PUC

Series: Monographs in Computer Science
and Computer Applications

Nº 6/76

FORMAL ASPECTS OF THE RELATIONAL MODEL

by

A. L. Furtado

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 209 — ZC-20
Rio de Janeiro — Brasil

Series: Monographs in Computer Science
and Computer Applications

Nº 6/76

FORMAL ASPECTS OF THE RELATIONAL MODEL*

by

A. L. Furtado

Series Editor: Sérgio E. R. Carvalho April, 1976

* This research was conducted under the sponsorship of the Brazilian Government Agency FINEP, under contract nº 244/CT, and developed by the Data Base Group of the Depto. de Informática-PUC/RJ.

Copies may be requested from:

Rosane Teles Lins Castilho, Head
Setor de Documentação e Informação
Depto. de Informática - PUC/RJ
R. Marques de São Vicente, 209 - Gávea
20,000 - Rio de Janeiro - RJ - BRASIL

ABSTRACT:

The relational model of data bases is studied from three interdependent view points. Relational data bases are first modelled by directed hypergraphs, a concept derived in a straightforward way from Berge's hypergraph theory. Then the abstract directed hypergraphs are interpreted using a linguistic model, and finally represented as a necessary step towards computer implementation.

The normalization of relations is briefly discussed in the context of the three approaches.

KEY WORDS:

Relational data bases, hypergraphs, directed hypergraphs, case grammars, semantic networks, adjacency lists, normalization of relations.

RESUMO:

O modelo relacional de bancos de dados é estudado de três pontos de vista interdependentes. Os bancos de dados relacionais são primeiro modelados por hipergrafos dirigidos, um conceito derivado diretamente da teoria de hipergrafos de Berge. Depois, os hipergrafos dirigidos abstratos são interpretados usando um modelo linguístico, e finalmente representados como um passo necessário para implementação em computador.

A normalização de relações é discutida brevemente no contexto das três abordagens.

PALAVRAS CHAVE: Bancos de Dados relacionais, hipergrafos, hipergrafos dirigidos, gramáticas de caso, redes semânticas, listas de adjacência, normalização de relações.

ACKNOWLEDGEMENTS

The author is indebted to G.C. Magalhães, N. Zi viani and T.H.C. Pequeno, participating in the HYADES project.

CONTENTS

1- INTRODUCTION

In this research the relational model of data bases [1] is examined under different formal aspects.

A relational data base (RDB) consists of domains (sets) $D_1, D_2, \ldots, D_m$ and relations among them. An n-ary relation R on domains $D_{i_1}, D_{i_2}, \ldots, D_{i_n}$, where the D_{i_j} are not necessarily distinct, is defined by $R \subseteq D_{i_1} \times D_{i_2} \times \ldots \times D_{i_n}$, i.e. a subset of the Cartesian product of the indicated domains.

It is required that all relations be in first normal form, which simply means that all domains in a relation must be simple, their elements being indecomposable objects.

Relations are represented as two-dimensional arrays or tables, whose columns denote the constituent domains and whose rows denote the n-tuples (or simply tuples), i.e. the combinations of elements from the constituent domains which stand in the relation.

The description of an RDB, by enumerating the domains and defining the relations, is called elsewhere [2] a schema, while the individual domain elements and tuples constitute the instances of the schema.

Queries to the RDB can be formally expressed through a relational algebra and a relational calculus [3] of equal expressive power but different degree of proceduralness, the calculus being less procedural and somewhat closer to a natural language formulation of the queries.

The original relational model, being purely algebraic, lacks the visual intuition provided by graph-theoretic formalisms. In fact, attempts to use graphs to model n-ary relations, for n greater than two, have failed; it will be shown in section 2 that we have to resort to directed hypergraphs - a concept derived in a straightforward way from the hypergraph theory of Berge [4] - in order to overcome the difficulties.

Also, the original relational model does not stress the meaning of the relations. In section 3 we shall consider the semantics of relations when interpreting the abstract directed hypergraphs.

The only data structure used with the original relational model is the relation table. Other data structures are used in actual implementations; in section 4 we shall show that some of these additional structures are in fact logically essential for representing the directed hypergraphs.

I all sections the conformity to the first normal form is demonstrated. In section 5 the further normalization leading to third normal form [5] will be reviewed in the context of the notions developed in the previous sections.

This research presents the theoretical foundations of project HYADES (Hypergraph Databases), being conducted at the Pontifícia Universidade Católica do R.J.

2- THE ABSTRACT MODEL

Binary relations can be displayed very simply and conveniently by way of directed graphs.

For n-ary relations a decomposition into binary relations has been tried. The first problem, of course, is that more than one decomposition is possible; thus, if $(a,b,c) \in R$ we can have either $a\ R_1(b\ R_2c)$ or $(a\ R_1b)R_2c$. However, there is a worse problem known as connection trap [1].

