987

RELATIONAL COMPLETENESS OF DATA BASE SUBLANGUAGES

by

E. F. Codd

IBM Research Laboratory
San Jose, California

ABSTRACT: In the near future, we can expect a great variety of languages to be proposed for interrogating and updating data bases. This paper attempts to provide a theoretical basis which may be used to determine how complete a selection capability is provided in a proposed data sublanguage independently of any host language in which the sublanguage may be embedded.

A relational algebra and a relational calculus are defined. Then, an algorithm is presented for reducing an arbitrary relation-defining expression (based on the calculus) into a semantically equivalent expression of the relational algebra.

Finally, some opinions are stated regarding the relative merits of calculus-oriented versus algebra-oriented data sublanguages from the standpoint of optimal search and highly discriminating authorization schemes.

RJ 987 (#17041)
March 6, 1972
Computer Sciences

1. INTRODUCTION

In the near future we can expect a great variety of languages to be proposed for interrogating and updating data bases. When the computation-oriented components of such a language are removed, we refer to the remaining storage and retrieval oriented sublanguage as a data sublanguage. A data sublanguage may be embedded in a general purpose programming language, or it may be stand-alone -- in which case, it is commonly called a query language (even though it may contain provision for simple updating as well as querying).

This paper attempts to establish a theoretical basis which may be used to determine how complete a selection capability is provided in a proposed data sublanguage independently of any host language in which the sublanguage may be embedded. The selection capability under discussion is a basic non-statistical one. In a practical environment it would need to be augmented by a counting and summing capability, together with the capability of invoking any one of a finite set of library functions tailored to that environment.

In previous papers[1,2] we proposed a relational model of data. With this model any formatted data base is viewed as a collection of time-varying relations of assorted degrees. In Section 2 of this paper, we define a collection of operations on relations, and this collection is called a relational algebra. This algebra may be used for a variety of purposes. For example, a query language could be directly based on it (however, we would propose that domain names be used instead of domain numbers). Later sections of this paper support its use as a yardstick of selective power of algebra-oriented data sublanguages. For each such language, one would investigate whether there is any operation of the relational algebra which cannot be defined in the candidate language.

An information algebra with a rather different orientation was proposed by R. Bosak.[5]

In Section 3, we define an applied predicate calculus--the relational calculus-- and introduce a number of concepts related to meaningful and reasonable queries. These concepts are somewhat related to those of J. L. Kuhns.[6,7,8] However, his relational data file is less structured than this author's relational model.

A data sublanguage (called ALPHA), founded directly on the relational calculus, has been informally described in [4]. This calculus can also be used as a means of comparing calculus-oriented data sublanguages with one another.

In Section 4, we provide an algorithm for translating an arbitrary alpha expression into a semantically equivalent sequence of operations in the relational algebra. This algorithm demonstrates that the relational algebra has at least the selective power of the relational calculus. As a consequence, we may shortcut the work of comparing any given algebra-oriented data sublanguage L_A with any given calculus-oriented language L_C by comparing L_A with the relational algebra and L_C with the relational calculus, as indicated in Fig. 1. In this figure, the Project MAC system called MACAIMS is cited as an example of an algebra-oriented language (see [9]).

In section 5, we consider the pros and cons of data sublanguages founded on a relational algebra versus those founded on a relational calculus.

