Ersch. in: Information technology: advancement, productivity and international cooperation; conference proc.; 3. Pan Pacific Computer Conference, Beijing, August 16 - 19, 1989. Band 1 / Chen Liwei, Chen Bingcong, Sun Qiangnan (eds.). - Beijing : Internat. Acad. Publ. [u.a.], 1989. - S. 498-504. - ISBN 7-8000-3073-3

Some Strange Ways to Use the NF^2 Relational Model

Bin Jiang and Marc H. Scholl
Department of Computer Science
Swiss Federal Institute of Technology Zurich
CH-8092 Zurich, Switzerland
E-mail: {jiang|mscholl}@ifi.ethz.ch

Abstract

The NF^2 relational model has been widely accepted as one of the extensions of Codd's relational model which better meet the requirements imposed by the so-called non-standard database applications. The manners in which the NF^2 relational model has been used so far are called standard ways of using the model. In this paper some non-standard ways to use the NF^2 relation model are presented. Their common characteristics are analyzed and the concept of "object motors" is introduced for new generation of computer systems.

1 Introduction

Because of its simplicity, Codd's relational model [9] has got great success for supporting business applications. Generally in these areas, a data unit is a simple tuple in a certain table and the fields (attributes) of the tuple do not possess internal structure, which meets exactly the first normal form (FNF) of the model.

But also because of its simplicity, Codd's model cannot sufficiently meet the requirements imposed by the so-called non-standard database applications, such as office automation systems, geographical information systems, engineering information systems supporting CAD/CAM, and knowledge base systems for AI (artificial intelligence) applications and research. The objects to be managed in these applications are not simple [2, 4, 8, 10, 12, 13, 18, 35].

Firstly, they may have complex internal structure. Each object may consist hierarchically of several subobjects which consist of, in turn, several subsubobjects and so forth. Besides, one object may be subobject of one or more other objects. Such objects are also known as complex objects (CO) [5, 11, 24, 25, 33].

Secondly, the relationship between them may be complex. Similar objects (w.r.t. properties or structure) may be regarded as an object class. A class may have one or more superclasses and/or subclasses. The subclasses may inherit behaviors and proporties from their superclasses. Some similar classes may, in turn, build a meta-class. Furthermore, objects may communicate with each other. Among others, this scenario is often called object-orientation (O_2) [3, 7, 17, 36].

Thirdly, they may be structurally too different from each other and too dynamic to have a few static schemas as in standard applications (D_2).

In order to model such objects, especially complex objects (CO), quite a lot of effort has been spent to extend Codd's relational model [1, 16, 20, 19, 27, 15] and[21, 33]. One of the widely accepted models is the so-called NF^2 (Non-First-Normal-Form) relational model, which is also called nested relational model (on the one hand, negation is generally unpleasant, on the other hand, negation is often unsafe in database terminology. But the term NF^2 is more pretty). The most important merit of the model is that attributes of a tuple in a relation could be relations themselves, so-called subrelations or relation-valued attributes.

The NF^2 relational model has been proposed not only as a logical DB model for modeling tables whose fields themselves could be tables [23], but also as a framework for physical database design even for "flat" relational DB's to improve the DBMS performance by materializing joins [34]. A characteristic feature of the applications for which the model has been designed is that many objects have the same or similar structure, i.e. they possess the same schemas. This is essentially the same way in which Codd's relational model has been used so far: corresponding to each structure type there is a table of tuples which have the same structure. We call this way standard way of using the NF^2 relational model in this paper. However, can one also model objects which are very different from each other and have an arbitrarily deep hierarchical or even recursive structure? It needs some non-standard ways, or strange ways to use the model. The objective of the paper is to present some solutions for these non-standard modelling problem using the NF^2 relations and to stimulate some discussions.

The organization of the paper is as follows. In section 2 the NF^2 model is presented and the modelling limitations are pointed out. Some strange ways to use NF^2 relations are described in section 3 through examples from the knowledge representation area. Section 4 is devoted to analyzing the essential merits of the ways, from which the new concept "object motors" (the most strange way) is introduced. In section 5, one of the object motors, i.e. Horn-clause motor is presented as an example to sketch the basic ideas.

