International Journal of Computer and Information Sciences, Vol. 13, No. 3, 1984

Familial Model of Data

Levent Orman[1]

Received February 1983; revised January 1984

The family of sets is proposed as the basic structure for modeling data. A family is created by indexing one set of objects by another to represent a directed binary association between two sets. Familial models are shown to have a number of distinct advantages in supporting diverse user views through a hierarchy of abstractions and a variety of derived data, and in describing themselves and other data models through metamodels. An algebra of families is introduced to provide a data definition, maintenance and processing language that is minimal, intuitive, algebraic and theoretically sound. The language is extended to a specification language for database application systems, largely eliminating the need for embedding database constructs into procedural programming languages.

KEY WORDS: Data model; family of sets; conceptual model; metamodel; data language; system specification.

1. INTRODUCTION

1.1. Principal Objectives

The principal objective of the familial model of data is to provide a data definition, maintenance and processing language that is minimal, intuitive, algebraic and theoretically sound. The language is also envisioned to be a spcification language for database application systems, largely eliminating the need for embedding database constructs into procedural programming languages.

The main construct of the model is the family of sets, which is used to represent a directed binary association between two sets of objects. A family is created by indexing one set of objects by another to represent an

[1] Cornell University, School of BPA, Malott Hall, Ithaca, New York 14853.

0091-7036/84/0600-0149$03.50/0

association. The algebra of families developed to fulfill the requirements specified above can be characterized as follows:

a. Algebraic: The language is based on the algebra of sets.
b. Minimal: Set operations and arithmetic comparison operations combined with indexing constitute the data sublanguage.
c. Intuitive: The language lends itself to syntactic transformations into graph oriented or English-like languages.
d. Extensible: The language can be extended to a declarative specification language for database application systems by introducing arithmetic and generalized set operations.

The effectiveness of a data language in facilitating interaction between the users and the system is closely related to the effectiveness of data description facilities. Consequently, a second objective of the familial model is to provide a modeling capability based on families of sets. A data model based on families of sets can be characterized by the following:

a. Each data model is a collection of sets and families of sets. A set is a named collection of objects and it contains all objects playing a unique role specified by its name. The sets STUDENT and COURSE in a university database for example, contain the students and courses respectively.
b. A family of sets is created by partitioning a set through indexing and it is used to denote a relationship between two sets of objects. Given the sets STUDENT and COURSE described above, indexing STUDENT by COURSE creates a family of sets by partitioning STUDENT. Each set of the family contains the students enrolled in a particular course, and hence captures the enrollment relationship between the sets STUDENT and COURSE.
c. The data retrieval language consists of opertions defined on sets and families. Since families are partitions, the operations defined on families are particularly useful in dealing with classification and aggregation problems that are common in data processing.
d. Each model can be described by a metamodel consisting of two families of sets. One describes the membership of objects in sets, the other describes the relationships between sets.
e. A hierarchy of abstracts are supported by two types of abstraction mechanisms. Generalization is achieved through subset-superset relationships. For example, STUDENT, FACULTY, and ADMINISTRATOR sets of a university database can be generalized to a PERSON superset. Aggregation is achieved by

combining a number of relationships into a single one. For example, STUDENT-COURSE and COURSE-FACULTY relationships can be combined into a single relationship among STUDENT, COURSE, and FACULTY, using the operators of the language.

1.2. Background

The use of sets for modeling data was first proposed by Childs[1] and Codd.[2] The subsequently developed relational theory established good design practices and guidelines to eliminate redundancy and storage anomalies, and to capture the inherent structure of data, through a number of normal forms and other related concepts summarized in Ref. 3. An alternative approach by binary[4–6] and functional[7] models was aimed at developing natural and semantically rich models to facilitate user interaction and sharing. The adoption of the three level schema framework by ANSI[8] and the recognition of the need for a conceptual model stimulated research in this approach, since deferring the dependency and redundancy considerations to the internal model, and using small units of information as building blocks are desirable in a conceptual model.[9] A third approach to conceptual modeling of data based on record based structures has been eloquently dismissed by Kent.[10,11]

The familial model falls into the second category. It uses families of sets (to represent directed binary associations) as building blocks, and data dependencies are treated as ordinary integrity constraints as opposed to being the determinants of the data structure. At the same time, many of the ideas in the familial approach are similar to or have been adopted from the relational theory. The algebraic data sublanguage, familial algebra, plays a role similar to the relational algebra,[12] although it is most similar to the data sublanguage SQUARE.[13] The light pen version of the familial algebra has counterparts in graph oriented binary models, most notably FORAL-LP.[14] The table oriented language Query by Example[15] is also an attempt in the same direction. The English-like syntax and the use of indentation to increase readability have been previously used by SEQUEL.[16] The concept of abstraction in the familial model has been adopted from the generalization abstraction of Smith and Smith.[17] The metamodel approach to data description has been utilized in the design of System R,[18] Ingres,[19] and Extended Relational Model[20] formalized the concept and introduced a metamodel language.

The familial model differs from the mostly graph oriented binary models in its algebraic orientation. Similarly, it differs from the functional model because of the algebraic and nonprocedural nature of its data

sublanguage as opposed to the procedural nature of the functional language DAPLEX. Also, the extension of a data sublanguage into an algebraic specification language for database application systems is a unique characteristic of the familial model although some isolated attempts have been made to formalize the application system development.[21–23]

The remainder of this paper explains the familial approach to modeling data in detail with sections on data description, data retrieval, user orientation, database maintenance, metamodel and applications programming. No implementation of the familial algebra currently exists and the design should be considered preliminary. The applications programming environment is not intended to provide a complete pogramming language but a facility powerful enough for most commercial data processing problems. Additional constructs would be needed to extend it to a complete programming language.[24]

2. DATA DESCRIPTION

2.1. Families of Sets

A family F from an index set I to a target set A (also denoted by $A_F[I]$) is a collection of sets $(Ai: i \in I)$ where each Ai is a subset of the target set A, corresponding to an element i of the index set I.[25,26] Consequently, a family is characterized by two sets called the target set and the index set, and a correspondence between the two. Each Ai is called a member set of the family. Some members may be null; and the union of the members is not necessarily equal to the target. A family $A_F[I]$ can also be viewed as a (set valued) function from the index set I to the target set A, and denoted as $A[I]$ whenever the family involved is obvious from the context. A family F with multiple index sets $I_1,..., I_n$ is also a collection of sets $(Ai_1,..., i_n: i_1 \in I_1,..., i_n \in I_n)$, each corresponding to an element of the Cartesian product $I_1 \times I_2 \times \cdots \times I_n$. A family is a more general concept than a set of sets since it can contain duplicate sets and distinguish them through their association with different index values.

