A Functional Method of Data Processing based on Relational Algebra

Maria E. Orlowska
Department of Computer Science
University of Queensland, Brisbane, Australia
Keith G. Jeffery
Systems Engineering Division, Rutherford Appleton Laboratory
Oxfordshire, U.K.

Abstract

The design specification of processes within a relational database system is addressed. A novel method based on a rigorous approach to the correspondence between the attribute requirements of a process, and the attribute content and structure of a relational database, is presented.

A method - including algorithmic assists - is presented, based on this representation.

1. INTRODUCTION

1.1 Objective

This paper introduces a new perspective and approach to a difficult area of systems engineering. The theoretical background is original, and the proposed use of the theory is a novel method of software development.

1.2 Context

The systems engineering cycle is well-defined; conventionally the 'waterfall model' (Boehm, 1984) is used in commercial methods (commonly called methodologies) such as

SSADM (NCC, 1990), JSD (Jackson, 1983) and others. More recent methods concentrate on progressive refinement with 'fast prototyping' (Mayhew, 1990) or the 'spiral model' (Boehm, 1986). In both types of method, the steps are the same, although they are repeated at progressively greater degrees of refinement in the more recent methods. The steps can be represented diagrammatically (Fig 1); this diagram shows clearly the steps or stages and the activity and/or deliverables at each stage in each of two areas of work: data analysis and procedural (process) analysis.

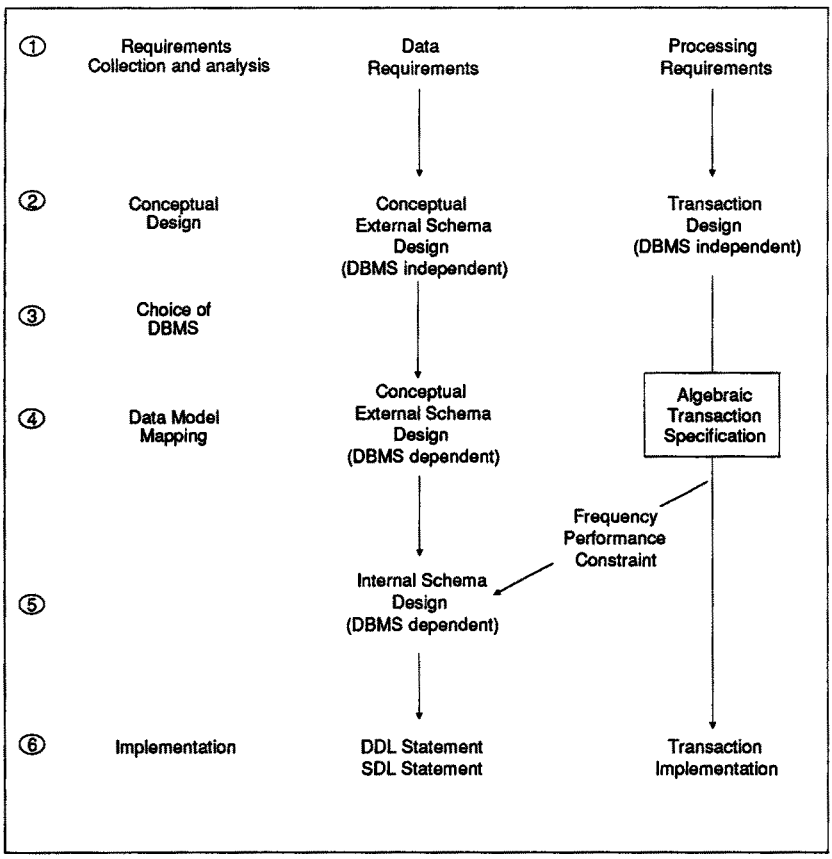

Figure 1.

The essential point of the diagram is that there are corresponding stages in each of the two areas of work, and that there are inter-dependencies. This area (the inter-dependencies)

has always been a weakness of system development methods, because of the separation of the data and process analyses. Worse, whereas data analysis has received much attention - resulting in well-known techniques with sound theoretical principles such as entity analysis (Chen, 1976) and normalisation (Codd, 1971) - process or procedural analysis has received less attention. Indeed, CASE tools from commercial vendors are available for the data analysis techniques, whereas equivalent tools for the process analysis techniques - such as dataflow diagrams (Jackson, 1983) or Petri-Nets (Solvberg, 1986) - are still in the domain of research. Thus, our intention was to concentrate on a framework for process analysis to cover the stages from user requirement analysis through to transaction implementation. We wished to ensure that the analysis method remained linked inter-dependently with the data analysis. We wished to provide a sound theoretical basis for process analysis. Finally we wished to develop a method that led towards the automated verification of specifications and generation of code.

