データベース管理システム 23－2
（1980. 1. 16）

A DATABASE MANIPULATION MODEL

Isamu KOBAYASHI
(SANNO College of Management and Informatics)

1. INTRODUCTION

A database is a collection of database relations each representing facts of a specific type in the real world. For retrieving and updating database relations, we have to provide various database manipulating operations. Some of them are relatively simple but some others are very complicated. It is desirable to have a finite set of database operations by whose combinations we can achieve any desired database manipulation. Hence we would like to clarify what the essential operations are. This may leads us to abstract the database operations. Several different levels of abstraction can be established.

In this paper three levels of abstract database operations are introduced. One is composed of only one operation called the alpha operation. It is specified by a pair of a calculus and a tuple generating function. The relational calculus is somewhat extended to enable definition of any logical function defined on a Cartesian product of databse relations.

The second level is composed of various algebraic operations, including relational algebra, information algebra, and navigations. The alpha operation can be decomposed into a procedural combination of these algebraic operations. However, such a decomposition is not always brings the best performance into executing an alpha operation.

The third level is composed of tuplewise operations that are obtained by a direct decomposition of alpha operations. They are necessary for achieving optimal implementation of certain database manipulations. Data manipulation languages can be designed with these three levels of abstract database operations taken into consideration.

2. FUNCTION OF TUPLES

A function whose domain is a Cartesian product of m database relations

$$\lambda: R_1 \times R_2 \times \ldots \times R_m \to V$$

is called a *function of lines* of *span* m regardless of what its range is. In particular, λ is called a *function of points* if m=1.

A function whose domain is a power set of database relation

$$\mu: P(R) \to V$$

is called a *function of areas* regardless of what its range is. Functions of lines and functions of areas are collectively called *functions of tuples*.

For example, 'overtime-charge' is a function of points defined on a relation EMP if both the 'overtime' and 'overtime-rate' are its attributes. In this case.

$$\text{overtime-charge}(t) \equiv \text{overtime}(t) \times \text{overtime-rate}(t)$$

for any tuple $t \in$EMP. We may write the right hand side simply as (overtime×overtime-rate)(t). If the 'overtime-rate' is an attribute of EMP but the 'overtime' is an attribute of another relation TRANS, then the 'overtime-charge' becomes a function of lines of span 2 defined on EMP TRANS, that is

$$\text{overtime-charge}(t_1, t_2) \equiv \text{overtime-rate}(t_1) \times \text{overtime}(t_2)$$

for $t_1 \in$EMP and $t_2 \in$TRANS.

The 'weekly-salary-total' of a department is a function of areas defined on the power set of EMP if the 'salary' is one of its attributes. For a subset, say 'MARKETING', of EMP, this function can be defined by

$$\text{weekly-salary-total(MARKETING)} \equiv \Sigma_{t \in \text{MARKETING}} \text{salary}(t)$$

using an *aggregate function* Σ.

Among the function of points, the following three types are called the *basic functions of points*, since they are the only data that can be directly obtained from the database relations:

(1) $\lambda(t) \equiv$ constant
(2) $\lambda(t) \equiv A_k(t)$: the k-th attribute value of tuple t.
(3) $\lambda(t) \equiv t \in R$: a monadic logical function that takes 'true' value if and only if t belongs to relation R.

From a set B of the basic functions of points, we can generate various functions of tuples. Operators defined in various value sets are extended to the operators combining functions of tuples. If an operator

$$f: V_1 \times V_2 \times \ldots \times V_m \to V_f$$

is defined in a Cartesian product of m value sets, then f can be extended to an operator

$$f': F_1 \times F_2 \times \ldots \times F_m \to F_f$$

with each F_k being a set of functions of tuples whose range is V_k and F_f being a set of functions of tuples whose range is V_f. The extention is achieved homomorphically by defining

$$f'(\zeta_1, \zeta_2, \ldots, \zeta_m)(z) \equiv f(\zeta_1(z_1), \zeta_2(z_2), \ldots, \zeta_m(z_m)),$$

where z is a tuple (point), an ordered set of tuples (line) or a set of tuples (area), and each z_k

is a point, a line or an area, which, as a set of point, is a subset of z.

An aggregate operator

$$\alpha: P(V) \to V$$

defined in a value set V can be extended to an aggregate operator

$$\alpha': \Pi_v \to M_v$$

where Π_v is a set of functions of points whose range is V, and M_v is a set of functions of areas whose range is V. The extention is achieved by defining

$$\alpha'(\lambda)(S) \equiv \dot{\alpha}_{t\varepsilon S}\lambda(t)$$

for $\alpha\varepsilon\Pi_v$ and $S\varepsilon P(R)$.