2.2. Modeling with Families

A familial model of data is a collection of named families. Each family represents a correspondence between one or more index sets and a target set. The families are named after their target sets whenever there is no ambiguity. Index sets denote sets of entities outside the system and contain system-generated unique identifiers for these entities as suggested in Ref. 27.

Example 2.1. A university environment with the entity sets STUDENT and COURSE can be modeled using the following families:

Family	Target	Index
SS#	SS#	STUDENT
DEPARTMENT	DEPARTMENT	STUDENT
NAME	NAME	STUDENT
COURSE#	COURSE#	COURSE
INSTRUCTOR	INSTRUCTOR	COURSE
DEPT	DEPARTMENT	COURSE
STUDENT	STUDENT	COURSE
GRADE	GRADE	STUDENT, COURSE

The semantics of the model is straightforward since each index set corresponds to an entity set, and each target set corresponds to an attribute. Each family represents a directed binary association between its index sets and its target.

3. DATA RETRIEVAL

3.1. Displaying Families

A family can be displayed in table form for human processing.

Example 3.1. Given $A = \{a_1, a_2\}$, $B = \{b_1, b_2\}$ and $B[A] = (Ba_1, Ba_2)$ where $Ba_1 = \{b_1, b_2\}$, $Ba_2 = \{b_2\}$. $B[A]$ is displayed as follows:

A	B
a_1	b_1
	b_2
a_2	b_2

Several families with common indices such as $B[A]$, $C[A]$, and $D[A]$ may be displayed as one table using the notation $B, C, D[A]$.

3.2. Families with Multiple Indices

A family $Z[Y]$ is said to be obtained by indexing the target Z by the index Y. In general, the target Z and the index Y may themselves be families; and indexing a family by another family produces a family containing all the sets created by indexing each element of the target by each element of the index. More formally, given two families $(Zi: i \in I)$ and $(Yj: j \in J)$, and another family $Z[Y]$ defined on the target sets of these families:

$$(Zi: i \in I)[(Yj: j \in J)] = (Ziyj \in Zi[Yj]: i \in I, j \in J)$$

Example 3.2. Given $B[A]$ as above; also given $C = \{c_1, c_2\}$ and $C[B] = (Cb_1, Cb_2)$ where $Cb_1 = \{c_2\}$ and $Cb_2 = \{c_1, c_2\}$. $C[B[A]]$ is obtained by indexing C by $B[A]$ and displayed as follows:

A	B	C
a_1	b_1	c_2
	b_2	c_1
		c_2
a_2	b_2	c_1

$Z[Y_1,..., Y_n]$ will be used to denote $Z[Y_n][Y_{n-1}],..., [Y_1]$, i.e. a family with multiple indices created by repetitively indexing Z by Y_n, $Y_{n-1},..., Y_1$.

Example 3.3. Given $B[A]$ and $C[B]$ as above; also given $D = \{d_1\}$ and $C[D] = (Cd_1) = (\{c_1\})$. $C[D; B[A]]$ is obtained by indexing C by $B[A]$, and further indexing the result by D as displayed below:

A	B	D	C
a_1	b_1	d_1	$\varnothing$
	b_2	d_1	c_1
a_2	b_2	d_1	c_1

where $\varnothing$ is the null set.

3.3. An Algebra of Families

An algebra of families consisting of set operations union ($\cup$), intersection ($\cap$) and difference ($\sim$), and comparison operations ($<, \leqslant, >, \geqslant, =, \neq$) in addition to indexing provides a subsetting language for data models based on families. Indexing as defined in section 3.2 is the major operation of the familial algebra. It is used to create complex families with multiple indices from a set of given families which themselves are defined by indexing sets. A special form of indexing involves the arithmetic comparison operators. Given two sets Y and Z and a comparison operator λ; $Z\lambda Y$ is defined in a similar manner to $Z[Y]$ as follows:

$$Z\lambda Y = (Zy\colon\ y \in Y)$$

where

$$Zy = \{z \in Z\colon\ z\lambda y\}$$

Example 3.4. Given $A = \{a_1, a_2\}$, $B = \{b_1, b_2\}$ as in Example 3.1. Let $a_1 = 5$, $a_2 = 3$, $b_1 = 6$, $b_2 = 4$; $B \geqslant A$ is a family of sets where each set

contains those elements of B greater than or equal to the corresponding element of A as displayed below:

A	B
5	6
3	6
	4

In general, an arithmetic comparison operator takes two families as arguments to produce a new family containing all the sets created by applying the operator to the individual elements of argument families. Given two families $(Zi: i \in I)$ and $(Yj: j \in J)$ and a comparison operator λ:

$$(Zi: i \in I)\,\lambda(Y_j: j \in J) = (Zi\gamma j \in Zi\lambda Yj: i \in I, j \in J)$$

Example 3.5. Given A and B as above; also given $C = \{c_1, c_2\}$ where $c_1 = 4$ and $c_2 = 6$, and $B[A] = (Ba_1, Ba_2)$ where $Ba_1 = \{b_1, b_2\}$ and $Ba_2 = \{b_2\}$; $C \geqslant B[A]$ is computed by comparing C to each element of $B[A]$ individually as displayed below:

A	B	C
5	6	6
	4	4
		6
3	4	4
		6

Note the similarity between this result and the result in Example 3.2; also note that the operators are executed in right to left order, i.e. they have long right scope.

