The relational model as a basis for document retrieval system design

Ian A. Macleod

Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada K7L 3N6

Document retrieval systems have existed for some time. The architecture of these systems tends to reflect the philosophy of the original applications for which they were designed and lack the generality typical of data base management systems. A relatively recent development in DBMS design has been the evolution of the relational model. Unlike tree and network models, this approach appears to be of significant interest as the basis for the design of a document retrieval system. This paper discusses the applicability of the relational model to the design of retrieval systems.

(Received October 1980)

1. Introduction

It is obvious that there is an increasing demand for suitable tools with which document retrieval systems can be constructed. One of the most significant developments in computing in recent years has been the prolific growth in small computer systems in departments and businesses which previously had no access to a local computing facility. At the same time the advent of word processing systems has resulted in a greater availability of machine readable data, making compilation of 'private' data bases for small libraries and reference collections more attractive.

Existing software is not generally applicable to these types of data bases and the costs of generating personalised software packages are often prohibitive. The common practice has been to design large and relatively inflexible systems for document retrieval applications. In many cases the creation of these systems was inspired by the need to obtain information from a particular data base and this motivation is often reflected in the query languages and search strategies provided. Examples of such systems are MEDLINE (National Library of Medicine, undated) and ORBIT (Seiden, undated). Other systems such as STAIRS (IBM, 1972) and SPIRES (University of Stanford, 1972) have been designed for more general applications, but here again the design reflects a particular view as to how data should be stored and retrieved.

Our main purpose is to develop a methodology for the design of retrieval systems in order to facilitate implementations which will be more powerful and flexible than existing systems. This paper is based upon the premise that the somewhat ad hoc approach to information retrieval, which has been prevalent in the past, is unsatisfactory in the light of present day needs. It is our belief that a more productive approach towards the development of a document retrieval system (DRS) is through the use of a formal data model. The inflexibility of current retrieval systems largely derives from the fact that they are based upon specially designed file organisations. This frequently means that a considerable portion of the implementation effort is spent on building variations of existing software. The file systems developed often tend to be biased towards the original retrieval philosophy and cannot easily be modified to reflect new concepts and ideas. Updating problems such as modifying incorrect documents and indexes are frequently ignored.

Despite the fact that there is very high degree of overlap in the functional requirements of document retrieval and data base management systems, they have largely been developed in isolation from one another. A consequence of this unfortunate circumstance is the characteristic data dependence of DRS. At the same time, DBMS have largely ignored the problems of conditionally retrieving textual data, as the developers of office information systems are now discovering.

A DRS is usually oriented around a few very simple structures, namely dictionaries. On the other hand, DBMS based on tree and network models are designed for use with information which is generally much more structurally complex than the type of bibliographic data normally associated with information retrieval systems. A further characteristic of most DBMS is that the data sublanguages associated with them are influenced by the physical structure of the data base and also tend to be highly procedural. Thus these systems are unnatural to use and have a great deal of unnecessary overhead in the context of bibliographic data and are particularly impractical for a minicomputer implementation. [One attempt to utilise an existing DBMS system for document retrieval is FIRST (Dattola, 1979) which uses a network model via an APL interface.] A relatively recent development in DBMS design has been the development of the relational model (Codd, 1970). Many of the structures implicit in retrieval systems, such as term–document matrices, synonym dictionaries and thesauri, are already in a form suitable for representation in a relational data base. Thus a relational model forms a natural way of viewing this type of data. Furthermore, relational query languages have been developed which are conceptually simple yet extremely powerful. This contrasts with the current state of DRS query languages. These latter tend to be extremely inflexible and not particularly powerful.

It seems apparent that there is a need for a new approach to document retrieval system design. The appealing physical characteristics of the relational model, coupled with the power of the associated query languages led us to investigate the appropriateness of document retrieval system design based on a relational view of data. A major aim of this paper is to show that the gap between information retrieval (IR) systems and DBMS can effectively be bridged by developing a methodology for IR system design based on the relational model.

