Establishing connections between Formal Concept Analysis and Relational Databases

Uta Priss

School of Computing, Napier University, Edinburgh, UK
www.upriss.org.uk
u.priss@napier.ac.uk

Abstract. The relationship between relational databases and formal concept analysis (FCA) has been the topic of several papers in the past. This paper intends to extend some of the ideas presented in the previous papers by analysing the relationship between FCA and two central notions of relational databases: database schemata and normalforms. An object-relational algebra is suggested in this paper as a possible future replacement of relational algebra.

1 Introduction

The relationship between relational databases and formal concept analysis (FCA, cf. Ganter & Wille (1999)) has been the topic of several papers in the past. Hereth (2002) discusses relational scaling and databases. He translates relational algebra, which provides a theoretical foundation for relational databases, into an FCA representation. This paper intends to extend the previous research on FCA and databases, by analysing areas of database design in which FCA methods could be useful for modelling and visualisation: the areas of database schemata and normalforms.

This paper attempts to develop the foundations for an "object-relational algebra", which one day may serve as a substitute for the traditional relational algebra used as the foundation of the database query language SQL (cf. Negri et al. (1991)). Object-relational algebra is compatible with FCA because it uses binary relations, which can be interpreted as formal contexts. Some of the context concatenations proposed in object-relational algebra have already been used in other FCA applications (such as described by Priss(1998)).

With respect to database normalforms, Hereth (2002) has already described the relationship between FCA and functional dependencies, which are essentially implications between different columns of the database tables. Related to functional dependencies are several normalforms, which are defined in traditional database theory as a means of reducing redundancy and avoiding update anomalies in relational databases. In this paper, it is argued that FCA can also be used to visualise normalforms, which could be beneficial primarily for educational purposes. To our knowledge, so far this relationship between normalforms and FCA has never been stated in any research papers.

2 An Object-Relational Algebra

In traditional database theory, a relational algebra is used as the underlying formalisation for query optimisation and formal semantics (cf. Negri et al. (1991)). This relational algebra is closely tied to SQL because most of its operations correspond directly to SQL keywords. Because of this close connection, it can be difficult to extend relational algebra to more general languages, such as object-relational formalisms and other relational operations. It is therefore of interest to study possible alternatives to relational algebra.

There are at least three types of algebras which are similar or related to relational algebra: first, there are relation algebras (i.e. "relation" instead of "relational"). They are studied in the field of algebraic logic and can be traced back to Peirce and de Morgan via Tarski (1941) and Schröder (cf. Pratt (1992) for an overview). Second, there are Krasner algebras, which have been explored with respect to their relationship to Peirce's algebra by Hereth Correia & Pöschel (2004). Third, different variations have been suggested to Codd's (1970) original relational algebra in the field of relational databases.

The approach suggested in this paper is to use relation algebra (i.e. a Tarski-Peircean algebra) instead of relational algebra as the foundation for relational databases. This relation algebra (RA) is interpreted, first, with respect to Boolean matrices (cf. Kim (1982)) and, second, with respect to formal contexts in FCA. The goal is to develop an algebra which is suitable both for relational databases but also for more general object-relational structures. We call this algebra "object-relational algebra".

In object-relational algebra, binary relations, which correspond to Boolean matrices and to (binary) formal contexts in FCA, are used to represent most aspects of a database schema, such as table JOINs and subtype relations. Because of this, the terms "relation" and "matrix" may be used interchangeably in this paper. In addition to binary relations, many-valued matrices, which correspond to many-valued contexts in FCA, are used to represent the values of relational database tables.

2.1 An interpretation of relation algebra with respect to FCA contexts

A Tarski-Peircean relation algebra (RA) consists of a set with operations for complementation ($^-$), dual (^{d}), logical OR ($\cup$), composition ($\circ$), null element (nul) and compositional one-element (dia). Maddux (1996) provides a detailed overview of these operations and of the axioms and basic theorems of this algebra. Further operations can be derived from the initial ones, such as logical AND ($\cap$) and the de Morgan complement of composition ($\bullet$). The complements of nul and dia are $\overline{nul}$ and $\overline{dia}$, respectively. It appears that RA has a large field of applications. Pratt (1993) explains that Chu spaces can be seen as an interpretation of RA. Maddux derives programming language semantics as an interpretation of RA.

In this paper the RA operations are interpreted with respect to Boolean matrices (in the sense of Kim (1982)). The RA operations cannot be applied to all Boolean matrices without further restrictions. For example, $\cup$ and $\cap$ require the matrices to have equal numbers of columns. In the case of composition (see the example below), the number of columns of the left matrix must equal the number of rows of the right matrix. Thus the operations are only partially defined on the set of all Boolean matrices. In matrices, logical AND and OR are calculated position-wise combining each value of the left matrix with the one at the same position in the right matrix. Different one-elements are required for matrices of different sizes. But all one-elements are of the form, $dia_{\mathtt{m},\mathtt{n}}$, with m rows and n columns, 1s on the diagonal and 0s otherwise. The null elements, $nul_{\mathtt{m},\mathtt{n}}$, are matrices with just 0's. Two new types of operations are of relevance for Boolean matrices: first, for matrices I and J, apposition and subposition consist of adding rows or columns (written as $I|J$ and $\frac{I}{J}$). Second, the reduction operation red deletes certain rows and columns from a matrix.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \circ \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} (ae+bg) & (af+bh) \\ (ce+dg) & (cf+dh) \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \circ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Example 1. Matrix composition.