The set operations union, intersection, and difference are an integral part of the familial algebra. When applied to sets, their set theoretic definitions are preserved; and they are generalized in two different ways to apply to families. A binary set operator union, intersection, or difference takes two families as arguments and produces a new family consisting of all the sets created by applying the operator to the individual elements of the argument families. Given two families $(Zi: i \in I)$ and $(Yj: j \in J)$ and a set operator γ:

$$(Zi:\ i \in I)\,\gamma(Yj:\ j \in J) = (Wij:\ i \in I, j \in J)$$

where

$$Wij = Zi\gamma Yj$$

Example 3.6. Given A, B, C, and $B[A]$ as above; also given $D = \{d_1\}$ and $C[B] = (Cb_1, Cb_2)$ where $Cb_1 = \{c_2\}$ and $Cb_2 = \{c_1, c_2\}$. $C[B[A]] \cup D$ is computed by taking the union of D with each element of $C[B[A]]$ and preserving the indices as displayed below:

A	B	
a_1	b_1	c_2
		d_1
	b_2	c_1
		c_2
		d_1
a_2	b_2	c_1
		c_2
		d_1

A unary set operator union or intersection takes one family as a right argument and produces another family by repetitively applying the operator to all sets with the same index values except for the leftmost index. This operation is also called aggregation over the leftmost index. Given a family $(Zi_1 \cdots i_n)$ and a set operator γ; a unary set operation is defined as follows:

$$\gamma(Zi_1 \cdots i_n) = (Zi_2 \cdots i_n)$$

where

$$Zi_2 \cdots i_n = \underset{i_1}{\gamma} Zi_1 i_2 \cdots i_n$$

Example 3.7. Given A, B and $B[A]$ as above;

$$\cup B[A] = (Ba_1, Ba_2) = \{b_1, b_2\} \cup \{b_2\} = \{b_1, b_2\}$$

Example 3.8. Given A, B, C, $B[A]$ and $C[B]$ as above:

$$\cap C[B[A]] = \cap(Cb_1a_1, Cb_2a_1, Cb_2a_2) = (Ca_1, Ca_2)$$

where $Ca_1 = Cb_1a_1 \cap Cb_2a_1$ and $Ca_2 = Cb_2a_2$ as displayed below:

A	C
a_1	c_2
a_2	c_1
	c_2

Note that the application of unary set operator always eliminates the leftmost index. Repetitive application of operators eventually eliminates all indices producing a simple set as in Example 3.7.

3.4. Variable Names

In a complex familial algebra expression the same set name may appear more than once. Since the algebra utilizes no variable names other than the set names, the situations where a set may play several different roles in an expression must be carefully distinguished from the situation where a set plays the same role in several places. More then one occurrence of a name in an expression requires the corresponding values to be the same (or null) at all times. This property is used in applying operators selectively to corresponding elements. The same set name can be used to play different roles only by using primed variables, i.e. X and X' may be used to distinguish two roles played by the set X in the same expression.

Example 3.9. Given A, B, C, and $B[A]$ as above; also given $C[A] = (Ca_1, Ca_2)$ where $Ca_1 = \{c_2\}$ and $Ca_2 = \{c_1\}$. $B[A] \cup C[A]$ is computed by selectively applying the union operator to elements with the same index value, and displayed as follows:

A	
a_1	b_1
	b_2
	c_2
a_2	b_2
	c_1

Example 3.10. Given A, B, C, $B[A]$ and $C[A]$ as above; $B[A] \cup C[A']$ is computed by treating A and A' as two independent roles played by the set A, and displayed as follows:

A'	A	
a_1	a_1	b_1
		b_2
		c_2
	a_2	b_2
		c_2
a_2	a_1	b_1
		b_2
		c_1
	a_2	b_2
		c_1

3.5. Algebra as a Data Sublanguage

The use of the familial algebra as a subsetting language is demonstrated below with some sample queries.

Example 3.11. Given the university environment of the Example 2.1, also given the inverses of all families as derived data (section 5); the following queries are first stated in English, then in familial algebra. The responses include the index values and would be displayed in table form:

a. Courses offered by the CS department:

$$\text{COURSE}[\text{DEPT} = \{\text{CS}\}]$$

b. Courses listed by department:

$$\text{COURSE}[\text{DEPT}]$$

c. Courses taught by instructor Smith in the MIS department:

$$\text{COURSE}[\text{INSTRUCTOR} = \{\text{SMITH}\}; \text{DEPT} = \{\text{MIS}\}]$$

d. Courses offered jointly by CS and MIS departments:

$$\cap\text{COURSE}[\text{DEPT} = \{\text{CS}, \text{MIS}\}] \quad \text{or}$$

$$\text{COURSE}[\text{DEPT} = \{\text{CS}\}] \cap \text{COURSE}[\text{DEPT}' = \{\text{MIS}\})$$

e. Instructors teaching at least one course in the CS department:

$$\cup\text{INSTRUCTOR}[\text{COURSE}[\text{DEPT} = \{\text{CS}\}]]$$

4. USER ORIENTATION

4.1. Requirements from a User Language

A database management system serves the diverse needs of many users. Different types of user interaction and languages may be appropriate for users with different needs and interests. Readability, ease of query construction, consistency, precision, conciseness, and intuitiveness may each appeal in different ways to different users of database languages. The familial algebra is concise, precise, and is based on a consistent theoretical framework. The price for conciseness and preciseness is paid in readability and intuitiveness as queries grow more complex. An immediate solution is the ability to break up complex queries into smaller, more easily comprehensible pieces. This ability will be achieved by introducing an assignment operator. In addition, for further improvement in readability and intuitiveness at the expense of conciseness, two new syntactical arrangements based on the familial algebra will be introduced.

4.2. Assignment

The assignment operator $\leftarrow$ is used to create sets and families from the existing ones. Given a family (Zij); $A \leftarrow (Zij)$ would create a new family (Aij) where $Aij = Zij$. Obviously, this operation would have the side effect of creating a new set A containing all the elements of Z involved in the family (Zij).

Example 4.1. Given A, B, C, $B[A]$, and $C[B]$ as in section 3; $Q \leftarrow \cap C[B[A]]$ would create a set Q and a family $Q[A]$ where $Q = \{c_1, c_2\}$, $Q[A] = (Qa_1, Qa_2)$ with $Qa_1 = \{c_2\}$ and $Qa_2 = \{c_1, c_2\}$. Comparison of this result with the result of example 3.8 is useful.

In some instances, it is useful to specify the indices of the new family explicitly to leave out some others. The unspecified indices are eliminated by applying the union operator to the result over those indices.

