Evaluating inter-entry retrieval expressions in a relational data base management system

by JAMES B. ROTHNIE, JR.
Department of Defense Computer Institute
Washington, D.C.

INTRODUCTION

Among the most important current concepts in data base technology is the relational model of data base management. The theory was introduced by Codd\(^1\) in 1970 and has since been expanded in a large number of articles by Codd,\(^2,3,4\) and other authors, for example.\(^5,6,7,8,9\) The relational concept models the user view of a data base. As a user view it offers many applications significant advantages over more traditional approaches such as that proposed by the CODASYL Data Base Task Group.\(^10,11\) Unfortunately some of the desirable properties of the relational model, and in particular the very important attribute called data independence, make the concept difficult to implement efficiently. This paper will describe an approach to solving one of the key implementation problems. The concepts discussed here have been embodied in an experimental system called DAMAS developed at the Massachusetts Institute of Technology. A complete discussion of DAMAS can be found in Reference 12.

A brief review of relational concepts

It will be assumed in this paper that the reader is familiar with the basic relational concepts but the terminology and notation will be very briefly reviewed.

In the relational user view a data base consists of a set of named relations. Each relation can be viewed as a simple table, a construct approximately equivalent to the more traditional idea of a file. With the tabular picture in mind, each row of a relation is called a tuple, where a tuple corresponds to a file entry or record in more familiar terminology. A relation is a set of tuples and, hence, no two tuples in a relation are identical. Each column is given an attribute name. A tuple contains one value (possibly null) for each attribute defined for the relation of which that tuple is a member. A relation called "employee", for example, can be defined having the attributes "name", "salary", and "department". A tuple of "employee" would provide a description of an individual in terms of those three attributes, for example, (John Jones, 12000, Sales).

The data structure in the relational model, then, is very simple. A data base consists essentially of a set of tables. The power of the relational model is derived from the mechanisms a user has available to extract information from this simple data structure. Codd has specified a language called DSL/ALPHA\(^2\) which can be used to express operations on a relational data base. This language is somewhat awkward syntactically and other authors have defined more congenial versions based on this original, for example.\(^12\) DSL/ALPHA, however, remains the most commonly known and we will use it to express retrieval problems in this paper. Two example expressions will adequately illustrate the relevant DSL/ALPHA constructs.

To extract from the "employee" relation the names and salaries of all employees earning more than $10,000 and place these names in a new working relation called W1 we would use the following expression:

```
RANGE EMPLOYEE E
GET W1 E. NAME, E. SALARY:
E. SALARY > 10000
```

The phrase "E. NAME, E. SALARY" is called the target list and it identifies those attribute values in selected entries which will be included in the new relation. "E. SALARY > 10000" is the qualification and it describes the condition which tuples must satisfy to be selected for the new relation. "RANGE EMPLOYEE E" is the range declaration. It defines E to be a tuple variable which ranges over the EMPLOYEE relation. The role of a tuple variable is essentially that of a bound variable in a set definition. For example, the following definition indicates which EMPLOYEE tuples would be selected for W1 and illustrates the role of E:

```
W1 = [E where E is a tuple of EMPLOYEE and the SALARY attribute of E is greater than 10000.]
```

Retrieval expressions, like this one, which contain only a single tuple variable play an important role in the implementation method to be presented and they are termed primitive Boolean conditions (PBC). The real usefulness of the tuple variable concept, however, arises only in expressions where more than one tuple variable is required in the qualification. Consider, for example, the following problem.

Suppose we add to the EMPLOYEE relation a new attribute, MANAGER, which indicates each employee's manager. If a tuple has the NAME attribute John Jones and MANAGER attribute Jack Smith, then Jack Smith is
In the qualification, "E. MANAGER = M. NAME" establishes the required managee/manager relationship between E and M while "E. SALARY > M. SALARY" expresses the requested salary relationship. The range declarations indicate that both tuple variables E and M range over EMPLOYEE. The key word SOME in the declaration for M indicates that M is existentially quantified. That is, to select a tuple for E to include in W2 we need to establish the existence of a pair (E,M) which will make the qualification true. We need not draw any data from M; we need only establish its existence.

