QUERY LANGUAGES FOR NESTED RELATIONAL DATABASES

Henry F. Korth and Mark A. Roth

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-87-45 December 1987

Query Languages for Nested Relational Databases

Henry F. Korth*
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

Mark A. Roth
Department of Electrical and Computer Engineering
Air Force Institute of Technology (AFIT/ENG)
Wright-Patterson AFB, OH 45433

December 2, 1987

Abstract

The nested relational model has proven useful in modeling databases of complex objects. In this paper we consider query languages designed specifically to exploit the power of this model. First, formal query languages are considered: a relational calculus defining the desired power of nested relational languages, and a relational algebra that provides a procedural language suitable for query optimization. Next, two higher-level languages are discussed and compared, SQL/NF, and Heidelberg Data Base Language (HDBL). Two extensions of these languages are considered. X-SQL/NF is a role-join extended version of SQL/NF that incorporates an ISA hierarchy into the semantics of the language. A recursive version of HDBL allows the definition of a transitive closure operation on nested relations.

1 Introduction

In recent years, there has been a growing interest in extending the relational model so as to make it applicable to a wider range of applications. One of the more important of these extensions is the *nested relational model*, also known as the NF2 or non-1NF (both standing for "not first normal form") model.

The nested relational model eliminates the first normal form constraint that all domains must be treated as atomic. Specifically, all domains that are not atomic are treated as relation-valued domains. The result is a nesting of relations within relations. The additional modeling power made available by nesting allows the representation of higher-level user interfaces as well as lower-level implementation details. At the high level, "objects" that have set-valued fields (e.g., the author set for a book, or the showing times for a movie) can be represented in a single tuple rather than several tuples. The correspondence between a tuple and an entity in the user view leads to a more natural interface. At the lower-level, such implementation techniques as clustering or repeating fields can be represented using the formalism of the nested relational model. This added power is most useful in applications of database in office information systems, computer-aided design, and knowledge bases, where one frequently encounters "complex objects," that is, entities which contain other entities.

This paper focuses on the user-level aspects of the nested relational approach. We discuss several query languages for nested relational database. We begin by considering formal query languages. The first, an algebraic language, provides us with a set of operations powerful enough to express queries, yet simple enough

*Research partially supported by NSF Grant DCR-8507724

for significant optimization. The second, a predicate calculus style language, serves to define the desired power of a nested relational language. These two languages do not have user-friendliness as an objective. Rather, the definition of an algebra equivalent to the calculus defines a procedural language of the requisite power to serve as the foundation for more user-oriented languages. The remaining languages we discuss are targeted at either application programmers or end-users.

Many of the user-oriented languages we discuss are based on the SQL language. SQL has gained wide acceptance in both industry and academia as a standard relational query language. By incorporating nested-relational concepts into SQL, several researchers have show that these concepts are readily applicable in a practical query language. In fact, the incorporation of nesting into SQL has allowed several "ugly" features of standard SQL to be eliminated (or kept only for the purpose of downward compatibility).

Although the SQL language is designed to be an end-user language, it is really most applicable for application programming. We illustrate with an example how a nested version of SQL facilitates the development of a forms-management interface. Later, we discuss an extended of a nested SQL to allow direct representation of generalization and specialization within the language. An entity may serve in several *roles* (for example, person, student, employee, manager, parent, child) and may have different attributes pertaining to each of the roles. By extending nested relational languages to include generalization and specialization, some role information can be added to the relational semantics, resulting in a higher-level language that places fewer demands on users. The representation of generalization and specialization is a major feature of object-oriented systems. The incorporation of this capability into SQL/NF is a step in bridging the gap between relational systems and object-oriented systems.

2 Formal Query Languages

This section summarizes recent work in the area of formal query languages for the nested relational model. Initial work focused on formulating an extended relational algebra and more recently on extending the relational calculus.

2.1 Relational Algebras

There has been a large amount of work on extending relational algebra for the nested relational model. Among these are [1,2,7,8,9,10,11,12,16,18,19,26,29,32,33]. The basic extensions made to the relational algebra consist of the following operators:

1. the classical relational operators extended to nested relations: union ($\cup$), difference ($-$), selection (σ), projection (π), and cartesian product ($\times$);
2. two restructuring operators: nest (ν) and unnest (μ).

