# ARCHITECTURE OF FUTURE DATA BASE SYSTEMS

Lawrence A. Rowe and Michael Stonebraker
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720

## 1. INTRODUCTION

High level user-friendly data base management systems (DBMS) are generating wide spread interest. At issue, however, is the best architecture of such systems and what (if any) lower level interfaces to make visible to application programmers.

In this paper we first discuss three typical programming interfaces which future DBMS systems may support. These interfaces are: 1) non-procedural, set-oriented (relational), 2) navigational (CODASYL), and 3) access method. After briefly comparing the language levels provided by each interface, four data base system architectures are described which can support a high level interface on top of one of the other two interfaces. We believe these architectures are the only reasonable candidates for future DBMS packages.

The remainder of the paper identifies the tradeoffs between the architectures. The discussion concentrates on the data manipulation aspects of a DBMS rather than on issues of concurrency control, protection, crash recovery, audit trails and integrity control. We recognize that these features are an important component of any data base facility, but they can be implemented in any of the four architectures.

The general conclusion of the paper is that each architecture offers advantages, which implies that DBMS implementors will have some hard choices to make. Moreover, because systems based on the candidate architectures will be (or are) commercially available, users selecting a DBMS will face roughly the same hard choices.

## 2. THREE DBMS INTERFACES

Figure 1 depicts the environment in which data base requests are made. An application program or interactive user generates requests to the run-time DBMS. The DBMS processes the request and returns data and in some cases status information indicating that a request could not be processed (e.g., an update might not be performed because the affected

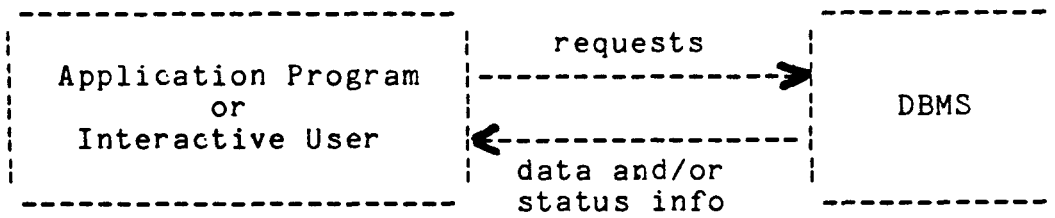


Figure 1. Data Base Management System Environment

data was not in the data base).

Requests can be passed to the DBMS in a high, intermediate or low level representation. These levels are illustrated by examining how one small example would be coded in each representation. The example uses a personnel data base for a company. The query is to find which employees in the public relations department have a job title of ad writer. The data base description and query for a relational interface are shown in Figures 2 and 3, respectively.

Three important points about any high level query language are illustrated by this example. First, the user presents his query to the DBMS in a form which describes what is to

```
EMP(name, dept#, jobTitle, ...)

DEPT(dept#, deptName, ...)
```

Figure 2. Relational Schema for Personnel Data Base

```
retrieve EMP.name
where DEPT.deptName = "public relations" and
    EMP.jobTitle = "ad writer" and
    DEPT.dept# = EMP.dept#
```

Figure 3. Sample Relational Level Query

be returned, rather than how to find the answer. Second, only one request must be passed to the DBMS. In contrast, a low level access method interface may require one request for each DEPT record until the public relations department record is found, and then one request for each EMP record. Third, all employee records that satisfy the query will be returned by a high level interface in response to the one request while only one record, one that might not qualify, is returned by a low level interface.

Figures 4 and 5 show the same example coded for a navigational interface using a stylized version of the CODASYL data manipulation language (DML). The function EXISTS indicates whether a record was returned by the last retrieve-statement. Notice that DML statements are intermixed with host language statements. One request is passed to the DBMS to find the public relations department record and one to find each ad writer employee record. In other words, if