1.3 Overview

The essence of the work is to consider together a hypergraph representation of a relational schema, and a functional view of transactions as sequences of atomic operators including - as a majority - the operators of the relational algebra. Thus, at the stage 4 of database system design (Fig. 1) we have introduced a component into the process analysis stream of the method. The method proposed here is restricted to relational theory; its possible extension to other paradigms is under investigation.

The method unifies data and process analysis within the relational theory at the point where normalized relations and relational algebra operators (with additional functional operators to cover the data processing functions outside of the scope of the relational algebra) meet.

2. PROCESS CONCEPT

Process requirements predicate the data manipulation capabilities of the system and include their expected frequencies and required performance. The processing requirements must be semantically correct and consistent so that the processing does not violate any constraints imposed by the information requirements. Before any formal or informal description of a process, it is essential to understand this notion. Indeed, it is this very notion which ties together the streams of data analysis and process analysis.

The processing requirements derived from the requirements analysis are translated - or preferably mapped - to the transaction design at the external (Stage 2 of Fig. 1) level.

The transaction design involves the integration of

(a) the process which satisfies the processing requirement

(b) the data which is necessary for input, and is required as output

(c) considerations of integrity, security, privacy, and performance

Our notion of process is a sequence of relational operators (that is they act on relations) drawn from the relational algebra and other functions for computation. These operators we call atomic operators; they are indivisible within the context of manipulating relations, and can be agglomerated into processes to meet the processing requirements.

Our notion of transaction is that optimal unit of data processing that performs an atomic operation on the database, and leaves the database in a consistent state. It is clear that in our scheme there are two levels of transaction, corresponding to the atomic operator and process levels. It is an interesting area of research to discover the relationship between the lower level transactions and relations, and the higher level transactions and more complex data structures. Indeed, the higher level transactions could include the notion

of locking intent. Further discussion of this is beyond the scope of the present paper. The classic approach to describe a process is the black box model, where a name is assigned to the process and two streams are identified: input and output.

Fig. 2

When describing transaction processing or a process in an Information Systems context, it is easy to distinguish between different kinds of inputs. These are:

(a) external entities that interact with the system, such as user's requests to activate the process, and interactively support the process execution by providing data entry

(b) stored data used by this process.

The output resulting from the successful process execution by the system can also be of two types. The report can be produced by the system for external usage by a user or the stored data are modified accordingly.

In summary, the following flows of data in the system may eventuate:

- between two processes
- from a data store to a process
- from a process to a data store
- from an external source to a process
- from a process to a sink

These flows are external to the process itself, and they "support" the process to secure its feasible execution.

3. *THEORETICAL FOUNDATIONS*

In this paper, we understand a process to be any activity on the database which is activated by an external entity and which results in either a report produced or the data modified. According to our assumption, the database of a design system is relational and fully normalized. We therefore assume that Stage 4 of the data analysis stream of the design process (Fig. 1) has been completed.

In this section, we first introduce a useful graphical representation of a normalized relational schema, then discuss and justify the relational algebra approach.

3.1 Representation of Normalized Relational Schemas

A database schema is naturally viewed as a hypergraph. This hypergraph is similar to an ordinary undirected graph with the exception that the edges are arbitrary non-empty sets of nodes, rather than just doubletons.

A hypergraph H is a pair (N, E) where N is a set of nodes and E consists of a non-empty subset of N, called hyperedges. H is reduced if no edge in E properly contains another edge and every node is in some edge. If R is a database schema over $U = \{A_1, ..., A_n\}$, then it may be viewed as the hypergraph $H_R = (U, R)$; that is, the attributes in U are nodes in the hypergraph and the relation schemas of R are edges of H_R. We shall simply use H_R in place of (U, R) when dealing with the hypergraph that R represents.

Example 1.

Let R = {ABC, BCD, DE, BE}, where U= ABCDE.