Among various functions of tuples generated from the set of B of basic functions of points, the following two play very important roles in database manipulation. One is the logical function of lines

$$\lambda: R_1\times R_2\times\ldots\times R_m \to \{\text{'true','false'}\},$$

which can be regarded as a predicate with m free variables.

A predicate can include bound variables in addition to free variables. They are bounded by universal or existential quantifications or by some other means. In its prenex normal form, λ can be written as

$$\lambda(t_1,t_2,\ldots,t_m) \equiv Q_1Q_2\ldots Q_p(\lambda'(t_1,t_2,\ldots,t_m,t_{m+1},t_{m+2},\ldots,t_{m+q}))$$

with $p\leq q$. Here t_k is a free variable for $0\leq k\leq m$, a quantified variable for $m+1\leq k\leq m+p$, and another bound variable for $m+p+1\leq k\leq m+p+q$. A Q_k is of the form t_k/R_{m+k} or $\exists t_k/R_{m+k}$, which are respectively a shorthand notation of

$$\forall t_k(\vee t_k\varepsilon R_{m+k}{}^{\vee}(\ldots)) \text{ and } \exists t_k(t_k\varepsilon R_{m+k}\wedge(\ldots)).$$

For example, in

$$\lambda(t_1) \equiv \exists t_2/R_2(A_1(t_1)=A_1(t_2)\wedge A_2(t_1)=\Sigma_{t_3\varepsilon R_3\wedge A_3(t_3)=A_3(t_1)}A_2(t_3))$$

defined for $t_1\varepsilon R_1$, t_1 is a free variable, t_2 is an existentially quantified variable, and t_3 is a variable bounded in the scope of aggregate function Σ.

As a special case, we may have predicates without free variables, that is, m=0. They cannot be regarded as a logical functions of tuples, instead they can be considered to be statements regarding the whole database.

The other is the function of tuples defined as an ordered set of n functions of tuples, that is,

$$(\zeta_1,\zeta_2,\ldots,\zeta_n)(z) \equiv (\zeta_1(z),\zeta_2(z),\ldots,\zeta_n(z))$$

with z being a point, a line or an area. For a given z, such a function generates a point (tuple). Hence we will call such functions *tuple generating functions*.

Generation of functions of tuples from the basic functions of points accords with a certain generative grammar. Let G be the set of productions of the grammar and Φ be the set of operators combining functions of tuples, quantifiers and other symbols such as those determining the operator precedence. We have a set $F(B,\Phi,G)$ of all the functions of tuples that can be generated from B according to G using Φ. This set represents the computational capability possessed by the given system.

3. ALPHA OPERATION

Database manipulation, in general, can be seen as a transformation from m input relations into m' output relations as shown in Figure 1. Obviously, this transformation can be regarded as a combination of m' transformations each transforming m input relations into one output relation. Each transformation can be considered as an m-ary set operation

$$\tau(R_1,R_2,\ldots,R_m)=R.$$

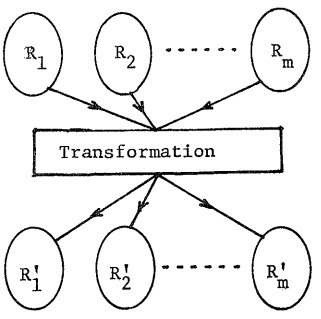

Fig. 1: Database Manipulation as an m-ary Set Operation.

There are a variety of set operations that can be applied to a database. They cannot be classified into a finite number of patterns. Therefore, we cannot prepare all such set operations as basic database operations.

However, if we could have a collection of elementary set operations by whose combination any database manipulation could be described, they would be a good abstraction. A number of trials to find such a collection of elementary set operations have been made. Information Algebra [CODASYL 1962] provided five set operations; Relational Algebra [Codd 1972] provided eight set operations; and Extended Set Theory [Childs 1968] provided about twenty set operations. Here, we will consider only one elementary set operation, which we call the *alpha operation* after Codd's Alpha Sublanguage [Codd 1971].

