DISCUSSION PAPER NO. 313

Nonprocedural Query Processing for Databases with Access Paths

by

Nancy Griffeth

December 1977

NORTHWESTERN UNIVERSITY
GRADUATE SCHOOL OF MANAGEMENT
EVANSTON, ILLINOIS 60201

Nonprocedural Query Processing for Databases with Access Paths

by N. Griffeth

Abstract. The use of "rules of inference" in database systems with access paths -- e.g., CODASYL and IMS databases -- is proposed to allow nonprocedural querying of the database systems. The kinds of access paths for which these rules of inference are required are isolated. It is shown that the rules of inference required for a CODASYL or an IMS database depend on the configurations of the edges in a diagram of the database.

Nonprocedural Query Processing for Databases with Access Paths

by N. Griffeth

I. Introduction.

The definition of a database must leave nothing to the imagination of the user. If it does, each user is free to supply his own interpretation to those parts of the database which have not been unambiguously specified. The result will be, at best, that the user is misled as to what the responses to his queries mean. At worst, the integrity of the database will be violated by inconsistent updating.

In this paper, it is shown that existing database systems require the user to place his own interpretation on those facts which are not explicitly stored in the database, but which may be deduced from facts which are explicitly stored. The facts which are explicitly stored are unambiguously defined due to definitions of data-items, sets, segments, etc. But there is no provision for specifying the rules by which facts may be deduced. This problem impedes the development of truly nonprocedural query languages by requiring the user to supply the reasoning by which the facts are deduced. The reasoning may be supplied in the form of a procedure which navigates the access paths in the database or in a simpler form, such as an expression in the relational calculus.

For the same reason, the construction of logical views from a physical database also depends on the user to supply a technique for forming the logical view from the physical database.

This problem can be rectified by provision of "inferential rules" in the database definition. The rules are used, in the place of user-provided techniques, to infer facts not explicitly stored in the database from those which are explicitly stored. In fact, an algorithm for inferring facts from a relational database system, using such inference rules, has been developed by Minker (8). The rules are stored as data in the database, so that potential ambiguity in the batabase is avoided.

Relational database systems contain no access paths. The presence of access paths in a database complicates the analysis of the inference procedures, because access paths may be used in two ways.

One of these ways uses access paths to reconstruct logical records which have been partitioned into several physical records for efficient storage and/or retrieval. The other way uses access paths to perform inferences. These two ways of using access paths will be treated separately. First, a primitive access path will be defined as a means of recovering explicitly stored facts. Procedures for determining which access paths are primitive in CODASYL and IMS databases will be developed. The remaining access paths, those used to perform inferences, must have inference rules

attached to them.

A general model for databases with access paths is also introduced, to handle a more general class of database system.

Minker's algorithm for relational database systems can thus be extended to database systems with access paths by adding an algorithm for recovering the explicit facts from such databases. Such an algorithm is described in section V. Minker's algorithm can then be used to perform inferences on the facts recovered by this algorithm.

The specification of inference rules for the appropriate access paths in a database system will then allow unambiguous traversal of the database for queries specifying data-items in widely separated parts of a database. Similarly, it will allow unambiguous construction of logical views from physical databases by treating references to the logical views as queries and processing them as above (see Stonebraker (9)).

II. Background.

The analysis of the roles of access paths in this paper depends on an analysis of the underlying relationships. The relationships used here are the same as those defined by Codd in the relational database model (4):

A RELATIONSHIP is a time-varying relation over a set of domains.

The use of inference rules -- i.e., rules for deducing a new relationship from a collection of existing relationships -- was proposed by Minker in (8). In a question-answering system, the inference rules can be used to deduce the relationships which are the answers to the queries.

The use of inference rules to respond to queries on CODASYL and IMS databases (3,7) is analysed in this paper. There are languages for traversing the access paths in these databases; for example, CODASYL's DML and IBM's DL/1. These languages, however, may require that an intricate procedure be written to respond even to a single query (2).

Many nonprocedural languages have been developed for responding to queries on relational databases. In particular, there are the relational algebra (5); Query-by-Example (10); and SEQUEL (1). These languages are all nonprocedural, i.e., no programs need be written to respond to queries; however, they do require, that the user supply any inferential reasoning required in finding the response to a query. In general, this takes the form of specifying the relations to be examined and the operations to be performed on the relations.

Dolk and Loomis (6) have also proposed an algorithm for nonprocedural access to a CODASYL database. The responses supplied by this algorithm to queries may be misleading if the user interprets

any part of the database differently from the designer, because the rules for making inferences are not specified either in the database or by the user.

