AN INTERPRETER FOR DECLARATIVE QUERY LANGUAGES

by

Wanyen Chang

Memorandum No. ERL-M498

9 January 1975

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

AN INTERPRETER FOR DECLARATIVE QUERY LANGUAGES

Wanyen Chang

Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California, Berkeley, California 94720

ABSTRACT

Users of a declarative query language can do retrieval and update without specifying the logical search path for their data, thereby achieving logical data independence, i.e. the data model of a data base can be modified while application programs and users in general are not affected by the change. This Report deals only with the case where the data model is a set of logically related relations. Logical data independence is achieved by letting users treat the data base as a single relation for retrieval and update. An Interpreter is designed to translate the user query into statement based on the relational model, by defining the appropriate search path. The ideas involved are illustrated using SQUARE as a sample query language. However, it is believed that the translation mechanism reported here is general enough to apply to any query language based on relational model that requires a user to specify the logical search path for the desired data.

Research sponsored by Naval Electronics Systems Command Contract N00039-75-C-0034

Section I Introduction

Data Independence & Relational Model

A common architecture for large shared data base systems is shown in Fig. 1. The advantage of this type of system architecture is that users are insulated from changes of the physical data organization, sometimes referred to as Physical Data Independence. In systems of this type, the logical data organization serves at least two functions:

(1) To help users understand the content of the data base,

(2) To be a common structure against which files may be defined by application programmers and queries may be simply expressed by casual users.

Relational Model is a data structure in which data files in a data base are viewed as relations (usually represented by a special type of Table — two dimensional array). It has been shown that as a logical data organization it is superior to other structures, especially with respect to the two abovementioned functions. For a detailed account of the advantages of relational model over other structures, readers are referred to Ref. [1].

Although the relational approach is capable of supporting physical data independence, changes in the model itself may affect the functioning of application programs and users in general. To illustrate this point, let us consider the following query directed to the sample data base, shown in Fig. 2(a):

Find the project titles of the projects managed by Frank Foo.

Expressed in Query Language SQUARE,[2] it reads:

$$\text{TITLE}^{R2}\text{PROJ\#}^{R1}\text{MGR}\ (\text{'Frank Foo'})$$

Expressed in Query Language ALPHA,[3] it reads:

```
RANGE   R1  X
        R2  Y
GET     W   Y.TITLE: ∃X((X.MGR=Frank Foo) &
                        (X.PROJ#=Y.PROJ#))
```

In general, a relation specifies a mapping between any two component attributes. In the sample data base, the mapping from MGR to PROJ#, specified by R1 is a simple data mapping, i.e., a function. However, the reverse mapping is complex, i.e., a one-many correspondence. We will write MGR $\xrightarrow{s}$ PROJ# and PROJ# $\xrightarrow{c}$ MGR to denote these facts. Now suppose the two data mappings become

$$\text{MGR} \xrightarrow{c} \text{PROJ\#} \quad , \quad \text{PROJ\#} \xrightarrow{s} \text{MGR}.$$

The data base will change to that as shown in Fig. 2(b). The changes in the data mappings may be induced by a policy change in the real world. In this case, a manager may now manage more than one project, but a project is managed by exactly one manager. The same query, again expressed in SQUARE and in ALPHA, now takes on a different look:

(SQUARE) $\text{TITLE}^{R2}\text{MGR}\ (\text{'Frank Foo'})$

(ALPHA) GET W Y.TITLE : Y.MGR = FRANK FOO.

Thus, application programs involving relations R1/R2 may have to be modified or rewritten, whenever the definitions of the two relations are modified. At this point, we might notice that the cost associated with the remodeling is not a logical consequence of the relational

model, rather it is due to the fact that the two query languages require their users to spell out the search paths for retrieving the desired data based on the relational model. As the model changes in time, the search paths may have to do so accordingly, thus breaking the so-called logical data independence. In this report, the user-defined search path is referred to as logical search path, since the system may take a different route to access the data.

Continuing the example, if the query languages allow the use of data base names in the statements, the above query could be expressed as

$$\text{TITLE}^{\text{DN}}{}_{\text{MGR}}(\text{'Frank Foo'})$$

in the modified version of SQUARE, and as

RANGE DN X

GET W X.TITLE : X.MGR=Frank Foo

in the modified version of ALPHA, where in both expressions DN stands for the data base name. It is easy to verify that the last two expressions are valid before and after the changes of the sample data base.

The Problem

In general, a query statement consists of two parts: the Qualification, and the Target list. In the above example, we see that the qualification part of the query in both languages may be considered as a mixture of the logical specification of the desired data and its

search path, the latter being model dependent. In the modified versions, however, the qualification part becomes a pure logical specification, therefore model independent. We also note that in the modified versions, users are, at least implicitly, viewing the data base as a single relation. The question then is that given the pure logical specification of some data, how does the system construct the appropriate search path for it? In other words, how is the system going to translate a query against a single relation into a query against a relational model (a set of relations)? That part of a relational data base system that is responsible for this task is referred to as the Interpreter. The report is concerned with the design and implementation of the Interpreter. To simplify the presentation, the interpretation mechanism is illustrated via a sample data base, and SQUARE is arbitrarily chosen as the target language. For convenience, the modified version of SQUARE will be referred to as DELTA. However, it will be evident that the ideas presented in this report are also applicable to other query languages based on Relational Model which require users to define logical search paths.

