An Extended Recursive Algebra for Nested Relations and its Optimization

Balaji Rathakrishnan Junguk L. Kim

Department of Computer Science
Texas A&M University
College Station, TX 77843

Abstract

This paper considers query optimization for nested relational databases based on a recursive algebra. A recursive algebra which has been proposed in the literature for nested relations has been extended to accommodate arbitrary algebraic expressions in any nested level of the query. This extended algebra can be used to conveniently express queries in two query languages proposed for the nested relational model. A semijoin based algebraic optimization strategy for queries expressible in this algebra is described, which is a recursive extension of an 1NF optimization technique with some modifications.

1 Introduction

The nested relational model is one of the third generation database models proposed to overcome some of the limitations of the relational model in handling complex hierarchical data. Like the relational model, the nested relational model allows declarative querying on the database and hence the flexibility of determining the optimal evaluation of user queries. However, due to the lack of a standard algebra and a query language for the nested relational model, query optimization for nested relational databases has not been studied extensively. This paper considers query optimization with respect to a recursive algebra on nested relations.

Ever since the nested relational model was proposed by Makinouchi [7], several algebras have been suggested for this model [3, 11, 12, 15]. These algebras are based on the *nest* and *unnest* operators, and simple extensions to the set operations ($\cup, \cap, -$) and the join operator. However none of these algebras define extensions to the select and project operations that can directly access "deeply" nested attributes. Such operations with respect to "deeply" nested attributes have to be performed by prior unnesting of these attributes to bring them to the highest nested level. Further, the join operators defined in these algebras do not allow joins between relations whose common attributes are at different nested levels.

Recursive algebras which allow direct access of deeply nested attributes have been defined by Colby [1] and Schek and Scholl [13] (SS algebra). Of these algebras, the SS algebra is a more natural extension of the 1NF algebra in that it allows arbitrary operations on relation-valued attributes at any nested level. Further, the SS algebra also has a close correspondence to two SQL-like languages proposed for nested relational databases (SQL/NF) [10] and the recursive SQL-like language developed for the Advanced Information Management Prototype (AIM-P) project [6, 8]). Both these languages have an extended Select-From-Where (SFW) construct which permits recursively embedded SFW constructs on relation-valued attributes in their SELECT and WHERE clauses. This recursive SFW construct can be directly translated into an expression belonging to the proposed extended version of the SS algebra.

Past work in query optimization for $\neg 1NF$ databases includes [1, 2, 4, 14]. In [1], a list of algebraic equivalences based on Colby's recursive algebra defined are provided. However, a systematic optimizing procedure is not discussed. In [2], calculus-based optimization of nested relation selections is studied. In [4], an approach for efficiently evaluating unnest-joins (join after completely unnesting the relations) is discussed. In [14], Scholl discusses query optimization in an 1NF conceptual scheme for an internal $\neg 1NF$ database. This approach is based on eliminating joins at the conceptual scheme level which are precomputed into $\neg 1NF$ relations in the physical database. Thus at

0730-3157/93 $3.00

the $\neg 1NF$ level, this approach considers only queries involving one nested relation. Generalized optimization of Select-Project-Join queries on nested relational databases is not considered by these approaches.

This paper considers the algebraic optimization of nested relational queries expressed in an extended version of the SS algebra assuming a $\neg 1NF$ relational model at both conceptual and physical database layers. Our approach is to utilize 1NF algebraic optimization principles to the nested relational case. Since it is a natural extension of the 1NF algebra, the SS algebra was found to be a suitable basis for algebraic optimization. We propose an extension to the SS algebra which makes it more expressive and convenient for applying 1NF optimization. First, the SS algebra is extended to allow arbitrary algebraic expressions–rather than just single operators–on relation-valued attributes at any nested level. Secondly, the algebra is extended to allow entire relations to appear in embedded algebraic expressions along with relation-valued attributes. This feature is needed to *join* relations with common attributes at different nested levels. With these extensions, the algebra is convenient for direct translation from the recursive Select-From-Where construct of the nested relational query languages as described above.