Name	FN	Pno	City	Room	Phone
Deppisch	U	1234	F	24005	5377
Horn	D	2345	HG	11301	5421
Jiang	B	3456	DA	11304	5420
Obermeit	V	4567	DA	24005	5377
Paul	B	5678	MZ	24014	2863
Scholl	M	6789	MTK	24012	5378
Waterfeld	W	7890	DA	11307	5420
Weikum	G	8901	MTK	24013	5379
Wolf	A	9012	MTK	11301	5421

Figure 1: A Simple Table and the Corresponding Relation

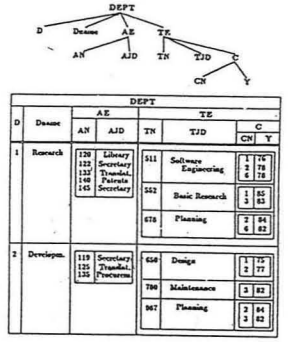

Figure 2: An NF^2 Relational Schema Tree and the Relation (Table)

2 Standard Ways of Using the NF^2 Relational Model

In order to manage simple tables which are useful in commercial applications, Codd and others have developed the (normalized) relational data model. In a database, each relation corresponds to a table, each tuple of a relation corresponds to a record (an object) in the corresponding table, and each attribute of a tuple of a relation corresponds to a field of the corresponding record in the table. Because most fields of the records are simple (elementary), the attributes in the relational model are atomic (without internal structure). Corresponding to the template of a table which determines the form of the records in the table, there is something called relation schema which determines the form of the tuples in the relation. These excellent correspondence is the reason why the Codd's relational model has obtained such a great success in business applications. Here is an example:

Although there are some tables which are somehow more complex, they are usually partitioned into several smaller tables and the tuples in the corresponding relations are related to each others by references (or foreign keys).

But in the so-called non-standard applications, objects are generally complex. Objects could be hierarchically structured. For example, a company consists of several departments. Each department (as an object) has its administrational and technical staff which attended some courses. Of course, such objects could also be modeled with Codd's "flat" relational model, i.e. in first normal form. But as pointed out by Bancilhon et al. [4], there are some problems:

- The most expensive operations, i.e. joins, are necessary to reconstruct hierarchical objects from scattered parts in different relations;
- Artificial identifiers must be introduced to partition real world objects;
- Objects with missing values must be handled specially; and
- It is generally very difficult for users to rationally partition or display hierarchical objects.

Among others, the NF^2 relational model has been introduced to address these issues. An NF^2 relation can be considered as a table of complex records (tuples). Each tuple can contain not only "atomic" attributes, but also attributes which have internal structure, i.e. they are relations themselves (the so-called subrelations). Each tuple of a subrelation (subtuple) may again have several atomic attributes and "sub-"subrelations and so on. However, all tuples in a relation, just as in Codd's relational model, obey the same schema. That means, each tuple in the relation has the same numbers of atomic attributes and "relation-valued" attributes, respectively. A subrelation in the relation schema again has a unique subschema in all tuples, and so on. In the following figure information about the company is stored in an NF^2 relation, as an example.

The major characteristics of the ways (or applications) in which the relational models, not noly NF^2 but also "flat", have been used so far is that all objects for a relation have the same structure, i.e. the same schema. That means, the number and the type of the attributes (atomic and relation-valued) are the same for tuples or subtuples in the corresponding position. For applications where objects have same or similar structure, the NF^2 model has

gained quite a lot of attention [23]. However, in many non-standard applications, especially for AI, objects may be much more complex than that: they are very different from each other (without similar structure), they are nested arbitrarily deep, they may share subobjects and so on. Examples include representations of Horn-clauses, or of a set of frames.

It seems as if the NF^2 relational model is not appropriate to treat such objects, because in the ways the model is used till now, there would be as many relations as objects, which is impractical. Some non-standard or strange ways must be found. Some of those will be sketched in the following section.

