Teaching Data Base Concepts Using APL *

J. Klebanoff, F. Lochovsky, D. Tsichritzis

Department of Computer Science
University of Toronto
Toronto, Canada
M5S 1A7
tel. (416) 928-5184

* This work was supported in part by IBM Canada Ltd.

Abstract

Rapid growth in the use of data base management systems has caused a shortage of personnel trained in data base concepts. Experience with a data base management system must be provided for students in this area.

Methods for teaching data base concepts using APL are discussed. The Educational Data Base System (EDBS), an APL based data base management system, is described. The design of exercises and games to be used with EDBS is discussed. It is concluded that EDBS used with suitable exercises is an effective tool for teaching data base concepts.

Introduction

In recent years the use of data base management systems (DBMS's) has increased at a phenomenal rate. There is a large number and wide variety of such systems on the market. A large proportion of business data processing is already being done using DBMS's and an even larger proportion will probably use a DBMS in the future. Consequently, there is a shortage of personnel trained in the use of data base management systems.

At the University of Toronto we are concerned with this problem. We are currently offering a course, Data Management Systems, which is designed to acquaint students in Computer Science or the Business School with data base management systems and concepts. An important component of such a course is experience with a data base management system. We have tried using a commercial DBMS (VANDL, an IMS-compatible system offered by IBM) but we found that it ran our small student jobs inefficiently and generally was not suited for our purposes. Therefore we have designed and implemented the Educational Data Base System (EDBS) specifically as an educational tool.

There are a number of requirements for such a system. It should be easily accesible. It should give quick turnaround or response time. The system must handle small volumes of data efficiently. It should offer a full and consistent range of high level data manipulation commands.

APL PLUS was chosen as the implementation and host language for a number of reasons. APL's primary attraction is that it provides good terminal support. We wanted EDBS to be online and APL allows us to provide such a system at a moderate cost. An important consideration too was APL's reliability and stability. We found that the powerful APL operators aided us in the implementation effort and were quite compatible with the high level data manipulation operators provided by EDBS. Since EDBS is to be used to train people to use a data base management system it is essential that they will not have trouble learning the host language. APL satisfies this requirement quite well. In short, we have found APL to be a very suitable host and implementation language for our Educational Data Base System.

EDBS Description

Data base management systems can generally be classified as one of three types -- hierarchic, relational or network. The difference is largely in the logical data structure presented to the user, which of course has its ramifications on the data manipulation language.

EDBS incorporates two systems; one is hierarchic and the other is relational. At the implementation level these two systems are quite similar and compatible. In fact, under EDBS, hierarchically structured data bases may also be viewed relationally so that a direct comparison of the two approaches can be made. A network system is also currently being designed as part of EDBS.

The EDBS hierarchic system is patterned after IMS [IMS 1971] and VANDL/1 [VANDL/1 1973]. The data is viewed as "segments" arranged in a tree structure, or "hierarchy". Each segment, except root segments, has a parent segment and may have any number of son segments. A segment is divided into a fixed number of fields.

To illustrate how EDBS is used we will use the traditional parts-suppliers data base in our examples. This is a data base used by a company which buys parts from various suppliers. Information in the data base indicates what parts are bought from which suppliers. The data base is kept simple for expository purposes and will only contain the names of the parts and the suppliers.

This data base can be modelled hierarchically as consisting of the following tree of segment types:

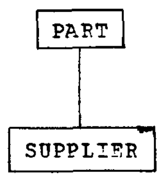

figure 1
Hierarchic model of part-supplier data base

The actual data base contains a number of PART segments, each of which may be the parent of any number of SUPPLIER segments.

The DML is procedural; only one segment may be manipulated at a time. Two pointers are maintained by EDBS for each user. The position pointer points to the segment last retrieved. The pointer-to-parent points to the current parent segment and is updated by the GET UNIQUE and GET NEXT commands.

There are three commands used for retrieving information. The GET UNIQUE (APL function GU) command is used to retrieve the next root segment of a given type. The GET NEXT (APL function GN) command retrieves the next segment. The GET NEXT WITHIN PARENT (APL function GNP) command gets the next segment under the parent pointed to by the pointer-to-parent. Each of these three commands may be qualified by specifying the segment type to be retrieved and a Boolean combination of conditions on the fields of that segment type. For example, the APL statement GU 'PART; (NAME = NUT) OR (NAME = BOLT)' will get the next part segment for a nut or bolt. (Actually, the data is not retrieved directly but is placed into a buffer.)