Section II The Interpreter

The plan for this section is as follows: First, the interpretation process of a sample query is shown. The sample data base then undergoes several changes, and it is shown how these changes can be absorbed through corresponding adjustments of the interpreter to make the input expressions remain correct. During the course of illustration, certain conditions on data bases and requirements on the system are explained and summarized into assumptions, so that the scope of the validity of the results presented in this report may be clearly understood.

The Sample Data Base

Case 1 of the sample data base is shown in Fig. 3(a). The key of each relation is underlined. The data mappings among the three attributes: DEPT, ITEM, and FLOOR, as defined by the data base, is explicitly shown in Fig. 3(b). The usual terminology for relational model is used here. An attribute which is a component of a key is referred to as prime attribute; otherwise it is non-prime. In relation EMP, NAME is a prime, DEPT, MGR, etc. are non-prime.

By the definition of a key, the value of a key uniquely determines, at any instant in time, the value of any attribute in the same relation. And we may state the fact by saying that each attribute in a relation is functionally dependent on the key. The value of a non-prime attribute may be functionally dependent on part of the key or other non-prime. In Fig. 4, the attribute ADDRESS in R1 is dependent on the attribute SUPPLIER, which is only part of the key, and attribute BUDGET in R2 is dependent on the attribute DEPT, which is a non-prime. In either case,

the attribute in question is said to be transitively dependent on the key. A relation in Third Normal Form, or TNF for short, is defined as a simple relation where every non-prime is non-transitively dependent on the key. Thus, the five relations in the sample data base are TNF's. For a full account of relational model, readers are referred to E. F. Codd's papers, [4], [5].

Two relations having common attributes may be combined by the operation JOIN, defined below. Let α, β, γ be attribute lists, such that $\alpha \cap \beta = \gamma$. Let R be a relation over α, s over β. The JOIN of R and S over γ is then defined by

$$R * S \equiv \{(a,\gamma,b) \mid (a,\gamma) \in R \text{ and } (r,b) \in S;$$
$$\text{or } a = \hat{}, (\gamma,b) \in S;$$
$$\text{or } b = \hat{}, (a,\gamma) \in R;\}$$

where $\hat{}$ means 'undefined.' An example of JOIN is shown in Fig. 5. Note that this definition is a generalization of the Natural Join, defined by E. F. Codd.[4] In general, JOIN is non-associative. For instance, three relations R1, R2 and R3 shown in Fig. 6 can be joined in two different orders, yielding different results.

Theoretically, a data base could be just any arbitrary set of TNF's. However, to enable a distinction between two data bases, it is reasonable to assume that the TNF's in a data base are logically related. We thus make the following assumption:

Assumption 1. The JOIN of all the TNF's in a data base defines a unique relation with distinct attribute names.

In fact, Assumption 1 asserts more than just the existence of logical

relationships among the set of TNF's in a data base. As JOIN is non-associative, the set of TNF's may be joined to yield several different realtions. By a unique relation, it is meant that a data base must contain information such that a specific relation may be chosen from the set of all possible relations that may be obtained by a JOIN operation. The reason for this will be clear later on.

To make sure that TNF's in a data base can only join on prime attributes and that the resultant relation does have distinct attribute names, we have

Assumption 2. An attribute can be a non-prime in at most one TNF in a data base.

We note that the sample data base does satisfy the above two assumptions.

The Intrepretation Mechanism

Now consider the query: Find those items sold on the second floor of the department store.

Expressed in SQUARE, it reads

$$ITEM^{SALES}{}_{DEPT}{}^{LOC}{}_{FLOOR}\ ('2')$$

Expressed in DELTA, it becomes

$$ITEM^{DN}{}_{FLOOR} = 2$$

where DN is the name of the sample data base (or the department store).

To translate into SQUARE a DELTA statement of the form ${}_{t}{}^{DN}{}_{s}$, where s is the source (or given) attribute and t is the terminal (or target) attribute, the Interpreter uses two data files — the Connection

Graph (C-graph) and the Node Index, and a tranlation algorithm — the T-algorithm.

(1) C-graph: The purpose of the graph is to show all the logical search paths that are valid in the system. Given a data base, a C-graph may be constructed with the following rules:

1. Each prime attribute is represented by a node, referred to as first-level nodes.
2. If a first-level node is not a simple key, it is marked as fictitious, otherwise, it is marked as real.
3. Keys of length $\ell \geq 2$ are represented by nodes of ℓ^{th}-level.
4. Two nodes K_1 and K_2, where $K_1 \cap K_2 = \phi$, are connected by an arc, if and only if, either $K_1 \rightarrow K_2$, or $K_2 \rightarrow K_1$, i.e., one is functionally dependent on the other.
5. Two nodes K_1 and K_2, where $K_1 \supseteq K_2$, are connected by an arc, if and only if, there exists no key K, such that $K \supseteq K_1 \supseteq K_2$.
6. Arcs in C-graph are associated with attributes common to the tables (TNF's) corresponding to the two terminal nodes.