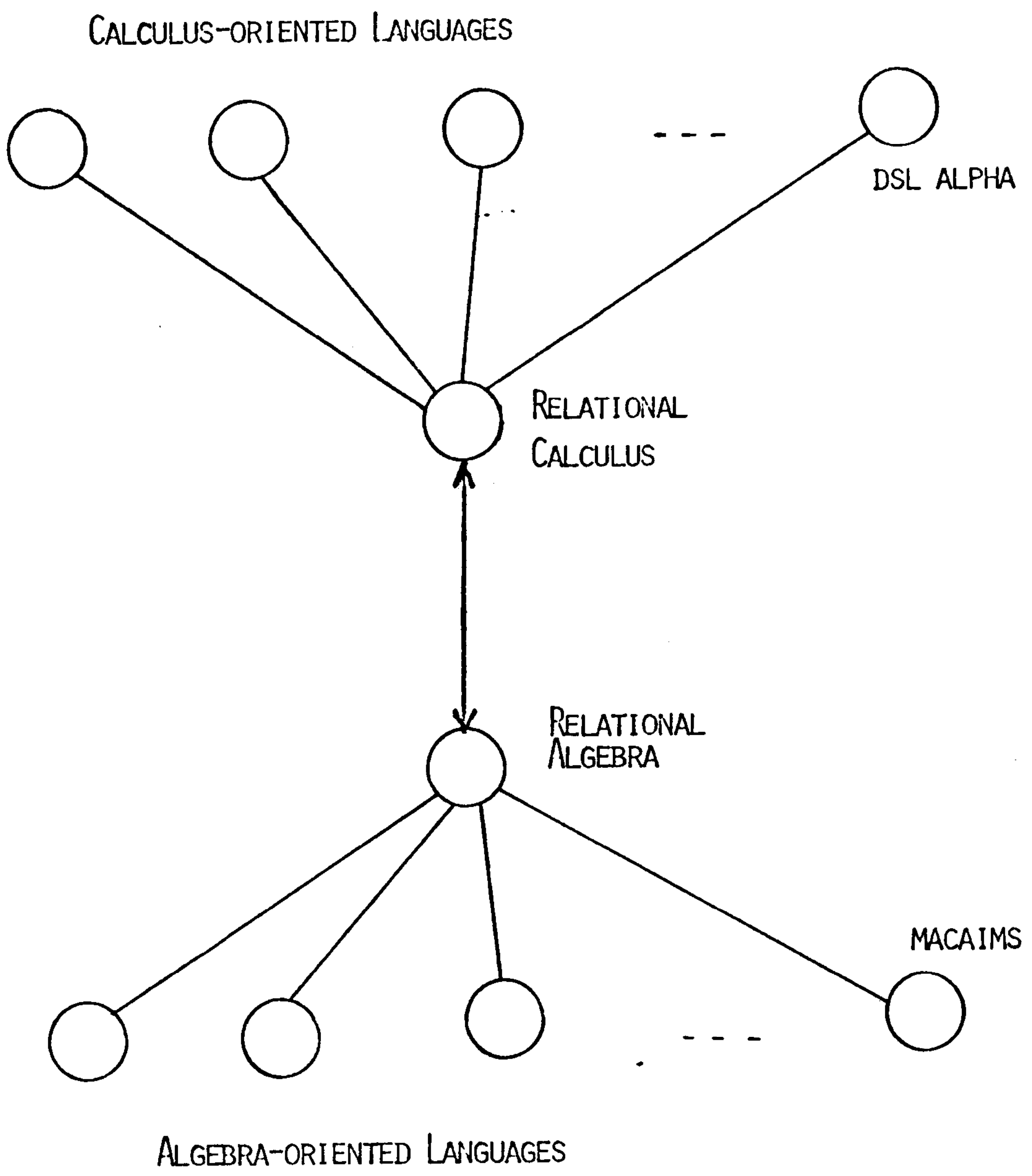

Fig. 1: COMPARISON SCHEME FOR DATA SUBLANGUAGES

2. A RELATIONAL ALGEBRA

2.1 Objective

The primary purpose of this algebra is to provide a collection of operations on relations of all degrees (not necessarily binary) suitable for selecting data from a relational data base. The relations to be operated upon are assumed to be normalized; that is, the domains on which they are defined are simple (see [2,3] and below). In Section 4, we shall discuss the selective power of this collection of operations.

Data selection is viewed as the formation (by some operation of the algebra) of a new normalized relation from the existing collection of relations. Presentation operations such as ordering a relation by values in one or more of its domains and factoring a normalized relation into unnormalized form are discussed elsewhere (see [4] and Appendix). These latter operations have no effect on the information content of retrieved data.

For notational and expository convenience, we deal with domain-ordered relations; that is, the individual domains of a given relation can be identified as the first, second, third, and so on. As pointed out in [1], however, in many practical data bases the relations are of such high degree that names rather than position numbers would be used to identify domains when actually storing or retrieving information.

2.2 Introductory Definitions

Our aim in this section is to define the kinds of domains (simple and compound) and the kinds of relations (normalized) which are operands for the operations to be subsequently defined.

The cartesian product of two sets C, D is denoted $C \times D$, and is defined by:

$$C \times D = \{(c,d): c \in C \wedge d \in D\}.$$

The expanded cartesian product χ of n sets $D_1, D_2, \ldots, D_n$ is defined by:

$$\chi(D_1,D_2,\ldots,D_n) = \{(d_1,d_2,\ldots,d_n): d_j \in D_j \text{ for } j = 1, 2, \ldots, n\}.$$

The elements of such a set are called n-tuples, or just tuples for short. When $n = 1$, $\chi(D_1) = D_1$ since no distinction is made between a 1-tuple and its only component.

Suppose $d = (d_1, d_2, \ldots, d_m)$ and $e = (e_1, e_2, \ldots, e_n)$. The concatenation of d with e is the $(m + n)$-tuple defined by

$$d^\frown e = (d_1,d_2,\ldots,d_m,e_1,e_2,\ldots,e_n).$$

R is a relation on the sets $D_1, D_2, \ldots, D_n$ if it is a subset of $\chi(D_1, D_2, \ldots, D_n)$. A relation is accordingly a special kind of set. Its members are all n-tuples where n is a constant called the degree of the relation. Relations of degree 1 are called unary, degree 2 binary, degree 3 ternary, degree n n-ary. The sets on which a relation is defined are called its underlying domains. For data base purposes, we are concerned with data consisting of integers and character strings (other types of primitive elements may be included in this definition if desired, with only minor changes in some of the definitions below).

A <u>simple domain</u> is a set all of whose elements are integers, or a set all of whose elements are character strings. A relation defined on simple domains alone is said to be <u>simple normal</u>. In the remainder of this paper, whenever the term relation is used without further qualification, it means simple normal relation.

A <u>compound domain</u> is the expanded cartesian product of a finite number (say k, $k \geq 1$) of simple domains; k is called the <u>degree</u> of the compound domain.

Two simple domains are <u>union-compatible</u> if both are domains of integers or both are domains of character strings. Two compound domains D, E are union-compatible if they are of the same degree (say m) and for every j ($j = 1, 2, \ldots, m$) the j^{th} simple domain of D is union-compatible with the j^{th} simple domain of E. Two relations R, S are union-compatible if the compound domains of which R and S are subsets are union-compatible.

2.3 Definitions of the Operations

The operations to be defined fall naturally into two classes: the traditional set operations (cartesian product, union, intersection, difference) and less traditional operations on relations (projection, join, division, restriction). We consider the traditional set operations first.

