples. To avoid anomalies, a database may be transformed several times, each time reaching a new level of *normal form*. Those normal forms are defined in form of basic assumptions and statements about dependencies between tables and their attributes (or columns in the unnamed perspective). The first normal

form is so basic that we usually take it for granted: any entry in a data table (or more precisely in the tuples of a table) has to be atomic, not a set. This idea is basic for relational databases and was introduced by Codd in [Co70], the more elaborated higher normal forms were introduced in the following years by Codd himself and others. Those use notions of several kind of *dependencies*, the most important arguably being the *functional dependency*.

Definition 7. *Let D be a data table and $X, Y \subseteq \mathbb{N}_0$. Then, D fulfills the* functional dependency $D : X \to Y$ *(or short $X \to Y$ if the concerned data table is un-ambiguous), if for all tuples $s, t \in D$ $\pi_X(s) = \pi_X(t)$ implies that also $\pi_Y(s) = \pi_Y(t)$.*

There are some simple inference rules, let $X, Y, Z \subseteq \mathbb{N}_0$:

Reflexivity *If $X \subseteq Y$ then $Y \to X$*

Augmentation *If $X \to Y$, also $X \cup Z \to Y \cup Z$*

Transitivity *If $X \to Y$ and $Y \to Z$, we have $X \to Z$*

Example 3. Let's consider the table "Southbalkandistricts" from Fig. 2. There, we have only two rows, so for any pair of sets $X, Y \subseteq \mathbb{N}_0$ we have only to study a few conditions. As the table has only two columns, we are only interested in functional dependencies involving subsets of $\{1, 2\}$ (all functional dependencies involving other sets are a mere by-product from our extension of the partial to full operators, and bring no useful information about the data table). As you can easily verify, the only non-trivial functional dependency in this context is $\emptyset \to \{2\}$, all others follow by reflexivity or augmentation.

3 Conceptual Modeling with Formal Concept Analysis

Formal Concept Analysis has been developed around the formalization of the concept 'concept', providing a rich theory to describe and analyze hierarchies of concepts derived from a given context. We assume the reader to be familiar with the notions of formal contexts and concept lattices as defined in [GW99]. For the treatment of representation of more complex situations involving the relations between concepts, Sowa developed the theory of conceptual graphs [So84] which inspired the development of Contextual Judgment Logic (cf. [Pr98]). In conceptual graphs, boxes represent objects and information about their type, while ovals represent relations between the objects in the boxes connected to the oval.

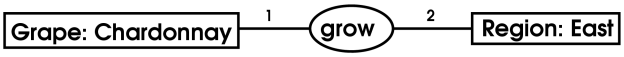

Figure 3. A simple conceptual graph

Example 4. Fig. 3 shows as an example a simple conceptual graph. The two boxes show information about two objects, in particular they indicate that the

object “Chardonnay” is of type “Grape” and the object “East” is of type “Region”. The oval in the middle indicates that these two objects are related, and this relation is called “grow”. As changing the location of nodes in a graph should not change its meaning, it is important to differentiate the meaning of the arcs going from a relation oval to the boxes. For this purpose the arcs are numbered from 1 up to n. The number of arcs going out from a relation is called the *arity* of the relation.

In [Wi97] Wille outlines an integration of the conceptual graphs and the constructions known from Formal Concept Analysis. In the following, Prediger, Wille and others developed for the Contextual Judgment Logic a mathematization of conceptual graphs (see for instance [PW99, Wi01]). The fundamental idea presented in [Wi97] is the transformation of Conceptual Graphs into a family of formal contexts, called a *power context family*.

Definition 8 (Power Context Family). *A* power context family $\vec{\mathbb{K}} := (\mathbb{K}_n)_{n\in\mathbb{N}_0}$ *is a family of formal contexts* $\mathbb{K}_k := (G_k, M_k, I_k)$ *such that* $G_k \subseteq (G_0)^k$ *for* $k = 1, 2, \ldots$. *The formal contexts* $\mathbb{K}_k$ *with* $k \geq 1$ *are called* relational contexts. *The power context family* $\vec{\mathbb{K}}$ *is said to be* limited of type $n \in \mathbb{N}_0$ *if* $\vec{\mathbb{K}} = (\mathbb{K}_0, \mathbb{K}_1, \ldots, \mathbb{K}_n)$, *otherwise, it is called* unlimited.