The hypergraph representation is the following:

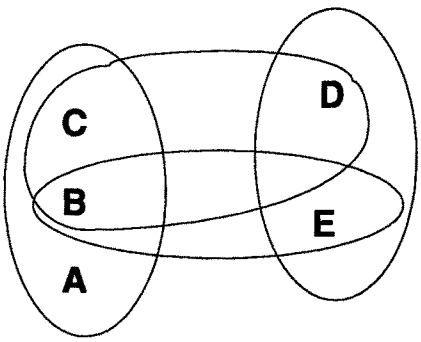

Fig. 3

In order to find a simple graphical representation of a normalized relational schema, it is necessary to express not only the set of relational schemas but also the set of basic constraints. A simple notation is introduced to both enrich hypergraph representation and cover some of the important relational constraints on the graph. To express a primary key of a relation schema, we interrupt the line of the relevant edge and insert into this break the letter denoting the primary key. For example, if R = ABC and A is the primary key, the following representation would be appropriate:

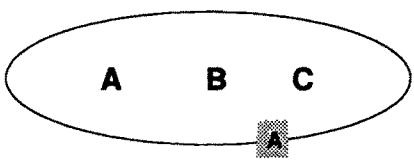

Fig. 4

The foreign keys do not need special graphical notation since they are nicely expressed by the hypergraph itself as the common part of any two hyperedges such that the primary key of one of them belongs to it. More generally, all inclusion constraints are expressed as an intersection of edges in H_R. Any relational schema can be fully represented by a hypergraph:

Example 2.

Let R be

DEPT (DEPT#, DNAME, MGR_EMP#, BUDGET)

EMP (EMP#, ENAME, DEPT#, SALARY)

where primary keys are underlined as follows:

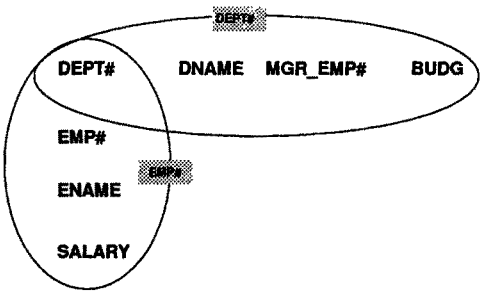

Fig. 5

The following are some useful observations about hypergraphs being used as a model for normalized relation schemas.

Let $R = \{R_1, ..., R_p\}$ be a normalized relation schema.

Let H_R be the hypergraph with edges $\{E_1, ..., E_p\}$.

Observation 1. H_R is connected (i.e., there is a path between each pair of nodes of H_R).

Observation 2. H_R is reduced.

Observation 3. Between any two edges E_i, E_j of H_R, a sequence of edges exists called J_{ij} (J_{ij} can be empty) such that

$$(E_i \cup \bigcup_{k \in J_{ij}} E_k) \cap E_j \supseteq K_j$$

where K_j is a superkey for R_j. (For further details, proofs and other propositions, see Orlowska, 1990.) Note that for a normalized relational schema intersection of two hyperedges of H_R, this intersection is not necessarily a superkey, as illustrated later in Example 3. We refer to these observations again in Section 4.

3.2 Relational algebra

We term a formal system that can express updates to relations an *update system. A query* to a relational database is a computation upon relations that yields other relations. *A query system* such as the relational algebra, is a formal system for expressing queries. Query systems form the underlying structure of query languages, the special purpose programming languages used in database systems to formulate commands.

Although a few query languages are based on relational algebra, most are based on either calculus or tableaux. The main reason is that the algebra is a procedural system, while the other two are non-procedural, that is, an expression in relational algebra gives a set of operations on relations and an order in which to perform them (up to certain associativities).

Thus, query languages based on non-procedural systems tend to be higher level, relieving their users of having to determine a derived answer. The burden of this determination naturally falls to the query language processor of the given database system.

However, where the user requirement is for data processing functions interspersed with calculation or transformation functions - such as the statistics, graphics or mathematical processing found commonly in scientific and engineering systems - then the relational calculus is less attractive. SQL is not a complete language, and lacks the very facilities needed not only by scientists and engineers, but also accountants, managers, and administrators. Furthermore, it is common for those users of a system to be able to

specify their requirements as a series of steps or processes rather than in an overall predicate form.