III. Performing Inferences in a Relational Database.

In a relational database, all queries must be expressed, and the responses to the queries found, by using domain-names and relation-names only. No access paths are available for traversal of the database. If any inferences are to be performed in responding to a query, the rules for performing the inferences must be supplied by the user (ordinarily); or, in Minker's algorithm, the rules of inference are part of the database definition.

Minker uses the example shown in Figure 1 to illustrate his algorithm for performing inferences in a relational database. The notation used here differs somewhat from Minker's. A relationship "Rname" over domains "Dom1","Dom2",...,"Domn" is written;

Rname(Dom1,Dom2,...,Domn)

If the n-tuple having value x1 on domain Dom1, x2 on domain Dom2, etc., belongs to the relationship Rname, then we write:

(x1,x2,...,xn) e Rname

Relationships:

Father (F-No,C-No)
Mother (M-No,C-No)
Husband(H-No,W-No)
Wife (W-No,H-No)
Sibling(1-No,2-No)

Domain Descriptions:

F-No = Father Number
M-No = Mother Number
C-No = Child
H-No = Husband Number
W-No = Wife Number
1-No = First Sibling Number
2-No = Second Sibling Number

Stored Database:

Father

F-No	C-No
1	2
1	3
3	5
7	8
7	10

Mother

M-No	C-No
2	11
4	9

Husband

H-No	W-No
3	4
8	9

Wife

W-No	H-No
2	6
11	10

Sibling

1-No	2-No

Inference Rules:

(1) If (w,h) e Wife and (m,c) e Mother then (h,c) e Father
(2) If (m,c1) e Mother and (m,c2) e Mother then (c1,c2) e Sibling
(3) If (f,c1) e Father and (f,c2) e Father then (c1,c2) e Sibling
(4) If (h,w) e Husband and (m,c) e Mother then (h,c) e Father

---

Figure 1. Example of a stored relational database with rules of inference.

It is not necessary to store every father-child or sibling pair in the database since these pairs may be deducible from other pairs which are stored. In the stored database in figure 1, no pairs which may be inferred from other stored pairs are stored; thus the database avoids duplications the storage of any fact.

Consider the following query:

"Who is the father of 11?"

The answer to this query may be found by noticing that 2 is the mother of 11; 6 is the husband of 2; and by inference rule (1), 6 is then the father of 11.

The following paragraph gives an informal description of Minker's algorithm. Minker defines two kinds of data in the database: the extensional data, which consists of the stored database, and the intensional data, which consists of the inference rules. We assume that queries are boolean conjunctions of relations.

The response to a query is found by first searcing the extensional data to find the answer. Then, if the answer was not found in the extensional data, the intensional data is searched for a rule which will transform the query to a new query. The same procedure is repeated for a queries thus generated, until an answer is found, or no more transformations are possible.

The importance of the rules of inference in determining the answer to a query can be seen by considering the following change to the above rules of inference:

(1) If (h,w) e Husband and (f,c) e Father then (w,c) e Mother
(2) If (f,c1) e Father and (f,c2) e Father then (c1,c2) e Sibling
(3) If (,c1) e Mother and (m,c2) e Mother then (c1,c2) e Sibling
(4) If (w,h) e Wife and (f,c) e Father then (w,c) e Mother

Then the response to the query:

"Who is the father of 11?"

is "The father of 11 is unknown." Similarly, the query

"Who is the mother of 5?"

elicits the response "The mother of 5 is unknown." when the first set of rules is used, but the response "The mother of 5 is 4." when the second set of rules is used.

The difference in responses is the result of a change in orientation of the users. One user seeks to determine the father of the children, given the mother; the other, to determine the mother, given the father. Such a change in orientation between two users would not be unlikely in a real database situation. Thus, it is essential that the rules of inference that were intended in the original database definition be made explicit.

IV. Performing Inferences in Database Models with Access Paths.

In this section, the use of inference rules in models with access paths is analysed. First, the role of an access path in a database is examined and the term access path is formally defined. Next, two types of access paths are compared to illustrate when inference rules can be used to remove ambiguity and when they cannot. Then, a characterization is developed of the situations in which inference rules can be used. Finally, this characterization is applied to the CODASYL and IMS database models to determine when inference rules are required to eliminate ambiguity in a database in each of the models.

A. The role of an access path in a database

An access path is a means for determining a "next" record (or group of "next" records) from a given record. Access paths are often implemented by pointers, but they may also be implemented using directories, record contents, physical positioning of records, etc. To maintain generality, the term access path will be defined as a means of relating one record to a set of records, as follows:

An ACCESS PATH is a class of mappings f from a record type to the powerset (i.e., the set of all subsets) of a record type (or types).