As an example, let $R = \{(a,b,c'),(a',b,c)\}$; we would expect that the directed graph below corresponds to R (Fig. 1).

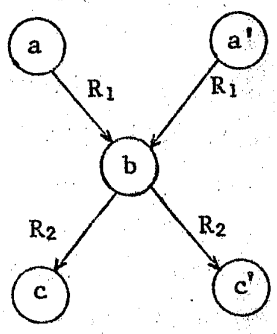

Figura 1: A ternary relation as a directed graph

The drawback is that the figure shows more than we intended: it shows that (a,b,c) and (a',b,c') also belong to R, which in our example is false.

Faced with this unsuitability of directed graphs to model n-ary relations for $n>2$, we resort to hypergraph theory [4].

A simple hypergraph H is an ordered pair (V,E) where

- V is a set of nodes, which are indecomposable elements - cf. first normal form in the Introduction;

- E is a set of edges, which are sets of nodes from V, i.e. $E \subseteq 2^V$ (the powerset of V).

Since in a set no element can appear repeatedly, no two edges in E are identical in the sense of comprising the same nodes. This is what is meant by a hypergraph being simple; the removal of this restriction leads to multiple hypergraphs.

A hypergraph is uniform of rank n, for n a positive integer, if all the edges have (are incident to) n nodes.

The notion of direction is added by requiring that the edges be ordered sets. We shall be interested only in uniform directed hypergraphs, where $E \subseteq V^n$ (the Cartesian product of V by itself n-1 times).

In a directed graph we identify two roles for the endpoints of an edge. Graphically, one role is distinguished by the presence of an arrowhead touching the nodes playing that role. Sometimes the roles are also distinguished by names, such as origin and destination, according to a particular interpretation of the graph.

In directed hypergraphs we shall defer the assignment of names to the roles until the next section. For the time being, roles will be designated by numbers, so that nodes appearing in the $i^{\underline{th}}$ place of the ordered sets (edges) in E are said to be playing role i. Note, incidentally, that the definition of uniform directed hypergraphs allows the same node to play different roles, even in the same edge.

For graphical representation the conventions proposed in [4] are adopted, with the addition that if a node v plays role i in an edge e, then we draw i lines between v and e (Fig. 2).

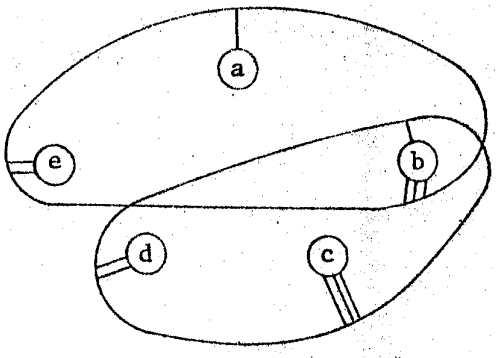

Figura 2: Uniform directed hypergraph with E = {(a,e,b),(b,d,c)}

As a generalization of the concepts of in-degree and out-degree, we define as the i-degree of a node the number of edges were the node plays role i.

In section 5 we shall need an extension to this concept. We shall have to talk of the number of edges incident to a set of nodes, each node playing an indicated role. Let $S = (s_1, s_2, \ldots, s_k)$ be an ordered set of nodes, and $J = (j_1, j_2, \ldots, j_k)$ an ordered set of distinct positive integers designating roles; the J-degree of S is the number of edges where each node s_p plays role j_p for p ranging from 1 to k.

The set V of nodes can be many-sorted, consisting of the union of different sets or domains. With this we almost have the desired correspondence to an n-ary relation. All that is missing is the proviso that to each role there must correspond only one domain, whereas the same domain can play more than one role; to make this clear let us define, as in the Introduction, $E \subseteq D_{i_1} \times D_{i_2} \times \ldots, \times D_{i_n}$, where the D_{i_j} are not necessarily distinct. Clearly, domain elements are thus assimilated to nodes and relation tuples to edges. From now on this definition is assumed for all directed hypergraphs to be discussed in this paper.

In the situation where the D_{i_j} are distinct we may use a simplified representation, placing the nodes of each domain at a different height and omitting the lines which distinguish the roles. The relation in Figure 1 is

shown in this representation; note the absence of the connection trap problem (Fig. 3).

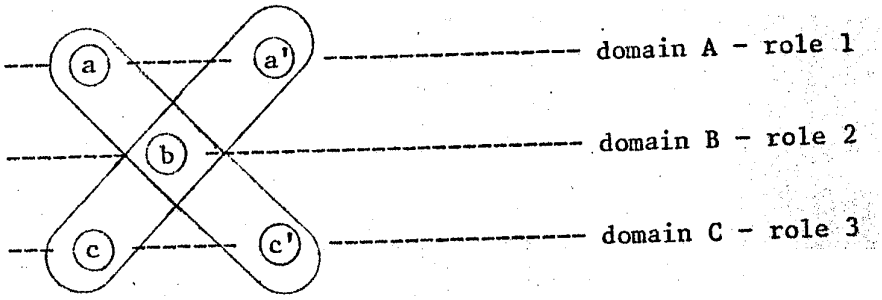

Figure 3: Directed hypergraph with different domains playing different roles.

