Journal of Theoretical and Applied Information Technology

www.jatit.org

DATABASE ALGEBRAS

[1] Pushpa R. Suri, [2] Sudesh Rani

[1] Reader, Department of Computer Sc. & Applications, K. U., Kurukshetra , India-1361191
[2] Ph.D. Student , Department of Computer Sc. & Applications, K. U. , Kurukshetra , India-136119
E-mail: pushpa.suri@yahoo.com , dhillon_sudesh@rediffmail.com

ABSTRACT

The algebraic approach to database query processing continues to be successful long after its introduction in relational database. This paper is devoted to database algebras. The purpose of this paper is to convey some idea of the general nature of database algebras in the hope of gaining some insight into why certain operations exist and of identifying common themes among these algebras.

Keywords: *Database, Query, Algebra, Relational algebra, Nested relational algebra, Object oriented algebra.*

1. INTRODUCTION

As computers become larger (in capacity), smaller (in size), and more powerful, it becomes easier to do more complex things with them than could be done in the past. This is as true of database management systems (DBMSs) as of any other type of software. The central feature of a DBMS is its data model, the formal abstraction which it uses to map real-world entities onto (logical) database entities. Many different DBMSs may implement the same abstract data model. Data models are becoming increasingly powerful and complex day by day, and with this power and complexity comes the need to ensure correct and efficient execution of queries on data with increasingly complex structure.

1.1. Why an Algebra?

When executing a query in any database system, we wish to minimize the utilization of CPU, memory, I/O, and communications resources. Given current hardware technology (i.e., the availability of cheap memory), what we really want to minimize is a query's response time, which for centralized systems is usually dominated by the I/O transfer time. An orthogonal issue to speed is correctness. Clearly, the fastest query is useless if it returns the wrong answer (or never returns any answer at all). To achieve correctness with respect to a data model, one normally designs a calculus for manipulating the objects in a database. This becomes the definitive standard for lower level implementation of the system in the sense that any lower-level query languages should be equivalent to the calculus in expressive power. Most of the calculi proposed for this purpose have been first-order. This calculus often becomes the basis of the user-level query language for the system, as it is non-procedural in nature. It has become clear, from experience with the relational model, that it is much easier to pose queries using a non-procedural language than with a procedural language, in which the query must specify not only what data is desired, but how to go about retrieving that data.

Of course, at some point we must decide on a sequence of operations to retrieve the data physically. For this purpose there must be routines to access the data and to perform operations on them (e.g., the relational join). But going directly from a user's query to procedure calls on the actual database will generally result in very inefficient query plans. We need to be able to rearrange the operations to produce an optimal sequence of data accesses. The formality of an algebra allows us to do this. Naturally, this algebra must be equipollent (equivalent in expressive power) to the calculus and translatable into actual calls on the database. It is an intermediate language. The relational algebra performs this function in the relational model, and is one reason why the model of query processing in relational systems is so attractive.

The processing of a query, then, occurs conceptually in four steps: 1) translation from the calculus representation to a functionally equivalent algebra representation; 2) logical transformations of

the algebraic query to standardize and simplify it; 3) generation of alternative algebraic strategies combined with alternative access and operator methods to produce plans for retrieving the data; and 4) selection of the cheapest plan based on a designer-specified set of criteria.

2. DATABASE ALGEBRAS

An algebra is formally defined as a pair (S, Θ), where S is a (possibly infinite) set of objects and Θ is a (possibly infinite) set of n-ary operators, each of which is closed with respect to S [9]. These operations will have certain properties (e.g. commutativity) which make rearrangement of some algebraic expressions possible; this is what makes algebras so desirable for query optimization.

A large number of recent systems operate on objects much more complex than those found in relational systems. A few of these have a corresponding algebra which they could use to optimize queries, but most do not. In addition, several "generic" algebras for non-relational systems have been proposed. These are not geared toward any particular system or implementation. In the following sections we describe the basic concepts and operations of these algebras, regardless of their connections with actual systems. For each algebra, we will provide the following as appropriate: a short summary of the system and algebra (including goals of the system, structure of the objects operated on, and all of the algebraic operators), descriptions of any normal forms or other restrictions imposed by the model, some of the more important results about algebraic equivalences, whether the operators work on object-based or value-based structures, a rough idea of the expressive power of the algebra, whether or not there is a corresponding calculus, implementation status (where applicable and known).