An alpha operation is the set operation which extracts all the lines (ordered sets of tuples) satisfying a logical function λ of lines out of a Cartesian product of m relations and, for each qualified line, generates a point (tuple) using a tuple generating function β of lines. It is

defined by

$$a[\beta:\lambda](R_1,R_2,\ldots,R_m)=\{\beta(\ell)\,|\,\ell\varepsilon s[\lambda](R_1,R_2,\ldots,R_m)\}$$

where

$$s[\lambda](R_1,R_2,\ldots,R_m)=\{\ell\,|\,\ell\varepsilon R_1\times R_2\times\ldots\times R_m\wedge\lambda(\ell)\}.$$

The s-operation defined as above is called a *search*. In particular, if m=1, the search operation

$$s[\lambda](R)=\{t\,|\,t\varepsilon R\ \lambda(t)\}$$

is called a *retrieval*.

The alpha operation must, therefore, be specified by two arguments, a logical function λ of lines (a predicate) and a tuple generating function β of lines, both defined on $R_1\times R_2\times\ldots\times R_m$, and have m operand relations. In Codd's terminology,

$$\beta:\lambda$$

is called an Alpha expression, in which the β is called the target list and the λ is called the relational calculus. In Information Algebra, the alpha operation is called bundling, in which the β is called the function of bundle (FOB) and the λ is called the bundling function.

In relational calculus, domains of free (tuple) variables are explicitly specified by range terms, that is,

$$t_1\varepsilon R_1\wedge t_2\varepsilon R_2\wedge\cdots\wedge t_m\varepsilon R_m\wedge\lambda(t_1,t_2,\ldots,t_m).$$

Range term of the form

$$\vee t_k\varepsilon R_k$$

is allowed only when it is conjunctively combined with another range term regarding t_k. In Alpha Sublanguage, such domain specifications together with quantifications for bound variables are described separately from the Alpha expression. In Codd's relational calculus, predicate λ is a logical combination of unit terms each composed of an attribute and another attribute or a constant combined by one of relational operators $=$, $\neq$, $>$, $<$, $\geq$ and $\leq$. Our definition somewhat relax this restriction. In fact, in our definition λ is an arbitrary logical function of lines. For example, we may have logical functions such as

$$A_1(t)<3\times A_2(t),$$

$$A_1(t_1)=A_2(t_2)+A_3(t_3),$$

$$\sin(A_1(t))+3\times\cos(A_2(t))>1/2$$

and

$$(A_1(t),q)/|A_1(t)||q|>0.75.$$

The last expression shows the search condition in cosine retrieval which frequently appears in pattern matching applications. $A_1(t)$ and q are respectively a pattern and a query vector, (p,q) is an inner product of p and q, and $|p|$ and $|q|$ are respectively the norm of p and q. Hence we deal with somewhat extended relational calculi.

Figure 2 illustrates an alpha operation. Database manipulation, in general, can be expressed by an alpha operation or a (procedural) combination of several alpha operations.

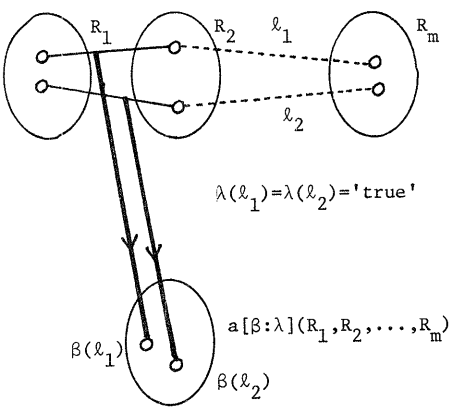

Fig.2: Alpha Operation

4. RELATIONA ALGEBRA

We will next discuss several sets of special alpha operations that plays certain important roles in database manipulation. Codd [1972] presented eight algebraic operations collectively called the *relational algebra*. Of them the following five are basic. The first two are binary set operations whose operands are two relations with a common set of attributes

$$A=\{A_1,A_2,\ldots,A_n\}.$$

(1) *difference* defined by

$$d(R_1,R_2)=a[\beta:\lambda](R_1)$$

with

$$\lambda(t)\equiv t'/R_2([A](t)\neq[A](t'))$$

and

$$\beta(t)\equiv t.$$

Here $[A](t)$ means concatenation of $A_k(t)$ values for $A_k\varepsilon A$.

(2) *union* defined by

$$u(R_1,R_2)=a[\beta:T](R_1,R_2,d(R_1,R_2))$$

where T is a logical function of lines that assigns 'true' value to every line (t_1,t_2,t_3) in $R_1\times R_2\times d(R_1,R_2)$, and

$$\beta(t_1,t_2,t_3)\equiv\begin{cases} t_1 & (\text{if } [A](t_1)=[A](t_2) \text{ or } [A](t_1)\neq[A](t_2)\wedge[A](t_1)=[A](t_3)) \\ t_2 & (\text{if } [A](t_1)\neq[A](t_2)\wedge[A](t_1)\neq[A](t_3)). \end{cases}$$

The above definition implies a procedural combination of two alpha operations. It may generate

a number of duplicated tuples, however, they are eliminated according to the definition of the alpha operation.

