PANDA:
A Pascal Network Data Base Management System

André Frank

Institute of Geodesy and Photogrammetry
Swiss Federal Institute of Technology (ETH)
Zurich, Switzerland

1. Introduction

PANDA is a set of database programmes entirely written in Pascal and especially suited for small (micro) computer systems; it was developed with interactive application and computer graphics in mind.

PANDA is based on the network approach to data management, as it is proposed by CODASYL |CODASYL 71|. PANDA is therefore similar to commercially available systems on main frames such as e.g. DBMS from Digital Equipment Corp. The programmes interface of PANDA was specially adapted to the use within Pascal programmes. Its use seems much easier for the programmer than the CODASYL proposal.

PANDA is small enough to be used on small systems. The code for a Motorola 6809 micro-processor is about 15 kbytes. PANDA, especially its buffer management, is further adapted to small computer systems.

PANDA is, however, not a system for the fast organization of data with little strucuture and does not feature an interactive query language or a report generator. PANDA is a tool for the programmer to facilitate the structuring of complicated data, not a utility for the end user.

2. Goals of PANDA

We wrote PANDA in order to facilitate the programming task of a group of researchers, working at geometric modelling including sophisticated graphical representation. The data structure we encountered seemed complex enough to use a complete database management system.

Through our experience with a CODASYL type DBMS (DBMS-10 from DEC) on similar problems we found that programming with such a system in COBOL is more difficult than using Pascal without this aid. We intended therefore to combine the advantages of the DBMS with the Pascal programming language.

In order to be able to transfer our programmes to small systems we needed a compact and easily transportable DBMS. We decided therefore to write it as far as possible in Pascal and make it as simple as possible.

Finally, we had to consider performance; we need efficient methods for data access in order to draw pictures directly from the stored data in the database. The goal was to be able to dispense completely with additional in programm data storage.

3. Reasons for the network data model

Most DBMS on small computers are based on the relational data model |Lewis 81|. The relational data model is based on a few simple concepts and theoretically well understood. |Codd 81| It has been used for some implementations on mainframes (INGRES, ORACLE, SQL) but generally the requirements on hardware are great and performance sometimes a problem.

The network model was designed in the late '60s considering more the hardware requirements than mathematical clarity. For data with complex structure and for treatments which rely heavily on this structure the network data model leads to probably easier implementations and higher performance |Frank. 82|.

An example: We consider two data elements, the one describing an employee and the other a departement:

employee	department
emp-nr, name, dep-nr	dep-nr, dep-name

If we are interested in knowing the name of the department the employee works in, then the most common implementation of the relational data model looks for a department with the same dep-nr as is registered in the employee's record using a B*-tree |Knuth 73|, giving two or more accesses to disk. The network implementation stores with the

employee record a pointer to the department and needs therefore only one access to disk. It is well known that the number of disk accesses are the most important factor influencing the performance of a DBMS.

4. Pascal as a programming language for a DBMS

The intention from the very beginning was to write PANDA completely in Standard Pascal |Jensen 74|. Of course commands for random accessing of files must be present, either as an extension to Pascal (as many Pascal compilers feature nowadays) or as calls to some functions written in another language.

A DBMS entirely written in Pascal would be a very valuable asset in order to be able to transfer programmes to different hardware. We intentionally renounced from any changes in the Pascal compiler in order to maintain portability and we were ready to pay some price for this advantage.

In order to be able to treat the different record types as they appear to the application programmer in the same way within PANDA, they must be declared as variant parts of an element record. This has the disadvantage that all records require equal amount of space, which means waste of space if some are smaller than others, but simplifies the whole data management extremely - the most trivial algorithm for buffer management may be used. This makes programmes shorter and running faster.

5. Influence of Pascal as an application programmer's language

The application programmer's interface proposed by CODASYL is suited for the embedding of the data manipulation language (DML) within COBOL (as an extension of the COBOL compiler).

It is well known, that this interface is not very easy to use for the application programmer. Reconsideration of the interface for a DBMS to be used in Pascal is therefore necessary.

All data manipulation language calls are functions which are evaluated as 'true' if they have been carried out completely - otherwise they are evaluated as 'false' and return in a field an exception code. This seems a syntactically easy solution.

Data manipulation language statements usually return a record requested but in network databases (sometimes called 'navigational' |Bachmann 73|) they are often position dependent and need some input to indicate the actual position in the data base, e.g. 'find next' needs input of the actual position in the set. This led in CODASYL to the two concepts of user working area where data are exchanged between the programme and the data base and the cursors which maintain position information for later data manipulation commands. Apparent problems of programmers using these concepts made us look for an idea easier to unterstand. The solution in PANDA tries to unify these two concepts in the element record which has a double function:

- it contains, in the variant part, the actual data of the user programme,
- it is an indicator of an actual position in the data base space.

This double function is convenient for the user: whenever something is needed he passes an element record and, depending on the call, either the data part is used or it is interpreted as a position. This makes the 'current' registers and the complex rules of updating them in the CODASYL proposal superfluous. As TYPE declarations for the data in the data base are automatically inserted in the programme, the programmer may declare as many element records as he needs in a procudure.