For each of the retrieval commands there is a corresponding GET HOLD command (APL functions GHU, GHN and GHNP) that, in addition to performing the retrieval and updating the pointers, prepares the data base so that a modification can be made.

The three hierarchic modification commands are REPLACE, DELETE and INSERT. The REPLACE command replaces the old value of the segment retrieved by a GET HOLD command with a new value supplied by the user. The DELETE command deletes the segment retrieved by the GET HOLD from the data base. The INSERT command inserts a new segment as a child segment of the segment retrieved in a GET HOLD.

Figure 2 shows two programs that perform simple manipulations of the hierarchic part-supplier data base.

Our relational system is based on Codd's ALPHA language [Codd 1971]. The data base is viewed as consisting of several relations. Each relation, like relations in the mathematical sense, can be thought of as a table or matrix. The rows of the table are called "tuples" and

```
     ∇ FINDSUPPLIER PARTNAME
[1]   ⍝ FIND ALL SUPPLIERS OF A PART
[2]    GU 'PART; (NAME =',PARTNAME,')'
[3]    →(STATUS≠0)/NOTFOUND
[4]   ⍝ GET THE FIRST OR NEXT SUPPLIER SEGMENT UNDER THIS PART SEGMENT
[5]    CYCLE:GNP 'SUPPLIER'
[6]   ⍝ IF THERE ARE NO MORE SUPPLIER SEGMENTS UNDER THE PART SEGMENT THEN RETURN
[7]    →(STATUS=4)/0
[8]   ⍝ ELSE READ THE NAME FIELD OUT FROM THE BUFFER
[9]    READ 'NAME'
[10]   →CYCLE
[11]  NOTFOUND:PARTNAME,' IS NOT A PART IN THE DATA BASE'
     ∇

     ∇ PARTNAME INSERTSUPPLIER SUPPLIERNAME
[1]   ⍝ INSERT A NEW SUPPLIER FOR A PART IN THE HIERARCHIC DATA BASE
[2]   ⍝
[3]   ⍝ GET THE PARENT OF THE NEW SEGMENT TO INDICATE TO EDBS WHERE THE NEW SEGMENT
[4]   ⍝ SHOULD BE PLACED.
[5]   ⍝ PLACE THE DATA BASE IN HOLD STATUS, THAT IS LOCK IT, WHILE THE INSERTION IS
[6]   ⍝ TAKING PLACE.
[7]    GHU 'PART; (NAME =',PARTNAME,')'
[8]    →(STATUS≠0)/NOTFOUND
[9]   ⍝ WRITE THE NEW SUPPLIER SEGMENT INTO THE BUFFER
[10]   SUPPLIERNAME WRITE 'SUPPLIER'
[11]   INSERT 'SUPPLIER'
[12]   →0
[13]  NOTFOUND:'PART ',PARTNAME,' WAS NOT FOUND IN THE DATA BASE'
     ∇
```

figure 4

the columns "domains". Our DML is non-procedural but more restricted than ALPHA.

The example parts-suppliers data base can be stored in one relation, which we will call PARTSUPPLIER, as pairs of the form: (part-name, supplier-name). Represented relationally, the data base looks like an n by 2 matrix.

There is one retrieval command, GET. The arguments of this function are a target list in which the domains to be retrieved are specified, an optional limit to the number of tuples to be retrieved and an optional qualification expression. The qualification expression contains a Boolean combination of natural joins and conditions to be satisfied by the domains of the relations. At most two different relations may be used in one GET command. A GETHOLD command is used to retrieve tuples, or selected domains of tuples, and prepare the data base so that these tuples can be updated.

There are three modification commands. Tuples retrieved by a GETHOLD command are actually updated with the UPDATE command. The INSERT command is used to insert a number of tuples into a relation. The DELETE command deletes from a relation all tuples satisfying a qualification expression.

Two typical programs that manipulate the data base using the relational commands are shown in figure 3.

Implementation

The EDBS hierarchic implementation is based on the use of "traces" [Lowenthal 1971]. A trace can be thought of as an address associated with a segment in a hierachical data base which allows acces to the data in the segment and provides some information on the ancestry of the segment.

EDBS orders the root segments and the sons of each segment. Therefore, at the implementation level, one can refer to the i'th root segment or the i'th son of a given segment. Furthermore, EDBS assigns a numeric encoding, called a "type code", to each segment type. Thus, a segment at level n in the hierarchy is uniquely identified by a trace consisting of n+1 integers. The first component of a trace specifies the segment type and the rest of the components specify the path to be followed down the tree to reach the segment. Therefore, the trace 2.1 specifies the first (root) segment of type 2 and the trace 3.1.2 specifies the second type 3 son segment of the first root segment.

