Binary-Relational Storage Structures

R. A. Frost
Department of Computer Science, University of Strathclyde, Livingstone Tower, 26 Richmond Street, Glasgow G1 1XH, UK

Any database, no matter how complex, can be represented as a set of binary-relationships. Consequently, a structure which can store such binary-relationships is logically sufficient as the storage mechanism for a general purpose database management system. For certain applications, the advantages of using such a structure would appear to outweigh the disadvantages. Surprisingly, however, very few systems have been built which use a binary-relational storage structure. The main reason would appear to be the difficulty of implementing such structures efficiently.

INTRODUCTION

The binary-relational view of the universe is increasingly being used during the data analysis stage of database system design. The approach is currently taught at a number of British Computer Science departments, and a similar method has been used by Shave[1] and described by Rock-Evans.[2] Use of the view, however, need not be limited to this aspect of systems work. Systems which have been specified using a binary-relational schema may also be implemented using a binary-relational storage structure. The conceptual tool which helps the analyst specify user requirements may also be the means by which the designer realizes the solution at a physical level.

Use of a common conceptual framework throughout the system project is not the only advantage which results from using a binary-relational storage structure. Simplified system design and improved data-independence also result. Surprisingly, however, very few systems have been built which use binary-relational storage structures. The main reason would appear to be the difficulty of implementing such structures efficiently.

The purpose of this paper is twofold: (i) to appraise the use of the binary-relational structure as the storage mechanism in a database management system, and (ii) to present the results of a survey, and analysis of, various implementations of this structure.

THE BINARY-RELATIONAL VIEW OF THE UNIVERSE

The binary-relational view regards the universe as consisting of entities with binary-relationships between them. An entity is any 'thing' which is of interest and can be identified. A binary-relationship is an association between two entities. The first entity in a relationship is called the subject and the second entity is called the object. A relationship is described by identifying the subject, the type of relationship, and the object. For example: IBM · employs · John

N—ary relationships such as 'John bought the car from Smiths' may be reduced to a set of binary-relationships by the explicit naming of the implied entity. For example:

sale # 1 . buyer . John
sale # 1 . seller . Smiths
sale # 1 . item sold . car

English description	*Corresponding set of binary relationships*
1. John is married to Sally	(John . married to . Sally)
2. *a* bought *b* from *c*	(sale #1 . buyer . *a*) (sale #1 . item . *b*) (sale #1 . seller . *c*)
3. John is a policeman	(John . ∈ . policeman)
4. John said Paul thinks that the moon is made of cheese	(moon . made of . cheese)#1 (Paul . thinks . #1)#2 (John . said . #2)

Figure 1. The binary-relational view.

The binary-relational view is used in artificial intelligence as well as database work. In Fig. 1, we give a few examples showing how parts of the universe may be represented by sets of binary-relationships. Notice that:

(i) no distinction is made between 'things' and 'properties' of things;
(ii) *n*-ary relationships are reduced to sets of binary relationships;
(iii) if a proper name uniquely identifies an entity, in that part of the universe in which we are interested, then that name may be used to represent the entity in the relationship. If this is not the case, then a new name must be introduced. For example, if patient names do not uniquely identify patients, then the situation: 'the patient called Smith, aged 27, is in ward 14', must be regarded as:

(patient #1 . named . Smith)
(patient #1 . aged . 27)
(patient #1 . is in . ward 14)

(iv) imagined as well as real relationships may be depicted.
(v) entity sets are regarded as entities.
(vi) ∈, the set membership relation, is treated like any other relation. This simplifies specification of integrity constraints and inference rules.

A detailed description of the binary-relational view, and of the binary-relational conceptual schema which derives from it, is given elsewhere.[3]

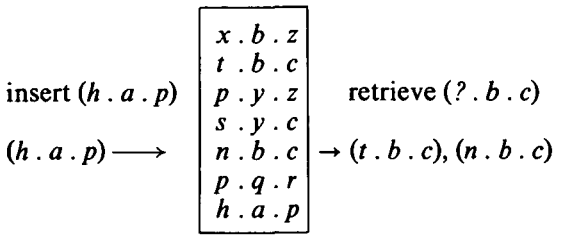
DEFINITION OF A BINARY-RELATIONAL STORAGE STRUCTURE

Any part of the universe, no matter how complex, can be thought of as a set of binary-relationships. Consequently,

CCC-0010-4620/82/0025-0358$05.00

a structure, within which representations of such relationships can be manipulated, is logically sufficient as the storage mechanism for a general purpose database system. We shall call such a structure a binary-relational storage structure.

Informally, we can describe a binary-relational storage structure as a data structure into which representations of binary-relationships (which we shall call triples) may be inserted and from which triples may be deleted or retrieved. Triples are inserted singly, and deleted or retrieved in sets. Such sets have one or more fields in common. The common fields correspond to the search key(s) which were used in the deletion or retrieval request. Thus, we may think of a binary-relational storage structure as a box which stores and manipulates triples. For example:



Feldman introduced the term 'simple associative forms' or SAFs to describe the seven basic ways in which triples may be retrieved.[4] Using our own notation, the seven forms are:

retrieve 1 $(a\ .\ ?\ .\ ?) \rightarrow$
retrieve 2 $(?\ .\ b\ .\ ?) \rightarrow$
retrieve 3 $(?\ .\ ?\ .\ c) \rightarrow$
retrieve 4 $(a\ .\ b\ .\ ?) \rightarrow$
retrieve 5 $(?\ .\ b\ .\ c) \rightarrow$
retrieve 6 $(a\ .\ ?\ .\ c) \rightarrow$
has $(a\ .\ b\ .\ c) \rightarrow$

The seventh SAF yields true or false depending on whether the triple is in the database. Other SAFs yield sets of triples which match on the given fields. For example, (? . *b* . ?) retrieves all triples with *b* as relationship type. Not all binary-relational structures described in later sections provide all seven SAFs. Those that do may be thought of as general associative memories since they provide complete content addressability to the set of triples.