DSL/ALPHA permits the use of any number of tuple variables, each of which can be unquantified, existentially quantified or universally quantified. The language supports standard retrieval and maintenance operations and offers a number of other features. For the purposes of this paper, however, we will confine our discussion to retrieval (GET) expressions involving two tuple variables, one unquantified and one existentially quantified.

The implementation problem

A key goal of the relational model is to facilitate data independence in a data base management system. Generally a system can be said to be data independent if users and application programs are unaffected by changes in the physical structure of the data base and by certain types of changes in the user view.

The relational model encourages a system architecture possessing this characteristic by offering a high level user view which is quite divorced from storage structure and search algorithm concepts. This gives a well conceived relational system flexibility in electing a physical representation and search procedures, based on performance considerations, and in changing those choices as required. This is in contrast to systems, typified by the DBTG proposal, which depend upon user procedures to identify desired data objects. The greater level of detail in the DBTG user view restricts the system's opportunity to make or change representational choices. Substantial changes on the physical side will almost certainly involve changes to user decisions and hence will require user involvement.

The challenge to the implementor of a data independent system is to achieve efficient execution in the absence of the detailed user guidance available to DBTG-like systems. This is particularly a problem for relational systems since their powerful user languages permit users to request complex computations very simply. It is critical that the system be able to handle such requests efficiently or execution times are likely to exceed practical limits. The problem primarily arises for retrieval requests involving multiple tuple variables. To see this, note that for the most simple-minded implementation technique, exhaustive search, the execution time for handling a one tuple variable request is proportional to N, the number of tuples in the relation. The similar measure for a selection expression involving M tuple variables, each ranging over the N tuple relation, is N^M, the number of unique ordered collections of N tuples taken M at a time. This, of course, is a very crude bound and it is easy to construct algorithms which can improve upon it. However, this simple analysis does illustrate the danger of an exponential explosion in computation for this type of query, a danger which does not exist in a PBC. The remaining sections of this paper will be devoted to the description of a technique for handling two tuple variable queries, a technique which offers a far better performance than the brute force bound.

AN APPROACH TO IMPLEMENTATION

The system macro-organization

The algorithms we will be discussing for evaluating two entry variable retrieval expressions can be conceptually partitioned into two major modules. One, called the storage module, is responsible for the physical representation of relations and for responding to certain types of requests from the second module, called the multi-tuple variable module (MTVM). These requests take the form of subroutine calls and they require the storage module to perform certain operations on the stored relations, including:

- returning a sequence of tuples satisfying some qualification;
- determining the existence of a tuple satisfying some qualification;
- eliminating from further consideration in the current computation all those tuples which satisfy some qualification.

The key to the organization of this technique is in the nature of the qualifications which the tuples to be identified by the storage module must satisfy. In each case, this qualification is a PBC, i.e., a one tuple variable qualification. The storage module appears to the MTVM as a set of rather powerful primitives for operating on relations given a PBC to identify the tuples of interest. It is the responsibility of the MTVM to use these PBC handling primitives as a part of an algorithm which is capable of handling more than one tuple variable. The main thrust of this paper is to describe how the MTVM accomplishes this for two tuple variable expressions.

There are in fact a variety of techniques available for handling PBC's (e.g., inverted files,\textsuperscript{14} multi-list files,\textsuperscript{15} and multiple key hashing)\textsuperscript{16} Each of these methods offers ad-
vantages over the others in certain situations, so that a system with important claims to generality should not depend on a single one of these alternatives for all relations. To deal with this problem the DAMAS implementation was designed to operate with an unlimited number of storage modules, all with identical interfaces to the MTVM but each offering a different means of storing relations and handling PBC's. The organization is sketched in Figure 1.

In this paper we will be focusing attention on the MTVM; the internal structure of storage modules will not be further considered. We will view each storage module as a collection of PBC handling primitives. Even the details of these primitive calls will be suppressed here; the missing details can be found in References 12 and 17. Our approach in this paper will be to follow the operation of the MTVM on an example problem and to suggest the associated storage module operations as appropriate.