The C-graph for the sample data base is shown in Fig. 7. Obviously, a real node in a C-graph represents a TNF in the corresponding data base. A fictitious node, on the other hand, serves two functions: (i) to ensure that the C-graph of a data base is always a connected graph under the construction rules; (ii) to act as a starting node in the construction of a logical search path, when the source attribute is a prime which appears in more than one TNF

Rules 4 and 5 are so designed that certain types of cycles in C-graph may be avoided. To fully appreciate the underlying idea, let us consider another data base shown in Fig. 8(a). The logical links

among the five TNF's can be represented by a graph shown in Fig. 8(b). Given certain information about a part, we may find the data of the related suppliers in two ways: (i) through the TNF (s#.p#), or (ii) through the TNF (s#.p#.w#). To ensure a consistent answer, it is necessary for the system to see that the projections of the two TNR's on attributes s# and p# be identical. The above consideration leads to the following two assumptions:

Assumption 3. Projections of the TNF's on the same set of prime attributes yield identical relations, i.e.

$$K^{T}_{i1} = K^{T}_{i2} = \cdots = K^{T}_{in},$$

where K are prime in every T_i.

Assumption 4. A compound key may be partially defined.

Based on these assumptions, the cycles shown in Fig. 8(b) may be eliminated by Rules 4 and 5. The resulting C-graph is shown in Fig. 8(c). Using the C-graph and the Node Index discussed below, proper logical search paths can be constructed for given DELTA statements.

(2) Node Index: The purpose of a Node Index is to map the given source and terminal attributes onto C-graph. The Node Index for the sample data base is shown in Fig. 9. Since a prime may appear in several TNF's, these TNF's are linked by a list. The head of the list is a first-level node. The rest of the nodes in the list are in order of their level number, the one with the highest level number being the next to the head. When an attribute is identified as a non-prime, the value in the third column of the Node Index indicates the location of the corresponding node in the C-graph. When it is a prime, the value is a pointer pointing to the head of a node list. The Node Index may

be a sorted array. Since, at any point in time, all the attribute names are known to the system, address calculation may also be used.

(3) T-algorithm: Given the two data structures — C-graph and Node Index, the Translation algorithm is quite obvious. The T-algorithm for the input form ${}_tDN_s$ is shown in Fig. 10. When s and t are both prime, the T-algorithm first check to see if they are part of a compound key. If they are, then both s and t are mapped onto the same node, otherwise a path connecting the two nodes must be constructed by using the Labelling Algorithm in Graph Theory (see Appendix A). Assuming no information regarding the access frequency of each TNF is available, the check is probably most efficiently done if the node list of each prime attribute is ordered by level number, as suggested in the construction of the Node Index.

Applying the T-algorithm to the sample query, we see that attribute FLOOR is mapped to node DEPT, ITEM to node ITEM, using the Node Index. The Labelling Algorithm then constructs the connecting path between the two nodes, i.e., N(DEPT) : L(DEPT) : N(D.I) : L(ITEM) : N(ITEM), where N stands for Node, L for Link. Converting node to TNF names, it becomes: LOC : L(DEPT) : SALES : L(DEPT) : CLASS. Written backwards as in SQUARE, and appending the source and the terminal attributes to both ends, we get

$$ {}_tCLASS_{ITEM}SALES_{DEPT}LOC_s $$

where s = FLOOR, t = ITEM. Since ${}_{ITEM}CLASS_{ITEM}$ is a redundant operation, the above expression can be simplified to

$$ ITEM\,SALES_{DEPT}LOC_{FLOOR} $$

which is exactly the same as the SQUARE statement shown earlier. The simplification rules are shown in Fig. 11. As the rules are quite obvious, no explanation is needed.

Adjustments

Case 2 of the sample data base is shown in Fig. 12(a) and 12(b). Specifically, the data mappings between DEPT and FLOOR now become complex, and the mapping from ITEM to FLOOR becomes simple. As a result, the TNF LOC may or may not exist depending on the decision of the particular system. The change of this type may be characterized by a non-prime moving from one TNF to another. To reflex the change, the Node Index is modified so that attribute FLOOR is now mapped to node ITEM, the rest being the same. Consequently, the T-algorithm now yields the expression

$$\text{ITEM}^{\text{CLASS}}\text{FLOOR}$$

which is the correct SQUARE statement in this case.

Case 3 of the sample data base is shown in Fig. 13(a) and 13(b). Now the data mappings among the three attributes are all complex. In this case, a non-prime (i.e. FLOOR) becomes a prime, and two new TNF's (i.e. LOC and PLACE) must be formed. The C-graph also changes and is shown in Fig. 14. Note that the C-graph now contains a cycle.

The three relations, SALES, LOC, and PLACE are said to form a Cyclic Chain. In general, a Cyclic Chain may be defined as a set of n relations where their keys can be ordered in such a way that

$$K_1 \cap K_2,\ K_2 \cap K_3,\ \cdots,\ K_{n-1} \cap K_n,\ K_n \cap K_1$$

are all non-empty. When a set of relations form a Cyclic Chain, there exists more than one way to join them, and the resultant relations are likely distinct. This, of course, is due to the fact that operation JOIN is generally non-associative. Thus, case 3 of the sample data base violates Assumption 1.