ADVANTAGES AND DISADVANTAGES OF USING A BINARY-RELATIONAL STORAGE STRUCTURE IN A DATABASE MANAGEMENT SYSTEM

In this section we are only concerned with the advantages and disadvantages which arise from the inherent nature of binary-relational structures. We are not concerned with advantages and disadvantages arising from the performance of such structures. We deal with those later. For the present we assume that an efficient implementation exists.

Several advantages and disadvantages of using binary-relational storage structures have been identified by people working with them. The advantages fall loosely into two categories: (a) simplification of system design and use; and (b) improvement in data-independence.

The first of these was recognized by Levien and Maron,[5] Feldman and Rovner,[6] Ash and Sibley,[7] and Titman[8] who also noted that a simple uniform design should lead to a more reliable system. Titman[8] and Johnson[9] recognized that use of a binary-relational storage structure should improve data-independence.

The disadvantages of using binary-relational storage structures are largely due to the fact that data may only be retrieved singly, and that groups of 'related' items may only be acquired by issuing several retrieval commands.

We now discuss the advantages and disadvantages under various sub-headings.

Advantage of a consistent conceptual framework

If the binary-relational view is also used for analysis and specification of the system, as is increasingly the case, then its use as a basis for the storage structure means that a consistent conceptual framework is applied throughout much of the project.

Simple interface with other modules

Interface between a binary-relational storage structure and other modules of a database management system consists of three procedures: (i) insert (triple); (ii) delete (partial specification of triple); (iii) retrieve (partial specification of triple).

Retrieval requests such as 'list all employees of IBM' are met by issuing a simple retrieval call (IBM. employs. ?) which delivers only that data which has been requested. As Ravin points out,[10] the programmer need not be concerned with details of the backing-store data structure, and no more workspace than that required to store the requested data is needed. In addition, the task of extracting data and formatting it according to user requirements is straightforward. This is demonstrated in ASDAS where user specifications of output reports are automatically translated to application programs which may be run against the data structure to produce the required output.[11]

File design is no longer necessary

File design is one of the most difficult tasks of information system design. We illustrate this with the following example: a system is concerned with parts, suppliers of parts and machines which are constructed from these parts. Suppose the system is required to provide the following types of output: (a) 'output the description and stock-on-hand for a given part' or (b) 'output the stock-on-hand and supplier of all parts used as components of a given machine'.

Initial data analysis yields the binary-relational model shown in Fig. 2. Analysis of access path requirements may be facilitated by the introduction of four basic access types, namely:

A1—access to an entity set
A2—access from an entity set to a given member of that set
A3—access from an entity set to all its members
A4—access from one entity to one or more entities related to it by a given relation.

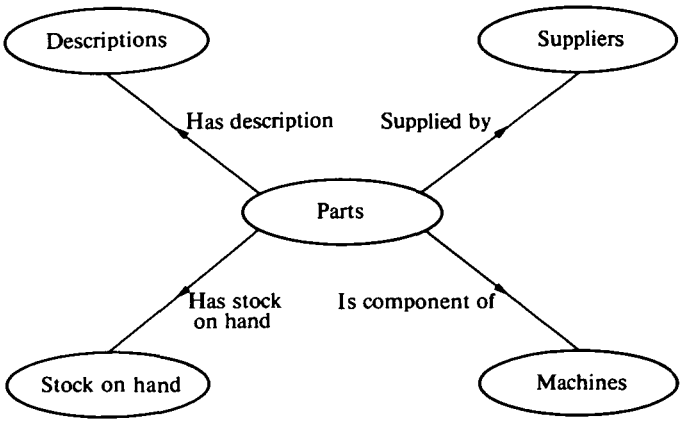


Figure 2. A binary-relational data model.

Using these definitions, analysis of the access path requirements of outputs (a) and (b) yields:

output (a): A1 to the set of parts
A2 from the set of parts to given part
A4 from part to description
A4 from part to stock-on-hand
output (b): A1 to set of machines
A2 from set of machines to given machine
A4 from machine to component parts
A4 from each component part to stock-on-hand
A4 from each component part to supplier.

Even for this simple set of requirements, choice of an appropriate storage structure is complicated. Consider the structure in Fig. 3 consisting of two hash tables. At

Hash on Part

Part	Stock on hand	Description

Hash on Machine

Machine	Part	Stock on hand	Supplier

Figure 3. A possible implementation.

first sight it might appear to be a viable solution. However, closer examination identifies several shortcomings, which include the following.

(a) Part/stock-on-hand relationships are replicated many times. As these are volatile relationships, database consistency will be difficult to maintain.
(b) Updating stock-on-hand for a given part is difficult.
(c) Part/supplier relationships are replicated many times thereby wasting space.

The problem is two-fold: in the first place, problem specifications must be much more comprehensive than that given above. In the second place, even if a complete set of requirements is available, and has been analysed, choosing an appropriate data structure is often difficult.

If, however, a binary-relational storage structure is available, specification and analysis are not so critical, and problems of file design are reduced. This is because all conceptual access paths are implemented equally efficiently in the binary-relational structure. As far as storage and retrieval of data are concerned, the analyst/designer has little else to do other than specify what entity-sets and binary-relations are to be represented by data.

Integration of multi-attribute retrieval requests into the overall system design is facilitated

To illustrate this we extend the example given above to include a third type of output: 'output those parts supplied by a given supplier and used as components on a given machine.' It is difficult to analyse the access path requirements of such multi-attribute queries without introducing aspects of implementation strategy. For example, using our previous notation, analysis might yield:

A1 to set of suppliers
A2 from set of suppliers to given supplier
A4 from supplier to parts
A1 to set of machines
A2 from set of machines to given machine
A4 from machine to component parts

However, this analysis is implementation dependent. It assumes that the request will be met by retrieving the set of parts supplied by a given supplier, and the set of parts used on a given machine, followed by computing the intersection of these two sets. This may not be the most appropriate strategy. In fact it is impossible to analyse multi-attribute retrievals in terms of access path requirements without introducing an implementation strategy.