The unnest operator transforms a relation into one which is less deeply nested by concatenating each tuple in the relation being unnested to the remaining attributes in the relation. The nest operator creates partitions based on the formation of equivalence classes. Two tuples are equivalent if they have the same values for the attributes which are not being nested. For each equivalence class a single tuple is placed into the result. The attributes being nested are formed into a nested relation containing all tuples in the equivalence class for those attributes.

All other extended operators which have been proposed can be expressed in terms of these basic operators. Van Gucht [32] showed that this basic algebra is complete in the sense of Bancilhon and Paradeans [4,21]. We mention also some of the more important extended relational algebra operators and their properties.

Structure preserving operators Since nested relations can be structured to preserve multivalued dependencies which are implicit in their scheme of nesting [20], operators which preserve these properties are useful. Extensions for union, intersection, difference, projection, and join [2,27] and nest [33] have been proposed.

Nested selection In the context of the VERSO model [5], [2] propose an extended selection operator which is capable of performing selections simultaneously from the relation and its nested components.

Statistical operators Operators to support statistical databases in the SSDB system [17] have similar functionality to operators for nested relational databases.

Finally, we note that two proposals have been made for a recursive algebra [11,30] for the nested relational model. In these algebras, operators may recursively appear within selection predicates and projection attribute lists. These algebras allow complex queries to be expressed much more naturally than the non-recursive algebras. They also show greater promise as an intermediate language for query translation and optimization [24], but do provide any more expressive power [10].

2.2 Relational Calculi

The expressive power of a query language can best be shown in the form of a non-procedural (declarative) language based on the predicate calculus. Initial work on nested relational languages, however, focused on algebraic (procedural) languages. It is surprising that a calculus for the nested relational model was not developed until several years after the model's introduction.

The first proposal for an extended relational calculus was made by Roth [26]. This work proposed an extended calculus which included only two basic extensions to the original relational calculus. These were two new atoms, one to specify an "element of" relationship and another to allow nested calculus expressions within simpler atoms. In addition, several semantic constraints on the types of extended calculus expressions were made to eliminate "unsafe" expressions, that is, those that produced infinite or powerset relations, or took infinite time to compute. With these restrictions, the extended algebra and extended calculus were shown to be equivalent in expressive power.

Another approach was taken in [18] by considering both set-valued attributes and aggregate functions. This approach considered nesting relations one level deep. The methodology of Klug [13] was used to prove the equivalence of an algebra and a calculus for a model with one-level nesting and aggregate functions.

3 SQL-like Query Languages

This section provides an overview of several language proposals designed to support user access to the nested relational model. For real-world users of a database system the terse algebra and calculus languages are too difficult to use. To satisfy these users, several "syntactically sugared" query languages have been defined and used in existing relational database management systems. Here we concentrate on extensions and adaptations of the SQL language, which is rapidly becoming an industry standard for relational databases. First we present a basic extension to the language, called SQL/NF [25], which operates on nested relations in a manner orthogonal to their design. Next we present a language proposal developed for the Advanced Information Management Prototype (AIM-P) [22,23,28] which includes additional data types within relations and a SQL-like language to support them. Finally, we present two extensions to this previous work, the incorporation of roles for attributes [24], and the use of recursive queries [14].

Before we begin, let us first introduce a sample application which we can use to illustrate the queries of the proposed languages. This example, a forms processing database application drawn from [31], is representative of the kind of advanced application for which the nested relational model was envisioned.

We will use four relations of a distribution company's database: cust-info, vendor-info, product, and stock. Figure 1 shows the attributes and nesting scheme of these relations. Note, how there is exactly one tuple associated with each entity; one tuple in cust-info for each customer, one tuple in vendor-info for each vendor, and so on. A traditional database implementation would require many more relations to store this information, distributing each entities data among several tuples in several relations.

