STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

A METHOD FOR MINING FUNCTIONAL DEPENDENCIES IN RELATIONAL DATABASE DESIGN USING FCA

KATALIN TUNDE JANOSI RANCZ AND VIORICA VARGA

Abstract. Formal Concept Analysis (FCA) is a useful tool to explore the conceptual knowledge contained in a database by analyzing the formal conceptual structure of the data. In this paper, we present a new method to optimize and extend a previous research on FCA and databases, by analyzing the functional dependencies in order to correctly build database schemata. Our method intends to mine functional dependencies in a relational database table. The novelty of our method is that it builds inverted index files in order to optimize the construction of the formal context of functional dependencies.

Key words: Formal concept analysis, Functional dependencies, Relational databases.

1. Introduction

From a philosophical point of view a concept is a unit of thoughts consisting of two parts, the extension, which are objects and the intension consisting of all attributes valid for the objects of the context. Formal Concept Analysis (FCA) introduced by [7] gives a mathematical formalization of the concept notion. A detailed mathematic foundation of FCA can be found in [3]. Formal Concept Analysis is applied in many different realms like psychology, sociology, computer science, biology, medicine and linguistics.

Formal Concept Analysis has been proved to be a valuable tool to represent the knowledge contained in a database, for instance logical implications in datasets. The subject of detecting functional dependencies in relational tables was studied in detail with different mathematical theories. Hereth (2002) presents the relationship between FCA and functional dependencies. He introduces the formal context of functional dependencies. In this context, implications hold for functional dependencies. Baixeries (2004) gives an interesting framework to mine functional dependencies using Formal Context Analysis. Detecting functional dependencies seems to be an actual theme, see [8].

Received by the editors: April 8, 2008.

2000 *Mathematics Subject Classification.* 03G10.

1998 *CR Categories and Descriptors.* H.2.4 [**Database Management**]: Systems – *Relational databases*.

This paper presents how some basic concepts from database theory translate into the language of Formal Concept Analysis and attempts to develop the functional dependencies.

We optimize an existing method introduced by [4], which provides a direct translation from relational databases into the language of power context families and calculates the functional dependencies in a relational table. In order to generate the tuple pairs necessary to build the formal context of functional dependencies we build inverted index files for the values of every attribute, because in the context of functional dependencies we need the tuple pairs, where at least one of the attribute values are common. In order to make the algorithm faster we reduce the size of the context file, which leads to fewer concepts in the lattice with useless dependencies.

2. Functional dependencies

The most used database model is the relational model. To give the structure of data [1] presents the unnamed and named perspective. The data is stored in data tables, which have attributes as columns and tuples as rows. In the named perspective the attributes are given by names, in the unnamed perspective they are given only by position. The operations on the relational model are based on algebra or on logic. The data integrity constraints of the model appear as functional dependencies.

We give the definition of a relational database from the unnamed perspective.

Definition 1. We define a relational database to be a tuple $\mathcal{D} := (\mathbf{dom}, \mathcal{N})$ with **dom** being the domain of the database and $\mathcal{N}$ being the set of named data tables in the database. A data table is any element $T \in \cup_{i \in \mathbb{N}_0} \mathcal{P}\left(\mathbf{dom}^i\right)$. The arity of T is $i \in \mathbb{N}_0$ such that $T \in \mathcal{P}\left(\mathbf{dom}^i\right)$ and is written $arity(T)$. For a tuple $t \in T$ we write $t\,[j]$ with $1 \leq j \leq arity\,(T)$ to denote the j-th value of the tuple.

Example 1. In database implementations the named perspective is used. The database scheme is composed of the table names with their attribute names. The example gives the relational scheme of a university database. Students are divided in groups; there can be many groups in one specialization. Students are marked at different disciplines.

```
Specialization [SpecID, SpecName, Language]
Groups [GroupID, SpecID]
Students [StudID, GroupID, StudName, Email]
Disciplines [DiscID, DName, CreditNr]
Marks [StudID, DiscID, Mark]
```

In this example $\mathcal{N}$ is composed of the named data tables: Specialization, Groups, Students, Disciplines, Marks, **dom** is the set of all attribute's values

of the tables. In case of each table the *arity* is the number of its attributes, so Groups table has arity 2, Students has 4. Let t be a tuple (row) of table Students, than $t[1]$ is the value of tuple t for StudID, $t[2]$ is the value of GroupID in the corresponding row, so on.