Consequently, it is difficult to integrate multi-attribute queries into an overall design strategy. The situation is simplified if a binary-relational storage structure is used. Access paths for multi-attribute retrievals need not be analysed to the extent described above, since all conceptual links between entities, which the user wants to store as direct links, can automatically be represented by equally efficient physical access paths. The choice of methods for servicing a multi-attributable retrieval request may be left to the application programmer since all possible methods will be available to him. He may, for example, decide to retrieve all parts supplied by a given supplier and then for each of these parts find out if they are used on the given machine.

The important point to note is that multi-attribute retrieval requests need only be specified by the analyst/designer. Their inclusion does not affect choice of storage structure, and consequently, the difficult task of analysing, and integrating the results of such analysis with other aspects of the system, may be avoided.

Multi-attribute retrieval may be inefficient

Use of a binary-relational structure for multi-attribute retrieval might not be the most efficient method (irrespective of the way in which the structure is implemented). To illustrate this, consider the example given by Martin[12]: 'retrieve (the names of) all 18 year old unemployed actresses with experience in movie-acting,

and talents for singing and sky-diving'. Using a binary-relational structure, one could issue the retrievals:

(?. aged. 18)
(?. job-status. unemployed)
(?. profession. actress)
(?. experience. movie-acting)
(?. talent. singing)
(?. talent. sky-diving)

and merge the resultant sets to obtain the required data. This is probably much faster than having to do a sequential search through the whole database, which would be necessary if the database were held as a file of records (name, age, job-status, profession, experience, talents) ordered on name.

However, the search would be even faster if the data were held as a file of records ordered on name, together with appropriate inverted list/file indexes. The shortest list (possibly talent = sky-diving) could then be used, and the records of sky-divers examined to see if they were 18 year old unemployed singing movie-actresses.

It is possible that use of the binary-relational structure could be improved by employing a different strategy such as retrieving the smaller sets first:

(?. experience. movie-acting)
(?. talent. sky-diving)
(?. talent. singing)

merging these to form set S, and then for each $s_i \in$ S issuing the retrievals:

(s_i. aged. ?)
(s_i. job-status. ?)
(s_i. profession. ?)

Whatever strategy is used it is unlikely that the binary-relational structure could compete with a tailor-made structure designed for a given type of multi-attribute retrieval.

Data-independence is improved

Data-independence refers to the independence of a database and the application programs which use it. In a data-independent system, application programs are insulated from the effects of changes made to the database, its organization, and the physical devices on which it is stored. New types of data and application programs may be added with minimal effect on existing programs.

In 1976, Kent[13] noted that, although record oriented structures give us a very efficient basis for processing data, they do not reflect the evolving nature of entities and relationships. The introduction of a new entity-type, for a new application program requires a new field to be added to a record. Since records are the units of communication between application programs and the database, any existing program using the modified record will have to be modified itself. In addition, records may be organized for fast search, but only on one key field. Other, often very much less efficient, access strategies must be used if the records are to be retrieved using a different field as key. Some access paths might not be implemented at all. In such a case, addition of a new application program, wanting to use that access path, might require major re-organization of the database. These arguments can be applied, to a lesser extent, to a system based on n-ary relations.

The advantage of using a binary-relational storage structure is that it can improve data-independence. The structure provides maximum flexibility as regards the addition of new entities and relations: relevant triples are simply inserted. In addition, if all seven SAFs are available, then any new application program, whose access path requirements are consistent with the conceptual data model, may be added, since all conceptual access paths exist and are implemented with equal efficiency.

'Related' entities must be retrieved separately

One major disadvantage of using a binary-relational structure is due to the fact that 'related' entities, such as the age, address, and salary of J. Smith, may not be retrieved as a group. In this example, three separate retrieval calls have to be issued: (J. Smith. aged. ?), (J. Smith. address. ?), (J. Smith. salary. ?). Such queries would be more efficiently, and simply, answered if appropriate records or n-tuples were stored.

Batch processing techniques may not be used to improve performance

The method of collecting transactions, sorting them, and running them against a sorted master file is called batch processing. Such techniques are commonly used to speed up processing since disc-head movement is minimized. Such improvement in performance is unlikely to be achieved if a binary-relational storage structure is used, since all relevant 'fields' of each 'master file record' have to be retrieved independently.

A SURVEY OF IMPLEMENTATIONS AND APPLICATIONS OF BINARY-RELATIONAL STORAGE STRUCTURES

Some early systems

Binary-relational storage structures have a long history, dating at least from the work of Levien and Maron[5] on a system called the Relational Data File (RDF). RDF was one of the first systems capable of performing logical inference over its database. An RDF database consists of a set of triples replicated four times and held in four separate arrays: one ordered by subject, one by relation, one by object and one by triple ID. Inferential capability is achieved by allowing the user to specify how certain relations can be derived from others. This information is held in a separate file called the 'intensional' file, as distinct from the 'extensional' file which contains the basic triples. An example of an intensional file entry is:

(x/GRADUATED FROM/y) ≡ (*for some* w)
(x/RECEIVED DEGREE/(w/AWARDED BY/y))

Requests for triples given in terms of such defined relations are dynamically interpreted in terms of the relations actually stored. The main advantage of this

approach is that it permits all of the information that is represented in a triple to be utilized instead of having to enter the information redundantly in each of the several ways in which it might be referenced.

Use of a binary-relational structure for RDF was chosen primarily to simplify file structure and not for logical or linguistic reasons. To quote Levien and Maron: 'the conventional organization of databases into files, which contain records, which in term contain fields, is too awkward for efficient logical analysis, in which the smallest unit of information must be directly accessible for computer manipulation'. The triple was recognized as the smallest unit of information, and consequently, the RDF data structure was based on it.

