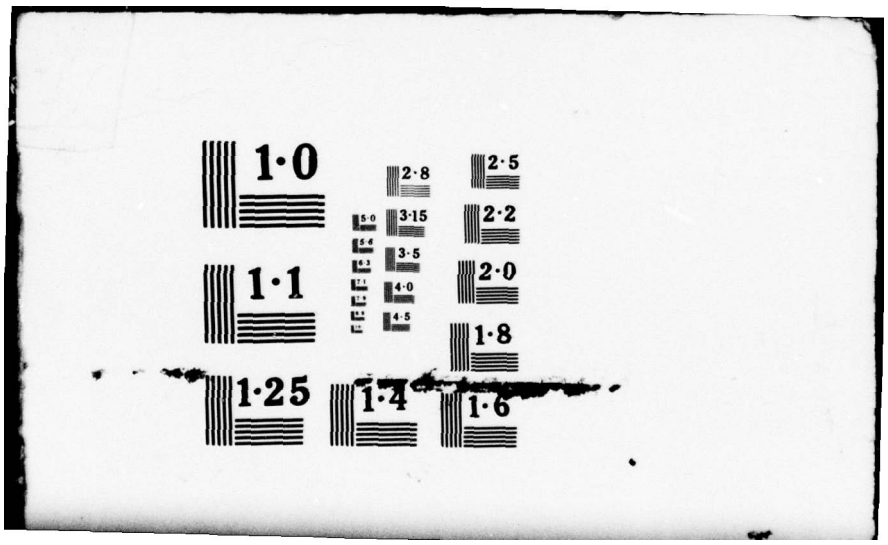
AD-A067 249 WHARTON SCHOOL PHILADELPHIA PA DEPT OF DECISION SCIENCES F/G 9/2
EXPLORING THE CONCEPT OF A CODASYL DATABASE MACHINE.(U)
JAN 79 R D HACKATHORN N00014-75-C-0440
UNCLASSIFIED 78-12-04 NL

1 OF 1
AD A
067249

END
DATE FILMED
6-79
DDC


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0

Wharton

Department of Decision Sciences

12

LEVEL II

DDC
RECEIVED
APR 12 1979
C

AD A067249

DDC FILE COPY

University of
Pennsylvania
Philadelphia PA 19104

This document has been approved for public release and sale; its distribution is unlimited.

EXPLORING THE CONCEPT OF A CODASYL DATABASE MACHINE

Richard D. Hackathorn

Working Paper 78-12-04
January, 1979 -- Draft #2

Department of Decision Sciences
The Wharton School
University of Pennsylvania

[This research was sponsored in part by the Office of Naval Research Grant Number N00014-75-C-0440.]

79 04 09 172

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: 78-12-04
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): EXPLORING THE CONCEPT OF A CODASYL DATABASE MACHINE.

5. TYPE OF REPORT & PERIOD COVERED: Technical Report. Jan 1978-Dec 1978

6. PERFORMING ORG. REPORT NUMBER: 78-12-04

7. AUTHOR(s): Richard D. Hackathorn

8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0440

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Department of Decision Sciences
University of Pennsylvania
Philadelphia, PA 19104

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 34p.

11. CONTROLLING OFFICE NAME AND ADDRESS:
Office of Naval Research
Department of the Navy
800 N. Quincy St., Arlington, VA 22217

12. REPORT DATE: January 1979

13. NUMBER OF PAGES: 31

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

see next page.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

408757 LB

ABSTRACT

This paper explores the concept of a "database machine" using the approach of the CODASYL data model. A database machine is defined as an integration of hardware and software for providing generalized database management capability in a physically separate device. The advantages of a database machine, along with functional specifications, are presented. Next, an illustration of using a database machine is given through an example of invoice processing using the SEED database management system on a DECsystem-10 computer. Finally, the implications of several design alternatives arising from the illustration are discussed.

ACCESSION for
NTIS White Section
DDC Buff Section
UNANNOUNCED
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and or SPECIAL
A

ABSTRACT

This paper explores the concept of a "database machine" using the approach of the CODASYL data model. A database machine is defined as an integration of hardware and software for providing generalized database management capability in a physically separate device. The advantages of a database machine, along with functional specifications, are presented. Next, an illustration of using a database machine is given through an example of invoice processing using the SEED database management system on a DECsystem-10 computer. Finally, the implications of several design alternatives arising from the illustration are discussed.

OUTLINE

1.0 INTRODUCTION

Database management systems (DBMS) have emerged in recent years as an important component of all large-scale information systems. This trend stems from a shift in perspective of data as simply the input/output of programs to data as a central focus of information processing. Increasing portions of the resources devoted to information systems are being allocated to the database management function.

Database management is distinguished from other simpler forms of managing data by the extent to which complex data structures are supported. File management systems provided simpler ways of accessing data through various access methods, such as index sequential. Although database management systems may use these access methods internally, the emphasis is on separating the definition of the data structures from the manipulation of the data, down to the level of each data item.

With the increased resources being devoted to database management systems, various approaches to satisfying the functions of a database management system have been tried. Numerous commercial DBMS packages have been developed and are in wide use (e.g., IMS, TOTAL, IDS, System/2000, IDMS, SEED, ADABAS, DBMS-10). An effort to standardize a uniform interface between the application program and the DBMS by

the CODASYL Data Base Task Group (DBTG) has made significant advances and has influenced many of the above commercial packages [CODASYL, 1971].

This paper will concentrate on database management from a slightly different standpoint. The focus of this paper is: Is it possible to build a machine that performs all the functions of a database management system using small-scale computer technology and the CODASYL data model?

Such a machine is referred to in this paper as a "database machine", although the literature often uses similiar terms, such as "back-end processor", or "database computer."

Consider the following definition for a database machine: A database machine (DBM) is an integration of hardware and software components to provide a generalized database management capability in a physically separate device. The hardware components consist of a large capacity storage device, based on a hard disk, and a processor. The software components consist of a database management system, operating system, and language translators. In the literature there is considerable debate on what should be implemented in hardware and what in software. To the author, this balance should be based solely on performance considerations and capabilities of existing technology. The intent or functions of the database machine remain the same.