One consequence of a data base failing to define a unique relation is that the data base is at a state which might be termed 'information-incomplete.' For example, a valid question against the data base might be, "Find the location of item A sold by the department D." Here, we are tacitly assuming that item A might be purchased from several departments. The system cannot respond to such queries, because either no answer can be found in the data base, or more than one answer might be given, which is clearly undesirable.

A simple way to eliminate cyclic chain is shown in Fig. 15(a). Here, we have a new TNF called BASE, which explicitly specifies the relationships among ITEM, DEPT, and FLOOR. In general, whenever we see a cyclic chain in a data base, we may add a similar BASE relation to eliminate the ambiguity created by the chain. In the current example, the C-graph of the modified data base now becomes as that shown in Fig. 15(b). There is no cycle in the C-graph, that is, the data base defines a unique relation. The T-algorithm now yields

$$\text{ITEM}^{\text{BASE}}_{\text{FLOOR}}$$

Note that a user may express the query in SQUARE as $_{\text{ITEM}}\text{PLACE}_{\text{FLOOR}}$. However, from Assumption 3, it follows that these expressions are equivalent.

Section III Conclusion

Data Bases

In addition to the four assumptions made in Section 2, we have implicitly assumed that data bases compatible with the Interpretation Mechanism also satisfy the following two assumptions:

Assumption 5. No functional dependence exists between any two keys K_1 and K_2, where

$$\text{Length } (K_1) = \text{Length}(K_2) \geq 2.$$

Assumption 6. C-graph of a data base contains no cycles.

The last two assumptions are made to avoid complex situations which might occur in a C-graph. Assumption 5 is really not as severe a restriction as it might appear to be, since very few real data bases would violate the assumption. On the other hand, Assumption 6 does somewhat restrict the scope of application. At the time this report was written, a study of the possible types of cycles which may occur in C-graph was being conducted. The results obtained so far strongly suggest that Assumption 6 may be totally discarded.

Input Forms

The T-algorithm presented here only treats the input form ${}_t{}^{DN}{}_s$. For other forms, like

$${}_t{}^{DN}{}_{s_1=a,\ s_2=b,\ \cdots}$$

$${}_t{}^{DN}{}_{s_1}\ {}^{DN}{}_{s_2}\ \cdots$$

$${}_t{}^{DN}{}_{s=a},\ \text{[group qualification]}$$

some modifications on the algorithm are needed. However, the principles involved in each case are the same.

Updates

An update operation can be considered as a retrieval followed by a rewrite operation done in the work space. Thus, the same interpretation mechanism is applicable. However, from the system's point of view, updating a prime attribute is slightly more involved than updating a non-prime. As already noted, a prime attribute may appear in several TNF's in a data base. Thus, in updating a prime, TNF's other than the ones on the search path may also be accessed. The task can be accomplished by a separate module which is triggered by the Node Index as soon as the target attribute is identified as a prime.

Extent of Logical Data Independence

In the case where logical data organization is a Relational Model, five types of changes can be identified:

1. Adding an attribute from one TNF to another
2. Shifting an attribute from one TNF to another
3. Creating a new TNF
4. Deleting an attribute from the data base
5. Destroying an existing TNF.

It is obvious that the effect of changes of type 4 and 5 on application programs cannot (and should not) be avoided. Therefore, in the brief discussion of data independence below, we will ignore these two types of changes. A user may interact with a data base system in four ways, i.e. (i) to retrieve, (ii) to update, (iii) to insert a record and

(iv) to delete a record. From what has been presented here, we see that retrieve and update operations can be made independent of logical data organization. Since the other two types of operations are on a record-by-record basis, something would have to be done to improve their degree of data independence. We will not get into that here. As far as the Interpretation mechanism is concerned, it will be useful in systems where users doing retrieval and update greatly out-number those doing insertion and deletion, i.e., wherever the system architecture as shown in Fig. 16 is appropriate.

ACKNOWLEDGEMENT

The author is deeply indebted to Professor L. A. Zadeh for his support and guidance throughout this work. He would also like to thank Professor M. R. Stonebraker for helpful discussions.

REFERENCES

1. C. J. Date, E. F. Codd, "The Relational and Network Approach: Comparison of the Application Programming Interfaces," IBM RJ 1401 (#21706) June 6, 1974.
2. R. F. Boyce, et al., "Specifying Queries as Relational Expressions: SQUARE," IBM RJ 1291 (#20240) Oct. 16, 1973.
3. E. F. Codd, "A Data Base Sublanguage Founded on the Relational Calculus," IBM RJ 893 (#15716) July 26, 1971.
4. E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," CACM, Vol. 13, No. 6, June, 1970.
5. E. F. Codd, "Further Normalization of the Data Base Relational Model," IBM RJ 909 (#15857) Aug. 31, 1971.

APPENDIX A

Labelling Algorithm

This algorithm is also known as Fundamental Algorithm in Graph Theory. At each stage of the algorithm, each node exists in one of 3 states

unlabelled	(indicated by blank)
labelled but not scanned	(indicated by 0)
labelled and scanned	(indicated by X)

Algorithm*:

(1) Initially s is labelled (0) and each other node is unlabelled.