RDF was developed primarily as a question-answering system, rather than a general purpose database management package. However, a language called FOREMAN was developed to aid user input of triples. Unfortunately, no continuation of this aspect of the system appears to have evolved. On the other hand, some of the inferential ideas seem to have influenced the work of Ash and Sibley[7] on a system called TRAMP, which is described later.

The next binary-relational system to appear was the LEAP programming language, developed by Feldman and Rovner during the early sixties and described by them.[6] LEAP is embedded in Algol, and besides having Algol-like types and statements, has facilities for manipulating sets and triples. For example, there are operators for inserting a member into a set, a loop statement of the form **foreach** *x* **in** *s* **do**—allowing a set to be processed, the **make** operator which allows the user to create a new triple, and commands corresponding to each of the seven simple associative forms (SAFs) of retrieval. The LEAP data structure consists of a set of triples replicated three times and held in three separate hash tables: one keyed on the combination relation-subject, one on the combination subject-object, and the third on the combination object-relation. The structure of these hash tables is such that all seven SAFs can be handled reasonably efficiently. A detailed description of the LEAP structure is given in Feldman and Rovner's paper.

One of the major uses of LEAP has been in interactive graphics systems. Applications include a system for block diagram problems,[14] and a system for the analysis of business decisions.[10] LEAP has also been used in pattern recognition systems. For example, a LEAP data structure has been used to store information about line drawings in a system which can analyse and compare drawings of three-dimensional objects.[15] The LEAP language has also been used to code pattern recognition algorithms.

Development and extension of some of the LEAP ideas is found in the work of Ash and Sibley on a system called TRAMP.[7] TRAMP consists of two packages of functions: the first manipulates a data storage structure which is very similar to that used in LEAP. The second package, called the relational memory, allows logical inference to be performed on the database. This deductive capability is similar to that provided by the intensional file in RDF. As in RDF, the whole strategy of TRAMP's inference mechanism is to allow the user to make a single, simple retrieval call, such that, where appropriate, a much more complex retrieval call will be generated to encompass all the defined implications of the simple call. Ash and Sibley's work is frequently cited in reviews and books on database systems (see for example Refs 17 and 18). However, direct continuation of the TRAMP project is not described in the literature. Neither do the ideas developed in TRAMP appear to have influenced any later system.

The LEAP data structure has also been used to develop an extension of PL/1.[19] The primary aim of this work was to provide a data structure for storing graphical information. Symonds also developed the ideas of Johnson[20] to show how graphical structures can be represented by a set of triples. This work led to the development of the SAM system[21] and from that to the Relational Memory (RM) system which was implemented by Lorie and Symonds.[21] RM was further extended to support *n*-ary relations and evolved into the Extended Relational Memory (XRM) described by Lorie.[23] XRM has been used as a storage subsystem for SEQUEL, GMIS, GXRAM, and QUERY BY EXAMPLE.

Some later binary-relational systems

Later work on binary-relational systems is largely unrelated to the earlier systems. Titman,[8] for example, makes no reference to any earlier work when describing his own system. Titman's system was developed to determine whether a database using a binary-relational storage structure could be implemented in a way which compared favourably, in cost performance terms, with conventional systems. In this system, triples are held in ordered arrays; one array for each relation. Each entity set is also held in a separate array and is called a value set. Figure 4 shows an example, given by Titman, of the structure for a simple bill of materials application. Part, name, and quantity are value sets. BM corresponds to the 'uses component' relation, and the array BMQ contains data representing the quantity of each component part used in each assembly. The subject of a BMQ 'triple' is the ID of a 'triple' from the BM relation. Notice that Titman's method of reducing an *n*-triple does not correspond to that described earlier, since no new implied entity has been identified and named. Titman explicitly

Part	
ID	Value
1	0099
2	0129
3	3172
4	3174

Name	
ID	Value
1	BOX
2	CARD
3	SCREW

Quantity	
ID	Value
1	1
2	2
3	4
4	6

BM		
ID	PART ID	Part ID
1	3	1
2	3	2
3	4	1
4	4	2
5	4	3

BMQ		
ID	BM ID	Quantity ID
1	1	3
2	2	4
3	3	1
4	4	1
5	5	1

Part Name		
ID	Part ID	Name ID
1	1	3
2	2	3
3	3	1
4	4	2

Figure 4. Bill of materials database.

identifies a relationship, by giving it an ID and then uses the relationship as an entity in another relationship.

Titman states that certain types of retrieval are particularly well suited to his structure. An example he gives is a request for a list of all parts in an assembly which are used in that assembly and no other. Such a request might be made when an assembly is made obsolete, for example. Titman claims that this retrieval can be answered quickly because only one relation is accessed, and in many cases the whole of this relation can be transferred into mainstore. We note, however, that in the example given this particular relation is likely to be very large. Nonetheless, since only one relation is involved, and since only data relevant to the retrieval need be transferred into mainstore, Titman's claim seems justified. Titman also points out that his structure is data-independent. For example, new relations may be easily added. Titman's analysis of his structure showed a reduction in storage requirements. The original data, held in a sequential hierarchical file, occupied 5M characters. The binary-relational form required only 1.5M characters. This is somewhat surprising. Titman gives the following reasons for this reduction: (i) some fields in the original records were blank; (ii) some field values had highly skewed distributions and (iii) some relations were small subsets of the cross product of their domains.

Titman's work has been cited in few publications. One paper was concerned with a new system called the Peterlee Relational Test Vehicle[26] and in that paper the author acknowledges that he was not the first to use value sets. Another system, the Non-Programmer Database Facility (NDB) developed by Sharman and Winterbottom[25] was directly influenced by Titman's work. NDB is a fully operational system which was developed at the IBM Hursley Laboratories, UK during the mid-seventies. The major objectives of NDB are flexibility and ease of use. In order to achieve these objectives, NDB has the following features: (i) database creation and restructuring is program controlled, (ii) triples may be retrieved using subject relation or object relation pairs as keys (note that only two of the seven SAF's are available), (iii) the storage configuration is automatically re-organized at regular intervals to improve response time.