EDBS requires the data base administrator to specify a limit to the number of instances of each segment type which can occur under one parent segment. This allows us to map traces into actual file components via a "position matrix". The position matrix for a segment at level n is an n-dimensional array that is indexed by the last n components of the traces for that segment. For example, if Ps is the position matrix for segments of type s then the segment specified by trace s.n1.n2.n3 is found in file component Ps[n1;n2;n3]. This puts a limit on the size and flexibility of EDBS data bases but we feel that this limit is not overly restrictive for pedagogical purposes.

There are a number of other data structures used in the EDBS implementation. Inverted lists are used to speed up searches. There is a field/domain table used to hold the names, data types, lengths etc. of each field or domain in the data base. A segment/relation table holds such information as a pointer to the position matrix, the maximum number of instances, the hierarchic level, the type codes of the parent and son segments, and identification of the fields or domains of each segment or relation in the data base. Finally, there is a translation table giving all the segment or relation names and their corresponding type codes.

```
    ∇ FINDSUPPLIER PARTNAME
[1]  ⍝ FIND ALL SUPPLIERS OF A PART IN THE RELATIONAL DATA BASE
[2]   GET 'PARTSUPPLIER;(PARTSUPPLIER.PARTNAME =',PARTNAME,')'
[3]  ⍝ READ THE SUPPLIER NAMES FROM THE BUFFER
[4]   READ 'PARTSUPPLIER.SUPPLIERNAME'
    ∇

    ∇ PARTNAME REMOVEPAIR SUPPLIERNAME;EXPRESSION
[1]  ⍝ REMOVE A (PART, SUPPLIER) PAIR FROM THE RELATIONAL DATA BASE
[2]   EXPRESSION←'(PARTSUPPLIER.PARTNAME =',PARTNAME,') AND '
[3]   EXPRESSION←EXPRESSION,'(PARTSUPPLIER.SUPPLIERNAME =',SUPPLIERNAME,')'
[4]   DELETE 'PARTSUPPLIER;',EXPRESSION
    ∇
```

figure 3

The data structures used to implement relational data bases are the same as those used to implement hierarchic data bases: at the implementation level, a relational data base looks like a hierarchic data base with a number of one level hierarchies. Each relation looks like a root segment and each domain like a field in a hierarchic data base.

A student environment is considerably different than the environment that a commercial DBMS is designed for. When used for training purposes, the data bases will typically be small. Only small volumes of data will be transferred. The application programs will only be run a few times after they are debugged. These conditions are exactly opposite to those found in most "real" commercial data base usage.

EDBS was tailored specifically for this training environment. The EDBS implementation does not make the optimizations necessary to handle large volumes of data efficiently. However, EDBS is quite suitable for small volumes of data. Furthermore, students will find it both easy and economical to debug their programs in APL.

We believe that, as a training facility, EDBS is a distinct improvement on commercial DBMS's. As can be seen, EDBS has a full range of data manipulation commands and is representative of both hierarchic and relational systems. Therefore it can give students a wide range of experience. Students should find EDBS both easy and convenient to use.

Data Base Exercises and Games

In order for EDBS, or any other data base system, to be used as an effective educational facility the students must be given a number of effective programming exercises for the students.

The first aim of programming exercises should be just to teach the data manipulation language (DML). Therefore the first exercises should involve only simple queries and some simple modification commands with as little interaction between users as possible. In these first data base modification exercises a student should only insert, change and delete pieces of data which "belong" to him, perhaps identified by his student number. These first exercises should be designed to give a comprehensive view of the DML and yet not be so numerous or lengthy as to burden or bore the student.

Further exercises should require more complex actions with involved interactions between the students. The exercises should be varied and interesting. We also need a protocol for interaction between the students. We feel that data base games can be used as programming exercises to solve these problems.

In the type of game needed the data base contains the current state of the game. A player must query the data base to discover the state of the game in order to plan his play. He must modify the data base to actually participate in the game. The rules of the game will constrain the players in the actions they may perform. Therefore, in playing the game the student will gain experience at both retrieving information and modifying the data base using a DBMS. The game should also illustrate what a DBMS can and cannot do for its users.

