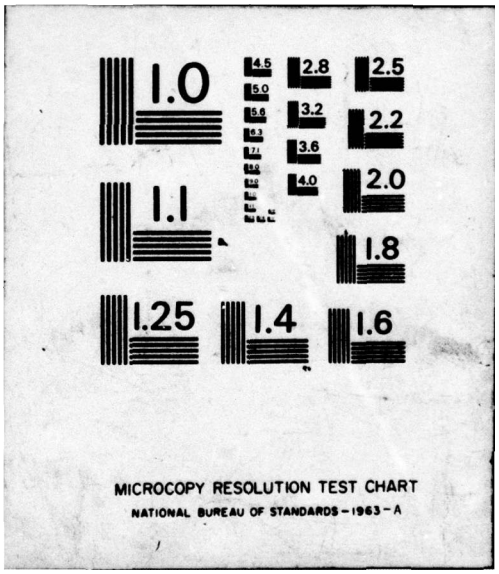
AD-A034 154 OHIO STATE UNIV COLUMBUS COMPUTER AND INFORMATION SC--ETC F/G 9/2
THE ARCHITECTURE OF A DATABASE COMPUTER. PART I. CONCEPTS AND C--ETC(U)
SEP 76 R I BAUM, D K HSIAO, K KANNAN N00014-75-C-0573
UNCLASSIFIED OSU-CISRC-TR-76-1 NL

END
DATE FILMED
2-77


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

TECHNICAL REPORT SERIES

COMPUTER & INFORMATION SCIENCE RESEARCH CENTER

THE OHIO STATE UNIVERSITY COLUMBUS, OHIO

(12)

(OSU-CISRC-TR-76-1)

THE ARCHITECTURE OF A DATABASE COMPUTER
PART I: CONCEPTS AND CAPABILITIES

by

Richard I. Baum, David K. Hsiao
and
Krishnamurthi Kannan

Work performed under
Contract N00014-75-C-0573
Office of Naval Research

Computer and Information Science Research Center
The Ohio State University
Columbus, Ohio 43210

September 1976

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: OSU-CISRC-76-1
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): The Architecture of a Database Computer, Part I. Concepts and Capabilities
5. TYPE OF REPORT & PERIOD COVERED: Technical Report,
6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s): Richard I. Baum, David K. Hsiao, Krishnamurthi Kannan
8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0573

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Office of Naval Research, Information Systems Program, Washington, D. C. 20360
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: [illegible]-A1

11. CONTROLLING OFFICE NAME AND ADDRESS
12. REPORT DATE: Sept [illegible] 76
13. NUMBER OF PAGES: 47

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
15. SECURITY CLASS. (of this report)
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Scientific Officer	DDC New York Area
ONR BRO	ONR 437
ACO	ONR, Boston
NRL 2627	ONR, Chicago
ONR 102IP	ONR, Pasadena

DISTRIBUTION STATEMENT A
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)

Database computer; security; clustering, partitioned content addressable memory; security atom; name mapping; structure memory; microprocessor; functional specialization

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

A hardware architecture for a database computer (DBC) is given in this paper. The proposed design overcomes many of the traditional problems of database system software and is one of the first to describe a complete data-secure computer capable of handling large databases.

This paper begins by characterizing the major problems facing today's database system designers. These problems are intrinsically related to the nature of conventional hardware and can only be solved by introducing new

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

architectural concepts. Several such concepts are brought to bear in the later sections of the paper. These architectural principles have a major impact upon the design of the system and so they are discussed in some detail. A key aspect of these principles is that they can be implemented with near-term technology. The rest of the paper is devoted to the functional characteristics and the theory of operation of the DBC. The theory of operation is based on a series of abstract models of the components and data structures employed by the DBC. These models are used to illustrate how the DBC performs access operations, manages data structures and security specifications, and enforces security requirements. Short ALGOL-like algorithms are used to show how these operations are carried out. This part of the paper concludes with a high-level description of the DBC hardware. The actual details of the DBC hardware are quite involved and so their presentation is the subject of Part II and Part III of this paper.

PREFACE

This work was supported by Contract N00014-75-C-0573 from the Office of Naval Research to the Computer and Information Science Research Center of The Ohio State University. The Computer and Information Science Research Center of The Ohio State University is an interdisciplinary research organization which consists of the staff, graduate students, and faculty of many University departments and laboratories. This report is based on research accomplished in cooperation with the Department of Computer and Information Science.

The research was administered and monitored by The Ohio State University Research Foundation.

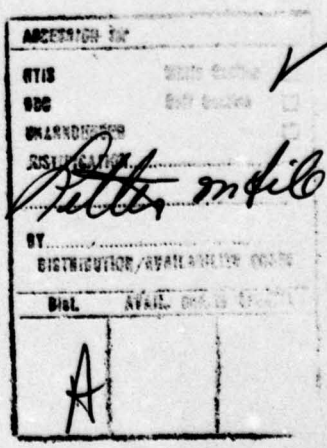


TABLE OF CONTENTS

THE ARCHITECTURE OF A DATABASE COMPUTER
PART I: CONCEPTS AND CAPABILITIES*

Richard I. Baum, David K. Hsiao and Krishnamurthi Kannan**

INTRODUCTION

A hardware architecture for a database computer (DBC) is given in this paper. The proposed design overcomes many of the traditional problems of database system software and is one of the first to describe a complete data-secure computer capable of handling large databases.

This paper begins by characterizing the major problems facing today's database system designers. These problems are intrinsically related to the nature of conventional hardware and can only be solved by introducing new architectural concepts. Several such concepts are brought to bear in the later sections of the paper. These architectural principles have a major impact upon the design of the system and so they are discussed in some detail. A key aspect of these principles is that they can be implemented with near-term technology. The rest of the paper is devoted to the functional characteristics and the theory of operation of the DBC. The theory of operation is based on a series of abstract models of the components and data structures employed by the DBC. These models are used to illustrate how the DBC performs access operations, manages data structures and security specifications, and enforces security requirements. Short ALGOL-like algorithms are used to show how these operations are carried out. This part of the paper concludes with a high-level description of the DBC hardware. The actual details of the DBC hardware are quite involved and so their presentation is the subject of Part II and Part III of this paper.

---

*This research is supported by the Office of Naval Research, Contract N00014-75-C-0573 and conducted at The Ohio State University.

**The last two authors are with the Department of Computer and Information Science, The Ohio State University, Columbus, Ohio; the first author is with IBM, Poughkeepsie, N.Y.

1. PROBLEMS AND SOLUTIONS

A number of major problems have been faced by database designers for a long time. These problems are of a very general nature and have frequently plagued builders of both hardware and software database systems. This section of the paper contains a discussion of these problems and of the architectural principles adopted in the DBC design which solve them.

1.1 The Problems of Database System Software

A. Name Mapping Complexity

The complexity of database system software is due, in large part, to the requirement for name mapping operations. Name mapping operations convert symbolic data names, called a query, into memory addresses which identify where the data named by the query can be found. Since the language which is used to name data is usually far more powerful than the addressing scheme implemented by the hardware, it is normally necessary to have rather involved name mapping algorithms. Name mapping algorithms must be highly optimized if they are to perform well. In particular, these algorithms must minimize their secondary storage access requirements. To accomplish this most name mapping algorithms use very complex auxiliary data structures to guide their operation.

To illustrate these problems consider the difficulties of the following typical name mapping scenerio. First, the query is used to access a "directory". The directory contains information which allows the algorithm to determine the approximate location of the requested data (this information thus potentially reduces the number of secondary storage accesses required by the algorithm). The information retrieved from the directory is then processed in some manner to yield secondary storage addresses. Finally, the secondary storage is accessed and the data is located. This software name mapping algorithm requires auxiliary data structures in both the directory and the secondary storage. These auxiliary data structures, which include elements such as pointers, allow rapid retrieval of data from the secondary storage and the directory. All of these auxiliary data structures must be properly maintained. This last requirement is the underlying cause for the great difficulty most contemporary systems have in executing update operations. Update

operations make changes to auxiliary structures and so they are frequently very time consuming. A classic example is the process of modifying a network of pointers.

B. Performance Bottlenecks

Database system software normally consists of several distinct functional parts which perform specific tasks. For example, separate software modules which perform query parsing, directory access, directory processing, data retrieval and update, and data security are usually found in contemporary database management systems. To have a well-balanced system with high throughput it is necessary for these modules to have diverse performance capabilities. Such diverse capabilities are difficult to achieve when these software modules are usually implemented with the same underlying hardware. When such performance capabilities cannot be met because of inherent hardware constraints, the system develops bottlenecks and its performance is consequently degraded. Contemporary database management systems are usually plagued by many such bottlenecks.

C. Data Security Overhead

Powerful data security facilities are generally a performance hinderance on contemporary systems. The most powerful data security mechanisms allow security specifications to be written in the query language of the system. To authenticate access operations it is therefore necessary to perform multiple name-mapping operations--one for determining the requested data and several for determining the data being effected by the security specifications. The use of name-mapping algorithms to carry out security enforcement is generally too much of a performance burden to be seriously considered in present systems.

D. Add-on Approach to Security

Security capabilities are frequently just an "add-on" to present systems. This kind of design philosophy opens the way to not only performance difficulties but also to questionable reliability. With the high degree of complexity of current systems it is extremely difficult to add on a security mechanism which will guarantee that all "backdoors" are, in fact, closed.

1.2 The Problems of Building DBC Hardware

A. The Need for Distant Technology

Attempts to build database computer hardware have been made before [1,2, 3,4,5]. These efforts have been plagued by a number of critical drawbacks. The most serious shortcoming in these systems has been their reliance on monolithic fully associative memories. Such memories are not feasible for supporting a large on-line database (i.e., at least 10^9 bytes).