3 Non-Standard Ways of Using the NF^2 Relational Model

3.1 Mapping Horn-Clauses to NF^2 relations

Generally, a Horn-clause is of the following form:

$$P_0(t_{01},...,t_{0n}) \leftarrow P_1(t_{11},...,t_{1r}) \wedge ... \wedge P_m(t_{m1},...,t_{ms})$$

where $P_i, 0 \leq i \leq m$ are predicate symbols with respective arities. P_0 is the "head" predicate of the clause and $P_1,...,P_n$ are the "body" predicates. The t_{ij}'s are terms which are defined as follows:

- A constant is a term;
- A variable is a term;
- An n-ary function symbol with n terms as parameters is a term;
- Nothing else is a term.

Assume that we have a set of Horn-clauses. Although each predicate or function symbol has a certain arity, each term might have arbitrarily deep hierarchy. Besides this, each body of a clause might contain an arbitrary number of predicates. Obviously, there is few structural similarity between the clauses which is, however, the very prerequirement to store the objects in a few relations in the conventional manner. But in a strange way, we can use the NF^2 model to store an **arbitrary** set of clauses with **arbitrarily** deep hierarchy and an **arbitrary** number of branches even in one relation! The following is a mapping from Horn-clauses to an NF^2 relation:

Each tuple of **H-CLAUSE** stands for a Horn-clause in the set. The atomic attribute HP-Name is the name of the head predicate. The subrelation **Prmtrs1** contains parameters of the head predicate. Each subtuple of **Prmtrs1** has only one atomic attribute which holds the address of the parameter (term) as a subtuple in the subrelation **Terms**. The

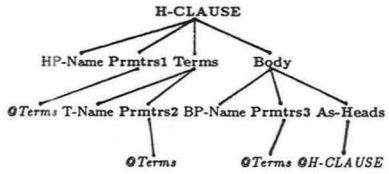

Figure 3: NF^2 Relation Schema for Horn-Clauses

subrelation **Terms** contains all terms of the clause, each one as a single subtuple. The atomic attribute T-Name is the name of the term which might hold either a function name, or a variable or a constant if the term is not a function. The sub-subrelation **Prmtrs2** contains the parameters of the term if the term is a function, otherwise it is empty. The sub-subtuples of **Prmtrs2** are, in turn, the addresses of the parameters in the subrelation **Terms** (arbitrarily deep hierarchical recursion!). Each subtuple in the subrelation **Body** stands for a predicate in the body of the clause. The atomic attribute BP-Name is the name of the predicate and the subrelation **Prmtrs3** contains parameters of the predicate in a way similar to **Prmtrs1** and **Prmtrs2**. The additional subrelation **As-Heads** contains the addresses of the clauses in the relation **H-CLAUSE** where this predicate is used as head predicate. Since the subrelation **Body** can contain arbitrary many subtuples, we can store any Horn-clause in our relation.

3.2 Mapping Frames to NF^2 relations

Quite probably because M. Minsky [26] had defined the concept "frames" not precisely enough, with intention or not, it has made lots of people dream about it, pleasant or heavy [6, 14, 29, 30, 31, 32, 37]. We imagine our frames simply as follows.

Each frame has a collection of slots which could be either atomic slots, or other frames (frame slots). The values of each slot might be controlled by various conditions. Each frame may occur as a value in one or more other frames but not within itself, neither directly nor indirectly.

A set of similar frames can form a frame-class with some common properties. A frame-class may have one or more super-classes from which it can inherit some properties. Each frame-class can also be subdivided into one or more subclasses, for which it is then the super-class. All frames together are called a frames space.