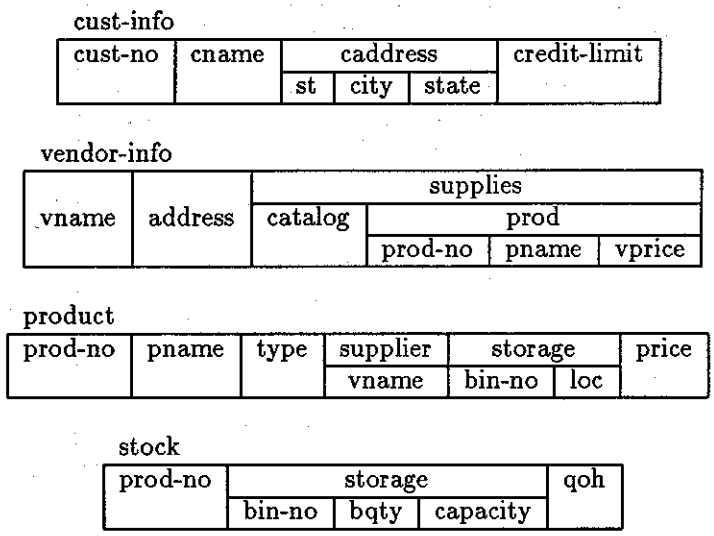

Figure 1: Relation headers for example distribution company

3.1 SQL/NF

In SQL, a basic query conforms to the structure

```
SELECT attribute-list
FROM   relation-list
WHERE  predicate
```

This SFW-expression can be conceptually executed by forming the cartesian product of all relations in the *relation-list*, choosing only tuples in this product that satisfy the *predicate*, and then choosing only those attributes in the *attribute-list*. If no qualification of tuples is needed then the `WHERE` clause can be omitted. In traditional databases, each relation is comprised strictly of scalar values. In nested relational databases, each relation may be comprised of other relations as well as scalar values. The principle of orthogonality has been usefully employed in defining the nested relational data structure. Wherever a scalar value could occur in a traditional relation, a relation can now occur. This simple transformation is now also employed in the SQL/NF language. SQL has the *closure* property where the result of any query on one or more relations is itself a relation. The principle of orthogonality suggests that we should allow an SFW-expression wherever a relation name could exist. Thus SQL/NF allows SFW-expressions in the `FROM` clause and in the `SELECT` clause. Relations have always appeared in the `FROM` clause of a query and so the substitution of a SFW-expression for a relation name is merely a convenience. However, the change to the `SELECT` clause is vital so that nested relations may be accessed with the full power of SQL.

Consider, for example, the query: Retrieve product number and bin numbers for products in stock which have one or more empty bins.

A solution is

```
SELECT prod-no, SELECT bin-no
                FROM   storage
FROM   stock
WHERE  <?,0,?> IN storage
```

The outer SFW-expression retrieves tuples from stock where a tuple in storage has a bqty value of 0. The "?" symbols are don't care values which match any value in the relation. The nested SFW-expression

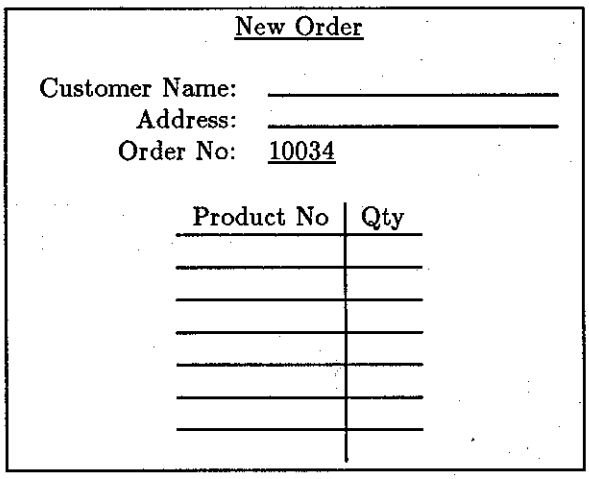

Figure 2: Form for placing a new order

selects only the bin-no attribute from the nested storage relation. If only those bin numbers for the empty bins are desired then a **WHERE** clause may be added to the nested SFW-expression as follows.

```
SELECT prod-no, SELECT bin-no
                FROM   storage
                WHERE  bqty = 0
FROM   stock
WHERE  <?,0,?> IN storage
```

Furthermore, since the same condition is now being tested in both the outer and nested SFW-expression, the query can be simplified by giving the newly derived nested relation a name, and referring to that name in the outer expression's **WHERE** clause. The following example uses the **EXISTS** predicate which yields true if its argument contains tuples and false if its argument is empty.

```
SELECT prod-no, (SELECT bin-no
                 FROM   storage
                 WHERE  bqty = 0) AS bins
FROM   stock
WHERE  EXISTS bins
```

We now show the utility of nested relations for storing and retrieving "forms" in the database. The distribution company takes orders for products from customers. A form is brought up on the screen as in Figure 2. The following steps take place in filling in the form. We assume that SQL/NF is embedded in a programming language which is controlling the form application. Program variables are indicated with a preceding dollar sign.