The next two are unary set operations. Let R be a relation with attribute set A.

(3) *selection* (or *restriction*) defined by

$$s[\lambda](R)=a[\beta:\lambda](R)$$

with

$$\beta(t)\equiv t.$$

The λ is an arbitrary logical function of points defined on R. It is no other than the retrieval operation we defined in 3. Hence we use the same notation.

(4) *projection* defined by

$$p[A'](R)=a[\beta:T](R)$$

where

$$\beta(t)\equiv(A_{k1}(t),A_{k2}(t),\ldots,A_{km'}(t))$$

with

$$A'=\{A_{k1},A_{k2},\ldots,A_{km'}\}.$$

This operation again includes elimination of duplicated tuples.

The last one is a binary operation whose operand relations are not necessarily with a common set of attributes. Let R_1 and R_2 be two relations respectively with attribute set

$$A_1=\{A_1,A_2,\ldots,A_i\}$$

and

$$A_2=\{A_{i+1},A_{i+2},\ldots,A_j\}.$$

(5) *multiplication* defined by

$$m(R_1,R_2)=a[\beta:T](R_1,R_2)$$

with

$$\beta(t_1,t_2)\equiv(A_1(t_1),A_2(t_1),\ldots,A_i(t_1),A_{i+1}(t_2),A_{i+2}(t_2),\ldots,A_j(t_2)).$$

The multiplication resembles making a Cartesin product. However, different from the Cartesian product, it generates a relation instead of a set of tuple pairs.

The following three are the set operations which Codd originally presented in addition to the above five. They are described as procedural combinations of the basic five operations. The first one is a binary operation applied to two relations with a common attribute set.

(6) *intersection* defined by

$$i(R_1,R_2)=d(R_1,d(R_1,R_2)).$$

The second is also a binary set operation but the two operand relations are not necessarily with a common attribute set.

(7) *join* defined by

$$j[\lambda](R_1,R_2)=s[\lambda'](m(R_1,R_2)),$$

where λ is an arbitrary logical function of lines of span 2 defined on $R_1\times R_2$, which has an equivalent effect to that of λ', a logical function of points defined on $m(R_1,R_2)$.

For example,

$$\lambda(t_1,t_2)\equiv A_1(t_1)=A_2(t_2)$$

defined on $R_1\times R_2$ is equivalent to

$$\lambda'(t)\equiv A_1(t)=A_2(t)$$

defined om $m(R_1,R_2)$.

The last operation is a binary set operation. Attribute set A_1 of the first operand relation R_1 must include attribute set A_2 of the second operand relation R_2.

(8) *division* defined by

$$v(R_1,R_2)=d(p[A_1-A_2](R_1),p[A_1-A_2](d(m(p[A_1-A_2](R_1),R_2),R_1))).$$

The division can also be defined directly by

$$v(R_1,R_2)=a[\beta:\lambda](p[A_1-A_2](R_1))$$

where

$$\lambda(t)\equiv t_2'/R_2\,\exists t_1'/R_1([A_2](t_1')=[A_2](t_2')\wedge[A_1-A_2](t_1')=[A_1-A_2](t))$$

and

$$\beta(t)\equiv t.$$

It is sure that all the relational algebra operations can be defined as an alpha operation or a procedural combination of alpha operations, and hence they can be described in terms of the alpha operation if it is implemented as a standard database procedure. However, relational algebra operations can be directly implemented to improve their execution performance.

Conversely, the search operation

$$s[\lambda](R_1,R_2,\ldots,R_m)$$

can be achieved by a procedural combination of several relational algebra operations. This was shown by Codd as the relational completeness of relational algebra [Codd 1972]. In this sense, the relational algebra is said to be equivalent in expressive power to that of relational calculus.

Note that the relational algebra is not equivalent in expressive power that possessed by the alpha operation. To make it equivalent, we must provide an additional operation:

(9) *transformation* defined by

$$t[\beta](R)\equiv a[\beta:T](R)$$

where β is an arbitrary tuple generating function of points.