2.3.1 <u>Traditional Set Operations</u> - The usual cartesian product $R \times S$ of relation R (degree m) with relation S (degree n) is a relation of degree 2, using the definition given in Section 2.2 above. The cartesian product employed in the relational algebra, however, yields an expanded product, a relation $R \otimes S$, whose degree is $m + n$. This product is defined by

$$R \otimes S = \{(r^\frown s): r \in R \wedge s \in S\}.$$

Union $(\cup)$, intersection $(\cap)$, difference $(-)$ are defined in the usual way, except that they are applicable only to pairs of union-compatible normal relations.

2.3.2 Projection - Suppose r is a tuple of the m-ary relation R. For $j = 1, 2, \ldots, m$ the notation $r[j]$ designates the j^{th} component of r. For other values of j, $r[j]$ is undefined. We extend the notation to a list $A = (j_1, j_2, \ldots, j_k)$ of integers (not necessarily distinct) from the set $\{1, 2, \ldots, m\}$ as follows

$$r[A] = (r[j_1], r[j_2], \ldots , r[j_k]).$$

When the list A is empty, $r[A] = r$. Let R be a relation of degree m, and A a list of integers (not necessarily distinct) from the set $\{1, 2, \ldots, m\}$. Then, the projection of R on A is defined by

$$R[A] = \{r[A]: r \in R\}.$$

Note that when A is a permutation of the list $(1, 2, \ldots, m)$, $R[A]$ is a relation whose domains are the same as those of R except for a change in order of appearance.

In Fig. 2, we exhibit some examples of projection. Later we shall make use of the fact that projection provides an algebraic counterpart to the existential quantifier.

$R(D_1$	D_2	$D_3)$
a	2	f
b	1	g
c	3	f
d	3	g
e	2	f

$R[1](D_1)$
a
b
c
d
e

$R[2](D_1)$
2
1
3

$R[3](D_1)$
f
g

$R[3,2](D_1$	$D_2)$
f	2
g	1
f	3
g	3

$R[3,2,2](D_1$	D_2	$D_3)$
f	2	2
g	1	1
f	3	3
g	3	3

Figure 2

A ternary relation R and five of its many projections.

2.3.3 Join - Let θ denote any of the relations $=$, $\neq$, $<$, $\leq$, $>$, and $\geq$. The θ-join of relation R on domain A with relation S on domain B is defined by

$$R[A \ \theta \ B]S = \{(r^\frown s): r \in R \wedge s \in S \wedge (r[A] \ \theta \ s[b])\},$$

providing every element of R[A] is θ-comparable with every element of S[B]. Note that x is θ-comparable with y if $x\theta y$ is either true or false (not undefined).

In Fig. 3, we exhibit some examples of joins.

R(A	B	C)
a	1	1
a	2	1
b	1	2
c	2	5
c	3	3

S(D	E)
2	u
3	v
4	u

R[C = D]S(A	B	C	D	E)
b	1	2	2	u
c	3	3	3	v

R[B = D]S(A	B	C	D	E)
a	2	1	2	u
c	2	5	2	u
c	3	3	3	v

R[C > D]S(A	B	C	D	E)
c	3	3	2	u
c	2	5	2	u
c	2	5	3	v
c	2	5	4	u

Figure 3

Relations R,S and three joins

Note that

$$R[C < D]S \cup R[C = D]S \cup R[C > D]S = R \otimes S.$$

The most commonly needed join is the join on =, which we call the equi-join. In the case of the equi-join, two of the domains of the resulting relation are identical in content. If one of the redundant domains is removed by projection, the result is the natural join of the given relations as defined in [2].

2.3.4 Division - Suppose T is a binary relation. The image set of x under T is defined by

$$g_T(x) = \{y: (x,y) \in T\}.$$

Consider the question of dividing a relation R of degree m by a relation S of degree n. Let A be a domain-identifying list (without repetitions) for R, and let $\bar{A}$ denote the domain-identifying list that is complementary to A and in ascending order. For example, if the degree m of R were 5 and $A = (2,5)$, then $\bar{A} = (1,3,4)$. We treat the dividend R as if it were a binary relation with the (possibly compound) domains $\bar{A}$, A in that order. Accordingly, given any tuple $r \in R$, we can speak of the image set $g_R(r[\bar{A}])$, and we note that this is a subset of $R[A]$.

Providing $R[A]$ and $S[B]$ are union-compatible, the division of R on A by S on B is defined by

$$R[A \div B]S = \{r[\bar{A}]: r \in R \wedge S[B] \subseteq g_R(r[\bar{A}])\}.$$

Note that, when R is empty, R divided by S is empty, even if S is also empty.

In Fig. 4, we exhibit two examples of division. These examples show that projection on the dividend preceding division can have a different effect from division followed by projection on the quotient.

R(A	B	C)
1	11	x
2	11	y
3	11	z
4	12	x

S(D	F)
x	1
x	2
y	1

$$R[C \div D]S = \phi$$

$$R[B,C][C \div D]S = \{11\}.$$

Figure 4

Division of R by S

Later we shall see that division provides an algebraic counterpart to the universal quantifier. Actually, division is definable in terms of the operations already introduced:[†]

$$R[C \div D]S = R[\overline{C}] - ((R[\overline{C}] \otimes S[D]) - R)[\overline{C}].$$

2.3.5 <u>Restriction</u> - Suppose R is a relation and A, B are domain-identifying lists for R. Let θ denote any of the relations $=, \neq, <, \leq, >,$ and $\geq$. The θ-restriction of R on domains A, B is defined by

$$R[A\ \theta\ B] = \{r: r \in R \wedge (r[A]\ \theta\ r[B])\},$$

† This observation was made by Paul Healey of IBM Research, San Jose.

providing every element of R[A] is θ-comparable with every element of R[B]. In Fig. 5, we exhibit two examples of restriction.