An entire RDB is then a family of directed hypergraphs of various ranks, possibly sharing some domains. It can also be viewed as one multiple directed hypergraph with labelled edges (the labels denoting the different relations), with the restriction that multiple edges cannot have the same label.

In correspondence with hypergraphs, several kinds of representative graphs can be defined [4]. Here, two kinds will be introduced: the schema representative graph and the instances representative graph.

In the former

- one set of nodes consists of one node for each domain;
- the other set of nodes consists of one node for each relation;
- the graph is bipartite: relation nodes are linked to domain nodes through edges labelled with the role played by the domain in the relation.

For example, let the RDB consist of the domains A,B,C and D, and the relations $R \subseteq A \times B \times C$ and $S \subseteq B \times D$ (Fig. 4).

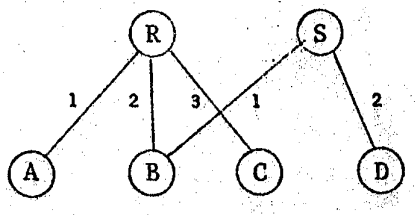

Figure 4: Schema representative graph

In the latter

- one set of nodes consists of the original RDB nodes;
- the other set of nodes corresponds to the edges of the original RDB;
- the graph is also bipartite: nodes in the second set are linked to nodes in the first one through edges labelled with the role played by the RDB nodes in the RDB edges.

Using the previous example, suppose that $R = \{(a,b,c)\}$ and $S = \{(b,d),(b',d')\}$ (Fig. 5)

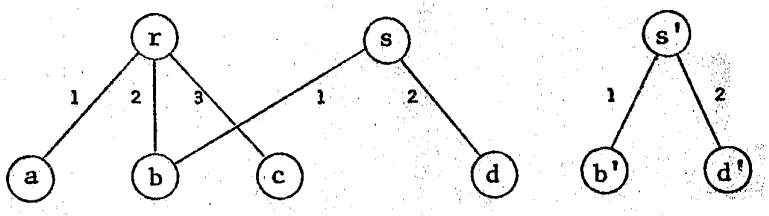

Figure 5: Instances representative graph

As noted in [4], well-known properties of graphs can be extended in a natural way to hypergraphs. It is to be expected that algorithms can be similarly adapted, which would be useful in problems such as partitioning an RDB into components for convenient storage allocation, studying connectivity, characterizing clustering patterns, etc.

An immediate consequence of adopting a graph-theoretical model is that, being assimilated to nodes, the domain elements exist by themselves. This is at variance with the original relational model [1] where the existence of a domain element is conditional to its presence in some relation tuple.

As usual, we shall admit that nodes can be created independently and are not deleted even if all edges incident to them are deleted; a node can only be deleted by an explicit action. An edge, of course, can be deleted either explicitly or if a node incident to it is deleted.

3- THE LINGUISTIC MODEL

The task here is to interpret the abstract directed hypergraphs in order to attach meaning to them.

An RDB purports to be a description of some mini-world. It consists of facts about the mini-world, and such facts are expressed by using words assembled to form sentences.

So it is natural to associate words (or phrases) to nodes and simple sentences to edges of the directed hypergraphs. Always in compliance to the first normal form requirement we do not use the full (recursive) definition of sentences, whereby a sentence constituent associated with a node might in turn contain a sentence.

It is usual to distinguish a surface and a deep treatment in linguistics, both of which are of interest here.

At the surface level the same fact many admit several paraphrases. For example, take the domains

S# - suppliers
P - parts
J - projects

and the relation

$$S \subseteq S\# \times P \times J$$

which could be interpreted in any of the following equivalent ways, letting $x \in S\#$, $y \in P$, $z \in J$:

- x supplies y to z

- y is supplied by x to z

- z obtains its supply of y from x

Although they express the same fact all these parapharases are useful in that they bring in turn to the front the element from each of the domains. This is convenient for expressing a simple query, e.g.: "which part is supplied by x to z?" It is also convenient for the formation of complex sentences (and complex queries) as the next example will illustrate.

For the example we introduce the following additional domains:

T - means of transportation
L - local agents
A - geographical areas

and the additional relations:

$R \subseteq L \times S\# \times A$

$D \subseteq P \times T$

which mean (one paraphrase only is given):

R - "x represents y in z" where $x \in L$, $y \in S\#$, $z \in A$*

D - "x is delivered by y" where $x \in P$, $y \in T$.