B. Incomplete Hardware Designs

Many DBC attempts [1,6,7,8,9,10] have led to machines that could not perform all of the functions necessary to support a viable database management system. In particular, some of them can support just one database management function in hardware such as directory processing or data retrieval; others cannot support a critical function, such as update, well. Previous DBC approaches have almost always lacked a data security capability -- such an omission makes the use of the computer in a data sharing environment very questionable indeed. A viable DBC must support all database management functions equally well.

1.3 Problem Solving Concepts

To overcome the problems described above a number of key design concepts were used in the DBC. These design concepts include both architectural principles and design philosophy.

A. Partitioned Content Addressable Memories

The use of hardware content addressing can significantly reduce the need for name-mapping data structures. Content addressable memories eliminate the need for knowing the actual location of a data item. In such a memory the notion of "actual location" is nonexistent; instead, all data is accessed by specifying its attributes. This kind of access gives us a very important capability: data items may be moved about without any need to modify name-mapping data structures. This is because few, if any, name-mapping data structures are needed in a content addressable memory. This characteristic greatly facilitates update operations.

A fully associative memory large enough to hold a complete database is not feasible. However, a storage system consisting of many blocks (called, partitions) of memory each of which is content addressable is quite feasible. We call this memory concept a partitioned content addressable memory (PCAM). It is possible to build PCAMs of widely varying performance characteristics. In particular, it is possible to design the access speed and capacity of a PCAM to meet a particular performance requirement. This flexibility allows us to design three PCAMs for use in the proposed DBC architecture with very different speeds and capacities. As will be seen later, a PCAM of gigabyte capacity is feasible with current technology.

B. Structure and Mass Memories

Since PCAMs are block-oriented, it is necessary to have some name-mapping data structures in the system. Our goal, of course, is to minimize their use as much as possible. This leads to the architectural concept of the structure memory. A DBC employing this concept has two memories. The mass memory contains the information making up the database and is by far the larger of the two memories. The mass memory contains only update invariant name-mapping data structures. Once an update invariant data structure is created for a data item it need never be modified so long as that data item continues to exist anywhere in the database. The data structures in conventional mass storage are not update invariant; they must be modified whenever the location of a data item changes. The structure memory contains all of the non-update invariant name mapping information necessary to locate data in the mass memory. To access the database the system first accesses the structure memory, obtains mapping information, processes it and then accesses the mass memory.

The proposed DBC employs the structure memory concept. Both the mass memory and the structure memory are PCAMs. They, of course, have very different functional characteristics.

C. Area Pointers

To simplify the name-mapping data structures that are still required by the DBC, a concept called the area pointer is used. An area pointer indicates the PCAM partition in which a data item may be found by employing content addressing. Unlike the location pointers used in contemporary systems, area pointers need not be modified when data items are moved around within a partition.

Conventional mass memories do not support the area pointer concept. Our mass memory, on the other hand, is a PCAM and so area pointer support comes naturally. Area pointers are stored in and managed by the structure memory.

D. Functional Specialization

The DBC contains a number of components with considerably different processing speed and memory capacity requirements. The mass storage and structure memory are examples of two such components. To keep any component from becoming a bottleneck we employ the architectural concept of functional specialization[1]. In a functionally specialized system, the components are individually designed to be optimally adapted to their function. The processing power and memory capacity of each component is determined by its role in the system. Because all major components are specialized (i.e., functionally separate from other components), estimation of their required processing power and memory capacity is much easier. In the proposed DBC each of the major components is a physically separate hardware component. This approach allows us to build a relatively well-balanced system and to avoid bottlenecks by providing each component with the right amount of processing power and memory capacity.

The proposed DBC has seven major functionally specialized components; the keyword transformation unit (KXU), the structure memory (SM), the mass memory (MM), the structure memory information processor (SMIP), the index translation unit (IXU), the database command and control processor (DBCCP), and the security filter processor (SFP). These seven components are the heart of a database computer that is able to support gigabyte database capacities while providing full retrieval, update and security capabilities.

E. Look-Aside Buffering

When an update operation occurs it is sometimes necessary to modify name mapping data structures. To insure the correct execution of the queries which follow the update, the execution of queries is normally postponed until the update operation and all of its related changes to data structures are complete. This is because the data involved in an update

[1] This term was suggested to us by E. Feustel.

operation could very well be the data the next operation depends on. Thus update operations can become bottlenecks in contemporary systems.

In our DBC changes to name mapping data structures induced by an update operation will be much fewer in number and much easier than in contemporary systems; the changes, nevertheless, will require some time. To reduce the wait-time, a look-aside buffer is used to store update commands temporarily. This buffer allows the results of updates to be immediately available to the rest of the system before they are permanently recorded. These changes can be stored in the look-aside buffer in much less time than it would take to permanently record them in the system. In this way, queries following an update operation do not have to wait for the permanent effects of that update operation to be actually stored before they are executed.

F. An Integral Data Security Mechanism

At the outset the security mechanism was made an integral part of the DBC design. This design philosophy not only allows us to construct a system that has no "backdoors" but also insured that all access requests are, in fact, controlled by the DBC's security mechanism. We achieved this by designing the security mechanism first and by then designing the rest of the system around it. The DBC supports a security specification language that is the same as the DBC's query language.

Security in the DBC is provided in terms of two distinct protection mechanisms. The first mechanism based the security atom concept [14] requries some form of cooperation from the creator of the database. This mechanism achieves enforcement in a rapid and elegant manner and is incorporated in the DBCCP. The second enforcement mechanism allows the creator wide latitude in the manner in which he can specify security related information. Since it generally requires more (and different) processing than the first, the second mechanism is incorporated in a functionally specialized component, the security filter processor (SFP). Such an architecture tends to lead to good performance while ensuring that security is not compromised.

G. Performance Enhancement by Clustering Techniques

A powerful clustering technique has been incorporated in the DBC, which allows the creator of the database to optimize access times. The placement of every record into the DBC can be controlled (in terms of its properties)

by the creator of the database in such a way that retrieval of records with similar properties may be accomplished with minimal access delays.

H. Advanced Technology

A database computer for the near future should take maximum advantage of the technology that is likely to be available then. This design philosophy is especially important in an era of rapidly developing technology such as the present one. The significant developments expected in the area of high speed bulk storage (semiconductors: CCDs and dense RAMs, magnetic bubbles and electron beam memories) and low cost processing power (microprocessors) dictate a major rethinking of conventional machine architectures.

For example, an all-electronic storage component may replace the fixed head disk as the fastest bulk storage device in the system. Since these all-electronic fixed head disk replacements will offer at least an order of magnitude improvement in access time, they will allow powerful data organizations that were previously not feasible to become practical as well as allowing a significant increase in the throughput of certain database system components. Low-cost random access memory will allow the widespread use of very large data buffers and independent functionally specialized memories throughout the system. Low-cost microprocessors coupled with low-cost bulk memory will allow parallel processing techniques to be used to construct memories with powerful search capabilities.

2. THE FUNCTIONAL CHARACTERISTICS OF THE DATABASE COMPUTER

The database computer must communicate with external systems and so a DBC interface must be defined. The functional characteristics of the DBC provide such an interface. The DBC functional characteristics define the data management and security features supported by the DBC and show how commands are sent to and executed by it.

2.1 A Back-end Machine

The DBC is not a general-purpose computer and does not have a typical operating system. Instead, it is a separate machine dedicated to database operations. Other computers and systems communicate with the DBC by using DBC access commands and by sending or receiving database information. The decision to design the DBC as a back-end machine to support database operations in a general-purpose computer system is a result of applying the concept of functional specialization. A number of advantages accrue from this decision [11]. First, the DBC is not constrained to be used with a particular kind of general-purpose computer system. Second, more than one system can share a DBC. In this way, the back-end DBC can serve many front-end computer systems. Third, several DBCs can become part of a general-purpose computer system to facilitate distributed database applications. This interconnection could be done with a geographically widespread communications network. Finally, all DBC access channels can be identified and controlled. This is necessary to insure that no "backdoors" into or out of the DBC exist.

We shall collectively call all of the systems which communicate with the DBC the program execution system (PES). We aggregate all these systems into one conceptual entity so that it will be easier to describe the operation of the DBC.

2.2 The Functional Model

The DBC proposed here implements the attribute-based model. This model has been extensively studied and is particularly well-suited to supporting contemporary database functions [12,13,14].

A. Queries -- The Symbolic Data Names Used by the DBC

Our definition of a database starts with two terms: a set AT of "attributes" and a set VA of "values". These are left undefined to allow the broadest possible interpretation. We shall denote a member of AT by at and a member of VA by v.

A record R is a subset of the cartesian product AT x VA. To simplify the notation we will assume without loss of generality that in a record all attributes are distinct. Thus, R is a set of ordered pairs of the form:

(an attribute, a value).

Records are physically stored in the mass memory. The set of all records in the mass memory is called a database (DB). The database may be partitioned into subsets called files. To distinguish among several files, each file is given a unique name F, called its file name.

The keywords of a record are those attribute-value pairs which characterize the record. In practice it is useful to consider only succinct keywords. We shall denote a keyword by the notation K.

A keyword predicate T(K) is true for a keyword K if K satisfies the condition specified by T. The most commonly used keyword predicate is the equality predicate E(K) which is true for K when K is the same as a certain keyword, say, K'. For this special case, we shall denote the keyword predicate by simply K'. Another common keyword predicate is the less-than predicate $LT_{at}(K)$. This predicate is true for K when the attribute of K is at and the value of K is less than some value, say, v . This keyword predicate shall be denoted by (at < v). This predicate can be easily generalized to handle other relational operators. All queries are made up of Boolean expressions of keyword predicates. Keyword predicates allow queries to specify just about any conceivable keyword property.

