Flexible Relations – Operational Support of Variant Relational Structures

Christian Kalus Peter Dadam
Department of Databases and Information Systems, University of Ulm
e-mail:{kalus,dadam}@informatik.uni-ulm.de

Abstract

The relational model is accepted for its simplicity and elegance. At the other side the simplicity causes the problem, that most semantic type constructs are not representable as a simple relation. Variant and heterogeneous structures belong to those constructs not adequatly supported by the simple relational model. In this paper we give an overview of the model of *flexible relations* that allows to model and process arbitrary heterogeneous structures, while preserving the relational philosophy of operating with a single constructor. As flexible relations support both the modeling and the operational aspect of variant structures seamlessly, our model truly helps to further bridge the gap between semantic and operational data models.
We discuss the structural part of the model and introduce an algebra for flexible relations. Further we examine a subclass of flexible relations, that can be processed as efficiently as the simple relational model, and show that this subclass possesses desirable structural normal form properties. In addition, we point out that our approach exceeds the object–oriented paradigm in modeling power, typing precision, and query optimization potential.

1 Introduction

The relational model ([Cod70]) is the most accepted operational data model due to its mathematical foundation, its simplicity and elegance. Both elegance and simplicity are mainly due to the fact that it uses a single constructor, the *relation*. A disadvantage of this aspect is that many useful modelling constructs are not adequatly supported by the relational model ([Ken79]). This fact caused the development of *semantic data models* (see [PM88], [HK87] for overviews). As semantic data models do emphasize the modelling aspect and typically do not offer query processing or data manipulation facilities, they have to be mapped onto operational data models. As the simple relational model is too weak to support these modelling aspects, extensions of the relational model have to be developed to bridge this semantic gap. The challenge that relational extensions have to meet is to integrate the intended modelling aspects without sacrifying the benefits of the original relational model. The most convincing extension to the simple relational model is the NF^2 model (see [AFS89] as an entry point) that supports the modelling construct *aggregation* and *association*, but still employs a single constructor, namely an extended relation constructor.

The overall aim of the model described in this paper is to support each form of variant (and non–variant) structures in a generic way, thus providing an "operational engine" for structural aspects currently not supported by the relational model. We will show that our model integrates homogeneous and heterogeneous structures seamlessly, thus enlarging the scope of the relational model without giving up the benefits of the simple relational model.

There are several forms of variant structures occuring in semantic data models, and we would like to motivate them with an *address* type. The most popular variant type is the *exclusive union* telling that exactly one of its subtypes has to be present. An example of an exclusive union is the inner–town address consisting either of a post-office box or a street, but never of both. Another form of heterogenity can be motivated by the "electronic part" of an address composed of the telephone number, the telex number, and the electronic mail address. These attributes do not need to occur together nor do they exclude each other. Instead we

**Proceedings of the 21th VLDB Conference
Zurich, Switzerland 1995**

```
TYPE    address =
TUPLE   zip code :    integer;
        city :        string;
        ONE OF        ( post-office box :   integer; )
                      ( TUPLE               street :        string;
                                            OPTIONAL        houseno:    integer;
                        END; )
        END;
        SOME OF       ( telephonno :        integer; )
                      ( telexno :           string; )
                      ( email-address :     string; )
        END;
END;
```

Figure 1: Desired representation of an *address* type

would like to express that *some* of these attributes are allowed to be present which is achieved by an *inclusive union*. A third form is a single *optional attribute*. For instance, a street may be accompanied by a house–number, but does not need to. If one puts these heterogeneous structures together with the zip code and the city, that shall always be present, an ideal representation should look like the one depicted in figure 1. There are two possibilities to map the address type onto the simple relational model. The first one is to normalize the conceptual type, i.e. to enumerate each allowed attribute combination, yielding 21 address relations[1]. The second alternative is to specify zip code and city to be *not null* attributes, and to allow null values for the other attributes. Besides the problem that there are several semantics for null values, the connection between the attributes gets completely lost[2]. Both alternatives are dissatisfying, i.e. the operational support of heterogeneous structures requires a proper extension of the relational model.

The structural kernel of our model consists of the definition of a *flexible scheme*, an extension of a relational scheme that allows to specify homogeneous and heterogeneous structures with a single, generic constructor. This aspect will be discussed in section 2. Section 3 deals with a corresponding extension of the relational algebra, putting the emphasis on the point that our algebra is adequate with respect to the structural part of our model, and preserves the consistency with the simple relational algebra at the other side. In section 4 the related aspects *equivalence* among flexible schemes, *restructuring* of flexible schemes, and *structural normal forms* of flexible schemes are discussed. Section 5 contains a comparison of our model to related approaches, especially to the object–oriented paradigm. Section 6 finishes with a summary and an outlook.

2 The model of flexible relations

There is no problem to extend the relational model by one constructor for exclusive disjunctions, another constructor representing inclusive disjunctions, a third constructor for optional attributes, and so on. This approach would not yield a satisfactory solution as

- one can easily imagine application scenarios that demand yet another form of variant structure requiring a fourth constructor with appropriate operators (and so on), plus

- supporting multiple constructors requires a query processing language whose operators are specific to certain constructors, i.e. in such a language it depends on the query context if an operator is applicable or not, destroying both efficiency and ease of use.

The arguments show that supporting variant structures in a relational model raises the challenge to find a *single* constructor that is *complete* in the sense of being able

[1] We assume here that the semantics of the *some of* construct in figure 1 is that at least one of its components has to be present, giving 7 possibilities for the electronic address. Multiplied with the three variants of the inner–town address yields 21 different combinations.

[2] Some approaches including SQL2 ([MS93]) provide the facility to specify relationships between attributes as integrity constraints of the form "(post-office box IS NULL AND street IS NOT NULL) OR ...". But nothing is said about the influence of these integrity constraints on query processing and so on. This aspect is considered by our model providing true integration of variant structures into an operational data model.

to model each possible form of heterogeneity, otherwise the benefits of the relational model get lost.

It is obvious that a simple relational scheme consisting of a set of attributes is too weak to meet these structural requirements. Therefore we added two integer values k_{min} and k_{max} that specify a *range of validity*. Together with the known set $\{ A_1, \ldots, A_k \}$ of attributes, a *choice scheme* forms a three-tuple

$$< k_{min}, k_{max}, \{ A_1, \ldots, A_k \} >$$

The intuitive meaning of a choice scheme is that its domain consists of each tuple that possesses m ($0 \leq k_{min} \leq m \leq k_{max}$) attributes out of $\{ A_1, \ldots, A_k \}$. The semantic type constructs introduced in section 1 are mapped onto a choice scheme as follows

- an exclusive union is represented by $< 1, 1, \{ A_1, \ldots, A_k \} >$, expressing that *at least* 1 and *at most* 1 and therefore exactly one of the attributes $A_1, \ldots, A_k$ has to occur.