It is not easy to extend SQL in a consistent way to form a suitable language for expressing these requirements. Indeed, it is practicable to use a third generation language and SQL statements. In contrast, it is possible to extend the relational algebra with additional functions that act on relations to provide a sufficiently complete data processing system (Jeffery and Gill, 1976) for any application area, so providing a set of operators that act on relations to perform the required functions which can be used with the relational operators. Chris Date has described the relational algebra recently (Date, 1990) as a 'high level and symbolic representation of the users' intent" (page 314).

We assume that the reader is familiar with definitions and properties of all relational operators (Yang, 1986). We refer to the operations union, intersection, difference, active complement, select, project, natural join, division renaming and theta-join, along with constant relations and regular relations, as the relational algebra, (for formal definition see Yang, 1986). Any expression legally formed using these operators and relations is an algebraic expression.

Observation 4. Given an algebraic expression L, and the current values of all the relations in L, we can evaluate L to yield a simple relation. L therefore represents a mapping from sets of relations to single relations. Actually, the set of attributes, the domains, and the set of comparators we use limit the mappings we may define.

The relation names, r_1, r_2, ..., r_p are analogous to program variables, where r_i ranges over relations on schema R_i. Our notation is a bit ambiguous in that we use r_i both as a relation name and to denote the current state of a relation. The same ambiguity arises when discussing variables in programs; this is the problem denotational semantics tries

to address. The ambiguity only becomes clumsy when we view an algebraic expression as a mapping. Since the result of every relational operation we use is a simple relation, every algebraic expression defines a function that maps a set of relations to a single relation. *The schema of the single relation depends only on the schemas for the set of relations.*

Let the schema of an algebraic expression L, denoted S(L), be the relation schema of the relation.

Observation 5. We can define S(L) recursively accordingly to the following rules.

a) If L is r_i, then S(L) is the relation schema for r_i.

b) If L is a constant relation, S(L) is the schema for the constant relation.

c) If $L = L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$, $\widetilde{L_1}$ or $\sigma_C(L_1)$ where C is some set of conditions, then $S(L) = S(L_1)$.

d) If $L = \pi_X (L_1)$, then $S(L) = X$.

e) If $L = L_1 \bowtie L_2$ or $L_1[C]L_2$, for some condition C, then $S(L) = S(L_1) \cup S(L_2)$.

f) If $L = \delta_{A1, A2, ..., Ak \leftarrow B1, B2, ..., Bk}(L_1)$, then $S(L) = (S(L_1) - A_1, A_2, ..., A_k) B_1, B_2, ..., B_k$.

In general, if L is an algebraic expression involving relation names $S_1, S_2, ..., S_q$ corresponding to schemas $W_1, W_2, ..., W_q$, then L is a mapping

$$L : Rel (W_1) \times Rel (W_2) \times ... \times Rel(W_q) \rightarrow Rel (S(L))$$

where Rel(R) is the set of all relations with schema R. Informally then, for all relational operations, there is exactly one, well-defined relation schema of the resulting selection

from the operation. This observation, in conjunction with Observation 1, 2, and 3, provides the core for our general idea of an algebraic transaction specification.

4. TRANSACTION SPECIFICATION LANGUAGE (TSL)

We specify the processing requirements of a database system using a transaction specification language. The language is the specification facility in the relation database design system and provides a number of useful functions as described below:

1. Model parameter specification

In addition to providing a means for describing the processes of the system, it provides a natural and easy way to define such parameters as frequency of transaction execution, number of instances of functions and so on.

2. Dynamic information specification

Control constructs and operations are used to capture access patterns and query and update activity. This can be used to redesign our relational database to better facilitate frequent requirements.

3. Effects of schema restructuring

Schema restructuring operations are automatically reflected in the process model (given by our TSL) as well as the data model.

4. Design simulations

Having a process model enables us to simulate the effect of database transactions.

5. Correctness verification of transactions skeletons

When a design is chosen, and a high-level specification for the application is completed, verification of correctness and consistency of a transaction can be performed.

The language contains a set of relational operations to model data retrieval and manipulation. Since the only information-carrying objects in a relational model are relations and constraints, they are basic elements for data access. Every transaction on a relational database is a sequence of relational operations or, in other words, is an algebraic expression. Note that any "insert into r" can be expressed by the union of target relation r and hypothetical relation with only the tuples which we intend to insert into r. Similarly, "delete from" is expressed by the difference operator. Any modifications can be expressed as a combination of deletion and insertion.