(2) If every node is either X or blank, stop; the X nodes are those reachable from s. Otherwise, choose a 0 node u. Change u to an X node. Change each blank node v such that u and v are connected by an arc to a 0 node. (This process is called scanning u.) Go to 2.

The Labelling Algorithm is used to find all the nodes reachable from a node s in a given graph. For our purpose, a second stop condition may be added to the algorithm, namely,

IF node t is labelled, THEN stop.

To indicate the path, it is only necessary to change the labelling method to the following:

Suppose L is the label of node u.

---

*This particular version is taken from R. M. Karp's class notes on Network Flow.

In scanning u, label a blank node V which is connected to u by L.V (a string of symbols).

When the algorithm stops, the label of node t now gives the path from s to t.

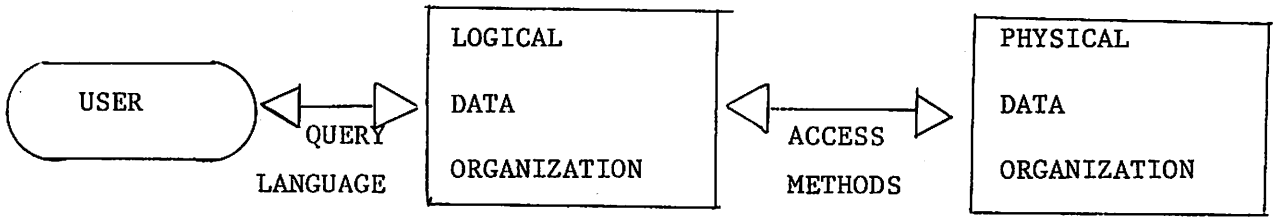


FIG. 1 DATA BASE SYSTEM ARCHITECTURE

R1 (MGR, AGE, SAL, PROJ#)

R2 (PROJ#, TITLE, TYPE)

(a)

R1 (MGR, AGE, SAL)

R2 (PROJ#, TITLE, TYPE, MGR)

(b)

FIG. 2 SAMPLE DATA BASE

EMP(NAME, DEPT, MGR, SAL, ...)

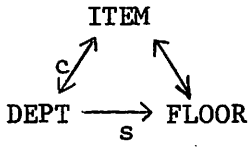
SALES(DEPT, ITEM, VOL 1, ...)

SUPPLY(SUPPLIER, ITEM, VOL2, ...)

LOC(DEPT, FLOOR)

CLASS(ITEM, TYPE

(a) SAMPLE DATA BASE OF A DEPARTMENT STORE



(b) DATA MAPPINGS

FIG. 3 CASE 1 OF THE SAMPLE DATA BASE

R1 (SUPPLIER, PART, ADDRESS)

R2 (EMP#, NAME, DEPT, BUDGET)

FIG. 4 TWO RELATIONS NOT IN THIRD NORMAL FORM

R

D_1	D_2
a	0
b	1

S

D_2	D_3
1	XX
1	YY
2	ZZ

R S

D_1	D_2	D_3
a	0	^
b	1	XX
b	1	YY
^	2	ZZ

FIG. 5 EXAMPLE OF JOIN

R_1

D	F
A	0
B	0
C	0
A	1
B	1
C	1
A	2

R_2

F	I
0	α
1	β
2	γ
0	γ
1	α

R_3

I	D
α	A
β	B
γ	A
β	A
β	C

$(R_1 \otimes R_2) \otimes R_3$

F	I	D
0	α	A
0	α	B
0	α	C
0	γ	A
0	γ	B
0	γ	C
1	α	A
1	α	B
1	α	C
1	β	A
1	β	B
1	β	C
2	γ	A

$R_1 \otimes (R_2 \otimes R_3)$

F	I	D
0	α	A
1	α	A
1	β	A
1	β	B
1	β	C
0	γ	A
2	γ	A
0	^	B
0	^	C

FIG. 6 NON-ASSOCIATIVITY OF JOIN

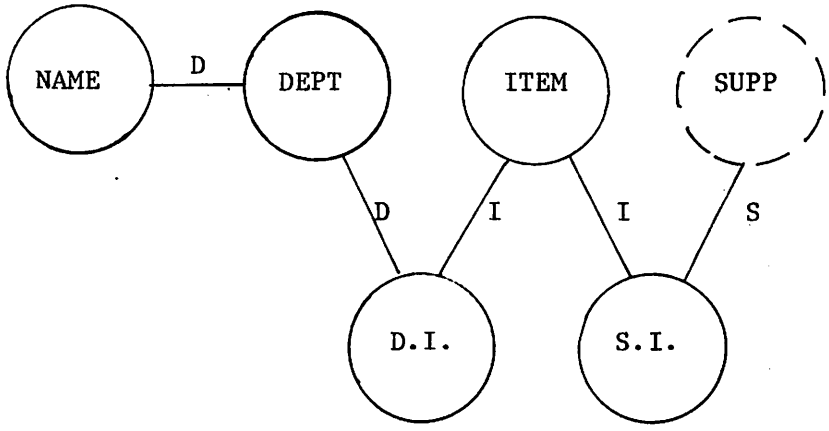


FIG. 7 C-GRAPH OF THE SAMPLE DATA BASE

SUPPLIER (S#, SN, ADDR)

