TOWARDS INTEGRATED INFORMATION SYSTEMS

I. A. Macleod
Associate Professor
Department of Computing & Information Science
Queen's University
Kingston, Ontario K7L 3N6

ABSTRACT

Data Base Management Systems, (DBMS), and Document Retrieval Systems, (DRS), have tended to evolve separately for both historical and functional reasons. This has resulted in systems which are somewhat less general than they might be. With the accessibility of software systems to an ever increasing range of users, there is a growing awareness that this situation should be amended. This paper outlines the advantages and disadvantages of implementing a DRS within a DBMS environment and examines some of the problems which must be resolved before we can effectively integrate both types of information system.

VERS DES SYSTEMES D'INFORMATION INTEGRES

RESUME

Les systèmes de gestion des bases de données (SGBD) et les systèmes de repérage de l'information (SRI) ont été développés indépendamment l'un de l'autre et ce, pour des raisons à la fois historiques et pratiques. Cette dichotomie a toutefois résulté en des systèmes qui sont beaucoup moins universels qu'ils ne devraient l'être. Avec une accessibilité accrue des systèmes de logiciels à toutes sortes d'usagers, l'on se rend compte de plus en plus que cette situation doit être corrigée. L'auteur présente les avantages et les inconvénients de l'utilisation d'un SRI dans un environnement de SGBD et il examine quelques-uns des problèmes qu'il faudra résoudre avant de pouvoir intégrer de façon efficace les deux types de systèmes d'information.

INTEGRATED INFORMATION SYSTEMS

INTRODUCTION

In general, data bases consist of collections of different types of records. One of the major tasks of a Data Base Management System, (or DBMS), is to maintain relationships among the different types. To this end, various models have been developed and each DBMS is constrained to a particular model. The development of these models has been influenced by the extent to which they reflect reality although they all tend to be somewhat artificial. At the present time there are three major models, hierarchic, network and relational. Respectively, these see the world organised in terms of trees, graphs and tables. In setting up a data base for processing within a DBMS based on one of these models, it is necessary to describe the relationships among the various record types so that they correspond to the underlying model. This description is called the schema. As the models correspond to particular types of data structure, languages have developed which are suited to the particular underlying data structures. Such languages are called data sublanguages. Sublanguage operations required to process one data base are often quite different from those required to search another based on a different model. Sublanguages are normally embedded within a host language and are used procedurally. At a higher non-procedural level, we have query languages which, in principle at least, can be less influenced by the data model and in some cases approximate to natural language queries.

Retrieval in a DBMS tends to be deterministic. That is, a record is accessed only if a particular field contains a specific value. Further, an important function of a DBMS is the ability to access records of different types where the relationship between two types might be dependent on the relation of the values of two fields. The problem in a DBMS environment is not how to retrieve the information in a particular record, but rather to extract information that may be scattered across a number of different record types. For example, we might have in a data base files the following three record types:

AUTHOR, ADDRESS;
AUTHOR, TITLE;
KEYWORD, TITLE;

We may wish to obtain information such as: "all the addresses of authors working in a an area described by the following keywords". Here the information can only be found if the various records involved have been, or can be, connected together in some way. This is one of the functions of the schema. The "art" of retrieving in a DBMS then involves using the connections defined in the schema to bring together diverse pieces of information.

Present day DRS, on the other hand, generally handle records of a single type. Since there is only one user generated record type, there is no true concept of a schema in a DRS. At least one field, and sometimes this is the only field, contains free text. A most important part of a DRS is the index. This may be created from manually supplied

keywords, in which case retrieval tends to be deterministic, or it may be automatically supplied from the free text by some sort of indexing program. In the latter case, index terms are often weighted with the weights intended to measure the likelihood of a particular term applying to a particular record or document. Retrieval in this type of environment tends to be probabilistic and the retrieved output is normally ranked according to its predicted correspondence to the user query. However it should be noted that the first type of retrieval environment is not really deterministic since it is generally impossible to exactly specify in terms of indexes what a document is "about". Queries are much less precise and are generally concerned with retrieving individual records whose textual content is "about" a topic in which the retriever is interested. The art here is in describing the topic.