For example, each set type in a CODASYL database defines an access path. The class of mappings defined by a set type having owner record type A and member record type B is

$$\{\ f: \text{domain}(B) \longrightarrow \text{SING}(\text{domain}(A))\ \}$$

where SING(X) denotes the set of singleton sets over a set X:

$$\text{SING}(X) = \{\ \{x\}:\ x \in X\ \}$$

There are several possible uses of access paths. An access path may be used to indicate a relationship between records. Or it may be used to perform inferences. In this section, we are concerned with differentiating these two uses of an access path.

B. Construction of relationships using access paths

A database can be defined by specifying a set of record types and access paths. An instance of the database will then be a collection of files (each file containing all the active records of a single record type) and mappings (one mapping for each access path in the database).

Any record in first normal form -- i.e., with all repeating groups "flattened" -- may be regarded as a member of a relationship. Also, any concatenation of first normal form records or any equijoin of first normal form records on common data-items may be regarded as a member of a relationship.

Thus, if we have a stored database, then for each access path in the database definition, a relationship may be constructed from the database. Let f denote the mapping belonging to the access path which is currently stored in the database. Let A denote the file which is the domain of f. Let B1,..., Bn be the files involved in the range of f. The mapping f might be defined as follows:

$$f : A \longrightarrow POW(B1 \times \ldots \times Bn)$$

where POW(X) denotes the powerset of a set X, i.e., the set of all subsets of X. Then the following construction gives the relationship corresponding to f:

1. For each record in A, convert A to first normal form. Then use f to find the set of records (or the set of tuples of records) related to the record in A.

2. For each record or tuple of records related to A, convert it to first normal form and concatenate it with the record in A.

C. Characterization of primitive vs. inferred relationships

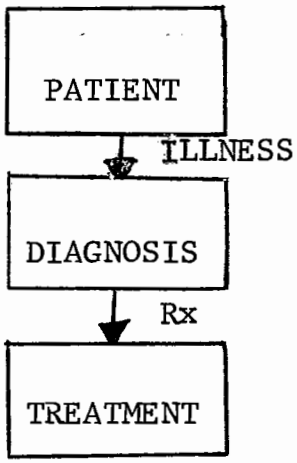


Figure 2. A Patient Diagnosis and Treatment database.

A relationship constructed from a database will be called a "primitive relationship" if it cannot be constructed by performing a nontrivial inference on the other relationships in the database. A relationship will be called an "inferred relationship" if it can be constructed by performing a nontrivial inference. An inference will be considered trivial if any of the relationships involved contains the inferred relationship or if all of the relationships involved uniquely determin the value of the inferred relationship.

Primitive relationships must be unambiguously defined in the definition of the stored database. Inferred relationships can be defined by inference rules.

EXAMPLES

Consider the access paths defined by the diagram in Figure 2, assuming that each arrow represents a 1:N relationship. The "Illness" and "Rx" relationships are primitive relationships, because they cannot be inferred from any other relationships. A "Patient-Treatment" relationship can be inferred from the "Illness" and "Rx" relationships, using the following operation:

Patient-Treatment
= Pr(Patient, Treatment) {Illness [Diagnosis=Diagnosis] Rx}

where Pr(x1,...,xn) denotes projection onto x1, ... ,xn and [x=y] denotes the equijoin on x and y.

On the other hand, if the relationships "Illness" and "Rx" are M:N instead of 1:N, then the relationship "Patient-Treatment" cannot be inferred from them. This follows because the value of the "Patient-Treatment" relationship may be an arbitrary subset of

domain(Patient) x domain(Diagnosis) x domain(Treatment).