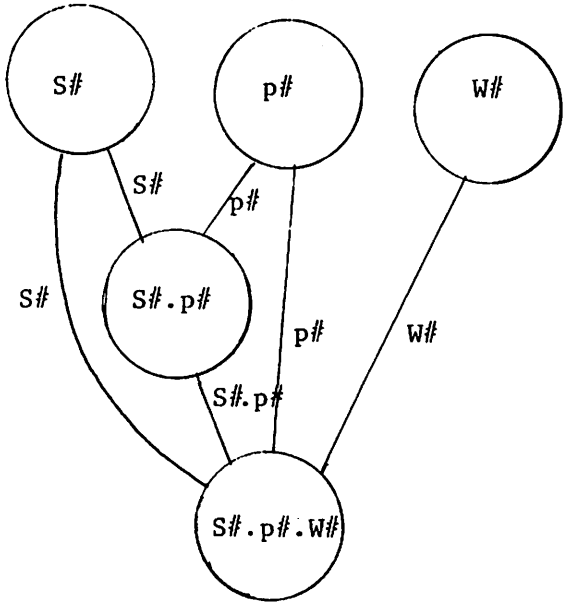
PART (p#, PN, QNT)

WAREHOUSE (W#, WN, SIZE, LOC)

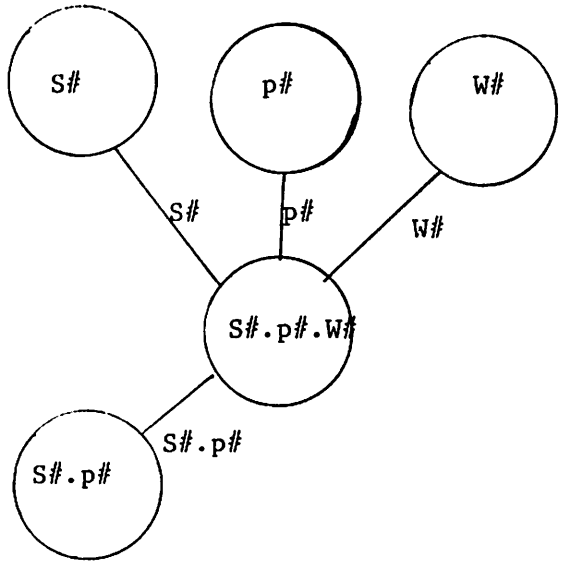
SP (S#, p#, VOL1, PRICE)

SPW (S#, p#, W#, VOL2)

(a)



(b)



(c)