2. Appropriateness of the relational model

The appropriateness of the relational model can be reasonably evaluated in terms of three basic parameters. First, there is the success, or otherwise, of the model in handling current approaches to document retrieval. Second, there is the ease with which systems may be constructed from a basic data base of documents. Third, the efficiency of the model needs to be determined. This is not to say that other aspects of a total system, such as the user interface, are unimportant. However, these 'accessory' capabilities are easily implementable on top of a basic system and need not be fundamental to the basic design.

CCC-0010-4620/81/0024-0312 $02.00

2.1 *Retrieval methods*

There are three main DRS models which have been suggested. These are the Boolean systems based on controlled vocabularies, weighted search systems (or probabilistic models) based on uncontrolled vocabularies and clustered systems based on automatic classification of documents. MEDLINE is an example of the first and STAIRS an example of the second. In the following sections, we shall show how the structures associated with each of these types of system can be represented in the relational model. Cluster based systems are purely experimental at the present time, but we will show how such systems are also potentially suited for representation in the model. A reasonable familiarity with SEQUEL is assumed in the following examples. Apart from the use of some non-standard functions (SEQUEL has the capability to include such functions from a library) and one proposed syntactic extension, the examples below are in the SEQUEL syntax defined by Chamberlin *et al.* (1976).

In the remainder of this paper we will describe a relation by its name followed by a parenthesised list of the names of the fields (or attributes) it contains. Thus a relation called DOCUMENTS representing a set of documents, where each document consists of a document number, an author name, a title and an abstract, can be represented as

```
DOCUMENTS (D#, AUTHOR, TITLE, ABSTRACT)
```

Boolean models. Most current retrieval systems are based on Boolean searching with a controlled vocabulary. Typically, in addition to the documents themselves, two other main structures are involved, namely a term dictionary and a structure illustrating hierarchical relationships among terms. These structures can be represented by the relations:

```
DICTIONARY (TERM, D#)
THESAURUS (TERM, SUB-TERM)
```

A typical Boolean search then might be the following, which is a request for documents indexed under both the terms '*water*' and '*pollution*'

```
SELECT D#
       FROM INDEX
       WHERE TERM = 'water'
INTERSECT
SELECT D#
       FROM INDEX
       WHERE TERM = 'pollution'
```

A slightly more complex retrieval request is the following, which retrieves the documents indexed by all the terms subordinate to '*aquatic*'.

```
SELECT D# FROM INDEX
       WHERE TERM IN
       SELECT SUB-TERM
              FROM THESAURUS
              WHERE TERM = 'aquatic'
```

It is readily apparent that arbitrarily complex requests can easily be constructed and that additional relations (such as 'see also' relationships) can also be readily incorporated.

Probabilistic models. In probabilistic models, uncontrolled vocabularies with weighted indexes are normally employed. Also lists of words not to be used as index terms are often employed. Thus in addition to the document relation, we might have the two relations:

```
INDEX (D#, TERM, WEIGHT)
STOP-LIST (TERM)
```

A typical search request is the following to find all documents related to the topics '*run-off*', '*water*' and '*pollution*'.

```
SELECT D#, RELEVANCE(*)
       FROM
       SELECT D#, WEIGHT
              FROM INDEX
              WHERE TERM IN
              ('run-off', 'water', 'pollution')
       GROUP BY D#
       ORDER BY RELEVANCE DESCENDING
```

The function RELEVANCE is not a standard SEQUEL function. It is a library routine which takes the weights for each document and combines them to produce a single relevance measurement for the document.

Another type of retrieval operation which is applicable to a probabilistic model is a feedback search which is basically a request to find all documents similar to some other retrieved set of documents. If RESULT is the already retrieved set of documents, this operation can be expressed as

```
SELECT D#, RELEVANCE(*)
       FROM INDEX
       WHERE TERM IN
             SELECT TERM
                    FROM RESULT, INDEX
                    WHERE RESULT.D# =
                    INDEX.D#
       AND TERM NOT IN STOP-LIST
       GROUP BY D#
       ORDER BY RELEVANCE DESCENDING
```

Here the result is joined with the index to get the set of index terms used to index all the documents previously retrieved. This set is then used as the new query.