Example 5. Following the transformation described in [Wi97], the conceptual graph shown in Fig. 3 is transformed into the following power context family:

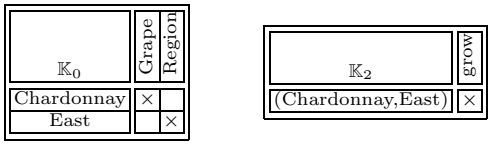

Figure 4. The power context family describing the simple conceptual graph above

Conversely, we can derive concept graphs from a given power context family, by using the concepts of the formal contexts $\mathbb{K}_k$ with $k \geq 1$ as the concepts for the relations, and write the objects in the tuples from the extent of this concept in adjacent boxes [PW99]. The concepts of $\mathbb{K}_0$ then correspond to the types, so we can write their names in a box, if the descriptor objects there are in the corresponding extent. It can easily be seen that the conceptual graph in Fig. 3 may be derived in this fashion from the power context family in Fig. 4. For a more precise description and a formal treatment of these transformation for simple concept graphs see [PW99].

4 From Relational Databases to Power Context Families

As noted in the last section, the concepts of the relational contexts of a power context family are considered to be the relations in the mathematization of

conceptual graphs. This leads naturally to the idea of treating the relations of relational databases as relational attributes, too. This *canonical database translation* of a database to a power context family is the first step in the process we call relational scaling:

Definition 9. *The power context family* $\vec{\mathbb{K}}(\mathcal{D})$ *resulting from the* canonical database translation *of the relational database* $\mathcal{D} = (\mathbf{dom}, N)$ *is constructed in the following way: we set* $\mathbb{K}_0 := (\mathbf{dom}, \emptyset, \emptyset)$ *and, for* $k \geq 1$*, let* G_k *be the set of all* k*-ary tuples and* $M_k \subseteq N$ *be the set of all named data tables of arity* k*. The relation* I_k *is defined by* $(g, m) \in I_k :\Leftrightarrow g \in m$*.*

Example 6.

$\mathbb{K}_2(\mathcal{V})$	NorthGrapes	NorthDistricts	SouthGrapes	SouthDistricts	EastGrapes	EastDistricts	SouthBalkanGrapes	SouthBalkanDistricts	SouthWestGrapes	SouthWestDistricts
(Gamza,North)	×									
(Cabernet Sauvignon,North)	×									
(Chardonnay,North)	×									
(Sauvignon Blanc,North)	×									
(Aligoté,North)	×									
(Dimiat,North)	×									
(Novo Selo,North)		×								
(Pavlikeni,North)		×								
(Svistkow,North)		×								
(Suhindol,North)		×								
(Lositza,North)		×								
(Liaskowets,North)		×								
(Roussenski Briag,North)		×								
(Mavrud,South)			×							
(Cabernet Sauvignon,South)			×							
(Merlot,South)			×							
(Pamid,South)			×							
(Misket red,South)			×							
...	...	...	...	...	...	...	...	...	...	...

Table 1. The top half of $\mathbb{K}_2$ of the power context family derived from $\mathcal{V}$

To illustrate the definition, we consider the power context family derived from our database example. As said in Example 2, all ten tables have arity 2. In Fig. 1 we see the top of the formal context $\mathbb{K}_2$ of the power context family derived from the relational database $\mathcal{V}$. Every tuple from the database belongs to exactly one attribute, giving the context a diagonal form. This is a typical effect, as it rarely happens that the same combination of values occurs in different relations.

The formal context $\mathbb{K}_0$ is not shown here. This context has no attributes – which is also clear from the definition, as there are no 0-ary relations in databases. The set of objects is huge. For our purposes, the exact set G_0 is not important. At least, it includes all values that appear in any tuple of the object set from

$\mathbb{K}_2$. Of course, depending from the implementation of the database, the set may be much larger, e. g. it may include all possible character strings.

As we have now a representation of the database using the notations from Formal Concept Analysis, we can easily apply methods from this theory. For example, since the very start of the development of Formal Concept Analysis, the investigation of dependencies between attributes has been of major interest. For details, we refer to [GW99], where the first algorithm is presented for the calculation of a base of implications in a given context, the so called *Duquenne-Guigues-Basis*. There are freely available programs to automatically calculate the base (e. g. ConImp from Peter Burmeister [Br01] or Concept Explorer from Sergey Yevtushenko). Based on similar results presented in [GW99], we have a simple procedure to determine a base of all functional dependencies of a data table.