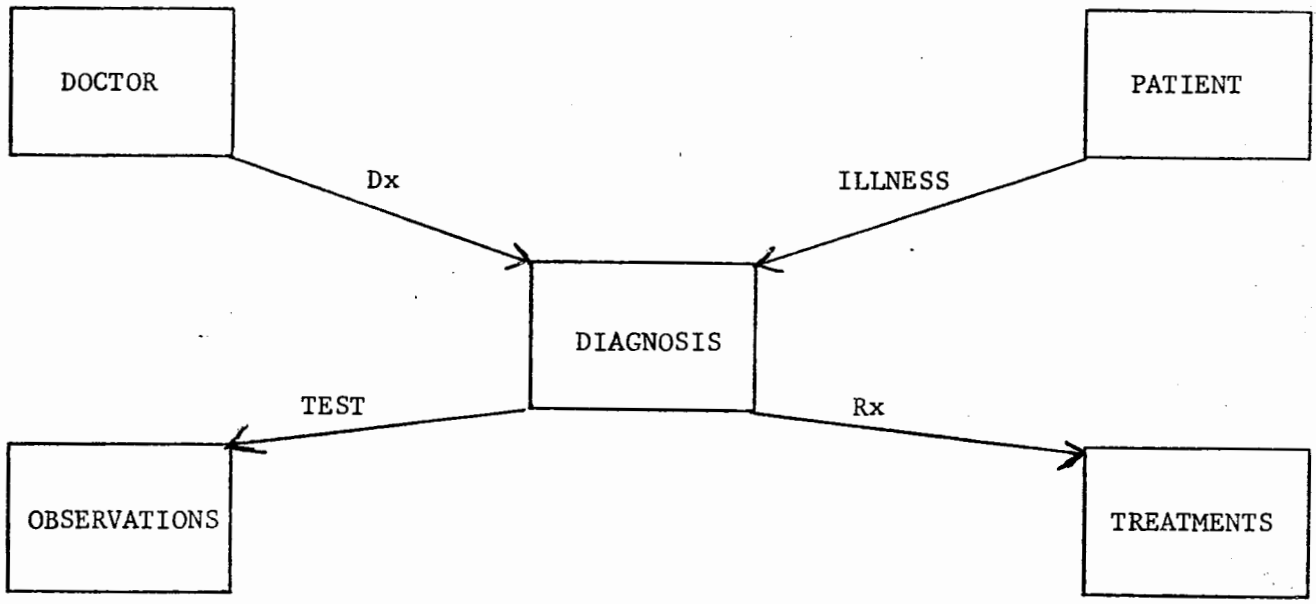
where domain(R) denotes the set of values that can be taken on by a member of relationship R (or a record of record type R). The values of the subrelationships "Illness" and "Rx" are just projections of the value of "Patient-Treatment" onto

domain(Patient) x domain(Diagnosis)
and
domain(Diagnosis) x domain(Treatment)

But arbitrary subsets of a space cannot be recovered from projections of the space onto proper subspaces.

D. Application to the CODASYL Database Model

In this section, an example of a CODASYL database is given to illustrate how ambiguity can be eliminated by the use of inference rules. Next, the primitive relationships -- i.e., those that must be defined by the database definition rather than by inference rules-- are isolated. All other relationships can be defined by inference rules.



Record types:

Doctor - gives the doctor identifying information (number, name, specialty, etc.)
Patient - gives the patient identifying information (number, name, room, etc.)
Diagnosis - describes what the doctor identified as being wrong with the patient and when (date, disease description, etc.)
Treatment - describes what was prescribed to cure whatever was wrong with the patient (date, treatment type; treatment number, etc.)
Observation - describes symptoms observed in the patient (date, symptom, severity, etc.)

Set types:

Dx - relates the doctor to the diagnoses he performed (each diagnosis is performed by a single doctor)
Illness - relates the patient to the diagnoses of his problems (each diagnosis relates to a single patient)
Rx - relates each treatment of a problem to the diagnosis of the problem, describing when the treatment was prescribed, how much etc.
Test - relates observations of a problem to the diagnosis of the problem

---

Figure 3. A CODASYL patient treatment database.

The potential ambiguity in the database defined in figure 3. arises in the inference of relationships between doctors and treatments; between patients and treatments; between doctors and observations; and between patients and observations. One set of inference rules for this database might be the following:

(1) If (doctor,diagnosis) e Dx and (diagnosis,treatment) e Rx then (doctor,treatment) e Doctor's-prescription
(2) If (patient,diagnosis) e Illness and (diagnosis,treatment) e Rx then (patient,treatment) e Patient's-treatment
(3) If (doctor,diagnosis) e Dx and (diagnosis,observation) e Test then (doctor,observation) e Diagnostic-procedure
(4) If (patient,diagnosis) e Illness and (diagnosis,observation) e Test then (patient,observation) e Patient's-symptoms

This set of inference rules says that doctors treat the patients they diagnose and patients are treated for the illnesses diagnosed in them. The observations correspond to symptoms that the patients exhibit, and are used by the doctor in reaching the diagnosis.

A second, equally plausible set of inference rules would be the following:

(1) If (doctor,diagnosis) e Dx and (diagnosis,treatment) e Rx
then (doctor,treatment) e Doctor's-prescription
(2) If (patient,diagnosis) e Illness and (diagnosis,treatment) e Rx
then (patient, treatment) e Patient's-treatment
(3) If (doctor,diagnosis) e Dx and (diagnosis,observation) e Test
then (doctor,observation) e Doctor's-progress-notes
(4) If (patient,diagnosis) e Illness and (diagnosis,observation) e Test
then (patient,observation) e Patient's-progress

In this case, the observations are used as a record of the Patient's progress and his response to treatment, rather than as the explanation of the diagnosis. Other interpretations are possible. The patient may not necessarily be treated by the doctor who diagnosed him. Similarly, the observations may be made by someone other than the diagnosing doctor.