- an inclusive union is equivalent to the choice scheme $< 0, k, \{ A_1, \ldots, A_k \} >$, meaning that at least none and at most all and therefore *some* of the attributes $A_1, \ldots, A_k$ may be present. Requiring that at least one of the attributes has to be present leads to the scheme $< 1, k, \{ A_1, \ldots, A_k \} >$.

- a single optional attribute conforms to the choice scheme $< 0, 1, \{ A \} >$.

- a conventional relational scheme does not leave any choice among the attributes and corresponds therefore to the choice scheme $< k, k, \{ A_1, \ldots, A_k \} >$, expressing that each attribute has to be specified.

Putting the characterized semantic constructs together, we are able to represent the address type of figure 1 by the *flexible scheme* depicted in figure 2. It demonstrates that our notation allows to express each involved semantic structure with a single, generic constructor. Note that we enhanced the notion of a choice scheme in figure 2 in that we do not only allow simple attributes to occur in the third component of choice schemes, but also choice schemes again. As unions occur as subtypes of tuples and vice versa, it is obviously necessary to define schemes supporting variant structures in a recursive manner. This recursive step leads to the final definition of a *flexible scheme*[3].

[3] As usual we assume the existence of a countably infinite set $\mathcal{U}$ of attributes. In addition, we expect that there is a set $\mathcal{D}$ of basic domains and a function $dom : \mathcal{U} \longrightarrow \mathcal{D}$ associating a domain with each attribute. As we are not particularly interested in the attribute values, we omit to properly define this relationship.

Definition 1 The set $\mathcal{FS}$ of flexible schemes is the smallest set that satisfies

1. $A \in \mathcal{FS}$, if $A \in \mathcal{U}$
2. $< k_{min}, k_{max}, \{ FS_1, \ldots, FS_k \} > \in \mathcal{FS}$, if $k_{min}, k_{max}, k \in \mathbb{N}$, $0 \leq k_{min} \leq k_{max} \leq k$, $FS_1, \ldots, FS_k \in \mathcal{FS}$

□

Up to now we only have developed a syntactic characterization of flexible schemes. We still have to associate a domain with them. There is a direct (and very efficient) way to determine the tuples forming the domain of a flexible scheme (see [Kal95]), but for a basic understanding it is more suitable to *flatten* a flexible scheme, with the goal to describe its domain as a set of legal attribute combinations. This representation is much like the *Disjunctive Normal Form* (DNF) of propositional logic. Therefore we called the algorithm that computes the flat representation the *dnf*-algorithm. It recursively "multiplies out" the components of a flexible scheme until an equivalent flat enumeration of its attributes is achieved.

As the outcome of the *dnf*-algorithm is very intuitive, we omit a formal definition and present an example instead. The application of the *dnf*-algorithm to the scheme $FS = < 4, 4, \{ A, B, < 1, 1, \{ C, D \} >, < 2, 3, \{ E, F, G \} > \} >$[4] yields the result

$$dnf(FS) = \{ ABCEF, ABCEG, ABCFG, ABCEFG, ABDEF, ABDEG, ABDFG, ABDEFG \}$$

Using the *dnf*-algorithm it is now easy to define the domain of a flexible scheme *FS*: The domain of *FS* consists of each tuple that is defined on an attribute set $X \in dnf(FS)$, i.e. if $Tup(X)$ denotes the set of tuples defined on the attribute set X, then $dom(FS) = \bigcup_{X \in dnf(FS)} Tup(X)$[5].

A second benefit that results from the *dnf*-function is that it is not only possible to map a flexible scheme onto a *dnf*, but that it is also possible to associate *each* DNF of attribute sets with a flexible scheme. Using elementary results of propositional logic we may

[4] For the remainder of the paper we will use the scheme $FS = < 4, 4, \{ A, B, < 1, 1, \{ C, D \} >, < 2, 3, \{ E, F, G \} > \} >$ as the running example. Although it looks very abstract at first glance, it is only slightly modified with respect to the concrete *address* scheme in figure 2. So we emphasize again that flexible schemes do possess practical relevance despite their somewhat abstract notation.

[5] Note that this flat, enumerative description of a flexible scheme corresponds to the "set of objects" approach of [Sci80] (see also [Mai83], ch. 12).

```
address =  < 4 , 4 , {  zip-code :   integer,
                        city :       string,
                        < 1 , 1 , {  post-office-box :  integer,
                                     < 2 , 2 , {        street :     string,
                                                        < 0 , 1 , {  houseno :  integer } >
                                     } >
                        } >,
                        < 1 , 3 , {  telephoneno :   integer,
                                     telexno :       string,
                                     email-addr :    string
                        } >
           } >
```

Figure 2: Representation of the *address* type as a flexible scheme

conclude that flexible schemes are *complete* in the sense that arbitrary heterogeneous structures can be mapped onto a flexible scheme. Summing up, it may be said that flexible schemes satisfy the structural goals of providing a *single*, *complete* constructor for variant structures.

A sample instance of a *flexible relation* based on the flexible scheme $< 4, 4, \{ A, B, < 1, 1, \{ C, D \} >, < 2, 3, \{ E, F, G \} > \} >$ might look as follows:

inst(FR)	A	B	C	D	E	F	G
	a_1	b_1	c_1		e_1	f_1	
	a_2	b_2		d_2		f_2	g_2
	a_3	b_3		d_3	e_3	f_3	g_3
	a_4	b_4	c_4		e_4		g_4
	a_5	b_5	c_5			f_5	g_5
	a_6	b_6		d_6	e_6		g_6
	a_7	b_7		d_7	e_7	f_7	
	a_8	b_8	c_8		e_8	f_8	g_8

3 The F–algebra — an algebra for flexible relations

In the previous section we have shown that flexible schemes provide a compact and generic description of heterogeneous structures. But little is gained by a generic constructor if it is not supported operationally. Therefore we developed the F–algebra, the operational part of the model of flexible relations, with the goal to provide a query processing language as powerful in its ability to process variant structures as flexible relations are on the structural side. The main emphasis in designing the F–algebra was put on

- **adequacy**: each operator should be applicable to any flexible relation with arbitrarily structured flexible schemes.
- **closeness**: each operator should result in a flexible relation again. The emphasis had to be put on the result schemes, which were intended to be well-formed flexible schemes, i.e. a precise and compact description of the operator's output.
- **efficiency**: the operators should not be more complex than their relational counterparts.
- **semantic connection to the relational algebra**: the intuitive meaning of the operators should be kept, and the F–algebra's operators should be *faithful* and *precise*[6] with respect to the relational algebra.