Because of the way in which users retrieve information from a DRS and because the information is usually quite limited, no need has been foreseen for constructing a DRS around any particular model. Rather a data structure applicable for a specific data organisation is built and a quite rigid query language is developed to access this data structure. Because of the highly iterative way in which queries are developed in response to partial information, a great deal of effort is often expended on optimising response times at the expense of data independence. In particular, the main index is often closely integrated with the actual documents of the data base. Some systems, notably STAIRS, (IBM, 1981), permit the data base administrator to run a utility package to generate indexes. More commonly, systems assume that indexes have been generated externally prior to data base creation. Because of the rigid nature of the underlying data structure, indexes must adhere to a rigidly predefined format so that it is not easily possible to allow users to construct their own index building programs within the system.

Some DRS systems provide a limited facility for handling structured data. STAIRS, for example, allows fixed format document attributes such as acquisition date, origin, etc., to be kept. However STAIRS differentiates very strongly between formatted fields and free text. Each is stored differently and each is accessed using a different set of commands. It is this lack of integration that has led to the separate development of DRS and DBMS and it is the basic problem to be solved if a satisfactory unified system is ever to be evolved.

SUITABILITY OF THE DBMS APPROACH

If there is ever to be an effective integration of DRS and DBMS, then DRS are going to have to be developed around some model which will also be appropriate for DBMS activities. At the same time, for this to occur, there is going to have to be some demonstrable advantage for DRS in following this path. In this section we will look at what advantages there might be.

We can identify the following major characteristics of a DBMS:

A. An underlying set of data structures;
This contrasts with the DRS situation where there is a single monolithic data structure. The advantage here is that new data and relationships can be added to an existing data base in a straightforward manner.

B. A data model;
The data model describes how the data structures are to be viewed and provides a great deal of data independence. It effectively means that the implementation of the language interfaces can be considered in isolation from the actual physical implementation of the data structures.

C. A schema;
Because the schema serves as the interface between the data base and the query language, it is possible in a DBMS to have multiple views of the same data. Individual users or groups of users, can have their own schema as long as it can be mapped into the basic schema. They can be restricted to certain fields of records and to certain types of records. Access to records may even depend on the values of certain fields of the records.

D. A data sublanguage;
The data sublanguage allows the data base to be accessed from a procedural language. DRS are usually stand alone systems. That is, they cannot be accessed from any other system. In particular there is no interface to a procedural language to permit highly specialised processing operations. Note that within the DBMS context index creation could be handled quite cleanly by emplying a data sublanguage.

E. A query language;
Query languages in the DBMS context, while more complex than their DRS counterparts, certainly are more exhaustive in scope. Most, if not all DRS, somewhat arbitrarily restrict the information that is potentially available. Furthermore, the languages themselves tend to be designed in a somewhat ad hoc way. There may, for example, be quite different commands for displaying the contents of a dictionary file and the contents of a document file.

F. Updating;
DBMS provide comprehensive update capabilities. Updating is treated as a typical operation in DBMS. It simply is not practical to regenerate the data base every time some relatively minor view of it changes. This contrasts with the DRS situation where updating is not normally permitted other than through a regeneration of the data base.

DISADVANTAGES OF THE DBMS APPROACH

On the other hand there are certain disadvantages which might be seen in the use of DBMS for the implementation of document retrieval systems.

A. Inappropriateness of the model;
This is the single most important disadvantage and is discussed in detail in the next section.

B. Complexity;
The price of flexibility is a complex user interface and a large piece of software which is both expensive to acquire and to maintain. As we noted earlier, most people use a DRS in a highly iterative way employing a simple easy to use query language. While this language may be severely limited in its capabilities, there is no doubt that many users of retrieval systems are relatively happy with this situation.

C. Lack of features to handle unformatted text;
This is one important respect in which DBMS are generally less flexible than DRS. This is in string handling. Most DRS permit some limited form of pattern matching to allow searches for partially specified strings. This is particularly necessary when working with natural language. Because words are untidy pieces of data in that they get mis-spelt, sometimes have more than one spelling, have grammatical variants and so on, it is normally the practice for a DRS to provide some sort of simple string matching capability.