The interpretation of RA with respect to Boolean matrices can be extended by considering the matrices to be the cross-tables of formal contexts in the sense of formal concept analysis (Ganter & Wille, 1999). RA becomes thus an algebra of operations on FCA contexts. This interpretation requires some further adjustments of the operations. In the rest of this paper, it is assumed that in each formal context, the set `G` of formal objects and the set `M` of formal attributes are linearly ordered and that if different formal contexts share a set of objects or attributes then these are in the same linear order. The RA operations may only be meaningful for formal contexts, if they are applied to formal contexts which share sets of objects and/or attributes. For example, composition of formal contexts usually assumes that each attribute of the left context equals an object of the right context.

To distinguish the traditional set-based FCA notations from matrices, type-writer font is used in this paper for sets and elements while italics are used for matrices and formal contexts. Unless stated otherwise, for the rest of this paper, $K = (\mathtt{G}, \mathtt{M}, I)$ denotes a formal context with `G` as set of objects and `M` as set of attributes. $\mathtt{H} \subseteq \mathtt{G};\ \mathtt{N} \subseteq \mathtt{M}$ are subsets and $\mathtt{g} \in \mathtt{G};\ \mathtt{m} \in \mathtt{M}$ are elements.

The interpretation of RA on FCA contexts necessitates the introduction of further operations so that matrices can be converted into sets (of objects or attributes) and vice versa. For a linearly ordered set `T` and a subset `S` of `T`, the operation $row_{\mathtt{T}}(\mathtt{S})$ denotes a row matrix of length $|\mathtt{T}|$ which has a 1 in the position of each element of `S` that is also in `T` and 0's otherwise. The operations $col_{\mathtt{T}}(\mathtt{S})$ and $dia_{\mathtt{T}}(\mathtt{S})$ are defined accordingly: $col_{\mathtt{T}}(\mathtt{S})$ producing a column matrix and $dia_{\mathtt{T}}(\mathtt{S})$ producing a matrix with 1's on the diagonal according to the positions of the elements in `S` and 0's otherwise. (Both $col_{\mathtt{T}}(\mathtt{S})$ and $row_{\mathtt{T}}(\mathtt{S})$ can be derived from $dia_{\mathtt{T}}(\mathtt{S})$ through composition with $\overline{nul}$).

With the notations from the previous two paragraphs, sets in a formal context can be converted into matrices as follows: $G := row_{\mathtt{G}}(\mathtt{G})$, $H := row_{\mathtt{G}}(\mathtt{H})$, $\underline{g} := row_{\mathtt{G}}(\{\mathtt{g}\})$, $M := col_{\mathtt{M}}(\mathtt{M})$, $N := col_{\mathtt{M}}(\mathtt{N})$, $\underline{m} := col_{\mathtt{M}}(\{\mathtt{m}\})$. Underlining is used to indicate that the matrices correspond to single-element sets and thus contain exactly one 1. For the remainder of this paper, row or column matrices with a single 1 shall be called "single-element matrices". The matrices can be converted back into sets using the following operations: $\mathtt{H} = \mathtt{set}_{\mathtt{G}}(H)$ and $\mathtt{N} = \mathtt{set}_{\mathtt{M}}(N)$.

The main operations of FCA can now be represented as summarised in table 1. The plus ($+$) operator, which is somewhat dual to the prime ($'$) operator originates from use in lexical databases (cf. Priss(1998) and Priss & Old (2004)). The operations on attributes $\underline{m}'$, $\underline{m}^+$, N' and N^+ are analogous to the ones in the table. It should be noted that for single-element matrices, the plus and prime operators yield the same result (i.e., $\underline{g} \circ I = \overline{\underline{g} \circ \overline{I}}$) because composing with $\underline{g}$ corresponds to selecting one row from I, or, in standard FCA terminology, the existence- and all-quantifiers are equivalent for single-element sets.

standard FCA	relation algebra
$\mathtt{g}I\mathtt{m}$	$\underline{g} \circ I \circ \underline{m} = (1)$
$\mathtt{g}' := \{\mathtt{m} \in \mathtt{M} \mid \mathtt{g}I\mathtt{m}\}$	$\underline{g}' = \underline{g}^+ := \underline{g} \circ I$
$\mathtt{H}' := \{\mathtt{m} \in \mathtt{M} \mid \forall_{\mathtt{g} \in \mathtt{G}} : \mathtt{g} \in \mathtt{H} \Longrightarrow \mathtt{g}I\mathtt{m}\}$	$H' := \overline{H \circ \overline{I}}$
$\mathtt{H}^+ := \{\mathtt{m} \in \mathtt{M} \mid \exists_{\mathtt{g} \in \mathtt{G}} : \mathtt{g} \in \mathtt{H}$ and $\mathtt{g}I\mathtt{m}\}$	$H^+ := H \circ I$