2.1. The Relational Algebra

The relational model was first described in [3] and became the model of choice for standard business data processing applications in the 1980s. The model consists of three things: 1) Relations, 2) An algebra to operate on those relations, and 3) Rules and guidelines for database design and maintenance (including integrity constraints, etc.).

In the relational model, real-world entities are represented by tuples (or records). A tuple has a fixed integral number of named attributes (or fields). An important restriction is that these fields be filled only with scalar values. This is known as the first normal form (or 1NF) restriction. A relation (or table) is a set of tuples with identical layouts. This layout is called the relation's schema. Relations are manipulated using the relational algebra, which is equipollent to the relational calculus. Five standard relational operators as defined in [7] or [8] are:

1) Union (∪): Two relations can be combined into one using a standard set-theoretic union (duplicate tuples are eliminated). The two relations must have the same schema.

2) Difference (–): The expression R –S describes a relation consisting of all tuples in R which are not also in S. The two relations must have the same schema.

3) Cartesian Product (×): R×S indicates the relation whose schema consists of all attributes of R followed by all attributes of S. Each tuple of R is "concatenated" with each tuple of S to form the set of tuples in R×S.

4) Projection (π): This operation, when applied to a set of tuples and given a list of attributes, returns the same set of tuples, removing from each tuple all attributes but the ones listed in the π expression. If any duplicates were created during this process, they are removed.

5) Selection (σ): Given a relation R, this operator applies a predicate to each tuple in the relation. If the tuple satisfies the predicate, it becomes part of the result of the selection expression; otherwise, it is dropped.

2.2. Nested Relational Algebras

This section describes several algebras which operate on non-first normal form relations.

2.2.1. The DASDBS Algebra

The main feature of the DASDBS (Darmstadt Database System) [2, 5, 6, 10] is an application-independent kernel based on non-first normal form (NF^2) relations. Specific applications will be implemented on top of this kernel. The kernel itself supports a subset of the NF^2 algebra which will be presented here. The kernel also provides transaction management facilities.There is no limit to the level

of nesting of the relations. The following seven operators for the NF^2 algebra are proposed:

1)-5) Cross-product (×), Union (∪), Difference (–), Selection (σ), Projection (π): These are defined as in the relational algebra.

6) Nest (ν): This operator is applied to a single relation and must specify what attributes are to be nested and the name of the single attribute which will replace them. Specifically, $\nu_{A^*=A}$ (R), where A is a set of attributes and A* is a single relation-valued attribute whose tuples have members of A as columns, is a new relation, R*. R* is formed by using the attributes in (attrs(R) – A) as a key.

7) Unnest (μ): This is the inverse of ν. $\mu_{A^*=A}$ (R) is obtained by replacing the single attribute A* with the set of attributes A and for each tuple of R, forming a set of tuples over ((attrs(R) – {A*}) ∪ A) such that for each of these tuples t, t[A] was a tuple in the A* attribute of this tuple of R.

It should be noted that only two new operators, ν and μ, have been introduced. The first five operators are the standard operators needed for relational completeness. The most important property of the μ and ν operators is that, while μ is always the inverse of ν, it is not always the case that ν (μ (R)) = R. This equality holds iff a certain functional dependency holds on the relation: all nonscalar fields must be functionally dependent on the set of scalar fields. This is known variously as Partitioned Normal Form (PNF) [11, 19]. Another interesting property of ν is that this operator can be used to perform the grouping phase of aggregate function computation in a fairly straightforward way.

As the DASDBS algebra is a fairly straightforward extension of the relational algebra, it is value-based (as opposed to object-based), just as is the relational algebra. That is, to get a handle on an object, one must use its value(s), not some abstract sort of name or identifier or surrogate for the object. In other words, the system is not object-oriented. No formal calculus has been defined for the DASDBS system.

2.2.2. The AIM Algebra

The AIM project at IBM-Heidelberg [14, 15, 4] is designed to handle an extension of the NF^2 data model, with the goal of supporting non-traditional database applications. The major extension is the ability to handle ordered lists (arrays), multisets, and tuple-valued attributes. The seven operators described above for the DASDBS system appear essentially unchanged in the AIM algebra, with the exception of selection. Also, the AIM algebra treats renaming (ρ) as a separate operator, not as part of the π operator. The operations are the following:

1)-6) Nest (ν), unnest (μ), projection (π), Cartesian product (×), union (∪), and difference (–): These are defined exactly as in the DASDBS algebra.

7) Selection (σ): This is similar to the DASDBS σ, but the AIM selection allows an arbitrary algebraic expression to appear in a σ formula. This mimics the SQL notion of nested queries.