Example 4.2. Given A, B, C, $B[A]$, and $C[B]$ as above; $Q[A] \leftarrow C[B[A]]$ is equivalent to $Q \leftarrow \cup C[B[A]]$ and creates a set Q and a family $Q[A]$ where $Q = \{c_1, c_2\}$ and $Q[A] = (Qa_1, Qa_2)$ with $Qa_1 = Qa_2 = \{c_1, c_2\}$.

Note that the assignment operator is very effective in summarizing the result of a complex familial algebra expression as a simple family by eliminating all the intermediate sets used in forming the expression. Assignment is also instrumental in database mdofification (see section 5).

4.3. Syntactic Variations

To further improve readability and simplify the query construction, two new version of the familial algebra each with syntactical modifications to the original, will be introduced.

4.3.1. A Light-Pen Syntax

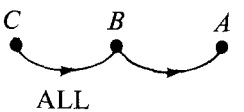
The first version involves replacing the unary set operators union and intersection with the English words SOME and ALL respectively and moving them next to the variables they apply to. This process brings the algebra closer to the grammatical structure of English making it more intuitive, but meanwhile destroying the ability to sequence operators in order of application, hence requiring parentheses to establish precedence. This version is espcially suitable for a graphic implementation to be used with a light pen where set names appear on the screen as nodes of a network and the user simply connects the nodes with directed arcs to formulate a query. Each arc connects a target to its index set and it is labeled with one of the keywords SOME or ALL or comparison operators. New nodes for constants

or new variables are created by simply typing them at the bottom of the screen.

Example 4.3. The following algebraic expressions are stated in both versions and as a graph for comparison:

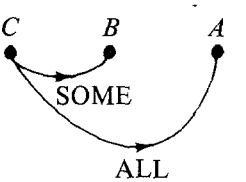
$$\cap C[B[A]] \tag{1}$$

$$C[\text{ALL } B[A]] \tag{1'}$$



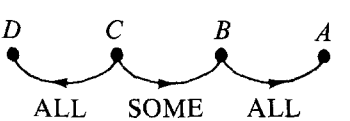
$$\cap\cup C[B; A] \tag{2}$$

$$C[\text{SOME } B; \text{ALL } A] \tag{2'}$$



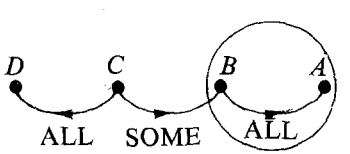
$$\cap\cup\cap C[D; B[A]] \tag{3}$$

$$C[\text{ALL } D; \text{SOME } B[\text{ALL } A]] \tag{3'}$$



$$\cup\cap C[D; \cap B[A]] \tag{4}$$

$$C[\text{ALL } D; \text{SOME}(B[\text{ALL } A])] \tag{4'}$$



Note that circles in the light pen syntax correspond to parentheses in algebra, i.e. they establish the precedence of the circled expression. Parentheses in Eq. 4′ have to be used to enforce the priority of the right-most intersection operation, which was automatic in the original algebra. This can easily be seen by comparing Eqs. 3 and 4.

4.3.2. An English-like Syntax

The last version to be introduced goes another step further in improving the readability and intuitiveness by utilizing visual aids such as indentation and a multiline structure at the expense of further reduction in conciseness. The expressions in this version are very close to English expressions and they can actually be completed to English sentences by using filler clauses, later to be ignored by the interpreter. The following characteristics distinguish this version from the previous one:

a. A multiline structure is employed where each variable name is placed on a different line.

b. The index of a family is indented with respect to the target of the family.

c. New key words such as EACH for intersection, and THE, A and AT LEAST ONE for union are utilized to express subtle differences in context. These subtle differences are already captured by the algebra in the sequence of operations, hence the key words will all be treated the same by the interpreter; however, the human users benefit from this more English-like expression of the contextual differences. The choice of EACH instead of ALL, for example, may be quite revealing in an English sentence as shown in the following examples.

d. English expressions can be used as fillers between variable names to complete sentences. These fillers are displayed in small print in the following examples and they will be ignored by the intrepreter.

e. Curly brackets around sets are dropped since constants cannot be confused with set names when only one set name is allowed per line.

Example 4.4. Given a university database containing the following families and their inverses:

NAME, DEPT[STUDENT]
NAME, RANK, PH#, DEPT[INSTRUCTOR]
STUDENT, INSTRUCTOR, COURSE#, DEPT[COURSE]

The following sample queries are expressed in English and all three versions of the familial algebra to provide a comparison and to demonstrate the use of the algebra for database querying.

a. Instructors in MIS department.

INSTRUCTOR[DEPT = {MIS}]
INSTRUCTORs in
DEPT = MIS

b. Instructors listed by department.

INSTRUCTOR[DEPT]
INSTRUCTORs listed by
DEPT

c. Instructors in the MIS department with the rank of professor.

INSTRUCTOR[DEPT = {MIS}; RANK = {PROFESSOR}
INSTRUCTORs in
DEPT = MIS with
RANK = PROFESSOR

d. Instructors teaching a course in the MIS department.

$\cup$ INSTRUCTOR[COURSE[DEPT = {MIS}]]
INSTRUCTOR[SOME COURSE[DEPT = {MIS}]]
INSTRUCTORs teaching
A COURSE in
DEPT = MIS

e. Students taking all the courses offered by the MIS department.

$\cap$STUDENT[COURSE[DEPT = {MIS}]]
STUDENT[ALL COURSE[DEPT = {MIS}]]
STUDENTs taking
ALL COURSEs offered by
DEPT = MIS

f. Students taking at least one course from each department.

$\cap\cup$STUDENT[COURSE[DEPT]]
STUDENT[SOME COURSE[ALL DEPT]]
STUDENTs taking
AT LEAST ONE COURSE from
EACH DEPT

g. Students taking a course listed by all departments.

$\cup$STUDENT[$\cap$COURSE[DEPT]]

STUDENT[SOME(COURSE[ALL DEPT])]

STUDENTs taking

A (COURSE listed by

ALL DEPT)

h. The departments of the students taking courses from the MIS department.