The proposed extended recursive algebra treats algebraic expressions both at the highest level (which corresponds to the query itself) and those embedded at any nested level uniformly, so that algebraic equivalences and optimization techniques can be applied uniformly to both. With this algebraic basis, we discuss an approach towards optimizing a class of nested relational queries which could be represented by a recursive SQL-like Select-From-Where (SFW) expression, or by an equivalent Select-Project-Join expression of the proposed extended SS algebra . A semi-join based query processing strategy is developed, which is a recursive extension of a modified 1NF query processing technique used in each nested level of the query.

2 Nested relations and the recursive algebra

The nested relational model adopted is that defined by Schek and Scholl [13]. We introduce the following example which would be used to illustrate the proposed algebra and its optimization.

Example 1 *Consider a distribution company database [5], consisting of the three nested relations,* **customers**, **new-orders** *and* **products**. *Figure 1 shows the schemes of these nested relations in a tree form. Values of the relations* new-orders, products *and* customers *are shown in Figure 2. These values would be used in other examples throughout this paper.*

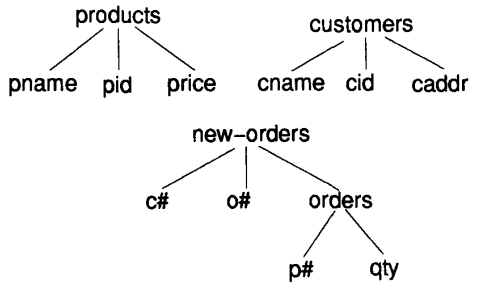

Figure 1: Scheme trees for the distribution company database

2.1 The Extended SS Algebra

The extended algebra is defined in terms of the set of allowable algebraic expressions and the operators are defined as constructors of more complex expressions from simpler ones, with nested relations and relation-valued attributes being the base cases. The aim of the algebra is to allow arbitrary algebraic expressions at any level of the nested relations by means of extending the definitions of the π and the σ operators. In the SS algebra, only single level extensions of the π and σ operators have been defined formally. Further, in the proposed algebra entire relations could be embedded in nested expressions thereby enabling joins between relations with "joinable" attributes at different nested levels.

We define initially, what are called *nested expressions* in which both nested relations and relation-valued attributes participate as operands. This class of expressions captures syntactically, both the actual external (highest level) algebraic expressions and also those encountered in project-lists and select conditions of higher level expressions. Then the actual class of highest level algebraic expressions which correspond to user queries is defined as a special class of *nested expressions*, called *external nested expressions*, in which only nested relations participate. Note that *external nested expressions* may contain *nested expressions* in their project-lists and select conditions.

For the sake of brevity, we provide a brief informal discussion of the algebra (for a full formal definition

products		
pname	*pid*	*price* ($)
Television	14562	856
Couch	15641	539
Luggage	32156	219
Walkman	12345	69

customers		
cname	*cid*	*caddr*
John	0123	1043, Socorro, CS
Ancin	0342	404D, First St., CS
Aaron	0341	15 Harvey, CS

new-orders			
c#	*o#*	*orders*	
		p#	*qty*
0341	9123	12345	500
		15641	35
0123	9124	14562	150
		12345	25
0341	9156	32156	125
		14562	75

Figure 2: Values of nested relations in Example 1

see [9]).

Nested expressions: A *nested expression* is defined in terms of a set of relation-valued attributes and nested relations. If R is a set of nested relations and A is a set of relation-valued attributes, then the set of valid nested expressions defined over A and R is represented by $NE(A, R)$. A nested expression is constructed from the attributes in A and relations in R as base cases and all the conventional 1NF operations are valid constructors of complex expressions from simpler ones. The project (π) and select (σ) operators differ from the 1NF case in that they allow nested expressions in their project-lists and select conditions respectively.