In relational database design the normalization theory is used to avoid redundancy. Normal forms use the notion of functional dependencies. The following definition uses the projection relational operator, see any database theory book [1], [6]. In the following definition we use the unnamed perspective, so the attributes are given by there number.

Definition 2. Let T be a data table and $X, Y \subseteq \mathbb{N}_0$. Then, T fulfills the functional dependency $D : X \to Y$, if for all tuples $s, t \in T$ $\pi_X(s) = \pi_X(t)$ implies that also $\pi_Y(s) = \pi_Y(t)$.

Example 2. Let be the next relational table:

`StudentInfos [StudID, StudName, Email, GroupID, SpecName]`

For all tuples $s, t \in$ `StudentInfos` for which $\pi_{GroupID}(s) = \pi_{GroupID}(t)$ implies that $\pi_{SpecName}(s) = \pi_{SpecName}(t)$.

So, the following functional dependencies hold:

$$GroupID \to SpecName$$

This means, we repeat for every student from a group the specialization name.

In the same manner we can see that:

$$StudID \to StudName, Email, GroupID, SpecName$$
$$Email \to StudID, StudName, SpecName, GroupID$$

In order to define the functional dependencies in a formal context, we need the notion of Power Context Family.

Definition 3. (Power Context Family). A power context family $\overrightarrow{\mathbb{K}} := (\mathbb{K}_n)_{n \in \mathbb{K}_0}$ is a family of formal contexts $\mathbb{K}_k := (G_k, M_k, I_k)$ such that $G_k \subseteq (G_0)^k$ for $k = 1, 2, ...$ The formal contexts $\mathbb{K}_k$ with $k \geq 1$ are called relational contexts. The power context family $\overrightarrow{\mathbb{K}}$ is said to be limited of type $n \in \mathbb{N}_0$ if $\overrightarrow{\mathbb{K}} = (\mathbb{K}_0, \mathbb{K}_1, ..., \mathbb{K}_n)$, otherwise, it is called unlimited.

The following definition from [4] gives the method to construct the power context family resulting from a relational database.

Definition 4. The power context family $\overrightarrow{\mathbb{K}}(\mathcal{D})$ 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 $(g, m) \in I_k :\Leftrightarrow g \in m$.

$\mathbb{K}_2$	`Groups`
(531, I)	X
(532, I)	X
(111, M)	X
(...)	...

TABLE 1. $\mathbb{K}_2$ for **Example** 1

$\mathbb{K}_3$	`Specialization`	`Disciplines`	`Marks`
(M, Mathematics, German)	X		
(I, Informatics, English)	X		
(...)	...		
(11, Databases, 6)		X	
(22, Algebra, 6)		X	
(...)		...	
(101, 22, 9)			X
(157, 22, 8)			X
(...)			...

TABLE 2. $\mathbb{K}_3$ for **Example** 1

Example 3. Let us construct the power context family of our Example 1. $\mathbb{K}_0$ has no attributes, because we haven't any 0-ary relation. $\mathbb{K}_2$ has in his one attribute table `Groups`, this is the only table with arity 2, objects are tuples from this table, see Table 1. The attributes of $\mathbb{K}_3$ are the tables with arity 3, objects are rows from these tables, the incidence relation shows which tuple to which table belongs (Table 2). $\mathbb{K}_4$ has one attribute, `Students` table name, see Table 3.

The formal context of functional dependencies is defined in the following way in [4].

Definition 5. Let $\overrightarrow{\mathbb{K}}$ be a power context family, and let $m \in M_k$ be an attribute of the k-th context. Than the formal context of functional dependencies of m with regard to $\overrightarrow{\mathbb{K}}$ is defined as $FD\left(m, \overrightarrow{\mathbb{K}}\right) := \left(m^{I_k} \times m^{I_k}, \{\ 1, 2, ..., k\}, J\right)$ with $\left((g,h), i\right) \in J :\Leftrightarrow \pi_i(g) = \pi_i(h)$ with $g, h \in m^{I_k}$ and $i \in \{\ 1, 2, ..., k\}$.

Example 4. Let us construct the formal context of functional dependencies for table `StudentInfos` of Example 2 notated by $FD\left(StudentInfos, \overrightarrow{\mathbb{K}}(Uni)\right)$. This is a relation from database with name Uni. It has arity 5, so it is an attribute of the $\mathbb{K}_5$ from the corresponding power context family $\overrightarrow{\mathbb{K}}(Uni)$.