8) Keying (χ): This operator is introduced to eliminate the problem of non-invertible unnestings. χ appends a key column to a relation before it is unnested then renested, and this ensures that nesting after an unnest will result in the original NF^2 relation. This is needed only when there is not already a key column in the relation.

9) Renaming (ρ): This operator renames an attribute of a nested relation. The renamed attribute may appear anywhere in the schema. Such an operator is needed only because no ordering is assumed to hold on the columns of a relation. If such an ordering were assumed, renaming would not be necessary and column numbers could be used instead.

As with DASDBS, no formal calculus has been defined for the AIM system. Null values are not supported in the formal algebra. Joins in AIM are defined using the ∩ operator as follows: if the join attributes are both scalar, two tuples will join with one another if the attribute values are equal (as in the relational natural join). If the join attributes are both relation-valued, the tuples join iff the intersection of the two relation-valued join attributes is not ∅. This definition of the join operator seems to imply that the AIM system is value-based rather than object-based.

2.2.3. The Vanderbilt Algebra

The algebra for NF^2 relations described in [7] was not defined for use with a particular system,

but rather as a general algebra for use with any NF^2 system, just as Codd's relational algebra [3] was intended to apply to any number of relational systems. The objects operated on by the Vanderbilt algebra are exactly the NF^2 relations, as with the DASDBS algebra. That is, a relation can have atomic and relation-valued attributes. The following operations can be applied to nested relations:

1)-4) Project (π), Union (∪), Difference (–), Cartesian Product (×): These are defined exactly as in the relational algebra.

5) Select (σ): This is similar to the relational σ operator but has been extended with set comparators and setvalued constants.

6) Nest (NEST): This is defined exactly as is the ν operator in the DASDBS and AIM NF^2 algebras.

7) Unnest (UNNEST): UNNEST is identical to the μ operator of the DASDBS and AIM algebras.

8) Flatten (UNNEST*): This operator simulates a sequence of UNNEST operations on a relation which will transform it into an equivalent flat (1NF) relation. That is, all possible unnestings are performed. Note that the order in which these are performed is irrelevant, since $\text{UNNEST}_{A=S}(\text{UNNEST}_{B=T}(R)) = \text{UNNEST}_{B=T}(\text{UNNEST}_{A=S}(R))$.

Joins in the Vanderbilt algebra are defined as they are for the AIM algebra. This exemplifies the fact that the algebra is strictly value-based. Also, it should be clear that this algebra has the same expressive power as the DASDBS and AIM algebras. Finally, it should be noted that no calculus corresponding to the Vanderbilt algebra has been defined.

2.2.4. The SQL/NF Algebra

SQL/NF [12, 11] is an extension of the SQL relational query language to handle non-first normal form relations. As with the previous three algebras, the domain of interest here is the set of NF^2 relations.The set of operators for the algebra is as follows:

1) Select (σ): [6] claims not to extend the relational σ operator, but their proofs use a σ which is extended with the ability to specify set-valued constants. This is the notion of σ we adopt here. Note that no set-based comparisons have been added; only equality may be tested for.

2) Union ($\cup^e$): Union is defined recursively for pairs of relations with the same schema. For flat schemas, it works exactly as does the relational ∪. But when a relation-valued attribute is encountered the $\cup^e$ operator is applied to this relation, recursively. That is, to preserve PNF, tuples with common values on the scalar fields will be combined into a single tuple in which the relation-valued fields will in turn be operated on by $\cup^e$.

3) Difference ($-^e$): This is defined recursively in a fashion similar to the $\cup^e$ operator.

4) Intersection ($\cap^e$): This is also defined similarly to $\cup^e$. Two tuples intersect if they agree on their scalar (key) attributes and the intersections of all of their nested attributes are non-empty.

5) Cartesian Product (×): This is defined as in the relational algebra.

6) Projection (π^e): This consists of a normal relational π (i.e., only a top-level attribute may be projected out, but it need not be scalar) followed by a unioning ($\cup^e$) of all the resulting tuples to remove duplicates.

7) Natural Join ($|\times|^e$): The natural join of two relations is defined only if all the common (joining) attributes are top-level attributes of the relation. Two tuples join only if they agree on all common scalar attributes and the intersections ($\cap^e$) of their common set-valued attributes are all non-empty.

8) Nest (ν): This is defined exactly as in the Vanderbilt algebra.

9) Unnest (μ): This is also defined exactly as in the Vanderbilt algebra.