Consider the distribution company database of Example 1. The following is a valid nested expression (say E_1) in $NE(attr(new\text{–}orders), \{products\})$:

$$\pi[pname, qty, price](\sigma[pid = p\#](orders \times products))$$

E_1 performs a join of the *orders* relation-valued attribute of the relation *new-orders* with the *products* relation. Note that nested expressions do not have a value of their own as they are defined over attributes. Hence they take a value only when they are evaluated against a tuple whose attribute values instantiate the attributes in the nested expression. The value of a nested expression E is defined in terms of a tuple and is denoted by $val(t, E)$. In the above example, E_1 does not have a value by itself except when instantiated over a tuple of *new-orders*.

External nested expressions: An *external nested expression* is a special case of nested expressions which are defined in terms of only nested relations. Thus the class of *external nested expressions* defined over a set of nested relations **R**, is given by $ENE(R) = NE(\phi, R)$. As mentioned before, an *external nested expression* may contain a *nested expression* in its project-lists or in its select-conditions.

The following is an example of an *external nested expression* in the class $ENE(\{customers, new\text{–}orders, products\})$:

$$E \equiv \pi[c\#, E_1 : invoice](new\text{–}orders)$$

E is an external nested expression which has the embedded nested expression E_1 shown in the previous example. E projects the $c\#$ attribute of the relation *new-orders* and evaluates the nested expression E_1 against each tuple of *new-orders* and forms a new relation-valued attribute called *invoice*.

Recursive SFW queries: A recursive SFW query (based on the SQL-like languages defined in [8, 10]) over a set of nested relations R is of the form:

SELECT
A_{i_1} [*as* B_1], A_{i_2} [*as* B_2], $\cdots$, A_{i_n} [*as* B_n]
FROM R_{i_1}, R_{i_2}, $\cdots$, R_{i_k}
WHERE $C_1 \wedge C_2 \wedge \cdots \wedge C_l$

where, $R_{i_j} \in R, 1 \leq j \leq k$ are nested relations,, each of the A_{i_j}'s is either an attribute of $R' = R_{i_1} \times R_{i_2} \times \cdots \times R_{i_k}$ or is a recursive SFW expression over $attr(R') \cup R''$, where R'' is a possibly empty set of nested relations. Similarly, each of the conditions C_j is either a simple attribute comparison as in flat SFW expressions or a set comparison between nested relations generated by SFW expressions over $attr(R') \cup R'''$, where R''' is a possibly empty set of nested relations. R'' and R''' allow relation-valued attributes to be joined with other nested relations at any nested level.

Recursive Select-Project-Join expressions: A recursive SPJ expression is of the form $\pi[L]\sigma(F)(R_1 \times R_2 \times \cdots \times R_m)$, where L and F may have embedded SPJ expressions in their project-lists and select conditions respectively. The recursive SFW query of definition 2.1 can be translated into the recursive SPJ expression: $\pi[L]\sigma(C_1 \wedge C_2 \cdots \wedge C_l)(R_{i_1} \times R_{i_2} \times \cdots \times R_{i_k})$, where the SFW subqueries in the $A'_{i_j}s$ and the $C'_k s$

are replaced by the corresponding recursive SPJ expressions. Note that, a recursive SPJ expression E over a set of nested relations R is an external nested expression in R (i.e., $E \in ENE(R)$). Similarly, a recursive SPJ subexpression E_1 over a set of attributes A and a set of relations R' is a nested expression (i.e., $E_1 \in NE(A, R')$).