FIG. 8 ANOTHER DATA BASE AND ITS C-GRAPH

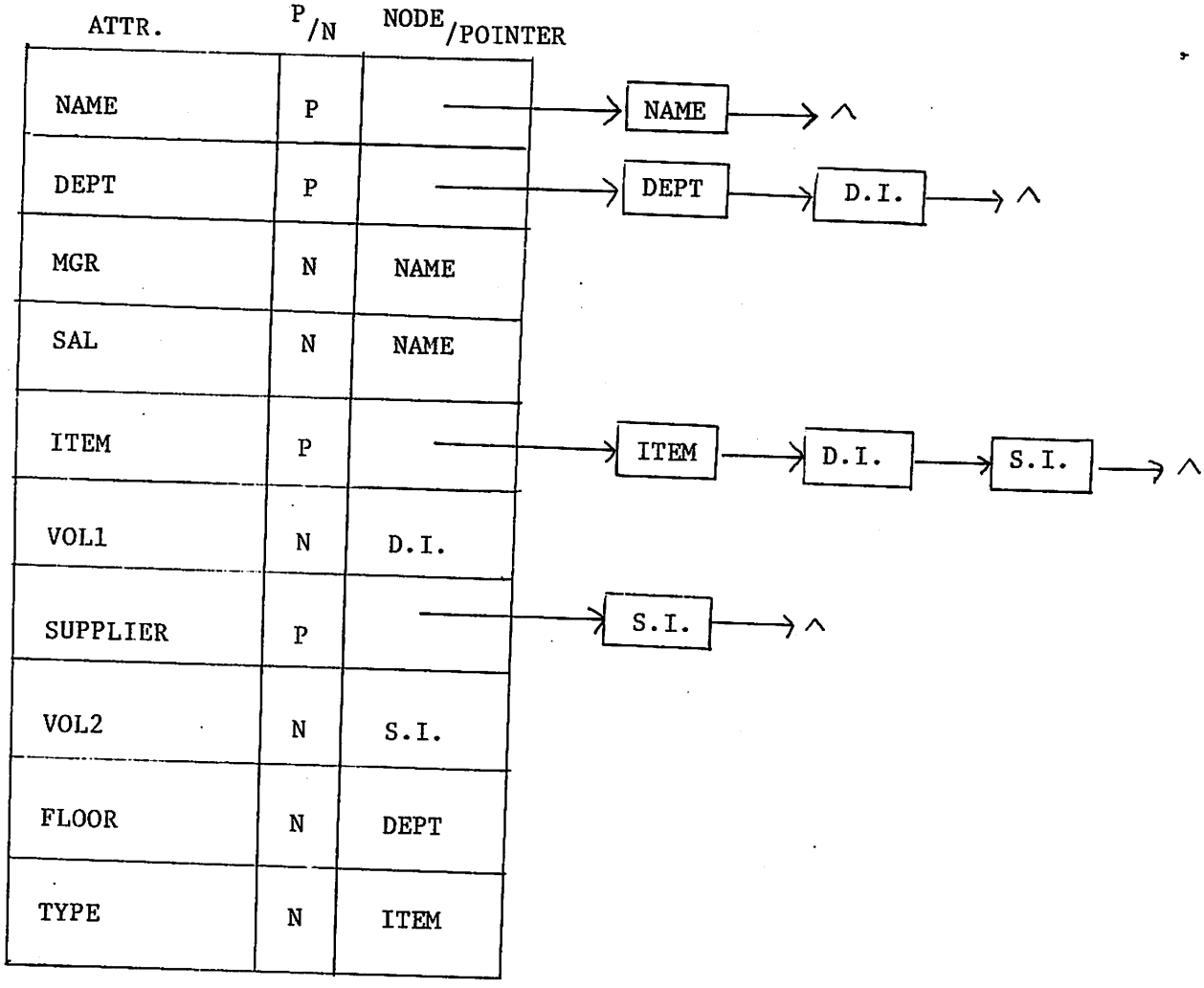


FIG. 9 NODE INDEX

INPUT FORM: $_{t}DN_{s}$

s = SOURCE NODE

t = TERMINAL NODE

1. IF s and t are both prime, then

   IF (s,t) is a higher-level node,

   THEN go to 4

   ELSE go to 2.

2. Map s and t to C-graph, by using the node index.

3. Find the connecting path between s and t, by applying the Labelling Algorithm to C-graph.

4. Convert nodes and arcs on the path to relation and attribute names.

5. Simplify the expression,and output instruction.

FIG. 10 T-ALGORITHM

1. $\overset{Z}{t}\overset{Y}{u}\overset{X}{v}\overset{}{s} \rightarrow \overset{Z}{t}\overset{X}{v}\overset{}{s}$

   if $u \supseteq v$.

2. $\overset{Z}{t}\overset{Y}{u}\overset{X}{v}\overset{}{s} \rightarrow \overset{Z}{t}\overset{Y}{u}\overset{}{s}$

   if $v = s$.

3. $\overset{Z}{t}\overset{Y}{u}\overset{X}{v}\overset{}{s} \rightarrow \overset{Y}{t}\overset{X}{v}\overset{}{s}$

   if $t = u$.

4. $\overset{Z}{t}\overset{Y}{u}\overset{X}{v}\overset{}{s} \rightarrow \overset{Z}{t}\overset{X}{u}\overset{}{s}$

   if $u \subseteq v$.

FIG. 11 SIMPLIFICATION RULES

EMP (NAME, DEPT, ...)

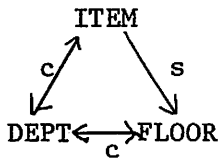
SALES (DEPT, ITEM, VOL1)

SUPPLY (SUPPLIER, ITEM, VOL2)

LOC (DEPT) ?

CLASS (ITEM, FLOOR, TYPE)

(a)



(b)

FIG. 12 CASE 2 OF THE SAMPLE DATA BASE

EMP (NAME, DEPT, ...)

SALES (DEPT, ITEM, VOL1)

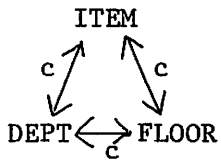
SUPPLY (SUPPLIER, ITEM, VOL2)

LOC (DEPT, FLOOR)

CLASS (ITEM, TYPE

PLACE (ITEM, FLOOR)

(a)



(b)