The conventional approach to supporting a DBMS is by a large software package running on the host processor under its operating system and performing operations on some large-scale disk storage unit connected to the host processor as a peripheral. This case is referred to as a "host resident DBMS". The shift to a database machine implies that a separate processor is added and is dedicated to support only the database management function. When compared with the functions performed by the host processor, a database machine is, in effect, "off-loading" or distributing the processing functions with a special purpose peripheral. The contrast between a conventional DBMS and a DBM is shown in Figure 1.

The concept of a database machine is not new. The first paper on the subject appeared four years ago [Canaday et.al. 1974]. This paper was a report on an experimental database machine implemented by R. H. Canaday and others at the Bell Telephone Laboratories. Within the last year, excitement on the topic has risen considerably. All of the major conferences on database management have presented topics directly concerned with database machines. This excitement has been caused by dramatic decreases in the cost of computer hardware, along with advances in associative memory techniques based on magnetic bubbles or LSI technology. Good literature reviews are given in Baum [1976], Berra [1977], Hsiao [1977], Lowenthal [1977], and Mohan [1978].

Recent articles in Datamation [Champine, 1978; Bray & Thurber, 1979] provide good overviews of the approaches and major efforts related to database machines.

1.1 ADVANTAGES

This section presents some of the advantages to the database machine approach over conventional approaches. It relies on the initial work by Canaday et. al. [1974].

The first advantage to the DBM approach is on the economy of a device that specializes in supporting the database management function. The host processor must be sufficiently generalized to process a wide variety of programs. Much of the overhead in the operating system and access methods can be trimmed in this fashion. The underlying assumption is that the savings resulting from the economy of specialization are greater than the economics of scale inherent in the host processor. With the trend towards decreasing hardware costs, this assumption will be increasingly valid through time.

The second advantage is enhanced transfer of databases. A continuing problem has been the transfer of data from one computer system to another, especially if such systems are from different vendors. This transfer is usually accomplished by an "unload/load" operation (i.e., taking a structured database, flattening the hierarchies into

sequential records, writing the data onto a magnetic tape, and finally reversing the procedure on the second computer). A database machine, however, can effect the transfer of data by its commonality of its database interface. As shown in Figure 2, the first host processor can construct a database on its database machine. That database can be transferred, as is, to a second compatible database machine so that the database is now available to the second host processor. No reformatting of the data is necessary.

A third advantage is data sharing. As shown in Figure 3, several host processors can share a common database machine. The host processors can be physically separated and may be from different vendors. A extension of data sharing is shown in Figure 4, in which one or more of the hosts are physically remote and communicate with the database machine via telecommunications lines. A further extension, as shown in Figure 5, is for several database machines to be interconnected with several processors through some kind of network facility. In this situation, there is a functional differentiation between nodes that are concerned with the processing of data for a certain application (i.e., processor nodes) and nodes that manage a specific collection of data (i.e., data nodes).

As will be shown below, these advantages are contingent on certain internal design alternatives in unobvious ways.

1.2 FUNCTIONAL SPECIFICATIONS

To have the database machine provide all the functions of current DBMS packages on large mainframe computers, the database machine has to perform more functions than simply manipulating the contents of a database. Shown in Figure 6 is a diagram of the typical processing flow with conventional database management systems. Roughly, the initial phases of the development of a database application deal with the definition of the database structure. This is followed by the compilation of the application program and finally by the execution of the application program with the data manipulation routines of the DBMS. Throughout the development and operation of the database application, there is a need to perform various utility functions, such as calculating statistics on database growth.

Figure 7 constrasts with Figure 6 by showing the development flow using a database machine. Various functions, such as composing the schema definition and editing, compiling, and executing the application program remain as part of the host processor. However, other functions, such as the actual processing of the schema definition, are handled within the database machine. The various functions performed by the database machine can be categorized as follows:

1. Data Definition Facilities permit the structural definition of the data to be processed separate from the manipulation of the data. In a CODASYL DBMS, processors should be present for the schema and sub-schema data definition language (DDL).
2. Data Manipulation Facilities permit the usual manipulation of the data, such as storing new record occurrences, modifying/deleting record occurrences, retrieving records based on sequential position, key values, and set membership, and establishing currency of records or sets.
3. Sequential File Management is a secondary facility to create sequential files as input to DDL processors, transaction processors, etc. or as output from query and report generation facilities.
4. Database Maintenance Utilities are other utilities that initialize a new database, dump contents of schema or data, analyze statistics of the database, query and report generation, unload/load facilities, etc.

2.0 AN ILLUSTRATION: INVOICE PROCESSING

To illustrate the concept of a database machine with a concrete example, an application program dealing with invoice processing in a small business is given. The initial version of this program was written by Rob Gerritsen using the SEED (and Micro-SEED) database management system [Gerritsen, 1978].

The INVOICE package deals with a data structure (shown in Figure 8) composed of customers, parts, and invoices. Customers submit one or more invoices. Each invoice has one or more line items on it. Each line item refers to a certain part in the inventory. The schema definition is given in Figure 9.

The particular function that will be illustrated is the printing of an invoice statement. A sample printout is given in Figure 10.

This example is useful since it exhibits many of the interfaces to the DBMS. A stylized version of the FORTRAN source code is given in Figure 11. The steps of the program are:

1. Open database for retrieval
2. Select specified invoice
3. Obtain customer data for invoice
4. Process each line item
5. Print total price
6. Close database

Although the program does not actually update the database (and hence may not be representative). it does have to update the value of the invoice number for the FINDC operation. Values. therefore. have to be communicated from the application program to the DBMS. as is done in updating programs.

The ways that this application program interface to the DBMS can be summarized as follows:

1. Invoke operations
2. Refer to record and set types
3. Set values for data items.
4. Obtain values of data items
5. Check error status (and other system variables)
6. Reset error status

3.0 DESIGN ALTERNATIVES

Given the INVOICE example using the SEED DBMS as the illustration. various alternatives for the important design aspects of the database machine as examined.