Example 2 *Consider the following query on the distribution company database of example 1: "For each order by customer Aaron which includes a television set, generate an invoice showing the ordering customer's name and address and the prices of the items ordered.". This query can be represented as a recursive SFW construct as follows:*

SELECT cname, caddr, (SELECT pname, qty, price
FROM orders, products
WHERE pid=p#) AS invce
FROM new-orders, customers
WHERE {Television} IN (SELECT pname
FROM orders, products
WHERE p#=pid)
AND c#=cid AND cname="Aaron"

cname	*caddr*	*invce*		
		pname	*price*	*qty*
Aaron	15 Harvey, CS	Luggage	219	125
		Television	856	75

Figure 3: The result of the SFW query

The result of this query on the database of Example 1 is shown in Figure 3. This query can be equivalently expressed as the recursive SPJ expression E shown in Figure 4. Now, E is an external nested expression in $ENE(\{new-orders, customers\})$. *Also, its project list contains a nested expression* $E_1 \in NE(attr(new\text{–}orders), R')$ *where* $R' = \{products\}$. *Similarly, the select formula has a nested expression* $E_2 \in NE(attr(new\text{–}orders), R')$. *Note that, an entire relation* products *appears in the embedded SFW expressions and nested expressions.*

3 Algebraic optimization of recursive SPJ expressions

Now we consider the algebraic optimization of recursive SPJ expressions described in the previous section. The main advantage of the recursive algebra as defined is that 1NF optimization techniques can be applied to this algebra recursively to every nested expression. In 1NF algebra, an important underlying heuristic is that join and cross-product are considered to be expensive. Hence the sizes of the relations participating in the join or cross-product operations are reduced by either performing selections and projections as early as possible [16, 17].

This basic strategy is retained for the optimization of recursive SPJ queries with some modifications. The modification is due to the addition of two more *expensive* operators to the algebra, viz., *select* involving nested expressions and set comparisons and *project* involving nested expressions on relation-valued attributes of the operand relation. Consider the nested expression $\pi[E_1](R)$, where E_1 is a nested expression in $NE(attr(R), R')$. Now, this projection is *expensive* because the nested expression E_1 should be evaluated for each tuple of the relation R. E_1 could be an arbitrarily complex expression and hence this can no longer be considered a simple projection. Similarly, consider the expression $\sigma[F](R)$, where F is of the form E_1 *cop* C or E_1 *cop* E_2, E_1 and E_2 being nested expressions in $NE(attr(R), R')$. To evaluate this selection, E_1 (or E_1 and E_2) has to be evaluated for each tuple of R followed by a set comparison.

With this modified view of what is *expensive*, the query processing steps are developed accordingly. In 1NF optimization, all projections and selections involving single relations are pushed down the query evaluation tree as far as possible, before selections and projections involving multiple relations are considered. However, in our approach, only flat projections and selections are pushed down the query tree. After this, in the key optimizing step, the relations are further reduced by using flat semijoins with respect to other relations. Only after this reduction, nested projections and selections involving single relations are performed against the reduced number of tuples. After reducing the relations in the external expression, the nested expressions in the project-lists and select atoms are optimized using the same procedure. Before going into the optimization steps, the algebraic equivalences which allow the optimizing transformations are defined:

3.1 Algebraic equivalences

We list some of the algebraic equivalences which are used in the optimizer described. These equivalences imply both scheme and value equivalences. The identical definitions for *nested expressions* and *external nested expressions* (except that nested expressions are defined in terms of a tuple) imply that these equivalences hold for both these classes of ex-

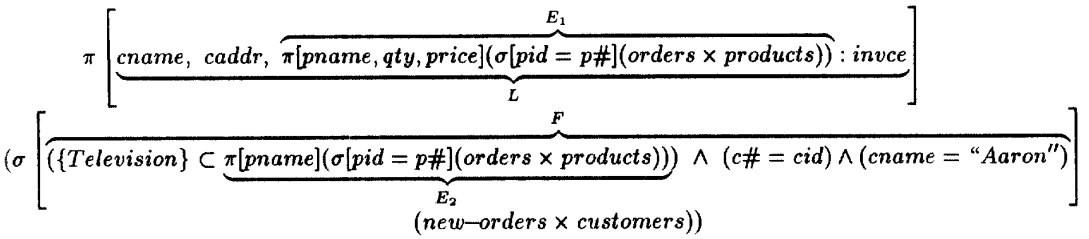

Figure 4: Recursive SPJ expression E of Example 2

pressions. Thus the equivalence $E_1 \equiv E_2$ implies that $val(E_1) = val(E_2)$, if E_1 and E_2 are external nested expressions, and $val(t, E_1) = val(t, E_2)$, if they are nested expressions.

