RISS—A generalized minicomputer relational data base management system

by DENNIS McLEOD
Massachusetts Institute of Technology
Cambridge, Massachusetts

and

MONTE MELDMAN
Forest Hospital
Des Plaines, Illinois

INTRODUCTION

With the recent growth in popularity of low cost, relatively powerful minicomputers, it is clear that associated data base management systems are required. Nearly all of the minicomputer systems that are currently available commercially have at most rudimentary data base management capabilities. Consequently, in this paper we discuss the design and implementation of a minicomputer relational data base management system: the Relational Inquiry and Storage System (RISS). RISS provides a "naive" user interface, to allow nonprogrammers to deal routinely with a data base without the aid of a programmer, as well as an applications program interface. These interfaces facilitate data base access, modification, and restructure. RISS is a generalized and context-adaptable system, but it maintains a degree of efficiency guaranteeing cost-effective operation.

Minicomputer data base management

In this paper we are concerned with describing effective data base management facilities for computers used by organizations that have small scale needs and a small budget. There are many organizations which have application environments that require data base management capabilities, but which cannot justify a large expenditure of funds on a computer system. Minicomputers go a long way toward solving the hardware cost problem. But, more significantly, it is clear that such an organization cannot afford to develop the required computer system software de novo. Unlike the approach of Taylor and Lloyd¹ (for example), in which a complete minicomputer information system is developed, we will assume that a presupplied operating system and language translator (or translators) will be used. An example of such a system, which we will use in this paper, is the Resource Time-Sharing System (RSTS-11) for the PDP-11 computer supplied by the Digital Equipment Corporation.

In addition, it would be extremely desirable to permit an organization to obtain (e.g., purchase) a generalized data base management system which "fits neatly on top of" the operating system. Whitney² refers to this type of data base management facility as a fourth generation data management system. Some general considerations appropriate to the development of such a system have been reviewed by McLeod.³ Since it is probably not possible to construct a generalized data base management system which can adapt to every aspect of a particular application environment, it is necessary to provide an effective interface between the data base management system and applications programs. But, whenever possible, the data base management system should allow nonprogrammers to deal directly with the data base. A large percentage of a user's needs should be satisfied without the need to consult a programmer. Therefore, it is necessary to develop a very high level (nonprocedural) language to allow the "naive" user to deal effectively directly with the data base.

Approaches to data base management

As pointed out by Codd and Date,⁴ there are two major approaches to data base management:

1. The network approach, as proposed by the CODASYL Data Base Task Group⁵
2. The relational approach, as proposed by Codd,⁶ among others

A continuing controversy exists between the proponents of the network approach (e.g., Bachman⁷) and the proponents of the relational approach (e.g. Codd). A detailed analysis and comparison of these two approaches is presented by Codd and Date.⁸ We believe that the relational approach is best suited to our goals and requirements. In support of the relational approach, Boyce, Chamberlin, King, and Hammer⁹ state that "Traditionally, files are structured to optimize a particular ap-
application program. . . A modern data base management system should be capable of responding to any new unanticipated query in uniform time without requiring a restructuring of the data and without impacting previously written queries." Specifically, for our purposes some of the most relevant advantages of the relational approach over the network approach are:

1. It provides a simpler and more unified user data model, resulting in systems that are easier to use and maintain.
2. It is much more data independent, and consequently results in systems that are more generalized. In addition, relational data bases are easier to alter, e.g., when new data relationship is discovered.
3. It is much easier to express data integrity constraints (limitations on the permissible data in a data base) in a generalized manner.
4. Data retrieval and modification requests are easier to express (in a generalized manner). These requests may be expressed in a way that is much less procedural.
5. The emphasis is on the use of sets (in the mathematical sense, not the CODASYL Data Base Task Group sense), rather than on handling one record at a time.
6. Sharing and protection requirements are more easily satisfied, due primarily to the simplicity of the underlying data base model and absence of highly distributed access paths.
7. Implementation issues are isolated from the logical data base model. This results in increased inter-system compatibility and, most significantly, allows a structured approach to implementation.

THE RELATIONAL APPROACH

Codd introduced the relational model of data "which appears to be the simplest possible data structure consistent with the semantics of information and which provides a maximum degree of data independence." As very concisely stated by Codd: "In the relational approach there exists an interface at which the totality of formatted data in a data base can be viewed as a collection of nonhierarchic relations of assorted degrees defined on a given collection of simple domains (domains whose elements are not decomposable as far as the data base management system is concerned)."