The three relations therefore are expressible by simple sentences. From them we may form complex sentences by coordination or by restrictive relativization [6], an example of the latter being:

* The domain element brought to the front does not always become the subject in a query, e.g. "in which area does x represent y?" for $x \in L$, $y \in S\#$.

"x represents y who supplies z which is delivered by w" where $x \in L$, $y \in S\#$, $z \in P$, $w \in T$.

Note that in each relative clause the domain element that is restricted is brought to the front and is represented by the pronoun.

We saw that several paraphrases, at the surface level, may convey a single fact. In each paraphrase a different element is brought to the front, and the syntactic categories of the elements also vary. By contrast, deep case grammars [7] provide a unique way of expressing a fact.

It is unfortunate that no consensus exists around a classification of cases that would be adequate for all applications [8]. For the purposes of this paper we shall use part of a classification proposed in [7]:

a - agentive
o - objective
d - dative
ℓ - locative
i - instrumental

Clearly cases are a good interpretation for the abstract roles of the directed hypergraphs. This identification in turn leads to identifying semantic networks [9] with the previously defined representative graphs.

Maintaining our orientation, we shall distinguish a schema semantic network, in which one set of nodes are named after the verbs dominating the relations, and the other set after the generic nouns (domains) supplying the case arguments of the verb. Our example in displayed using the abbreviations introduced before (Fig. 6).

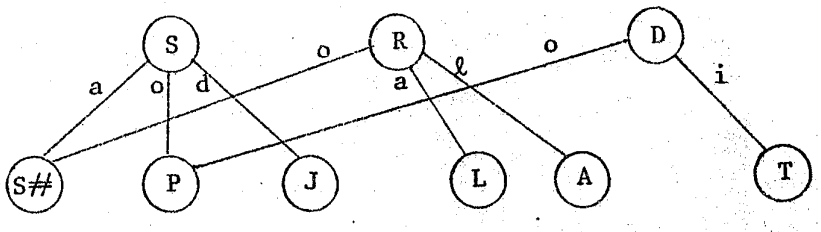

Figure 6: Schema semantic network

We shall also distinguish an instances semantic network. A possible fragment of an instances semantic network corresponding to the above schema is given (Fig. 7).

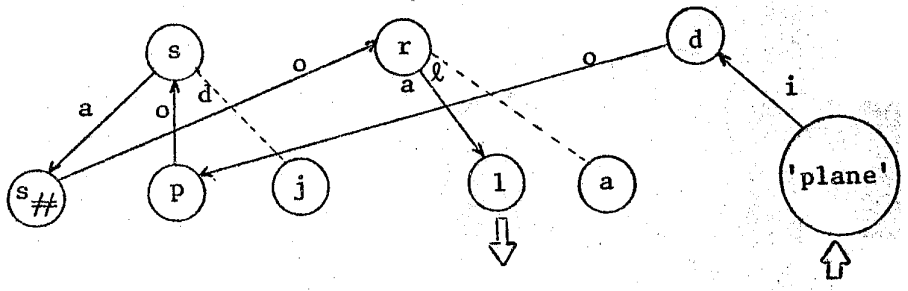

Figure 7: Fragment of instances semantic network

In Fig. 7, in addition to showing a fragment of an instances semantic network, we show part of the navigation [10] i.e. traversal, that must be performed in order to answer the query:

> "which agents represent some supplier that supplies some part that is delivered by 'plane'?"

Note how the adopted paraphrases determine a direction in the query which is inverted when proceeding towards the answer. Even more complex queries involving alternative or simultaneous restrictions, may induce more complex (acyclic) subgraphs than the linear paths of the example above; quantification, as in the "some " of the example, must also be considered.

The query above would be expressed in a compact way using the relational calculus [3]. Such format is unsuitable for casual users, but the difficulty does not come from the mathematical notation alone; rendered into English, the calculus expression would still look very much different from the initial query, by undoing its nested scheme of successive restrictions (given by the embedding of relative clauses) and principally by expressing itself in terms of tuples and quantification on tuples. The latter feature reflects again the tuple-oriented rather than domain-oriented approach of the original RDB model.

On the other hand, relational calculus expressions allow us to associate, in a query, domains that are not connected in the directed hypergraphs expressing the relations. This is done when such domains are homogeneous to each other, so that their elements can be linked through the comparison operators $=$, $\neq$, $>$, $<$, $\geq$, $\leq$*. We simply note that these operators also express (binary) relations which can be considered conceptually as part of the model, regardless of the corresponding edges being explicitly drawn or not.

Now consider another query on the same example: "which agents deal with air companies?". A human being would promptly notice that this query is equivalent to one already encountered before: "which agents represent some supplier that supplies some part that is delivered by 'plane'?"