The example multi-tuple variable module operation

Figure 2 shows three example relations and a GET statement in DSL/ALPHA. The GET statement contains two tuple variables T1 and T2 which range over the pictured relations R1 and R2 respectively. T2 is existentially quantified. The result of processing the statement is R3 and it contains the A4 and A5 attribute values from tuples R1 which satisfy the rather complicated qualification. The reader should examine this qualification and understand its logic. A tuple T1 of relation R1 will be selected if:

1. Its own attributes satisfy the condition (T1.A1<3 OR T1.A3=1) and T1.A2>6
and
2. There exists a tuple T2 is R2 such that the pair of tuples (T1, T2) have attributes which satisfy the condition T1.A4<T2.A3 and T2.A1=T1.A1

The details of this qualification are not important of themselves, but it is helpful to understand them in considering the operation of the MTVM in evaluating this expression.

The mission of the MTVM is to evaluate the complete tuple selection expression. It accomplishes this by decomposing the expression into PBC's (one tuple variable qualifications) for evaluation by the storage modules. PBC's are derived from the original GET statement, possibly in combination with data retrieved from the relations during the course of the evaluation. The philosophy which guides the construction of PBC's and calls to the storage modules is an effort to derive as much information as possible from each access of the data base. Assuming that these accesses are the most expensive operations performed in the evaluation process, this is an attempt to gain maximum benefit for the cost.

The first action which the MTVM takes on encountering our example expression is to construct a PBC which eliminates from further consideration any tuple of R1 which can be discarded on the basis of its own attributes. For example, any tuple with attribute A1 greater than 2 and A3 not equal to 1 can be eliminated in this way. The MTVM implements this concept by constructing a PBC, called PBC1, which identifies that subset of R1 which cannot be eliminated in this way. The tuples satisfying PBC1 will then be subjected to further analysis. For this qualification, PBC1 is:

(T1.A1<3 OR T1.A3=1) and T1.A2>6

In general PBC1 can be constructed by setting all conditions involving T2 in the original qualification to TRUE and simplifying. The MTVM asks the storage module handling R1 to restrict its attention to the subset identified by PBC1 for all subsequent operations on the tuple variable.
Note that in the example relation this subset consists of the tuples identified by the numbers 1, 2, 3, 6, 7, 8, 9, 10 and 14. Then the MTVM asks the module to return one tuple from this subset. Let this tuple be called E1.

Suppose that E1 is the tuple (1, 7, 1, 3, 1). The next task of the MTVM is to determine if this tuple satisfies the complete qualification. This involves ascertaining whether or not there exists a tuple in R2 which when paired with E1 satisfies the complete qualification.

To accomplish this the MTVM constructs a new PBC, called PBC2. PBC2 has the property that, if there exists a tuple in R2 for which PBC2 is true, then the tuple E1 satisfies the complete qualification. In general, PBC2 is derived by substituting the attribute values of E1 for the attributes associated with the tuple variable T1 and simplifying. For this particular qualification and choice of E1, PBC2 is:

\[3 \leq T2.A3 \text{ AND } T2.A1 = 1\]

As it happens, there is a tuple (1, 1, 5, 2, 1) in R2 which satisfies PBC2 so the target list from E1, (3, 1), is included in the new relation R3.

At this point, it is possible to repeat the cycle by drawing a new value from the PBC1 subset, constructing a new PBC2, checking for qualified tuples in R2 and so forth. However, in this and many other cases, it is possible to gain additional benefits from the initial iteration. Specifically, given the existence of the tuple (1, 1, 5, 2, 1) in the relation R2, the MTVM can construct a PBC, called PBC3, which will identify additional tuples in R1 which fully satisfy the qualification because of the existence of this tuple in R2.