$\mathbb{K}_4$	`Students`
(101, 531, Irene Cates, IreneCates@email.com)	X
(157, 111, Maria Jillian,MariaJillian@yahoo.com,)	X
(234, 532, Frank Orlando,FrankOrlando@email.com)	X
(...)	...

TABLE 3. $\mathbb{K}_4$ for **Example** 1

The attributes of $FD\left(StudentInfos, \overrightarrow{\mathbb{K}}(Uni)\right)$ are the attributes of table `StudentInfos`, the objects are pairs of tuples from this table. The incidence relation of the context shows that the attribute is common to the tuple pair from the row, see Fig. 1. We give short names to students and emails, in order to fit the picture in page.

	StudID	StudName	GroupID	SpecName	Email
(1,'a',531,'Info','aa')(2,'b',531,'Info','bb')			×	×	
(1,'a',531,'Info','aa')(3,'c',531,'Info','cc')			×	×	
(1,'a',531,'nfo','aa')(4,'d',532,'Info','dd')				×	
(1,'a',531,'Info','aa')(5,'e',631,'Mathe','ee')					
(1,'a',531,'Info','aa')(6,'f',631,'Mathe','ff')					
(2,'b',531,'Info,'bb')(3,'c',531,'Info','cc')			×	×	
(2,'b',531,'Info','bb')(4,'d',532,'Info','dd')				×	
(2,'b',531,'Info','bb')(5,'e',631,'Mathe','ee')					
(2,'b',531,'Info','bb')(6,'f',631,'Mathe','ff')					
(3,'c',531,'Info','cc')(4,'d',532,'Info','dd')				×	
(3,'c',531,'Info','cc')(5,'e',631,'Mathe','ee')					
(3,'c',531,'Info','cc')(6,'f',631,'Mathe','ff')					
(4,'d',532,'Info','dd')(5,'e',631,'Mathe','ee')					
(4,'d',532,'Info','dd')(6,'f',631,'Mathe','ff')					
(5,'e',631,'Mathe','ee')(6,'f',631,'Mathe','ff')			×	×	
(1,'a',531,'Info','aa')(7,'a',631,'Mathe','a2')		×			
(2,'b',531,'Info','bb')(7,'a',631,'Mathe','a2')					
(3,'c',531,'Info','cc')(7,'a',631,'Mathe','a2')					
(4,'d',532,'Info','dd')(7,'a',631,'Mathe','a2')					
(5,'e',631,'Mathe','ee')(7,'a',631,'Mathe',...					
(6,'f',631,'Mathe','ff')(7,'a',631,'Mathe','a2')			×	×	

FIGURE 1. $FD\left(StudentInfos, \overrightarrow{\mathbb{K}}(Uni)\right)$

In order to mine functional dependencies in the context defined in definition 5, we need the following proposition (see [4]).

Proposition 1. Let $\mathcal{D}$ be a relational database and m a k-ary table in $\mathcal{D}$. For two sets $X, Y \subseteq \{ 1, ..., k\}$ 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 $FD\left(m, \overrightarrow{K}(\mathcal{D})\right)$.

Example 5. Let us construct the concept lattice for

$$FD\left(StudentInfos, \overrightarrow{\mathbb{K}}(Uni)\right)$$

using the Concept Explorer (ConExp) from site http://sourceforge.net, see Fig. 2. We can see the implication in the context of functional dependencies, the software shows us too:

$$GroupID \to SpecName$$

This is a transitive functional dependency, so the table isn't in 3NF. The software also find the following functional dependencies:

$$StudID \to StudName, Email, GroupID, SpecName$$

$$Email \to StudID, StudName, SpecName, GroupID$$

where the right hand side ($StudID$ and $Email$) are candidate keys of the table `StudInfos`.

3. Method description

This section presents how our method constructs the context of functional dependencies of a database table.

In the first step, we introduce the structure of the table to be designed and some significant tuples. It is not necessary to use all the rows of a database table, but it is important to select varied tuples, with different styles of data, in order to get as many conclusions as possible. Using definitions 3, 4, 5 we construct the formal context of functional dependencies to find existing functional dependencies as implications in the constructed table. In order to optimize the construction of the formal context, we build inverted index files for the values of every attribute. With inverted indexes the number of rows in the data file of formal context resulted by our method is half of the same value resulted using Hereth's method in [4]. In the consequence we can reduce the time to build the concept lattice for functional dependencies and we can eliminate useless dependencies.