Definition 10. *Let $\vec{\mathbb{K}}$ be a power context family, and let $m \in M_k$ be an attribute of the kth context. Than the* formal context of functional dependencies of m with regard to $\vec{\mathbb{K}}$ *is defined as* $\mathrm{FD}(m, \vec{\mathbb{K}}) := (m^{I_k} \times m^{I_k}, \{1, 2, \ldots, k\}, J)$ *with* $((g, h), i) \in J :\Leftrightarrow \pi_i(g) = \pi_i(h)$ *with* $g, h \in m^{I_k}$ *and* $i \in \{1, 2, \ldots, k\}$.

Analogously to the approach taken by [GW99, Proposition 28], we can now formulate the following proposition:

Proposition 1. *Let $\mathcal{D}$ be a relational database and m a k-ary table in $\mathcal{D}$. For two sets $X, Y \subseteq \{1, \ldots, k\}$ we have the following equality: The columns Y are functionally dependent from the columns X if and only if $X \to Y$ is an implication in* $\mathrm{FD}(m, \vec{K}(\mathcal{D}))$.

Example 7. Let us consider the small table "Southbalkandistricts" shown in Fig. 2. Then, the context $\mathrm{FD}(\texttt{SouthBalkanDistricts}, \mathcal{V})$ looks as follows:

	1	2
(Sungurlare,South-Balkan),(Sungurlare,South-Balkan))	×	×
(Sungurlare,South-Balkan),(Rozova Dolina,South-Balkan))		×
(Rozova Dolina,South-Balkan),(Sungurlare,South-Balkan))		×
(Rozova Dolina,South-Balkan),(Rozova Dolina,South-Balkan))	×	×

The only implication in this context is easy to find, it's $\emptyset \to \{2\}$, which corresponds indeed to the only functional dependency found in example 3. The important point is that Formal Concept Analysis provides a rich set of methods to treat implications, and that those methods can directly be applied to functional dependencies. Of course, the two-column tables from our example do not provide much complexity, but nevertheless the principal procedure should become clear.[2]

[2] Actually, there is still much room for efficiency improvements, but this is beyond the scope of this paper.

5 Relational Scaling

Conceptual Information Systems connect theory and practice. The data to be analyzed and presented comes from the real world and is usually stored in some kind of (relational) database. The data structures and the algorithms are images of the theory. The transition between these two domains consists in building up the connection between the data and the application used to build the Conceptual Information System, e. g. by coding SQL-Queries into the data files of the TOSCANA-Systems. After this, the system could be applied to the database. Even if the underlying data, the tuples, were changing, the system needed no further adaption, as all definitions in the system depend solely on the not changing properties.

However, in another way those systems were not apt to change. If the intensional side, the schema of the database changed, the system often was difficult to adapt – if not the original author of the system was available, the way how the data had to be transformed became quickly obscure. Using the methods introduced in this paper, we can formally describe this procedure, make transparent where external information is introduced, and clarify where corrections have to be made.

The canonical database translation introduced in the last section helps us, to use mainly the language of Formal Concept Analysis to describe all the necessary transformations.

Example 8. Let $\vec{\mathbb{K}} := \vec{\mathbb{K}}(\mathcal{V})$. For a start, we will perform the construction of a (very small) TOSCANA system.[3] This means, we have to derive a formal context $\mathbb{L}$ from the given power context family.

A TOSCANA system is used to analyze objects of the same type. In our database, we have several homogeneous subsets of objects, for now we select the grapes. To define new sets and attributes, we have to consider, that in Formal Concept Analysis concepts are not relations in the usual sense, but their extents are. Therefore, we can use the relational operators on the extents of concepts. As long as the meaning is clear, we will omit the extent operator and write e. g. $m := n \times o$ which means that m is a new attribute whose extent is defined to be the cross-products of the extents from concepts specified by n and o.

Thus, we define for $\mathbb{L}$ as object set $G := \texttt{Northgrapes}^{\pi\{1\}} \cup \texttt{Southgrapes}^{\pi\{1\}} \cup \ldots \texttt{Southwestgrapes}^{\pi\{1\}}$.

For the attributes we can define for every region an attribute "grows in ...", e. g. for the South-West region by $\texttt{grows\_in\_SouhWest} := \texttt{Southwestgrapes}^{\pi\{1\}}$. All those definitions yield the attribute set M of $\mathbb{L}$.