3.1 LEVEL OF DATA MANIPULATION LANGUAGE

The first design alternative is the level of the language used to manipulate the data. The usual CODASYL DML is highly procedural. nagivation oriented. and tightly coupled with the application program. To illustrate these points. consider the processing of the following problem: "Find all the parts on invoice X that have been

backordered." To analyze the processing necessary for this problem, assume the following facts: (1) 10% of parts have been backordered; and (2) 20 line-items per invoice on the average.

For the case of using usual CODASYL DML, the host program will first invoke a FIND CALC for invoice X, then 20 FIND NEXT followed by FIND OWNER for each line-item in the invoice, and a final FIND NEXT to get the end-of-set condition. This totals to 42 FIND operations. Further, the data for the invoice and the 20 line-items will need to be transferred into the UWA. Therefore, the CODASYL DML will need 42 FIND and 21 GET operations for the processing.

In contrast, a high-level DML that was similar to a query language would need considerably less operations. Specifically, the operations needed would be: FIND CALC for invoice X. FIND conditional on line-item with backorder status, a GET for the invoice data, and two GETs for the parts backordered. Therefore, a high-level DML will need 2 FIND and 3 GET operations.

In summary, the reduction in data transfer between a high-level DML and a CODASYL DML would be 1/21 on data to the DBM and 3/21 on data from the DBM -- a considerable savings! This example illustrates the fact that the CODASYL DML was designed with the assumption of tight coupling between the DBM and the host program -- an assumption that

makes the normal DML unsuitable as the primary DML for a DBM.

3.2 HOST PROGRAM INTERFACE

The second design alternative is how to embed the DML for the DBM in the host application program -- a point that current literature on database machines has ignored. This section will explore three ways of handling the DML: (1) DML subroutine library; (2) subroutine CALL format; and (3) READ/WRITE format.

The first case would be to construct a DML subroutine library on the host processor for the particular language translator (e.g., FORTRAN) used by the application program. When compared to a resident DBMS, the source code should not have to be changed. Depending on the particular DML operation, the subroutine will handle communications with the DBM so that it is transparent to the application program.

The advantages of this case are: (1) no change in the source code of the application programs; (2) any changes in DBM conventions are hidden from the application program. The disadvantage is that over thirty subroutines have to be written for each combination of host processor and language translator connected to the DBM, thus making the DBM difficult to connect to new computer systems.

The second case would be to have a single common subroutine that would handle the communications to the DBM. The only argument to the DBM subroutine would be a character string that is to be sent to the database machine. The INVOICE program would require some changes, as is shown in Figure 12. Note that the syntax of the string can be condensed greatly, as was done in DBLOOK [Gerritsen, 1978].

The DBM subroutine would not only communicate with the database machine, but also perform the following functions: (1) updating error status indicator after each operation; (2) transmitting value changes in data items; (3) aborting operations if error status is nonzero; and (4) updating data items after GET operation.

The advantages would be: (1) only minor changes to existing application programs; and (2) single subroutine interface to the DBM. The disadvantages would be: (1) a new subroutine needs to be written for each host processor and language translator; (2) a special operation is required when critical data items are changed; and (3) error status has to be updated after each operation.

The third case for interfacing an application program to a DBM is directly through the standard READ/WRITE statements. All communication would be through the READ/WRITE statements as if the DBM was a reader/punch device. No additional subroutines would be necessary.

Further, there would be no need for a user working area (UWA).

The INVOICE source code with the READ/WRITE statements is shown in Figure 13. Note that the program appears longer than in the prior case. However, the total length of the program is shorter because no UWA is included. The operating system needs to define a logical I/O device that is connected to the DBM. The value for each data item must be sent to/from the program, as is shown in statements 101 and 202. Note that the program can be streamlined considerably if FORTRAN allowed the construction:

```
WRITE (DBM) 'FINDC INVCE'
```

The advantages of the READ/WRITE statement case would be: (1) nothing is hidden in subroutine calls or user working areas; (2) the sub-schema capability is retained; and (3) the total program length is shorter. The disadvantages are: (1) program needs to know the exact formats of data items contained in the sub-schema; (2) more statements are required to perform the DML operation; (3) the READ/WRITE routines must be interrupt-driven and buffered; (4) the error status has to be explicitly reset to zero; and (5) explicit operation is required to obtain the error status and other DBMS system variables.

3.3 USER WORKING AREA

An important design alternative to the previous three cases for DML operations is the placement of the user working area (UWA). In the first two cases, the UWA was actually maintained in two places -- one in the application program and one in the database machine. Depending on the database operation, one or the other UWA would be changed; hence, any discrepancies between the two UWA's had to be rectified by the subroutine in the host processor. In contrast, the third case using simple a READ/WRITE statement has no explicit UWA in the host processor; however, values of data items are stored in the host program space. Any change in values of error status, currencies, etc. will have to be explicitly performed by the host program.

3.4 COMMUNICATIONS INTERFACE

Another design consideration is the actual communication interface between the host processor and the DBM. This interface has to be considered both from the logical "hand-shaking" protocol and from the electrical specifications. The alternatives for this interface are numerous. The simplest alternative would be an asynchronous RS-232 300-baud ASCII full duplex line with a simple stimulus/response hand-shaking and with no error detection or correction. As networking protocols are standardized and

accepted, the DBM will have to accommodate them.

Another aspect of the communications interface is the data conversion problem. How will the data items be stored internally to the DBM? If it is a binary representation, it will be dependent on the architecture of the DBM or host computer. If it is strictly character representation, then significant conversion overhead will be incurred for processing numerical quantities.

3.5 NON-DML OPERATIONS

Prior to executing the application program for a database, other operations must be performed in preparation. For instance, the schema and sub-schema needs to be processed, and the database initialized. The design alternatives related to how these functions are provided are important. Further, there seems to be a tradeoff in processing effeciency between providing DML operations and non-DML operations. Since the functions of the database administrator are separate from the functions of the application programmer, a separation of DML operations (which are associated with the application program) and non-DML operations (which are associated with the administration of the database) is warranted.