An inverted index (or inverted file) is an index data structure, a sequence of (*key*, *pointer*) pairs where each pointer points to a *record* in a database which contains the key value in some particular field. The inverted index is a central component of a typical search engine indexing algorithm. For databases in which the records may be searched based on more than one field, multiple indices may be created.

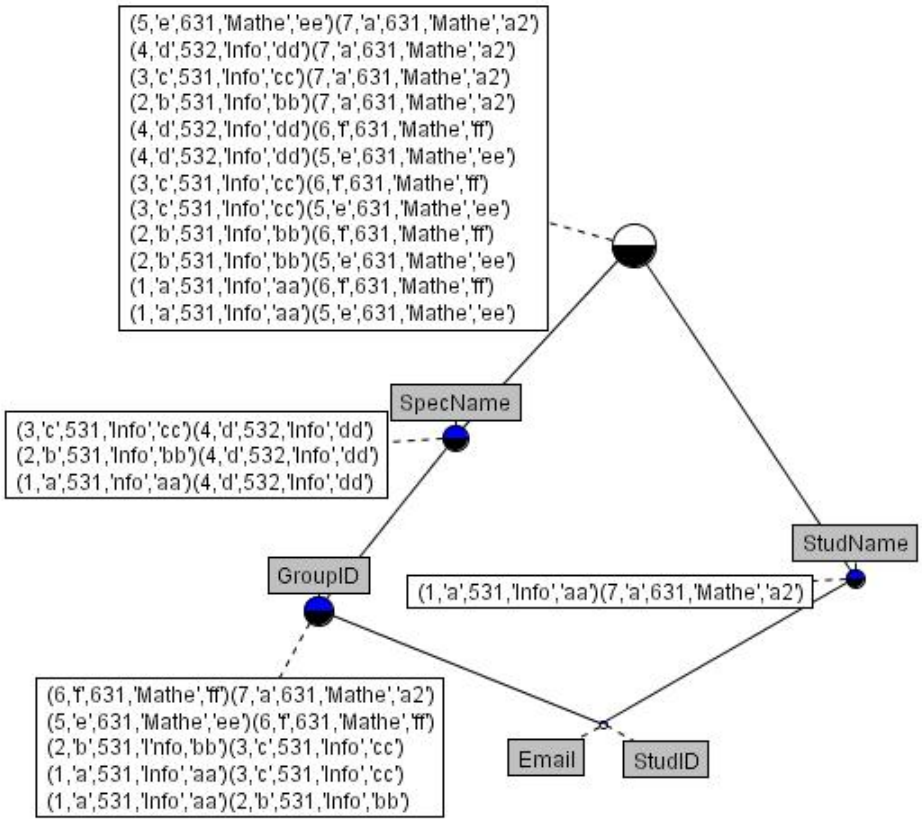

FIGURE 2. Conceptual lattice for $FD\,(StudentInfos, \rightarrow(Uni))$

Value	Row numbers
v_{1j}	rnr^1_{1j}, rnr^2_{1j}, ...
v_{2j}	rnr^1_{2j}, rnr^2_{2j}, ...
...	
v_{mj}	rnr^1_{mj}, rnr^2_{mj}, ...

TABLE 4. Inverted index InvInd_j

We use the following notations for the j-th inverted file, which contains the different values of the attribute a_j: v_{1j}, v_{2j}, ..., v_{mj} and for each value the row numbers, where the corresponding attribute value appears, see Table 4.

Rownrs	*StudID*	*StudName*	*GroupID*	*SpecName*	*Email*
1	1	a	531	Info	aa
2	2	b	531	Info	bb
3	3	c	531	Info	cc
4	4	d	532	Info	dd
5	5	e	631	Mathe	ee
6	6	f	631	Mathe	ff
7	7	a	631	Mathe	a2

TABLE 5. Table StudentInfos

StudID	
value	row nrs
1	1
2	2
3	3
4	4
5	5
6	6
7	7

StudName	
value	row nrs
a	1,7
b	2
c	3
d	4
e	5
f	6

GroupID	
value	row nrs
531	1,2,3
532	4
631	5,6,7

SpecName	
value	row nrs
Info	1,2,3,4
Mathe	5,6,7

Email	
value	row nrs
aa	1
bb	2
cc	3
dd	4
ee	5
ff	6
a2	7

TABLE 6. Inverted index files for table StudentInfos

The context of functional dependencies being constructed, we build the concept lattice. In the top of the concept lattice will be tuple pairs, in which are no common values of the corresponding attributes, so we can omit to generate these pairs of tuples. Pairs of form (t, t), where t is a tuple of the table, have all attributes in common, these objects will arrive in the bottom of the lattice, so they can be omitted too, because they don't change the implications in the context. Finally from the resulted context we generate the functional dependencies.