Cluster based models. A cluster based retrieval system basically consists of a group of clusters where a cluster is a set of related documents. Each group is described by a set of index terms in much the same way as a document is in a probabilistic model. A number of cluster based models have been suggested, see for example Salton (1971) and van Rijsbergen (1970). Here we will assume that the objective of a search is to retrieve the most relevant cluster. A cluster model can be represented by the relations:

```
CLUSTERS (C#, D#)
DESCRIPTION (TERM, WEIGHT, C#)
```

The first relation is the set of clusters with each cluster identified by a unique number. The second relation represents the centroids. Then a query to find the cluster most related to the query '*run-off, water* and *pollution*' could be expressed as

```
SELECT C#, MAX (RELEVANCE)
       FROM
       SELECT C#, RELEVANCE(*)
              FROM
              SELECT C#, WEIGHT
                     FROM DESCRIPTION
                     WHERE TERM IN
                     ('run-off', 'water', 'pollution')
              GROUP BY C#
```

Here MAX is a standard SEQUEL set function which returns the tuple with the maximum value of the specified attribute.

2.2 *Dictionary construction*

Given a 'clean' document data base, the main requirement in building a retrieval system is to create indexes or dictionaries for the document collection. Dictionary construction requires access to individual words in sections of text. Basic relational models provide no such string processing capability. A technique which seems to be compatible with the basic relational philosophy is to decompose individual attributes into groups of tuples using the converse of the GROUP clause of SEQUEL. A suggested syntax, compatible with the general SEQUEL syntax, could be

SPLIT BY *function*(*attribute*) HAVING *boolean*

Here the *function* will be the name of a library procedure which splits each tuple into a set of tuples by processing the *attribute* passed as a parameter in some way. The names of the attributes created are assigned by the new attribute names in the SELECT clause.

For example, suppose we want to create a weighted index from the DOCUMENTS relation, using the individual words of the abstracts as a basis for the index, and the function TERMS is written to extract from each tuple a set of tuples consisting of the document number, a term from the abstract. This function could then be utilised thus

```
ASSIGN TO BASIC-INDEX (D#, TERM):
        SELECT UNIQUE D#, TERM
                FROM DATABASE
                SPLIT BY TERMS (ABSTRACT)
                HAVING TERM NOT IN STOPLIST
```

From this basic index, a number of weighted indexes can then be easily constructed using normal SEQUEL constructs. For example, a common method uses weighting based on inverse document frequency of terms. Such an index could be built as follows

```
ASSIGN TO COUNT-INDEX (TERM, COUNT)
        SELECT TERM, COUNT(*)
                FROM BASIC-INDEX
                GROUP BY TERM
```

This operation provides a list of terms and the number of documents in which they occur.

```
ASSIGN TO INDEX (TERM, D#, WEIGHT):
        SELECT TERM, D#, 1/COUNT
        FROM BASIC-INDEX, COUNT-INDEX
        WHERE BASIC-INDEX.TERM = COUNT-
        INDEX.TERM
```

This operation joins the basic index with the count of the index terms to produce the final result.

Obviously dictionary construction is often most efficiently carried out in conventional programming language. SEQUEL has a data sublanguage facility designed for embedding in a host language and an equivalent to the SPLIT operation can easily be defined in this context as well as in the higher level environment in which it appears here.

2.3 *Efficiency*

The simplest way of viewing a relation is simply as a sequential file of records and this is the way in fact that relations are sometimes organised. Obviously, this will provide very slow direct access to particular attribute values, and in practice attributes which are accessed frequently normally have associated with them an index. (Note here that by index we mean an internal structure used to provide some sort of direct access capability as opposed to a document index.) In SEQUEL, indexes can be created explicitly at any time. It should be noted that the presence or absence of an index for a particular attribute does not affect the capability of the language, only its performance.

A problem with the relational model in a general environment is that it apparently can be quite inefficient. This potential inefficiency derives from the fact that all linkages between relations are derived dynamically (for example by 'join' operations). This contrasts with other models where it is more common to represent logical relationships explicitly in the structure of the data base specified by the schema. However, it should be noted that work by Date has shown that exactly the same basic file organisation can be used to represent all three data models (Date, 1980). In this environment he has found that the relational model actually required fewer disc accesses than the other two models although it was significantly more expensive in terms of storage requirements.