Table 1. FCA terminology translated into relation algebra

$\mathtt{N}^+$	$I \circ N$	$\exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{m} \in \mathtt{N}$ and $\mathtt{g}I\mathtt{m}$	at least one, some
$\mathtt{G} \setminus \mathtt{N}^+$	$\overline{I \circ N}$	$\neg\exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{m} \in \mathtt{N}$ and $\mathtt{g}I\mathtt{m}$	none
$\mathtt{N}'$	$\overline{\overline{I} \circ N}$	$\neg\exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{m} \in \mathtt{N}$ and $\neg(\mathtt{g}I\mathtt{m})$	relates to all
$\mathtt{G} \setminus \mathtt{N}'$	$\overline{I} \circ N$	$\exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{m} \in \mathtt{N}$ and $\neg(\mathtt{g}I\mathtt{m})$	does not relate to all
$(\mathtt{M} \setminus \mathtt{N})^+$	$I \circ \overline{N}$	$\exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{m} \notin \mathtt{N}$ and $\mathtt{g}I\mathtt{m}$	relates to those that are not only
$\mathtt{G} \setminus ((\mathtt{M} \setminus \mathtt{N})^+)$	$\overline{I \circ \overline{N}}$	$\neg\exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{m} \notin \mathtt{N}$ and $\mathtt{g}I\mathtt{m}$	relates to those that are only
$(\mathtt{M} \setminus \mathtt{N})'$	$\overline{\overline{I} \circ \overline{N}}$	$\neg\exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{m} \notin \mathtt{N}$ and $\neg(\mathtt{g}I\mathtt{m})$	relates to all outside
$\mathtt{G} \setminus ((\mathtt{M} \setminus \mathtt{N})')$	$\overline{I} \circ \overline{N}$	$\exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{m} \notin \mathtt{N}$ and $\neg(\mathtt{g}I\mathtt{m})$	does not relate to all outside

Table 2. Eight quantifiers

Operations on FCA contexts have been known for a while (cf. Ganter & Wille (1999)). But to our knowledge there has not been a significant amount of interest among FCA researchers in the RA that underlies such operations. The reason for this may be that in many FCA applications, context operations are not of relevance because they may not lead to interesting properties of the lattice visualisations. We argue, however, that in the field of databases, such context operations are of major importance because they can express quantified relationships between database tables. A many-valued database table can be interpreted as a binary matrix by replacing every non-NULL value with 1 and every NULL value with 0. Using such binary matrices, quantified relationships among database tables can be described using RA as summarised in table 2. The plus and prime operators and their complements roughly correspond to eight basic natural language quantifiers: “some”, “none”, “all”, “not all”, “only”, “not only”, “all outwith”, “not all outwith”. This should be of interest to relational databases because the currently most frequently used database language SQL supports “exists” as a basic quantifier, but not any of the other seven. Hansen & Hansen (1996, p. 202) observe that the lack of an "all" quantifier in SQL creates problems for database users. Another advantage of this approach is that in contrast to SQL which always requires a three-valued logic because of NULL (cf. Negri et al. (1991)), in this approach many operations can be performed in the two-valued binary logic.

2.2 Compositional Schemata

One goal for this paper is to construct a complex formal context which summarises all tables of a given relational database. In order to reach this goal, the notion of a "compositional schema" is introduced. A compositional schema (as shown in figure 1) consists of several formal contexts, some of which are composed using any of the quantifiers in table 2. Such compositional schemata are called "relational context schemata" by Priss (1998). Similar concatenations using formal contexts are discussed by Ganter & Wille (1999) and, for example, by Faid et al. (1997).

	A	B	C
B	1	2	3
A	4	5	6
D	7	8	9

	A	B	C
B			L
A		J	$J \circ L$
D	I	$I \circ J$	$I \circ J \circ L$

Fig. 1. A compositional schema

The numbering of the nine cells as presented in the left hand side of figure 1 shall be used in the remainder of this paper. The compositional schema in figure 1 is built from the formal contexts $K_J := (\mathtt{A}, \mathtt{B}, J)$; $K_I := (\mathtt{D}, \mathtt{A}, I)$ and $K_L := (\mathtt{B}, \mathtt{C}, L)$. Because K_I and K_J share the set A, a context $K_{I \circ J} := (\mathtt{D}, \mathtt{B}, I \circ J)$ can be formed. Similarly, a context $K_{J \circ L}$ can be formed. Instead of the existence quantifier used in the construction of the matrices in cells 6, 8 and 9, any of the other seven quantifiers from table 2 can be used. A further context $K_{I \circ J \circ L}$ can be formed in cell 9 to complete the schema. It should be noted that while $K_{J \circ L}$ and $K_{I \circ J}$ are formed by composing the context to the left with the one above, $K_{I \circ J \circ L}$ is formed by composing the context to the left with the context two steps above (or the context two steps to the left with the one above). An exception is if J is a reflexive, transitive relation, in which case $J \circ J = J$ and $I \circ J \circ L = I \circ J \circ J \circ L$. The cells 1, 2, 4 can be left empty or cells 2 and 4 can be filled with an identity relation dia.