R	A	B	C
	p	2	1
	q	2	3
	q	5	4
	r	3	3

R[B = C]	A	B	C
	r	3	3

R[B > C]	A	B	C
	p	2	1
	q	5	4

Figure 5

A relation R and two of its restrictions.

This operation is introduced because of its direct use in Section 4. It is definable in terms of the θ-join already introduced. Thus,

$$R[A\ \theta\ B] = (R[A^\frown B = A^\frown B](R[A][A\ \theta\ B]R[B]))[L],$$

where L is a list identifying all the domains of R in ascending order, and $A^\frown B$ denotes the concatenation of list A with list B. The θ-join of R with S is likewise definable in terms of cartesian product and θ-restriction:

$$R[A\ \theta\ B]S = (R \otimes S)[A\ \theta\ B].$$

2.4 Sample Queries

Suppose a data base includes the following two relations:

Symbol	Relation Name	Domain 1	Domain 2	Domain 3
S	suppliers	supplier #	supplier name	location
T	supply	supplier #	part #	

Table 1 below lists nine queries, along with appropriate algebraic expressions E for them.

■ Find the supplier numbers of the suppliers each of whom supplies:

j	Item j	E_j
1.	Something	T[1]
2.	Nothing	S[1] - T[1]
3.	Part 15	(T[2=1]{15})[1]
4.	Something, but not part 15	T[1] - ③
5.	Not part 15	S[1] - ③
6.	A part other than 15	(T[2=1](T[2] - {15}))[1]
7.	Part 15 only	③ - ⑥
8.	At least parts 12, 13, 15	T[2÷1]{12, 13, 15}
9.	All parts supplied	T[2÷2]T

Table 1

Examples of Algebraic Expressions

- Find the locations of those suppliers each of whom supplies item j in the table above

$$(S[1 = 1]E_j)[3].$$

These examples demonstrate that reasonably complicated queries can be concisely expressed in terms of the relational algebra.

3. RELATIONAL CALCULUS

Having defined a relational algebra, we now consider an applied predicate calculus which may also be used in the formulation of queries on any data base consisting of a finite collection of relations in simple normal form.

3.1 Alphabets, Terms, and Formulae

The alphabets for this calculus are listed in Table 2 below:

Individual Constants	$a_1, a_2, a_3, \ldots$
Index Constants	1, 2, 3, 4, ...
Tuple Variables	$r_1, r_2, r_3, \ldots$
Predicate Constants	
monadic	$P_1, P_2, P_3, \ldots$
dyadic	$=, <, >, \leq, \geq, \neq$
Logical Symbols	$\exists, \forall, \wedge, \vee, \neg$
Delimiters	[ ] ( ) ,

Table 2

The Alphabets of the Relational Calculus

Under the intended interpretation, a one-to-one correspondence is established between the monadic predicates (as many as are needed) and the relations in the given data base. Suppose the relations are $R_1, R_2, \ldots, R_N$. Then, P_j indicates membership of tuples in relation R_j $(j = 1, 2, \ldots, N)$. A monadic predicate followed by a tuple variable is called a range term. The range term $P_j r$ is interpreted as stating that tuple variable r has relation R_j as its range.

An indexed tuple has the form $r[N]$ where r is a tuple variable and N is an index constant. Its purpose is to identify the N^{th} component of a tuple r.

Let θ be one of the predicate symbols $=$, $\neq$, $<$, $\leq$, $>$, and $\geq$. Let λ, μ be indexed tuples and α an individual constant. Then, $\lambda\theta\mu$ and $\lambda\theta\alpha$ are called join terms. The terms of the relational calculus are of only two types: range terms and join terms.

The well-formed formulae (abbreviated WFF) of the relational calculus are defined recursively as follows:

1. Any term is a WFF;
2. If Γ is a WFF, so is $\neg\Gamma$;
3. If Γ_1, Γ_2 are WFFs, so are $(\Gamma_1 \vee \Gamma_2)$ and $(\Gamma_1 \wedge \Gamma_2)$;
4. If Γ is a WFF in which r occurs as a free variable, then $\exists r(\Gamma)$ and $\forall r(\Gamma)$ are WFFs;
5. No other formulae are WFFs.

The usual conventions are adopted for saving parentheses and avoiding duplicate use of bound variables. Table 3 gives examples of WFFs of the relational calculus.

WFFs with no range terms	Free Variables
$r_1[3] > a_1$	r_1
$\exists r_1(r_1[3] > a_1)$	none
$\exists r_1(r_1[3] = r_2[5])$	r_2
$\forall r_1 \exists r_2((r_1[3] = r_2[5]) \wedge (r_2[1] = a_1))$	none
$(r_1[3] = r_2[5]) \wedge \exists r_3(r_3[1] = a_1)$	r_1, r_2
WFFs with range terms only	
$P_6 r_3$	r_3
$P_6 r_3 \wedge P_7 r_2$	r_2, r_3
$P_5 r_3 \vee P_6 r_3$	r_3
$P_5 r_3 \wedge \neg P_6 r_3$	r_3
Range-Separable WFFs	
$P_8 r_1 \wedge (r_1[3] = a_1)$	r_1
$P_7 r_2 \wedge \exists P_8 r_1(r_1[3] = r_2[5])$	r_2
WFFs not Range-Separable	
$P_8 r_1 \vee (r_1[3] = a_1)$	r_1
$P_7 r_2 \wedge \exists r_1((r_1[3] = r_2[5]) \vee P_8 r_1)$	r_2
$\neg P_8 r_1 \wedge (r_1[3] = a_1)$	r_1

Table 3

Examples of WFFs of the Relational Calculus

3.2 Range Separability

With each tuple variable in a WFF, we need to associate a clearly defined range. The following definitions are aimed at this goal.