A generalized list structure is used to store the triples. Entities are represented by three-field structures: the first field contains a pointer to the entity set to which the entity belongs, the second field contains a pointer to the triples in which the entity is involved, and the third field contains the 'value' representing the entity. The value may be a variable length name or external identifier such as Part #1234, or it may be null in which case the three-field structure acts as an internal identifier. Relationships are represented by linked structures such as that shown in Fig. 5 (a). A similar structure is also used to provide access from an entity set to its members as shown in Fig. 5 (b). The complete interlinked structure is very much like a graph and corresponds closely to the structure which the binary-relational view regards the universe as having.

Access to data in NDB is always via entity sets. For example, the query (John. employed by. ?) must be qualified by a statement to the effect that (John. ϵ. people). This retrieval would then be processed as follows:

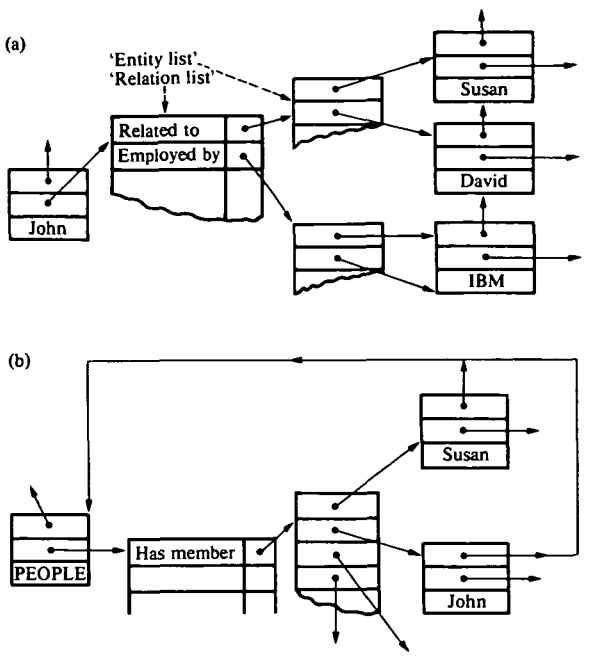


Figure 5. Representation of (a) relationships and entities in NDB; (b) set membership structure in NDB.

Locate the entity set 'people'
Follow the pointer to the relation list for 'people'
Locate the 'has member' relation
Follow the pointer to the entity list
Locate 'John'
Follow pointer to relation list for 'John'
Locate 'employed by' relation
Follow pointer to (list of) required entities

NDB has been used for many applications at a number of locations.

Another system which uses a structure similar to that proposed by Titman is the WELL system developed by Munz.[26] However, Munz does not refer to Titman's work nor to the NDB system. In the WELL system, the database is regarded as a graph called a web. Nodes of the web correspond to entities, and edges to binary-relationships. The web structure is implemented as a set of sorted arrays. An array is maintained for each relation and is ordered on subject entity. A *B*-tree-like organization is used as an index to the set of arrays. The database is organized so that the majority of accesses are from subject to object entity. Consequently, a sequential search of an array is not often required. To improve efficiency, a facility for batching insertions and deletions of 'triples' is provided. To access part of a WELL web, the user specifies a subweb in which some nodes have values and some have the word ANY. The subweb acts as a pattern which selects that part of the web (if any) which it matches. Values are then returned for the nodes containing ANY. For example, the subweb shown in Fig. 6 specifies a request for the names of all managers who manage more than seven persons who work on the project named 'DBS' and earn more than £8000. 'Count' is a mechanism for stating, the number of times a certain relationship must exist.

In Hungary, Futo *et al.*[27] have investigated the efficiency of using the PROLOG programming language to make deductions over a binary-relational database.

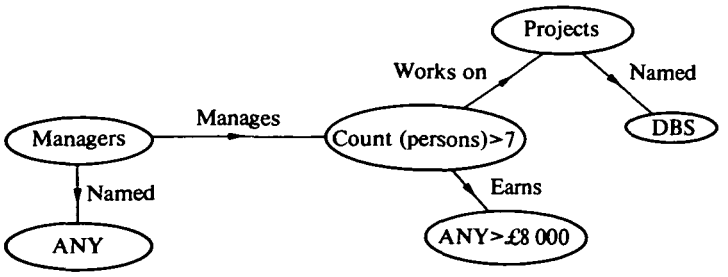


Figure 6. A WELL subweb output specification.

The particular database used contained data relating to 57 drug preparations together with 135 deduction rules. The system was reported as having retrieved and printed all deducible interactions between any given pair of preparations in an average of 2–3 seconds. In addition the program developed was capable of (i) retrieving all data about a given preparation; (ii) listing all preparations containing given agents; (iii) retrieving all data about a given agent and (iv) listing all agents belonging to a given chemical group. The implementation uses an extended PROLOG which includes use of backing store files and a faster table look-up technique.

Systems currently under development

Results of a survey, carried out by the author during 1980, suggest that there are only two other database systems which use binary-relational storage structures: the FACT machine, and a simple database system called ASDAS. Both of these systems are currently being developed at Strathclyde University.

The FACT machine. The FACT machine[28] is one of the first systems to use 'generic' information to expand the scope of a query. A 'generic' fact is one which is attributed to a set, but which applies to all members of that set. For example, the fact 'all people drink water' is a generic fact. In the FACT machine, generic information is identified by use of the quantifier 'for all' (∀). For example (∀ people . drink . water). All members of the powerset of the entity with the quantifier ∀ inherit the property indicated by the other fields in the triple. The effect is as though the property were stored explicitly with each member of the powerset.