Neither null values nor empty sets are supported in the algebra, which avoids some interesting problems regarding the unnesting of an empty set in a system not supporting nulls. Also, it is clear that, like the rest of the algebras discussed thus far, this algebra is purely value-based. A formal calculus corresponding to the SQL/NF algebra has been defined and proved equivalent to the algebra.

2.2.5. The NRDM Algebra

The Nested Relational Data Model (NRDM) is a system being developed and implemented at Indiana University [19]. Once again, the domain of objects is the set of relations with relation-valued attributes, just as in the previous algebras. This algebra also enforces the PNF restriction on its data, which again has an effect on the definition of the algebraic operations. These operators are defined as follows:

1)-3) Union ($\cup^e$), Difference ($-^e$), Project (π^e): These are defined in the same manner as in the SQL/NF algebra.

4) Select (σ^e): This is extended from the relational σ to allow membership testing in sets, set comparators, and selections from any level of the hierarchical structure.

5) Join ($|\times|$): Joins are allowed only when the join attributes are atomic. Joins are identical to the relational natural join, and can occur only when the join attributes are at the top level.

6) Nest (ν): This works essentially as in the previous algebras, but is restricted to yield a structure with only one set-valued attribute at the highest level. That is, a ν operation must nest at least all of the relation-valued attributes at the top level of the structure, resulting in just one relation-valued attribute. This can be worked around by joining two or more relations which have only one set-valued top-level attribute.

7) Unnest (μ): This merely inverts the ν operator.

Null values are not supported in this algebra. No calculus has been defined to correspond specifically to this algebra, but, like with the other algebras, it is not hard to envisage an adaptation of the SQL/NF calculus [11] to this algebra. It also shares with all of the previously defined algebras the characteristic of being value-based. There are no formal results on the expressive power of this algebra, but it almost certainly has the same expressive power as SQL/NF, due to the PNF restriction.

2.3. Database Algebras for advanced data models

This section describes algebras for advanced data models other than the nested relational model. Algebras described in detail here include algebras for relational aggregates, summary tables, complex objects, and object-oriented systems.

2.3.1. An Algebra for Aggregates

In [1] an algebra is developed which extends the relational algebra to include aggregate computations. The set of objects over which this algebra operates is the set of 1NF relations. The operators are thus identical to the relational algebra operators, with the addition of an operator to handle aggregates, which will be described below. For completeness, we present all of the operators:

1)-4) Projection, Cross product, Union, and Difference are the standard relational operators.

5) Restriction: This is used in place of selection, and results in an equivalent algebra. It is written as $e[X \Theta Y](I)$, where e is an algebraic expression and I is an instance of a relation. It restricts the result to contain those tuples whose X component stands in relation Θ to its Y component, where Θ is either = or <. It differs from selection in that many definitions of selection allow constants to appear in the predicate as well as values from the actual tuples.

6) Aggregate formation: This is denoted by $e<X, f>(I)$, where e is an algebraic expression, I is an instance of a relation, X is a set of attributes of the relation, and f is a function. The result of this operation is the set of tuples over the attributes ($X \cup \{F\}$), where F is an attribute indicating the result of applying the function f to the X attributes of each tuple of the original relation. More precisely, this operator partitions its input on the attributes X, applies the function f to each partition, and returns the X-value and associated f-value for each partition. As a limiting case, X can be empty, and we return simply a single value (for example, the average value of a single column, taken over the entire relation).

In [5] a calculus is defined for this algebra and their equipollence is proved. Clearly, this algebra is value-based and null values are not allowed in this algebra.

2.3.2. A Complex Object Algebra

In [16], an algebra and calculus are developed for a model whose objects are sets of values constructed from the set and tuple type constructors in an arbitrary fashion; this is a pure complex object

model. The model is also purely value-based, thus equality of two entities is determined by traversing their entire structures. Sets in the model are strictly homogeneous, and cyclic schemas are not allowed. The following operators are provided for complex objects:

1)-3) Union (∪), Intersection (∩), and Difference (–): Each of these takes two inputs of the same set type and produces a result of that type. They are defined exactly as in set theory. As in set theory, ∩ is redundant.

4) Cross product ($cross_A$): This is an n-ary operator which forms a set of n-tuples from n inputs, each of which must be a set. The value is the cross product of the input sets. The subscript indicates the name of the new tuple type contained in the result set.

5) Rename (rename A→B): This renames all occurrences of type A in the input expression to type B. No values are changed.