Figure 2 shows a compositional schema which can be constructed for any formal context $K := (\mathtt{G}, \mathtt{M}, I)$ where $\mathtt{P}(\mathtt{G})$ and $\mathtt{P}(\mathtt{M})$ denote the power set of G and M, respectively, and $\in$ and $\ni$ denote the "element of" relation: $\mathtt{H} \ni \mathtt{g}$ and $\mathtt{m} \in \mathtt{N}$. Again, any of the 8 quantifiers from table 2 can be used in cells 6 and 8. This compositional schema summarises information relevant to K. for example, the rows of cell 8 show the chosen quantifier for every subset of G. It can be shown that if the set $\{\mathtt{N}, \mathtt{M}\backslash\mathtt{N}\}$ is chosen instead of $\mathtt{P}(\mathtt{M})$, if $I \circ \in$ is chosen for cell 6, and a lattice diagram is constructed for cells 2, 3, 5, 6, all 8 quantifiers according to table 2 can be read from the lattice diagram.

	G	M	P(M)
M		dia	$\in$
G	dia	I	$I \circ \in$
P(G)	$\ni$	$\ni \circ I$	

Fig. 2. A compositional schema for $(\mathtt{G}, \mathtt{M}, I)$

Compositional schemata can also be utilised for summarising information about object-oriented class hierarchies. The relational composition, $\circ$, seems to be a natural means for expressing inheritance with respect to object-oriented modelling. Figure 3 shows a compositional schema for a class in the sense of object-oriented modelling. The matrix I_{inst} describes which instances belong to which classes; I_{sub} contains the class hierarchy; I_{attr} describes which classes have which attributes. It should be noted that I_{attr} only refers to the existence of attributes not their values. For example, I_{attr} could specify that a class "employee" contains the attributes "employeeID", "name" and "address". This compositional schema assumes that attributes are named uniquely across the whole class hierarchy. But this condition can always be fulfilled by initially prefixing all attributes with the name of the class in which they are defined, and then, optionally, dropping those prefixes which are not required. The relation I_{sub} is assumed to be reflexive and transitive, thus $I_{inst} \circ I_{sub} \circ I_{attr} = I_{inst} \circ I_{sub} \circ I_{sub} \circ I_{attr}$.

	Classes	Classes	Attributes
Classes			I_{attr}
Classes		I_{sub}	$I_{sub} \circ I_{attr}$
Instances	I_{inst}	$I_{inst} \circ I_{sub}$	$I_{inst} \circ I_{sub} \circ I_{attr}$

Fig. 3. A compositional schema for class hierarchies

According to Atkinson et al. (1989), there are at least four types of inheritance: substitution, inclusion, constraint and specialisation inheritance. Because various degrees of these types can be combined in actual systems, the notion of inheritance can be quite confusing. Object-relational algebra helps to clarify two of the four types: inclusion inheritance is the ordering of classes that emerges from I_{inst} because it refers to subset relations among the sets of instances of classes. Specialisation inheritance is the ordering of classes that emerges from I_{attr} because it refers to subset relations among the sets of attributes of classes. It should be noted that the concept lattice of the formal context based on cells 5, 6, 8 and 9 in figure 3 contains the information about both inclusion and specialisation inheritance and also about I_{sub} by itself in a single diagram - although in practical applications this diagram can become too large to be graphically displayed. The other two types of inheritance are based on behaviour (substitution inheritance) and values (constraint inheritance) and cannot be explained using just binary matrices. Constraint inheritance requires a many-valued relation instead of $I_{inst} \circ I_{sub} \circ I_{attr}$ which contains the actual values of the attributes for each instance. Such many-valued matrices are further discussed in the next section.

2.3 Relational database tables in a compositional schema

	Tables	Tables	Columns
Tables			I_{attr}
Tables		dia	I_{attr}
Keys	I_{inst}	I_{inst}	$I_{inst} \circ I_{attr}$

Fig. 4. A compositional schema for relational databases

The basic compositional schema for relational databases in figure 4 is a simplified version of the one for classes in figure 3. In relational databases, tables (or entity types) cannot be subtypes of other tables. For example, it cannot be expressed that a manager table shall be a subtype of a staff table. Therefore instead of an I_{sub} matrix, cell 5 is filled by an identity matrix dia. It follows that cell 9 contains $I_{inst} \circ I_{attr}$ because $I_{inst} \circ I_{attr} = I_{inst} \circ dia \circ I_{attr}$.