A range WFF is a quantifier-free WFF, all of whose terms are range terms. A range WFF over r is a range WFF whose only free variable is r. A proper range WFF over r is a range WFF over r satisfying two constraints: the syntactic constraint that either $\neg$ does not occur at all or that it immediately follows $\wedge$; and the semantic constraint that, whenever r occurs in two or more range terms, the range predicates in those terms must be associated with relations which are union-compatible. The syntactic constraint prohibits specifying the range of a tuple variable r by merely stating that relation R (say) is not the range of r. The two constraints together prohibit tuple variables from having ranges which are other than the given relations or relations which can be generated from them by applying union, intersection, and difference to union-compatible pairs of relations.

Both bound and free variables must have clearly defined ranges. Suppose Δ is a WFF having r as a free variable, but containing no range term in r. Let Γ be a proper range WFF over r. To introduce Γ into $\exists r(\Delta)$ or $\forall r(\Delta)$, we replace $\exists r$ by $\exists\Gamma r$, and $\forall r$ by $\forall\Gamma r$. These are called range-coupled quantifiers and are defined by the equations:

$$\exists\Gamma(\Delta) = \exists r(\Gamma \wedge \Delta)$$
$$\forall\Gamma(\Delta) = \forall r(\neg\Gamma \vee \Delta).$$

Now, we can define a class of WFFs having clearly defined ranges for all its variables. A WFF is range-separable if it is a conjunction of the form

$$U_1 \wedge U_2 \wedge \ldots \wedge U_n \wedge V,$$

where

1) $n \geq 1$;
2) U_1 through U_n are proper range WFFs over n distinct tuple variables;
3) V is either null (in which case the formula is simply $U_1 \wedge U_2 \wedge \ldots \wedge U_n$), or it is a WFF with the three properties:
    a) every quantifier in V is range-coupled;
    b) every free variable in V belongs to the set whose ranges are specified by $U_1, U_2, \ldots, U_n$;
    c) V is devoid of range terms.

One consequence of these requirements if that a range-separable WFF has at least one free variable.

3.3 Alpha Expressions

If the range-separable WFFs of the relational calculus were used as relation-defining expressions without further augmentation, they would lack the much-needed capability of defining projections of relations. Accordingly, we consider simple alpha expressions of the form

$$(t_1, t_2, \ldots, t_k) : w$$

where

1) w is a range-separable WFF of the relational calculus;
2) $t_1, t_2, \ldots, t_k$ are distinct terms, each consisting of a tuple variable or an indexed tuple variable;

3) the set of tuple variables occurring in $t_1, t_2, \ldots, t_k$ is precisely the set of free variables in w.

Set brackets { } are omitted because they are syntactically superfluous. The list $(t_1, t_2, \ldots, t_k)$ is called the <u>target list</u> and w the <u>qualification expression</u>.

Suppose $\rho_1, \rho_2, \ldots, \rho_n$ are the distinct tuple variables in order of their first occurrence in the target list of a simple alpha expression z:

$$z = (t_1, t_2, \ldots, t_k) : w.$$

Suppose that relations $S_1, S_2, \ldots, S_n$ (not necessarily distinct) are the ranges of $\rho_1, \rho_2, \ldots, \rho_n$, respectively. Then, z denotes a certain projection of that subset of $S_1 \otimes S_2 \otimes \ldots \otimes S_n$ whose elements satisfy the qualification expression w. The projection in question is indicated in an obvious way by the indices associated with the tuple variables.

Now follow examples of queries in the form of simple alpha expressions. The data base of Section 2.4 is assumed. Predicates P_1, P_2 are the range predicates for relations S (suppliers) and T (supply), respectively.

■ Find the supplier number of those suppliers who supply part 15.

$$r_2[1] : P_2 r_2 \wedge (r_2[2] = 15).$$

■ Find the supplier numbers of those suppliers who supply something other than part 15.

$$r_2[1] : P_2 r_2 \wedge (r_2[2] \neq 15).$$

- Find the supplier names and locations of those suppliers who supply part 15.

$$(r_1[2], r_1[3]): P_1 r_1 \wedge \exists P_2 r_2 (r_2[2] = 15 \wedge r_2[1] = r_1[1]).$$

- Find the locations of suppliers and the parts being supplied by them (omitting those suppliers who are supplying no parts at this time).

$$(r_1[3], r_2[2]): P_1 r_1 \wedge P_2 r_2 \wedge (r_1[1] = r_2[1]).$$

The concept of simple alpha expression can be generalized without losing its desirable range properties. An <u>alpha expression</u> is recursively defined as follows:

1) Every simple alpha expression is an alpha expression;
2) If $t: w_1$ and $t: w_2$ are alpha expressions, so are
$$t: (w_1 \vee w_2)$$
$$t: (w_1 \wedge \neg w_2)$$
$$t: (w_1 \wedge w_2);$$
3) No other expressions are alpha expressions.