4.0 CONCLUSIONS

Through the invoice processing example, several design alternatives were discussed in terms of their subtle implications to function and performance of a database machine. In particular, noted were the important design alternatives of DML language level, User Working Area location, and non-DML operations. The compatibility of DML operations with DDL structuring (and other database maintenance functions) were concluded to be weak; hence, a separation of these functions is warranted.

In general, the implementation of a database machine using a normal CODASYL database management system (e.g., SEED) and current implementation techniques (e.g., pointers and chains, rather than associative memory) does not seem to offer sufficient performance benefits. By extending the CODASYL DML (such as was proposed by Germano & Thakur [1978] and others), performance could be significantly increased. In any case, the use of a database machine may be more than adequately justified based on other benefits, such as: (1) independence of data between the application program and the database; and (2) convenience of performing database administration functions.

5.0 REFERENCES

Banerjee, J., Hsiao, D.K., Baum, R.I. Concepts and capabilities of a database computer. ACM Transactions on Database Systems, 3(4), December 1978, 347-384.

Baum, R.I., and Hsiao, D.K. Database computers: A step towards data utilities. IEEE Transaction on Computers C-25, 12, December 1976, 1254-1259.

Benbasat, I., and Goldstein, R.C. Data base systems for small business: Miracle or Mirage? Database, 9(1), Summer 1977, 5-8.

Berra, P.B. Data Base Machines. ACM SIGIR Forum, Winter 1977.

Bray, O., and Thurber, K.J. What's happening with data base processors? Datamation, January 1979, 146-156.

Canaday, R.H. et.al. A back-end computer for data base management. Communications of the ACM, 17(10), October 1974, 575-582.

Champine, G.A. Four approaches to a data base computer. Datamation, December 1978, 101-106.

CODASYL. Data Base Task Group Report. ACM, October 1969 and April 1971.

Gerritsen, R. SEED Reference Manual. International Data Base Systems, July 1978.

Gerritsen, R., and Hackathorn, R.D. Micro-SEED and its application. Proceedings of Electro 79, April 1979.

Germano, F., and Thakur, M. SEEDFE: A FOR EACH data language for SEED. Working Paper 78-12-08. Department of Decision Science, The Wharton School, University of Pennsylvania, 1978.

Hsiao, D.K., and Madnick, S.E. Data base machine architecture in the context of information technology. Proceedings of the Third International Conference on Very Large Data Bases, October 1977.

Lowenthal, E.I. A survey: The application of data base management computers in distributed systems. Proceedings of the Third Conference on Very Large Data Bases, October 1977.

Mohan, C. An overview of recent data base research. Database, 10(2), Fall 1978, 3-24.

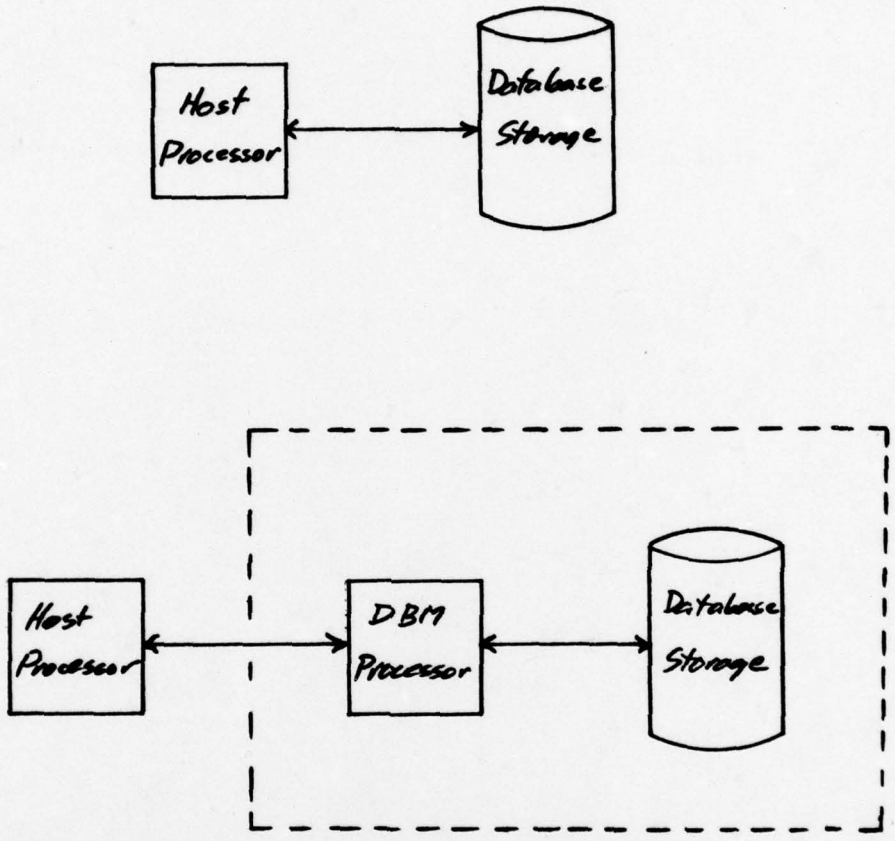
Ozkanakhan, E.A. et.al. RAP: An associative processor for relational data bases. Proceedings of the 1975 National Computer Conference, Anaheim CA. May 1975.

Rosenthal, R.S. The data management machine: A classification. Third Workshop on Computer Architecture for Non-Numeric Processing, May 1977, 35-39.

Stonebraker, M. A distributed database management system. Memorandum M78/23. UCB Electronics Research Laboratory, May 1978.

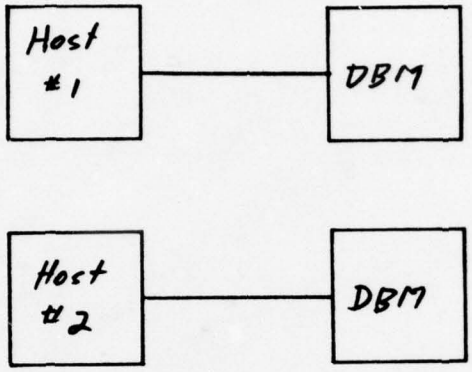
Yao, S.B. et.al. The Oregon Report: Data-base systems. IEEE Computer, 11(9), September 1978, 46-60.