In an attempt to apply some of the recent developments in the area of relational data bases to the minicomputer environment, it was decided to design a minicomputer relational data base management system. We have also implemented this system, calling it the Relational Inquiry and Storage System (RISS).

For any particular computer installation, RISS may be used to support one or more data bases. As an example, let us focus on one data base in the system, called "personnel information." A data base consists of relations and domains. One of the relations in this particular data base is called "employee," and is described below by a table representation:

<table>
<thead>
<tr>
<th>Name</th>
<th>Number</th>
<th>Department</th>
<th>Sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>K. Smith</td>
<td>11135</td>
<td>Research</td>
<td>f</td>
</tr>
<tr>
<td>N. Greenberg</td>
<td>11136</td>
<td>Research</td>
<td>f</td>
</tr>
<tr>
<td>R. Jones</td>
<td>1125</td>
<td>Data Processing</td>
<td>m</td>
</tr>
</tbody>
</table>

The rows of the table correspond to tuples of the relation (records), and the columns correspond to instances of particular domains of the data base. A domain is a particular class of data values (objects). More than one column of a relation may have the same domain. A useful representation of a domain is a datatype. A datatype may be viewed as a specification of the class of objects a domain may contain. For example, the datatype "Sex" may be specified by stating that there are three possible values: "f" (or "female"), "m" (or "male"), and null (unknown). Of course, a datatype specification will often be more complicated than this.

The "employee" relation above has four columns. Each column is a distinct domain in this case. Each of the three tuples in this relation contains one data item (possibly null) for each column of the relation. Implicit here is the fact that tuples may be added to and deleted from the relation as "employees" are added or deleted. Also, the value of any column for a given tuple may be altered at any time. The structure of a relation may also be changed, e.g. by adding a new column; in some sense this results in a substantial change in the meaning of the relation.

RISS employs, without loss of generality, normalized relations, so that they may be represented by a table. The table representation of a relation should not contain essential ordering information. The table may be ordered in a useful manner, but the order should be re-constructable from data in the relation. Each tuple of the relation may be distinguished by its primary key, that is, by one or more columns which uniquely specify that tuple.

In the "employee" relation, either "Name" or "Number" may be used as a primary key.

INTERFACE LEVELS

Liskov and Zilles have discussed the use of abstract data types (levels of abstraction), and Aiello has reviewed the support provided for this by existing programming languages. We believe that this approach to data management is one that has great potential value in conjunction with data base management systems. In addition, it seems advantageous to provide a structured (layered) approach to the data base management system. In this approach, each type of user is allowed to approach the system at a level consistent with his current needs.

In RISS, there are two interface levels. First, the "naive" user interface level is for routine (trained) users of the system, such as data clerks, as well as for managers and researchers who use the system occasionally. This
RISS functions

The RISS functions are a set of callable routines which may return information to the caller. These functions are callable from programs written in the language used to implement RISS, which is the same language that is used to write RISS applications programs. In effect, these functions define an extension of the (host) language. Functions are provided to facilitate the creation and deletion of relations to/from a data base, the addition and deletion of columns to/from a relation, the return of information about the columns (domains) of a relation, and the return or alteration of the value of a specified column for a particular tuple in a relation.

At this point, it is important to observe that in RISS we have not dealt with domains in the general sense that is most desirable. That is, we have not permitted the user to express a variety of types of integrity constraints. The ways in which the user may specify and/or restrict the permissible objects in a given domain are indeed very limited. The only type of constraint allowed is the specification of the storage representation, such as an integer or a varying length character string. We believe that it would be extremely advantageous to allow many other types of constraints such as “all the values of a column must be selected from a prespecified finite set of objects”, or “all values in the ‘name’ column of relation ‘employee’ must exist in the ‘person’ column of relation ‘master list’. “ We are currently investigating the needs of the users of RISS in this area, as well as the applicability of abstract data types to the handling of domains.

Data base query languages

At this point, we should note that Codd identifies five types of languages which have been developed for the query and modification of relational data bases:

1. element-by-element
   In this type of language only one tuple of a relation may be referenced at a time. The RISS (applications) program interface level is an example of this type of language.

2. algebraic
   These types of languages, such as MACaims and the Multics RDMS, involve operations on entire relations.

3. relational calculus
   Codd’s ALPHA, the primary example here, is a language based on the calculus of relations.