Subsets of M may be grouped together as *scales*. In case of TOSCANA-like systems we also provide line diagrams for the concept lattices of the scales.

With the introduction of *relational scaling* in [PW99], a new level of complexity has been added to this process. We do not only want to build up one

[3] We select TOSCANA here as it is the best known. We could have written the name of any existing program that uses basically a formal context as the basic structure of investigation.

single formal context, but a complete power context family. Not only attributes of objects, but also relations between objects have to be defined. Of course, this can be done too using intensional enrichment.

Example 9. We want to extend the system from the last example. Therefore, we not only derive grapes as object set, but the wine districts and the regions too. Additionally, we want to keep information about the relations between the objects.

The goal is to construct a system using concept graphs as shown in Fig. 3 as user interface to enable the user to ask queries like "which districts lie in the southern region" without restricting the set for responses to only one object set.[4] Furthermore, the system could allow the user to construct new relations based on the existing ones. This way, the user can adapt the system more easily to his own conceptualizations.

In our new system, the object set of $\mathbb{L}_0$ is larger than it was in the last example. Now, it is the complete set **dom** (or at least the set of all entries that are present in any tuple in any table) Additionally, the meaning of attributes in $\mathbb{L}_0$ should now be seen as that of "type" in the technical sense of the concept graph example above.

Formally, the definition of attributes of the higher level attributes from $\mathbb{L}_i$ with $i \geq 1$ is basically the same as in the last example. To avoid confusion between attributes of $\mathbb{L}_0$ and $\mathbb{L}_1$ as well as to enhance the readability, we now note the arity of the attribute to by writing e. g. $(2, \texttt{grows\_in}) := \texttt{Northgrapes} \cup \texttt{Southgrapes} \cup \ldots \cup \texttt{Southwestgrapes}$ to underline that the attribute `grows_in` is of arity 2.

So far, we only transformed already present information to get a more natural model. Often, when constructing a Conceptual Information System, new informations are added that were implicitly in the minds of the domain experts and users before. Here this could be information about the color and origin of the grapes or informations about the wine districts. In our language, those information can be represented as a new relation that is added as a whole from some external data source.

These were simplistic descriptions of the engineering process when building Conceptual Informaton Systems. Using the presented notation it can be clarified how the conceptual scales of the information system are constructed by defining new attributes and how exactly they depend from the underlying data source. Additionally, we can see where external knowledge gets introduced into the system, e. g. by using a specific classification of values for building a conceptual scale. Those transformations enrich the meaning of the representation, why we call them *intensional enrichment*.

[4] The use of conceptual graphs as a query language has already been proposed by Sowa in [So84].

6 Further Research

This article presents only first steps towards an integration of Database Theory and Formal Concept Analysis. So far, we have shown some basic notational devices to describe the process of relational scaling as a two-step procedure consisting of the canonical database translation, which transforms the database into a power context family, and the intenstional enrichment, which transforms the resulting power context family into another, which is closer to the conceptual model.

Some of the topics to be approached next can certainly bring some enhancements for our work on conceptual structures, for example the use of the named perspective for the presentation and construction of concept graphs. This would ease the communication using those graphs as well as enhance their expressiveness for their application in the realm of data modeling.

A topic that has been intensively covered in database theory and has not at all been addressed by this paper is the question of *domain independence*. This relates to the question if a query alway yields the same result independent from the actual content and schema of the database. Translated to the area of Conceptual Information System we have of course the same problem: Is the power context family that is the result of intensional enrichment (whose definitions are independent from the actual content) different if we choose a different set **dom**? Do the answers to user queries change if G_0 is enlarged?

When we studied the transition from one power context family to another by intensional enrichment, we also noted that sometimes new knowledge may be introduced. This naturally increases the amount of information stored in the power context family. However, currently we lack a notion of equivalence for two power context families that may be derived one from the other without adding new knowledge. This may be done by extending the notion of *conceptual content* in [Wi00]. However, for this we have to be able to distinguish the cases when new information is introduced and when not.

Furthermore, while the relational database model is well studied and very successful applied in the real world, the concept of the tuple as basic information unit seems too restrictive for many in the field of data modeling. For this reason, an investigation of forthcoming data models from semantic and object-oriented databases [HK87, ABD+89] may be helpful.

References

[AHV95] S. Abiteboul, R. Hull, V. Vianu: *Foundations of databases.* Addison-Wesley, Reading – Menlo – New York 1995.