1. $\pi[L_{flat}; L_{nested}](E) \equiv$
   $\pi[L_{flat}; L_{nested}](\pi[L_{flat}; attributes(L_{nested})](E))$
   where L_{flat} consists of only flat projections and L_{nested} consists of only nested projections; $attributes(L_{nested})$ refers to the set of attributes involved in the nested expressions in L_{nested}. This equivalence allows prior projecting out of attributes involved in nested projections.

2. $\pi[L; A_{i_j} : B_j](E) \equiv \pi[L; B_j](\pi[L; A_{i_j} : B_j](E))$.
   This equivalence allows prior evaluation of the nested expressions in the project-list which can be pushed below cross-products.

3. $\pi[L](\sigma[F](E)) \equiv$
   $\pi[L](\sigma[F](\pi[L; attributes(F)](E)))$,
   where L consists of only attributes and no nested expressions and $attributes(F)$ refers to the set of attributes of E referenced in the select-atoms of F. This equivalence allows prior projection of only attributes which would be needed for future *selects* and those which appear in the target list.

4. $\pi[L_1; L_2](E_1 \times E_2) \equiv \pi[L_1](E_1) \times \pi[L_2](E_2)$,
   if L_1 and L_2 involve only attributes from E_1 and E_2 respectively.

5. $\sigma[F_1 \wedge F_2 \wedge F_{12}](E_1 \times E_2) \equiv$
   $\sigma[F_{12}](\sigma[F_1](E_1) \times \sigma[F_2](E_2))$,
   where F_1 and F_2 involve only attributes from E_1 and E_2 respectively and F_{12} involves attributes from both E_1 and E_2.

6. $\sigma[F_{12}](E_1 \times E_2) \equiv$
   $\sigma[F_{12}]((E_1 \ltimes_{F_{12}} E_2) \times (E_2 \ltimes_{F_{12}} E_1))$,
   where F_{12} involves only select atoms which involve attributes from both E_1 and E_2.

7. If $E_1 \equiv E_2$, where $E_1, E_2 \in NE(attr(E), R)$, then
   (a) $\pi[E_1 : B_1; L'](E) \equiv \pi[E_2 : B_1; L'](E)$ and
   (b) $\sigma[E_1\ cop\ C](E) \equiv \sigma[E_2\ cop\ C](E)$.

   These equivalences allow independent optimization of nested expressions embedded in project lists and select atoms.

3.2 The optimization strategy

Now we describe the various steps involved in processing a recursive SPJ expression. The steps describe the various transformations from the initial expression and the final algebraic expression serves as the query evaluation plan, with the operations performed in the order indicated by the final expression. For the sake of simplicity, the steps are illustrated with the transformations applied to the expression $E = \pi[L](\sigma[F](R_1 \times R_2))$, where R_1 and R_2 are nested relations. The procedure can be easily extended for multiple nested relations. The expression E after the following transformations is shown in Figure 5.

1. Split the project list L into two segments, L_{flat} and L_{nested} as described in equivalence (1). Similarly, split the selection formula F into two formulas, F_{flat} involving only atomic attribute comparisons and F_{nested} involving set comparisons against nested expressions. The splitting of the select formula is needed to avoid expensive nested selects which are postponed to be performed after further reduction based on flat selects and semijoins.