		Columns	
	Tables	Key Columns	Other Columns
Tables	dia	I_{attr}	
Keys	I_{inst}	$I_{inst} \circ I_{attr}$	

Fig. 5. A relational database schema

The first row and column in figure 4 do not contain any useful information and can be omitted. The only reason for presenting the schema in that manner is to demonstrate that relational databases are a special case of object-relational databases. After omitting the first row and column, the attributes can be grouped so that all key attributes appear on the left side of I_{attr}. Figure 5 shows the emerging schema and figure 6 shows an example. Figure 5 (or 6) contains all the information that is contained in a single relational database. Using the example in figure 6, the employee table (Emp) can be retrieved as follows: $\underline{Emp} = col_{\mathtt{Tables}}(\{\mathtt{Emp}\})$ retrieves the positions of the key attributes among the formal objects. $\mathtt{G}_{Emp} = \mathtt{set}(red_{\mathtt{Keys}}(\underline{Emp'}))$ is the set of formal objects of `Emp`; $\mathtt{M}_{Emp} = \mathtt{set}(red_{\mathtt{Columns}}(\underline{tEmp'}))$ is the set of formal attributes; $I_{Emp} = red_{\mathtt{G}_{Emp},\mathtt{M}_{Emp}}(I_{inst} \circ dia(\underline{Emp}) \circ I_{attr})$ is a binary matrix for the relation of `Emp` and, finally, $C_{Emp} = (\mathtt{G}_{Emp}, \mathtt{M}_{Emp}, I_{Emp})$ represents a formal context for `Emp`. The matrix I_{Emp} is binary and contains a 1 for every non-NULL value and 0 for every NULL value. The actual values of the table can be stored using a mapping $mv()$ from binary matrices to many-valued matrices. The full database table `Emp` thus corresponds to the many-valued formal context $(\mathtt{G}_{Emp}, \mathtt{M}_{Emp}, mv(I_{Emp}))$.

All tables of a relational database can be combined in this manner in one many-valued matrix (as in figure 6) if three assumptions are made:

– First, if the same name of an attribute is used in different tables it should have a similar meaning - otherwise the name should be prefixed by the table name.

	Emp	Work	Proj	eID	pID	ename	eaddr	cost	pname	ptime
tEmp	1			1		1	1			
tWork		1		①	①			1		
tProj			1		1				1	1
e1	1			1		paul	UK			
e2	1			2		mary	UK			
e3	1			3		carl	USA			
e4	1			4		sue	USA			
p1			1		1				A	2 wk
p2			1		2				B	2 wk
p3			1		3				C	30 wk
e1/p1		1		1	1			100		
e1/p2		1		1	2			100		
e1/p3		1		1	3			200		
e2/p1		1		2	1			400		
e3/p2		1		3	2			100		
e3/p3		1		3	3			200		

Fig. 6. An example of a relational database schema

– Second, entities are identified by keys.

– Third, the multiple inheritance anomaly (see below) is avoided.

With respect to the second assumption, in many relational databases, tables use foreign keys as part of composite keys, but the exact same set of key attributes is usually unique to each table. For such databases, each entity is unique to a single table and the cells 7, 8, 9 in figure 4 will simply show concatenations of the individual relations. But in some cases, relational databases contain hidden inheritance structures, which contain overlapping entities. For example, a database could contain separate tables for staff and managers but some employees could be both staff and manager. In this case, the database needs to be carefully designed to avoid a multiple inheritance anomaly. The schema in figure 5 helps identifying such possible problems. A possible problem occurs if the same keys are used in different tables and those different tables use the same attributes. In figure 6, all tables have different keys. Even though `eID` and `pID` are used in more than one table, they are part of a composite key in `Work` which is different from their use in `Emp` and `Proj`. A multiple inheritance anomaly would occur if a key is used in different tables relating to the same attribute, but the attribute has different values in each table. In that case the mapping $mv()$ cannot be defined.

The formal context in figure 5 contains all of the information that is contained in a power context family (Wille, 2002). This representation is different from Hereth's (2002) representation of relational database tables in FCA. Because power context families can represent conceptual graphs, a connection from relational databases to conceptual graphs can thus be established. The general idea of object-relational algebra is to define an algebra that uses compositional schemata as basic elements, that uses operations from RA and that contains a binary RA as a subalgebra. Due to space limitations, further details cannot be discussed in this paper.

2.4 Object-relational algebra and SQL

Using the definitions so far, it can now be attempted to translate SQL expressions using the RA operations described so far. Obviously, this is a complex task, only the beginnings of which can be explored in this paper. Two of the basic SQL operations are projection and selection. Projection corresponds to composition with a dia matrix from the right and selection corresponds to composition with a dia matrix from the left. For example, `select ename,eaddr from Emp` corresponds to $I_{Emp} \circ dia(\underline{ename} \cup \underline{eaddr})$ Selection is usually based on constraints. In object relational algebra, such constraints often require the use of many-valued matrices. For each many-valued attribute a so-called value context V_{attr} is constructed depending on the type of the attribute. Due to space limitations the details cannot be discussed in this paper. But the construction is similar to conceptual scaling (Ganter & Wille, 1999) or relational scaling (Hereth, 2002) in FCA. To provide an example for a select statement:
`select ename,eaddr from Emp where ename = 'mary' and eaddr = 'UK'` corresponds to
$dia((V_{ename} \circ \underline{mary}) \cap (V_{eaddr} \circ \underline{UK})) \circ I_{Emp} \circ dia(\underline{ename} \cup \underline{eaddr})$.