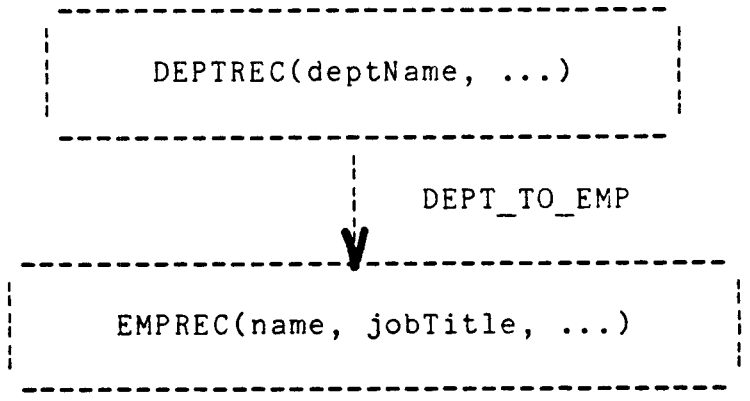


Figure 4. CODASYL Data Structure Diagram for Personnel Data Base

```
retrieve DEPTREC where deptName = "public relations"
if EXISTS(DEPTREC) then
    while not EMPTY(DEPT_TO_EMP) do
        retrieve EMPREC within DEPT_TO_EMP
            where jobtTitle = "ad writer"
            /* found ad writer */
    end
```

Figure 5. Sample CODASYL Level Query

there are n ad writers, at least n+1 DBMS requests must be made.[1] We consider this an intermediate level interface.

The point is not that the network data model is inappropriate; rather that the CODASYL DML is not a high level interface. A high level interface similar to that suggested by Figure 3 can be written for a DBMS that supports a network data model.

Figure 6 shows our sample query written in an access method level language. The query is based on records stored in DEPT and EMP files. Notice that in the worst case a request is made to the DBMS for each record stored. As in an intermediate level interface, only one record is returned for each request.

The comparison of language levels is summarized in Figure 7. The data association category refers to where relationships between data items, for example, who is the manager of a department, is represented.

```
retrieve DEPT
while EXISTS(DEPT) and
  DEPT.deptName not= "public relations" do
    retrievenext DEPT
retrieve EMP
while EXISTS(EMP) do
    if EMP.jobTitle = "ad writer" and
      EMP.dept# = DEPT.dept# then
        /* found ad writer */
    retrievenext EMP
end
```

Figure 6. Sample Access Method Level Query

[1] More requests will be required if the data base does not provide access paths for directly retrieving department records. Moreover, if the predicate in the employee record selection were changed slightly (e.g., salary>25K) the member records in the DEPT_TO_EMP set would have to be searched sequentially.

| | Relational | CODASYL | Access Method |
|---|---|---|---|
| # DML requests | 1 | 1 (or more) per record retrieved | 1 (or more) per record retrieved |
| data retrieved | set of records | 1 record | 1 record |
| language level | high | medium | low |
| data association | in DML | partly in DML, partly in host language | in host language |

Figure 7. Comparison of Language Levels

## 3. POSSIBLE ARCHITECTURES

To support a high level language interface, such as the relational one above, on top of lower level interfaces, such as the intermediate level CODASYL or a low level access method, there are four reasonable architectures.

1. A high level interface can be supported on top of an intermediate level interface, such as a CODASYL system. Programs can be written for either interface, and a language processor invoked to translate the high level specification to a low level program (see Figure 8). The query language for UNIVAC's DMS-1100 is an example of this architecture [UNIVAC].

2. A high level interface can be supported on top of a low level access method interface. Again, a language processor is invoked to process the high level language. This architecture is illustrated in Figure 9. TANDEM's data base system ENFORM is organized in this fashion [TANDEM].

3. A high level interface can be supported on top of an interface which is not visible to an application programmer. No commitment is made to an internal low level DML, but presumably one exists. An example of this approach is INGRES (see Figure 10) [Stonebraker 76].