A keyword predicate is true for a record R if some keyword K in R satisfies the keyword predicate. A query is a proposition given by a Boolean expression of keyword predicates. A query is true for R if this proposition holds for the keywords in R; such a record is said to satisfy the query. The set of all records in DB (or in a file of DB) that satisfy a query Q will be called its response set and denoted by Q(DB)

(or Q(F)). Every query is written in disjunctive normal form.

$$Q^1 \vee Q^2 \vee \ldots \vee Q^k$$

where each conjunct Q^i of the query has the form:

$$T_1^i \wedge T_2^i \wedge \ldots \wedge T_n^i$$

where T_j^i are keyword predicates. Some examples of queries follow. The query $K_1 \wedge K_2$ is true for R when K_1 and K_2 are both in R. The query $K_1 \wedge$ (Salary < 10,000) is true for R when K_1 is in R and there is a keyword in R whose attribute is Salary and whose value is less than 10,000. More elaborate queries can be formed if they are in disjunctive normal form.

B. Security Specifications -- The Protection of Data

A database access or simply an access is the name of a DBC operation which transfers information to or extracts information from DB. Examples of accesses are retrieve, insert and delete. Let ACC denote the set of the names of all the accesses available in DBC. Let a member of ACC be represented by a and a subset of ACC by A.

A security specification is a relation

$$S: DB \rightarrow 2^{ACC} \quad \text{where } 2^{ACC} \text{ is the power set of ACC.}$$

Thus, for a record R in DB, the security specification, S(R) = A, indicates which subset A of accesses is permitted on R.

A file sanction or simply a sanction is defined as the couple (Q,A) where Q is a query, and A is a subset of ACC. A sanction (Q,A) induces a relation $S.FS_Q$ over records R of the database such that

$$S.FS_{Q,A}(R) = \begin{cases} A \text{ if R satisfies Q.} \\ ACC, \text{ otherwise.} \end{cases}$$

Thus, a sanction induces a security specification which indicates that only the accesses in A may be performed on the records satisfying Q. When R does not satisfy Q, all accesses may be performed on it. In this case we say that no sanctions of (Q,A) are applicable to R. The sanction is a very powerful type of security specification since it allows the full power of the query language (i.e., Q) to be used to specify records to be protected.

Consider a file named F and a set of sanctions where

$$S = \{(Q_1,A_1), (Q_2,A_2),\ldots,(Q_m,A_m)\}.$$

A <u>database capability</u> (F,S) induces a security specification $S.DC_{F,S}$ over the elements of R of F such that

$$S.DC_{F,S}(R) = \bigcap_{i=1}^{m} S.FS_{Q_i,A_i}(R)$$

In words, $S.DC_{F,S}(R)$ is the set of all accesses granted for R by one or more file sanctions in S and not denied by any sanction of S. Security specifications are therefore stored in the DBC as database capabilities. The database capabilities specify exactly what access operations are allowed on records. The DBC maintains database capabilities for each active user.

For example, consider the database capability $\{(Q_1,A_1), (Q_2,A_2)\}$. Suppose Q_1 and Q_2 specify overlapping sets of records as shown in Figure 1. Then the records in the intersection of Q_1 and Q_2 have the access privileges, $A_1 \cap A_2$ associated with them.

C. Command Execution -- The Processing of Access Requests

An access command has the form <U,(F,Q),a> or the form <U,(F,R),a>. U represents the name of the user issuing the command, a is an access, (F,Q) represents the response set Q(F) on which the access is to be performed, and (F,R) represents a record R of F that is to be used in the access. Before an access is executed, file F must be protected from unauthorized access by the user U. This is accomplished by first employing U to locate the appropriate database capability (F,S). Then for the command <U,(F,Q),a>, the access a is performed on each record R of Q(F) for which $S.DC_{F,S}(R)$ contains a. For the command <U,(F,R),a> the access a is performed on R if a is in $S.DC_{F,S}(R)$. If any data need be sent to the user as a result of the access command, it is sent to the PES to be routed to that user.

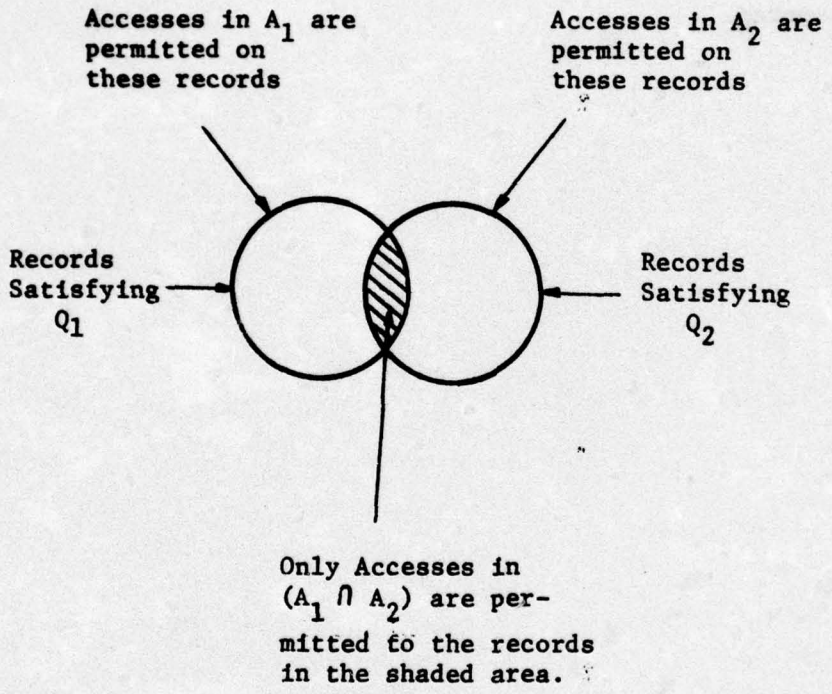


Figure 1. The Security Specification Induced by $\{(Q_1,A_1), (Q_2,A_2)\}$

2.3 The Need for Front-end Support

Before a user issues any access commands for a file, the database capability specifying the user's access rights to that file is sent to the DBC by the PES. An access command is rejected by the DBC unless the appropriate database capability is found. It is the responsibility of the PES to send the correct database capabilities to the DBC and to authorize the use of access operations by users by constructing appropriate database capabilities. In this way our DBC design does not impose any restriction on the nature of the PES's security mechanisms or on the authorization policies it supports.

3. THEORY OF OPERATION

A model which describes the basic components of the DBC and how they interact to realize the DBC's functional characteristics is now given. In the presentation we do not emphasize the intricacies of hardware design. Instead, we describe the operation of the components at a conceptual level. In Part II and Part III of the paper, we shall show how these components can actually be implemented with existing and emerging technology.

The theory of operation is presented in two sections. In the first section a data model is developed. In the next section we show how the data model described above is realized by the DBC with the aid of functionally specialized components.

3.1 The Data Model

The need for auxiliary data structures arises from the fact that the mass memory is not fully associative. Therefore, a technique to minimize mass memory accesses is required to insure high performance. We shall employ a PCAM-based mass memory to implement the structure memory concept. The mass memory's content addressability allows it to contain only update invariant mapping structures. The data model will allow us to determine the nature of the information to be kept in the structure memory.

When a PCAM partition is used to store records, record placement within the partition does not affect the system's performance. When a set of records is not placed in the same partition, the system's performance can be affected since multiple PCAM accesses may be required to retrieve the records. To address this problem a database is normally partitioned into groups of records whose records should all be physically close to each other. The exact nature of "closeness" is dependent on the properties of the memory. For example, on a disk with movable read/write heads, records could be considered close if they are stored in the same cylinder. This seems reasonable since the cost of initially accessing a cylinder of the disk is usually much greater than the cost of immediately following subsequent accesses to the same cylinder. The underlying reason for this is the requirement for mechanical motion to access a new cylinder.
In the data model we shall consider records to be close when they are stored in the same partition of the PCAM mass memory. To distinguish

partitions in the mass memory PCAM from those in other PCAMs, we shall call each of these partitions a minimal access unit (MAU).

There are many reasons for placing one record close to another record. A basic reason, related to performance, is the likelihood that these records will be accessed simultaneously. There are other reasons for grouping records. For example, compartmentalization of records for security reasons is one. Precisely what features of these records allow the designer to deduce a particular record grouping does not concern us at this time. Our goal as builders of generalized hardware to support a database system is not to choose a specific way to partition the database but instead to provide a general mechanism with which many possible partitionings may be realized. Such a mechanism will be presented shortly.

Let there be L MAUs in the mass memory and let L be called the minimal access unit count. All L MAUs are of fixed size. We denote the minimal access unit size by $|MAU|$. Associated with the database DB is the set of records denoted by M(DB) and defined as {R:R is in DB}.

If the set M(DB) is further partitioned into L subsets and each of these subsets represents the records which are placed in a MAU, then the union of the subsets is called a database configuration of M(DB). The size of a record, i.e., the number of bits needed to represent it in memory is denoted by $|R|$. A datebase configuration is valid if each subset X of M(DB) satisfies the constraint

$$\left(\sum_{R \in X} |R| \right) \leq |MAU|$$

In other words, a database configuration is valid if all of the records of M(DB) fit into MAUs of the mass memory. A valid database configuration results in a memory map which describes how the records are placed in the mass memory.

Each MAU is represented by a unique name called the minimal access unit address (MAU address), denoted by f where $0 \leq f < L$. Let M_f represent the contents of the f-th MAU.

The DB storage structure is defined as the ordered sequence

$$(M_0, M_1, \ldots, M_{L-1}).$$

This sequence represents the distribution of records in the MAUs.

Let F be a file whose records contain just m different keywords denoted by $K_1, K_2, \ldots, K_m$. To keep track of the MAUs in which records containing the keyword K_i are to be found, we form the set of $D(F,K_i)$ defined as

$$\{f \mid R \text{ is in } F \text{ and } K_i \text{ is in } R \text{ and } R \in M_f\}.$$

$D(F,K_i)$ is called a directory entry and each element f of $D(F,K_i)$ is called an index term. In words, $D(F,K_i)$ is the set of all names of MAU which contain one or more records with the keyword K_i.