Any of these interpretations would be valid in certain circumstances. The danger that one person may assume that one interpretation holds must be avoided in the database definition. The inclusion of the appropriate definitions for primitive relationships and inference rules for inferred relationships will accomplish this.

Next, we must consider how to apply the definitions of primitive and inferred relationships to CODASYL databases, in order to decide what kinds of definitions and what inference rules must be supplied for the database. There are three cases to consider:

(1) A record type R is a member of set type A and owner of set type B. Is the relationship between the owner Q of set type A and the member S of set type B primitive or inferred?

(2) A record type R is the owner of two set types, A and B. Is the relationship between the member Q of set type A and the member S of set type B primitive or inferred?

(3) A record type R is the member of two set types, A and B. Is the relationship between the owner Q of set type A and the owner S of set type B primitive or inferred?

In the first case, the application of the equijoin operation allows us to infer the relationship between the owner record type Q of a set type A and the member record type S of a set type B owned by the member record type R of set type A. In the example of figure 3:

Patient-Diagnosis-Treatment = Illness [Diagnosis=Diagnosis] Rx

In the second case, if two set types are owned by the same record types, then the equijoin allows us to infer the relationship between them:

Observation-Diagnosis-Treatment = Test [Diagnosis=Diagnosis] Rx

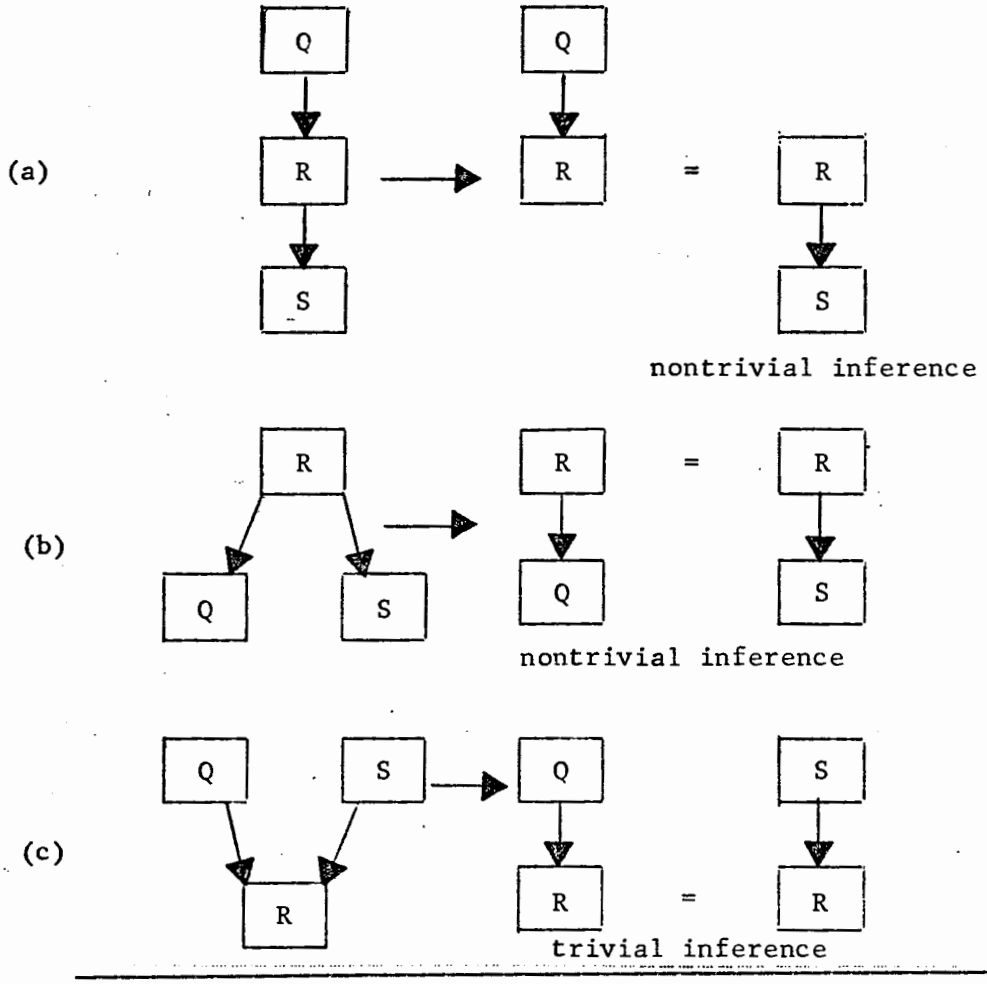


Figure 4. (a) An inferred relationship.
(b) An inferred relationship.
(c) A primitive relationship.