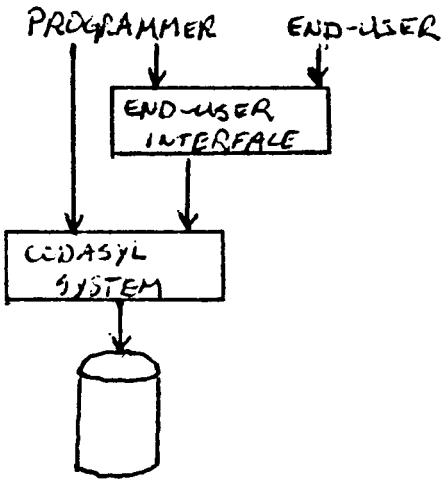


Figure 8. CODASYL Oriented Architecture (Option 1)

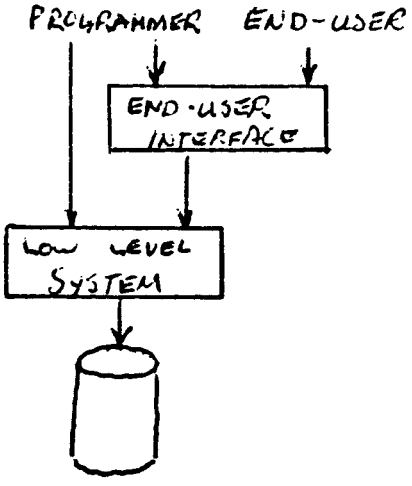


Figure 9. Another Two Layer Architecture (Option 2)

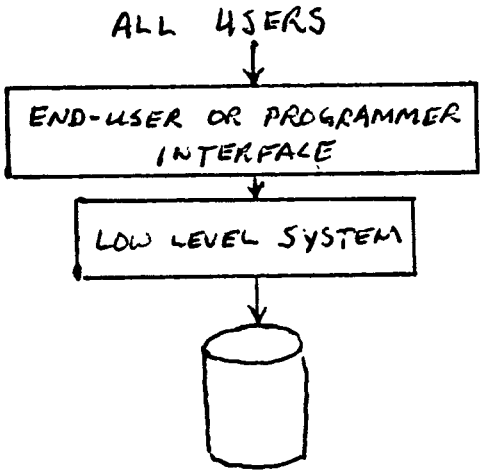


Figure 10. The INGRES Architecture (Option 3)

4. A high level and an intermediate level interface can be supported on top of a low level interface. In this architecture two language processors are needed and the low level interface must handle both end user languages. Figure 11 shows an example of this architecture. To the best of our knowledge, no working example of this architecture exists.

Two other possible architectures are not considered. One is a navigational interface on top of a high level interface. This architecture would be slow and clumsy because it is like simulating an assembly language on top of PL/1. The other is to present all three interfaces, one on top of the other. In other words, change option 4 so that the end user system interfaces to the CODASYL system rather than directly with the low level system. This architecture is not considered further because it offers approximately the same advantages and disadvantages as option 4.

To motivate the points of comparison between these alternatives, in the next section we discuss some assumptions about future software and hardware. The evaluation of the four alternatives is presented in Section 5.

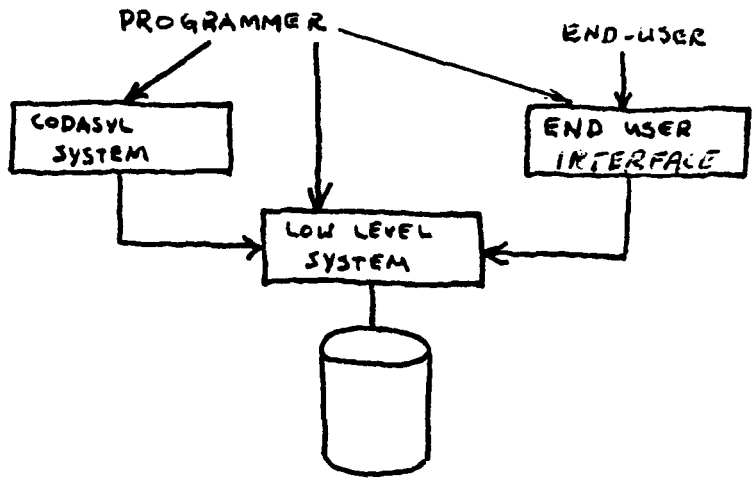


Figure 11. A Final Architectural Option (Option 4)

## 4. THE NATURE OF THE FUTURE

This section contains a list of assumptions about hardware and software whose impact on the design and use of systems in the future must be considered. We believe most of the assumptions are self-evident and so we present them without much discussion.