The directory of file F is defined as the set DIR(F) defined as

$$\{D(F,K_1), D(F,K_2),\ldots,D(F,K_m)\}.$$

The directory of a file represents the structural information needed to access the mass storage. We shall see how it is used shortly.

As mentioned earlier, the DBC allows the creator of a file to enhance performance by allowing records of the file to be identified as a group (or a cluster) and by accessing such records with minimal access delay. Let us motivate the concept of clustering and the resulting performance improvement by a simple example. Let a file F (to be placed in the DBC) have n records of which we choose four records for our discussion. These four are shown in Figure 2a.

In figure 2b we have shown an arbitrary placement of records in the two MAUs that have been made available in the database for the file F. Now, if a query for retrieval is received in the form, "Retrieve records which satisfy the conjunct $(K_1 \wedge K_3)$", then the DBC has to make two MAU accesses. However, if the records are placed in the MAUs grouped according to the occurrence of keywords (K_1,K_2 and K_3) in a record, then the resulting configuration will be as shown in Figure 2c. Such a configuration will facilitate the retrieval of all records which satisfy the given query with a single access to the mass memory.

The above discussion implies two things: First, the creator of the file has an idea of the type of queries that will be made on the file. Second, the system (DBC) provides him with a mechanism of effectively conveying that knowledge to the DBC. While we, as system designers, cannot predict how much knowledge a creator may have of his file usage, we must ensure that he is provided with an easy yet powerful mechanism to utilize that knowledge to his best advantage. The mechanism that we have adopted and shall describe here is capable of being naturally integrated into the query language used by the users.

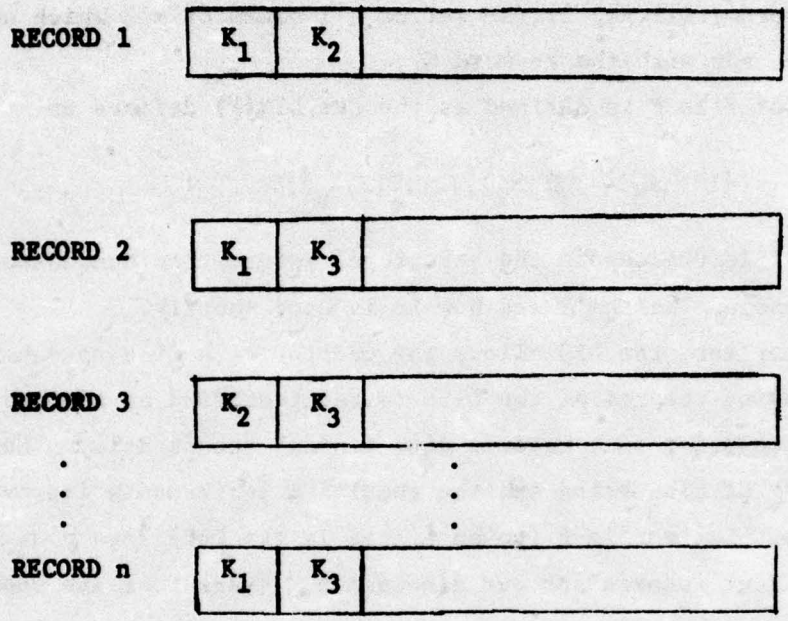


Figure 2a. Records Belonging to a File

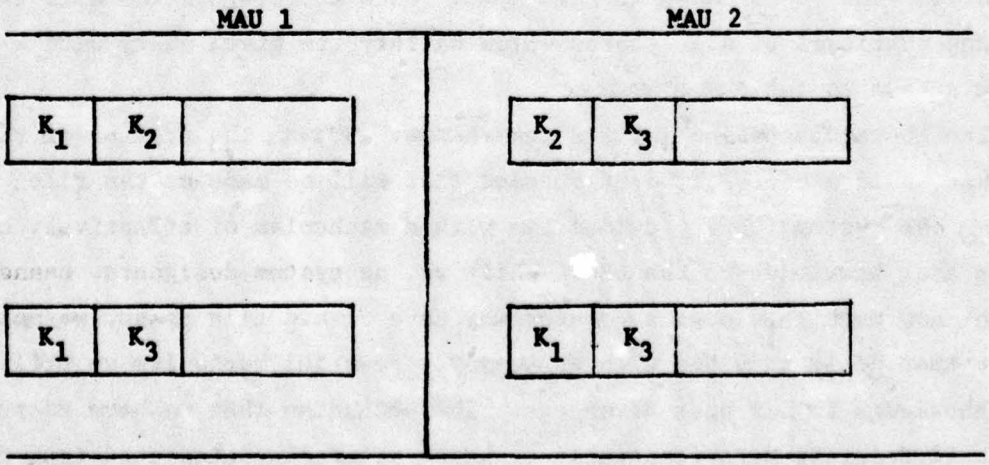


Figure 2b. An Arbitrary Assignment of Records to MAUs

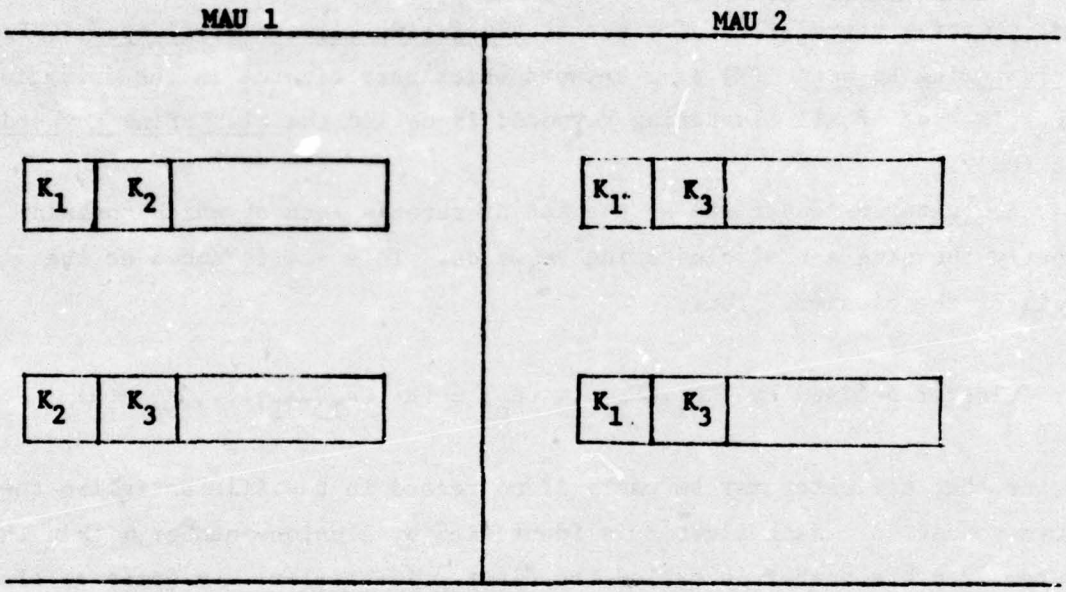


Figure 2c. An Assignment of Records to MAUs which Results in Minimum Number of Access for Certain Queries

A mandatory clustering condition (MCC) is a query which is used to interrogate the MAUs assigned to a file in order to determine which MAUs are eligible containers of a given record. [We use the word "interrogate" in a literal sense here; i.e., to ask questions about the MAUs.]

An optional clustering condition (OCC) is a query which is used to determine which MAU among a set of MAUs will ultimately contain a record.

A clustering condition (CC), either MCC or OCC, is formed by considering a disjunctive normal form of a set of clustering keyword predicates (CKPs). A clustering keyword (CK) is a keyword which participates in the formation of a CKP. The set of all clustering keywords is called the clustering keyword set (CKS).

A cluster c is defined as the set of records each of which contain exactly the same set of clustering keywords. This set is known as the basis of the cluster. Thus,

$$\text{Cluster defined by } (CK_1, CK_2, \ldots, CK_n) \equiv \{R \mid CK_1, CK_2, \ldots, CK_n \in R\}$$

Notice that a cluster may be empty if no record in the file satisfies the above condition. Each cluster is identified by a unique number within the system. Such a number is called the cluster identifier. An optional clustering condition is associated with a number called a cluster weight (CW). We shall elaborate on the use of cluster weights shortly.

We now describe how the above concepts can be used to place a record in the database.

A record for insertion is associated (by the creator) with a single mandatory clustering condition (MCC) and a set of optional clustering conditions $\{OCC_1, OCC_2, \ldots, OCC_q\}$. In addition the record contains a set of clustering keywords as part of the record definition. Obviously, in order to produce meaningful clusters, a record's clustering keyword set must satisfy its MCC.