In the third case, we are considering two set types having a common member; e.g., "Dx" and "Illness" in the above example. Note that defining a common member in two set types is the standard technique, in the CODASYL database model, for defining a many-to-many relationship. This many-to-many relationship should behave like an arbitrary many-to-many relationship, i.e., as mentioned at the end of section IV.C. and discussed in more detail in section IV.E. below, it should not be possible to infer it from subrelationships.

In fact, the relationship can be constructed using the equijoin on the common member record type of the two set types. But in this one case, all of the relationships involved in constructing the relationship "Doctor-Patient-Diagnosis" uniquely determine the constructed relationship. Thus the inference is trivial (see section

IV.C.), so that "Doctor-Patient-Diagnosis" is a primitive relationship.

Thus the primitive relationships in a CODASYL database may be described as those which are constructed from groups of "confluent sets", i.e., sets having a common member record type. Figure 4 summarizes the results developed here.

E. Application to the IMS Database Model

In this section, an example of an IMS database is developed to illustrate how ambiguity can be eliminated from a hierarchical model by the use of inference rules. Then, the primitive relationships of the IMS model are distinguished from the inferred relationships, as was just done for the CODASYL model.

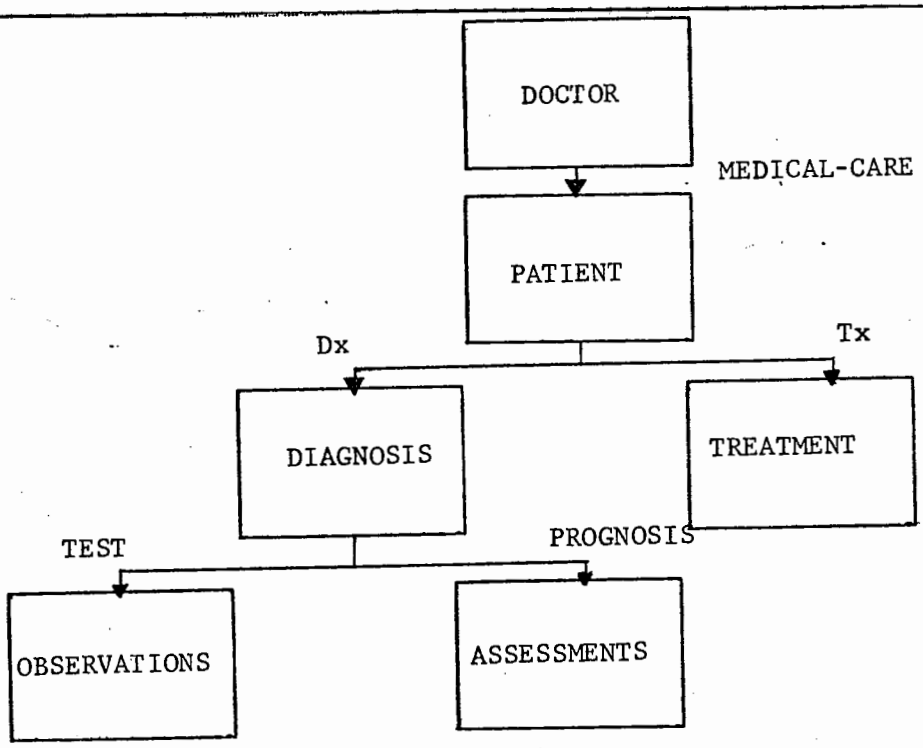
Consider the following sets of inference rules for the database in figure 5:

(1) If (patient,diagnosis) e Dx and (patient,treatment) e Rx then (diagnosis,treatment) e Contra-indication-check
(2) If (diagnosis,observation) e Test and (diagnosis,assessment) e Prognosis then (observation,assessment) e Condition

and

(1) If (doctor,patient) e Medical-care and (patient,diagnosis) e Dx then (doctor,Dx) e Doctor's-Dx
(2) If (doctor,patient) e Medical-care and (patient,treatment) e Rx then (doctor,treatment) e Doctor's-Rx
(3) If (doctor,diagnosis) e Doctor's-Dx and (doctor,treatment) e Doctor's-Rx then (diagnosis,treatment) e Check-Contra-indications
(4) If (diagnosis,observation) e Test and (diagnosis,assessment) e Prognosis then (observation,assessment) e Condition

The first set of rules implies that the database users are looking at all of the diagnoses and treatments of a given patient to determine if any of the treatments are contra-indicated by any of the diagnoses. The second set of rules implies that the users are also looking for contra-indications, but this time they are checking to see whether a doctor prescribed any treatments contra-indicated by any of his own diagnoses.