Zornes, J.A. Data base management systems on mini-computers. Database, 9(1), Summer 1977, 9-13.

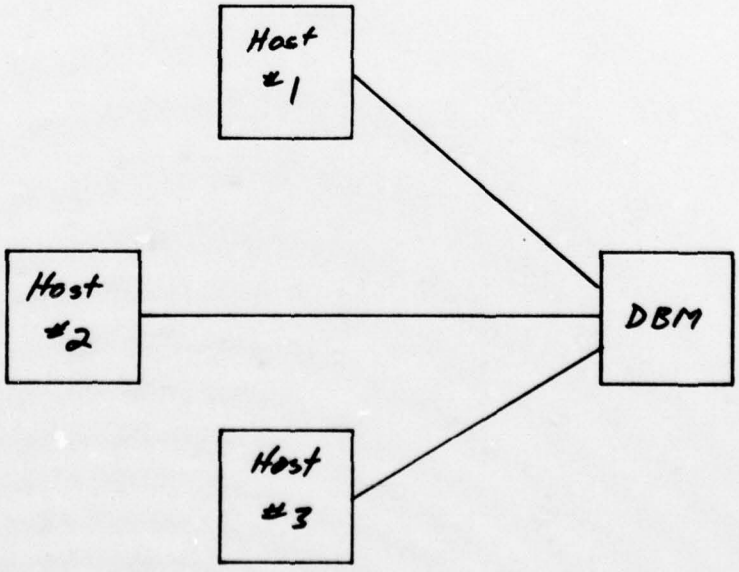


Host Resident DBMS vs. Database Machine

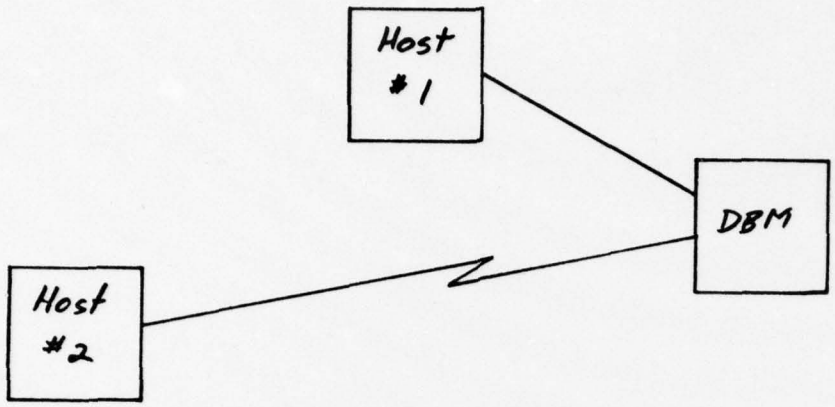
Figure 1



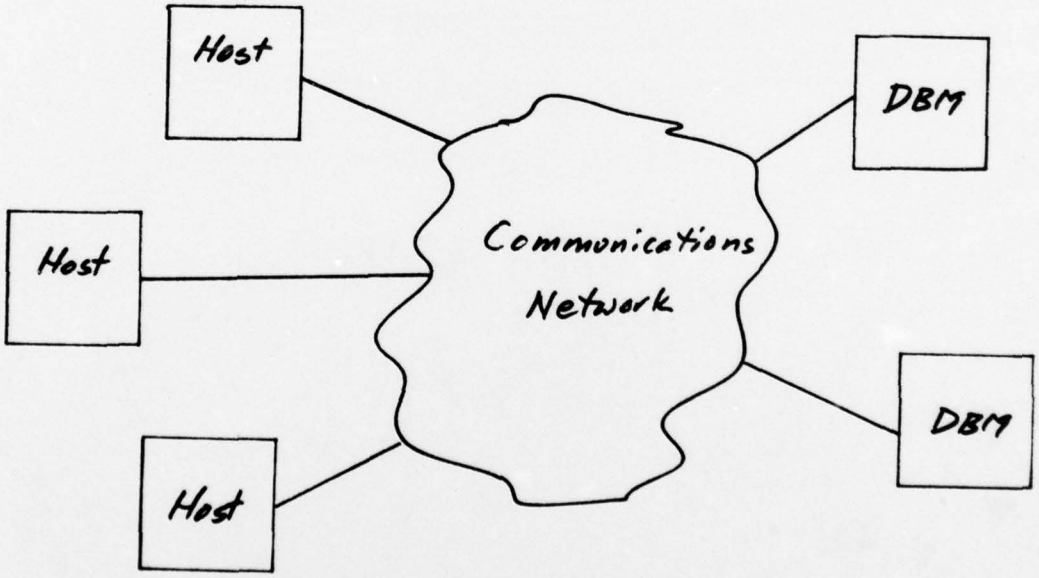
Data Transfer
Figure 2