Obviously, it is impossible to map such a frames space to an NF^2 relational database in the standard manner. But in a strange way, it does work. The following is a mapping (here we use only a very simplified version, the complete frames space, and its

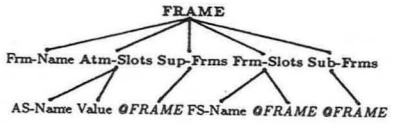

Figure 4: Mapping Frames to an NF^2 Relation

mapping to NF^2 relational databases will be precisely discussed in a forthcoming report):

Each tuple in the relation **FRAME** presents a frame with name Frm-Name which can be used as an identifier of the frame. In the subrelation **Atm-Slots**, each subtuple stands for an atomic slot with AS-Name for its name and Value for its value. The subrelation **Frm-Slots** contains all slots of the frame (each as a subtuple) which again are frames. FS-Name is the name of the frame slot and *@FRAME* is the address where the slot frame is stored as another frame (tuple) in the relation **FRAME**. The subrelation **Sup-Frms** comprises all addresses of the tuples in **FRAME** which stand for the so-called *class frames* from which this frame inherits properties. If the frame is an *object frame*, something like instance, the subrelation **Sub-Frms** is empty.

Both, class and object frames, are stored within this relation. In the case of a class frame a tuple in **FRAME** stands for a class frame with Frm-Name as its name. The subrelation **Atm-Slots** (resp. **Frm-Slots**) contains all atomic (resp. frame) slots which are common for all object frames belonging to this frame class. The subrelation **Sup-Frms** (resp. **Sub-Frms**) holds the addresses of the class frames (as tuples in the relation **FRAME**) which are superframe classes (resp. subframe classes) of the actual frame class. From the superframe classes the actual frame class inherits all porperties, i.e. slots, and to the subframe classes it propagates its properties.

Obviously, this is a very very brief sketch of the whole frame story and there is no real system which uses such a simple frame concept. But here we did not want to give a complete discusssion of a complex frame concept. It was more important to present the basic ideas. In the next section we analyze this approach in further detail.

4 The Most Strange Way to Use the NF^2 Relational Model

4.1 Analyzing the Basic Ideas of the Mappings

In the last section we have seen that it is possible to store a set of very different objects in one single relation. The basic ideas are:

- Every "attribute" in the usual sense is treated as a subtuple with two atomic attributes of a certain subrelation. One attribute is the name of the "attribute", and the other one holds its value. The number of the attributes in a relation is fixed but the number of the subtuples of a subrelation is not. Thus by transforming all "attributes" in the usual sense to subtuples in a certain subrelation, we eliminate the restriction of a fixed number and name of attributes in a relation schema. In other, more abstract words, we solve the problems on the Y-axis with solutions on the X-axis. Examples are the subrelations **Prmtrs**i $(0 \leq i \leq 2)$, **Terms**, **Body** in the relation **H-CLAUSE** and all subrelations in the relation **FRAME**.
- All hierarchical structures are represented by several tuples or subtuples of the same relation or subrelation in the way that each tuple or subtuple describes only one layer of the hierarchy. The depth of the hierarchy in a relation (schema) is fixed, but the number of tuples or subtuples is not. Thus by transforming the hierarchical object into several tuples or subtuples, we eliminate the limitation of a fixed depth of the hierarchy of objects in a relation. That is, we solve problems on the Z-axis with solutions on the X-axis. The excellent X-axis!
- Because each tuple or subtuple presents only one layer of an object or subobject, it is somehow autonomous. This implies that it may be "shared" between several superobjects. So our Y-axis has solved the "sharing"-problem too, by transfoming network structure into a hierarchical one.

Generally, we can map any object modeled with any known data model to one NF^2 relation using this approach.

4.2 The Most Strange Way to Use Something: Throw it Away and birth of Object Motors

Earlier, we analyzed data objects from a structural point of view and according to the *structural similarity*, they were grouped into several classes. For each class we extracted an object schema for the relation where the objects would be stored. But now, the objects become more and more different from each other so that we would have as many schemas as objects. Naturally, this does not work. However, if we observe the *meta-structural similarity* of all object, we suddenly note that we can now pack the entire world (our object space) into one relation! From chaos to order, from one extreme to another.

If we can map the entire world into only one NF^2 relation, that is, there is only one schema on earth, do we need the schema at all?! If we would not need the schema, what would we need the whole NF^2 relational data model for?! Is it not astonishing that we have gone from using the NF^2 model to solve some problems and it has worked very well, but all of a sudden, the answer is that we could throw the NF^2 model away and it works even better! From order to chaos, from one extreme to another once again!