FIG. 13 CASE 3 OF THE SAMPLE DATA BASE

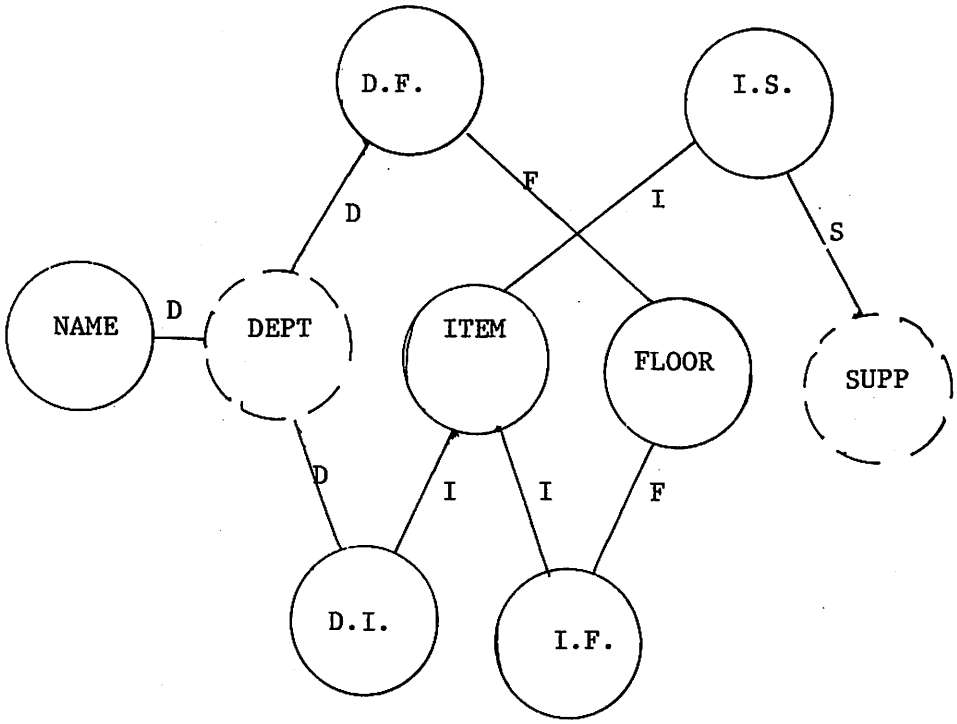


FIG. 14 C-GRAPH FOR CASE 3

EMP (NAME, DEPT, ...)

SALES (DEPT, ITEM, VOL1)

SUPPLY (SUPPLIER, ITEM, VOL2)

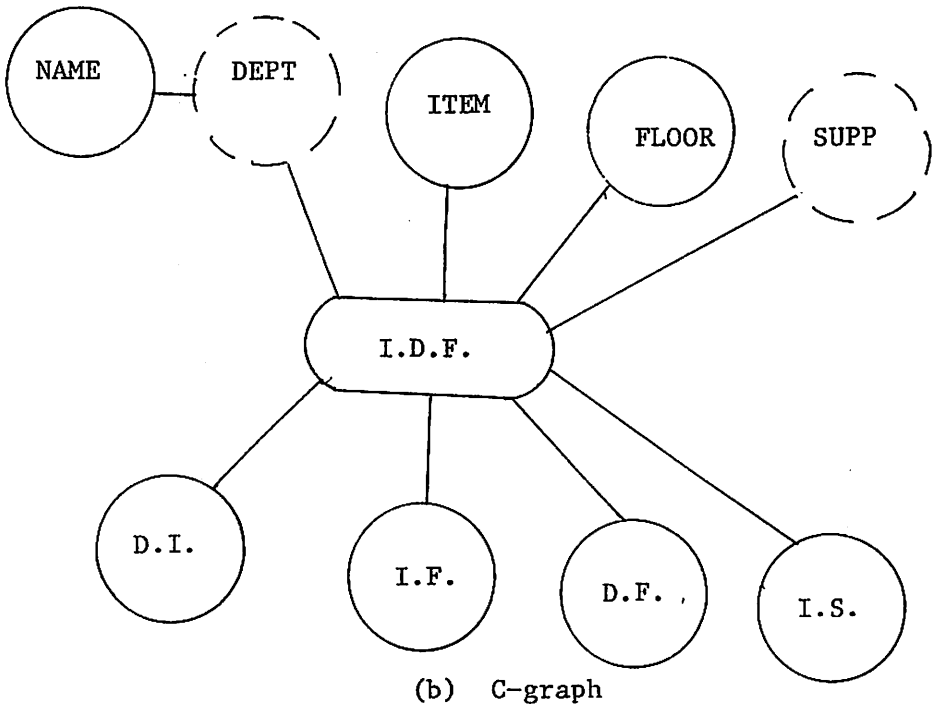
CLASS (ITEM, TYPE)

LOC (DEPT, FLOOR)

PLACE (ITEM, FLOOR)

BASE (ITEM, DEPT, FLOOR)

(a) Case 3 modified



(b) C-graph

FIG. 15 ELIMINATION OF CYCLIC CHAIN

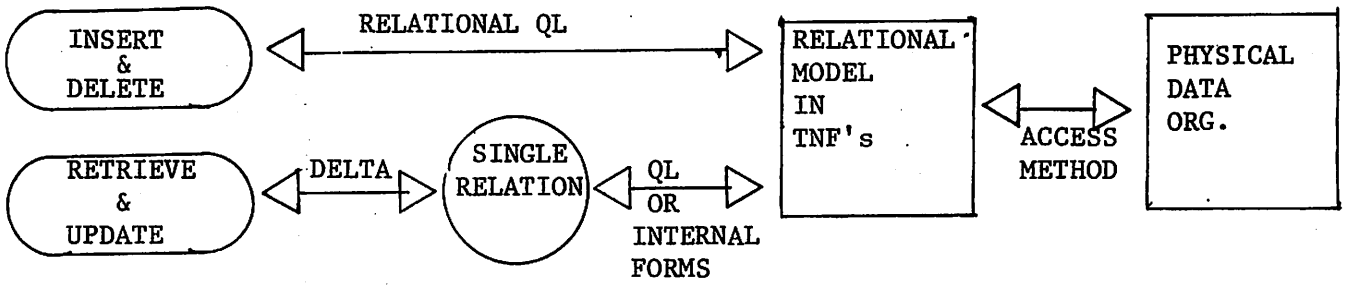


FIG. 16 PROPOSED SYSTEM ARCHITECTURE