Data Sharing
Figure 3



Remote Data Sharing
Figure 4



Network DBM
Figure 5

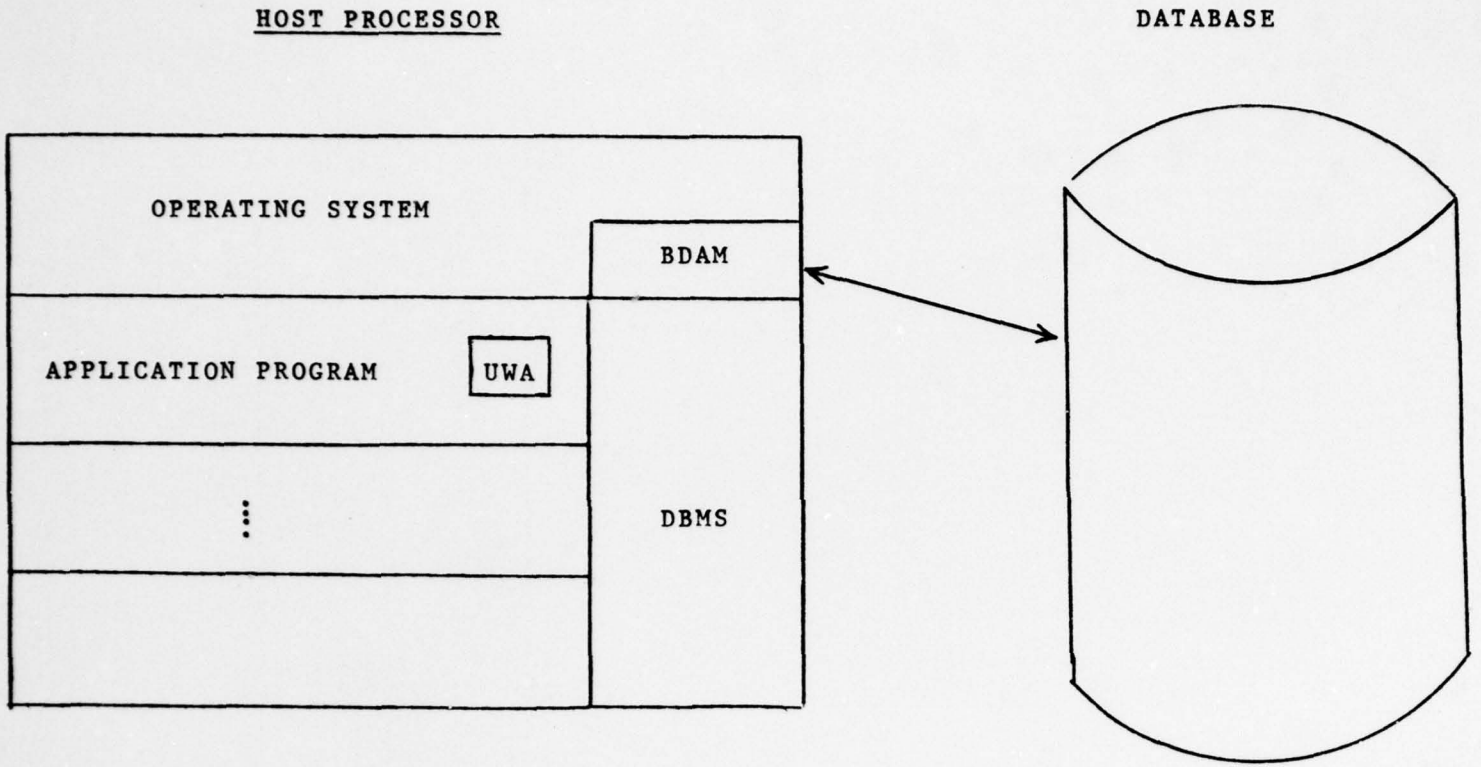

HOST PROCESSOR
DATABASE
OPERATING SYSTEM
BDAM
APPLICATION PROGRAM
UWA
DBMS

FIGURE 6

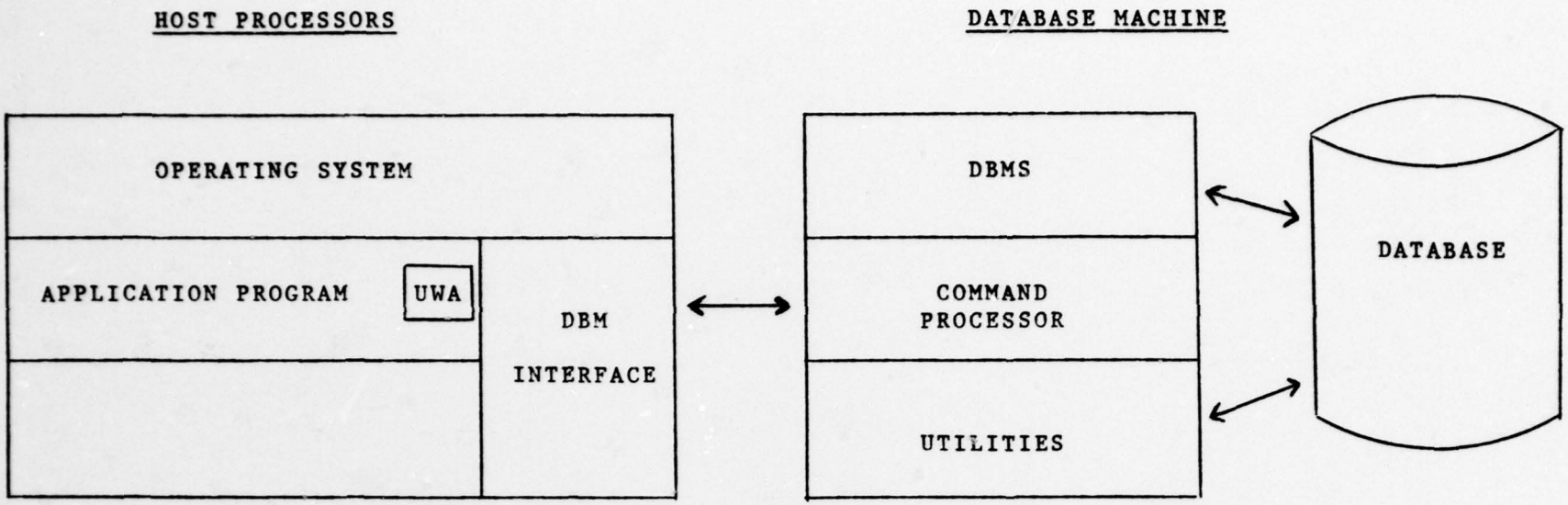
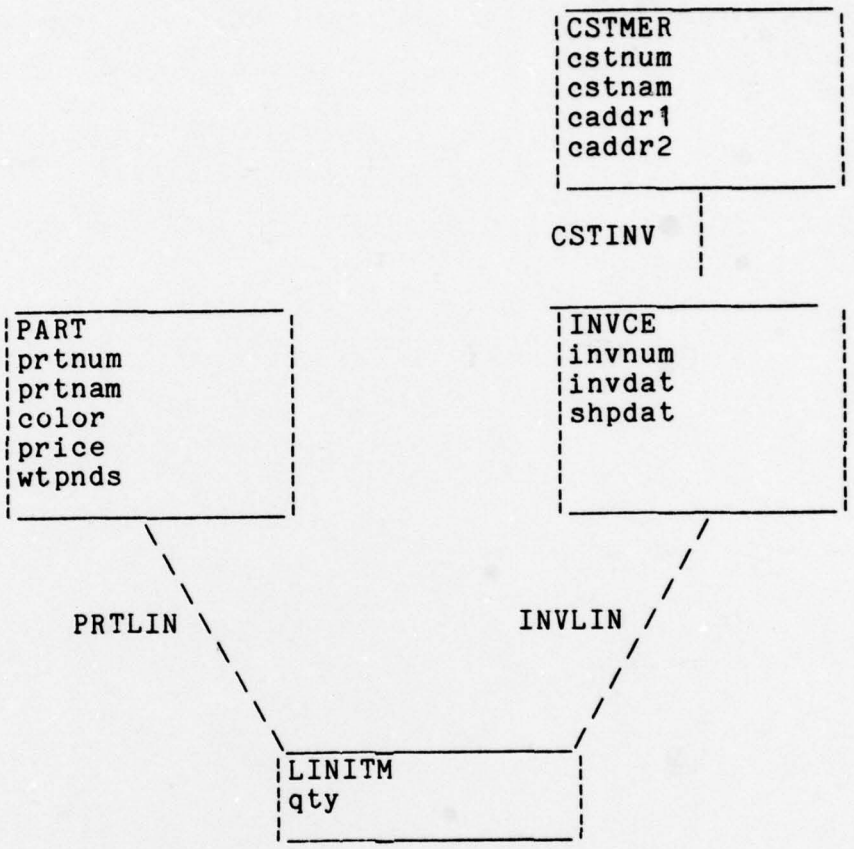