The shorter form of the query cannot be directly answered by the automatic processor, because the deal relation is not mentioned in the data base. So the processor should be able to translate it into the second form, and the choice of the strategy whereby the processor will do this is an indicator of how far one proposes to go into the realm of artificial intelligence. A rather simple but effective strategy is to allow the introduction of definitions [11]; the deal relation would be defined in an obvious way

* In [17] we introduce the concept of classes, a class being a set of domains whose elements are comparable through these operators.

in terms of the represent, supply and be delivered relations. More complex strategies involve the use of inference [12], whereby, in a sense, the processor itself would be led to compose the definition.

Again let us add another domain to our data base:

M - managers

and suppose that one wants to describe somehow the non-simple sentence "x knows that y supplies z to w", where x ε M, y ε S#. z ε P, w ε J.

We said that complex sentences would only be formed through coordination and relativization, and the above sentence contains a subordinate nominalized clause. First we show why this is not permitted in the model - the instances representative graph illustrates the point (Fig. 8).

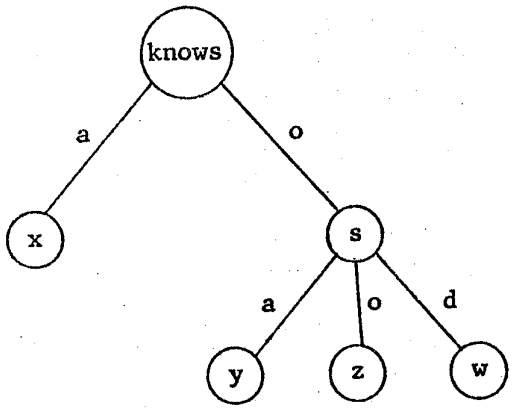

Figure 8: A forbidden configuration

Here, what is known is the entire subordinate clause rather than one of its elements, as it happens whith relativization. The illegality of this construct can be seen under several points of view: it contains a relation which is not in first normal form, it cannot be the representative graph of any of our directed hypergraphs, one case argument of the know relation is a sentence.

One would argue that the need to cope with such situations shows that Codd's first normal form is too restrictive. However, we feel that the great simplicity which comes from its adoption warrants the effort to overcome the difficulties that it may raise, which is what we do in the sequel for the particular problem just mentioned.

We allow ourselves to assign labels to sentences, and use the labels in lieu of the sentences as case arguments. So, we can now have the decomposition into two simple sentences: "v: y supplies z to w", and "x knows v"; or, in one complex sentence: "x knows the fact (v) that y supplies z to w" (Fig. 9).

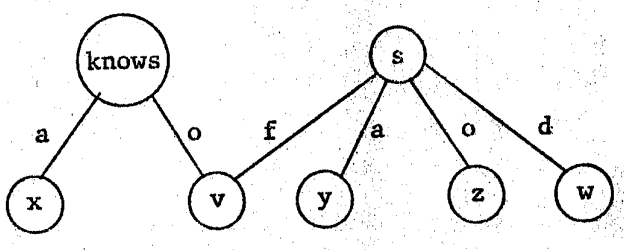

Figure 9: Using labelled sentences

Note that we had to introduce a domain of sentence (or fact) labels and a sentence label case (f).

A different problem has to do with the notion of characteristics [13]. A characteristic is a relation between objects that exist by themselves and others that exist only for describing the former, as for example the address of a supplier, the colour of a part, the quantity of a part supplied to a project, etc.

Apart from this rather "relative" distinction based on the more or less essential nature of the objects, characteristics pose a problem in connection with case grammars: not only actions but also states, qualities, etc. must be considered in an RDB, and in such situations should still a verb or verb phrase be thought to dominate the sentence (e.g. is coloured)? also, for the almost open-ended classifications of categories such as adverbials,

should we accordingly allow a proliferation of cases? (cf. Fillmore's "modality" [7]).

The recognition of characteristics as a special type of relation is found for example in [13]. Although seeing the merits of this approach, we still prefer to treat all relations uniformly.

4- THE ACCESS MODEL

In this section the representation of the directed hypergraphs will be discussed in terms of logical data structures adequate for computer processing.

Graph representations can be extended in natural ways for representing hypergraphs. In [4] the incidence matrix of a hypergraph is introduced. The adjacency matrix of a hypergraph H of rank n with m nodes is an m x m x ... x m (n times) array A, where $A(i_1,i_2,\ldots, i_n) = 1$ if the set of nodes $i_1,i_2,\ldots,i_n$ is an edge of H, otherwise $A(i_1,i_2,\ldots,i_n) = 0$.

For relatively sparse graphs the adjacency list [14] is particularly covenient. It consists of a one-dimensional pointer array where each entry corresponds to a node. For each node v, the corresponding pointer leads to the list of nodes adjacent to v (Fig. 10).