In order to determine the MAU(s) in which the record could be placed, we need to know the identities of the clusters whose basis satisfy the MCC associated with the record and the MAUs in which these clusters reside. Obviously, if we have to access each MAU to determine the cluster identifiers our purpose of performance enhancement would be defeated. Thus, it is essential that clustering information is kept outside the MAUs themselves, perhaps, in the structure memory. Now having determined the set of MAUs where the record can be placed, we use the optional clustering conditions to choose one of the many MAUs determined earlier. This is done as follows. Let CW_i be the cluster weight of the optional clustering condition OCC_i. Then define, for MAU f,

$$OW(M_f) = \sum_{i=1}^{q} \begin{cases} CW_i, & \text{if } MAU_j \text{ contains a record which satisfies } OCC_i. \\ 0, & \text{otherwise.} \end{cases}$$

The record is then placed in the f-th MAU such that,

$$\forall k \, (OW(M_f) \geq OW(M_k))$$

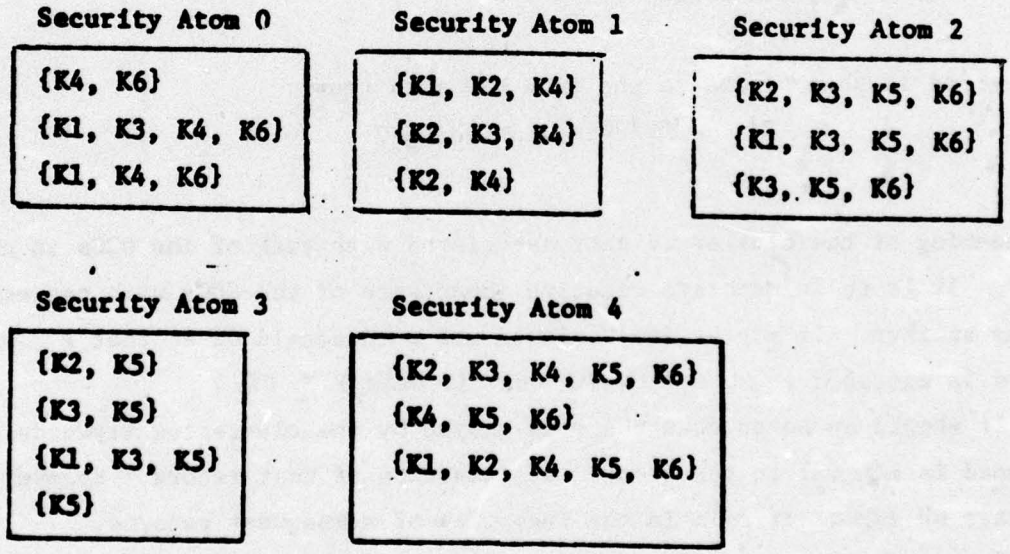
The meaning of the cluster weights associated with each of the OCCs is now clear. It is to incorporate relative importance of the OCCs with respect to one another. It may be desirable to set a threshold OT so that a record is assigned to the f-th MAU only if $OW(M_f) \geq OT$.

It should be noted that the role played by the clustering keywords of a record is minimal in the process of insertion of that record. However, it plays an important role in the insertion of subsequent records.

We now show how the DBC can group records for security purposes. Certain attributes of a file may be designated as security attributes by the creator of the file. A <u>security keyword</u> is a keyword whose attribute is a security attribute. Each record belonging to such a file with security attributes contains a set of security keywords (possibly empty). This set defines a <u>security atom</u>. A record is said to belong to a security atom if and only if its security keywords define the security atom in question. The concept of security atoms is due to [14]. In figures 3a, 3b and 3c, we have illustrated this concept by means of an example [19].

{K4, K6}	{K1, K3, K5, K6}	{K1, K4, K6}
{K2, K3, K5, K6}	{K2, K3, K4, K5, K6}	{K2, K4}
{K2, K3, K4}	{K1, K3, K4, K6}	{K3, K5, K6}
{K1, K3, K5}	{K3, K5}	{K1, K2, K4}
{K5}	{K1, K2, K4, K5, K6}	
{K4, K5, K6}	{K2, K5}	

Figure 3a. Records (only keywords in the records are shown) to be Partitioned into Security Atoms · Keywords K4,K5,K6 are security keywords.



Security atoms sets and their corresponding security keywords:

Security atom 0	{K4, K6}
Security atom 1	{K4}
Security atom 2	{K5, K6}
Security atom 3	{K5}
Security atom 4	{K4, K5, K6}

Figure 3b. The Security Atoms of the Records of Figure 3a.

Security Keyword Conjunct	Security Atoms Satisfying Conjunct
K4	0, 1, 4
K5	2, 3, 4
K6	0, 2, 4
K4 ∧ K5	4
K4 ∧ K6	0, 4
K5 ∧ K6	2, 4
K4 ∧ K5 ∧ K6	4

Figure 3c. Security Atoms Satisfying Boolean Conjuncts of Security Keywords

The concept of security atom can be used to implement a penetration-proof protection mechanism when file sanctions are specified in terms of security keywords only. This is so because of two reasons. First, security atoms are disjoint (i. e. a record will belong to exactly one and only one atom) and second, a file sanction made up of security keywords will apply to either all records of an atom or none at all. Thus, it is easy to create a list of security atom identifiers and the applicable file sanctions (or, better still, the corresponding access privilege sets). Whenever an access is requested, the security atom(s) described by the keywords in the query or record are looked up in the list. If the access is permitted by the access privilege set of the atom(s), then the request is accepted; otherwise it is rejected. It may be argued that a creator may wish to protect his records at the subatomic level or in a manner which effects portions of different atoms. In such cases, full search of the file sanctions is necessary to determine which of the file sanctions are applicable to an access request. Thus, the data model supports two protection mechanisms. The first is geared towards reducing security costs to a minimum, while the other aims at providing maximum flexibility to the user. For the sake of convenience, we shall call the protection mechanism based on security atoms Type A protection mechanism. The other protection mechanism based on full file sanctions search will be called Type B protection mechanism.

From the above discussions, we conclude that the data model specifies three steps by which a record may be evaluated for placement. First, the MAU where the record is to be placed, is determined by the clustering conditions specified by the creator for the record; second, the cluster to which it belongs is determined by the clustering keywords in the record; and finally the security atom (if the creator has chosen to specify file sanction in terms of security keywords) to which a record belongs is determined by the set of security keywords appearing in the record.

3.2 The Basic DBC Operations

The basic DBC operations are security enforcement, record insertion record retrieval and record deletion. We first give a brief description

of these operations and relate them to their supporting components. Then we show in some depth the data structures and algorithms involved in the operations.

3.2.1 The Role of Security Enforcement

The security filter processor (SFP) and the database command and control processor (DBCCP) jointly maintain the database capabilities for the active users of the system. In order for them to correctly enforce a security policy, the proper database capabilities must be provided by the PES. A table is kept for each user with the database capabilities for each active file. Let each table entry have the form:

$$(F,\{(Q_1,A_1),(Q_2,A_2),\ldots,(Q_n,A_n)\})$$

where the set of couples is a database capability.

Commands of the form

(U, (F,Q), a) and

(U, (F,R), a)

pass through the SFP or the DBCCP depending on the type of protection mechanism chosen by the user. If the creator has chosen Type A protection mechanism, the DBCCP converts the file sanctions into a list called the atomic access privilege list (AAPL). The AAPL has the form

$$(U,\ F,\ \{(SAN_1,\ APD_1),\ (SAN_2,APD_2),\ldots,(SAN_p,APD_p)\})$$

where SAN_i is the name of the i-th security atom of the file F and APD_i is the access privilege set associated with SAN_i for the user U. In forming the AAPL, the DBCCP makes use of all the DBC components except the mass memory and the SFP. This results in minimal delay in creating the list. If the creator has chosen Type B protection mechanism , the SFP takes over the maintenance and usage of the file sanctions.

Records are sent into the DBC by way of commands of the form

$$(U,\ (F,R,MCC,\{OCC_i\}),\ \text{"insert"})$$

When such a command is received by the DBCCP, the record to be inserted is

checked for security clearance with the aid of the AAPL (Type A protection mechanism) or the file sanctions. If the result of the check indicates that the record may be inserted, then the DBCCP proceeds with the actual insertion process.

When a command (U, (F,Q), "retrieve") is received by the DBCCP, the query Q undergoes a similar check. If the check is successful, the mass memory is instructed to retrieve the relevant records which form the response set Q(F). Each record in the set Q(F) is tagged with the user identification and file name, (F,U,R). If the user has specified Type B protection mechanism, then the retrieved records are subject to a security check by the SFP before the records are passed on to the PES. This is because the records may contain keywords (in addition to and including those that are required to satisfy the query Q) which satisfy the query parts of file sanctions. The access privilege sets of such file sanctions then become applicable to the records. As a result some of the retrieved records may not be passed onto the user. Such a drop in precision is part of the price a user pays for the wide latitude the system provides in specifying security information.

To execute the command (U,(F,Q), 'delete') the query Q is put through a similar check. If the access "delete" is not granted, the command is rejected. If the access is granted, the mass memory is instructed to proceed with the access. In case of type B protection mechanism, as each record is accessed, it is sent to the SFP for a check against the set of file sanctions. The rationale for this check is the same as the one given for the "retrieve" command. If the check is successful, the mass memory proceeds to delete the record from the database; otherwise, the record is not deleted.

3.2.2. Name-Mapping and the System Components

The retrieve and delete commands both employ Q as a parameter. The subsequent processing of Q that is necessary to execute these commands had the greatest effect in determining the architectural components of the system. We shall now provide an introduction to these components.

A query Q in these commands is in a disjunctive normal form as follows:

$$(T_1^1 \wedge \ldots \wedge T_{n_1}^1) \vee \ldots \vee (T_1^m \wedge \ldots \wedge T_{n_m}^m)$$

where T_j^i are keyword predicates. The i-th conjunct of this query is denoted by Q^i. To form the response set Q(F) the mass memory must be given two arguments: a query Q and a MAU address f. Given these arguments the mass memory will locate all records in M_f that satisfy the query Q. We had earlier seen that each of the index terms in the directory entry of a keyword defined an MAU address f. In the light of later discussion (on clustering techniques and the security atom concept) it became apparent that the index terms must carry information not only about MAU addresses but also about cluster identifiers and security atoms. Thus an augmented directory entry for a keyword K of file f is defined as

$$D(K,F) \equiv \{ (f,c,s) \mid \exists R \ni R \in M_f,\ R \in \text{cluster } c,\ R \in \text{security atom } s \text{ and } K \in R \}$$

The triple (f,c,s) will be called an augmented index term. In cases where the user has chosen Type B protection mechanism the security atom concept is not applicable and the third member of an augmented index term is null. In future discussions, by index terms we will always mean augmented index terms. To obtain MAUs for a conjunct Q^i, all index terms for keywords satisfying each T_j^i of the conjunct must be found. Once found, a set intersection operation is performed over the index terms. The resulting index terms are those whose keywords will make the conjunct Q^i true. The MAU address is derived from these index terms are then used as arguments to retrieve records from the mass memory.