HOST PROCESSORS
DATABASE MACHINE
OPERATING SYSTEM
APPLICATION PROGRAM
UWA
DBM
INTERFACE
DBMS
COMMAND
PROCESSOR
UTILITIES
DATABASE

FIGURE 7



INVOICE Data Structure

Figure 8

INVOICE SCHEMA DEFINITION

Figure 9

```
SCHEMA NAME IS INVOICES
    DATABASE SIZE IS 5 PAGES
    PAGE SIZE IS 256.
RECORD NAME IS CSTMER
       LOCATION MODE IS CALC USING CSTNUM DUPLICATES NOT.
    CSTNUM TYPE IS CHARACTER 10.
    CSTNAM TYPE IS CHARACTER 20.
    CSTAD1 TYPE IS CHARACTER 20.
    CSTAD2 TYPE IS CHARACTER 20.
RECORD NAME IS INVCE
       LOCATION MODE IS CALC USING INVNUM DUPLICATES NOT.
    INVNUM TYPE IS CHARACTER 10.
    INVDAT TYPE IS CHARACTER 8.
    SHPDAT TYPE IS CHARACTER 8.
    SLSMAN TYPE IS CHARACTER 20.
RECORD NAME IS LINITM
       LOCATION MODE IS VIA INVLIN SET.
    QTY TYPE IS FIXED.
RECORD NAME IS PART
       LOCATION MODE IS CALC USING PRTNUM DUPLICATES NOT.
    PRTNUM TYPE IS CHARACTER 10.
    PRTNAM TYPE IS CHARACTER 20
    PRICE TYPE IS REAL.
    WTPNDS TYPE IS REAL.
    COLOR TYPE IS CHARACTER 1.
SET NAME IS CSTINV
    MODE IS CHAIN LINKED TO PRIOR
       ORDER IS FIRST
       OWNER IS CSTMER
       MEMBER IS INVCE LINKED TO OWNER
         SET SELECTION IS LOCATION MODE OF OWNER.
SET NAME IS INVLIN
       MODE IS CHAIN
       ORDER IS LAST
       OWNER IS INVCE
       MEMBER IS LINITM
         SET SELECTION IS LOCATION MODE OF OWNER.
SET NAME IS PRTLIN
       MODE IS CHAIN LINKED TO PRIOR
       ORDER IS LAST
       OWNER IS PART
       MEMBER IS LINITM LINKED TO OWNER
         SET SELECTION IS LOCATION MODE OF OWNER.
```

```
B>invoice


ENTER INVOICE NUMBER: 2005


CUSTOMER :2000                                    INVOICE  : 2005
Wharton Novelty                                   ORD DATE : 78-06-30
3600 Spruce                                       SHIP DATE: 78-07-15
Philadelphia, PA                                  SALESMAN : Jerry Lewi


PRTNUM    PRTNAM          C      PRICE    QTY     XPRICE
------    ------          -      -----    ---     ------
A1        Widget          B      12.00     40     480.00
B3        Wire Harness    M     165.00     32    5279.99
A2        Thingamajig     G      34.00     16     544.00
                                                ----------
                                                6303.99


ENTER INVOICE NUMBER:

B>
```