The natural JOIN in relational databases is fairly closely related to relational composition. This is because a natural JOIN combines two tables based on a shared column. A difference between an SQL JOIN and $\circ$ is, however, that depending on how the natural JOIN is formed (e.g. whether subqueries are involved), the query can return any number of duplicates. This is why such queries are often formulated using “select distinct” which eliminates the duplicates.

Consider the following example based on figure 6. Three database tables are Emp (an employee table), $Proj$ (a table for projects) and $Work$ (a table which shows the relationship of which employee works on which project). In SQL, it is fairly easy to retrieve all employees which work on a project:

```
select * from Emp, Work where Emp.eID=Work.eID
```

It is somewhat more difficult to retrieve all employees which do not work on any project:

```
select * from Emp where not exists
(select * from Work where Emp.eID=Work.eID)
```

And it is even more difficult to retrieve all employees who work on all projects:

```
select * from Emp where not exists
(select * from Proj where not exists
(select * from Work where Emp.eID=Work.eID
and Work.pID = Proj.pID))
```

A comparison with table 2 shows that the number of negations (relational complements) in SQL corresponds to the ones in table 2. But in contrast to the relational composition in the second column from table 2, the SQL statements vary in complexity. Of course, all of the statements above could be written using 2 subselect statements so that they are of equal complexity. But in our experience, many average database users would have extreme difficulty formulating SQL statements of such complexity. Most users would probably create a view or a temporary table, if they were asked to retrieve employees on all projects, to avoid using two subselect statements. In object-relational algebra, all of the statements above would be formed using $dia(I_{Work} \circ \underline{pid}) \circ I_{Emp}$ and using any of the quantifiers in the second column of table 2.

3 Normalforms and FCA

3.1 A brief introduction to database normalforms

This section explores a second means of using FCA for modelling an aspect of relational databases. In relational database design, it is of importance to avoid data redundancy and anomalies that can occur when data is inserted, deleted or updated (cf. any textbook on relational databases). The standard solution to such problems is database normalisation. Several of so called "normalforms" are in use, some of which rely on the notion of "functional dependency". Hereth (2002) has shown that a database can be translated into a power context family and that the functional dependencies of such a database correspond to FCA implications in a certain formal context. In this paper, FCA is used as a means for visualising the structures of the different normalforms by representing functional dependencies as implications in a lattice as Hereth (2002) proposed. It is not suggested that these visualisations solve any computational problem or create new means for practical implementations. Instead, the visualisations are meant to serve as explanatory aids. In our experience, many students tend to experience a great deal of difficulty in grasping the idea of normalisation. Thus a visual aid should be beneficial.

This section mainly focuses on the Second and Third Normalform (2NF and 3NF) and on the Boyce Codd Normalform (BCNF), which are all briefly described below. It should be noted that beyond the textbook versions of the standard normalforms, there have also been some other normalforms proposed in the literature. The sequence of the normalforms has emerged due to historical reasons, which explains why BCNF was inserted between 3NF and 4NF. The normalforms, 2NF, 3NF and BCNF, rely on functional dependencies within a single table. Functional dependencies can either describe the current state (an attribute is dependent on another one) or a prescriptive state (an attribute must always depend on another one). Thus it is not always unambiguous as to what constitute the functional dependencies of a database table. Other types of dependencies utilise JOINs (such as 5NF) and inter-table dependencies. In general, although current database researchers consider the study of normalforms as completed, it might still be interesting to revisit this area from an FCA viewpoint, to shed some light on the differences and relations between the normalforms and to investigate whether there are any new insights for FCA implications to be drawn from database research.

According to Hansen & Hansen (1996) "a relation is in first normalform (1NF) if the values in the relation are atomic". In other words, the values are elements, not sets, tuples or subtables. Relational databases usually fulfil this normalform automatically (although modern object-relational databases may not - but that is a different story). A database table is in 2NF if it is in 1NF and "no non-key attribute is functionally dependent on just a part of the key" (Hansen & Hansen, 1996). A database table is in 3NF if it is in 2NF and there are no transitive functional dependencies, where "transitivity" refers to the existence of functional dependencies among non-key attributes. A database table is in BCNF if it is in 3NF and every non-trivial, left-irreducible functional dependency has a candidate key as its determinant (where a "trivial" implication is one where the left-hand side and the right-hand side are identical or where the left-hand side is either the empty set or the full set of attributes; "determinants" are left-hand sides of functional dependencies and "candidate key" means that these attributes could be used as an alternative key). Other normalforms, such as 4NF and 5NF, shall be ignored in this paper.

3.2 FCA visualisations of database normalforms

To use FCA for visualising normalforms, the functional dependencies (including the dependencies on the key attributes) are considered as an implicational basis of a lattice. This lattice is called the "FD lattice" of a database table for the remainder of this paper. It can be used to visually inspect the normalforms. Because by definition all attributes of a database table must be dependent on the key, the meet of the key attributes must equal the bottom node of the FD lattice. The easiest normalform to be visualised is BCNF as described in the following proposition.