An algorithm which forms the response set Q(F) is given in Figure 4. In the algorithm, we have temporarily ignored security considerations for the sake of clarity. In line 5 of this algorithm, the index terms are fetched from all directory entries (D(K,F)) whose keyword K satisfied T_j^i and are placed in a set $\omega(j)$. In line 8 MAU addresses are extracted. In lines 3-6, one set $\omega(j)$ is formed for each keyword predicate

0. **begin**
1. **For** $i = 1,2,\ldots,m$ **do**
2. **begin**
3. **For** $j = 1,2,\ldots,n_i$ **do**
4. **begin**
5. $\omega(j) \equiv \{(f,c,s) \mid K \text{ satisfies } T_j^i \text{ and } (f,c,s) \in D(F,K)\}$
6. **end**
7. $\theta(i) \equiv \bigcap_{k=1}^{n_i} \omega(k)$
8. $\theta'(i) \equiv \{f \mid (f,c,s) \in \theta(i)\}$
9. **end**
10. $\Sigma \equiv \{(Q^k,f) \mid f \in \theta'(k)\}$
11. $Q(F) \equiv \bigcup_{(Q^k,f)\in\Sigma} \{R \mid R \in M_f \text{ and } R \text{ satisfies } Q^k\}$
12. **end**;

Figure 4. A Name Mapping Algorithm

T_j^i in Q^i. Then in line 7 these sets are intersected to give the set $\theta(i)$. In line 8, the MAUs to be searched are extracted from the index terms obtained in line 7. (Line 7 carries out an intersection operation since the keyword predicates of Q^i are ANDed together.) In lines 1-9, a set $\theta(i)$ is formed for each conjunct Q^i and finally in lines 10-11 the records are retrieved from the mass memory. In line 10 the response set is defined as the union of the following sets

$$\{ R \mid R \in M_f \text{ and } R \text{ satisfies } Q^k \}.$$

This algorithm also shows how the data structures defined in the data model are used for name-mapping. The content addressability employed by the DBC will, in fact, allow the actual realization of the data structures to be just as simple as those illustrated here.

This algorithm shows us what the structure memory must do. The structure memory must store directory entries and be able to accept a keyword predicate T and retrieve all index terms for all keywords which satisfy T_i (as in line 5). Clearly, the structure memory will also have to be able to add, delete and modify directory entries as well. It also shows us the nature of the structure memory information processing, namely, set manipulation (line 7). These observations help us outline the architecture of the DBC. The DBC contains at least six functionally specialized components: the database command and control processor (DBCCP), the security filter processor (SFP), the mass memory (MM), the structure memory (SM), the structure memory information processor (SMIP), and the index translation unit (IXU). The DBCCP is responsible for translating DBMS commands into lower level commands for the mass memory and coordinating the actions of the other components. The MM contains DB, the SM stores the directory entries and the SMIP is a set operation processor. The index translation unit is responsible for extracting the MAU addresses from the augmented index terms. The organization of these components to a first order detail is shown in Figure 5. We shall see later in Part II that a seventh component, namely the keyword transformation unit (KXU), is needed from the point of view of an efficient physical realization of the DBC.

3.2.3 The Operation of the SM, SMIP and MM

The theory of operation continues with an exposition of the operating

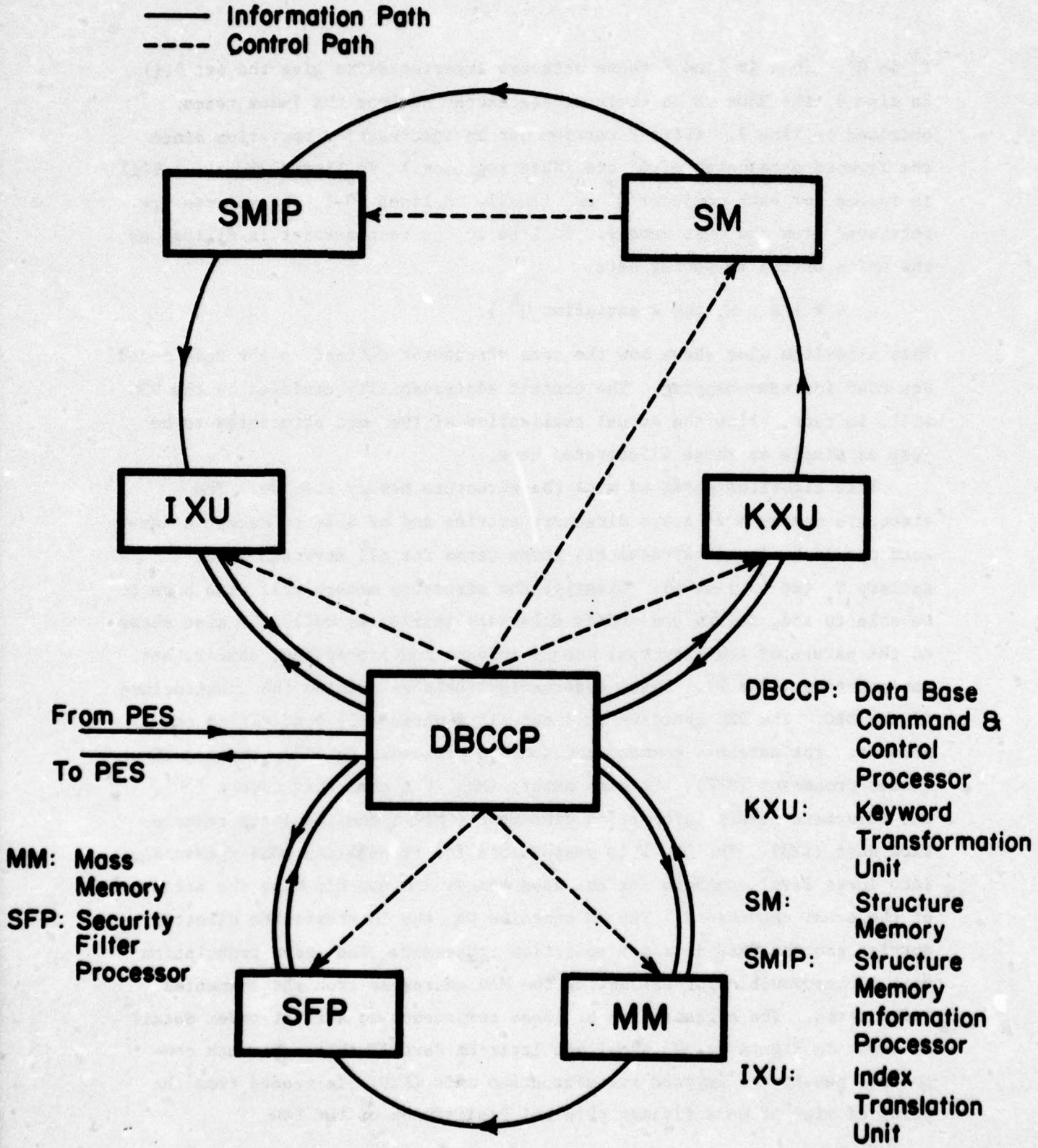


Figure 5. Architecture of DBC

principles of the structure memory, structure memory information processor, and the mass memory. The carefully tailored functional characteristics of these components allow them to readily carry out numerous steps of DBS algorithms to be given. The description of the components that follows is only conceptual in nature, the actual hardware organization used to realize them is given in Part II and Part III of the paper.

A. Structure Memory

The SM is the repository of the directories of the files in the DB. Each index term (f,c,s) of D(F,K) is stored in the SM as the tuple (F,K,f,c,s). The contents of the SM may therefore be viewed as a set, known as structural memory basis SMB, of such tuples defining the directories of all files.

The SM retrieve command has the form SM<retrieve,(F,T)> where F is a file name and T is a keyword predicate. The command is carried out by constructing a set containing all index terms of each directory entry D(F,K) whose keyword K satisfies T. Formally, the SM executes the command SM<retrieve,(F,T)> by outputting the set

$$\{(f,c,s) \mid (F,K,f,c,s) \in SMB \text{ and } K \text{ satisfies } T\}$$

The insert command has the form SM<insert,(F,K,f,c,s)> and is executed by adding (f,c,s) to the set D(F,K). In other words, the insert command is executed by replacing SMB with SMB $\cup$ (F,K,f,c,s).

The delete command has the form SM<delete,(F,K,f,c,s)> and is executed by removing (f,c,s) from D(F,K). Formally, the deletion command is executed by replacing SMB with SMB - (F,K,f,c,s).

To model its operations the SM can be viewed as a PCAM with M content-addressable blocks. The SM partitions the set SMB into N subsets, designated SMB_i, $0 \leq i < N$, where $N \leq M$. Each subset is stored in one or more blocks of the PCAM.

The retrieve command is executed by first applying to T a hash function which maps it into an integer j where $0 \leq j < N$. Then the set SMB_j is searched by accessing the appropriate block(s) of the PCAM to locate and retrieve the tuples (F,K,f,c,s) whose keyword K satisfies T.

Insert and delete commands are executed by applying to K a hash function which maps it into an integer j. The tuple (F,K,f,c,s) is then added to or removed from the subset SMB_j by accessing the appropriate block of the PCAM.

The nature of the hash function will strongly influence the kinds of keyword predicates that may be used by the system. This issue along with a description of how the sets SMB_j are stored in the PCAM and how SM and its PCAM is realized are addressed in Part II of this paper.

Consideration is now given to the fact that the SM is a two-level system containing a directory entry storage and a look-aside buffer. We now extend the aforementioned operations to the two-level SM. Let the directory entry storage be represented by the set SMB defined above. The time required to update this set (i.e., add or delete an element) is fairly long compared to the time required to update, say, a fast access semiconductor RAM memory. The look-aside buffer allows SM update operations to appear as though they were executed immediately.