For practical purposes we recommend that the games be designed to be played in an intermittent mode. The game should be scheduled to be played over a period of time on the order of one or two weeks. Each player should be allowed to make his moves intermittently at times convenient to him; there should be no requirement to play in turn. This mode of play should alleviate the problem of scheduling times when both terminals and the students are available.

We have develcped two data base games. One game simulates the stock market. An EDBS hierarchic data base is used to store bids, asks, the most recent prices, dividend notices and the players' holdings and cash resources. A person plays the game by looking at the offers and accepting one or posting his own offer. He then must make the appropriate modifications to the data base.

A program that browses through the bids for a company's shares is shown in figure 4. A student may use simple programs, such as the one in figure 4, to browse through the data base and then select bids and asks himself, or he may write more intelligent programs which make the selections for him. In either case he will learn how to manipulate the data base.

The second game is an inventory control game which uses a relational data base. Each student plays the part of a company which has a number of projects to complete. Each project requires some materials and produces others. Each player must buy the materials needed for his projects from the other players and sell his products to them. A player may also speculate in the commodities. The students must maniplulate the data base to do their buying selling and completing of their projects. A more complete description of the games can be found in [Klebanoff et al 1974].

```
     ∇ LOOKATBIDS COMPANYNAME
[1]    GU 'COMPANY, (NAME = ',COMPANYNAME,')'
[2]    →(STATUS≠0)/NOTFOUND
[3]    (15ρ' '),'COMPANY ',COMPANYNAME
[4]    ''
[5]    (READ 'OUTSTANDING');' SHARES OUTSTANDING'
[6]    'LAST PRICE:$';READ 'PRICE'
[7]    ''
[8]    'CURRENT BIDS ARE:'
[9]    ' PLAYER   AMOUNT  PRICE'
[10] CYCLE:GNP 'BID'
[11] ⍝ RETURN IF THERE ARE NO MORE BIDS FOR SHARES OF THIS COMPANY
[12]   →(STATUS≠0)/0
[13] ⍝ READ THE PLAYER NUMBER, QUANTITY, AND PRICE FIELDS FROM THE BUFFER
[14]   ((READ 'PLAYER'),(READ 'QUANTITY'));'$';READ 'PRICE'
[15]   →CYCLE
[16] NOTFOUND:COMPANYNAME,' IS NOT A COMPANY IN THE DATA BASE'
     ∇
```

FIGURE 4

Conclusion

We believe that EDBS is an excellent training facility for data base management system application programmers. EDBS is implemented in APL and its commands are accessed as APL functions. Therefore, the system is interactive and easily accessible. EDBS is useful because it is representative of a wide range of data base management systems and because it is designed to be efficient in a student environment.

APL, with the APL PLUS file system, has proven to be a very suitable host and implementation language for EDBS. It has provided us with good terminal support, a powerful file system and an easily learned host language.

A good set of exercises is as important in teaching data base management concepts as a suitable data base management system. The exercises must be instructive and interesting but should not place an undue burden on the student. We recommend the use of standard programming exercises to acquaint the students with the DML and recommend the use of data base games to train the students in using a DBMS to perform more complex data manipulations. Used with the recommended type of exercises, EDBS can be a very effective tool for teaching data base concepts.

Bibliography

[CODASYL 1971]
CODASYL Systems Committee, Data Base Task Group Report, April 1971

[Codd 1970]
E. F. Codd, "A Relational Model of Data for Large Shared Data Banks", CACM, Vol. 13, No. 6 (June 1970), pp. 377-381

[Codd 1971]
E. F. Codd, "A Data Base Sublanguage Founded on the Relational Calculus", ACM SIGFIDET Workshop -- Data Description Access and Control, 1971, pp. 35-68

[IMS 1971]
Information Management System IMS/360, Application Description Manual, GH20-0765-1 IBM, White Plains New York, 1971

[Klebanoff et al]
J. Klebanoff, F. Lochovsky, A. Rozitis, D. Tsichritzis, EDBS User's Manual, CSRG Technical Report #40, University of Toronto, 1974

[Lochovsky 1973]
F. Lochovsky, An Educational Data Base Management System, MSc thesis, Department of Computer Science, University of Toronto, 1973

[Lowenthal 1971]
E. I. Lowenthal, A Functional Approach to the Design of Storage Structures for Generalized Data Management Systems, PhD dissertion, University of Texas at Austin, August 1971

[VANDL/1 1973]
VANDL/1 -- Vancouver Data Language One, F.R.P.Q. Description and Operation Manual (Second Edition), SC09-0007-01 IBM, February 1973