In summary, the F–algebra comprises the relational operators *projection*, *selection* and *cartesian product*, an *extension* operator to add a new column, the set operators *union*, *minus* and *intersection* and a *restriction* operator that checks for the presence or absence of attributes.

The projection operator shall serve as an illustrative example and will be discussed in more detail. The other operators are sketched at the end of this section.

3.1 Projection

The intuitive meaning of a projection is to drop columns. To do so, one specifies a set X of attributes that shall survive the projection. As the result of a projection, a relation containing the attributes $X \cap$

[6] The notions *faithful* and *precise* are adapted from [Mai83]. An extended operator ρ^f is said to be faithful to its basic counterpart ρ if both provide the same result applied to a non–extended relation. An extended operator ρ^f is said to be precise to its basic counterpart ρ if there is a mapping *map* from the extended model $\mathcal{R}^f$ to the basic model such that *map* is a homomorphism for ρ^f and ρ .

$attr(R)$ is produced[7]. The intuition of a projection is directly reflected by the relational definition

$$sch(\pi_X(R)) = X \cap sch(R)$$
$$inst(\pi_X(R)) = \{t[X \cap sch(R)] \mid t \in inst(R)\}$$

One of the design goals of the F–algebra was to preserve the intuitive meaning of the relational operators. Therefore our extended π^f operator has to eliminate columns of a flexible relation in the very same way. This effect is informally presented in figure 3. Note that the projection onto $\{A, C, E, F\}$ does *not* mean that the result tuples are defined on *each* of the specified attributes. As $\{A, B, C, D, E, F, G\}$ is a *superset* of the attributes of the input relation's tuples, $\{A, C, E, F\}$ is a superset of the output relation's tuples consequently, too. This statement leads directly to the definition of $inst(\pi^f_X(R))$:

$$inst(\pi^f_X(FR)) = \{t[X \cap attr(t)] \mid t \in inst(FR)\}$$

To be able to define $sch(\pi^f_X(R))$ analogously to the relational projection one needs a flat description of a flexible scheme. Fortunately, this can be achieved with the *dnf*–function introduced in section 2, that represents a flexible scheme as a set of legal attribute sets. As a relational scheme is merely a set of attributes, the result of the *dnf*–function can be interpreted as a set of relational schemes. Now the analogy to the relational definition can be maintained by defining that all legal attribute sets have to be intersected with the projection attributes:

$$dnf(sch(\pi^f_X(FR))) = \{X \cap Y \mid Y \in dnf(sch(FR))\}$$

The effect of this definition on the example scheme is shown in figure 3. This definition satisfies the fourth design goal, the close connection to the relational algebra, but it fails to satisfy the other three goals. The definition is neither adequate as it does not take a flexible scheme as input but merely a flat description of a flexible scheme, nor is it really closed as the output is again flat and not a compact flexible scheme. Finally, the definition is not efficient as it depends on both the *dnf*– and the dnf^{-1}-function which can be very costly in some cases. Hence the main challenge in the design of the projection operator (and the whole F–algebra) is to find scheme transformations that accept flexible schemes as input and map them directly onto flexible schemes. How this problem was solved is discussed in the next section.

[7] We do not require $X \subseteq attr(R)$ here. This leads to a slightly modified, but equivalent definition and better reflects the connection to the F–algebra.

3.1.1 Direct computation of a projection scheme

It is easier to explain the scheme transformation of the projection if one starts with a single level scheme $sch(FR) = < k_{min}, k_{max}, \{A_1, \ldots, A_k\} >$. Assume that the projection attributes are $X = \{A_1, \ldots, A_m\}$. Due to our arguments in the previous section we know that $\{A_1, \ldots, A_m\}$ will be the superset of the result tuples' attributes, i.e. the result scheme will look like $sch(\pi^f_X(FR)) = < k'_{min}, k'_{max}, \{A_1, \ldots, A_m\} >$. In this scheme the new lower bound k'_{min} and the new upper bound k'_{max} still have to be determined. The new bounds must be chosen such that exactly those tuples satisfying the *inst*–definition are members of the result scheme. The computation of k'_{min} can therefore be regarded as a "worst case analysis": how many attributes does a tuple of the input scheme have *at least* in common with the projection attributes? The way how k'_{min} is determined is depicted in figure 4. The upper attribute "interval" in figure 4 represents the input scheme's attributes, while the second interval shows that the projection attributes are the first m attributes of the input scheme. A tuple being member of the input scheme contains at least k_{min} attributes. To create the worst case we place these k_{min} attributes at "the end" of the input scheme's attributes, i.e. a worst case tuple possesses the attributes $\{A_{k-k_{min}+1}, \ldots, A_k\}$ (see third interval in figure 4). The projection leaves the intersection of the projection attributes and the attributes of the worst case tuple (see lower interval in figure 4). The width of this interval $k_{min} + m - k$ is the new lower bound. As emphasized by the dotted line the value $k_{min} + m - k$ may be less than zero, i.e. the resulting interval may be empty. The correct lower bound of a projection scheme is therefore $k'_{min} = max(k_{min} + m - k, 0)$.

k'_{max} is obtained analogously by a best case analysis. A best case tuple consists of k_{max} attributes located at "the start" of the input scheme's attribute interval to share as many attributes with the projection attributes as possible. The intersection of the best case tuple's attributes with the projection attributes results in the new upper bound $k'_{max} = min(k_{max}, m)$.

The definition for single level schemes is now complete. The generalization to an arbitrary, multiple level scheme $sch(FR) = < k_{min}, k_{max}, \{FS_1, \ldots, FS_k\} >$ is achieved by the following argumentation: In the single level case the attributes $\{A_1, \ldots, A_m\}$ were those that survived the projection. In the multiple level case a subscheme FS_i survives the projection if it contains at

A	B	C	D	E	F	G
a_1	b_1	c_1		e_1	f_1	
a_2	b_2		d_2		f_2	g_2
a_3	b_3		d_3	e_3	f_3	g_3
a_4	b_4	c_4		e_4		g_4
a_5	b_5	c_5			f_5	g_5
a_6	b_6		d_6	e_6		g_6
a_7	b_7		d_7	e_7	f_7	
a_8	b_8	c_8		e_8	f_8	g_8