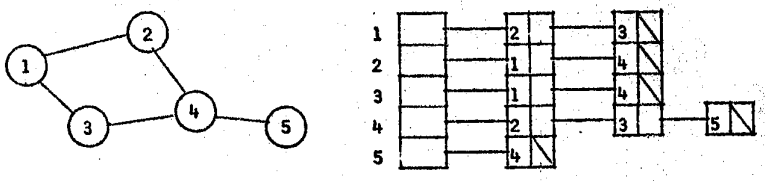

Figure 10: Adacency list of a graph

Note that each list - cell contains two fields: one to store the node which together with v forms an edge, and a pointer to another node, if any, also forming an edge with v. For a hypergraph of rank n the list-cells will have n fields: the other n-1 nodes in an edge plus the pointer field; besides, the same edge requires the presence of n such cells, each cell attached to the list of each node involved. Hence, for a hypergraph with m nodes and ℓ edges the total number of fields is

$$m + \ell \cdot n^2$$

Another format for the adjacency list is possible. A two-dimensional array with n columns and ℓ rows is added; each row contains n pointers to the nodes which form an edge. With this, two-field list-cells are sufficient: the first field points to the appropriate row in the edge array, while the second field leads to another list-cell as before. The total number of fields is

$$m + 2.\ell.n + \ell.n = m + 3.\ell.n$$

This second format, which is patently uneconomical for graphs, is suitable for hypergraphs of higher rank, the break-even point with the first format being clearly n=3. An example is given below (Fig. 11).

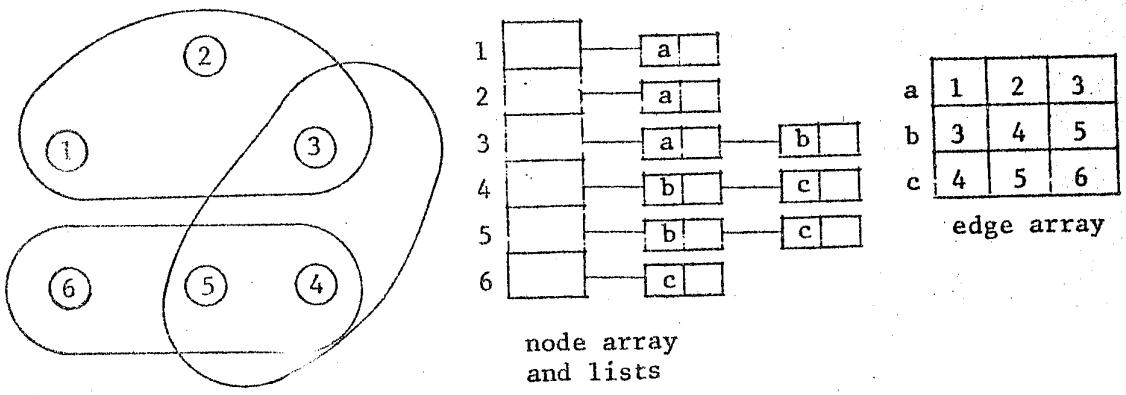

Figure 11: Adacency list of a hypergraph

The order of appearance of the nodes of an edge in the edge array is immaterial if we are representing undirected hypergraphs. For directed hypergraphs the column order matters, and also the lists for the occurrence of a node in edges where the node plays different roles should be separated. Besides, if the set of nodes is many-sorted, separate node arrays must be used for each domain. If there is more than one directed hypergraph (more than one relation) an equal number of edge arrays must be used. These are the extensions required by the full generality of RDBs.

Its is interesting to note that in the current data base terminology

- the edge array is the relation table, which is the fundamental data structure in the original relational model;
- if the edge array is stored column - wise it is called a collection of (totally) transposed files;
- the lists constitute inverted files;
- the node arrays are data pools.*

The adjacency list data structure is redundant, because the lists duplicate the structural information conveyed by the edge array. In fact, all the relational algebra operations [3] can be performed using the edge array alone. It is slightly less trivial that the same operations can be performed using the node array plus lists exclusively (in this event the a, b and c in Fig. 11 would cease to be pointers to rows of an edge array, to merely become distinct "names" for edges; note that if the lists of nodes 3,4,5, for example, are intersected the common name b is found, indicating that there is an edge incident to the three nodes).

Redundancy is a useful feature for recovering from a loss of information in a computer - implemented data base. More importantly, it provides a capability that is directly relevant to the present discussion, namely efficient two-way access between nodes and edges (or equivalently domain elements and tuples, words and sentences).

The question of which nodes are incident to a given edge can be efficiently answered using the edge tables, whereas the node arrays plus lists are convenient for determining which edges are incident to a given node. Recall how in section 3 the need for alternate navigation between nodes and

* For simplicity we have been omitting the array of the actual elements(values) from the domains, which exists "in parallel" with the pointer arrays.

edges was suggested (see Fig. 7).