In general, PBC3 is constructed by replacing the T2 attributes in the original qualification with the contents of some tuple in R2 which satisfies PBC2, and simplifying. PBC3 in this case is:

\[(T1.A1 < 3 \text{ OR } T1.A3 = 1) \text{ AND } T1.A2 > 6 \text{ AND } T1.A4 \leq 5 \text{ AND } 1 = T1.A1\]

Since PBC3 is constructed from a known tuple of R2, any tuple of R1 which satisfies it will pass the complete qualification. In this instance when PBC3 is applied to R1, the additional tuples 2 and 3 are selected and their target lists are added to R3.

At this point, then, the MTVM returns to the storage module handling R1 and asks for another tuple E1 from the PBC1 subset. Suppose that this time E1 is (2, 8, 2, 4, 2). Again PBC3 is constructed, and for this tuple substitution we derive:

\[4 \leq T2.A3 \text{ AND } T2.A1 = 2\]

However, an examination of R2 will reveal that there is no tuple which satisfies this qualification. Hence E1 is not selected and its target list is not added to R3.

From here the MTVM can simply conclude the iteration and return for another tuple of the PBC1 subset. However, even the failure of a tuple to satisfy the qualification provides important information about the relations involved, and the system can take advantage of this information to avoid some later searching. Specifically, this failure indicates that there does not exist any tuple in R2 whose attributes satisfy the conditions in PBC2. Hence we can eliminate from further consideration any tuple in R1 whose attribute values will generate the same PBC2. Furthermore, if no tuple in R2 satisfies PBC2 then, clearly, no tuple exists which satisfies a qualification which is the same as PBC2 except that 4 is replaced by a larger value. An examination of the full qualification indicates that any tuple having a 4 or greater in attribute A4 and a 2 in attribute A2 will generate such a PBC2. The MTVM constructs a new PBC, called PBC4, to identify such tuples and it requests the storage module for R1 to eliminate these tuples from the PBC1 subset. In this case, PBC4 is:

\[4 \leq T1.A4 \text{ AND } T1.A1 = 2\]

Tuples 7, 8, 9, and 10 are thereby discarded from the subsequent search.

After two complete iterations, then, we are left with only one tuple from the original PBC1 subset remaining to be considered. This is tuple #14 (3, 8, 1, 3, 1). However, this tuple need not be processed because its target list (3, 1) is identical to a tuple already present in R3. Since a relation is a set it can have no two identical members and a second insertion of (3, 1) will have no effect on the relation. Hence there is no reason to process the tuple. In order to recognize such situations, whenever a new tuple is added to R3 the MTVM can construct a PBC, called PBC5, which identifies all other tuples with identical target lists. The storage module for R1 is then asked to eliminate such tuples from the PBC1 subset. In this case, PBC5 is:

\[T2.A4 = 3 \text{ AND } T1.A5 = 1\]

If tuples satisfying PBC5 had been discarded when tuple #1 was selected for inclusion in R3, then at this point all tuples would have been handled and the algorithm complete.

**An example performance analysis**

This contrived example has illustrated the approach taken in the DAMAS implementation. While it was possible to construct a situation in which this specific algorithm was effective, it is equally possible to cite examples in which a simpler algorithm would be more effective. This is true because the use of the procedure steps associated with PBC3, PBC4 and PBC5 will not always identify enough tuples to justify their overhead. For this reason, each of these mechanisms is treated as an option to be employed or not depending on the situation.

Experimentation with DAMAS verified speculation that the decision to employ each option or not would have an important impact on performance and the optimal choices would vary from query to query. The following example from the experimental results will illustrate that point.
A relation called R1 was defined and loaded with 640 synthetically constructed tuples. The relation was assigned ten attributes named A1 through A10. Attribute values were randomly generated integers with all values, except those for A4, uniformly distributed from 1 through 16. The values for A4 were uniformly distributed integers ranging from 1 through 400. The storage module for R1 used a technique called multiple key hashing. The details of this method are not relevant to an understanding of the example results. However, it is useful to know that the technique as applied here was most effective at identifying tuples with a given value for A1, somewhat less effective for A2 and A3, and still less effective for the other attributes.