It is sometimes claimed that it is not appropriate to implement a DRS within a DBMS. With the exception of systems such as SPIRES, the designers of current DRS do not seem to have foreseen the possibility that individuals, or small groups of individuals, would wish to build and maintain their own data bases. The vast majority of current users of retrieval systems use both a system and data provided by some external agency. For all practical purposes, local retrieval systems are virtually non-existent. It is probably for this reason that no need has been seen for suffering the overhead of providing sublanguage and schema facilities. Note that there is a "chicken and egg" situation here. It could be said that DRS applications do not make use of such facilities because they don't exist, given that existing DBMS lack features which are essential for a DRS. However with the increasing availability of computer technology, it is at least physically easier to enter small data bases than once was the case and often they are available as a fallout of other data handling operations. Furthermore with the now almost universal use of time sharing systems, computer terminals are much more accessible. Individuals now find it convenient to maintain computerised collections of records. The needs of these individuals are not met by existing systems.

In summary, the difficulties in performing document retrieval within a DBMS lie with the general inability of a DBMS to handle strings together with their overall complexity. The latter relates both to the software complexity and to the difficulty of phrasing simple iterative searches in a natural way within a complex query language. If these two main difficulties can somehow be avoided, then it would be much more attractive to implement a DRS on top of a DBMS.

IMPLEMENTATION OF A DRS WITHIN A DBMS

A number of proposals have been made for the integration of document retrieval capabilities into a data base management system. One suggests the network model, (Dattola, 1979), and two others suggest the use of the relational model, (Macleod, 1981), (Schek and Pistor, 1982). Dattola does not appear however to have functionally integrated DRS and DBMS. What he suggests is that a DBMS be used to store and manage documents indexed under very broad categories. More refined retrieval is provided using a separate retrieval system. Retrieval is seen as a two stage process. First, documents are retrieved from the data base using the broad document descriptors. Next a "pure" document retrieval system is used to abstract relevant documents from this retrieved set. While Dattola has used a network model for the original document storage, it does not appear that the network model was chosen for any reason other than availability.

Macleod, (1979), on the other hand, proposes a single integrated approach. He uses the relational model and shows how many DRS operations can be carried out reasonably comfortably within it. The query language used is SQL, (Chamberlin and Boyce, 1974), and some suggestions are made for enhancements to the language in order to improve its suitability as a vehicle for DRS. The most important of these is an extension which would permit literal string values to be replaced by patterns. Another proposed extension is at the cosmetic level where a macro facility is suggested in order to hide some of the less elegant SQL constructs.

A problem with the relational model derives from the concept of normalisation. (Crawford, (1980), provides a more complete discussion of this aspect). A logical collection of data does not necessarily correspond to a single tuple in a relation. For example, a data collection might consist of titles, authors and index terms, where a title may be associated with more than one author and more than one index term. A logical record would take the form:

Title; Author...; Term...;

As it stands, this type of record does not correspond to a tuple in a relation since it contains non-atomic values. The record must first be normalised so, for example, the record:

Title; A1, A2; T1, T2, T3;

would be represented by 6 tuples of the form:

Title; A1; T1;
Title; A2; T1;
Title; A1; T2;
Title; A2; T2;
Title; A1; T3;
Title; A2; T3;

There are still problems with this representation, particularly in updating, caused by the duplication of information within the relation.

A preferable solution is to represent this information by a number of relations, as for example:

```
TITLES   (Document#, Title)
AUTHORS  (Document#, Author)
TERMS    (Document#, Term)
```

So we would have:

```
TITLES:          AUTHORS:        TERMS:
D1; Title        D1; A1;         D1; T1;
                 D1; A2;         D1; T2;
                                 D1; T3;
```

The number of tuples is not increased and in fact they occupy less space. The problem now is that the information has become scattered across three separate relations. The logical relationships among the separate tuples are no longer obvious so that the conceptually simple act of retrieving what the user sees to be a single document may well entail join operations across a number of relations. (This problem can be partially solved by implementing an artificial relationship, sometimes called a view, which allows a group of physically separate relations to be viewed as a single relation. For example, SEQUEL takes this approach.)

Schek too has suggested a relational like approach. However he has made somewhat more radical modifications to the original model. He makes the, not unreasonable, suggestion that the un-normalised view is the natural one. Whereas Macleod has suggested hiding the explicit use of joins by using macros, Schek suggests the retention of the original unnormalised information. In particular he suggests that un-normalised relations be permitted.