Example 6. There is a simple example on how to build inverted index file. Let be the following rows in table `StudentInfos` and the inverted index files for every attribute, see Table 5 and 6.

The previous considerations alow us to formulate the next algorithm to build the context of functional dependencies, inverted index files being constructed in the same time.

Algorithm
for each inserted row in table T **do**
begin
let k be the number of row
let $e_{k1}, e_{k2}, ..., e_{kn}$ be the attribute values of row k
for j:=1 to n **do** // for every attribute value
begin
search e_{kj} in the j-th inverted index file //search the attribute
// value in the corresponding inverted file
if find, let v_{lj} be the value in the inverted file
such that $e_{kj} = v_{lj}$ **then**
begin // k is added to the list of row numbers for value v_{lj}
build the array list $al_{kj} = \{rnr^1_{lj}, rnr^2_{lj}, ...\}$
add k in al_{kj}
add k in the j-th inverted index in the list of row numbers
for value v_{lj}
end
else //value e_{kj} doesn't exist in the corresponding inverted index
//we insert it, k is the first row with value e_{kj} of attribute j
insert new line in the j-th inverted index file with values (e_{kj}, k)
end
// In order to insert tuple pairs as rows in the cex file:
build $al_k = \bigcup_{j=1}^{n} al_{kj}$ //al_k contains the row numbers,
// which have attributes in common with row k
//if al_k is empty, no row will be inserted in cex file
if $al_k \neq \emptyset$ **then**
for $s = 1$ to count(al_k) **do**
insert in cex file tuple (k, $al_k(s)$)
end

Example 7. Let be the next table:

```
StudMarks [StudID, StudName, GroupID, Email, SpecID,
 SpecName, Language, DiscID, DName, CreditNr, Mark]
```

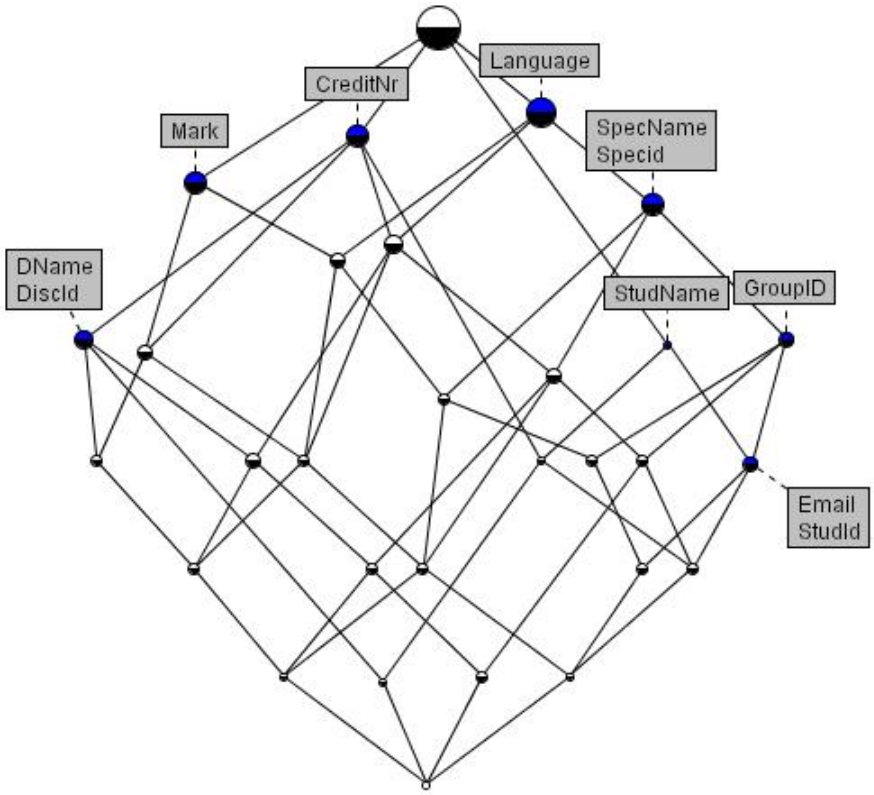

FIGURE 3. Conceptual lattice for $FD\left(StudMarks, \overrightarrow{\mathbb{K}}(Uni)\right)$ with 92 rows in the table StudMarks