Sample Printout of INVOICE Program

Figure 10

INVOICE RETRIEVAL PROGRAM -- PRINT INVOICE STATEMENT
HOST RESIDENT DBMS FORMAT

FIGURE 11

```
      [Insert User Work File Here]
            :
      IMPLICIT INTEGER (A-Z)
      REAL XPRICE,TPRICE
      DATA CRT /.../
C
C  OPEN DATABASE FOR RETRIEVAL
      CALL DBOPEN('INVALL  ',0,0)
C
C  ASK FOR INVOICE NUMBER AND FIND INVOICE
100   [Write "Enter Invoice Number"]
      READ(CRT,---) INVNUM              [Read invoice number]
      IF (INVNUM.EQ.0) GO TO 500        [Do more invoices?]
      CALL FINDC(INVCE,FIRST)           [Find invoice]
      IF (ERRSTA.EQ.0) GO TO 200        [Does invoice exist?]
      ERRSTA = 0                        [Reset error status]
      [Write "invoice does not exist"]
      GO TO 100
C
C  GET CUSTOMER DATA AND PRINT INVOICE HEADER
200   CALL FINDO(CSTINV)                [Find owning customer]
      CALL GET(CSTMER)                  [Get customer data]
      CALL GET(INVCE)                   [Get invoice data]
      WRITE(CRT,---)CSTNUM,INVNUM, ...
      TPRICE=0.0
C
C  GET PART DATA AND PRINT EACH LINE ITEM
300   CALL FINDPO(NEXT,INVLIN)          [Find next line item]
      IF(ERRSTA.NE.0) GO TO 400         [Any more line items?]
      CALL GET(LINITM)                  [Get line item data]
      CALL FINDO(PRTLIN)                [Find owning part]
      CALL GET(PART)                    [Get part data]
      XPRICE=PRICE*QTY
      TPRICE=TPRICE+XPRICE
      WRITE(CRT,---) PRTNUM,PRTNAM, ...
      GO TO 300
C
C  PRINT TOTAL PRICE
400   ERRSTA = 0                        [Reset error status]
      WRITE(CRT,---) TPRICE
      GO TO 100
C
C  CLOSE DATABASE
500   CALL DBCLOS
      STOP
      END
```

INVOICE RETRIEVAL PROGRAM -- PRINT INVOICE STATEMENT
DBM SUBROUTINE CALL FORMAT

FIGURE 12

```
      [Insert User Work File Here]
            :
      IMPLICIT INTEGER (A-Z)
      REAL XPRICE,TPRICE
      DATA CRT /.../
C
C  OPEN DATABASE FOR RETRIEVAL
      CALL DBM ('DBOPEN INVALL,0,0')
C
C  ASK FOR INVOICE NUMBER AND FIND INVOICE
100   [Write "Enter Invoice Number"]
      READ(CRT,---) INVNUM              [Read invoice number]
      IF (INVNUM.EQ.0) GO TO 500        [Do more invoices?]
      CALL DBM ('PUT INVNUM')           [set INVNUM value]
      CALL DBM ('FINDC INVCE')          [Find invoice]
      IF (ERRSTA.EQ.0) GO TO 200        [Does invoice exist?]
      ERRSTA = 0                        [Reset error status]
      [Write "invoice does not exist"]
      GO TO 100
C
C  GET CUSTOMER DATA AND PRINT INVOICE HEADER
200   CALL DBM ('FINDO CSTINV')         [Find owning customer]
      CALL DBM ('GET CSTMER')           [Get customer data]
      CALL DBM ('GET INVCE')            [Get invoice data]
      WRITE(CRT,---)CSTNUM,INVNUM, ...
      TPRICE=0.0
C
C  GET PART DATA AND PRINT EACH LINE ITEM
300   CALL DBM ('FINDPO NEXT INVLIN') [Find next line item]
      IF(ERRSTA.NE.0) GO TO 400         [Any more line items?]
      CALL DBM ('GET LINITM')           [Get line item data]
      CALL DBM ('FINDO PRTLIN')         [Find owning part]
      CALL DBM ('GET PART')             [Get part data]
      XPRICE=PRICE*QTY
      TPRICE=TPRICE+XPRICE
      WRITE(CRT,---) PRTNUM,PRTNAM, ...
      GO TO 300
C
C  PRINT TOTAL PRICE
400   ERRSTA = 0                        [Reset error status]
      WRITE(CRT,---) TPRICE
      GO TO 100
C
C  CLOSE DATABASE
500   CALL DBM ('DBCLOS')
      STOP
      END
```

INVOICE RETRIEVAL PROGRAM -- PRINT INVOICE STATEMENT
DBM READ/WRITE FORMAT

FIGURE 13

```
      [No User Work Area Needed!]
      IMPLICIT INTEGER (A-Z)
      REAL XPRICE,TPRICE
      DATA CRT,DBM /.../
C
C  OPEN DATABASE FOR RETRIEVAL
      WRITE (DBM,11)
11    FORMAT ('DBOPEN INVALL,0,0')
C
C  ASK FOR INVOICE NUMBER AND FIND INVOICE
100   [Type "Enter Invoice Number"]
      READ(CRT,---) INVNUM              [Ask for invoice number]
      IF (INVNUM.EQ.0) GO TO 500        [Do more invoices?]
      WRITE (DBM,101) INVNUM            [set INVNUM value]
101   FORMAT ('INVNUM=',I6)
      WRITE (DBM,102)                   [Find invoice]
102   FORMAT ('FINDC INVCE'/'READ ERRSTA')
      READ (DBM,103) ERRSTA             [Get error status]
103   FORMAT (I6)
      IF (ERRSTA.EQ.0) GO TO 200        [Does invoice exist?]
      WRITE (DBM,104)                   [Reset error status]
104   FORMAT ('ZERO ERRSTA')
      [Type "invoice does not exist"]
      GO TO 100
C
C  GET CUSTOMER DATA AND PRINT INVOICE HEADER
200   WRITE (DBM,201)                   [Find owning customer]
201   FORMAT ('FINDO CSTINV')
      WRITE (DBM,202)                   [Get customer data]
202   FORMAT ('GET CSTMER')
      READ (DBM,---) CSTNUM,CSTNAM, ...
      WRITE (DBM,203)                   [Get invoice data]
203   FORMAT ('GET INVCE')
      READ (DBM,---) INVNUM,INVDAT, ...
      WRITE(CRT,---) CSTNUM,INVNUM, ...
      TPRICE=0.0
C
C  GET PART DATA AND PRINT EACH LINE ITEM
300   WRITE (DBM,301)                   [Find next line item]
301   FORMAT ('FINDPO NEXT INVLIN')
      READ (DBM,---) ERRSTA
      IF(ERRSTA.NE.0) GO TO 400         [Any more line items?]
      WRITE (DBM,302)                   [Get line item data]
302   FORMAT ('GET LINITM')
      READ (DBM,---) QTY
      WRITE (DBM,303)                   [Find owning part]
303   FORMAT ('FINDO PRTLIN')
      WRITE (DBM,304)                   [Get part data]
304   FORMAT ('GET PART')
      READ (DBM,---) PRTNUM,PRTNAM, ...
            :                           [And so on...]
```