5. INFORMATION ALGEBRA

CODASYL Development Committee [1962] presented five set operations as a set of elementary set operations. They were not devised with regard to the database management, but they can be considered as a database manipulation abstraction. The first three of these five are union, intersection and difference, which we have already defined in relational algebra. The fourth is called the bundling. Although the bundling was defined in a little conventional manner and few people recognized its equivalence to the alpha operation, the essentials of the bundling are those of the alpha operation.

The last operaion is the *summary* operation, which was called the glumping. It can be defined by

$$g[\gamma:\lambda](R)=a[\beta:T](a[\lambda:T](R))$$

with λ being a function of points defined on R and γ being a pair

$$\gamma\equiv(\alpha_1,\alpha_2,\ldots,\alpha_n:\lambda_1,\lambda_2,\ldots,\lambda_n)$$

composed of a list of n aggregate functions, $\alpha_1,\alpha_2,\ldots,\alpha_n$, and a list of n functions of points, λ_1, $\lambda_2,\ldots,\lambda_n$.

The firstly applied alpha operation $a[\lambda:T](R)$ generates a relation with only one attribute A (duplicate tuples are eleimnated), that is,

$$a[\lambda:T](R)=\{(\lambda(t))\,|\,t\in R\}.$$

For the next alpha operaion, the β is defined by

$$\beta(t')=(a_1,a_2,\ldots,a_n)$$

with

$$a_k=\alpha_{k\,(t\in R\wedge\lambda(t)=A(t'))}\lambda_k(t).$$

This operation classifies R according to the λ (usually called the control break) value, and then generates a (summary) tuple for each class.

Though the summary operation can be defined as a procedural combination of two alpha operaions as shown above, it can be implemented much more efficiently using a sequential processing applied to R In the sequence of λ value). A summary operaion is illustrated in Figure 3.

Union, intersection, difference, alpha operation and summary operation are collectively called the *information algebra*. Information algebra seems to be convenient in describing conventional data processing like various file maintenance operations.

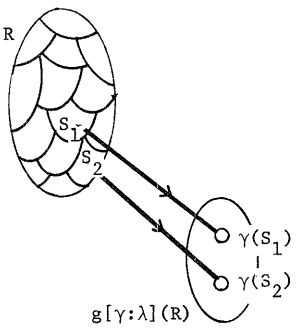

Fig.3: Summary Operation.

6. Imaginary Tuples

Bundling operation in Information Algebra was originally devised as an abstraction of collating m files (relations). In fact, collation is a basic procedure in the file maintenance which is the most important operation in traditional data processing. In the file maintenance involving one master and one or more transaction files, an alpha operation

$$a[\beta:\lambda](R_1,R_2,\ldots,R_m)$$

with

$$\lambda(t_1,t_2,\ldots,t_m)\equiv\bigwedge_{k=2}^{m}(A_1(t_1)=A_k(t_k)),$$

where R_1 is the master (relation) and $R_2,\ldots,R_m$ are transactions (relations), is a basic operation.

In many applications frequently occur the cases where a match is found except for some relations. For example, let R_1,R_2 and R_3 be three relations respectively representing an inventory master, a warehousing transaction and a delivery transaction in an inventory control application. Tuples in these relations are to be matched by a common primary key A_p. In this case, we have to consider seven different matches each specified by one of the following conditions:

1° $\lambda_1(t_1,t_2,t_3)\equiv A_p(t_1)=A_p(t_2)\wedge A_p(t_1)=A_p(t_3)$ (defined on $R_1\times R_2\times R_3$)

2° $\lambda_2(t_1,t_2)\equiv A_p(t_1)=A_p(t_2)\wedge t_3/R_3(A_p(t_1)\neq A_p(t_3))$ (defined on $R_1\times R_2$)

3° $\lambda_3(t_1,t_3)\equiv A_p(t_1)=A_p(t_3)\wedge t_2/R_2(A_p(t_1)\neq A_p(t_2))$ (defined on $R_1\times R_3$)

4° $\lambda_4(t_2,t_3)\equiv A_p(t_2)=A_p(t_3)\wedge t_1/R_1(A_p(t_1)\neq A_p(t_2))$ (defined on $R_2\times R_3$)

5° $\lambda_5(t_1)\equiv t_2/R_2 t_3/R_3(A_p(t_1)\neq A_p(t_2)\wedge A_p(t_1)\neq A_p(t_3))$ (defined on R_1)

6° $\lambda_6(t_2)\equiv t_1/R_1 t_3/R_3(A_p(t_1)\neq A_p(t_2)\wedge A_p(t_2)\neq A_p(t_3))$ (defined on R_2)

7° $\lambda_7(t_3)\equiv t_1/R_1 t_2/R_2(A_p(t_1)\neq A_p(t_3)\wedge A_p(t_2)\neq A_p(t_3))$ (defined on R_3).