The full adjacency list* also enhances the original relational model in approximately the same way as the network model enhances the hierarchical one [15]: as in the trees of the hierarchical model a domain element may be repeated several times, the same happens in the original relational model where a domain element may be repeated in several tuples. Both in networks and in the adjacency lists a domain element is represented only once; in the adjacency list it is represented in the node array, whereas the edge table simply contains pointers, in the rows corresponding to the appropriate edges, to this unique occurrence. The unique representation feature is particularly useful from the point of view of consistency and integrity requirements.

We find that the present approach distinguishes data bases still further from the conventional file systems, where the unit of information is the "record" with multiple items, which often mirrors the card lay-out of the initial input. It would appear that the tuple - oriented approach still conforms to the record - oriented organizations.

At the access level the benefits of first normal form are readily perceived. The entries in the edge arrays are always single pointers to single domain elements in the node arrays.

Another consequence, is that the representative graph is, in the CODASYL terminology [2], a simple plex. To clarify this notion, we shall draw again the schema representative graph of Fig. 6, upside down and with single or double arrowheads, where

- a single arrowhead pointing from a node of type α to a node of type β indicates that there can be only one instance node of type β for each instance node of type α;

* In practice this structure must be further enhanced by the use of directories for storing schema information.

- a double arrowhead pointing from a node of type α to a node of type β indicates that there can be several instance nodes of type β for each instance node of type α (Fig. 12).

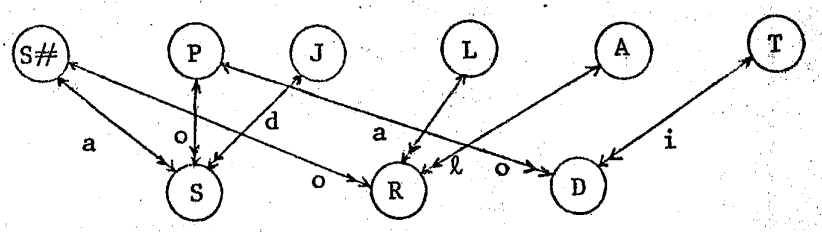

Figure 12: A Schema representative graph as a simple plex

The fact that in one direction all arrowheads are single characterizes simple plexes, which alone can be implemented directly in the CODASYL proposed system via the set organization. For each connection in the schema a set would be defined, having owners of some domain record type, and members of some relation record type (e.g. one set has owner records S# and member records S).

In addition, for allowing access within each domain and each relation, every one of these record types determines sets whose members are their instances and whose owner is always the system.

We do not intend to propose this CODASYL - style implementation for adjacency lists as the best one available, but simply to show a further point of contact between the relational and network models as a matter of theoretical interest.

5- NORMALIZATION

Codd has also proposed two further normalization steps, leading to the second and third normal forms [5].

This further normalization has to do with the choice of the primitive relations, i.e. with the design decision of what relations whould be directly implemented in the RDB (other relations should be derivable from these).

The discussion of normalization will be used here for reviewing some implications of our approach.

The concept of functional dependence is the basis for normalization. A set of domains B is functionally dependent on a set of domains A in a relation R if, at every instant of time, each combination of domain elements from A is associated with only one combination from B. For example, in an academic RDB, a relation associating rooms, time, and courses exhibits a functional dependence of courses on the set of the two domains rooms and time, because only one course can be lectured in a given room at a given time.

A set A of domains on which the set of all the other domains in a relation R depend is called a key.

Since second normal form is simply a preliminary step towards the third, we shall proceed to the definition of the latter [16]:

- A relation R is in third normal form if it is in first normal form and, for every domain collection C of R, if any domain not in C is functionally dependent on C, then all domains in R are functionally dependent on C.

Consider a relation with the meaning: "employee x works in department y under chairman z". The relation is not in third normal form because given the department its chairman is determined, but the other domain - employees - is not dependent on departments (a department can have

several employees).

The third normal form requirement is equivalent to constrain all relations to express single facts. It is noteworthy that although the example relation is expressible in a single sentence it conveys two basic facts: "employee x works in department y" and "department y is headed by chairman z".

The inconvenience of relations not in third normal form is that we may wish to assert, cancel, or update one of the two or more facts compressed in a single relation, which causes the so-called insertion, deletion, or update anomalies: in the original works relation if a chairman is assigned to a department but the department still has no employees we cannot record the assignment since the pattern of that relation includes employees; if all employees, but not the chairman, are transferred to another department the information that the chairman is still assigned to the department is lost, for the same reason; finally, if the chairman of a department is replaced all the tuples referring to employees working in the department have to be updated (several updates caused by a single event). Avoiding the anomalies is the reason for splitting the original relation.