1. Today's date is automatically filled in from the system clock, and the next available order number is filled in.

2. The user enter's the customer name in the CNAME field, and a query is made to look up the other customer information for verification:

   ```
   SELECT caddress into $CADDRESS
   ```

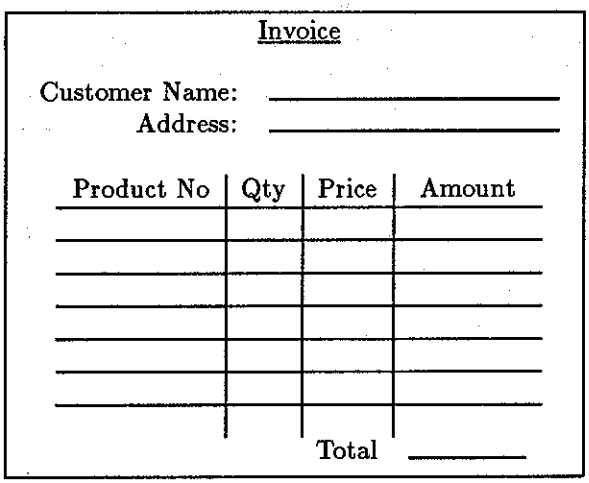

Figure 3: Form for an invoice

```
FROM    cust-info
WHERE   cname = $CNAME
```

The form would allow stepping through the query if more than one tuple was retrieved.

3. Product numbers and quantities would then be entered in the PROD fields. Each product would be checked against the product relation to verify the correct product was being ordered and against the stock relation to see if enough quantity was on hand to satisfy this order.

4. The customer number, order number, date, and product information would then be stored in a *neworder* relation. Each tuple in neworder corresponds to exactly one order placed using this form.

Among several actions that could take place at this point, let us assume that the order can be filled and now an invoice needs to be generated (see Figure 3). A report writer can get the information it needs with the following query, producing a relation with a tuple corresponding to each invoice.

```
SELECT cname, caddress,
       (SELECT prod-no, qty, price, (price*qty) AS amount
        FROM prod, product
        WHERE prod.prod-no = product.prod-no) AS orders,
       (SUM(SELECT amount FROM orders)) AS total
FROM   neworder, cust-info
WHERE  neworder.cust-no = cust-info.cust-no
```

Of course, there are many more actions which need to be accomplished even with the simple example we have used above. The key point was to illustrate the naturalness and ease which nested relations and the SQL/NF language gives to a non-traditional data processing problem, like forms management.

The design of SQL/NF attempts also to eliminate some arbitrary restrictions of SQL and to simplify the syntax where possible. Examples of this include allowing query expressions to appear in the **FROM** clause (already allowed in an indirect way through the view mechanism) and a simple, more powerful specification of aggregate functions. In addition, the **GROUP BY** mechanism can be superseded by the **NEST** operation and the ability to use SFW-expressions in the **SELECT** clause, further simplifying the set of constructs the user needs to deal with. We illustrate the later with an example.

The role information is stated in X-SQL/NF using the **define roles** statement:

define roles
mname ISA *ename*
ename ISA *name*
dname ISA *name*
dssn ISA *ssn*

Although the semantics of a **define roles** declaration is simple, the overloading of the dot operator leads to semantic complication. X-SQL/NF allows a dot expression to appear anywhere an attribute name could have appeared in SQL/NF. The expression $A.B$ could mean

1. The B field of a tuple on relation A.
2. The relation B which is nested within a tuple of relation A.
3. The B property of the A field of a tuple (on a relation defined implicitly).