The seven different matches are illustrated in Figure 4.

To each of the above seven matches, an appropriate tuple generating function of lines (of span 1, 2 or 3) must be given for generating a new tuple when a match has been found. Then the new inventory master is obtainable by seven separate alpha operations each corresponding to one of the

seven different matches, followed by union operations combining the results, that is,

$$u(a[\iota:\lambda_5](R_1),u(u(a[\beta_1:\lambda_1](R_1,R_2,R_3),u(a[\beta_2:\lambda_2](R_1,R_2),u(a[\beta_3:\lambda_3](R_1,R_3),$$
$$a[\beta_4:\lambda_4](R_2,R_3)))),u(a[\beta_6:\lambda_6](R_2),a[\beta_7:\lambda_7](R_3)))),$$

where ι is an identity tuple generating function

$$\iota(t) \equiv t,$$

and $\beta_1,\beta_2,\beta_3,\beta_4,\beta_6$ and β_7 are tuple generating functions defined respectively on $R_1\times R_2\times R_3$, $R_1\times R_2$, $R_1\times R_3$, $R_2\times R_3$, R_2 and R_3. The above very awkward description can be simplified if we introduce *imaginary tuples* defined below into the relations to be collated. The imaginary tuple τ_k of relation R_k is a tuple to be supplemented to R_k, which acts as if it were a real tuple with appropriate attribute values when λ value is to be evaluated but acts as a *null* tuple when β value is to be calculated. That is,

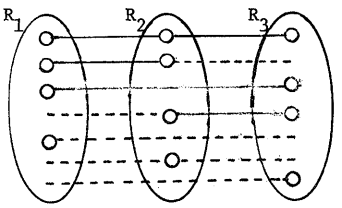

Fig.4: Different Matches in Collating Relations.

$$\lambda(t_1,t_2,\ldots,t_{k-1},\tau_k,t_{k+1},\ldots,t_m)$$

becomes 'true' if and only if

$$t_1\in R_1\wedge t_2\in R_2\wedge\ldots\wedge t_{k-1}\in R_{k-1}\wedge t_{k+1}\in R_{k+1}\wedge\ldots\wedge t_m\in R_m\wedge t_k/R_k(\wedge\lambda(t_1,t_2,\ldots,t_{k-1},t_{k+1},\ldots,t_m))$$

is satisfied and

$$\lambda(t_1,t_2,\ldots,t_{k-1},t_k',t_{k+1},\ldots,t_m)$$

would become 'true' if an appropriate tuple t_k' were appended to R_k. The tuple generating function β must be calculated without t_k' in this case.

In the above condition for activating imaginary tuple τ_k, other tuples $t_1,t_2,\ldots,t_{k-1},t_{k+1},\ldots,t_m$ can also be imaginary tuples. However, at least one of them must be a real tuple. Let us denote $R_1\cup\{\tau_k\}$ by $\bar{R}_k$.

Having introduced an imaginary tuple into each of R_1, R_2 and R_3, we can describe the subject inventory control example simply as

$$a[\beta:\lambda](\bar{R}_1,\bar{R}_2,\bar{R}_3)$$

with

$$\lambda(t_1,t_2,t_3)\equiv A_p(t_1)=A_p(t_2)\wedge A_p(t_1)=A_p(t_3)$$

and

$$\beta(t_1,t_2,t_3)\equiv\beta_1(t_1,t_2,t_3),\ \beta(t_1,t_2,\tau_3)\equiv\beta_2(t_1,t_2),\ \beta(t_1,\tau_2,t_3)\equiv\beta_3(t_1,t_3),$$
$$\beta(\tau_1,t_2,t_3)\equiv\beta_4(t_2,t_3),\ \beta(t_1,\tau_2,\tau_3)\equiv t_1,\ \beta(\tau_1,t_2,\tau_3)\equiv\beta_6(t_2)\text{ and }\beta(\tau_1,\tau_2,t_3)\equiv\beta_7(t_3).$$

Introduction of the imaginary tuples not only enables simpler description of some traditional data processing operations but also enables an efficient implementation of the operations to which a sequential collation procedure is applicable.

7. NAVIGATIONS

Sometimes it is convenient to distinguish relationship relations from other entity relations. In particular, data models like Entity-Relationship Model [Chen 1976], Information Space Model [Kobayashi 1975] and Coset Model [Bachman 1974] assume some normal forms regarding relationship relation [Kobayashi 1980]. When we use such a model as the basis of database design, some special alpha operations become very important.