> **Proposition**. A database table is in BCNF if
> 1) all non-trivial implications of its FD lattice have a left-hand side which meets in the bottom node of the lattice;
> 2) if the lattice contains any non-key attributes, then no true subset of the key attributes can meet in the bottom node; and
> 3) if a set consisting of only non-key attributes meets in the bottom node, then these must be the only non-key attributes in the lattice.

This proposition is simply a translation of the BCNF definition into FCA terminology: for a determinant to be a candidate key means that the meet of its attributes equals the bottom node because all keys meet in the bottom node. Item 2) and 3) of the proposition ensure that the database table is also in 2NF and 3NF, respectively. Because of item 1), the conditions for 2NF and 3NF are much simpler than in the general case because only implications based on the bottom node need to be considered. It should be noted that 2NF and 3NF only restrict implications from key to non-key attributes and among non-key attributes. Implications from non-key to key attributes are not restricted in 2NF and 3NF, which is why items 2) and 3) make careful consideration of key and non-key attributes.

The examples of FD lattices in figure 7 shall help to illustrate some of these points. In this figure, key attributes are distinguished from other attributes by enclosing their names in a box. If the lattice contains only two nodes, then either the lattice is isomorphic to the one depicted in the far left example in figure 7, or it has only key attributes, in which case, more than one attribute can be attached to the bottom node. In general for BCNF lattices of any size, attributes must either be attached to the bottom node or they must be attached to co-atoms. If any of the co-atomic attributes meet above the bottom node, then the filter above this node must be a complete Boolean lattice (so that no subset of the attributes implies the other attributes).

2NF and 3NF are more difficult to visualise. One problem is that there is no FCA notion of key versus non-key attributes. The lattice on the left side of figure 8 is identical to the lattice in the middle from an FCA viewpoint. But the left lattice is not in 2NF because non-key attribute b depends on the partial key c. The lattice in the middle is in 2NF and 3NF but not in BCNF because c implies b but c is not a candidate key. The decomposition of this lattice into two lattices that are in BCNF is shown on the right.

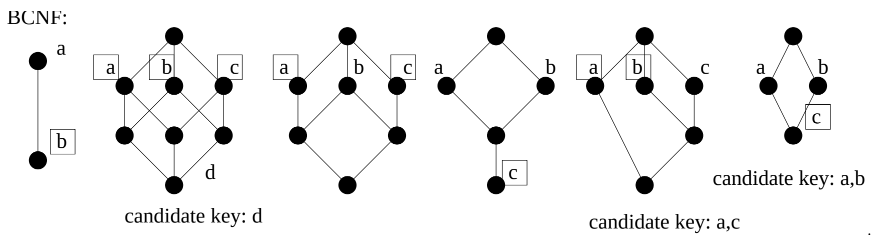

Fig. 7. FD lattices for relations in BCNF

This decomposition does not preserve the dependencies, because now a implies c. It is a known fact in the literature that BCNF decomposition can sometimes not be achieved with dependency preservation. It would be an interesting FCA question, whether from a lattice structural approach it can be determined in which cases BCNF is predictably dependency preserving and in which cases it is not.

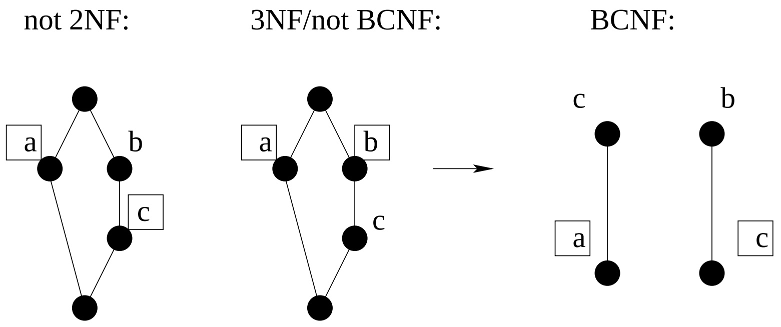

Fig. 8. The role of keys and BCNF decomposition

Figure 9 shows some more examples of lattices which are not in BCNF. It should be pointed out that the sequence of the normalforms (first, second, third, ...) is somewhat random because of its historical origin. In our experience with tutorials, students tend to detect and remove transitive dependencies (the main 3NF problem) before they tackle partial key dependencies (i.e., 2NF problems). Thus 3NF may in some way be more basic than 2NF.

For determining whether an FD lattice is 2NF or 3NF, all non-trivial implications other than the ones whose left-hand side meets in the bottom node need to be checked. Obviously, this can mean that many nodes need to be checked and - from a practical viewpoint - it may be easier to check the implication list directly without considering the lattice. But as explained before, the lattice visualisations can still help to convey an understanding of what the different normalforms mean. For 2NF, the meet of any subset of key attributes needs to be checked. In the examples in figure 9, a and b imply d in the middle lattice and c implies a in the right lattice. These are both violations of 2NF. For 3NF, it first needs to be determined whether the lattice is in 2NF. In the next step, only non-key attributes need to be considered. If any implications can be found among non-key attributes at all, then 3NF is violated.