$\cup\cup$DEPT[STUDENT[COURSE[DEPT′ = {MIS}]]]

DEPT[SOME STUDENT[SOME COURSE[DEPT′ = {MIS}]]]

DEPTs of

THE STUDENTs taking

SOME COURSEs from

DEPT′ = MIS

i. Loyal students are the students taking at least one course from their own department.

LOYAL STUDENT←$\cup\cup$STUDENT[COURSE[DEPT[STUDENT]]]

LOYAL STUDENT←STUDENT[SOME COURSE[SOME

DEPT[STUDENT]]]

LOYAL STUDENT←

STUDENTs taking

AT LEAST ONE COURSE from

THE DEPT of the same

STUDENT

j. Students taking no course from their own departments.

STUDENT ~ LOYAL STUDENT

STUDENT ~

LOYAL STUDENT

5. DATABASE MAINTENANCE

5.1. Null Sets

A null set $\varnothing$ is a set with no elements. A null set of employees is no different from a null set of courses except for its name. A family may contain a null set if a particular index object has no corresponding objects in

the target set. This situation arises when an attribute does not apply to a particular object in a set. Since the uniformity of the objects in a set in terms of the relationships they participate in is important in capturing semantics, the null sets are not allowed in the declared families. This assumption has important consequences in database maintenance. Obviously, the null set is indispensable in derived data as demonstrated in section 3.

5.2. Insertion and Deletion

Insert $\downarrow$ and delete $\uparrow$ operations are used to insert or delete entities (i.e. internally generated surrogates corresponding to entities). Each operator takes a family with a single set as a right argument and inserts a new entity into that set, or deletes all the entities of that set with proper adjustments to the indices and to the sets defining the family.

Example 5.1. Given the university database containing the families SS#, NAME, YEAR[STUDENT], and their inverses; $\downarrow$STUDENT[YEAR = {JUNIOR}] would add a new student to the set of juniors by including the new student in the specified set. Obviously, the new student would also be included in the set STUDENT, and the value 'JUNIOR' would be added to the set YEAR if it is a new value. The SS# and the name of the new student would be defined as 'unknown' since every object in the set STUDENT is required to have corresponding objects in the sets SS#, NAME and YEAR. This follows the null set restrictions of section 5.1. Note that the inverse STUDENT[YEAR] of the family YEAR[STUDENT] should be defined to insert a student with a given year.

Example 5.2. Given the same database:

$$\uparrow \text{STUDENT}[\text{SS\#} = \{321587744\}; \text{NAME} = \{\text{SMITH}\}]$$

would delete the set of students identified above from the set STUDENT and from all families they participate in. As a side effect, all objects related (through some family) to a deleted object and to no other object are also deleted. This requirement follows from the effort to eliminate the null sets and the dangling objects (i.e. objects with no relationships to other objects) from the declared data. This requirement is natural in a set theoretic model where the sets contain the currently active objects (extension) rather than all possible objects (intension) as in types or domains. Under this requirement the SS# 321587744 will be deleted from SS# as a side effect of the above operation unless it is also the SS# of another student!

5.3. Modification

Modification $\leftarrow$ (or assignment) is used to modify the existing values, sets and families, in addition to creating new sets and families as described

in section 4.2. This is accomplished by replacing every set of the family on the left with the corresponding set on the right hand side. Some sets on the right may correspond and replace many sets on the left in case some indices of the left argument do not have corresponding indices on the right. The reverse situation (i.e. some indices on the right with no corresponding indices on the left) leads to aggregation over and elimination of those indices by applying the union operator as explained in section 4.2.

Example 5.2. Given the same database as above:

$$\text{YEAR}[\text{STUDENT}[\text{SS\#} = \{321587744\}]] \leftarrow \{\text{JUNIOR}\}$$

would change current standing of the specified student to 'junior.'

Example 5.3. Given the same database:

$$\text{YEAR}[\text{STUDENT}[\text{YEAR} = \{\text{JUNIOR}\}]] \leftarrow \{\text{SENIOR}\}$$

would change all juniors to seniors.

5.4. Derived Data

All sets and families created using the assignment operator of section 4.2 constitute the derived data. They can be used in the same way as the declared data except that their extension changes as the declared data is modified since they are defined in terms of the declared data. The data derived by a user may be restricted to the individual workspace as temporary variables or stored permanently in the model for later use.

Inversion operator i is used exclusively for deriving inverse families from the declared families. The careful reader might have observed that in the previous examples the inverse families have been assumed to exist without actually declaring them. The inverse of a family can be derived using the i operator that takes a family as an argument and returns the index objects corresponding to each distinct target object.

Example 5.4. The inverse STUDENT[YEAR] of the family YEAR[STUDENT] can be derived as follows:

$$\text{STUDENT}[\text{YEAR}] \leftarrow i\,\text{YEAR}[\text{STUDENT}]$$

5.5. User Views

A user view in a familial model is a subset of the model with no restructuring. It is used to mask part of the model from a user to make it more managable. It may contain both declared and derived data as long as they

are identified as such. In addition, a user may derive data using the families in his own view. The data derived by a user is local and temporary until it is explicitly stored permanently.

A major problem with user views in all data models is updating the derived data in a view without the full knowledge of the underlying declared data. In general, the derived data cannot be updated without additional information about the underlying declared data.[28] The familial model attacks this problem by requiring that each user view contain all the declared data relevant to that view and by restricting the update operations to the declared data. Consequently, a user view cannot simply consist of derived data, but all derived data in view must be derivable from the declared data included in that view. This approach does not produce unmanagable views as it would in network and relational models, since the familial models has the smallest unit of information as its building block (i.e. directed binary association) and the views are naturally subsets of the model with no restructuring.

Example 5.5. Given the university environment of Example 4.4, a user view including a direct relationship NAME[COURSE#] between the course numbers and the names of instructors of those courses would also be required to contain the underlying families NAME[INSTRUCTOR], INSTRUCTOR[COURSE], and COURSE[COURSE#] since

$$\text{NAME[COURSE\#]} \leftarrow \text{NAME[INSTRUCTOR[COURSE[COURSE\#]]]}$$

Note that COURSE[COURSE#] itself is a family derived from COURSE#[COURSE], which should also be included in the view. It is easy to show that although NAME[COURSE#] is a very useful family for querying the database without involving any internal identifiers, it would lead to ambiguities when updated. NAME[COURSE# = {CS101}] ← {SMITH} for example may be interpreted to modify the name of an instructor using NAME[INSTRUCTOR], the instructor of a course using INSTRUCTOR[COURSE] or the course# of a course using COURSE#[COURSE]. It is of course possible to instruct the system to select one of these interpretations consistently by sacrificing generality[7,28]; but in general a user has to see the underlying structure to update a database.