$\xrightarrow{\pi^f_{\{A,C,E,F\}}(FR)}$

A	C	E	F
a_1	c_1	e_1	f_1
a_2			f_2
a_3		e_3	f_3
a_4	c_4	e_4	
a_5	c_5		f_5
a_6		e_6	
a_7		e_7	f_7
a_8	c_8	e_8	f_8

$$FS = <4, 4, \{ A, B, <1, 1, \{ C, D \}>, <2, 3, \{ E, F, G \}> \}>$$

$$\Downarrow dnf$$

$$dnf(FS) = \{ABCEF, ABCEG, ABCFG, ABCEFG, ABDEF, ABDEG, ABDFG, ABDEFG\}$$

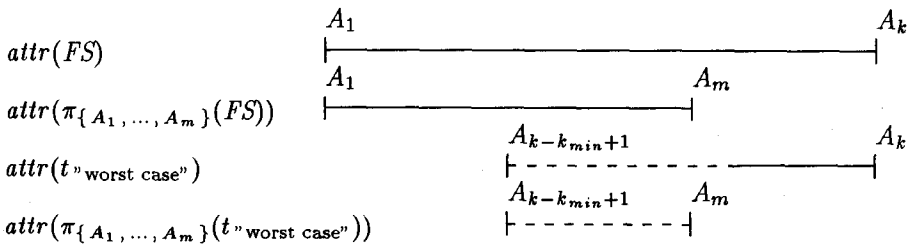
$$\Downarrow \cap \{A, C, E, F\}$$

$$dnf(\pi^f_X(FS)) = \{ACEF, ACE, ACF, AEF, AE, AF\}$$

$$\Downarrow dnf^{-1}$$

$$\pi^f_X(FS) = <3, 3, \{A, <0, 1, \{C\}>, <1, 2, \{E, F\}>\}>$$

Figure 3: example of a projection in the model of flexible relations



$$\Longrightarrow \quad k'_{min} = max(k_{min} + m - k, 0)$$

Figure 4: Computation of the lower bound k'_{min} of a projection scheme

least one of the projection attributes. The number m of surviving subschemes, which influences the computation of k'_{min} and k'_{max} (see above), has now to be chosen such that $attr(FS_i) \cap X \neq \emptyset$ $(i = 1..m)$ and $attr(FS_j) \cap X = \emptyset$ $(j = m+1..k)$. As a surviving subscheme is allowed to possess attributes not contained in the projection attributes, those superfluous attributes have to be removed, too. Therefore the projection operator has to be applied recursively to the subschemes of a multiple level schemes. This argumentation leads to the final definition of the scheme transformation of the projection in the F-algebra:

Definition 2 Let $sch(FR) = < k_{min}, k_{max}, \{ FS_1, \ldots, FS_k \} >$ be a flexible scheme, let X be the set of projection attributes and assume w.l.o.g. that the subschemes of $sch(FR)$ are ordered such that

$$attr(FS_i) \cap X \neq \emptyset \ (i = 1..m)$$
$$attr(FS_j) \cap X = \emptyset \ (j = m+1..k)$$

Then the result scheme of a projection is defined by

$$sch(\pi^f_X(FR)) =$$
$$< k'_{min}, k'_{max}, \{ \pi^f_X(FS_1), \ldots, \pi^f_X(FS_m) \} >$$
$$\text{with } k'_{min} = max(k_{min} + m - k, 0)$$
$$\text{and } k'_{max} = min(k_{max}, m)$$

An atomic scheme $sch(FR) = A$ is transformed by

$$sch(\pi^f_X(FR)) = \begin{cases} A, & \text{if } A \in X \\ \top, & \text{else}^8 \end{cases}$$

□

[8] $\top$ is a special flexible scheme whose domain consists only of the empty tuple <>, i.e. $dom(\top) = \{ <> \}$ and $dnf(\top) = \{\emptyset\}$.

$$\pi^I_X(FS) =$$
$$\pi^I_X(<4,4,\{A,B,<1,1,\{C,D\}>,<2,3,\{E,F,G\}>\}>) =$$
$$<3,3,\{\pi^I_X(A),\pi^I_X(<1,1,\{C,D\}>),\pi^I_X(<2,3,\{E,F,G\}>)\}> =$$
$$<3,3,\{A,<0,1,\{\pi^I_X(C)\}>,<1,2,\{\pi^I_X(E),\pi^I_X(F)\}>\}> =$$
$$<3,3,\{A,<0,1,\{C\}>,<1,2,\{E,F\}>\}>$$

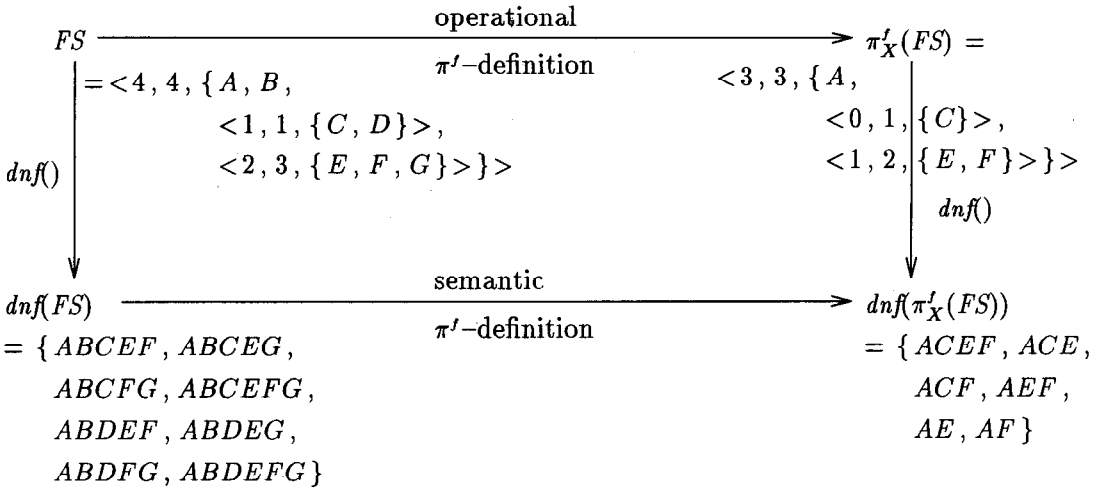


Figure 5: Direct computation of a projection scheme and its relationship to the *dnf*–based computation

The application of this definition to the sample scheme $FS = <4,4,\{A,B,<1,1,\{C,D\}>,<2,3,\{E,F,G\}>\}>$ and the projection attributes $X = \{A,C,E,F\}$ is depicted in figure 5. Now we have two definitions for the scheme transformation of a projection: First, there is a flat description along the *dnf*-function that realizes the connection to the relational algebra. Secondly, we have a recursive definition that works directly on flexible schemes and satisfies the design goals *adequacy*, *closeness* and *efficiency*. But this definition does hardly resemble the relational projection, so we need a connection to the flat definition that reflects the similarity to the relational algebra. This connection is achieved by verifying the commutativity of the diagram in figure 5. In this example it is easy to see that the *dnf*-function serves as a homomorphism between the operational and the semantic definition of a projection, i.e. the diagram commutes. The proof that this equation holds in general is contained in [Kal95].