For example, suppose we want to list all titles by author A1 described by terms T1 and T2. In SQL we would write:

```
SELECT TITLE
       FROM TITLES, AUTHORS, TERMS X, TERMS Y
       WHERE AUTHOR = "A1"
       AND AUTHORS.D# = TITLES.D#
       AND TITLES.D# = X.D#
       AND X.D# = Y.D#
       AND X.TERM = "T1"
       AND Y.TERM = "T2"
```

The same query in STAIRS would be:

```
SEARCH
1      A1.AUTHORS AND T1.TERMS AND T2.TERMS

BROWSE 1 TITLE
```

In Schek's model it would be:

```
SELECT TITLE
       FROM DATABASE
       WHERE TERMS ⊆ {"T1", "T2"}
       AND AUTHORS ⊆ {"A1"}
```

Of the three queries, the first is clearly the least attractive while there is not a great deal to choose between the other two. The Schek model also handles adjacency of terms. The following query retrieves titles from documents containing T1 followed by T2.

```
SELECT TITLE
       FROM DATABASE
       WHERE TERMS ⊆ {<*, "T1", "T2", *>}
```

Here the angle brackets denote a list of consecutive objects and the "*" is a "match anything" symbol. Schek also provides "nest" and "unnest" operators to transform relation from normalised to unnormalised and vice versa. What effectively he is doing, is providing a mechanism for transforming a set of attribute values into a string and then providing limited string or template matching operations. A separate indexing mechanism is provided for the strings based on fragment indexing, (Schuegraf and Heaps, 1976). In our earlier discussions we noted that STAIRS attempts the integration of text and formatted fields within a single system, with only moderate success. Schek's model represents a much more elegant solution to this problem.

SUMMARY

While some work has been done on the integration of document retrieval and data base management within the same system, much remains to be done. While the relational model has at first glance a number of appealing characteristics in comparison with the other models, there are a number of problems associated with it. Some we have already seen. Others occur when we come to consider the integration of EFS. There is no natural concept of hierarchy within relations. In many collections of data, hierarchies arise naturally. Filing systems are an example of such a system. A more general problem which is not specific to the relational model is that only a single type of record can be retrieved. This is not as bad as the DRS situation where only a single type of document can normally be stored in an active database, and it is not in general a severe restriction in typical data base applications. However, in a conventional office environment it is normal to construct files containing documents of many different types. Documents types might, for example, include letters, forms, resumes, reports, papers, and so on. These different documents may be brought together in related groups, (e.g. envelopes, files, in-trays, stapled bundles). In addition to the external documents, internal documents will be created to describe and manage groupings, (e.g. named file folders, circulation

lists, calendars, envelopes). These internal documents may well become useful documents in their own right and there may be no logical distinction between the two.

Current DRS are typified by systems such as MEDLINE and BRS. These types of retrieval systems seem to be used primarily in conjunction with particular data bases. A few more flexible systems, such as SPIRES, (1974), allow users to build and manage their own data bases easily, but they are not widely available. This is probably one of the main reasons there has been little development into data models suited for textual data. A real need for future DRS development will be the availability in the marketplace of systems incorporating new ideas.

REFERENCES

CHAMBERLIN, D. and BOYCE, R. "SEQUEL: A Structured English Query Language", in Proceedings of the 1974 ACM-SIGMOD Workshop on Data Description, Access and Control. Ann Arbor, Michigan, (May 1974), pp. 249-264.

CRAWFORD, R. G. "The Relational Model in Information Retrieval", in Journal of the American Society for Information Science. Vol. 32, No. 1 (January 1981), pp. 51-64.

IBM. "STAIRS/CMS Terminal User's Guide", SB11-5545, 1981.

MACLEOD, I. A. "SEQUEL as a Language for Document Retrieval" in Journal of the American Society for Information Science. Vol. 30, No. 5 (September 1979), pp. 243-249.

MACLEOD, I. A. "A Data Base Management System for Document Retrieval Applications, Information Systems, Vol. 6, pp. 131-137.

SCHEK, H.J. and PISTOR, P. "Data Structures for an Integrated Data Base Management and Information Retrieval System", in Proceedings of the Eighth International Conference on Very Large Data Bases. Mexico City, (September 1982), pp.197-207.

SCHUEGRAF, E.J. and HEAPS, H.S. "Query Processing in a Retrospective Document Retrieval System that uses Word Fragments as Language Elements", Information Processing and Management, Vol. 12, pp. 283-292.

SPIRES Users Manual, Stanford University, 1974.