6. METAMODEL

6.1. Set Relationships

A model description consisting of a listing of families is not complete since some objects may play multiple roles by participating in multiple sets. Consider the introduction of the families

SS#, SALARY[EMPLOYE] and
YEARS OF SERVICE, TITLE, DEPT[STAFF]

into the university database of Example 4.4. It is not evident from the listing of families that each member of INSTRUCTOR is also a member of EMPLOYEE (i.e. INSTRUCTOR $\subseteq$ EMPLOYEE) and consequently INSTRUCTOR is related to SS# and SALARY in addition to its own attributes; nor is it evident that EMPLOYEE $\simeq$ INSTRUCTOR $\cup$ STAFF (where $\simeq$ denotes set equality). Moreover the listing of families does not reflect the fact that some instructors may also hold staff positions (i.e. intersecting sets).

Obviously, a complete description of a model should include set relationships unless all sets of the model are assumed to be disjoint. This assumption is harmful (although common in current commercial systems) since allowing some sets to be subsets of others leads to effective abstraction where lower level objects (such as INSTRUCTOR and STAFF) are combined and named as sets to produce higher level objects (such as EMPLOYEE). Separating and hiding lower level objects from the users of the higher level objects results in increased manageability. The explicit set relationships are also important in recognizing the multiple roles played by some objects and enforcing integrity constraints in a consistent manner. Consequently a complete data description should include subset-superset relationships, intersecting sets, and equivalent sets, in addition to index-target relationships.

6.2. Metamodel Description and Manipulation

A complete description of a familial model can be included in the model itself as a metamodel with two families, each capturing one of the two basic conditions defining a familial model:

a. the membership condition that links objects to sets is captured by the family SETNAME[OBJECT]
b. the indexing condition that links the index objects to their targets is captured by the family TARGET[INDEX]

SETNAME[OBJECT] captures the membership condition by specifying the sets containing each object where SETNAME contains the names of all sets in the model and OBJECT contains all objects of the model including surrogates. Note that OBJECT contains the objects themselves, not their surrogates. This is the only difference between the model and the metamodel and is triggered by the fact that the metamodel deals with the objects of the model that are already in the system, as opposed to the model dealing with

real life objects outside the system representable in the model only by surrogates. INDEX and TARGET are subsets of OBJECT. TARGET[INDEX] captures the indexing condition by specifying the target objects linked to each index object hence describing the roles (surrogate or value) the objects play with respect to each other. In general, INDEX and TARGET intersect and their union equals OBJECT.

The families of the metamodel are minimal, complete, and conceptually elegant, and many useful families may be derived from them. Some useful families indicating subset-superset, attribute-entity, and multiple role relationships are derived as follows:

a. Supersets of a set X are the sets containing all the elements of X.

$$\text{SUPERSET}[\text{SET}] \leftarrow \cap\text{SET}'[\text{OBJECT}[\text{SET}]]$$

b. Subsets of a set X are the sets which have X as a superset.

$$\text{SUBSET}[\text{SUPERSET}[\leftarrow i\,\text{SUPERSET}[\text{SET}]$$

c. Multiple roles played by some elements of a set lead to intersecting sets. The sets intersecting a set X are the sets that have at least one common element with X.

$$\text{INTERSECTINGSET}[\text{SET}] \leftarrow \cup\text{SET}'[\text{OBJECT}[\text{SET}]]$$

Note that subsets and supersets of X are also intersecting sets of X.

7. APPLICATIONS PROGRAMMING

7.1. Database Application Systems

Raw data is rarely useful in modern organizations. A variety of algorithms may be employed for analyzing and processing data before it is usable. These algorithms are written using high level procedural programming languages as apposed to the nonprocedural and algebraic nature of data sublanguages such as familial algebra and relational algebra. Interfacing nonprocedural data sublanguages with procedural programming languages have been attempted[29–32] and found to be less than satisfactory in studies by Stonebraker[33] and Prenner[34] because of basic incompatibilities between procedural and nonprocedural languages. The dichotomy between data retrieval and algorithmic processing has also effected the organizational structure and use of information in organizations.[35] The familial algebra lends itself to extension into an algebraic programming language for database applications by introducing additional primitive

operations on families, hence eliminating all the disadvantages of interfacing to procedural language. Some of these operations are introduced in this section; others may be defined in a similar fashion see (Ref. 24).

7.2. Counting

The 'count' operator ρ is used to compute the cardinality of sets. It returns the cardinality of a set as a singleton whose only element is an integer. When applied to families, it returns a family of singletons each of which is the count of the corresponding set.

Example 7.1. The following queries involve the count operator:

a. Number of students.

$$\rho \text{ STUDENT}$$

b. Number of courses each student is taking.

$$\rho \text{ COURSE[STUDENT]}$$

7.3. Arithmetic Operations

Addition (+), Subtraction (−), Multiplication (×), Division (/),- Exponentiation (*), Maximum (Γ), and Minimum (L) preserve their common definitions when applied to singletons by treating the single element of the singleton as a scalar, hence returning another singleton.

Example 7.2.

a. $\{3\} + \{4\}$ returns $\{7\}$

b. $\{3\} \Gamma \{4\}$ returns $\{4\}$

When applied to families of singletons, they return a family of singletons created by applying the operator to the individual elements of the argument families. Given two families $(Zi: i \in I)$ and $(Yj: j \in J)$ and an arithmetic operator α:

$$(Zi: i \in I)\ \alpha\ (Yj: j \in J) = (Wij: i \in I, j \in J)$$

where

$$Wij = Zi\ \alpha\ Yj$$

Note that the variable name restrictions of section 3.4 still apply to establish correspondence between indices.