The objective function used in comparing the various combinations of options available in DAMAS was the minimization of page accesses. The rationale for this choice was based on the observation that in the DAMAS computer system environment (MULTICS), page accesses account for an overwhelming fraction of the total cost of querying a data base. Since a paged memory can serve as a gross model of any large database environment, the results should be roughly extendable to other computer systems.

The first example query we will consider is the following:

```
RANGE R1 T1
RANGE R1 T2 SOME
GET W1 T1. all attributes: T1.A2=5 AND T1.A1<T2.A5 AND T2.A2=5 AND T2.A3=5
```

When this retrieval expression was evaluated without using any of the special options, PBC3, PBC4 and PBC5, DAMAS performed 304 data page accesses. Of these, 248 occurred while searching R1 for a tuple satisfying PBC2—and failing to find one. A look at PBC2 will reveal generally the reason for this performance. PBC2 was of the form:

```
C<T2.A5 AND T2.A2=5 AND T2.A3=5
```

where C was replaced by the specific value drawn from attribute A1 of the tuple E1 used to generate PBC2. Because of the way that the storage module handled R1, a tuple satisfying this PBC might have occurred on eight different pages for any value of C. When there was no tuple in R1 which satisfied PBC2 the storage module searched all of these pages. As it happened there were 31 tuples in the PBC1 subset which failed to satisfy PBC2. Since each of these incurred eight accesses, the total of 248 for this portion of the algorithm resulted.

The relatively high cost of handling the case of negative results of PBC2 suggests that the use of PBC4 might have been fruitful. Recall that PBC4 provides a means of identifying some tuples which will fail PBC2 without performing a complete iteration for each of them. PBC4 in this case would be of the form:

```
T1.A1>C
```

with C replaced by the A1 attribute of a tuple which failed on PBC2. As mentioned above, the storage module was effective in handling PBC's involving A1 so that processing of this PBC would not be excessively costly. Furthermore, the nature of this PBC suggests that a large fraction of the tuples which would fail PBC2 might be identified by processing PBC4.

When the experiment was actually performed, the first PBC4 constructed was:

```
T1.A1>6
```

Processing this expression required 12 page accesses and the result was the elimination of all remaining tuples which would fail PBC2. Hence at a cost of 12 page accesses required for PBC4, 248 page accesses were avoided. It might be argued that this substantial success was a fortunate accident. It is true that some other choices for E1 would produce less effective results. However, for this problem the highest cost possible for using PBC4 was 248 page accesses, still a substantial reduction from 248. In any case, the point that significant savings can be achieved with this option is clearly demonstrated.

The use of PBC3 in this case saved only 5 page accesses. This option was less effective than PBC4 because there were fewer tuples in the PBC1 subset which satisfied PBC2 and because the cost of handling each of them was less than the cost for those which fail PBC2. The use of PBC5 was clearly precluded because the target list contained all of the attributes of R1. Since relations are defined to contain no duplicates, searching for an identical target list was certain to be fruitless. If, however, the target list had been a single attribute, A1, then the use of PBC5 would have been valuable because of the high probability of identifying duplicates.

While PBC4 was effective in this example there are many instances in which it will be useless and, in fact, costly to employ. Consider, for example, the following retrieval expression:

```
RANGE R1 T1
RANGE R1 T2 SOME
GET W2 T1. all attributes:
```

For this problem PBC4 is of the form:

```
T1.A2=C1 AND T1.A3=C2
```

where C1 and C2 are replaced by attribute values drawn from tuples which fail PBC2. The probability that a random tuple will satisfy PBC4 is, 1/256. Hence the use of PBC4 in this example is very unlikely to yield results which justify its overhead.

Strategy selection

The above performance analysis should clearly illustrate the following important point. For this approach
to handling multi-tuple variable expressions to be effective, it is essential to employ an adequate method of deciding which options should be used in a particular situation. DAMAS did not include a facility for automatic strategy selection; users were asked to indicate which options they wished to employ. In the long run, however, it is clearly desirable, in the interest of data independence, user simplicity, and optimal choices, for the system to make this determination. The discussions in the last section which examined why the use of a certain option was effective in a given situation suggest a methodology for predicting which options will be hopeful. This section further explores that notion. A more detailed consideration of this topic can be found in Reference 19.