Before generic information can be used, something must be known of the set structure within the database. Such information may be represented as a two dimensional array. For example:

	People	Employees	Smith
People	1		
Employees	1	1	
Smith		1	1

indicating that 'employees' is a subset of 'people', etc. Using the information in this array, and the generic fact given above, the query (Smith . drink . ?) may be expanded to:

(Smith . drink . ?)
(∀employees . drink . ?)
(∀people . drink . ?)

The storage structure currently used in the FACT machine, for the basic triple storage, consists of a 'master file' of triples together with a set of inverted lists. For each entity x there are three inverted lists: one giving the address of all triples with x as subject, one giving the address of all triples with x as relation, and one giving the address of all triples with x as object. To service a retrieval request, the relevant inverted lists are identified and merged to form an intersection list. The addresses in this list are then used to locate the required triples. Planned enhancements to the FACT machine include: (i) use of a special purpose processor called LEECH to speed up triple retrieval,[29] and (ii) use of new hardware techniques to represent the set-structure array. Much of the processing time of the FACT machine has been found to be taken up in determining which sets an entity belongs to. Consequently, a hardware representation of this information has been considered. At first, the cost would seem prohibitive: for 10 000 entities one would expect that an 100 000 000 element array would be required. Malone,[30] however, has found that this is not necessarily the case. The arrays tend to be fairly sparse and the entries clustered. Consequently, only limited parts of the array need be represented. Malone proposes that small, flexibly-linked, hardware modules could be used. This technique is analogous to using linked lists to represent sparse arrays in mainstore. McGregor and Malone have produced an outline design for such hardware modules which is capable of being implemented in LSI circuitry. However, the cost of manufacturing such hardware has not yet been estimated.

Although the FACT machine is still in the development stage, a prototype system has been applied, experimentally, to several problem areas. These include the following.

(i) Machine aided reviewing of literature. One feature of the FACT machine is that it can recognize 'sets' of entities which have several properties in common. This process, called 'clustering' in the FACT terminology, is carried out quite automatically. Blake has found clustering to be a useful aid in the reviewing of literature.[31]

(ii) The clustering feature also enables the FACT machine to generate hypotheses. The existence of a large set of entities with several apparently unrelated properties in common could indicate some 'correlation' between these properties. This aspect of the FACT machine is being studied with respect to factors related to gastro-intestinal disorders.

(iii) The FACT machine has been used by Malone to create a personal information retrieval system containing information relating to several hundred documents, with an index containing a few thousand terms.

(iv) The FACT machine is currently being considered for use with the Autoprog system.[33] Autoprog is a programming environment which encourages and facilitates the use of pre-coded program modules. In such a system, it is often difficult to identify all

program modules which are affected (and often need to be re-compiled) by a change made to a particular module. The problem is similar to the problem of identifying all members of the powerset of a set. This problem has been tackled in the FACT machine and it is hoped that the solution will be appropriate for the Autoprog system.

ASDAS—A simple database management system

ASDAS is also under development at Strathclyde University.[11] Like NDB, ASDAS is specifically designed for ease of implementation and use. At present it is in the experimental stage and is being used as a 'test vehicle' for research into new storage organizations, input/output languages, and schema definition languages. The storage structure currently in use consists of a set of triples replicated and held in six dynamic hash tables. In the first table the triples are hashed on subject, in the second on relation, in the third on object, in the fourth on subject and relation, and so on. This allows the seven SAF's to be answered with more or less equal efficiency. Dynamic hashing is a relatively new method of data organization.[33] The method is similar to normal hashing but differs in that a dynamic hash table can grow and shrink as the database changes size. This improvement over conventional hashing is achieved by the use of an index structured as a forest of binary trees. Two additional advantages of using dynamic hashing are: (i) retrieval of a triple requires only one disc access; (ii) the technique does not suffer from the problems normally associated with collision and deletion.

At present, the interactive query language developed for ASDAS is very basic. However, a fairly powerful report specification language is available. The language is called WAROUT and is based on ideas developed by Warnier.[34] As an example of the use of WAROUT, consider the specification of a list of all 20 year old students, whose hometown is Hull, in which the name and details of courses attended is to be printed:

```
'for all' ∈, students ['if' hometown is = Hull
                       'and' aged = 20
                       'then' [named,
                               attends [coursename is,
                                        taught by [named]
                                        ]
                               ]
                       ]
```

Specification of requirements in WAROUT may be regarded as the navigation of a route through the database. The starting point for the route is either a single entity or a set of entities with some property in common. In the example above, the starting point is specified by 'for all' ∈, students [which is translated, quite straightforwardly into a set of commands containing the command RETRIEVE (? . ∈ . students). The route is continued down paths which are identified by the specification of relation names such as 'hometown is', 'aged', 'named', 'attends', etc. Some paths are only followed to retrieve entity representations which are to be used within the context of a conditional statement, e.g. 'if' hometown is = 20. Other entities may be regarded as 'throughroutes' whose value is not to be printed, e.g. in the statement '. . . taught by [named]', the representation of the lecturer is not to be printed, although the representation of the lecturer's name is to be printed.

Translation of WAROUT specifications, by ASDAS, is made simple by the nature of the binary-relational storage structure interface. For example, the specification:

```
'for all' ∈, students [named,
                       attends class [held in room]
                      ]
```

would be translated into code with the following structure:

```
retrieve (? . ∈ . students);
stack students delivered;
while students left on the stack
  do begin
     retrieve (peek . named . ?);
     print name delivered;
     retrieve (peek . attends class . ?);
     stack classes delivered;
     while classes left on the stack
       do begin
          retrieve (peek . held in room . ?);
          print room number delivered;
          pull from stack
          end;
     pull from stack
     end
```

In addition to WAROUT, Frost *et al.* have developed a conceptual schema definition language which is based on the binary-relational view of the universe.[3] The language, called SCHEMAL, contains a set of concepts which corresponds to a subset of first-order predicate calculus. A characteristic of SCHEMAL is that the consistency of schemas written in it can be determined. An example of a SCHEMAL statement is:

$$(x \cdot \text{is} \cdot w) \Leftarrow [(x \cdot \text{colour} \cdot y) \wedge (x \cdot \text{weight} \cdot z) \wedge (w \cdot \text{colour} \cdot y) \wedge (w \cdot \text{weight} \cdot z)]$$

This states that the colour and weight of an entity uniquely identify it, in the database in question.