3.2 Overview of the F–algebra

In summary the F–algebra consists of the basic operators projection, selection, cartesian product, extension, union, minus, intersection, and restriction. Due to lack of space we omit to present the algebra in detail and refer to [Kal95] for the complete definition. To give an impression of the F–algebra we recapitulate the design goals characterized at the start of this section. The semantic connection of the F–algebra to the relational algebra is kept by appropriate *inst*– and *dnf(sch)*–definitions leading to the following theorem [Kal95]:

Theorem 1 The extended operators projection, selection, cartesian product, union, minus, and intersection of the F–algebra are faithful and precise with respect to their counterparts in the relational algebra. □

Of course, there are positive differences to the relational algebra, for example the set operators can be applied to arbitrary input relations that do not have to be "set compatible", i.e. that do not have to possess identical schemes. The adequacy and closeness of the F–algebra with respect to the structural part of our model is achieved by operational scheme transformations applying directly to flexible schemes. The following theorem guarantees the identity of the two scheme transformations for each operator [Kal95]:

Theorem 2 For each operator ρ of the F–algebra, the operational definition of the scheme transformation ρ^{op} (based upon the *sch*–definition) is *correct with respect to* the semantic definition ρ^{sem} (based upon the *dnf(sch)*–definition), i.e. for each flexible scheme FS the equation $dnf(\rho^{op}(FS)) = \rho^{sem}(dnf(FS))$ holds. □

As flexible relations deal with heterogeneous sets the structure–related operators of the F–algebra demand further explanation. The restriction operator ψ checks for the presence $\psi_A(FR)$ or the absence $\psi_{\neg A}(FR)$ of an attribute A in a flexible relation FR. It can be regarded as a formalization of the IS NOT NULL resp. IS NULL test in SQL. The type guard operator $\tau_{FS \rightarrow FS'}(FR)$ takes a flexible relation FR, a subscheme FS of $sch(FR)$, a restricted scheme FS' with $dom(FS') \subset dom(FS)$ and selects those tuples of FR whose FS–part belongs to the domain of FS', i.e. $inst(\tau_{FS \rightarrow FS'}(FR)) = \{ t \mid t \in inst(FR) \wedge t[attr(FS)] \in dom(FS') \}$. Thus the type guard checks for specific structural variants, e.g. $\tau_{AllEmployees \rightarrow Technicians}(Employees)$ extracts the technicians out of an employee relation (with appropriately defined and named schemes). Formally the type guard can be derived from the restriction by forming the DNF of FS', but mapping it onto the membership test of flexible schemes is much more efficient.

With the aid of the type guard we can express that the computation of a property depends upon the actual structural variant, namely by $prop(FR) = \bigcup_i^f expr_i(\tau_{sch(FR) \rightarrow V_i}(FR))$, where $expr_i$ is the implementation of the property $prop$ in the variant V_i. This feature allows us to integrate methods and method overriding in the algebraic processing and in the algebraic optimization step provided that the method implementations $expr_i$ are expressible within the algebra[9]. Further it enables us to strictly separate between flexible relations (i.e. sets of heterogeneous tuples) plus methods at the logical level and implementation issues at the internal level. To compare our approach with the object–oriented paradigm we have developed equivalence transformations for the type guard and we can show that in case of a horizontal fragmentation of a flexible relation into homogeneous sets each type guard can be eliminated resulting in the OO–style expression. This aspect can be sketched as follows: Suppose FR is partitioned at the internal level into homogeneous sets S_i with structure V_i. Then we can put the equation $FR = \bigcup_i^f S_i$ into the expression $prop(FR)$ described above. The first transformation step is to push the type guard into the union, which works as for the selection. Now the type guards are directed against the homogeneous sets directly. All which is now left to do is to identify redundant type guards $\tau_{sch(FR) \rightarrow V_i}(S_i) = S_i$ and unsatisfiable type guards $\tau_{sch(FR) \rightarrow V_i}(S_j) = \perp \ (i \neq j)$ which are eliminated by the rule $FR \cup^f \perp = FR$. These simple transformation rules yield the desired expression $prop(FR) = \bigcup_i^f expr_i(S_i)$ which exploits the horizontal fragmentation. Of course our approach works for vertically fragmented or unfragmented flexible relations, too. Thus the F–algebra provides true data independence for heterogeneous relations and makes method–like computation of properties that depend on structural variants accessible to the optimization component.

To demonstrate the practical relevance of our model we have designed FSQL, an extension of SQL supporting variant structures. Some key features of FSQL are: FSQL contains syntactic constructs to express type guards and restriction and in contrast to SQL it considers the corresponding scheme transformations. FSQL supports methods whose implementation may depend upon structural variants and it offers a generic IF-THEN-ELSE and CASE construct to express dependent computation directly. An application of the generic CASE construct is the *multi–way join* joining a master relation with different dependent relations guarded by a specified condition. This operator which is the key aspect of [AB91] comes for free in our model. To reverse the heterogeneity of flexible relations FSQL offers default expressions that replace missing attributes, thus filling up the gaps in a flexible relation.

As FSQL is based upon the F–algebra and its derived operators we may conclude that our model provides user–friendly access to heterogeneous relations with a theoretically sound foundation.

4 Restructuring and normalizing flexible schemes

The question if two schemes are equivalent is an important question in scenarios like view construction, schema simplification, database integration and many others. The formal basis of the notions *information capacity, schema dominance* and *schema equivalence* are presented in [Hul86] and [AABM82]. While simple relational schemes are equivalent only if they are identical [Hul86], the same does not hold for more complex data models. In the context of the FORMAT MODEL ([HY84]), which supports tuple and set constructors in arbitrary order, it was shown that structurally different schemes may be equivalent and *restructuring rules* were presented that map schemes onto equivalent ones. In [AH88] a data model that extends the Format Model by an exclusive union constructor was considered and the rules were extended to capture it. A major result of [AH88] is that the restructuring rules transform schemes of this model into an unambigious normal form.

One of the restructuring rules of [AH88] is depicted in figure 6. It states that a tuple constructor (represented

[9] Regarding this aspect it is advisable to consider domain operations in the algebra as for example [Güt89] does.

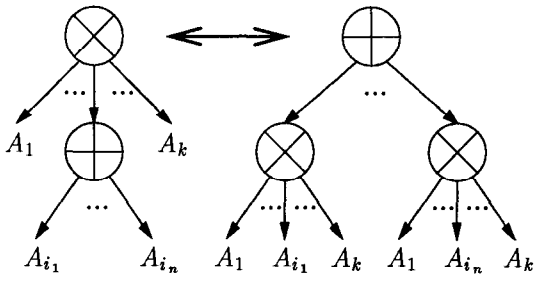


Figure 6: Example of a restructuring rule

by $\bigotimes$) with k components whose i-th component is a union constructor (represented by $\bigoplus$) with n components is equivalent to a union constructor with n components, each component being the tuple constructor of the left scheme having the i-th component replaced by a component of the union constructor.