Currently we have successfully implemented a relational system (Macleod, 1980). Relations are represented by sequential files of tuples and indexes may be dynamically created at any time. Indexes are constructed from a static multinode tree. Updating causes the index to progressively degenerate. An alternative technique is to use a dynamic tree index based on an organisation such as B-trees (Knuth, 1973). The advantage of the static tree is that it is less expensive to build and maintain but on the other hand is less satisfactory if a large number of on-line updates are performed. Our own choice was based on the hypothesis that on-line updates of an indexed attribute are relatively infrequent in a document retrieval system. Held and Stonebraker (1978) and Stonebraker (1980) give arguments for and against dynamic tree indexes.

Given a tree index into a file of tuples, the operations which would be needed to implement the simple Boolean search shown above would be:

(1) select D# from all tuples in INDEX where the value of the TERM attribute is '*water*' (that is, look up '*water*' in the TERM index for INDEX,);

(2) select the D#s for the '*pollution*' tuples;

(3) sort both results;

(4) form the intersection of the two sets.

It can be seen at once that this is precisely the same sequence of operations as would be carried out in a special purpose system. The same observations can be made of all the other examples. Thus it is apparent that no automatic decrease in efficiency derives from the use of the more general relational model.

3. Problems with the relational model

While this paper is presented in support of the relational model, it would be naive to say that such an approach is necessarily a panacea for all the potential shortcomings in retrieval system design. There are a number of potential problem areas with relational organisations.

3.1 *Generality*

The very generality of the relational model may impose severe management problems. It may be necessary in practice to impose constraints on how a user can access relations. For example, a join operation over unindexed attributes of two large relations may not be desirable.

3.2 *Syntax*

While a SEQUEL type language is readily usable by computer scientists, this certainly will not be generally the case. We have proposed elsewhere modifications to SEQUEL, mainly via a macro capability, which would make it more appropriate for general users (Macleod, 1979).

3.3. *Inappropriateness*

While we stated earlier that structures in document retrieval are naturally tabular this is not universally true. Tree organisations are not easily adaptable to a 2-dimensional environment. Thus the EXPLODE feature of MEDLINE which derives all the subordinates of a term in an hierarchical index, cannot be naturally stated in a single SEQUEL statement. However, the more general operations of moving up or down a tree one level at a time can be easily stated.

3.4 *Semantics*

There are a large number of semantic problems in practical data bases. These are not peculiar to the relational model but nor are they solved by it. McLeod (1978) gives a comprehensive listing of these. However, again because of the simplicity of the environment, semantic problems do not seem to be severe in document retrieval. For example it can be seen that every relation we have given is naturally in fourth normal form.

4. Conclusions

Despite the possible problems noted above, we feel that it is apparent that any shortcomings of the relational model are far outweighed by its potential advantages and it certainly deserves consideration as a practical model for the design and implementation of document retrieval systems. One of the most serious problems in DRS research at the present time is the lack of a suitable framework within which to carry out experiments. Proponents of new ideas often find themselves in the position where they virtually have to build a new retrieval system from scratch in order to obtain some practical results. This has been, and continues to be, a major obstacle in the effective evaluation of new techniques and is probably a major reason why so few experimental ideas ever carry over into practical systems. Hopefully our examples above show how easily retrieval systems based on quite different concepts can be represented naturally within a relational model. One justification of a relational system therefore is as an experimental test bed within which to evaluate new techniques.

In a general system, response time for queries will always be somewhat slower than those of a special purpose document retrieval system. However, as we have shown above, retrieval in conventional systems is no different from basic retrieval in a relational system. Thus there is no fundamental reason why practical working systems cannot be developed using the relational model. In return for some performance sacrifice, access will be gained to a much more powerful system. Yet at the same time we have a very simple basic system. As our discussion on the representation of relations shows, a relational file organisation can be identical to that currently used in many retrieval systems, yet a flexibility and adaptability can be provided which we have not seen demonstrated on any existing conventional system.

Acknowledgement

This work was partially supported by a grant from the National Science and Engineering Research Council of Canada.

References

Chamberlin D. D. *et al.* (1976). SEQUEL 2: unified approach to data definition, manipulation, and control, *IBM Journal of Research and Development*, Vol. 20, pp. 560–575.