Example 7.3. Given $A = \{a_1, a_2\}$, $B = \{b_1, b_2\}$, $C = \{c_1, c_2\}$, $B[A] = (Ba_1, Ba_2)$, and $C[A] = (Ca_1, Ca_2)$ where $Ba_1 = \{b_1\}$, $Ba_2 = \{b_2\}$, $Ca_1 = \{c_1\}$ and $Ca_2 = \{c_2\}$. $B[A] + C[A]$ is computed by selectively applying the operator $+$ to the singletons with the same index value, and displayed as follows:

A	
a_1	$b_1 + c_1$
a_2	$b_2 + c_2$

Example 7.4. Given the same families: $B[A] + C[A']$ is computed by applying the $+$ operator to all pairs of singletons, and displayed as follows:

A'	A	
a_1	a_1	$b_1 + c_1$
	a_2	$b_2 + c_1$
a_2	a_1	$b_1 + c_2$
	a_2	$b_2 + c_2$

The symmetric arithmetic operators $+$, $\times$, Γ, and L also have unary versions. They take one family of singletons as a right argument and create another family of singletons by repetitively applying the operators to all singletons with the same index values except for the leftmost index. This operation is also called arithmetic agrregation over the leftmost index and it eliminates the leftmost index. Given a family of singletons $(Zi_1 \cdots i_n)$ and a symmetric arithmetic operator α:

$$\alpha(Zi_1 \cdots i_n) = (Zi_2 \cdots i_n)$$

where

$$Zi_2 \cdots i_n = \underset{i_1}{\alpha} Zi_1 \cdots i_n$$

The similarities between the definitions of the set operations of section 3.3 and the arithmetic operations are interesting to observe where the only major difference is the restriction of the arithmetic operations to families of singletons.

Example 7.3. Given A, B, C, and $C[A]$ as above; also given $C[B] = (Cb_1, Cb_2)$ where $Cb_1 = Cb_2 = \{c_1, c_2\}$. $+C[B; A]$ is computed by summing Cb_ia_i values over b_i as displayed below:

A	C
a_1	$c_1 + c_1$
a_2	$c_2 + c_2$

Since $Cb_1a_1 = Cb_2a_1 = \{c_1\}$ and $Cb_2a_1 = Cb_2a_2 = \{c_2\}$.

Example 7.4. Given the university environment of section 6.1, the following queries involving arithmetic operations are expressed in English and in algebra:

a. Total salary expense of the university.

$$+\text{SALARY}[\text{EMPLOYEE}]$$

b. Total salary expense for instructors in each department.

$$+\text{SALARY}[\text{INSTRUCTOR}[\text{DEPT}]$$

c. The difference between the average salaries of instructors and staff.

$$\text{INSAVERAGE} \leftarrow +\text{SALARY}[\text{INSTRUCTOR}]/\rho\ \text{INSTRUCTOR}$$
$$\text{STAFFAVERAGE} \leftarrow +\text{SALARY}[\text{STAFF}]/\rho\ \text{STAFF}$$
$$\text{INSAVERAGE} - \text{STAFFAVERAGE}$$

d. The difference between the average salaries of instructor and staff in each department.

$$\text{INSAVERAGE}[\text{DEPT}] \leftarrow +\text{SALARY}[\text{INSTRUCTOR}[\text{DEPT}]]/\rho\ \text{INSTRUCTOR}[\text{DEPT}]$$
$$\text{STAFFAVERAGE}[\text{DEPT}] \leftarrow +\text{SALARY}[\text{STAFF}[\text{DEPT}]]/\rho\ \text{STAFF}[\text{DEPT}]$$
$$\text{INSAVERAGE}[\text{DEPT}] - \text{STAFFAVERAGE}[\text{DEPT}]$$

7.4. Inner and Outer Products

The familial algebra accommodates inner and outer products without introducing additional primitives, but by using the arithmetic operations and the ability to apply these operators selectively by matching index variables as explained in section 3.4.

Example 7.5. Given the same university environment with the following families

$$\text{COURSE, STUDENT, GRADE}[\text{ENROLLMENT}]$$
$$\text{CREDIT}[\text{COURSE}]$$
$$\text{POINT}[\text{GRADE}]$$

captures the information about enrollment, credit value of each course and the point value of each letter grade. The GPA of each student can be computed as follows:

POINT[COURSE; STUDENT] ← POINT[GRADE[ENROLLMENT [COURSE; STUDENT]]]

COURSEPOINT[COURSE; STUDENT] ← CREDIT[COURSE] × POINT[COURSE; STUDENT]

TOTALPOINT[STUDENT] ← + COURSEPOINT[COURSE [STUDENT]; STUDENT]

TOTALCREDIT[STUDENT] ← + CREDIT[COURSE[STUDENT]]

GPA[STUDENT] ← TOTALPOINT[STUDENT]/TOTALCREDIT [STUDENT]

Note that the second and the third lines above constitute an inner product of credit and point values along the course index. It is easier to see the inner product by visualizing CREDIT[COURSE] as a one dimensional array where each element corresponds to a value in COURSE, and visualizing POINT[COURSE; STUDENT] as a two dimensional array where rows correspond to courses and columns correspond to students.

8. CONCLUSION

The familial model of data has the following principal characteristics:

a. The building blocks of the model are sets and families of sets.

b. The existing theory of sets and families is utilized to design a nonprocedural data sublanguage based on set algebra.

c. The familial model is minimal in terms of the number of data constructs and the number of algebraic operations defined on them.

d. The familial algebra as a data sublanguage, is concise and intuitive, and lends itself to syntactic transformations into graph oriented or English-like languages.

e. The familial model can distinguish duplicate elements through their association with different index values. This capability leads to the ability to count and do arithmetic. The familial algebra is extended to a complete declarative specification language for database applications by introducing arithmetic and generalized set operations.

f. Maintenance operations are also incorporated into the familial algebra.

g. A family represents a directed binary association between two sets,

which is the smallest unit of information in a data model. Having the smallest unit as a building block has the following advantages:

1. The model can support user views without any restructuring, but by simply selecting the relevant families. Consequently, creating and maintaining user views becomes a more systematic task. This capability is important for a conceptual model in a multi-level schema environment.
2. Other data models with larger building blocks can be described in and accessed by a familial environment. This capability is important in facilitating transition from existing data models to a familial framework and in supporting a heterogeneous multi-database environment.

h. A metamodel approach to data description is taken. The metamodel can be described and manipulated in the same environment, by the same familial algebra. No additional constructs or operations are required for the metamodel.

i. Sets of the model are not required to be disjoint. Consequently:
   1. Multiple roles played by an entity can be represented consistently and without redundancy.
   2. Subset-superset and membership conditions are used for effective abstraction in a set theoretic sense.