The problem of having equivalent, but syntactically different schemes applies to flexible relations, too. An example of schema equivalence in our model is depicted in figure 7 that contains three flexible schemes that do all describe the same domain, i.e. that are equivalent. The first scheme states in a compact way that 2 up to 3 of the attributes $\{A, B, C, D\}$ have to be present. The second scheme splits this information by saying that either exactly 2 or exactly 3 of the mentioned attributes may occur. The third scheme is special in the way that it consists of a *complete enumeration* of all valid attribute combinations. In that form both the k_{min}– and the k_{max}–value can be statically determined: $k_{min} = k_{max} = 1$ at the top level, $k_{min_i} = k_{max_i} = card(subscheme_i)$ at the second level. Such a flexible scheme does therefore bear exactly the information computed by the *dnf*–algorithm and we will call those schemes being in *disjunctive normal form* (DNF).

This third scheme corresponds to the normal form developed in [AH88], i.e. the DNF is the result of the transformations described in [AH88] when these transformations are applied to types consisting only of tuple and union constructors and not possessing any set constructor. Besides the advantages of the DNF, that is being a normal form and being easy to compute, it has several disadvantages:

- The storage costs of DNF schemes are exponential in the number of attributes. The time complexity of comparing two DNF schemes is exponential[10], too.
- For arbitrary flexible schemes the membership test is in NP relative to the input size. Measured in the number of attributes (for a better comparison with the relational model) the membership test and other algorithms on flexible schemes including the scheme transformations of the algebra have exponential time complexity for DNF schemes.

[10] Usually processing costs are measured in the size n of the input yielding $n \log n$ for the comparison of two schemes in DNF. But our intention is to compare our model with the relational one. Therefore we have chosen the number of attributes as the common basis.

4.1 Minimal normal form of flexible schemes

The disadvantages of the DNF can be avoided if one utilizes the capability of flexible schemes to express heterogeneous structures in a compact way (as the first scheme in figure 7 does). Hence arises the need to transform flexible schemes into a dense form with as few schema nodes as possible. Obviously, the way one has to pursue is to apply the restructuring rules in a direction that yields "smaller" schemes with less schema nodes, e.g. from right to left in figure 6. The challenge of reducing flexible schemes is to answer the question if applying the rules towards smaller schemes always leads to an unambigious normal form.
As one can show this cannot be achieved for arbitrary flexible schemes. That is, there are schemes for which applying the restructuring rules in different orders leads to different final schemes. The crucial point whether there exists a *minimal normal form* is the number of occurences of attributes. There is a subclass of $\mathcal{FS}$ that we have called the class $\mathcal{DFS}$ of *disjoint flexible schemes*. It contains all flexible schemes that possess each attribute at most once, e.g. the first scheme in figure 7 belongs to $\mathcal{DFS}$, while the other two schemes in figure 7 possess multiple occurences of attributes and belong hence to $\mathcal{NDFS} = \mathcal{FS} - \mathcal{DFS}$. At first glance one might think that members of $\mathcal{DFS}$ are normalized by definition. A counterexample is the scheme $FS = <3, 3, \{A, B, <2, 2, \{C, D\}>\}>$. FS is a member of $\mathcal{DFS}$, but it can be reduced to $FS' = <4, 4, \{A, B, C, D\}>$. This reduction is achieved by the *tuple in tuple* restructuring rule described in [Kal95]. In summary, [Kal95] contains seven restructuring rules applying to schemes in $\mathcal{DFS}$ and it can be shown ([Sch94],[Kal95]) that reducing members of $\mathcal{DFS}$ by applying the rules in arbitrary order yields an unambigious *minimal normal form.*

The second important property of $\mathcal{DFS}$ is that the time complexity of the relevant algorithms, including the membership test, is polynomial in the number of attributes. The result is due to the fact that a scheme in $\mathcal{DFS}$ with n attributes possesses $o(n)$ schema nodes and that the most costly node-local operation is the intersection of two attribute sets which is $o(n \log n)$.

$$
\begin{aligned}
&<2,3,\{A,B,C,D\}> \\
\equiv &<1,1,\{<2,2,\{A,B,C,D\}>,<3,3,\{A,B,C,D\}>\}> \\
\equiv &<1,1,\{<2,2,\{A,B\}>,<2,2,\{A,C\}>,<2,2,\{A,D\}>,<2,2,\{B,C\}> \\
&\quad <2,2,\{B,D\}>,<2,2,\{C,D\}>,<3,3,\{A,B,C\}>, \\
&\quad <3,3,\{A,B,D\}>,<3,3,\{A,C,D\}>,<3,3,\{B,C,D\}>\}>
\end{aligned}
$$

Figure 7: Equivalent flexible schemes

It would be a pleasant result if each flexible scheme could be brought into minimal normal form. Unfortunately this is not the case. First, not every flexible scheme possesses a representative in $\mathcal{DFS}$[11] , e.g. there is no member of $\mathcal{DFS}$ whose *dnf* is $\{A, AB, BC\}$. For schemes in $\mathcal{NDFS}$ that do not have a representative in $\mathcal{DFS}$ one can easily show that they do *not* possess a minimal normal form. There are restructuring rules applying to members of $\mathcal{NDFS}$, like the rule depicted in figure 6, but the output of the reduction process is not unique and depends upon the order in which the rules are applied[12].

The seven restructuring rules mentioned above leading to the minimal normal form apply only to schemes that are already members of $\mathcal{DFS}$. Hence it remains the question how members of $\mathcal{NDFS}$ can be brought into $\mathcal{DFS}$ if they possess a representative in $\mathcal{DFS}$. We have developed an algorithm dnf^{-1} that takes a flat description of a flexible scheme and decides if there exists a representative in $\mathcal{DFS}$ possessing the specified DNF. A complete description of the algorithm can be found in [Kal95]. It is not surprising that the algorithm is very expensive in extreme cases as it involves numerous tests on common subexpressions. Fortunately the dnf^{-1}–algorithm is needed rarely as the algebra is closed in $\mathcal{DFS}$ except the union and the intersection operator. The remaining problem can be attacked with the restructuring rules applying to members of $\mathcal{NDFS}$. These restructuring rules do often apply in practical cases, and serve therefore as low cost shorthands so that the expensive dnf^{-1}–algorithm must rarely be employed.