It could be argued that this complication is undesirable from a user point-of-view. However, it is argued in [24] that the overloading of the dot operator is actually a semantic simplification. Consider, for example, the expression *manager.salary*. There could be a *manager* relation with a salary attribute (case 1 above). However, it is also possible that *salary* is an attribute appearing within a nested subrelation of the *manager* relation, since languages such as SQL/NF allow users to omit the name of the nested subrelation of *manager* if no ambiguity results (case 2 above). Finally, it is possible that *manager* must be cast in its role as *employee* and that another relation, *department(employee, salary)* must be used to obtain the salary of the manager (case 3 above). Which of the three cases applies does not change the user-level semantics of the expression. The user seeks the salary of the person (who happens to be a manager). The manner in which the system obtains this information depends on the underlying database scheme, but it can be argued that these distinctions are not important to the user. The important semantic issue is that "manager's salary" means the amount of money paid to the manager and not, for example, the salary of all employees under the manager. Although the semantic definition can be made only implicitly in X-SQL/NF, we are no worse off in this regard than in a standard relational language.

Although the dot operator may not appear overloaded to the user, there is a severe overloading of the operator from the standpoint of operational semantics. The subtleties of parsing dot expressions and generating a translation into the nested relational algebra are discussed in [24].

We now illustrate the expression of queries in X-SQL/NF with a few examples from our sample database as defined above.

Example 1: Find the salary of the manager of the toy department.

```
SELECT mname.salary
FROM   employee
WHERE  dname = "toy"
```

The expression *mname.salary* appears to be ambiguous since salary is an attribute of a nested subrelation of *manager*. However, the fully-qualified name for *salary* is *manager.employee.salary*. The latter dot expression does not contain *mname*. Thus, the role-join semantics of *mname.salary* can be deduced unambiguously by the query processor.

Example 2: Suppose that a dependent with a social security number of 787-12-1188 has submitted a medical form. Find the social security number of the employee with this dependent.

```
SELECT ename.ssn
FROM   employee
WHERE  EXISTS <?,787-12-1188> IN dependent
```

station	{destinations}
Frankfurt	Stuttgart, Munich
Stuttgart	Munich, Frankfurt
Munich	Frankfurt, Stuttgart, Salzburg
Salzburg	Munich, Wien
Wien	Salzburg

Figure 5: Train relation [14].

In this example, the *ssn* attribute does not appear with a relation appearing in the **FROM** clause, nor in a nested subrelation. X-SQL/NF requires only that leftmost names in a dot expression appear in the **FROM** clause, nor in a nested subrelation. The linkage to the *person* relation is determined by the role information given in the **define roles** section. Note that in this example, as in the one above, no ambiguity results.

3.5 An Extension for Recursive Relations

In this section, we review Linnemann's proposal [14] to extend HDBL with a recursive query mechanism. This proposal hopes to further bridge the gap between database systems and advanced applications with particular emphasis on knowledge-based systems. We illustrate the approach with some examples drawn from [14].

Example 1: Consider a train connection database, showing for each station a set of destination station that are directly reachable. An example relation is shown in Figure 5.

HDBL is powerful enough to get cities which can be reached a fixed number of connections away, but the problem of computing all change connections cannot be solved without a recursive mechanism. A proposed solution is as follows:

```
SELECT [station: r.station,
        destinations: r.destinations UNION
                      (SELECT d
                       FROM   t IN trains, d IN destinations
                       WHERE  EXISTS c IN destinations:
                                  c = t.station)
       ]
FROM   r IN trains
```

The expression for destinations is an equation with left hand side destination. The solution of the equation can be computed by a loop which keeps adding cities to destinations (by executing the inner SFW-expression) until there is no change to destinations. For the relation in Figure 5 the result is shown in Figure 6.

The corresponding query in the classical relational model (extended for recursion) would be more difficult to construct because of the lack of the nested set attribute. This is cumbersome for the user as well as for optimization of recursive queries.

4 Conclusion

It is interesting to review the features of nested relational query languages to see how well they meet the needs of the new database applications that originally motivated the research.

We have seen by an example that the nested relational model lends itself directly to the support of application-programmer interfaces for the design of forms management systems. Sets arise naturally in such

station	{destinations}
Frankfurt	Stuttgart, Munich, Frankfurt, Salzburg, Wien
Stuttgart	Munich, Frankfurt, Stuttgart, Salzburg
Munich	Frankfurt, Stuttgart, Salzburg, Munich, Wien
Salzburg	Munich, Wien, Frankfurt, Stuttgart, Salzburg
Wien	Salzburg, Munich, Wien, Stuttgart, Frankfurt

Figure 6: Result of recursive query on train [14].

contexts. By relieving the programmer from much of the complexity of set operations, a nested relational interface facilitates the development of new forms and the modification of existing forms.