### 4.1. Increased Complexity

The major trend in user needs will be to maintain more information on-line and to develop more complex applications which access shared information. More data bases will be integrated and they will contain a greater range of data which will be accessed by a more diverse user community. Because a high level language is the most satisfactory tool for developing these complex applications, we are only interested in data base architectures which can support a high level DML.

### 4.2. Distributed Data Bases

Current trends toward geographically distributed data bases will continue. For some applications the advantages of

distributing the control of the data base toward the ultimate user or creator will outweigh the advantages of a large centralized facility with remote access.

## 4.3. Hardware Advances

Clearly, the current cost and performance trends in CPUs and main memory will continue and, as a result, microprocessors will be nearly free. For the next decade disks will remain the predominant storage medium for data bases, except perhaps for data base catalogues and/or small data bases which may be stored in semiconductor memories.

The major hardware advance that will impact data base systems will be the likely viability of one or more of the data base machine architectures being proposed by the research community. The impact of these special devices may be even greater after 1985 when millions of bits of random access memory and several microprocessors will be available on a single chip.

## 4.4. Standards

It appears that national standards for data base systems will evolve over the next decade. The proposal currently being considered for adoption was developed by CODASYL and includes a navigational DML for a network data model.

If it is adopted, some manufacturers may not adopt the standard because they currently have successful products which are incompatible or because they question the technological superiority of the standard. On the other hand, the existence of a standard may influence the design of future data base systems. Architectures which make it easy to implement products that meet the standard may be choosen even though other alternatives may be better from a technological viewpoint.

We see the issue of standardization as essentially a marketing issue rather than a technological one [Stonebraker 80]. The focus of this paper is on technological issues, although we do discuss how difficult it would be to implement a navigational interface on each of the proposed system architectures.

# 5. EVALUATION OF THE ALTERNATIVES

This section assesses the impact of choosing each of the architectures listed in Section 3. The architectures are compared with respect to single machine efficiency,

distributed data bases, data base machines, software complexity and standardization.

## 5.1. Single Machine Efficiency

Three of the four architectures support a visible intermediate or low level DML interface. Programs for which efficiency is a very serious issue can be coded against this interface; others can use the high level facility. Only option 3 does not support a visible low level interface and thus will suffer a performance penalty. This penalty arises from at least two sources:

a) The algorithm used by the data base system to translate a high level command into a sequence of low level commands is inferior to the one that would be chosen by a clever programmer.

b) There is extra overhead associated with executing the sequence of low level commands because the code generated by the data base system to manipulate the data once it has been fetched is less efficient than that which a clever programmer could write.

The first penalty arises because the data base system may choose an inferior algorithm and fetch more data than the clever programmer's algorithm. On the other hand, if both choose the same algorithm, the clever programmer will presumably still win because he is manipulating qualifying records in a high level programming language (e.g., COBOL, FORTRAN, PL/1, etc.) whereas the data base system is generating general-purpose code to accomplish the same functions. The general purpose code is usually slower. One research group that implemented a prototype data base system reported that the performance penalty due to b) above was about 15% [Blasgen 78]. These penalties are application and programmer dependent.

In any case, architecture option 3 will pay a performance penalty of about 15%. Assuming that there is no performance penalty associated with an intermediate level interface compared to a low level interface, the other options are equally attractive.

## 5.2. Distributed Data Bases

All options, except option 3, will suffer a performance penalty when accessing distributed data bases because they have a record-at-a-time data base system interface that must be supported. For example, the sample query presented above

is translated into a sequence of commands each fetching a single employee record.

Suppose the public relations department employees are present on 3 different machines in a network as shown in Figure 12. Lower level commands must be sent from the originating machine to the other two machines and responses received. The number of messages which must be transmitted approaches one per public relations department employee (or more).

The alternative is to send the application program to each of the other sites. Although such an option is possible, dynamic migration of programs is very difficult. Also, the bookkeeping becomes quite complex if the transmitted program does not finish execution on the current machine and it had to be transmitted to another site, and so forth.