On the other hand, if we have only one schema for all objects, we could code all information about the one and only schema, which is not much at all, into the program, or even more, we could implement it in hardware. So we could have a very compact, efficient mechanism to handle all objects (at least in a branch). The resulting implementation could be called an **OBJECT MOTOR**!

For management of our knowledge in any areas with a computer, we must select an appropriate framework to represent it, e.g. predicate logic, or frames, or production systems, or one of the semantic networks, just as we have to select a certain programming language to code our algorithms, e.g. Algol, or Fortran, or Pascal, or C, or Smalltalk. Though for some tasks, predicate logic is more appropiate than frames to represent the knowledge or conversely, there are hardly any significant syntactical differences between predicate logic (or frames) for medicine and for machine production, just as there are hardly any significant syntactical differences in the programming language C in medicine and in machine production. Therefore, for each framework with a strict, widely accepted definition we can make a motor, e.g. a "frame motor", a "Horn-clause motor", or a "concept motor" for KL-ONE [7]. If we want to represent our knowledge in one of the frameworks, we can use the respective motor to manage and manipulate it. In the next section we sketch a "Horn-clause motor" as an example.

5 One of the Object Motors: Horn-Clause Motor

In subsection 3.1 we gave a general definition of Horn-clauses. This definition is widely accepted independent from the knowledge areas where Horn-clauses are used. Through the so-called *NF^2-analysis* we have obtained a NF^2 schema of Horn-clauses in Figure 3. According to the NF^2 schema, the physical layout of every tuple (Horn-clause) is fixed.

For instance, before HP-Name, T-Name, BP-Name one byte is used respectively for giving the length of the names (as string) and before **Prmtrs***i* ($0 \leq i \leq 2$), **Terms**, **Body** and **As-Heads** one byte is used respectively for giving the number of the subtuples in the subrelations. The lengths of *@Terms* and *@H-CLAUSE* are fixed, e.g. two and four bytes, respectively. Now, one clause physically looks as follows:

LHPN HP-Name NP1 *@Terms ... @Terms*
NT LTN T-Name NP2 *@Terms ... @Terms ...*
LTN T-Name NP2 *@Terms ... @Terms*
NB LBPN BP-Name NP3 *@Terms ... @Terms*
HASH *@H-CLAUSE ... @H-CLAUSE ...*
LBPN BP-Name *@Terms ... @Terms*
HASH *@H-CLAUSE ... @H-CLAUSE*

where LHPN, LTN's and LBPN's give the lengths of the following names respectively and NP1, NP2's, NP3's, NT, NB and NASH's give the numbers of subtuples of the following subrelations, respectively.

We call such a string an entry. Since the inner structure of the entry is fixed, each field of the entry can be addressed and interpreted in a fixed manner. For example, we always get the address of the first T-Name as follows:

$$Add + 1 + LHPN + 1 + NP1 \times 2 + 1 + 1$$

where *Add* is the address of the entry.

Our Horn-clause motor is built on such a fixed structure and it can interpret every field of every entry in a fixed manner. In other words, the users can apply the motor on entries to use or manipulate them as if they would use a microscope on things invisible. Now, it is not longer neccessary to look up describtions in catalogs since the motor knows everything in advance. Therefore, it works very fast and it is also obvious that the representation utilizes the storage very efficiently.

For a certain framework of the knowledge representation, some operations are typical and important while others are not. For instance, with Horn-clauses it is more typical to access a body predicate with all parameters, or all addresses of the clauses where the body predicate is head predicate, than to get all @H-CLAUSE's within a certain interval. Such typical operations can be coded into the motor as integrated standard operations in addition to the elementary operations.

With such a mechanism, users write down their knowledge not according to some relation schemas but according to the grammar (syntax) of the respective framework used for the representation of the knowledge. The input, e.g. Horn-clauses, is syntactically tested (if no syntax-oriented editor has been used) and then stored in the database. For using the knowledge, the entries are fetched from the database and then the object motor can use the entries.