Current work on ASDAS includes the development of automatic means of generating procedures from SCHEMAL statements which maintain database consistency. The ultimate aim of the ASDAS project is to produce a database management system which can be used by someone with little or no experience of computing. It is believed that use of a binary-relational storage structure will facilitate the achievement of this aim.

REVIEW OF IMPLEMENTATIONS AND SUGGESTIONS FOR ALTERNATIVE APPROACHES

The simplest implementations are those of Titman[8] and Munz.[26] In these systems the triples are held in ordered arrays: one array for each relation. This scheme is simple, extendible and storage efficient. However, it suffers from limited and slow retrieval capabilities.

The hashing scheme of LEAP appears to be the most

widely used implementation strategy. Variations are used by Ash and Sibley,[7] Symonds,[19] Crick *et al.*[21] and others. The advantages of this method are that all seven SAFs are implemented with more or less equal efficiency and new relations can be added easily. Disadvantages are (i) data is replicated, (ii) it is not simple to program and (iii) the method inherits all of the problems associated with conventional hashing: collision resolution, difficulties with deletion and the need to estimate the maximum hash table size before use.

The linked-list structure of NDB overcomes the need to replicate data somewhat, and Sharman and Winterbottom[25] estimate that an NDB database is likely to require less total storage than the corresponding conventional realization. The major disadvantage of the structure is that retrieval of a triple is likely to be quite slow. (In fact only two of the seven SAFs are implemented at all efficiently). Although the physical structure of an NDB database is automatically re-organized in order to cluster elements according to their access patterns, a typical retrieval is likely to require several disc transfers.

The inverted list structure currently used by the FACT machine also avoids replication of the triples. However, the saving in space is partially offset by the space required by the lists. All seven SAFs are implemented, though not with equal efficiency. Retrievals of type (x . ? . ?) are processed faster than retrievals of type (x . y . ?). This is because the former requires only one inverted list to be retrieved, whereas the latter requires two lists to be retrieved and merged to give a list of addresses of required triples. These triples are not necessarily held together on backing store and often require several disc accesses to be fetched.

The dynamic hashing scheme of ASDAS would appear to be the fastest software implementation of a binary-relational structure. A single triple requires only one access to backing store, and to retrieve N triples with equal key requires only ROUND $((N/B) + 1)$ accesses, where B is the number of triples per block. This is because triples with equal key are held in blocks which are chained together. However, the ASDAS structure is extremely inefficient in terms of storage space. Six copies of the triples are maintained and this also leads to slow triple insertion.

In all of the implementation strategies reviewed so far, retrieval requests of the form 'list all people aged over 18' may only be processed by issuing the retrieval (? . aged . ?) and subsequently examining the object field of each triple returned to see if it contains a value greater than 18. However, if the LEECH processor were used with the FACT machine, as has been proposed, then this type of 'range' retrieval could be executed as fast as requests of the form (? . aged . 18). Thomson[35] has estimated that the LEECH processor could scan data at a rate of about 178 megabytes per second. Assuming that a sufficiently fast backing store were available, it would take about one tenth of a second to scan a database containing one million 16 byte triples. This is the same order of magnitude as the retrieval time to be expected with ASDAS. The problem with using LEECH, or any other sequential scan content addressable device such as CAFS[36] is that retrieval time tends to be linearly proportional to database size. The situation can be improved by using several such devices in parallel.

An alternative approach, for implementing a binary-relational structure, is to develop one of the other ideas used in the FACT machine. Instead of building a hardware matrix for the set-membership relation alone, a separate matrix could be used to represent each binary-relation in the database. This approach would result in very fast retrieval and would simplify the processing of multi-attribute searching. However, at present, the cost of such a structure, even for a small database, is likely to be enormous.

CONCLUDING COMMENTS

We began by introducing the binary-relational view of the universe and followed this by giving a definition of a binary-relational storage structure. We then discussed the relative merits of using such a structure as the storage mechanism in database management systems. We continued with a survey of the few systems which use binary-relational structures, and then completed the study by reviewing the implementation strategies used by these systems. Our conclusions are as follows.

(i) Use of a binary-relational structure may simplify system design and improve data-independence in some circumstances.
(ii) Use of a binary-relational structure might be inappropriate if there are many multi-attribute retrieval requests, and/or batch processing of records is required frequently. Note that batch processing of triples from a single binary-relation could be very fast. It is the batch processing of records that brings little benefit.
(iii) Assuming that the major part of the database resides on disc, then the fastest software implementation currently in use has a retrieval time equivalent to one disc transfer.
(iv) Economically viable sequential scan, content addressable devices are, currently, no faster than the best software implementations for databases greater than one million triples.

Our overall impression is that binary-relational storage structures are easy to integrate into an overall system design, and provide a high degree of flexibility and modifiability. Their use, however, is limited by their inappropriateness for multi-attribute retrieval and batch processing of records, and by the fact that current implementations are slow and/or space inefficient. We believe that an improvement in performance will only be possible through the design of special-purpose hardware such as that currently being considered for use with the FACT machine.