4. mapping oriented
   SQUARE by Boyce, Chamberlin, King, and Hammer provides an example of this type of language. In SQUARE the user learns a simple query specification format which is powerful and concise. An English language oriented version of SQUARE has been implemented by Boyce and Chamberlin. Query by Example by Zloof is another rather ingenious example of this type of language. The RISS “naive” user interface level is related to these languages, although it is certainly less general.

5. natural language
   Codd’s RENDEZVOUS can truly deal with “casual” users (users who have little knowledge of the data base management system), by allowing them to express queries in natural language. The system will ask the user for help if it cannot completely understand the query. Clearly this type of language is extremely desirable, but nontrivial to implement.

As mentioned previously, we believe that it is advantageous to utilize a structured approach to the relational data base management system interface. A good approach might be a four level system, such as:

1. a set of element-by-element primitives (lowest level), such as the RISS primitive functions
2. a set of operations based on the relational calculus, such as ALPHA
3. a language, such as SQUARE, for “naive” users (or the RISS “naive” user interface level)
4. a natural language based level for “casual” users, such as RENDEZVOUS

As a compromise, RISS has levels of the first and third types only.

RISS “naive” user interface level

The RISS “naive” user interface level has three basic components, which we will discuss in some detail: a relation editor, a retrieval package, and a data base manipulation and maintenance package.

The editor

It seems very important that users who perform editing operations on relations (such as creating and updating tuples) be provided with a cohesive package which supports their needs but isolates them from unnecessary details. RISS provides such a package: the RISS editor. The user enters the editor, specifying the relation to be edited, and may then use the editor commands to examine and modify the relation.
The editor design is based on that of the Multics editor "edm." A relation is viewed as an ordered list of tuples. There is a "current tuple pointer", which points to the first tuple in the relation when the editor is entered, and may be moved by editor commands so that it points to any tuple in the relation. The editor contains commands that allow the user to:

1. move the pointer an integer number of tuples forward or backward
2. move the pointer by searching a column for a specified value (e.g. for character strings an "exact match" or "substring" search)
3. delete one or more tuples after (including) the current tuple
4. create a new tuple after the current tuple
5. display or change the value of a column of the current tuple (several ways are provided), or of all tuples in the relation
6. provide descriptive information about the relation (e.g. the (domain) name of a column)

The retrieval package

The RISS retrieval package is designed to allow the user to retrieve and analyze the data in RISS relations. It provides commands which facilitate:

1. selection of a set of retrieved tuples (retrieved set) on the basis of a column value comparator (e.g. sex="male", age>18)
2. modification of the retrieved set by forming the union or intersection of the retrieved set with a new set of tuples selected by a column value comparator
3. extraction of a subset of the columns of the tuples of the retrieved set
4. printing a tabular or other type of report based on the retrieved set
5. printing simple statistical information (e.g. mean, median, for numerical data) for the retrieved set
6. forming several groups of the data in a particular column by specifying a range of values for each group, and obtaining a list of how many tuples of the retrieved set fall into each group
7. producing a list of all the distinct values of a particular column for the retrieved set and the frequency of occurrence of each distinct value.

Of course, complex combinations of the above operations are also possible.

The maintenance and manipulation package

The RISS data base maintenance and manipulation package allows the user to perform various operations necessary for the maintenance and routine use of a data base. Not all of these operations need be accessible to all users. They include facilities for relation creation and deletion, relation copying, sorting, and merging ("joining"), adding and deleting columns to/from a relation, etc.

IMPLEMENTATION

An implementation of RISS has been completed using the RSTS-11 computer system at Forest Hospital in Des Plaines, Illinois. The language used, both to implement RISS and to allow applications programs to access RISS (the host language), is called Basic-Plus. A relatively powerful translator for this high level language is supplied with the RSTS-11 system.

Although implemented in a high level language via a translator which is not a strict compiler but is partially interpretive, the RSTS-11 implementation of RISS is sufficiently efficient. The basis for this statement is the general opinion of the RISS users that the system is very responsive. This is not to say that there are not some "bottlenecks" in the initial version of RISS, but that in general these appear not to be serious with respect to the context of the minicomputer environment.