Finally, it is worth pointing out that all conventional database technologies at the page level can still be applied. For example in the extensible DBMS "DASDBS" [28], the CRM (complex record manager) is replaced by an object motor and the SSM (stable storage manager) can still be used by the object motor. A complete design of an object motor will be presented in a forthcoming report.

6 Conclusion

To meet the properties of objects in non-standard database applications, i.e.

$$CO_2 + D_2$$

the paper has studied some ways in which the NF^2 relational model can be used. First, the standard ways using flat and NF^2 relations were presented and the limitations in the treatment of complex objects were pointed out. Then with two examples, i.e. Horn-clauses and frames, the non-standard ways to use the NF^2 model have been demonstrated. After that, the essential ideas and techniques were described, the essence of the approach was pointed out and the concept of *object motors* was introduced. Finally, the Horn-clause motor, one of the object motors, was decribed to demonstrate the basic ideas.

Note that this approach is suggested to apply only for object spaces where the objects are very different from each other, e.g. in the AI area. Other issues like access to such objects is discussed in [22], mapping a query language to such internal representations etc. will be discussed in future.

7 Aknowledgement

Inspiration by Prof. Dr. Hans-Jörg Schek is gratefully acknowledged. The authers would like also to thank Dagmar Horn and Andreas Wolf who provided helpful discussions and comments on an early version of the paper.

References

[1] S. Abiteboul and N. Bidoit. Non first normal form relations to represent hierarchically organized data. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 191–200, 1984.

[2] H. Afsarmanesh, D. McLeod, D. Knapp, and A. Parker. An extensible object-oriented approach to databases for VLSI/CAD. In *Proceedings of the International Conference on Very Large Databases*, pages 13–24, 1985.

[3] F. Bancilhon. A logic-programming/object-oriented cocktail. *ACM-SIGMOD Record*, 15(3):11–21, September 1986.

[4] F. Bancilhon, T. Briggs, S. Khoshafian, and P. Valduriez. FAD, a powerful and simple database language. In *Proceedings of the International Conference on Very Large Databases*, pages 97–107, 1987.

[5] F. Bancilhon and S. Khoshafian. A calculus for complex objects. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 53–59, 1986.

[6] D. G. Bobrow, R. M. Kaplan, M. Kay, D. A. Norman, H. Thompson, and T. Winograd. GUS, a fram-driven dialog system. *Artificial Intelligence*, 8:155–173, 1977.

[7] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216, 1985.

[8] M. J. Carey, D. J. DeWitt, D. Frank, G. Graefe, M. Muralikrishna, J. E. Richardson, and E. J. Shekita. The architecture of EXODUS extensible DBMS. In A. Buchmann, U. Dayal, and K. Dittrich, editors, *Object-Oriented Database Systems, Topics in Information Systems*. Springer-Verlag, 1987.

[9] E. F. Codd. A relational model for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970.

[10] P. Dadam, K. Küspert, F. Andersen, H. Blanken, R. Erbe, J. Günauer, V. Lum, P. Pistor, and G. Walch. A DBMS prototype to support extended NF^2 relations: An integrated view on flat tables and hicrarchies. In *Proc. ACM SIGMOD Conf. on Management of Data*, pages 356–366, Washington, 1986. ACM, New York.

[11] U. Dayal, F. Manola, A. Buchmann, U. Chakravarthy, D. Goldhirsch, S. Heiler, JH. Orenstein, and A. Rosenthal. Simplifying complex objects: The PROBE approach to modelling and querying them. In *Proceedings of the GI Conference on Database Systems for Office Automation, Engineering, and Scientific Applications*, 1987.

[12] U. Dayal and J. M. Smith. PROBE: A knowledge-oriented database management system. In *On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies*, pages 227–258. Spring-Verlag, 1985.

[13] K. R. Dittrich. Object-oriented database systems: The notion and the issues. In *Proceedings of the International Workshop on Object-Oriented Database Systems*, 1986.