To keep our discussion rigorous and systematic, let us remember the notions of transaction discussed in section 3. We distinguish the atomic transaction, expressible using only one relational operation, and the process transaction representing a unit of processing meaningful to the user and consisting of a sequence of atomic transactions.

In order to represent more complex data manipulations, the flow control between process transactions can be expressed using statements exactly as found in conventional procedural programming:

a) FOR_WHILE

b) IF_THEN_ELSE

Additionally, the invocation of a function (atomic operator) within a process-transaction can be achieved using the conventional statement:

c) PERFORM (parameters)

It may be necessary to bind specific values to variables used as parameters:

d) ASSIGN

While it is possible to envisage the introduction of processing directives - possible as a feature of the language compilation optimization - such as:

e) PARALLEL

f) SEQUENTIAL

to indicate that the following operators are to be executed in parallel (if possible) or strictly sequentially as directed.

Now we informally describe our method of efficient transaction specification:

Let $R = \{R_1, ..., R_p\}$ be a normalized relational schema. Let H_R be the hypergraph for R. Every atomic transaction T is a function defined on some subsets of attributes $U = \{A_1, ..., A_n\}$, denoted $T(A_1, ..., A_k)$ where $\{A_1, ..., A_k\} \subseteq U$.

We project $\{A1, ..., A_k\}$ onto the hypergraph H_R and identify all those edges E_i, $i \in \{1 ... p\}$, such that $E_i \cap \{A_1, ..., A_k\} \neq \emptyset$.

The following cases may occur:

1. $\exists \; i \in \{1 ... p\}; \qquad \{A_1, ..., A_k\} \subseteq E_i$

2. $\exists \; \{ i,j\} \subset \{1 ... p\}; \; \{A_1, ..., A_k\} \subseteq E_i \cup E_j$

1 and 2 can be generalised as follows:

$$\exists \; I = \{i_1, ..., i_l\} \subseteq \{1 ... p\}; \; \{A_1, ..., A_k\} \subseteq \bigcup_{i \in I} E_i$$

Obviously, all E_i, $i \in \{i_1, ..., i_j\}$ are the representations of all relations which must be involved in computation of transaction T. (We show soon that not only those E_i's are involved, but some others as well.)

Observation 2 from section 4.1 now needs to be analyzed. If all attributes of transaction T belong to one edge of the H_R graph, then obviously the construction of the appropriate

relational operator is simple, and may be directly derived from the schema definition of Observation 5. We describe this case formally later.

A more complex case may be

$$\{A_1, ..., A_k\} \subseteq \cup E_i, \quad i \in I$$

In this case, some join computation is unavoidable. The proper decision here is critical to correctness of the transaction. To place this computational problem in its proper perspective, the following first illustrates the problem simply:

Example 3.

Assume that $R = \{R_1, R_2, R_3, R_4, R_5\}$; $\forall$ i $r_i(R_i)$ be a relation on schema R_i.

$R_1 = \overline{A}BCDK$ $\quad K_1 = A$

$R_2 = \overline{B}EF$ $\quad K_2 = B$

$R_3 = \overline{C}I$ $\quad K_3 = C$

$R_4 = \overline{\overline{CD}GJ}$ $\quad K_4 = CD$

$R_5 = \overline{\overline{CDEFH}}$ $\quad K_5 = CDEF$

Let us show H_R for this R:

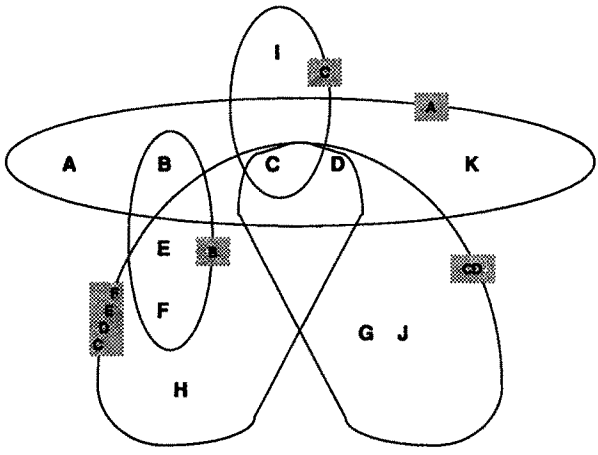

Fig. 6

Let T be the transaction T(B, H).