The look-aside buffer may be conceptually represented by an ordered set LKA of SM update commands:

$$\text{command}_1, \text{command}_2, \ldots, \text{command}_k$$

where command_i preceeded command_{i+1} in time. The look-aside buffer has two functions: it acts as a command queue for the SM and it contains the information which allows the SM to appear updated. The look-aside buffer would be realized with high-speed random access memory and so its access time would be much less than that of the directory entry storage.

Whenever an update command is received by the SM it is placed in LKA. If an insert (delete) command negates the effect of a previous delete (insert) command then the insert (delete) command is not added to LKA. and the negated delete (insert) command is removed from LKA.

To execute a retrieval command the two level SM first examines LKA for commands which add index terms (f,c,s) to directory entries D(F,K) whose keyword K satisfies T. All index terms so found are output. Then the set SMB is searched for additional index terms. When an index term (f,c,s) of a directory entry D(F,K) whose K satisfies T is retrieved from SMB it is checked in the following way: If there is a command in LKA to delete (f,c,s) then that index term is not output from SM.

B. Structure Memory Information Processor

The SMIP is a processor for set manipulation. Set manipulation

operation are performed by maintaining an intermediate set in the SMIP while the argument sets which modify it are passed through the SMIP. The SMIP's intermediate set is designated SW and consists of couples (m,d) called SMIP data units. The first part m of the couple is called the key and the second part d is called the data. Operations are performed on SW by identifying a SMIP data unit and by performing an operation on it. There are two kinds of SMIP commands. The first kind of SMIP command is represented by SMIP<m,g> where m is a key and g is a manipulation function. The manipulation function can do two things: first, it can specify how the data part of a SMIP data unit (m,d) with key m should be modified; and second, it can specify what should be done if no SMIP data unit with key m is in SW. When no SMIP data unit with key m is found and no action is specified by g then SMIP takes no action. The second kind of SMIP command has of the form SMIP<g> where g specifies an action that is to occur.

To illustrate the set manipulation functions, let us show how the SMIP can be used to perform an N-set intersection. Let X_i represent one of these N sets and let x_{ij} represent an element of X_i. The algorithm which performs the intersection is shown in Figure 6.

In lines 1-4 of the algorithm a SMIP data unit of the form $(x_{1i}, 1)$ is created for each element of X_1. In steps 5-11 each element of the sets $X_2, X_3, \ldots, X_n$ is examined and whenever a matching SMIP data unit is found its data part is incremented by 1. When these steps are completed, SW contains SMIP data units which indicate in how many of the sets X_i each element of X_1 appears. Those elements appearing in all sets make up the set $X_1, \ldots, X_n$. In line 12 all such elements are retrieved from the SMIP.

The SMIP is also realized with a PCAM. To model the operation of the SMIP, a PCAM with M content-addressable blocks is used. The SMIP partitions the set SW into N subsets designated SWB_i where $N \leq M$. Each subset is stored in one or more blocks of the PCAM.

The command SMIP< m,g > is executed by applying to m a hash function which maps it into an integer j where $0 \leq j < N$. Then SWB_j is searched for a SMIP data unit with the key m. If it is found, g is applied to its

```
0.  begin
1.  For each element x_1i of X_1 do
2.      begin
3.          execute the command SMIP<x_1i, "create (x_1i,1)">
4.      end
5.  For j = 2, 3, ..., N do
6.      begin
7.          For each element x_ji of X_j do
8.              begin
9.                  execute SMIP<x_ji, "replace (x_ji,d) with (x_ji,d+1)">
10.             end
11.     end
12. Execute the SMIP<"retrieve the key m from all (m,d) where d = N">
13. end
```

Figure 6. An N-set Intersection Algorithm Using the SMIP

data part. If no SMIP data unit is found, then any other action that g specified is carried out on SWB_j. The command SMIP<g> is executed by ordering each block of the SMIP PCAM to perform the operation specified by g. In Part II of this paper we shall go into detail about the construction of the SMIP and its PCAM.

C. Mass Memory

The MM is the repository of the database itself. To retrieve data from the DB, queries and MAU addresses in which data reside must be given to the MM. (These MAU addresses are normally provided by the SM, SMIP and IXU after processing a given query Q.)

Mass memory commands have two forms. The first form MM<a,U,(F,Q),f> specifies an access type a, a user U, a query Q, and a MAU address f. It is executed by performing access a on the records in M_f satisfying Q. While executing this command the MM may use the SFP to validate an access. The second form, MM<a,U,R,f>, is used to insert a record R into M_f.

3.2.4 The Execution of Record Operations

In this section we discuss the record insertion, deletion and retrieval operations. We indicate the basic requirements of these operations and give algorithms to show how these operations are carried out by the DBC's components.

Record retrieval is simply the formation of the response set Q(F); an abstract algorithm to do this has already been given in Figure 4. An equivalent algorithm, using explicit SM, SMIP and MM commands is given in Figure 7. This algorithm executes the DBC retrieval command <U,(F,Q), "retrieve"> as discussed in Section 2.2, where Q has the form

$$(T_1^1\ T_2^1\ \ldots\ T_{n_1}^1) \vee \ldots \vee (T_1^m\ T_2^m\ \ldots\ T_{n_m}^m).$$

```
1.  begin
2.    Q(F) ≡ { }
3.    Σ ≡ { }
4.  For i = 1, 2, ..., m do
5.    begin
6.      α_1 ≡ SM<F,T_1^i>; SMIP<"reset">
7.      For every element (f,c,s) in α_1 do
8.        begin
9.          SMIP<"create ((f,c,s),1)">
10.       end
11.     For j = 2, 3, ..., n_i do
12.       begin
13.         α_j ≡ SM<F,T_j^i>
14.         For every element (f,c,s) in α_j do
15.           begin
16.             SMIP<(f), "replace ((f,c,s),x) by ((f,c,s),j) if x = j-1">
17.           end
18.       end
19.     ω ≡ SMIP<"output key (f,c,s) for all elements ((f,c,s),d), where d=n_i">
20.     Σ ≡ Σ U {(Q^i, (f,c,s) | (f,c,s) ε ω}
21.   end
22. Σ' ≡ ( )
23. For every element (Q^k, (f,c,s)) in Σ do
24.   begin
25.     IXU<Extract (f) from (f,c,s)>
26.       Σ' ≡ Σ' U ((Q^k, (f))
27.   end
28. For every element (Q^k, (f)) in Σ' do
29.   begin
30.     Q(F) ≡ Q(F) U MM<retrieve, U, (F,Q^k), (f)>
31.   end
32. end
```

Figure 7. An Algorithm to Form Q(F)

In lines 5-21 of the algorithm, one conjunct Q^i of the query is processed. In lines 6-10 all index terms for T_1^i are fetched from the SM and data elements are loaded into the SMIP. The command SMIP<"reset"> clears the SMIP storage areas. In lines 11-18, all index terms for each T_j^i are fetched from the SM and the elements of all of these sets are intersected in the SMIP. The SMIP command in line 16 insures that multiple occurrences of the same index term for one keyword predicate will not affect the intersection process by incrementing the occurrence count x of a SMIP data unit at most once for each T_j^i. In line 19 all index terms in the intersection are retrieved and in lines 22-27 they are used to build a set containing the information needed to issue MM commands. In lines 3-21 all conjuncts are processed and all of the information needed to access the MM is then placed in Σ. Finally, in lines 28-32 the MM commands are formed and executed resulting in actual retrieval of the records.

The SM, SMIP, IXU, MM and DBCCP would operate in parallel to execute the above algorithm. The DBCCP would execute the control statements of the algorithm while the other components are executing commands. To do this the DBCCP would order a component to execute a command and then continue to process the algorithm as far as possible. The SM, SMIP and IXU operate in a tightly coupled parallel fashion. Whenever an element is output from the SM (line 6 or 13) it is sent directly to the SMIP where it is processed (line 9 or line 16). Whenever an element is output from SMIP (as in line 19) it is directly sent to the IXU for translation. [This parallelism is not evident in the algorithm]. The MM can also operate in parallel with other elements. The controller can create MM commands on-the-fly when it executes line 26 and send them directly to the MM. This technique would be equivalent to the operations specified in lines 26 and 28-32. The MM could also execute commands for other queries while this algorithm was being executed. The SFP would, of course, also be executed in a parallel with the algorithm to check (if necessary) all records leaving the DBC.

Record insertion requires three major operations. First, the MAU which is to contain the record is chosen. Then, the information in the

structure memory is updated. Finally, the record is placed in the mass memory. The last two operations are carried out by the structure memory and the mass memory, and we shall discuss them later in the paper.

The MAU selection operation is a two step process. First, all MAUs satisfying the mandatory cluster condition associated with the record are found; second, the MAUs are examined to find the one with the greatest optional clustering weight. Since both the MCC and the set of OCCs are queries made of clustering keyword predicates, the SM, SMIP and IXU can once again be used in parallel to extract the required MAU address(es). An algorithm which does this is given in Figure 8. In this algorithm the mandatory clustering condition for the record to be inserted is represented by

$$MCC \equiv MCQ_1 \vee MCQ_2 \vee \ldots \vee MCQ_m$$

where MCQ_i is a conjunct of clustering keyword predicates of the form

$$(CKP_1^i \wedge CKP_2^i \wedge \ldots \wedge CKP_p^i)$$

There are n optional clustering conditions (OCC_1, OCC_2, ... OCC_n) each of which is in the disjunctive normal form. Each OCC_i is associated with a cluster weight cw_i.

In lines 1-5, the set of index terms (f,c,s) whose cluster component identifies the cluster satisfying the conjuncts of the MCC are determined. In line 6, all of these index terms are identified in one set ω, and in line 7, the MAUs are extracted and placed in ω'. The SM and SMIP are used in line 4 to obtain the index terms while the IXU is used in line 7 to extract the MAU. In lines 9 thru 12, the OCCs are processed in a similar fashion by the SM, SMIP, and the IXU to produce the set θ. In line 13 and 14 the total cluster weights associated with each f in ω' are calculated. Finally in line 15 the MAU with the largest weight is chosen.