The MTVM selects a strategy by making a yes-no decision on the use of each of the special options PBC3, PBC4, and PBC5. Each of these decisions is based on an approximate calculation designed to indicate whether the costs avoided in using an option exceed the costs incurred in using it. The information concerning page accessing costs and numbers of tuples identified is obtained from the storage modules via special calls available to the MTVM. A sample decision on the use of PBC4 will illustrate the types of data obtained from the storage module and the calculations based on these data.

The MTVM requests the storage module to provide three items of information:

1) The expected number of tuples in the PBC1 subset which will be identified by one application of PBC4. This is obtained from the product of the probability that a random tuple will satisfy both PBC1 and PBC4 and the number of tuples in the relation. The call to the storage module includes PBC1 and PBC4 as arguments. The returned value is called N4.

2) The expected cost in page accesses of processing PBC2 when the result is false. Call this result C2.

3) The expected cost in page accesses of processing PBC4. Call this result C4.

Given these values the cost avoidance can be simply computed as N4 × C2, the number of tuples identified times the cost of processing PBC2 for each one. If this value is greater than the cost incurred, C4, then PBC4 is used.

The same type of calculation can be performed for PBC3 and PBC5. Intuitively the approach seems likely to make reasonable choices in most situations but it has not been implemented and tested experimentally. Further exploration into the problems of strategy selection are clearly required.

SUMMARY

This paper has described via example a methodology for handling relational data base expressions involving two tuple variables. Somewhat in the manner of McDonald et al., this technique assumes the existence of one or more mechanisms for handling one tuple variable expressions (PBC's) and employs these as primitives in handling higher level expressions. The technique therefore is not concerned, at this level, with the detailed list manipulating operations by which the DIAM approach handles inter-tuple associations. Such machinations might occur within the storage modules but they are invisible at the level of the MTVM where this paper was centered.

Unlike the McDonald approach, the technique described here does not involve a fixed sequence used in a largely identical way for all queries. Instead, this approach offers the possibility of using three options which in some situations can yield very substantial performance efficiencies. These options attempt to utilize the increasing information about the database which is acquired with each access to a relation. Experimentation has verified that significant cost savings are possible when the appropriate combinations of these options are employed. An important difficulty, however, is the process of deciding which options to employ. A simple approach to automating these decisions was briefly discussed.

ACKNOWLEDGMENTS

The work was supported in part by the Cambridge Project under contract DAHC15 69 C 0347 of the Advanced Research Projects Agency and in part by the MIT Civil Engineering Systems Laboratory. Their support is gratefully acknowledged. Special thanks to Mr. John Rothnie for his able graphics assistance.

REFERENCES

10. CODASYL Data Base Task Group—April 71 Report.
11. "CODASYL Data Description Language," Journal of Development
Systems, Ph.D. Dissertation, Dept. of Civil Engineering, Mass. In-
stitute of Technology, September 1972.
on Data Description, Access, and Control, Ann Arbor, May 1974.
15. Martin, L. D., "A Model for File Structure Determination for Large
On-line Data Files," File Organization, Selected Papers from File 68-
AIAA DAG Conference, Amsterdam Swetsand Zeiglinger, New York
16. Rothnie, J. B. and T. Lozano, "Attribute Based File Organization in
a Paged Memory Environment," Comm. ACM 17 No. 2, February
17. Rothnie, J. B., "An Approach to Implementing a Relational Data
on Data Description, Access, and Control, Ann Arbor, May 1974.
18. Corbato, F. J. and V. A. Vyssotsky, "Introduction and Overview of
the MULTICS System," Proc. AFIPS 1965 FJCC, Vol. 27, AFIPS
Systems, Ph.D. Dissertation, Dept. of Civil Engineering, Mass. In-
20. Senko, M. E., E. B. Altman, M. M. Astrahan and P. L. Fehder,
"Data Structures and Accessing in Data Base System," IBM