6) Powerset (powerset $_A$): This returns the set of all subsets of its input, which must be a set. The subscript indicates the name of the type contained in the result set.

7) Set-collapse (set-collapse): Given a set of sets, the member sets are all unioned together to produce a single set.

8) Replace (ρ<G>(R)): This operation iterates through a set of objects and performs the transformations indicated by G. G is a "replace specification". G is built recursively as follows: base specifications are constants, names of types appearing in the input type, and input parameters (each specification G can have input parameters which are algebraic expressions of a specified type; these are distinct from the input to the "replace" operator). Recursive specifications are built up from base specifications using the tuple and set constructors (i.e., a tuple or set of specifications is also a specification). Conditional and applicative specifications are also used recursively to build other specifications. A conditional specification can use the comparators "=" and "∈", and returns an object as its result only if the predicate is true. It acts like a relational selection. Finally, an applicative specification applies one of the other algebraic operators to each element of the input set (each element may, of course, have had other specifications applied to it prior to being given to the algebraic operator).

2.3.3 An Object-Oriented Database Algebra

[17] describes an object-oriented data model and algebra. Structurally, the model supports scalar values and the tuple and set type constructors, as well as inheritance among tuple types. One important extension, though, is that sets need not be homogeneous. Everything in a database is an object with its own unique identity, thus everything in a field of a tuple and every element in a set (homogeneous or not) is an object identifier. Unlike many object-oriented models, however, this one does not support the notion of methods associated with object types (classes). Tests for both identity and equality of two objects are allowed. Equality here means "deep equality"; i.e., an entire hierarchical structure is traversed, and all parts of it must be equal to the corresponding parts of the other.The algebra supports the following operators:

1)-3) Union (Union), Intersection (Intersect), and Difference (Subtract): Each of these takes 2 sets of object identifiers and produces a new (set-valued) object. Each is defined exactly like its set-theoretic counterpart, and equality of object identifiers (i.e., identity of the objects referred to) is used to test the set elements for equality.

4) Combining two objects (Combine): This operation is defined on any pair of objects such that either i) both are tuples, ii) both are sets, or iii) one is a set and the other a tuple. In case (i), the result is a new tuple object with all the fields of each input tuple. In case (ii), the result is computed by invoking Combine on each element of the Cartesian product of the inputs. In case (iii), Combine invokes itself on every pair (*t, e*), where *t* is the input tuple and *e* is an element of the input set.

5) Partitioning an object (Partition attr_list): The input to this operator must be either a tuple or a set. If it is a tuple, the result is a new tuple containing only the attributes in attr_list (or null if the tuple does not contain all the attributes in attr_list). If the input is a set, the output is a new set containing the result of applying Partition to every tuple in the set which contains all of the attributes in attr_list. Thus it can perform relational projections.

6) Predicates (Choose Pred): A predicate (Pred) consists of terms of the following form connected by $\wedge$ and $\neg$: obj IsNull, obj1 - obj2, obj1 $\equiv$ obj2 (identity test), and obj InClass C (where C is the name of some class). In the expression Choose Pred (obj), if obj is a set, the result is a new set containing (the identifiers of) all objects in the set for which Pred is true. Otherwise the result is obj if Pred applied to obj is true and null if it is false.

3. SUMMARY

We conclude the survey by summarizing the salient features of the algebras described above. Table 1 summarizes the NF^2 algebras and other database algebras presented in previous sections. For algebras the interesting features include any restrictions (e.g. normal forms) required of a database in order for the operations of the algebra to be applicable; whether or not the algebra allows for value-based or object-based operations or both; and an unannotated list of the algebraic operators provided.

4. CONCLUSION

In this paper we have seen many different ways of defining an algebra and notice the central nature of the relational algebra that is virtually all of the algebras are based on this algebra in some way with some algebraic extensions. Specifically, these algebraic extensions imply that the relational model falls short in the areas of modelling constructs (not rich enough), expressive power (cannot do, e.g., a general transitive closure), computational power (absence of aggregates, etc.), historical databases (i.e., we cannot express the query "what was true at time x?"), and object-based notions (which can help alleviate some of the anomalies encountered in 1NF relations). But for data more complex than NF^2 relations, a clearly favorite set of primitives has not yet emerged (indeed, only a handful of algebras for complex objects exist). More work remains to be done in this area.

5. REFERENCES

[1] A. Klug, "Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions", *Journal of the ACM*, Vol. 29, No. 3, 1982, pp. 699-717.