Comparisons of code written with a CODASYL DML and PANDA show that PANDA needs considerably less code (1/3 in a representative example).

6. Buffer management

Performance of a DBMS is primarily dependent on disk accesses (using at least 30 msec each). To minimize physical disk accesses all DBMS maintain some part of the database not only in the files but also in the main memory (buffer). The management of this buffer is crucial to performance. The goal must be that as often as possible a record needed can be found in this buffer and no physical access to disk is necessary.

In a graphical application very often a picture has to be redrawn: more or less the same data is used again. A clever buffer management could avoid additional disk I/O.

Most known DBMS's divide the data file in pages. Pages are the units that are brought in buffer, kept there and finally copied back to the file, mostly using a 'least recently used strategy'. Unfortunately, pages are relatively large (1 kbytes as a minimum) so only a few may be kept in buffer at any time. Very often a page contains only one or few records useful for the task at hand - so in much space only few useful data are stored.

PANDA therefore is based on a completely different idea. The buffer contains element records, not pages |Kaehler 81|.

A record is brought in when needed and only this record's space is used. Consequently, the buffer space can be filled out completely with useful data; more records can be kept in buffer and fewer accesses to disk are necessary. In the graphic application we developed PANDA for, we expect to keep one complete picture in buffer and to be able to redraw a picture without any access to disk.

This type of buffer management facilitates

the realisation of the transaction concept.

An atomic transaction is a bundle of operations of which either all or none are carried out. This is a very important concept to maintain data base integrity: either a change in data is made completely or not at all - no intermediate states are ever visible to the user |Lampson 81|.

In PANDA only the data in the buffer is changed during a transaction. At the end of the transaction, in one action, either all the changes in the buffer are copied in the file or the transaction is abandoned and the changes in the buffer are thrown away. This is similar to the shadow page mechanism |Reuter 81| but much simpler and faster.

7. Data description

The data description in PANDA could be more comfortable. The application programme is requested to prepare some simple Pascal procedures for inclusion in the PANDA source. The PANDA system contains a preprocessor (also written in Pascal) for inclusion of files in a Pascal source file. Some of the files are also used for inclusion in the application programmes.

After the data description is made, the PANDA routines can be compiled (if the Pascal compiler supports separate compilation). This is advantageous compared with the usual approach in DBMS whereby the routines are compiled beforehand and the adaptation to the task at hand is through data values. The compilation with the data declaration makes possible the inclusion of some application dependent values in the executable code.

8. Data manipulation

PANDA features commands for data manipulation, mostly similar to the CODASYL proposal, although the combination of cursor and user working area reduces the number of commands. Here only a short overview:

environment:

Initialise the PANDA system
terminate the PANDA system
end transaction (commit)
abort transaction

find a record in the data base

find next in set
find owner in set
find element based on given data values (global)
find element within set, based on given data values

update the data base

- store new record
- modify data values
  (includes possible changes in position to maintain set order)
- delete record
- insert a record in a set
- remove a record from a set

9. What can PANDA be used for

In a great number of applications the data structure and the procedures to maintain this structure claim a great part of the programmer's total effort. This seems true for such differing problems as programmes for financial analysis and other accounting problems, compilers, geometric modelling and graphics.

In all these cases we can identify data elements which occur in unknown number and which are related to each other:

- departments containing employees
- houses identified by addresses (street, house number) but also further defined by co-ordinate values for their corners.

In all these cases programming may benefit from the data structuring that can be defined before coding starts and which is afterwards enforced for all programmers. Furthermore all the routines for maintaining the data are already written, including the management of buffers in order to keep always the data most needed in the main memory.

Furthermore the PANDA code is well structured. It is easy to adapt it to an application's special needs.

Literature

|Bachman 73| Bachman, W.C., The programmer as a navigator, ACM Comm. 1973

|CODASYL 71| CODASYL Data Base Task Groupe (DBTG) Report, April 1971

|Codd 81| Codd E.F., Relational Database: A Practical Foundation for Productivity, ACM Comm., Vol. 25, No. 2, p. 109, Febr. 1982

|Frank 82| Frank A., PANDA, Bericht, Institut für Geodäsie und Photogrammetrie, ETH Zürich, 1982

|Jensen 73| Jensen K., Wirth N., Pascal, User Manual and Report, Springer Verlag 1974

|Kaehler 81| Kaehler T., Virtual Memory for an Object-Oriented Language, BYTE, Vol. 6. No. 8, p. 378, Aug. 1981

|Knuth 73| Knuth D., The art of computer programming, Vol. 3, Sorting and Searching, Addison Wesley 1973

|Lampson 81| Lampson B.W., Paul M., Siegert H.J. (Eds.) Distributed Systems - Architecture and Implementation, Springer-Verlag 1981

|Lewis 81| Lewis, Proceedings of the SIGSMALL-SIGMOD conference 1981

|Reuter 81| Reuter A., Fehlerbehandlung in Datenbanksystemen (Datenbank-Recovery), Hauser Verlag, München 1981.