It may be interesting to mention that the RSTS-11 RISS implementation was accomplished in less than 4 man-years. Work began in February, 1974 and the initial version was completed and released for general use in August, 1974. Since its release, RISS has run with no modifications to the software. We believe that this satisfying result is due to the structured approach taken in the design and implementation (such as the structured interface). In addition, a surprisingly small amount of data base maintenance (by programmers or experts) has been required. That is, "naive" users have been able to maintain their own data bases for the most part.

Storage of information

All information necessary to support RISS data bases is stored in files provided by the RSTS-11 file system. The scheme adopted for RISS is to use three files for each relation. First, the relation descriptor file contains a description of the relation, including the number of tuples currently in the relation, along with a description of each column of the relation and the corresponding domain. Second, the tuple file contains an entry for each tuple in the relation. Each entry is a list of pointers (possibly null), one for each column of the relation. These entries are stored sequentially as fixed-length blocks in the current implementation. The pointers in the tuple file point to the actual data items, which are stored in the third file, the column value file. Actually, if the data item is capable of being stored in a space smaller than that occupied by a pointer (three words), then it is stored in place of the pointer. Additionally, there is a file that contains the names of all relations and domains in a data base (a data base "directory").

All files are accessed via the RSTS-11 mechanism which
RISS functions

As discussed previously, the RISS functions operate on a low level, and unlike the "naive" user interface level, their basic structure is implementation dependent. Specifically, this is because they actually manipulate the relation representation. The standard set of RISS functions present in the RSTS-11 implementation deals with the relation file representation described in the previous section (above). In this scheme, tuples are referenced by their index (row number in the table representation), and columns in a similar manner (column number). If another relation representation is desired, or a modification of the simple table representation is desired, such as inverting one or more columns (e.g., the primary key), then a modified set of functions can be constructed. Such a method for inverting columns in RISS relations is currently under development.

It is important to note that, due to the limitations in the facilities for modular programming in RSTS-11, it was necessary to design a "preprocessor" for all RISS programs. This preprocessor scans a program for references to RISS functions, and inserts the code for the functions used to a temporary module, which is then compiled. Thus, a compiled module that performs no external calls (to access RISS functions) is produced.

Limitations

At this point, it seems appropriate to mention what appear to be some of the most significant limitations of the RISS system (Forest Hospital implementation):

1. The handling of domains is insufficient. Specifically, no general method has been provided for expressing domain integrity constraints, i.e., limitations on the types of admissible values (objects) in a domain.
2. Concurrent access to relations in a database is limited, since the implementation thereof is based on the scheme provided by the host operating system. Many users are allowed to examine a relation simultaneously, but only one user may update that relation (a standard locking mechanism is employed).
3. Column naming may be ambiguous. Only one name is assigned to each column of a relation: the name of the underlying domain. Thus if two columns in a relation have the same name, they must be referenced by their column number. A unique name for each column in a relation should be added, and users should not have to remember column numbers.

4. Retrieval involving more than one relation is not simple. In order to formulate a query that involves more than one relation, it is necessary to explicitly create a temporary relation (via one or more joins) using all the relations involved in the query.

5. Protection is based on the limited scheme provided by the host operating system. Read and/or write permission may be given to groups of users, but further refinement of protection constraints (e.g., to columns of a relation) is not possible.

6. Since joins are represented as "snapshots" of a database, it is possible for inconsistencies to arise between the relations and the (implicit) joins when updates occur. Consequently, a type of database integrity is not strictly enforced.

Other limitations exist, but in the interest of brevity they will not be discussed here. In addition, a survey of the users of RISS is being made to determine what modifications to RISS are necessary and/or desirable.

CONCLUSIONS

It has been shown that a relational database management system is practical in the minicomputer environment. Although some compromises in power and generality were necessary, a useful and cost-effective implementation has been produced. The implementation has been produced in a rather short amount of time and is surprisingly "self-maintaining". In addition, the implementation was accomplished using an existing minicomputer operating system and host language, and at a low development cost.

ACKNOWLEDGMENTS

The authors would like to thank Robert Pellicore of Forest Hospital for his helpful suggestions concerning the RISS applications program interface (level). Professors Michael Hammer and Barbara Liskov of Project MAC at the Massachusetts Institute of Technology have been helpful in providing useful comments on the subject matter of this paper. The referees have also made many useful comments on the content of the paper. Kathy Vick, Nancy Wingren, and Edie Johnson of Forest Hospital have provided many helpful comments on the RISS "naive" user interface level. Marsha Baker of the Department of Electrical Engineering at M.I.T. assisted in the preparation of the manuscript.
REFERENCES