2. Move down the projection of L' below the selection using equivalence (3).

3. Move down the projection of L'' below the cross-product by equivalence (4).

$$
\begin{aligned}
E &\equiv \pi[L_{flat}; L_{nested}](\pi[\underbrace{L_{flat}; attributes(L_{nested})}_{L'}](\sigma[F_{flat} \wedge F_{nested}](R_1 \times R_2))) && (1)\\
&\equiv \pi[L_{flat}; L_{nested}](\sigma[F_{flat} \wedge F_{nested}](\pi[\underbrace{L'; attributes(F)}_{L''}](R_1 \times R_2))) && (2)\\
&\equiv \pi[L_{flat}; L_{nested}](\sigma[F_{flat} \wedge F_{nested}](\underbrace{\pi[L^1](R_1)}_{R'_1} \times \underbrace{\pi[L^2](R_2)}_{R'_2})) && (3)\\
&\equiv \pi[L_{flat}; L_{nested}](\sigma[F^{12}_{flat} \wedge F_{nested}](\underbrace{\sigma[F^1_{flat}](R'_1)}_{R''_1} \times \underbrace{\sigma[F^2_{flat}](R'_2)}_{R''_2})) && (4)\\
&\equiv \pi[L_{flat}; L_{nested}](\sigma[F^{12}_{flat} \wedge F_{nested}](\underbrace{(R''_1 \ltimes_{F^{12}_{flat}} R''_2)}_{R'''_1} \times \underbrace{(R''_2 \ltimes_{F^{12}_{flat}} R''_1)}_{R'''_2})) && (5)\\
&\equiv \pi[L_{flat}; L_{nested}](\sigma[F^{12}_{flat} \wedge F'_{nested}](R'''_1 \times R'''_2)) && (6)\\
&\equiv \pi[L_{flat}; L_{nested}](\sigma[F^{12}_{flat} \wedge F^{12}_{nested}](\underbrace{\sigma[F^1_{nested}](R'''_1)}_{R^{iv}_1} \times \underbrace{\sigma[F^2_{nested}](R'''_2)}_{R^{iv}_2})) && (7)\\
&\equiv \pi[L_{flat}; L_{nested}](\sigma[\underbrace{F^{12}_{flat} \wedge F^{12}_{nested}}_{F_{12}}](\underbrace{(R^{iv}_1 \ltimes_{F^{12}_{nested}} R^{iv}_2)}_{R^v_1} \times \underbrace{(R^{iv}_2 \ltimes_{F^{12}_{nested}} R^{iv}_1)}_{R^v_2})) && (8)\\
&\equiv \pi[L_{flat}; L'_{nested}](\sigma[F_{12}](R^v_1 \times R^v_2)) && (9)\\
&\equiv \pi[L_{flat}; L^{12}_{nested}; names(L^1_{nested}); names(L^2_{nested})](\sigma[F_{12}](\pi[L^1_{nested}](R^v_1) \times \pi[L^2_{nested}](R^v_2))) && (10)
\end{aligned}
$$

where $names(L)$ is the list of names of the components appearing in L.

Figure 5: Optimizing transformations on recursive SPJ expression

4. Split F_{flat} into $F^1_{flat} \wedge F^2_{flat} \wedge F^{12}_{flat}$, as in equivalence (6) and move down the selection corresponding to F_1 and F_2.

5. In this step, this algorithm again differs from the conventional 1NF optimizer. The semijoin with respect to F^{12}_{flat} is performed to reduce each of the relations before applying the nested selections and projections. In the above expression for E, R''_1 and R''_2 are replaced by their semijoins with each other with respect to F^{12}_{flat}. If the nested expressions in L_{nested} and F_{nested} do not involve one of the relations R_1 or R_2, then the corresponding semijoin could be skipped.

6. Now that all possible reduction in the number of tuples has been done based on flat selections, nested selection is performed on the remaining tuples to further reduce the relations involved in the cross-product. However, before nested selection, the nested expressions in F_{nested} are optimized using this algorithm recursively. Let F'_{nested} be the optimized version of F_{nested}.