Option 3, on the other hand, supports only a high level interface which means that only one command ("retrieve public relations department employees who have a job title of ad writer") is sent to each site. This means the message traffic will be one multidestination transmission followed by transmission of only the qualifying records rather than a separate multidestination transmission to request each employee record and transmission of perhaps more records. In distributed systems in which message transmission

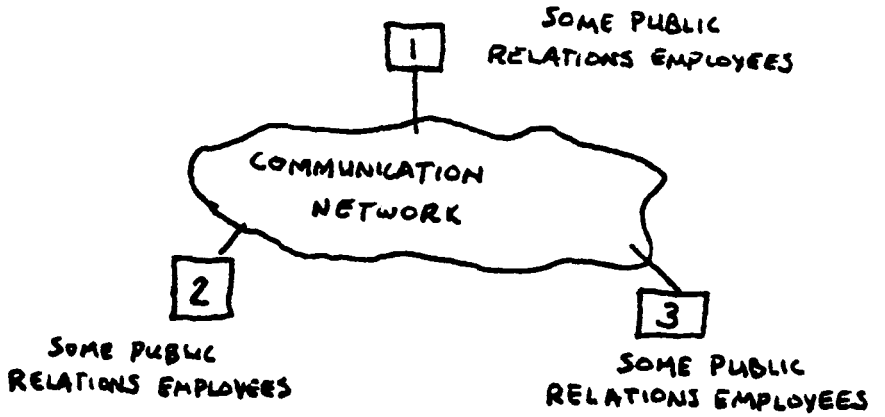


Figure 12. A Distributed Data Base Environment

dominates other costs (e.g., long-haul networks such as the public packet-switched networks or in local networks with many processors contending for the transmission capacity) this option will be more efficient.

In the other architectures both high and low level commands are transmitted. Hence, programs coded against the high level interface can use the strategy suggested for option 3; other programs would transmit low level commands. This may avoid some of the performance penalty for options 1,2 and 4, but only at an increase in data base system complexity.

## 5.3. Data Base Machines

Virtually all data base machine proposals depend on some form of distributed processing [Banerjee 78, Copeland 73, DeWitt 78, Schuster 78, Stonebraker 79]. The general idea is illustrated in Figure 13. A data base query by an application program is translated into one (or more) requests that can be processed by the data base machine. This request is processed by one or more of the dedicated "back-ends" without further intervention by the host machine. The hardware and/or software for the back-end machines is special purpose and consequently can be made more efficient

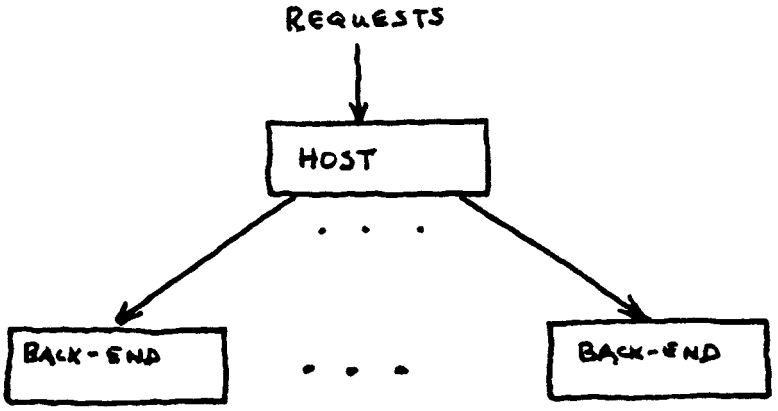


Figure 13. Back-End Machine Architecture

than the host machine.

We claim that only option 3 can make effective use of this architecture because it takes 3000-5000 (or more) host instructions to set up a call to the back-end(s) and to process the return(s). This overhead includes: (1) issuing a system call from the application program, (2) checking the validity of the request, (3) writing the request to the back-end(s), (4) switching to run another process, (5) receiving the back-end(s) response(s), (6) restarting the application program that made the request, and (7) communicating the back-end(s) response. With comparable overhead the host can process many low level requests directly (e.g., retrievenext). Thus, the host can in many cases service the request as efficiently as an auxiliary processor.

Because options 1, 2, and 4 process low level commands, the above overhead will prohibit them from making effective use of the back-ends. Only option 3 can pass high level requests to the back-end. To use the public relations department example again, option 3 can process the request for the query with one call to the back-end. The communication overhead is incurred only once. The other options incur the overhead approximately once per employee retrieved.