REFERENCES

1. D. L. Childs, Feasibility of a Set Theoretical Data Structure—A General Structure Based on a Reconstituted Definition of Relation. Proc. IFIP Congress, pp. 162–172 (1968).
2. E. F. Codd, A Relational Model of Data for Large Shared Data Banks, *Comm. ACM* **13**(6):377–387 (1970).
3. J. D. Ullman, Principles of Database Systems. Computer Science Press, Maryland (1980).
4. J. R. Abrial, Data Semantics. In *Database Management*, J. W. Klimbie and K. L. Koffeman (eds.), North Holland Publising Co., Amsterdam (1974).
5. G. Bracchi, P. Paolini, and G. Pelagatti, Binary Logical Associations in Data Modeling. In *Modeling in Database Management Systems*, G. M. Nijssen (ed.), North Holland Publishing Co., Amsterdam (1976).
6. M. E. Senko, DIAM as a Detailed Example of the ANSI/SPARC Architecture. In *Modeling in Database Management Systems*, G. M. Nijssen (ed.), North Holland Publishing Co., Amsterdam (1976).
7. D. W. Shipman, The Functional Data Model and the Data Language DAPLEX, *ACM Transactions in Database Syst.* **6**(1):140–173 (March 1981).
8. ANSI/X3/SPARC Study Group on Database Management Systems. Interim report (February 1975); also FDT, ACM-SIGMOD, 7(1):1–26 (1975).

9. G. M. Nijssen, A Gross Architecture for the Next Generation Database Management Systems. In *Modeling Database Management Systems*, Proc. IFIP TC2 Working Conference, Freuderstadt, G. M. Nijssen (ed.), North Holland Publishing Co., Amsterdam (1976).
10. W. Kent, Data and Reality, North Holland Publishing Co., Amsterdam (1978).
11. W. Kent, Limitations of Record-Based Information Models, *ACM Transactions on Database Systems* **4**(1):107–131 (1979).
12. E. F. Codd, Relational Completeness of Database Sublanguages. Courant Computer Science Symposia, Vol. 6, in *Database Systems*, R. Rustin (ed.), Prentice Hall, Englewood Cliffs, New Jersey (1972).
13. R. F. Boyce, D. D. Chamberlin, F. W. King, and M. M. Hammer, III, Specifying Queries as Relational Expressions: The SQUARE Data Sublanguage, *Comm. ACM* **18**(11):521–628 (1975).
14. M. E. Senko, A Query-Maintenance Language for the Data Independent Accessing Model II, *Inform. Syst.* **5**(4):257–272 (1980).
15. M. M. Zloof, Query by Example: A Database Language, *IBM Systems Journal* **16**(4):324–343 (1977).
16. D. D. Chamberlin and R. F. Boyce, SEQUEL: A Structured English Query Language, Proc. Conf. on Management of Data (1974).
17. J. M. Smith and D. C. P. Smith, Database Abstractions: Aggregation and Generalization, *ACM Transactions on Database Syst.* **2**(2):105–133 (1977).
18. M. M. Astrahan *et al.*, System R: Relational Approach to Database Management, *ACM Transactions on Database Systems* **1**(2):97–137 (June 1976).
19. M. Stonebraker *et al.*, The Design and Implementation of INGRES, *ACM Transactions on Database Syst.* **1**(3):189–222 (September 1976).
20. E. F. Codd, Extending the Database Relational Model to Capture More Meaning, *ACM Transaction on Database Systems*, **4**(4):397–434 (1979).
21. R. Bosak *et al.*, An Information Algebra, *Comm. ACM* **5**(4):190–204 (April 1962).
22. M. Hammer *et al.*, A Very High Level Language for Data Processing Applications, *Comm. ACM* **20**(11):832–840 (November 1977).
23. L. Orman, An Array Theoretic Specification Environment for the Design of Decision Support Systems, *Policy Anal. and Infor. Syst.* **6**(4):373–391 (1982).
24. L. Orman, A Familial Specification Language for Database Application Systems, *Computer Languages*, **8**(3):113–124 (1983).
25. C. Berge, Topological Spaces, MacMillan Co., New York (1963).
26. P. R. Halmos, Naive Set Theory, Van Nostrand Publishing Co., New York (1960).
27. P. A. V. Hall, J. Owlett, and S. J. P. Todd, Relations and Entities. In *Modeling Database Management Systems*, G. M. Nijssen (ed.), North Holland (1976).
28. U. Dayal and P. H. Bernstein, On the Updatability of the Relational Views. Proc. 4th Int. Conf. Very Large Database, pp. 368–377, Berlin (September 1978).
29. E. Allman, M. R. Stonebraker, and G. Held, Embedding a Relational Data Sublanguage in a General Purpose Programming Language, *ACM SIGPLAN Notices* **11**:25–35 (March 1976).
30. D. D. Chamberlin *et al.*, SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control, *IBM J. of Res. and Develop.* **20**(6):560–575 (November 1976).
31. L. Orman, An Information Base for Procedure Independent Design of Information Systems, Proc. AFIPS National Computer Conference, pp. 817–821 (1980).
32. J. W. Schmidt, Some High Level Constructs for the Data of Type Relation, *ACM Transactions on Database Syst.* **2**(3):247–261 (1977).

33. M. R. Stonebraker and L. A. Rowe, Observations on Data Manipulation Languages and Their Embedding in General Purpose Programming Languages, Proc. Very Large Databases (1977).
34. C. J. Prenner and A. R. Lawrence, Programming Languages for Relational Database Systems, Proc. AFIPS National Computer Conference, pp. 849–855 (1978).
35. S. L. Alter, A Study of Computer-aided Decision Making in Organizations, Ph.D. Dissertation, Massachusetts Institute of Technology (June 1975).