While it is doubtful that many queries will attain the complexity of alpha expressions of the non-simple kind, it would be artificial to exclude them from the theory.

3.4 <u>Relational Completeness</u>

Now we can introduce a basic notion of selective power. An algebra or calculus is <u>relationally complete</u> if, given any finite collection of relations

R_1, R_2, ..., R_N in simple normal form, the expressions of the algebra or calculus permit definition of any relation definable from R_1, R_2, ..., R_N by alpha expressions (using a set of N range predicates in one-to-one correspondence with R_1, R_2, ..., R_N). We shall apply this notion in Section 4 to the algebra of Section 2.

4. REDUCTION

The objective of this section is to show that the relational algebra defined in Section 2 is relationally complete. We proceed by exhibiting an algorithm for translating any simple alpha expression z into a semantically equivalent algebraic expression T. We then extend this algorithm to deal with any alpha expression whatsoever.

The semantic equivalence is based partly on the intended interpretation (described in Section 3) of the alpha expression, and partly on an arbitrarily specified one-to-one correspondence $P_j \leftrightarrow R_j$ $(j=1,2,\ldots,N)$ between range predicates and the N simple normal relations of whatever relational data base is given.

The reduction algorithm for simple alpha expressions may be best understood by supposing for the moment that, instead of merely producing several algebraic expressions (which are finally combined into one composite expression T), these expressions are evaluated as they are produced. The effect of this evaluation would be roughly as follows. First, the ranges of the cited tuple variables are generated by retrieving certain data base relations and by taking unions, intersections, and set differences as necessary. Second, a cartesian product of these ranges is formed. From this product the final relation is eventually extracted. Third, tuples that do not satisfy the combination of join terms are removed from the product. Fourth, the remaining product is whittled down by projection and division to satisfy the quantification in the alpha expression. Finally, a projection as specified by the target list is performed, and we have the required relation. A more formal account now follows.

4.1 Reduction of Simple Alpha Expressions

Let the given alpha expression be

$$z = t : w,$$

where 1) $t = (t_1, t_2, \ldots, t_k)$ is the target list;

2) $w = U_1 \wedge U_2 \wedge \ldots \wedge U_p \wedge V$ is the qualification (a range-separable WFF);

3) there are $q \geq 0$ bound variables in V;

4) all p of the variables free in w occur in t (thus, $k \geq p$).

Step 1

For convenience, we apply four transformations to z yielding a new alpha expression with the same denotation:

1) convert V to prenex normal form (if it is not already in this form) without expanding the range-coupled quantifiers (in the normalization the range-coupled quantifiers behave just like ordinary quantifiers);

2) keeping the leading quantifiers unchanged, convert the remaining subformula (the so-called matrix) to disjunctive normal form;

3) wherever a join term using relation θ is immediately preceded by $\neg$, eliminate the symbol $\neg$ and replace θ by its complement (the complements of $=, \neq, <, \leq, >,$ and $\geq$ are $\neq, =, \geq, >, \leq,$ and $<$, respectively);

4) systematically apply an alphabetic change to the variables in the alpha expression resulting from 3) so that the variables become

$r_1, r_2, \ldots, r_{p+q}$ in the order of their first occurrence in the qualification (note that the first occurrence of a bound variable is with its quantifier).

Let the alpha expression resulting from these four transformations be

$$z' = t : U'_1 \wedge U'_2 \wedge \ldots \wedge U'_p \wedge V'.$$

The bound variables in V' are r_j, where $j=p+1, p+2, \ldots, p+q$. Let the quantifier ($\exists$ or $\forall$) associated with r_j in V' be Q_j, and let the proper range WFF over r_j in V' be U'_j.

Step 2

For $j=1, 2, \ldots, p+q$ form a defining equation for the range S_j (say) of tuple variable r_j. The algebraic expression on the right-hand side of this equation is obtained from U'_j (the proper range WFF over r_j) by applying the following rewriting rules to U'_j:

1) $P_i r_j \rightarrow R_i$
2) $\vee \rightarrow \cup$
3) $\wedge\neg \rightarrow -$
4) $\wedge \rightarrow \cap$, providing 3) is inapplicable.

For example, if $U'_j = P_3 r_j \wedge \neg P_2 r_j$, then the defining equation for S_j is

$$S_j = R_3 - R_2.$$

In this case, R_3 and R_2 must be union-compatible (by definition of proper range WFF).

Step 3

Associated with each relation R_i $(i=1,2,\ldots,N)$ in the given data base is the union of R_i with all the data base relations that are union-compatible with R_i. Call this union $\mathscr{U}(R_i)$. For any subset S_j of $\mathscr{U}(R_i)$, define $\mathscr{U}(S_j) = \mathscr{U}(R_i)$. Form the defining equation

$$S = X_1 \otimes X_2 \otimes \ldots \otimes X_{p+q},$$

where $X_j = S_j$ if r_j is free or existentially bound,
$\mathscr{U}(S_j)$ otherwise.

The special treatment accorded the universal quantifier permits correct handling of the case in which $S_j = \phi$ and r_j is universally quantified.* The alternative of replacing the range-coupled universal quantifier in the definition of alpha expressions by an extensional counterpart of the type proposed by Kuhns in [8] was rejected because the extensional quantifiers do not possess the complementary property analogous to:

$$\forall r \Delta \equiv \neg \exists r \neg \Delta$$

for any WFF Δ. This condition is needed for the first transformation in Step 1.