7. Now F'_{nested} is split according to equivalence (6) and the selections are pushed below the cross-product:

8. The reduced relations R^{iv}_1 and R^{iv}_2 are now further reduced by replacing them by their respective semijoins with the other with respect to F^{12}_{nested}. This semijoin reduces the relations to only the tuples which would participate in the final join, so that the number of evaluations of L_{nested} is reduced. Again, if L_{nested} does not involve one of the relations, then the corresponding semijoin could be skipped.

9. Now the nested expressions in L_{nested} are transformed using this algorithm recursively, to give the final optimized expression. Let L'_{nested} be the optimized version of L_{nested}.

10. Now separate L'_{nested} into L^1_{nested}, L^2_{nested} and L^{12}_{nested}, and push down the projections of L^1_{nested} and L^2_{nested} below the cross-products using equivalences (2) and (4) respectively.

4 Conclusion

In this paper, we have described an optimization technique for nested relational queries based on a recursive algebra. We found that, by an uniform treatment of nested algebraic expressions along with external expressions, we could use the same optimization procedure at every nested level. It was also found that semijoins which have been used in the 1NF relational model for query optimization could be used not only to reduce the number of tuples involved in a join operation but also to avoid expensive recursive select and project operations. The basic approach is to postpone nested operations after reduction in the number of tuples based on flat operations. It should be noted that the emphasis is not on the actual 1NF optimization technique used at each level, which could be different and more sophisticated.

References

[1] L. S. Colby. Recursive algebra and query optimization for nested relations. In *Proceedings of ACM-SIGMOD 1989 Annual Conference, Portland, Oregon*, pages 273–283, 1989.

[2] Jan Van den Bussche. Evaluation and optimization of complex object selections. In *Deductive and Object-Oriented Databases: Second International Conference*, pages 226–243. Springer-Verlag, New York, December 1991.

[3] G. Jaeschke and H.-J. Schek. Remarks on the algebra of first normal form relations. In *Proceedings of ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 124–138, 1982.

[4] Henry F. Korth. Optimization of Object-Retrieval Queries. In *Advances in Object-Oriented-Database Systems: 2nd International Workshop on Object-Oriented Database Systems*, pages 352–357. Springer-Verlag, New York, September 1988.

[5] Henry F. Korth and Mark A. Roth. Query languages for nested relational databases. In *Nested Relations and Complex Objects in Databases*, pages 190–204. Springer-Verlag, New York, 1987.

[6] Volker Linneman. Nested relations and recursive queries. In *Nested Relations and Complex Objects in Databases*. Springer-Verlag, New York, 1989.

[7] A. Makinouchi. A consideration on normal form of not-necessarily-normalized relations in the relational data model. In *Proceedings of the 3rd International Conference on Very Large Data Bases*, pages 447–453, October 1977.

[8] P. Pistor and R. Traunmueller. A database language for sets, lists and tables. *Information Systems*, 11(4):323–336, 1986.

[9] Balaji Rathakrishnan. Query optimization in nested relational databases. Master's thesis, Texas A&M University, 1992.

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

[11] Mark A. Roth and James E. Kirkpatrick. Algebras for nested relations. *Database Engineering*, 7:154–162, 1988.

[12] Mark A. Roth, Henry F. Korth, and Abraham Silberschatz. Extended algebra and calculus for ¬1NF relational databases. *ACM Transactions of Database Systems*, 13(4), December 1988.

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

[14] Marc H. Scholl. Theoretical foundation of algebraic optimization utilizing unnormalized relations. In *Proceedings of the International Conference on Database Theory*, pages 380–396, September 1986.

[15] Stan J. Thomas and Patrick C. Fischer. Nested relational structures. *Advances in Computing Research*, 3:269–307, 1986.

[16] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems–Volume II: The New Technologies*. Computer Science Press, Rockville, 1989.

[17] E. Wong and K. Youssefi. Decomposition–A strategy for query processing. *ACM Transactions of Database Systems*, 1(3):223–241, September 1976.