[14] R. Fikes and T. Kehler. The role of frame-based representation in reasoning. *Communications of the ACM*, 28(9):904–920, 1985.

[15] P. C. Fischer and S. J. Thomas. Operators for non-first-normal-form relations. In *Proc. IEEE Computer Software and Applications Conf.*, pages 464–475, 1983.

[16] A. L. Furtado and L. Kerschberg. An algebra of quotient relations. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1977.

[17] A. Goldberg and D. Robson. *Smalltalk 80: The Language and its Implementation.* Addison-Wesley, 1983.

[18] T. Häder, K. Meyer-Wegener, B. Mitschang, and A. Sikeler. PRIMA: a DBMS prototype supporting engineering applications. In *Proceedings of the International Conference on Very Large Databases*, pages 433–442, 1987.

[19] R. Hull and C. K. Yap. The format model: A theory of database organization. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 205–211, 1982.

[20] B. E. Jacobs. On database logic. *Journal of the ACM*, 29(2):333–362, April 1982.

[21] G. Jaeschke and H.-J. Schek. Remarks on the algebra of non-first-normal-form relations. In *Proc. ACM SIGACT/SIGMOD Symp. on Principles of Database Systems*, pages 124–138, Los Angeles, March 1982. ACM, New York.

[22] B. Jiang. Text signature supporting retrieval of complex objects, 1987. Technical Note: DVS1-1987-A1.

[23] A. Kanaegami. KAPPA - knowledge base management system on PSI, 1986.

[24] R. Lorie, W. Kim, D. McNabb, W. Plouffe, and A. Meier. Supporting complex objects in a relational system for engineering databases. In W. Kim, D. S. Reiner, and D. S. Batory, editors, *Query Processing in Database Systems.* Springer, 1985.

[25] V. Lum, P. Dadam, R. Erbe, J. Guenauer, P. Pistor, G. Walch, H. Werner, and J. Woodfill. Design of an integrated DBMS to support advanced applications. In *Proceedings of the International Conference on Foundations of Data Organization*, 1985.

[26] M. Minsky. A framework for representing knowledge. In P. H. Winston, editor, *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, 1975.

[27] Z. M. Ozsoyoglu and L. Y. Yuan. A normal form for nested relations. In *Proc. ACM SIGACT/SIGMOD Symp. on Principles of Database Systems*, pages 251–260, Portland, March 1985. ACM, New York.

[28] H.-B. Paul, H.-J. Schek, M. H. Scholl, G. Weikum, and U. Deppisch. Architecture and implementation of the Darmstadt database kernel system. In *Proc. ACM SIGMOD Conf. on Management of Data*, San Francisco, 1987. ACM, New York.

[29] U. Reimer and U. Hahn. On formal semantic properties of a frame data model. *Computers and Artificial Intelligence*, 4(4):335–351, 1985.

[30] U. Reimer and H.-J. Schek. A frame-based knowledge representation model and its mapping to nested relations, 1988. Submitted for Publication.

[31] E. Rich. *Artificial Intelligence.* McGraw-Hill, 1983.

[32] N. Roussopoulos. CSDL: A conceptual schema definition language for the design of data base applications. *IEEE Transactions on Software Engineering*, 5(5):481–496, September 1979.

[33] H.-J. Schek and M. H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):137–147, June 1986.

[34] M. H. Scholl, H.-B. Paul, and H.-J. Schek. Supporting flat relations by a nested relational kernel. In *Proc. Int. Conf. on Very Large Databases*, pages 137–146, Brighton, September 1987. Morgan Kaufmann, Los Altos, CA.

[35] M. Stonebraker. Object management in POSTGRES using procedures. In A. Buchmann, U. Dayal, and K. Dittrich, editors, *Object-Oriented Database Systems, Topics in Information Systems.* Springer-Verlag, 1987.

[36] P. Wegner. Perspectives on object-oriented programming. Lecture Note, 1987.

[37] P. H. Winston. *Artificial Intelligence.* Addison-Wesley, 1977.