A difficulty with the use of SQL/NF, or any relational language for that matter, within a programming language interface, is the fact that typical programming languages are record-oriented, while relation languages are set-oriented. Various "fourth generation" languages have attempted to solve this problem for the non-nested relational model, usually by means of iterative constructs. Recursive HDBL can be viewed as the nested-relational equivalent of a fourth generation language. It achieves its power through recursion. Recursion is a natural means of accessing recursively-nested relational databases and offers the same computing power as the introduction of iteration. The incorporation of recursion into nested relational languages makes them competitive with logic query languages such as Datalog.

The role extension of SQL/NF is a step towards the incorporation of the class-subclass semantics of object-oriented databases within a nested relational framework. Still lacking within the nested relational model is a means of representing executable code within the data model. Such a representation would serve as an analog to the *methods* of the object paradigm.

The design of nested relational query languages remains an area of active research. The languages presented in this survey illustrate the potential of the nested relational model, though much work remains to be done. One of the most pressing items on the research agenda is an extension of the theory of relational query processing to nested relations. Because of the ability of nested relations to represent both high-level user interfaces and low-level implementation strategies, such a theory could have important practical application. The issue of query optimization is especially interesting for the recursive variants of nested relational languages. Recent research on logic query processing may be influential in solving this problem.

References

[1] Serge Abiteboul and Nicole Bidoit. *Non First Normal Form Relations: An Algebra Allowing Data Restructuring.* Technical Report 347, Institut National de Recherche en Informatique et en Automatique, Rocquencort, B.P. 105, 78153 Le Chesnay Cedex, France, 1984.

[2] Serge Abiteboul and Nicole Bidoit. Non first normal form relations to represent hierarchically organized data. In *Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Waterloo*, pages 191–200, April 1984.

[3] F. Anderson, V. Linnemann, P. Pistor, and N. Südkamp. *Advanced Information Managament Prototype (AIM-P) – User Manual of the On-line Interface of the Heidelberg Data Base Language (HDBL)*

Prototype Implementation (Release 1.1). Technical Note TN86.01, Heidelberg Scientific Center, IBM Germany, 1986.

[4] François Bancilhon. On the completeness of query languages for relational data bases. In *Proceedings of the 7th Symposium on Mathematical Foundations of Computer Science, Zakopane, Poland (Lecture Notes in Computer Science)*, Springer-Verlag, September 1978.

[5] François Bancilhon, D. Fortin, S. Gamerman, J. M. Laubin, P. Richard, Michel Scholl, D. Tusera, and A. Verroust. VERSO: A relational backend database machine. In David K. Hsiao, editor, *Advanced Database Machine Architecture*, pages 1–18, Prentice-Hall, 1983.

[6] Catriel Beeri and Henry F. Korth. Compatible attributes in a universal relation. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Los Angeles*, pages 55–62, March 1982.

[7] Nicole Bidoit. *The VERSO Algebra or How to Answer Queries with Fewer Joins.* Rapport de Recherche 353, Universite de Paris-SUD, Centre d'Orsay, Laboratoire de Recherche en Informatique, Bat. 490, 91405 Orsay, France, 1987.

[8] Patrick C. Fischer and Stan Thomas. Operators for non-first-normal-form relations. In *Proceedings of the 7th International Computer Software Applications Conference, Chicago*, pages 464–475, November 1983.

[9] Gerhard Jaeschke. *An Algebra of Power Set Type Relations.* Technical Report 82.12.002, Heidelberg Scientific Center, IBM Germany, 1982.

[10] Gerhard Jaeschke. *Nonrecursive Algebra for Relations with Relation Valued Attributes.* Technical Report 84.12.001, Heidelberg Scientific Center, IBM Germany, 1984.

[11] Gerhard Jaeschke. *Recursive Algebra for Relations with Relation Valued Attributes.* Technical Report 84.01.003, Heidelberg Scientific Center, IBM Germany, 1984.