[11] Fortunatley, the class $\mathcal{DFS}$ is sufficiently large as there are more than $n!$ members of $\mathcal{DFS}$ with up to n attributes. Nevertheless, the cardinality of $\mathcal{NDFS}$ is larger, namely 2^{2^n}, telling that there exist members of $\mathcal{NDFS}$ that do not possess an equivalent member of $\mathcal{DFS}$.

[12] An illustrative example is the scheme FS with $dnf(FS) = \{AB, AC, AD, BC, BD, BE\}$. There are three structurally different minimal representations of FS, each consisting of 13 nodes, allowing the conclusion that there is no minimal *normal form* in $\mathcal{NDFS}$.

5 Comparison to related approaches

Some attempts have been made to support variant structures in relational data models ([AB91], [DC89]). These approaches share the problem that the notion of a precise, statically typed scheme is abandoned and that the proposed query languages either do work only interactively when variant parts are touched ([DC89]) or are only able to produce query results that are more variant than the input leading quickly to useless diversified "patchwork" ([AB91]). A more competitive approach is the data model of the LILOG-DB project ([Lud90], [BGL+91]) where connections between attributes may be specified with the boolean operators *and*, *or* and *not*, i.e. in a form "address ... (HAS street AND HAS house–number AND NOT HAS post-office box) OR (HAS post-office box AND ..." The major disadvantage of LILOG-DB is that the EFTA algebra defined upon their model ([LW91]) does not take the structure information of the input schemes into account. In contrast, it works on arbitrarely typed heterogeneous sets. Therefore it is not evident what the effort of precisely specifying schemes serves for.

5.1 Comparison to object–oriented data models

Formal object-oriented models ([SLR+93], [LRV88], [EA91]) typically define the domain of a type τ to be the set of all objects with type $\tau' \leq \tau$, with $\leq$ being an appropriate subtype relationship ([CW85]). Therefore an object-oriented type is never an arbitrary polymorphic set, but always a complete lattice.

Let us first demonstrate that flexible schemes are capable of simulating object–oriented types: Let τ be a tuple–valued type consisting of the attributes $\{A_1, \ldots A_k\}$ and assume that the set $\{A_1, \ldots A_n\}$ of relevant attributes is known. Then the lattice under τ conforms to the scheme $FS = < k+1, k+1, \{A_1, \ldots A_k, <0, n-k, \{A_{k+1}, \ldots A_n\}>\}>$[13],

[13] Even if the set of relevant attributes is unknown or varies over time, the simulation does not fail: Each novel attribute A_{n+1}

i.e. a flexible scheme that must contain the attributes $\{A_1, \ldots A_k\}$ and may possess any subset of $\{A_{k+1}, \ldots A_n\}$. So a lattice can be represented as a simple two–level flexible scheme, and it is a rather easy task to transform object–oriented algebras onto our model.

Our model is not behaviourally object–oriented, i.e. inheritance or other dynamic aspects are not supported. Nevertheless, flexible relations meet the structural requirements to place a behaviourally object–oriented model on top of them. From the structural point of view the key property to support dynamic aspects is subtyping, and of course we are able to represent the different notions of subtyping [BW90] in our model: Object–oriented subtyping and subset subtyping at the tuple level of two flexible schemes FS_1 and FS_2 can be expressed by appropriate relationships between $dnf(FS_1)$ and $dnf(FS_2)$[14]. Subset subtyping at the level of attribute domains is achieved in our model by a new form of integrity constraint which we have called *attribute dependency*. Details about attribute dependencies can be found in [KD95].

In addition to being able of simulating OO features, our model exceeds the OO approach in its modeling capabilities, as a flexible scheme allows to restrict the domain to *meaningful* types. The disadvantage of the lattice approach continues in query processing. For example, the result type of the union operator has to be defined as the least upper bound of the input types to be consistent with the lattice properties ([BK89], [SLR+93]). One can easily define a type $\tau \leq lub(\tau_1, \tau_2)$ for which neither $\tau \leq \tau_1$ nor $\tau \leq \tau_2$ holds, i.e. $dom(\tau_1 \cup^{oo} \tau_2) \supseteq dom(\tau_1) \cup dom(\tau_2)$, while our algebra assures that $dom(\tau_1 \cup^{f} \tau_2) = dom(\tau_1) \cup dom(\tau_2)$ holds. This causes problems in OO data models, e.g. with updatable views, that we do not have. As a last point both our approach and OO data models require type guards ([BO91]) before variant parts may be accessed. As shown in section 3.2 we are capable of identifying redundant or unsatisfiable type guards at compile time, while OO models have to check most of them at run time, as they are not able to disallow non–occuring subtypes. Further optimization potential similar to *qualified relations* [CP83] stems in our model from attribute dependencies and is discussed in [KD95].

can be added to FS by "upgrading" FS to the scheme $<k+1, k+1, \{A_1, \ldots A_k, <0, n-k+1, \{A_{k+1}, \ldots A_{n+1}\}>\}>$. Such an upgrade does not invalidate the existing member tuples of the scheme, so it can be done at any time.

[14] Subset subtyping $FS_1 \leq^{s} FS_2$ holds when $dnf(FS_1) \subseteq dnf(FS_2)$, and object–oriented subtyping $FS_1 \leq^{oo} FS_2$ holds when $\forall X \in dnf(FS_1)\ \exists Y \in dnf(FS_2) : X \supseteq Y$.

6 Summary

In this paper we have introduced the basic concepts of the model of *flexible relations* that improves the relational model on providing modeling and operational support of variant structures. The central feature of the model is the notion of *flexible schemes* that allow to model homogeneous and heterogeneous structures in a uniform way. Flexible schemes combine the relational paradigm of providing a single constructor with the aspect of completeness, i.e. each heterogeneous structure is expressible as a flexible scheme.

The F–algebra defined on flexible relations has been presented and its *adequacy* with respect to the structural part of the model and the *semantic relationship* to the relational algebra have been shown. We have outlined how advanced operators that process variant structures in a concise and elegant way can be derived from the basic ones. These derived operators can be regarded as the intermediate step towards FSQL, a high–level language providing user–friendly access to flexible relations. Due to lack of space the user–friendly interface to flexible relations could only be sketched but section 3.2 gave an impression of how method–like processing, generic IF–THEN–ELSE and CASE expressions, and default values are supported by FSQL and based upon the F–algebra and its derived operators.

The need of a more powerful scheme mechanism compared to a simple relational scheme led to the problem that equivalence among flexible schemes cannot be decided in structural, syntactical terms. A disjunctive normal form on flexible schemes could be derived from previous results ([AH88]). Problems of the DNF, especially its processing costs, were identified and the condition, under which a very cheap *minimal normal form* is available, were determined. These investigations led to the subclass $\mathcal{DFS}$ of those flexible schemes, in which each attribute occurs at most once. An interesting open question is if a minimal normal form exists if flexible relations are extended by a set constructor, i.e. if our model is joined with the NF^2 data model [AFS89].