Let R_1 and R_2 be two relations respectively with attribute set A_1 and A_2. Let R be a relationship relation defined between R_1 and R_2 (R_1 and R_2 are not necessarily different relations). The attribute set of R is composed of A_{r1} which is a subset of A_1, A_{r2} which is a subset of A_2, and a set A_o of other attributes. Here $[A_{r1}]$ and $[A_{r2}]$ are respectively the selected primary key of R_1 and R_2.

The following four alpha operations are collectively called the *navigations* along relationship relation R:

(1) $n_f(t_1,R_2,R)=a[\beta:\lambda](R_2,R)$

with

$$\lambda(t_2,t)\equiv[A_{r1}](t)=[A_{r1}](t_1)\wedge[A_{r2}](t)=[A_{r2}](t_2)$$

and

$$\beta(t_2,t)\equiv(A_1(t),A_2(t),\ldots,A_h(t),A_{i+1}(t_2),A_{i+2}(t_2),\ldots,A_j(t_2))$$

where

$$A_o=\{A_1,A_2,\ldots,A_h\}\text{ and }A_2=\{A_{i+1},A_{i+2},\ldots,A_j\}.$$

(2) $n_b(R_1,t_2,R)=a[\beta:\lambda](R_1,R)$

with

$$\lambda(t_1,t)\equiv[A_{r2}](t)=[A_{r2}](t_2)\wedge[A_{r1}](t)=[A_{r1}](t_1)$$

and

$$\beta(t_1,t)\equiv(A_1(t),A_2(t),\ldots,A_h(t),A_{h+1}(t_1),A_{h+2}(t_1),\ldots,A_i(t_1))$$

where

$$A_1=\{A_{h+1},A_{h+2},\ldots,A_i\}.$$

(3) $n_a(R_1,R)=a[\beta:\lambda](R_1)$

with

$$\lambda(t_1)\equiv t/R([A_{r1}](t)\neq[A_{r1}](t_1))$$

and

$$\beta(t_1) \equiv t_1.$$

(4) $n_b(R_2,R)=a[\beta:\lambda](R_2)$

with $\lambda(t_2) \equiv t/R([A_{r2}](t) \neq [A_{r2}](t_2))$

and

$$\beta(t_2)=t_2.$$

The n_f (forward navigation) and n_b (backward navigation) retrieve all the tuples that are connected respectively with t_1 and with t_2 by some tuples in R. The n_a (apex navigation) and n_t (terminal navigation) retrieve all the *apices* and all the *terminals* respectively with respect to R. By these four operations, we can traverse the relationship relation R as shown in Figure 5.

Navigations are special alpha operations; however, they can be implemented directly in some more efficient ways, in particular, when binary relationship relations are physically represented in a different way from that in which entity relations are represented.

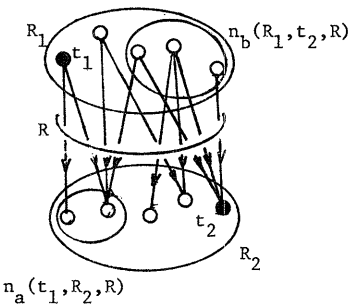

Fig.5: Forward and Backward Navigation.

8. TUPLE-BY-TUPLE OPERATIONS

We may have a data manipulation language with only one statement corresponding to the alpha operation. Because of the relational completeness, we may have another data manipulation language with statements embodying the relational algebra plus transformation (See 4). Also, we may have one with statements corresponding to the information algebra. These statement languages can be enriched with statements embodying navigations. (Without navigations discrimination between entity and relationship relations is not so significant.)

These statement languages are in most cases suitable for casual users who make queries to the database in an interactive processing mode. However, there are other user types, for example, realtime users who use the database in their routine works, parameter users who retrieve and update the database in batch mode, and programmers who implement end-user languages for various types of users. The above statement languages are not always suitable for these users. This is because all the alpha operations are "set" operations.

It is true that the alpha operation is powerful enough in describing a diversity of data manipulations. Given a data manipulation, we can describe it as an alpha operation or a procedural combination of several alpha operations, which we will call an *alpha construct*. However, the alpha construct have several difficulties.

(1) It usually contains many redundant data transfers.
(2) Integrity and security become difficult problems.
(3) It is not fit for being integrated into the host language.