[12] Gerhard Jaeschke and Hans-Jörg Schek. Remarks on the algebra of non first normal form relations. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Los Angeles*, pages 124–138, March 1982.

[13] Anthony Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the ACM*, 29(3):699–717, July 1982.

[14] Volker Linnemann. Non first normal form relations and recursive queries: An SQL-based approach. In *Proceedings of the Third International Conference on Data Engineering, Los Angeles*, pages 591–598, February 1987.

[15] David Maier, David Rozenshtein, and J. Stein. Representing roles in universal relation scheme interface. *IEEE Transactions on Software Engineering*, 11(7):644–652, July 1985.

[16] Gültekin Ozsoyoğlu and Z. Meral Ozsoyoğlu. An extension of relational algebra for summary tables. In *Proceedings of the 2nd International (LBL) Conference on Statistical Database Management, Los Angeles*, pages 202–211, September 1983.

[17] Gültekin Ozsoyoğlu and Z. Meral Ozsoyoğlu. SSDB–an architecture for statistical databases. In *Proceedings of the 4th Jerusalem Conference on Information Technology, Jerusalem*, pages 327–341, May 1984.

[18] Gültekin Ozsoyoğlu, Z. Meral Ozsoyoğlu, and Victor Matos. *Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregate Functions.* Technical Report, Department of Computer Engineering and Science, Case Western Reserve University, Cleveland, OH, 1985.

[19] Z. Meral Ozsoyoğlu and Gültekin Ozsoyoğlu. A query language for statistical databases. In W. Kim, D. Reiner, and D. Batory, editors, *Query Processing in Database Systems*, Springer-Verlag, 1984.

[20] Z. Meral Ozsoyoğlu and Li-Yan Yuan. A new normal form for nested relations. *ACM Transactions on Database Systems*, 12(1):111–136, March 1987.

[21] J. Paredaens. On the expressive power of the realtional algebra. *Information Processing Letters*, 7(2):107–111, February 1978.

[22] Peter Pistor and F. Anderson. Designing a generalized NF2 model with an SQL-type language interface. In *Proceedings of the Twelfth International Conference on Very Large Databases, Kyoto*, pages 278–285, August 1986.

[23] Peter Pistor and R. Traunmüller. A data base language for sets, lists, and tables. *Information Systems*, 11(4):323–336, 1986.

[24] Srinivasen Ramakrishnan. *Design and Implementation of a Translator for SQL/NF with Role Joins.* Master's thesis, The University of Texas at Austin, Austin, Texas, December 1986.

[25] Mark A. Roth, Henry F. Korth, and Don S. Batory. SQL/NF: A query language for ¬1NF relational databases. *Information Systems*, 12(1):99–114, 1987.

[26] Mark A. Roth, Henry F. Korth, and Abraham Silberschatz. *Extended Algebra and Calculus for ¬1NF Relational Databases.* Technical Report TR-84-36, Department of Computer Science, University of Texas at Austin, December 1984. revised January 1986.

[27] Mark A. Roth, Henry F. Korth, and Abraham Silberschatz. *Null Values in ¬1NF Relational Databases.* Technical Report TR-85-32, Department of Computer Science, University of Texas at Austin, December 1985.

[28] Hans-Jörg Schek and Peter Pistor. Data structures for an integrated data base management and information retrieval system. In *Proceedings of the Eighth International Conference on Very Large Databases, Mexico City*, pages 197–207, September 1982.

[29] Hans-Jörg Schek and Marc H. Scholl. *An Algebra for the Relational Model with Relation-Valued Attributes.* Technical Report DVSI-1984-T1, Technical University of Darmstadt, Darmstadt, West Germany, 1984.

[30] Hans-Jörg Schek and Marc H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):137–147, 1986.

[31] N. Shu, Y. Lum, F. Tung, and C. Chang. Specification of forms processing and business procedures for office automation. *IEEE Transactions on Software Engineering*, 8(5):499–512, September 1982.

[32] Dirk Van Gucht. On the expressive power of the extended relational algebra for the unnormalized relational model. In *Proceedings of the Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego*, pages 302–312, March 1987.

[33] Dirk Van Gucht and Patrick C. Fischer. High-level data manipulation languages for unnormalized relational database models. In *Proceedings of the XP/7.52 Workshop on Database Theory, Austin*, August 1986.