Besides the aspect that structural reduction of flexible schemes leads exactly in $\mathcal{DFS}$ to the minimal normal form, one can show that relevant algorithms on flexible relations, like membership test and the algebraic operators, possess polynomial complexity (measured in the number of attributes) in $\mathcal{DFS}$, while these algorithms have an exponential worst case complexity for general flexible schemes. These results allow the conclusion that the simple relational model can be extended to the much larger class $\mathcal{DFS}$ without losing any positive property of the simple relational model.

Finally, we compared our model with other approaches supporting heterogeneous structures, especially to the object–oriented paradigm. We did show that flexible relations are able to simulate object–oriented classes, and outperform the OO approach in modeling and typing precision and provide a higher potential of query optimization.

Acknowledgements

We would like to thank the anonymous referees for their valuable suggestions.

References

[AABM82] P. Atzeni, G. Ausiello, C. Batini, and M. Moscarini. *Inclusion and equivalence between relational database schemata.* Theoretical Computer Science 19, S. 267–285 (1982).

[AB91] R. Ahad and A. Basu. *ESQL: A Query Language for the Relation Model supporting Image Domains.* In [ICDE91], pp. 550–559.

[AFS89] S. Abiteboul, P.C. Fischer, and H.-J. Schek, editors. *Nested Relations and Complex Objects in Databases*, Darmstadt, Deutschland (1989). Springer-Verlag, LNCS 361.

[AH88] S. Abiteboul and R. Hull. *Restructuring Hierarchical Database Objects.* Theoretical Computer Science 62, S. 3–38 (1988).

[BGL+91] S. Benzschawel, E. Gehlen, M. Ley, T. Ludwig, A. Maier, and B. Walter. *LILOG–DB: Database Support for Knowledge Based Systems.* In Text Understanding in LILOG, pp. 501–594. LNAI 546, Springer–Verlag (1991).

[BK89] F. Bancilhon and S. Khoshafian. *A Calculus for Complex Objects.* Journal of Computer and System Sciences 38(2), S. 326–340 (Apr. 1989).

[BO91] P. Buneman and A. Ohori. *A Type System that reconciles Classes and Extents.* In Proc. 3rd Int'l Workshop on Database Programming Languages, pp. 175–186, Nafplion, Greece (Aug. 1991).

[BW90] K. Bruce and P. Wegner. *An Algebraic Model of Subtype and Inheritance.* In F. Bancilhon and P. Buneman, editors, Advances in Database Programming Languages, pp. 75–96. ACM Press Frontier Series (1990).

[Cod70] E.F. Codd. *A Relational Model of Data for Large Shared Data Banks.* Communications of the ACM 13(6), S. 377–387 (June 1970).

[CP83] S. Ceri and G. Pelagatti. *Correctness of Query Execution Strategies in Distributed Databases.* ACM Transactions on Database Systems 8(4), S. 577–607 (Dec. 1983).

[CW85] L. Cardelli and P. Wegner. *On Understanding Types, Data Abstractions, and Polymorphism.* ACM Computing Surveys 17(4), S. 471–522 (Dec. 1985).

[DC89] H. Dreizen and S.-K. Chang. *Imprecise Schema: A Rationale for Relations with Embedded Subrelations.* ACM Transactions on Database Systems 14(4), S. 447–479 (Dec. 1989).

[EA91] D. Eichmann and D. Alton. *A Polymorphic Relational Algebra and Its Optimization.* In [ICDE91], pp. 680–689.

[Güt89] R.H. Güting. *GRAL: An Extensible Relational Database System for Geometric Applications.* In Proc. 15th Int'l Conf. on Very Large Databases, pp. 33–44, Amsterdam, Netherlands (Aug. 1989).

[HK87] R. Hull and R. King. *Semantic Database Modeling: Survey, Applicatiions and Research Issues.* ACM Computing Surveys 19(3), S. 201–260 (Sept. 1987).

[Hul86] R. Hull. *Relative Information Capacity of Simple Relational Database Schemata.* SIAM Journal of Computing 15(3), S. 856–886 (Aug. 1986).

[HY84] R. Hull and C.K. Yap. *The Format Model: A Theory of Database Organization.* Journal of the ACM 31(3), S. 518–537 (July 1984).

[ICDE91] *Proc. 7th Int'l Conf. on Data Engineering*, Kobe, Japan (Apr. 1991).

[Kal95] C. Kalus. *Supporting Variant Structures in a Generalized Relational Data Model (in German).* Phd thesis in preparation, University of Ulm (1995).

[KD95] C. Kalus and P. Dadam. *Record Subtyping in Flexible Relations by means of Attribute Dependencies.* In Proc. 11th Int'l Conf. on Data Engineering, Taipeh, Taiwan (Mar. 1995).

[Ken79] W. Kent. *Limitations of Record-Based Information Models.* ACM Transactions on Database Systems 4(1), S. 107–131 (Mar. 1979).

[LRV88] C. Lecluse, P. Richard, and F. Velez. *O_2, an Object-Oriented Data Model.* In Proc. ACM SIGMOD International Conference on Management of Data, pp. 424–433, Chicago, Illinois (June 1988).

[Lud90] T. Ludwig. *A Brief Overview of LILOG-DB.* In Proc. 6th Int'l Conf. on Data Engineering, pp. 420–427, Los Angeles, California (Feb. 1990).

[LW91] T. Ludwig and B. Walter. *EFTA: a database retrieval algebra for feature–terms.* Data & Knowledge Engineering 6(2), S. 125–149 (Mar. 1991).

[Mai83] D. Maier. *The Theory of Relational Databases.* Computer Science Press (1983).

[MS93] J. Melton and A.R. Simon. *Understanding the new SQL: a complete guide.* Morgan Kaufmnn Publishers (1993).

[PM88] J. Peckham and F. Maryanski. *Semantic Data Models.* ACM Computing Surveys 20(3), S. 153–189 (Sept. 1988).

[Sch94] C. Schiekel. *Reduction and equivalence of generalized relational schemes – theoretical foundations and algorithmic solutions (in German).* Master's thesis, University of Ulm (July 1994).

[Sci80] E. Sciore. *The Universal Instance and Database Design.* Phd thesis, Princeton Univ., Princeton, NJ (1980).

[SLR+93] M. Scholl, C. Laasch, C. Rich, H.-J. Schek, and M. Tresch. *The COCOON Object Model.* Technical Report 93-02, University of Ulm (Feb. 1993).