The question is how to construct the appropriate operator(s) to build a "proper" relation on schema (B, H) and at the same time, have a guarantee of correctness and consistency of the database after transaction completion arises. It should be clear that

$$r_2 (R_2) \bowtie r_5(R_5)$$

would not be a correct elementary transaction to finally produce a relation on schema (B,H) simply because $EF = E_2 \cap E_5$ is not a superkey for either E_2 nor E_5.
The other possibility would be

$$r_1(R_1) \bowtie r_5(R_5)$$

This is also not successful since

$$E_1 \cap E_5 = CD$$

but CD is a superkey for R_3 but not for R_1 or R_5.
The only correct computation can be carried out in the following way:

$$r_1(R_1) \bowtie r_2(R_2) \bowtie r_5(R_5)$$

because

$E_1 \cap E_2 = B$ (B is a superkey for E_2)

$E_1 \cap E_5 = CD$ ⎫
⎬ $(CD \cup EF)$ is a superkey for R_5
$E_2 \cap E_5 = EF$ ⎭ or $(E_1 \cup E_2) \cap E_5 \supseteq CDEF$ where CDEF is a key of E_5
(see Observation 2)

In conclusion, to construct semantically valid (sensible) execution of any transaction T(B, H), one must compute the natural join between three relations.

r_1, r_2 and r_5

and then make the projection on (B, H) (and continue if required with any relational operator on schema (B, H), such as select with condition).

Note that we have not used any semantic meaning of the relations involved and that the above statement is always valid.

We generalize this example and show a formal algorithm for join computation (or rather, join construction) when the transaction's attributes belong to more than one edge of H_R.

For clarity of presentation, we introduce one more graphical notation: *complete intersection graph for H_R.*

Let $R = \{R_1, ..., R_p\}$ be a database schema over R.

The *complete intersection graph* for R, denoted I_R, (equivalent in its expressive power to H_R) is the complete undirected graph on nodes R_1, R_2, ..., R_p and with the edge labels chosen from the subsets of R. For an edge
$e = (R_i, R_j)$, the label of e, denoted LE(e) is $R_i \cap R_j$. An intersection graph for R is any subgraph of I_R formed by removing only the edges. (We generally omit any edge e where $LE(e) = \emptyset$.) A subgraph of I_R, we call a *superkey connected graph* if and only if each edge is a superkey for some R_i and union of all nodes is equal to R.

Example 4.

Let R = {R_1, R_2, R_3, R_4} be the following data schema over R = ABCDEFG

$R_1 = \overline{ABC}$

$R_2 = \overline{C}DE$

$R_3 = \overline{E}DFG$

$R_4 = \overline{BDFG}$

where keys are marked for each schema. The complete intersection graph is shown below (Fig. 7):

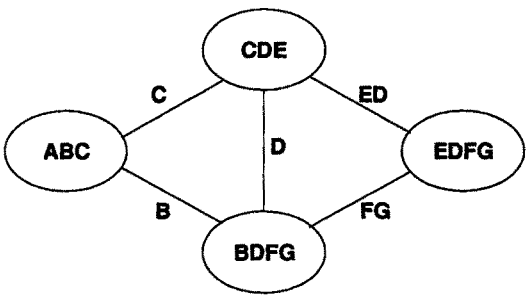

Fig. 7

The intersection graph for R may be the following:

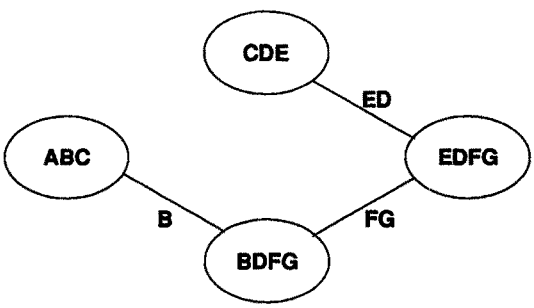

Fig. 8

The superkey connected graph of I_R is the following subgraph:

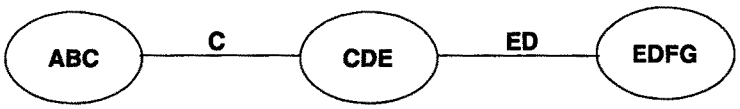

Fig. 9

C and ED are both superkeys in R and union of all nodes is equal to R.