Segment types:

Doctor - gives the identifying information for the doctor (number, name, specialty, etc.)
Patient - gives the identifying information for the patient (number, name, etc.)
Diagnosis - describes the reason given by the doctor for the patient's problem
Treatment - describes the procedures prescribed by the doctor to cure/improve the patient's problem
Observations - describes the symptoms observed in the patient
Assessments - describes the doctor's opinion of the patient's condition

Figure 5. An IMS patient treatment database.

A consideration of how primitive relationships may be distinguished from inferred relationships in an IMS database concludes this section. In an IMS database, the parent-child relationships are M:N, in the sense that a single patient in the database of figure 5 could be related by the "Medical-care" relationship to several doctors. Each physical patient record is, of course, related to only one physical doctor record (its parent), but in the relationship recovered by the procedure of section IV.B., each patient may be related to several doctors. The latter relationship represents the logical relationship, which is the relationship of interest here.

Thus, each path in an IMS database defines a relationship which may be an arbitrary subset of the cartesian product of the domains of the record types along the path. Thus the path (Doctor,Patient), (Patient,Diagnosis),(Diagnosis,Observation) defines a relationship whose value may be any subset of the following set:

domain(Doctor) x domain(Patient) x domain(Diagnosis) x domain (Observation)

To demonstrate that not all values of such a relationship can be recovered from a fixed set of subrelationships, consider the projection of n-space over the real numbers onto subspaces in the following example:

Let A = { (1,...,1), (2,...,2) , ... , (n,...,n)}
and
let B = { (1,2,...,2) , (2,1,...,1) , (3,...,3) , ... , (n,...,n) }

Note that these two sets have identical projections onto the first coordinate and onto the second through nth coordinates. Thus they cannot be recovered from these projections. Given any pair of projections, a similar example of a pair of subsets of n-space whose projections are identical on the subspaces could be constructed. This result can then be extended inductively to n subspaces.

This shows that each path of an IMS database corresponds to a primitive relationship. Thus the relationship defined by a path in an IMS database should be explicitly defined by the database definition.

The only other case that needs to be considered is the case where a parent record type has multiple child record types. In this case, the relationship over all of the record types can be constructed using the equijoin over each of the parent-child relationships. From figure 5:

Patient-Diagnosis-Treatment = Dx [Patient=Patient] Rx

Thus this kind of relationship is inferred.

V. A General Model for Database Definitions.

In this section, a general model is developed for defining the "primitive access paths" in a database. A primitive access path corresponds to a primitive relation: primitive relations are constructed by following primitive access paths.

Definitions

The following definitions will be required for the general model:

Data-item - the smallest unit of data that can be referenced by a database system; a set of allowable values is associated with a data-item.

Data-item value - an occurrence of a data-item, with a particular value assigned to it
Data-aggregate - a collection of data-items and data-aggregates
Data-aggregate value - a collection of data-item values and data-aggregate values
Record type - a named collection of data-items and data-aggregates
Record - an instance of a record type; i.e., a collection of data-item values and data-aggregate values.
File - a collection of records of a single record type
Access path - a class of mappings f from one record type to the powerset (i.e., set of all subsets) of another record type (or types)
Traversal rule - a restriction on the class of all access paths in a database model
Database definition - a collection of record types and access paths
Database - a collection of records and mappings

In the following discussion, a traversal rule will be defined to limit the access paths in the CODASYL and IMS models to those configurations which define primitive relationships.

A. Application of the general model to CODASYL and IMS databases

Some access paths in the IMS database defined in figure 5 are:

{ f:Doctor ⟶ POW(Patient) }
{ f:Treatment ⟶ POW(Patient x Doctor) }
{ f:Diagnosis ⟶ POW(Observations) x POW(Assessments) }

where POW(x) denotes the powerset of x.

Some access paths in the CODASYL database defined in figure 3 are:

{ f:Diagnosis ⟶ SING(Doctor) }
{ f:Treatment ⟶ SING(Diagnosis x Doctor x Patient) }

where SING(x) denotes the set of all singleton sets over a set x, i.e., the set of all sets having only one member.