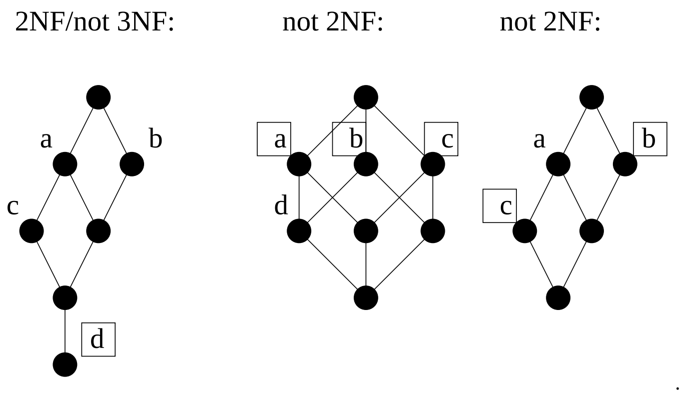

Fig. 9. FD lattices for relations in/not in 2NF and 3NF

4 Conclusion

This paper presents two applications of FCA with respect to relational databases. Object-relational algebra is suggested as a means for managing database schemata. FCA visualisations of database normalforms can serve as a means for exploring internals of such normalforms. Both topics still leave many open research questions.

A long-term goal for the research undertaken for this paper is to, ultimately, build FCA database exploration software, which can be used as a front-end to relational databases of any content. As far as we know, the only tool which currently exists that can be used in that manner is Cernato. But Cernato is a commercial tool and not freely available. Also it focuses on scaling of attributes and does not facilitate joining of database tables. As Cernato demonstrates, such FCA tools require user interaction and probably some degree of user training. It is not possible for users to simply “press a button” and view results. But other database exploration tools as used in Business Intelligence also require user interaction and training. Thus, in our opinion, FCA stands a chance to compete with such tools. To minimise the workload of users, an FCA database front-end would have to be carefully designed. We hope that research as presented in this paper might highlight some of the structures that could lead the way.

References

1. Atkinson, M.; Bancilhon, F.; DeWitt, D.; Dittrich, K.; Maier, D.; Zdonik, S. (1989). *The Object-Oriented Database System Manifesto.* In: Proceedings of the First International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan. p, 223-240.
2. Codd, E. (1970). *A relational model for large shared data banks.* Communications of the ACM, 13:6.
3. Faid, M.; Missaoui, R.; Godin, R. (1997). Mining Complex Structures Using Context Concatenation in Formal Concept Analysis. In: Mineau, Guy; Fall, Andrew (eds.), Proceedings of the Second International KRUSE Symposium. p. 45-59.
4. Ganter, B.; Wille, R. (1999). *Formal Concept Analysis.* Mathematical Foundations. Berlin-Heidelberg-New York: Springer, Berlin-Heidelberg.
5. Hansen, Gary; Hansen, James (1996). *Database Management and Design.* Prentice Hall.
6. Hereth, J. (2002). *Relational Scaling and Databases.* In: Priss; Corbett; Angelova (Eds.) Conceptual Structures: Integration and Interfaces. LNCS 2393, Springer Verlag. p. 62-76.
7. Hereth Correia, J.; Pöschel, R. (2004). *The Power of Peircean Algebraic Logic (PAL).* In: Eklund (Ed.) Concept Lattices. LNCS 2961, Springer Verlag. p. 337-351.
8. Kim, K. H. (1982). *Boolean Matrix Theory and Applications.* Marcel Dekker Inc.
9. Maddux R. (1996). *Relation-algebraic semantics.* Theoretical Computer Science, 160, p. 1-85.
10. Negri, M.; Pelagatti, G.; Sbattella, L. (1991). *Formal Semantics of SQL Queries.* ACM Transactions on Database Systems, 17, 3. p. 513-534.
11. Pratt, V.R. (1992). *Origins of the Calculus of Binary Relations.* Proc. IEEE Symp. on Logic in Computer Science, p. 248-254.
12. Pratt, V.R. (1993). *The Second Calculus of Binary Relations.* Proc. 18th International Symposium on Mathematical Foundations of Computer Science, Gdansk, Poland, Springer-Verlag, p. 142-155.
13. Priss, U. (1998). *Relational Concept Analysis: Semantic Structures in Dictionaries and Lexical Databases.* (PhD Thesis) Verlag Shaker, Aachen 1998.
14. Priss, U.; Old, L. J. (2004). *Modelling Lexical Databases with Formal Concept Analysis.* Journal of Universal Computer Science, Vol 10, 8, 2004, p. 967-984.
15. Tarski, A. (1941). *On the calculus of relations.* Journal of Symbolic Logic, 6, p. 73-89.
16. Wille, Rudolf (2002). *Existential Concept Graphs of Power Context Families.* In: Priss; Corbett; Angelova (Eds.) Conceptual Structures: Integration and Interfaces. LNCS 2393, Springer Verlag, p. 382-395.