[ABD+89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, S. Zdoink: The object-oriented database system manifesto. In: *Proc. of Intl. Conf. on Deductive and Object-Oriented Databases (DOOD)*, pages 40–57, 1989.

[Br01] P. Burmeister: *Formal Concept Analysis with ConImp: Introduction to the Basic Features.* `http://www.mathematik.tu-darmstadt.de/ags/ag1/Software/DOS-Programme/` (a shortened german version has been published in [SW00] as *ConImp - Ein Programm zur Formalen Begriffsanalyse*)

[Co70] E. F. Codd: A relational model of data for large shared data banks. *Comm. of the ACM*, 13(6):377-387, 1970.

[Co72] E. F. Codd: Relational completeness of database sublanguages. In R. Rustlin (ed.): *Courant Computer Science Symposium 6: Data Base Systems*, pages 65–98, Prentice-Hall, Englewood Cliffs, NJ, 1972.

[DS01] H. S. Delugach, G. Stumme (eds.): *Conceptual Structures: Broadening the Base.* LNAI 2120. Springer, Berlin – Heidelberg – New York 2001.

[EGSW00] P. Eklund, B. Groh, G. Stumme, R. Wille: A Contextual-Logic Extension of TOSCANA. In: [GM00], 453–467.

[GM00] B. Ganter, G. W. Mineau (eds.): *Conceptual Structures: Logical, Linguistic, and Computational Issues.* LNAI 1867. Springer, Berlin – Heildelberg – New York 2000.

[GW99] B. Ganter, R. Wille: *Formal Concept Analysis : Mathematical Foundations.* Springer Verlag, Berlin – Heidelberg – New York, 1999.

[He00] Joachim Hereth: *Formale Begriffsanalyse und Data Warehousing.* Diplomarbeit, TU Darmstadt 2000.

[HS01] J. Hereth, G. Stumme: Reverse Pivoting in Conceptual Information Systems. In: [DS01], 202–215.

[HSWW00] J. Hereth, G. Stumme, U. Wille, R. Wille: Conceptual Knowledge Discovery and Data Analysis. In: [GM00], 421–437.

[HK87] R. Hull, R. King: Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19:201–260, 1987.

[Ki96] Ralph Kimball: *The Datawarehouse Toolkit.* John Wiley and Sons, New York 1996.

[MR91] H. Mannila, K.-J- Räiha. *The Design of Relational Databases.* Addison-Wesley, Reading – Menlo Park – New York, 1991.

[Pr98] S. Prediger: *Kontextuelle Urteilslogik mit Begriffsgraphen. Ein Beitrag zur Restrukturierung der mathematischen Logik.* Dissertation, TU Darmstadt. Shaker, Aachen 1998.

[PW99] S. Prediger, R. Wille: The Lattice of Concept Graphs of a Relationally Scaled Context. In: [TC99], 401–414.

[So84] J. F. Sowa: *Conceptual Structures: Information processing in mind and machine.* Addison-Wesley, Reading 1984.

[St00] Gerd Stumme: Conceptual On-Line Analytical Processing. In: K. Tanaka, S. Ghandeharizadeh, Y Kambayashi (eds.): *Information Organization and Databases.* Chpt. 14. Kluwer, Boston – Dordrecht – London 2000, 191–203.

[SW00] G. Stumme, R. Wille (eds.): *Begriffliche Wissensverarbeitung: Methoden und Anwendungen.* Springer, Berlin – Heidelberg – New York 2000.

[TC99] W. Tepfenhart, W. Cyre (eds.): *Conceptual Structures: Standards and Practices.* LNAI1640, Springer, Berlin – Heidelberg – New York 1999.

[VG01] Holger Vornholt, Joachim Grau (eds.): *Wein Enzyklopädie*, Scrito Medien, Frankfurt am Main – Belgium 2001.

[Wi97] R. Wille: Conceptual Graphs and Formal Concept Analysis. In: D. Lukose, H. Delugach, M. Keeler, L. Searle, J. F. Sowa (eds.): *Conceptual Structures: Fulfilling Peirce's dream.* LNAI 1257, Springer, Berlin – Heidelberg – New York 1997, 317–331.

[Wi00] R. Wille: Contextual Logic summary. In: G. Stumme (ed.): *Working with Conceptual Structures. Contributions to ICCS 2000.* Shaker, Aachen 2000, 265–276.

[Wi01] R. Wille: *Boolean Judgment Logic.* In [DS01], 115–128.