* Incidentally, a data base system should warn a user whenever it encounters an empty range S_j or an empty universe $\mathscr{U}(S_j)$ in the interpretation of a query, and inform him of the pertinent condition.

Let the degree of relation X_j be n_j $(j=1,2,\ldots,p+q)$. Let

$$\mu_j = \sum_{i=1}^{j-1} n_i .$$

Then, the domain with position J in relation X_j has position $J + \mu_j$ in the cartesian product S.

Step 4

Remove the range-coupled quantifiers (if any) from V' to yield the quantifier-free WFF V''. If V'' is null, form the defining equation

$$T_{p+q} = S.$$

Otherwise, form a defining equation for T_{p+q}, the right-hand side of which is an algebraic expression obtained from V'' by applying the following rewriting rules:

1) $\vee \rightarrow \cup$
2) $\wedge \rightarrow \cap$
3) $(r_j[J]\theta r_k[K]) \rightarrow S[(J+\mu_j)\theta(K+\mu_k)]$
4) $(r_j[J]\theta a) \rightarrow S[(J+\mu_j)\theta 1]\{a\}$

where a is an individual constant

θ is one of $=, \neq, <, \leq, \geq, >$.

Note that the symbol $\neg$ does not occur at all in V'' due to the third transformation in Step 1.

Step 5

Form q defining equations for T_{j-1} $(j=p+q, p+q-1, \ldots, p+1)$ which reflect the effect of the quantifiers of V' starting with the innermost Q_{p+q} and proceeding to the outermost Q_{p+1}. The equation for T_{j-1} is:

$$\begin{aligned} T_{j-1} = \; & T_j[C_j] && \text{if} \quad Q_j = \exists \\ & T_j[C_{j+1} \div D_j]S_j && \text{if} \quad Q_j = \forall, \end{aligned}$$

where $C_j = (1,2,\ldots, \mu_j)$
$D_j = (1,2,\ldots, n_j)$.

Step 6

Form a defining equation for T which takes into account the projection specified in the original target list t:

$$T = T_p[C_1 \frown C_2 \frown \ldots \frown C_k],$$

where, for $h=1,2,\ldots,k$

$$\begin{aligned} C_h = \; & (1+\mu_j, 2+\mu_j, \ldots, \mu_{j+1}) && \text{if} \quad t_h = r_j \\ & (J+\mu_j) && \text{if} \quad t_h = r_j[J] \end{aligned}$$

and μ_j is defined as in Step 3.

Step 7

By means of simple substitution, form an equation which defines T directly in terms of $R_1, R_2, \ldots, R_N$ (and their respective degrees $n_1, n_2, \ldots, n_N$) by eliminating $X_1, X_2, \ldots, X_{p+q}, S_1, S_2, \ldots, S_{p+q}, S, T_{p+q}, \ldots, T_p$ from the equations generated above. We can assume that equations for $\mathcal{W}(R_i)$ $(i=1,2,\ldots,N)$ are given, since they represent a property of the given data base rather than of the queries.

4.2 Reduction of an Arbitrary Alpha Expression

The qualification w of an alpha expression z is either a range-separable WFF (this is the case treated in Section 4.1) or a logical combination of $m > 1$ such formulae (say $w_1, w_2, \ldots, w_m$) over a common set of free variables using the connectives $\vee$, $\wedge$, and $\wedge\neg$. Let t be the target list. To convert z to algebraic form, we use the reduction of Section 4.1 to generate algebraic defining equations Z_i for each z_i where

$$z_i = t : w_i \qquad (i=1,2,\ldots,m).$$

We then form a defining equation for z, the right-hand side of which is obtained from the given logical expression for w in terms of w_i by applying the rewriting rules:

1) $\vee \rightarrow \cup$
2) $\wedge\neg \rightarrow -$
3) $\wedge \rightarrow \cap$ if 2) is inapplicable
4) $w_i \rightarrow Z_i$ $(i=1,2,\ldots,m)$.

Simple substitution of the expressions for Z_i in this equation yields a defining equation for z.

4.3 Example of a Reduction

Suppose the data base relations include the following:

Symbol	Relation Name	Degree n	Range Predicate	Domain 1	Domain 2	Domain 3
R_1	suppliers	3	P_1	supplier #	name	location
R_2	projects	2	P_2	project #	name	
R_3	supply	3	P_3	supplier #	part #	project #

Further suppose $\mathcal{U}(R_i) = R_i$ (i=1,2,3). *Consider the query:*

- Find the name and location of all suppliers, each of whom supplies all projects.

An alpha expression which represents this query is:

$$(r_1[2],\ r_1[3]):\ P_1 r_1 \wedge \forall P_2 r_2 \exists P_3 r_3 \Big((r_1[1] = r_3[1]) \wedge (r_3[3] = r_2[1])\Big).$$

Applying the reduction procedure of Section 4.1, we obtain the following defining equations:

$$S_i = R_i \qquad (i=1,2,3)$$

$$S = S_1 \otimes S_2 \otimes S_3$$

$$T_3 = S[1=6] \cap S[8=4]$$

$T_2 = T_3[1,2,3,4,5]$

$T_1 = T_2[(1,2,3,4,5) \div (1,2)]S_2$

$T = T_1[2,3].$

4.4 Validity of the Reduction Algorithm

The reduction algorithm is based on two simple lemmas. Proof of these is left to the reader.