The primitive access paths in the IMS database defined in figure 3 are:

{ f:Observations ⟶ POW(Diagnosis x Patient x Doctor) }
{ f:Assessments ⟶ POW(Diagnosis x Patient x Doctor) }
{ f:Treatment ⟶ POW(Patient x Diagnosis) }

The primitive access paths in the CODASYL database defined in figure 3 are:

{ f:Observations ⟶ SING(Diagnosis) }
{ f:Treatments ⟶ SING(Diagnosis) }
{ f:Diagnosis ⟶ SING(Doctor x Patient) }

In general, the primitive access paths in a CODASYL database correspond to the confluent sets in the database. Thus each primitive access path has the form:

$$f : R \longrightarrow SING\ (S1 \times \ldots \times Sn)$$

for record types R, S1, ... , and Sn, with Si the owner of a set type and R the member. The primitive access paths in an IMS database correspond to the paths. Thus each primitive access path has the form:

$$f : Sn \longrightarrow POW\ (S1 \times \ldots \times Sn\text{-}1)$$

for segment types S1, ... , Sn-1, and Sn, where each Si is the parent segment type of Si+1.

B. Traversal algorithm

Minker's algorithm provides a technique for finding a member of an inferred relationship, given the primitive relationships in a database. Application of this algorithm to databases with access paths will require an additional algorithm to find those members of a primitive relationship which satisfy a given query. Referring to the discussion in section IV.B., the following algorithm will serve to search the extensional data for the members of a primitive relationship satisfying the query. It is assumed that each primitive relationship in the database is associated with an access path by the database definition. The notation a b will denote the concatenation of tuples (first-normal-form records) a and b.

T1: [Locate the access path] Search the database definition for the access path corresponding to the primitive relationship. Let

$$f : F \longrightarrow POW(G1 \times \ldots \times Gn)$$

be the mapping from this access path which is currently stored in the database. G., G2, ..., Gn, and F and files.

T2: [Search the file F for records satisfying the query] For each record r belonging to F, convert r to first normal form. The result will be a set {r1,...,rn}. For each ri which satisfies those parts of the query relevant to F, perform step T3.

T3. [Search the image f(ri) for records satisfying the query] For each concatenation s1 ... sn of records in f(ri), convert s1 ... sn to first normal form. The result will be a set {s11 ... sin, ..., sm1 ... smn} of records. For each sj1 sj1 ... sjn satisfying the query, return r1 sj1 ... sjn as a response to the query.

This algorithm may not be particularly efficient. In particular, the efficiency of the algorithm will depend on the relative distributions of physical records in the database and on the types of queries. A

possible technique for improving the efficiency of the algorithm would involve the choice of record types, given a collection of queries and the likelihood of each query.

VI. Conclusions.

The definition of a database may allow each individual user his own interpretation of some of the relationships which can be recovered from the database. This can result in misunderstanding of the responses to queries and to inconsistency in the updating of the database.

To avoid this problem, each relationship that can be recovered from the database must be defined, either as a primitive relationship, independent of the other relationships in the database, or as an inferred relationship, in terms of how it may be inferred from the other relationships in the database.

These two kinds of relationships can be distinguished from each other in the CODASYL and IMS database models by the configurations of the access paths from which primitive relationships are constructed. In general, a "traversal rule" can be defined which describes the primitive access paths of a database. Relationships constructed from these access paths must be defined as primitives of the database. All other relationships must be defined by rules of inference.

BIBLIOGRAPHY

1. Astrahan, M. M., and Chamberlin, D. D., "Implementation of a Structured English Query Language", Proceedings of the International Conference on Management of Data, San Jose, California, May 1975.
2. Bachman, C. W., "The Programmer as Navigator", CACM 16 11 (Nov. 1973).
3. CODASYL Data Base Task Group Report, ACM, New York (April 1971).
4. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks", CACM 13 6 (June 1970).
5. Codd, E. F., "Relational Completeness of Data Base Sublanguages", Courant Computer Science Symposium (May 1971).
6. Dolk, D. R., and Loomis, Mary E., "A Methodology for the Design Generalized Query Processors for CODASYL Databases", Proceedings of the IEEE Computer Software and Applications Conference, Chicago, Illinois (Nov. 1977).
7. IBM, Information Management System/Virtual Storage (IMS/VS) System/Application Design Guide, Manual SH20-9025.
8. Minker, J., "Performing Inferences over Relational Data Bases", Proceedings of the International Conference on Management of of Data, San Jose, California, (May 1975).
9. Stonebraker, Michael, "Access Control in a Relational Data Base Management System by Query Modification", Proceedings of the 1974 ACM National Conference, San Diego, California (Nov. 1974).
10. Zloof, Moshe, and de Jong, S. Peter, "The System for Business Automation (SBA): Programming Language", CACM 20 6 (June 1977).