The graph at Fig. 8 is not a superkey connected graph since the edges B and FG are not superkeys, However, the union condition is satisfied.

It is important to understand that the superkey connection graph provides:

(a) the complete definition of the between relation connections such that

(i) complex data structures can be accessed

(ii) join conditions can be expressed

(b) a structural property of the overall data (between relations) that can be stored and used for expedited access.

This graphic representation will be useful to provide an efficient method of construction of a *superkey connection set* for R.

For a relational schema $R = \{R_1, ..., R_p\}$, we define a *superkey connection set* J_R as follows:

$$J_R = \{J_{ij}\} \quad \text{for } i<j, \qquad i, j = 1, ..., p$$

such that

J_{ij} = {set of all edges in H_R which are merged to establish a superkey connected subgraph of I_R including E_i and E_j}.

We propose the following method to compute Jij sets for J_R. This is, however, only an overview. For details, see (Orlowska, 90).

Procedure: J_{ij} construction

Identify a superkey connection graph SI_R for I_R.

For the pair of nodes (n, m) check:

if this pair is connected to S_{IR} graph, then corresponding $J_{nm} = \varnothing$.

Otherwise

Find the union of all nodes connected to n (the first star of node n; $star^{(1)}(n)$) and check superkey connection of this set with node m.

or

Find the $star^{(1)}(m)$ and check its superkey connection with the node n.

If one of these conditions is satisfied then

Identify the node(s) in the star which are essential for providing superkey connection and

Jnm = {set of identified nodes}

Otherwise construct the next star and repeat the above procedure.

Example 5.

We show the superkey connection set J_R for all pairs of edges for R from Example 3.

$R = \{R_1, R_2, R_3, R_4, R_5\}$

$J_R = \{J_{12}, J_{13}, J_{14}, J_{15}, J_{23}, J_{24}, J_{25}, J_{34}, J_{35}, J_{45}\}$ where

$J_{12} = J_{13} = J_{14} = J_{34} = J_{35} = J_{45} = \varnothing$

because for each pair (i, j), $E_i \cap E_j \supseteq K_i$ or $E_i \cap E_j \supseteq K_j$; in other words, those nodes of the J_R graph are superkey connected.

$J_{15} = \{2\}$ $J_{23} = \{1\}$ $J_{24} = \{1\}$ $J_{25} = \{1\}$

because, for example, to connect edges E_1 and E_5 by superkey connection, we must join R_1, R_5 and R_2 as we informally discussed previously in Example 3.

5. RELATIONAL ALGEBRA PROCESS SPECIFICATION

Once theoretical foundations have been established, the method of transaction specification is simple and effective and may be summarised as follows:

Given

(1) Relational database design

$R = \{R_1, R_2, ..., R_p\}$

$K = \{K_1, K_2, ..., K_p\}$

$\forall_i$, K_i is a primary key for R_i.

(2) Narrative specification of the set of transactions T or Data Flow Diagram.

Output: Formal relational specification of the set T.

Method:

(1) Define H_R for R

- all inclusion dependencies are automatically identified
- foreign keys dependencies are established automatically.

(2) Construct connection set J_R for H_R using J_{ij} construction.

(3) For each transaction, identify atomic transactions. For each atomic transaction, use the following mapping:

- map all attributes of transaction $T(A_1, ..., A_k)$ into H_R

- identify edges E_i such that $(A_1, ..., A_k) \cap E_i \neq \emptyset$.

If only one edge exists, say E_i, such that $E_i \cap \{A_1, ..., A_k\} \neq \emptyset$, then

$E_i \cap \{A_1, ..., A_k\} \neq \emptyset$, then

if $\{A_1, ..., A_k\} \subset E_i$ then

$r \leftarrow \pi_{\{A1, ..., Ak\}} (r_i)$ and

$r \leftarrow \sigma_C(r)$

else

$r \leftarrow \sigma_C(r_i)$

else

Let $\{A_1, ..., A_k\} \cap \{E_{i1}, E_{i2}, ..., E_{in}\} \neq \emptyset$ for each different pair of edges (E_{i1}, E_{i2}), (E_{i2}, E_{i3}) ... (E_{in-1}, E_{in}) (the order of the pairs is irrelevant). Find the corresponding entry in the set J_R and

$$r \leftarrow \underbrace{r_{i1} \bowtie, ..., \bowtie r_{in}}$$

(relations from J_R connections)

$r \leftarrow \pi_{\{A1, ..., Ak\}} (r)$

$r \leftarrow \sigma_C(r)$

where C is a set of conditions concerning only the subset of $\{A_1, ..., A_k\}$, r_i is a relation on schema R_i.