Codd, E. F. (1970). A relational model of data for large shared data banks, *Communications of the ACM*, Vol. 13, pp. 377–387.

Date, C. J. (1980). An introduction to the unified database language, *Proceedings of the Sixth International Conference on Very Large Data bases*, Montreal, pp. 15–29.

Dattola, R. T. (1979). FIRST: flexible information retrieval system for text, *Journal of the American Society for Information Science*, Vol. 30, pp. 10–14.

Held, G. and Stonebraker, M. (1978). B-trees re-examined, *Communications of the ACM*, Vol. 21, pp. 139–143.

IBM (1972). *Storage and Information Retrieval System* (*STAIRS*) *Program Reference Manual*, Form No. 5734-XR3. IBM Corporation, New York.

Knuth, D. E. (1973). *The Art of Computer Programming: Searching and Sorting*, Vol. 3, pp. 473–480, Addison-Wesley, Reading, Mass.

Macleod, I. A. (1979). SEQUEL as a language for document retrieval, *Journal of the American Society for Information Science*, Vol. 30, pp. 243–249.

Macleod, I. A. (1980). The Mistral/11 retrieval system, Technical Report, Department of Computing and Information Science, Queen's University, Ontario.

McLeod, D. (1978). A Semantic Data Base Model and its Associated Structured User Interface, Technical Report MIT/LCS/TR-214, Laboratory for Computer Science, Massachusetts Institute of Technology.

National Library of Medicine (undated). *MEDLINE Reference Manual.* Bibliographic Services Division, National Library of Medicine.

Rijsbergen, C. J. Van (1970). A clustering algorithm, *The Computer Journal*, Vol. 13, pp. 113–115.

Salton, G. (1971). *The SMART Retrieval System—Experiment in Automatic Document Processing*. Prentice-Hall, Englewood Cliffs, NJ.

Seiden, H. R. (undated). *ORBIT System Information.* System Development Corporation, Santa Monica, California.

Stonebraker, M. (1980). Retrospection on a database system, *ACM Transactions on Database Systems*, Vol. 5, pp. 225–240.

University of Stanford (1972). *SPIRES User's Manual.* Systems Documentation Library, Stanford Computation Center, University of Stanford, California.

Book review

Distributed Data Bases, by I. W. Draffan and F. Poole, 1981; 374 pages. (*CUP*, £15·00)

This book is based on an advanced course held at Sheffield in 1979 as part of the CREST series. The text consists of a collection of papers covering many aspects of distributed database systems. The papers illustrate very clearly that distributed database technology covers not just the now familiar database aspects, but a far wider area including such fields as communication networks and query analysis, decomposition and distributed processing. Furthermore, many of the problems and difficulties associated with non-distributed database systems become even more taxing in a distributed environment, for example, integrity, reliability, consistency, access control, performance and recovery mechanisms.

An obvious danger in presenting a series of papers within a single volume is the lack of continuity of discussion from one chapter/paper to the next. Nevertheless, the book covers a wide variety of topics beginning with a paper which places distributed database systems in perspective discussing such aspects as communications links, design philosophies and distribution costs. A later paper discusses the components of a distributed system and identifies many of the areas of database technology made more difficult by distribution. Other areas such as query decomposition, optimisation and distributed processing are also identified and form the subjects of later papers. The book also discusses concurrency control and processing synchronisation, update strategies when data is replicated and recovery procedures. Several papers cover aspects of distributed database administration and control from design considerations to granting access, testing and policing procedures. Finally, as an appendix, a case is made against the CODASYL proposals being adopted as an ANSI standard. The argument is not convincing, but it begs more fundamental questions relating to the usefulness, expected life-span and investments placed on national standards, especially in such a rapidly changing area as computer technology as a whole.

I found the book interesting despite areas of repetition among some of the papers. Those which covered distributed query analysis, optimisation and processing were of particular interest, as were those which discussed design philosophies and distributed architectures. This book should be of interest to those engaged in all aspects of database research as it serves a useful purpose in bringing together a series of papers covering an expanding, highly topical and intellectually demanding field.

D. M. R. Bell (Aberdeen)