REFERENCES

1. M. J. R. Shave (1981) Entities, functions and binary relations: steps to a conceptual scheme. *The Computer Journal* **24** (No. 1), 42–45 (1981).
2. R. Rock-Evans, Data analysis. *Computer Weekly*, IPC Electrical-Electronic Press Ltd (1980).
3. R. A. Frost, A. D. McGettrick and R. K. Welham. The simplified binary-relational view of the universe and a conceptual schema definition language based on it. Department of Computer Science, Strathclyde university, internal report (Oct. 1981).
4. J. A. Feldman, Aspects of associative processing, Technical note 1965–13, MIT Lincoln Laboratory, Lexington, Massachusetts (1965).
5. R. E. Levien and M. E. Maron, A computer system for inference execution and data retrieval. *Communications of the ACM* **10** (No. 11), 715–721 (1967).
6. J. A. Feldman and P. D. Rovner, An ALGOL-based associative language. *Communications of the ACM* **12** (No. 8), 439–449 (1969).
7. W. L. Ash and E. H. Sibley, TRAMP: an interpretive associative processor with deductive capabilities. *Proceedings of the ACM 23rd National Conference*, pp. 144–156. Brandon/Systems Press, Princeton, New Jersey (1968).
8. P. Titman, An experimental database system using binary relations in *Data Base Management, Proceedings of the IFIP-TC-2 Working Conference, Cargese, Corsica,* ed. by Klimbie and Koffeman, pp. 351–361. North-Holland, Amsterdam (1974).
9. C. I. Johnson, Interactive graphics in data processing: Principles of interactive Systems, *IBM Systems Journal* **7** (No. 3/4), 147–173 (1968).
10. J. Ravin and M. Schatzoff, An interactive graphics system for analysis of business decisions, *IBM Systems Journal* **12** (No. 3), 238–256 (1973).
11. R. A. Frost, A simple database system aimed at the naive user. *Proceedings of 6th ACM European Regional Conference on Systems Architecture*, pp. 234–240, IPC Business Press Ltd, London (Feb. 1981).
12. J. Martin, *Computer Database Organisation*, p. 403. Prentice-Hall, Englewood Cliffs, New Jersey (1975).
13. W. Kent, New criteria for the conceptual model, in *Systems for Large Data bases*, ed. by P. C. Lockemann and E. J. Newhold. North-Holland, Amsterdam (1976).
14. L. A. Belady, M. W. Blasgen, C. J. Evangelisti and R. D. Tennison, A computer graphic system for block diagram problems. *IBM Systems Journal* **10** (No. 2), 143–161 (1971).
15. J. Gips, A syntax-directed program that performs a three-dimensional perceptual task. *Pattern Recognition* **6**, (No. 3/4), 189–199 (1974).
16. F. Hayes-Roth, Representation of structured events and efficient procedures for their recognition. *Pattern Recognition* **8** (No. 3), 141–150 (1976).
17. M. E. Senko, Data structures and data accessing in database systems past, present, future. *IBM Systems Journal* **16** (No. 3), 208–257 (1977).
18. C. J. Date, *An Introduction to Database Systems*, 2nd Edn, pp. 191–201. Addison-Wesley, Reading, Massachusetts (1977).
19. A. J. Symonds, Auxiliary-storage associative data structure for PL/1. *IBM Systems Journal* **7** (No. 3/4), 229–245 (1968).
20. T. E. Johnson, *Mass storage Relational Data Structure for Computer Graphics and Other Arbitrary Data Stores.* MIT, Department of Architecture report, Cambridge, Massachusetts (1967).
21. M. F. Crick and A. J. Symonds, *A Software Associative Memory for Complex Data Structures.* IBM Technical report G320-2060 (1970).
22. R. A. Lorie and A. J. Symonds, A relational access method for interactive applications, in *Database Systems, Courant Computer Science Symposia*, Vol. 6. Prentice-Hall, Englewood Cliffs, New Jersey (1971).
23. R. A. Lorie, *XRM—an Extended (n-ary) Relational Memory*, Report No. G320–2096. IBM Cambridge Scientific Center, Cambridge, Massachusetts (1974).
24. S. Todd, The Peterlee relational test vehicle—a system overview. *IBM Systems Journal* **15** (No. 4), 285–307 (1976).
25. G. O. H. Sharman and N. Winterbottom, *NDB: Non-Programmer Database Facility*, IBM Technical Report TR 12.179. IBM UK Laboratories Ltd., Hursley Park, Winchester SO21 2JN, UK (1979).
26. R. Munz, The WELL System: A multi-user database system based on binary-relationships and graph-pattern-matching. *Information Systems* **3** (No. 2), 99–115 (1978).
27. I. Futo, F. Daruas and P. Szeredi, Application of PROLOG to development of QA and DBM systems in *Logic and Data Bases*, ed. by H. Gallaire and J. Minker. Plenum Press, New York (1977).
28. D. R. McGregor and J. R. Malone, The FACT Database System, *Proceedings of Symposium on Research and Development in Information Retrieval, Cambridge.* Butterworths, Sevenoaks, Kent (1980).
29. D. R. McGregor, R. G. Thomson and W. N. Dawson, High performance hardware for database systems, in *Systems for Large Databases*, ed. by P. C. Lockemann and E. J. Newhold. North-Holland, Amsterdam (1976).
30. J. R. Malone, Personal communication (1981).
31. M. L. Blake, Towards routine machine state-of-the-art reviewing: condensations of bibliographic search output. *Information 6.* Oxford (1981).
32. J. A. Mariani and D. R. McGregor, AutoProg—A Software Development and Maintenance System. *The IUCC Bulletin* **3** (No. 1) (1981).
33. P. Larson, Dynamic Hashing, BIT **18**, 184–201 (1978).
34. J. D. Warnier, *Logical Construction of Programs.* H. E. Stanfert Kroese BV, POB 33 Leiden, Netherlands. (1974).
35. R. G. Thomson, *A Special Purpose Processor for Database Systems.* PhD Thesis, Strathclyde University, Scotland (1977).
36. V. A. J. Maller, The content addressable file store—CAFS. *ICL Technical Journal* **1** (No. 3), 265–279 (1979).

Received August 1981