For example, let us consider a simple file maintenance operation, which is written as an alpha construct

$$u(d(R_1,a[\phi:\lambda](R_1,R_2),a[\beta:\lambda](R_1,R_2)),$$

where ϕ is a tuple generating function that extracts the first component of a pair of tuples, that is,

$$\phi(t_1,t_2)=t_1.$$

The above alpha construct updates master relation R_1 using transaction relation R_2. It includes four alpha operations that must be executed procedurally like

begin R:=a[ϕ:λ](R_1,R_2); R':=d(R_1,R); R'':=a[β:λ](R_1,R_2); R_1:=u(R',R'') end;

(The last assignment statement implies that the old master is to be replaced by the new master.) It is obvious that this procedure contains many redundant data transfers, that can be avoided if the whole process is executed in parallel using a traditional sequential collation. Integrity and security should be specified for the whole file maintenance operation. It is very difficult to specify them for each component of the alpha construct.

In order to avoid these difficulties, we may decompose the alpha operation into certain finer operations, and combine these finer operations in an appropriate sequence to achive the given data manipulation more efficiently. By arranging them in a proper sequence, we can remove duplicate data transfers, that were inavoidable in a procedural execution of alpha operations. Also, the integrity and security problems can be dealt with much more easily.

Three classes of such finer operations can be considered. The first one fetches a tuple or an ordered set of tuples into one or a set of *workspaces*, Usually one workspace is provided for each database relation. A workspace can accommodate a tuple at a time. The first class is called the *read class* operation. The second class includes operations to generate a new tuple using one or

more tuples in the workspaces. The result is stored into an appropriate workspace. This class is called the *calculate class*. It also contains some control transfer operations that determine the operation to be executed next according to the value of a certain function of tuples.

The last class, which we will call the *write class*, is composed of the update operations, which add, delete or replace a tuple in a database relation with reference to a tuple currently in a specific workspace.

These three classes are collectively called *tuple-by-tuple operations* (piped-mode operations [Codd 1971]). Among them, the calculate class operations can be committed to the care of the host language, while the other classes can be implemented as database routines to be supplemented to it.

Read class operations can be obtained by decomposing the search operation $s[\lambda](R_1,R_2,\ldots,R_m)$. The qualified tuples are arranged in a certain sequence, for example, in the sequence of the value of a specified function λ' of lines, and fetched one by one according to this sequence. A *cursor* must be provided for each relation R_k. If relationship relations are distinguished from entity relations, another type of read class operation can be obtained by decomposing navigations.

Write class operations are composed of add, delete and replace a tuple. If relationship relations are distinguished from entity relations, some additional write class operations that update relationship relations together with entity relations on which they are defined can be provided.

9. CONCLUSION

We have so far discussed three different abstractions of data manipulation; one composed of only one set operation, the alpha operation; one composed of various algebraic operations; and one composed of three classes of tuple-by-tuple operations. The first two can be implemented as a statement language for casual users. The last one leads us to implementing host language type database management systems.

There can be many more data manipulation abstraction levels which are suitable for various classes of end-users. End-user languages for such users can be implemented using the host language enriched by the database routines. Different from the three abstractions thus far presented, they are usually not necessarily equivalent in expressive power to the alpha operation.

REFERENCES

[Bachman 1974] C.W.Bachman, The Data Structure Set Model, *Proc. ACM SIGMOD '74*, pp.1-10.

[Chen 1976] P.P.Chen, The Entity-Relationship Model: Toward a Unified View of Data, *ACM Trans. Database Systems*, 1 (1), pp.9-36.

[Childs 1968] D.L.Childs, Description of a Set-Theoretic Data Structure, *Proc. AFIPS '68 FJCC*, pp. 557-568.

[CODASYL 1962] CODASYL Development Committee, An Information Algebra: Phase I Report, *Comm. ACM*, 18 (10) pp.580-588.

[Codd 1971] E.F.Codd, A Data Base Sublanguage Founded on the Relational Calculus, *Proc. ACM SIGFIDET '71*, pp.35-68.

[Codd 1972] E.F.Codd, Relational Completeness of Data Base Sublanguages, in "Data Base Systems," Courant Computer Science Symposium, 6, Prentice-Hall, Englewood Cliffs , NJ, pp.65-98.

[Kobayashi 1975] I.Kobayashi, Information and Information Processing Structure, *Information Systems*, 1 (2), pp. 39-50.

[Kobayashi 1980] I.Kobayashi, An Overview of the Database Technology, SANNO College of Management and Informatics, TRCS-4. Also to be included in "Advances of Information Systems Science," 8, Plenum Press, NY.