Although it may be cost effective to off-load the work of a low level interface onto a back-end machine, a high level interface will be more effective in utilizing back-end processors. Notice the similarity between the discussion for data base machines and that for distributed data bases.

## 5.4. Standardization

Assume that a data base system standard similar to the CODASYL proposal being considered is adopted and that a marketing decision is made to offer a product that meets the standard. What impact would this have on the choice between the alternative architectures?

Options 1 and 4 would be easy to extend to meet the standard because they have a definite interface in the architecture which corresponds to the standard. Option 2 would also be relatively easy to extend by implementing an interface that meets the standard, essentially turning option 2 into option 4.

Concerning option 3, the possibility of simulating a low level interface on top of a high level one was rejected earlier. The other alternative is to turn option 3 into option 4. Without careful preplanning, this transition could be very difficult. For example, data bases created through one interface should be accessible through the other interface. This requires that the access method implementations of the

two interfaces be similar. Moreover, the higher level interfaces must provide consistent external views of the database. Work on these problems has yielded only partial successes [Zaniolo 79].

Options 1 and 4 are attractive; option 2 is less attractive; and, option 3 is by far the least attractive.

## 5.5. Software Complexity

Option 3 should be the least complex data base system implementation. It will be simpler than option 2 because no commitment need be made concerning how data base services are provided (e.g., crash recovery and integrity control). In option 2 such services must be provided by the low level interface. Consequently, either the implementation of the data base system is constrained as to where services are provided or they must be provided twice, once for each interface. In either case, the result is likely to be increased system complexity.

Option 2 is simpler than option 4 because only one language interface is needed. Lastly, depending on what end user languages are supported, option 1 or 2 might be simpler. In general, it is expected that the interface in 2 might be cleaner than the interface in 1, if only because it may not have been designed by committee.

Thus, software complexity (and hence maintenance costs) increases from 3 to 2 to 1 to 4.

# 6. CONCLUSIONS

The foregoing discussion suggests that option 3 is best if standards are not an issue and a performance penalty can be tolerated. Moreover, it may be that the performance issue will assume less importance when data base machines are introduced.

If compliance with a standard is important, options 1 and 4 are best (1 being preferable to 4 because of code complexity). Consequently, options 1 and 3 appear to dominate. They are, unfortunately, incompatible in that option 3 is incompatible with a possible national standard while option 1 is incompatible with distributed data base and specialized hardware technologies.

## ACKNOWLEGEMENTS

We want to thank Brad Wade who read an earlier draft of this paper and made several suggestions to improve it.

## REFERENCES

[Banerjee 78] Banerjee, Jayanta et. al. Concepts and capabilities of a database computer. TODS, 3, 4 (Dec. 1978).

[Blasgen 78] M. Blasgen. Personal communication.

[Copeland 73] Copeland, G. P. et. al. The architecture of CASSM: a cellular system for non-numeric processing. Proc. First Ann. Wkshp on Comp. Arch. (1973).

[Dewitt 78] Dewitt, D. J. DIRECT - a multiprocessor organization for supporting relational data base management systems. Proc. Fifth Ann. Symp. on Comp. Arch. (1978).

[Schuster 78] Schuster, S. A., et. al. RAP.2 - an associative processor for data bases. Proc. Fifth Ann. Symp. on Comp. Arch. (1978).

[Stonebraker 80] M. Stonebraker. A tale of two standards. Working paper (Aug. 1980).

[Stonebraker 79] Stonebraker, M. MUFFIN: a distributed data base machine Proc. First Ann. Conf. on Dist. Comp. Huntsville, AL (Oct. 1979).

[Stonebraker 76] M. Stonebraker, et. al. The design and implementation of INGRES. Trans. on Database Sys., 1, 3 (Sept. 1976), pp. 189-222.

[TANDEM] ENFORM reference manual. Tandem Computers, Inc.

[UNIVAC] Data management system (DMS 1100) data manipulation language programmer reference. Document UP 7908.

[Zaniolo 79] C. Zaniolo. Design of relational views over network systems. Proc. ACM-SIGMOD Conf. (May 1979).