PRTV, an efficient implementation for large relational data bases

Stephen Todd, IBM UK Scientific Centre, Peterlee, England. 5 August 1975

INTRODUCTION

PRTV [6] is a research prototype interactive data base. It is intended for use on its own or as part of a specialist system. The principle objectives are to provide: a) high level flexible data support, b) functional extensibility and c) reasonable efficiency. The prototype is also designed as a method of generating and testing ideas in a wider data base research programme.

The prototype will run under OS, VS or VM/CMS. Input comes from a terminal or an input file.

PRTV has been fully operational since August 1973. The largest data base handled requires 50 megabytes and consists of about 50 relations, one containing 65,000 tuples of 128 columns. As far as we know this is more than an order of magnitude larger than any data base run under any other relational system [7]

INTERFACE

The interface to PRTV is a relational algebra [2] . The operations are union (+), intersection (.), difference (-), projection (%), selection (:), and join (*). Join covers cartesian product and equi-join.

Relations are held in named variables with a PL/I-like syntax for assignments and expressions. As an example consider a library with two basic relations, both with degree three, and structure BOOKS(ACQ#,AUTHOR,TITLE) and LOANS(ACQ#,BORROWER,DATE_OUT). The query 'Who has "Persuasion" on loan?' is posed as BOOKS*LOANS:TITLE='Persuasion'. The join is a natural join, and is followed by the select.

To simplify queries of the type "What is the effect of such and such?", an APL-like workspace is provided in which changes can be tested but not commited.

The algebra has been streamlined [4] . This gives three advantages. The concepts of 'domain name inheritance' and 'relationship'[2] are formalized. A generalized difference that is simpler to use is provided. Finally it permits addition of a large class of user extensions in a very simple manner.

The generalized difference produces tuples in its result that are in the first input relation and for which there is no tuple in the second which matches in the columns with matching names. Thus IN=BOOKS-LOANS will assign to IN details of all the books not currently on loan. The relations differenced need not be defined on the same domains.

Since functions that apply to one tuple at a time are treated as relations, their inclusion is particularly easy. Such functions are those which create relations with computed fields, and user defined selections. As an example of the second, the user writes a PL/I function that tests for one string in another.

```
LIBINDEX:PROC(A,B) RETURNS (BIT);
DCL (A,B) CHAR (*) VAR;          /* test for second parameter in first */
RETURN (INDEX(A,B)¬=0); END;     /* return '1'B if it is */
```

Then BOOKS*LIBINDEX(TITLE,'relations') is all books with 'relations' in the title.

User extensions that cannot be implemented in this way use a relational file interface. A relational read file is a snapshot of the data in a relation. The tuples are ordered and the file has a cursor. A relational expression is turned into a file and the tuples are transferred one at a time into the programme. A relational write file is used for transferring data from PL/1 to the data base. The file is written tuple by tuple and then when it is closed it

is turned into a workspace relation and given a name.

It is important to note that relations, read files and write files are all distinct. A tuple cannot be read from a relation, a relational write file cannot take part in a union and so on.

A final interface feature is the variable binding. Normally variables are evaluated when they are used on the right hand side of an assignment. If binding by name is specified, the evaluation does not take place till the variable assigned is used. This gives a defined relation or subschema capability.

IMPLEMENTATION

Basic relations entered into the system are stored in files, one record per tuple. The files are indexed, sequential, fully sorted and use front and middle compression on the records. They are constructed using direct access. The files are never updated; change sets are used that are eventually merged into a new file.

Relations resulting from assignments are not usually held in files. A control block is kept that defines how the relation can be created from the base files using file operations which correspond to the relational operations. When binding by name has been used the control blocks also contain references to the defining relations. When assignment is made, no work is carried out on the files, only on the control blocks. This incremental compilation is called deferred operation.

Operations on base files are only carried out for listing, cardinality and reading a relational file. There is a data base administrator feature to force an assigned relation to be stored as a file.

When execution is to be carried out, an optimization is performed. The control block is transformed to a tree, with the input files at the bottom and the result at the top. The optimization is similar to that referred to by Smith [5] and Sequel [1] , but more comprehensive than either.

A set of file operations can often be reorganized into another set which will give the same result but take less time to execute. For example it will usually be quicker to make selections from two files and write the results than to write the files and select from the result. The optimization code looks at relational control blocks and modifies them to make use of this. This reorganisation is carried out on a tree form of the string.

The following are considered.

1 Filters are moved as far down a tree as possible. This causes the selections to be executed as soon as possible.

2 Projections of projections are merged into one projection.

3 Projections involving sorts are moved as far as possible towards the top of the tree, for latest possible execution.

4 Projections not involving sorts are moved as far as possible to the bottom of the trees.

5 Expressions involving several of the set operators can be reorganized according to the standard rules of commutativity, distribution and so on. The sizes of the files are used to optimize this. Estimates are made of the sizes of the intermediate values. At present, no statistics are kept to help make these estimates more accurate.

6 A search is made for common subtrees within the tree. The common value can be realized as a brick, which will prevent the duplication of the operations. Often the cost of creating the brick is greater than the cost of repeating the operations. The optimizer will estimate both costs and choose the cheaper alternative.

7 The optimizer can also make choices concerning alternative implementations of the operations. For example join can be implemented as a merge or as a double loop. The merge is quicker but requires suitably sorted input. If the inputs are not sorted, a decision is made as to whether to sort and merge or to use the double loop. Secondary indices can be used to assist both selects and joins, decisions as to when and how to use them will be made by the optimizer.

It is interesting to notice that the optimizer can only operate effectively because of the use of deferred operations. It allows control blocks to grow more complex than those a user would cause as a result of a single input line. This permits optimization over an entire group of user statements, while giving the impression of immediate execution of each statement. Deferred operation makes possible the use of optimizing compiler techniques in an interpretive environment.

Further details of optimization are given elsewhere [3,5] .

When the tree has been optimized it is used to control the execution of the operations. In most cases the file operations take sorted input and develop sorted output. Then it is not necessary to realize fully the intermediate files that are passed from one node to another. Rather, records can be passed a few at a time up the tree as requested by the upper node. PRTV does this whenever possible.

Sometimes intermediate files do have to be fully realized and stored. This can happen for project, which does not necessarily give sorted output. The output has to be fully worked out and sorted before any records are passed up the tree. We call the node where an intermediate file must be realized on disc a break point of the tree.

ACKNOWLEDGEMENT

The author wishes to acknowledge all who have worked on the PRTV project. In particular, Garth Notley was responsible for much of the basic design, Pat Hall carried out the optimization work, and the project was managed by Terry Rogers.

REFERENCES

1. Astrahan, M.M., Chamberlin, D.D. Implementation of a structured english query language. to appear CACM.
2. Codd, E.F. A relational model of data for large shared data banks. CACM 13 6 (June 70).
3. Hall, P.A.V. Optimization of a single expression in a relational data base system. IBM UKSC report 76, March 75.
4. Hall, P.A.V., Hitchcock, P. and Todd, S.J.P. An algebra of relations for machine computation. Conference record of second A.C.M. symposium on principles of programming languages, Palo Alto (Jan 75) pp 225-232.
5. Smith, J.M., Chang, P.Y. Optimizing the performance of a relational algebra data base interface. to appear CACM.
6. Todd, S.J.P. PRTV a technical overview. IBM UKSC report 75 (May 75).
7. Codd, E.F. (chairman) Panel Discussion on relational data base management. AFIPS Conference Proceedings vol. 44 (1975 NCC).