Lemma 1

Let α be one of the three logical connectives $\vee$, $\wedge$, $\wedge\neg$ and σ_α the corresponding set operator $\cup$, $\cap$, $-$. Let Γ,Δ be WFFs in the relational calculus, each having r as the only free variable. Then,

$$\{r : \Gamma\alpha\Delta\} = \{r : \Gamma\} \ \sigma_\alpha \ \{r : \Delta\}.$$

Lemma 2

Let R,S be relations of degree m,n, respectively. Let Γ be a WFF of the relational calculus with free variables r, s. Let T be defined by

$$T = \{(r^\frown s) : r \in R \wedge s \in S \wedge \Gamma\}.$$

Then

a) $T[\omega^{0}_{m}] = \{r : r \in R \wedge \exists s(s \in S \wedge \Gamma)\}.$

b) $T[\omega^{m}_{m+n} \div \omega^{0}_{n}]S = \{r : r \in R \wedge \forall s(s \notin S \vee \Gamma)\},$

where ω^{p}_{q} = ~~1,2,...,p~~ $p+1, p+2, \ldots, q$ for any ~~positive~~ integers ~~p~~ $q > p \geqq 0$.

5. CALCULUS VERSUS ALGEBRA

A query language (or other data sublanguage) which is claimed to be general purpose should be at least relationally complete in the sense defined in this paper. Both the algebra and calculus described herein provide a foundation for designing relationally complete query languages without resorting to programming loops or any other form of branched execution -- an important consideration when interrogating a data base from a terminal.

One advantage that might be claimed for the algebraic approach is its freedom from quantifiers. However, the calculus appears to be superior to the algebra in four respects.

1) Ease of Augmentation - As pointed out earlier, relational completeness represents a very basic selective power, which in most practical environments would need to be enhanced. The most natural type of enhancement is the introduction of a capability of invoking any of a finite set of library functions while staying within the algebraic or calculus framework (whichever is selected). Inspection of alpha expressions reveals three distinct locations within such expressions for potential invocation of library functions. The first of these locations is within the target list to provide some transformation of the retrieved relation. The second is the replacement of a join term by a truth-valued function of one or more tuple variables (possibly indexed). The third is the replacement of an indexed tuple variable within a join term by a function (yielding an integer or character string, for example). Such enhancements readily fit into the calculus framework. In the algebraic

framework, however, all such functions have to be re-cast in the form of mappings from relations to relations. This gives rise to circumlocutions.

2) Scope for Search Optimization - The relational calculus permits a user to request the data he desires by its properties. This is an ideal starting point for search optimization. The algebra, on the other hand, requires the user to formulate a sequence of algebraic operations that will generate the desired data from the data base relations. For queries other than very simple ones, the properties of the desired data tend to get hidden in the particular operation sequence (one of many possible ones) which the user selects.

Therefore, starting from an algebraic source language, one has the choice of locally optimizing the execution of each operation (a very limited form of optimization) or tackling the difficult problem of analyzing sequences of such operations to discover the intended defining properties of the desired data.

3) Authorization Capability - Highly discriminating authorization must be based on the defining properties of the data requested by the user rather than on a particular algorithm specified by the user for retrieving that data. The arguments in 2) thus apply with equal force to the system's authorization capability.

4) <u>Closeness to Natural Language</u> - Clearly, the majority of users should not have to learn either the relational calculus or algebra in order to interact with data bases. However, requesting data by its properties is far more natural than devising a particular algorithm or sequence of operations for its retrieval. Thus, a calculus-oriented language provides a good target language for a more user-oriented source language.

6.0 ACKNOWLEDGMENT

The author is indebted to Frank Palermo of IBM Research, San Jose, for many valuable comments and criticisms. In particular, he discovered an error in the treatment of empty ranges in Section 4.1, and suggested the introduction of the universes $\mathscr{U}(S_j)$. In addition, Dr. Palermo has programmed the relational algebra described in this paper, and subjected the reduction algorithm to a number of tests on a computer.

REFERENCES

1. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks", Comm. ACM 13, June 1970, 377-387.

2. Codd, E. F., "Further Normalization of the Data Base Relational Model", Courant Computer Science Symposia 6, "Data Base Systems", New York City, May 24-25, 1971, Prentice-Hall.

3. Codd, E. F., "Normalized Data Base Structure: A Brief Tutorial", Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego, available from ACM, New York.

4. Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus", Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access, and Control, San Diego, available from ACM, New York.

5. CODASYL Development Committee, "An Information Algebra", Comm. ACM 5, April 1962, 190-204.

6. Kuhns, J. L., "Logical Aspects of Question Answering by Computer", Proc. Third International Symposium on Computer and Information Sciences, Miami Beach, Florida, Dec. 1969, Academic Press.

7. Kuhns, J.L., "Interrogating a Relational Data File: Remarks on the Admissibility of Input Queries", Rand Corporation R-511-PR, November 1970.

8. Kuhns, J.L., "Quantification in Query Systems", Proc. Symposium on Information Storage and Retrieval, April 1-2, 1971, available from ACM, New York.

9. Strnad, A. L., "The Relational Approach to the Management of Data Bases", Proc. IFIP Congress, Ljubljana 1971, North-Holland.

APPENDIX

For presentation purposes, it may be desirable to convert a normalized relation to unnormalized form. The operation of factoring accomplishes this. Let R be a relation, and A a domain-identifying list for R. The A-factoring of R is defined by

$$\lambda(R,A) = \{(x, g_R(x)) : x \in R[A]\}$$

where g_R is as defined in Section 2.3.4. Repeated application of this operation with successively smaller lists A can convert R to a multi-level hierarchical relation.