[2] C. Schaffert, T. Cooper, B. Bullis, M. Kilian, and C. Wilpot, "An Introduction to Trellis/Owl", *Proceedings of 1st OOPSLA Conference*, Portland, Oregon, 1986, pp. 9-16.

[3] E. Codd, "A Relational Model of Data for Large Shared Data Banks", *CACM*, Vol. 13, No.6, 1970, pp. 337-387.

[4] G. Jaeschke, G., "Recursive Algebra for Relations with Relation Valued Attributes", *Tech. Report 85.03.002*, IBM Heidelberg Scientific Center, 1985.

[5] H. J. Schek, "Towards a Basic Relational NF2 Algebra Processor", *Proceedings of International Conference on FODO,* Kyoto, Japan, 1985, pp. 173-182

[6] H. J. Schek, and M. Scholl, "The Relational Model with Relation-Valued Attributes", *Inform. Systems*, Vol. 11, No. 2, 1986, 137-147.

[7] H. Korth, and A. Silberschatz, "Database System Concepts", second edition, *McGraw-Hill*, New York, 1991.

[8] J. Ullman, J. "Principles of Database and Knowledge-Base Systems", Vol. 2., Computer Science Press, Rockville, Maryland, 1989.

[9] L. L. Dornhoff, and F. E. Hohn, "Applied Modern Algebra", *Macmillan*, New York, 1978.

[10] M. H. Scholl, "Theoretical Foundations of Algebraic Optimization Utilizing Unnormalized Relations", Proceedings *of International Conference on Database Theory*, Rome, 1986, pp. 380-396.

[11] M. Roth, H. Korth, and A. Silberschatz, "Extended Algebra and Calculus for 1NF Relational Databases", *Technical Report: CS-TR-84-36*, 1984.

[12] M. Roth, H. Korth, and D. Batory, "SQL/NF: A Query Language for 1NF Relational Databases", *Inform. Systems*, Vol. 12, No. 1, 1987, pp. 99-114.

[13] P. C. Fischer, and S. J. Thomas, "Operators for Non-First-Normal-Form Relations", Proceedings *of IEEE COMPSAC*, 1983, pp. 464-475.

[14] P. Dadam, K. Kuespert, F. Andersen, H. Blanken, R. Erbe, J. Guenauer, V. Lum, P. Pistor, and G. Walch, "A DBMS Prototype to Support Extended NF2 Relations: An Integrated View of Flat Tables and Hierarchies", *Proceedings of ACM SIGMOD Conference*, Washington, DC, 1986, pp. 356-367.

[15] P. Pistor, and F. Andersen, "Designing a Generalized NF2 Model with an SQL-Type Language Interface", *Proceedimgs*

of VLDB Conference, Kyoto, Japan, 1986, pp. 278-285.

[16] S. Abiteboul, and C. Beeri, "On the Power of Languages for the Manipulation of Complex Objects", *Tech. Report No. 846,* INRIA, 1988.

[17] S. Osborn, "Identity, Equality, and Query Optimization", *Lecture notes in computer science on Advances in object-oriented database systems*, **N**o. 334, Springer-Verlag, Berlin, Germany, 1988, pp. 341-356

[18] V. Deshpande, and D. Van Gucht, "An Implementation for Nested Relational Databases", *Proceedings of VLDB Conference*, Los Angeles, 1988, pp. 76-87.

[19] V. Deshpande, and P. A. Larson, "An Algebra for Nested Relations", *Research Report CS-87-65,* University of Waterloo, 1987.

Algebra	**Restrictions**	**Objects / Values**	**Operators**
DASDBS	None	Value based	x, ∪,-, σ, π,ν, μ
AIM	None	Both	x, ∪, -, σ, π,ν,μ, χ, ρ
Vanderbilt	None	Value based	x, ∪, -, σ, π, NEST, UNNEST, UNNEST*
SQL/NF	PNF	Value based	x, $\cup^e$, $-^e$, $\cap^e$, σ, π^e, $\|x\|^e$, ν, μ
NRDM	PNF	Value based	$\cup^e$, $-^e$, σ^e,π^e, $\|x\|$,ν,μ
An Algebra for Aggregates	None	Value based	Projection,Cross product, Union, Difference, Restriction, Aggregate formation
A Complex Object Algebra	None	Value based	∪, ∩, −, $cross_A$, rename A→B, $powerset_A$, set-collapse, ρ<G>(R)
An Object-Oriented Database Algebra	None	Object based	Union, Intersect, Subtract, Combine, Partition attr_list, Choose Pred

Table 1. Database algebras