Turning to our model, we can make the following remarks:

a. At the abstract level. The notion of a set of domains A being a key in a relation R is translatable in terms of the J-degree of a set of nodes S. Let each j_p in $J = (j_1, j_2, \ldots, j_k)$ designate a domain, and hence the entire J designate a set of domains A. If for all combinations S from A, the J-degree is equal to one then A is a key in R. Similarly, functional dependence and third normal form can be easily defined.

b. At the linguistic level. Requiring, as we do, that all primitive relations be expressible as simple sentences is a necessary but not a sufficient condition for their being in third normal form, as shows the previous example. This reinforces the point that these questions should be considered at a deep semantic (linguistic) level [13]: given a fact we must see if it can be further decomposed into more basic facts (this is done in [12] for attaining mechanical "understanding" and allowing certain kinds of inference).

The question now becomes: how deeply should we go in this analytical process?

c. At the access level. As long as we do not consider a computer implementation, no "cost" is incurred when we continue towards finer decompositions. At the access level, however, one realizes that having a larger number of relations means to have a larger number of arrays, and to fragment the information to a point where an extensive reconstruction will have to take place for answering almost any common query.

So we see that at the abstract level we can characterize the normalization precisely, and therefore could determine what relations should be decomposed. However it is better to postpone the decision and try to determine, at the linguistic level, which are the basic single facts; indeed we learn that, to an extent, as we decompose a deeper understanding is achieved. At the access level the cost of such decomposition is established and thus the desirability of a compromise is realized.

We feel that the compromise should be determined by pragmatic criteria. Except if we wish to have the capabilities of an artificial intelligence system [12,13] it is not difficult, by surveying the habits of the intended users, to find which are considered to be the basic facts; also, since the anomalies are the main immediate motive for decomposition, we may invert the problem and decide to decompose only if anomalies are likely to occur in real practice.

As a rather extreme case, consider the following relation: "employee x worked in department y under chairman z in the year w". This is "historical" information and, as such, not subject to modifications; hence no anomalies will ever arise.

More generally, if a basic fact can be asserted, negated, or updated independently, then it should constitute a separate relation.

Apart from these considerations, the minimum extent of decomposition in our model comes from the requirement that primitive relations be expressible as simple sentences (which might be termed s-normal form), and that derived

relations be expressible as complex sentences formed by coordination and restrictive relativization only.

6. CONCLUSION

The models described provide three interconnected approaches to RDBs. We suggest that they must be considered together when designing a relational data base system.

One such system is the aim of project HYADES. Until the present stage, the logical data structure [17] and part of the physical storage organization [18] have been defined.

REFERENCES

1 - CODD, E.F. A relational model for lage shared data banks. CACM,13 (16): 377-87, June 1970.

2 - CODASYL DATA BASE TASK GROUP REPORT. New York, ACM, 1971.

3 - CODD, E.F. Relational completeness of data base sublanguage In: Data Base Systems. Englewood Cliffs, N.J., Prentice - Hall, 1972.

4 - BERGE,C. Graphes et hypergraphes. Paris, Dunod, 1970

5 - CODD, E.F. Further normalization of the data base relational model. In: Data Base Systems. Englewood Cliffs, N.J., Prentice-Hall, 1972.

6 - STOCKWELL, R.P. et alii. The major syntactic structures of English. New York, Holt, Rinehart & Winston, 1973.

7 - FILLMORE,C.J. Towards a modern Theory of case. In: Modern Studies in English. Englewood Cliffs, N.J., Prentice - Hall, 1969.

8 - BRUCE, B. Case System for natural language. Artificial Intelligence, 6 : 327-60, 1975.

9 - SIMMONS, R. Semantic networks: Their computation and use for understanding English sentences. In: Computer models of Thought and language. New York W.H. Freemam, 1973.

10- BACHMAN, C. The programmer as navigator. CACM, 16 (11) : 653-58, Nov. 1973.

11- THOMPSON, F.B. & BOZENA, H.T. Practical natural language processing: The REL system as prototype. Advances in Computers, 13 : 109-68, 1975.

12- SCHANK, R.C. Conceptual information processing. Amsterdam, North-Holland, 1975.

13- ROUSSOPOULOS, N. & MYLOPOULOS, J. Using semantic networks

for data base management. In: VERY LARGE DATA BASE CONFERENCE, Framingham, 1975.

14- AHO, A.V. et alii. The design and analysis of computer algorithms. Reading, Addison-Wesley, 1974.

15- DATE, C.J. An introduction to database systems. Reading, Addison-Wesley, 1975.

16- CODD, E.F. Recent investigation in relational data base systems. Yorktown Heights, IBM T.J. Watson Research Center, 1974.

17- FURTADO, A.L. & BRODIE, M.L. A data structure for fast relational algebra operations. Rio de Janeiro, PUC, Departamento de Informática. (to appear).

18- MAGALHÃES, G.C. et alii Especificação de um interface para um banco de dados relacional. In: INTERNATIONAL SYMPOSIUM ON METHODOLOGIES FOR THE DESIGN AND CONSTRUCTION OF HARDWARE AND SOFTWARE SYSTEMS, Rio de Janeiro, 1976.