In Fig. 3 is the concept lattice of the functional dependencies context for the table `StudMarks` with 92 rows. The implications in this lattice, which are functional dependencies in the table can be seen as follows: the concept with label `GroupID` is a subconcept of concept with labels `SpecID` and `SpecName`. This means, in every tuple pair where the `GroupID` field has the same value, the specialization ID and name is the same. So we have the following implications, which are functional dependencies:

$$GroupID \rightarrow SpecID,\ SpecName$$

In the same manner the following functional dependencies can be read from the concept lattice:

$$DiscID \rightarrow CreditNr$$
$$DName \rightarrow CreditNr$$

1 < 43 > SpecId ==> SpecName Language;
2 < 43 > SpecName ==> SpecId Language;
3 < 26 > GroupID ==> SpecId SpecName Language;
4 < 15 > DiscId ==> DName CreditNr;
5 < 15 > DName ==> DiscId CreditNr;
6 < 13 > StudId ==> StudName GroupID Email SpecId SpecName Language;
7 < 13 > Email ==> StudId StudName GroupID SpecId SpecName Language;
8 < 13 > StudName Language ==> StudId GroupID Email SpecId SpecName;
9 < 11 > StudName Mark ==> StudId GroupID Email SpecId SpecName Language CreditNr;
10 < 11 > SpecId SpecName Language CreditNr Mark ==> StudId StudName GroupID Email;
11 < 11 > StudId StudName GroupID Email SpecId SpecName Language CreditNr ==> Mark;
12 < 10 > SpecId SpecName Language DiscId DName CreditNr ==> StudId StudName GroupID Email Mark;
13 < 10 > StudName DiscId DName CreditNr ==> StudId GroupID Email SpecId SpecName Language Mark;

FIGURE 4. Implications, namely functional dependencies in the table StudMarks

$$SpecID \rightarrow Language$$
$$SpecName \rightarrow Language$$
$$StudID \rightarrow StudName$$
$$Email \rightarrow StudName$$

The implications in this lattice given by Conexp can be seen in Fig. 4.

4. Conclusions and further research

This paper presents a method to optimize and to reduce the workload of the users in the process of mining functional dependencies in a relational database. Our method proves that FCA visualization makes easier to manage database schemata and normal forms.

Our goal for further research is to develop an FCA exploration software based on this method to be a front-end to relational database of any content. This software would read any type of database table and construct the context of functional dependencies, this way would not be necessary to introduce the tuples by hand. As a part of this solution, we would use the resulted implications to propose the structure of the database tables.

References

[1] Abiteboul, S., Hull, R., Vianu, V.: Foundations of databases. Addison-Wesley, Reading - Menlo - New York (1995)
[2] Baixeries, J.: A formal concept analysis framework to mine functional dependencies, Proceedings of Mathematical Methods for Learning, (2004).
[3] Ganter, B., Wille, R.: Formal Concept Analysis. Mathematical Foundations. Springer, Berlin-Heidelberg-New York. (1999)

[4] Hereth, J.: Relational Scaling and Databases. Proceedings of the 10th International Conference on Conceptual Structures: Integration and Interfaces LNCS 2393, Springer Verlag (2002) 62–76
[5] Priss, U.: Establishing connections between Formal Concept Analysis and Relational Databases. Dau; Mugnier; Stumme (eds.), Common Semantics for Sharing Knowledge: Contributions to ICCS, (2005) 132–145
[6] Silberschatz, A., Korth, H. F.,Sudarshan, S.: Database System Concepts, McGraw-Hill, Fifth Edition, (2005)
[7] Wille, R. : Restructuring lattice theory: an approach based on hierarchies of concepts. In: I.Rival (ed.): Ordered sets. Reidel, Dordrecht-Boston, (1982) 445–470
[8] Yao, H., Hamilton, H. J.: Mining functional dependencies from data, Data Mining and Knowledge Discovery, Springer Netherlands, (2007)
[9] Serhiy A. Yevtushenko: System of data analysis "Concept Explorer". (In Russian). Proceedings of the 7th National Conference on Artificial Intelligence KII-2000, p. 127-134, Russia, 2000.

Sapientia University, Tg-Mures, Romania
E-mail address: tsuto@ms.sapientia.ro

Babes-Bolyai University, Cluj, Romania
E-mail address: ivarga@cs.ubbcluj.ro