According to our definition of a process, any process is able to be described in the form of a sequence of relational operations (enriched by specified statements in Section 4.

Example 6

To continue with Example 3:

Let $T_1(A, K)$

By projecting (A,K) into H_R (Fig. 6), we can easily identify that

$(A,K) \subset R_1$

There is no join computation in the evaluation of this transaction, then

$r \leftarrow \pi_{\{AK\}} (r_1)$

$r \leftarrow \sigma_C(r)$

Now, let T_2 (B, H)

By projecting (B, H) into HR, we see that R_1 and R_5 or R_2 and R_5 are involved. This time, the join is unavoidable. Corresponding values of J_R are

$J_{15} = \{2\}$ in the first case

or

$J25 = \{1\}$ in the second.

According to our procedure

$r \leftarrow r_1 \bowtie r_2 \bowtie r_5$ or

$(r \leftarrow r_1 \bowtie r_5 \bowtie r_5)$

$r \leftarrow \pi_{\{B\ H\}} (r)$

$r \leftarrow \sigma_C(r)$

When all steps are finally combined, we have

$$r \leftarrow \sigma_C(r)\ \pi_{\{B\ H\}}\ (r1 \bowtie r5 \bowtie r2)$$

Note that to construct a correct algebraic expression as a formal specification of a transaction, we do not use any semantic meaning of the involved attributes. Only condition 'C' for all σ_C statements needs to be directly *taken* from the narrative transaction specification.

CONCLUSION

We argue that there is an easy way to map the informal specification of transactions acting on a normalized relational database to a purely procedural specification, using relational algebra.

The method presented concentrates on the correspondence between the required attribute specifications of the process and the attribute specification of the relations in the database. The correspondence is exemplified by the hypergraph and the properties of the hypergraph (and its derivatives) provide the procedural specification.

The method should reduce substantially the number of errors in overall process specification. It has full potential for automation.

Further research is ongoing to:

(a) provide a CASE tool based on this method to support process design for relational systems

(b) utilise the hypergraph to explore complex data structures and simplify relational mappings

(c) consider the correctness and performance aspects of the process transaction to atomic transactions mapping

(d) explore further the language considerations required to express between-process flow systems - especially with respect to parallel processing.

REFERENCES

Boehm, B. 'Software Life-Cycle factors' in Vick, C.R. and Ramamoorthy, C.C. *Handbook of Software Engineering*. Van Nostrand Reinhold Company Inc., New York, 1984.

NCC. *SSADM Version 4: Four-volume Guide*. NCC-Blackwell, 1990.

Jackson, M. *System Development*. Prentice Hall, 1983.

Mayhew, P. 'Software Prototyping: Implications for the People Involved in Systems Development'. Proceedings CAiSE90, pp. 290-305. *Lecture Notes in Computer Science, No. 436*. Springer-Verlag, 1990.

Boehm, B. 'A Spiral Model of Software Development and Enhancement'. *ACM SIGSOFT Software Engineering Notes* 11 (4), pp. 22-42, 1986.

Chen, P. 'The entity-relationship model - towards a unified view of data'. *ACM TODS* 1(1), pp. 9-36, 1976.

Codd, E.F. 'Normalized Data Base Structure: A Brief Tutorial'. Proceedings *ACM-SIGFIDET Workshop on Data Descriptions, Access and Control,* San Diego, California, Nov. 11-12, 1971.

Solvberg, A. 'Behaviour-Specification in an Information Systems Analysis Context'. Proceedings *BNCOD* pp. 71-86. Cambridge University Press, 1986.

Jeffery, K.G. & Gill, E.M. 'The Design Philosophy of the G-EXEC System'. *Computers and Geosciences 2* pp. 215-229, 1976.

Date, C.J. *An Introduction to Database Systems,* Fifth Edition. Addison-Wesley, 1990.

Chao-Chih Yang, *Relational Databases.* Prentice-Hall, 1986.

Orlowska, M.E. 'On characterization of a normalized relational database'. (Submitted) 1990.