Like the record retrieval algorithm, this algorithm requires three basic operations - directory entry retrieval, set intersection and index translation.

The process of physically adding a record or removing a record from the mass memory is quite simple due to its content addressability. The

Let $MCC \equiv MCQ_1 \vee MCQ_2 \vee \ldots \vee MCQ_m$

$MCQ_i \equiv CKP_1^i \wedge CKP_2^i \ldots \wedge CKP_p^i$

$OCC_i \equiv OCQ_1^i \vee OCQ_2^i \ldots \vee OCQ_q^i$

1. **begin**
2. **for** i = 1, 2, ... m **do**
3. **begin**
4. $\omega_i \equiv \{ (f,c,s) \mid$ cluster $c \in M_f$ and $\forall R \in c$ = R satisfies $MCQ_i \}$
5. **end**
6. $\omega \equiv \omega_1 \cup \omega_2 \cup \ldots \cup \omega_m$
7. $\omega' \equiv \{ f \mid (f,c,s) \in \omega \}$
8. **for** i = 1, 2, ... n **do**
9. **begin**
10. $\theta_i \equiv \{ (f,c,s) \mid$ cluster $c \in M_f$ and $\forall R \in c$ = R satisfies $OCC_i \}$
11. $\theta_i' \equiv \{ f \mid (f,c,s) \in \theta_i \}$
12. **end**;
13. **for** every f in ω' **do**
14. $OW_f = \sum_i \begin{cases} cw_i \text{ if } f \in \theta_i' \\ 0, \text{ otherwise} \end{cases}$
15. Find $f_r \ni OW_{f_r} \geq OW_f \ \forall f \in \omega'$
16. **end**

Figure 8. An Algorithm to Choose an MAU for a Record

deletion process may, however, also require modification of the structure information in the SM. This will occur whenever the record deleted is the last record in a cluster that contains certain keywords. When a record R in cluster c, security atom s, and MAU f is removed from M_f, the index term (f,c,s) must be removed from each directory entry D(F,K) in SM where K is a keyword appearing in R but in no other record in the cluster c. To handle this operation we provide the mass memory with the capability of determining whether or not a keyword appears in more than one record of a cluster. Full details of the insertion and deletion algorithms are discussed in Part II and Part III of this paper.

4. THE TECHNOLOGY OF THE DBC

The DBCCP, SFP, and the IXU are conventional processors that would be specially microprogrammed for their task. The SMIP and the MM employ currently available technology in a new way. The SM can also be built with available technology but the most powerful SM organizations employ new technology that will become available in the near future.

The SM is most dependent on technological developments. Its PCAM could be built today by using a fixed-head disk as the storage medium. Each block of the PCAM would be stored on one or more tracks of the disk. The memory would be accessed by reading and searching the track(s) representing a block. This organization would have two limitations: First, the block access time would be relatively slow (5ms or greater); this is a potential system bottleneck. Second, the PCAM would consist of many relatively small blocks and so only equality predicates could be readily handled by the SM. This is because the small block size implies small hash table buckets which, in turn, implies that the hash function must be used for exact-match searches. This is because inequality searches would cause access to large number of small blocks.

The rapid development of electronic bulk memory technologies (CCDs and RAMs [16] magnetic bubbles [17,18] and electron beam memories [15]) may make an all-electron fixed-head disk replacement available very soon. This would allow the construction of a much faster PCAM-based SM which would not be a bottleneck. An "electronic-disk" PCAM would still, however, have many small blocks and so would suffer from the same keyword predicate limitations as a fixed-head disk PCAM.

The availability of cheap and very powerful microprocessors opens the way to a very powerful PCAM organization. This PCAM consists of a small number of very large content-addressable blocks and is realized by a large number of microprocessor-memory pairs as shown in Figure 9. This kind of PCAM would be capable of supporting a much greater variety of keyword predicates. This is because all keywords of a given attribute could probably be stored in a single PCAM block and so, therefore, any predicate could be applied to all keywords of that attribute with a single access.

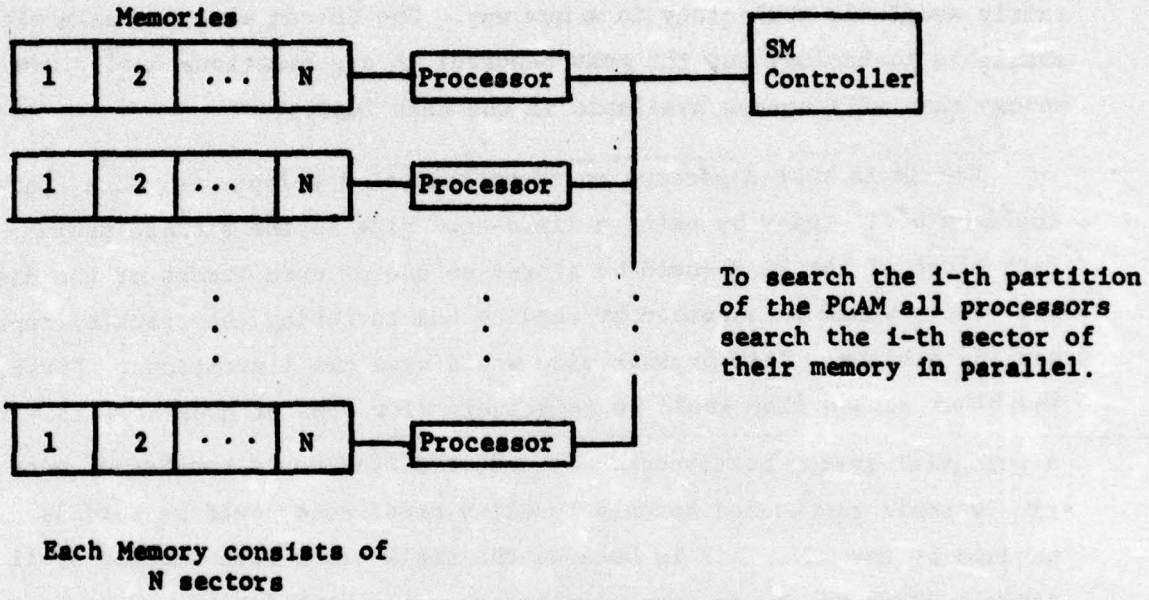


Figure 9. The Architecture of the Structure Memory

Since almost all keyword predicates used in a database system would, in all likelihood, only deal with a specific keyword attribute, it follows that most keyword predicates likely to appear in a query would be processable by the SM.

The SMIP (see Figure 10) is primarily a processing element and is consequently not limited by memory technology. The small amount of memory required by this component can be realized with current technology. The SMIP achieves very high speed by using many processor-memory pairs to execute operations in parallel. The SMIP is feasible with today's technology and could become quite inexpensive in the future as RAMs and microprocessors become cheaper.

The mass memory (see Figure 11) uses a moving head disk to realize a PCAM. Each cylinder of the disk represents one PCAM block. For high performance, all of the data on a cylinder is accessed in parallel, searched and stored in a buffer in a single disk revolution. The mass memory therefore uses a cylinder-size interleaved buffer memory, multiple read/write assembly registers and a fast processing unit. This can be done with current technology.

A detailed description of the logical and physical structure of the computer is given in Part II and Part III.

For maximum applicability to current computer systems we designed an independent DBC with a very simple interface. The DBC directly implements the attribute-based data model and a very powerful query language based on Boolean expressions of keyword predicates. The DBC supports a security specification called a database capability. This construct allows access privileges to be given to sets of records named by queries. The power of the database capability comes from its ability to protect any data item that is retrievable.

The theory of the DBC's operation was based on clusters of records of like properties stored in MAUs (Minimal Access Unit). The structure memory contained directory entries that enabled the DBC to determine the MAUs where records characterized by keywords were to be found in the database. The nature of the directory entries and the structure of the queries determined the properties of the access algorithms. These properties, in turn, influenced the structure of the machine. They showed us

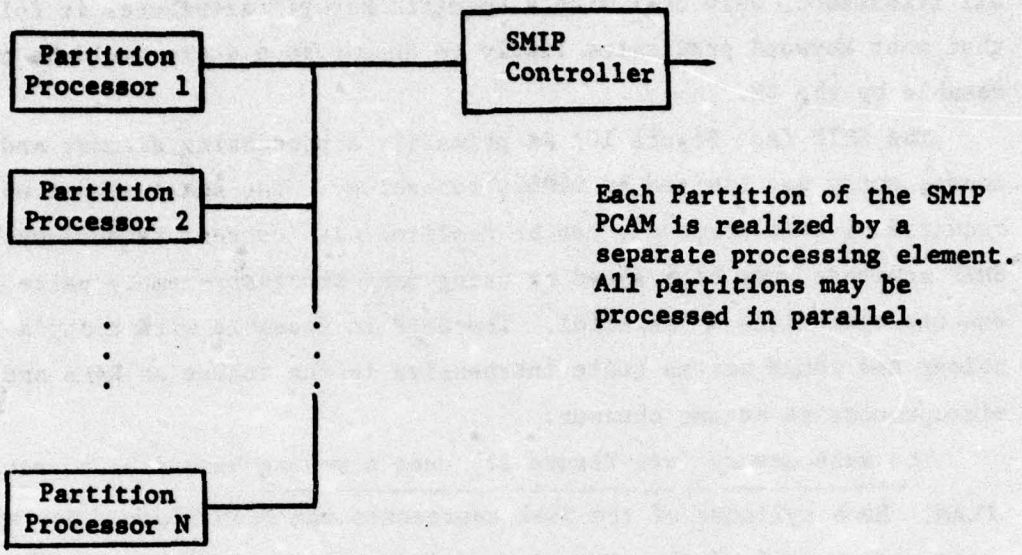


Figure 10. The Architecture of the SMIP

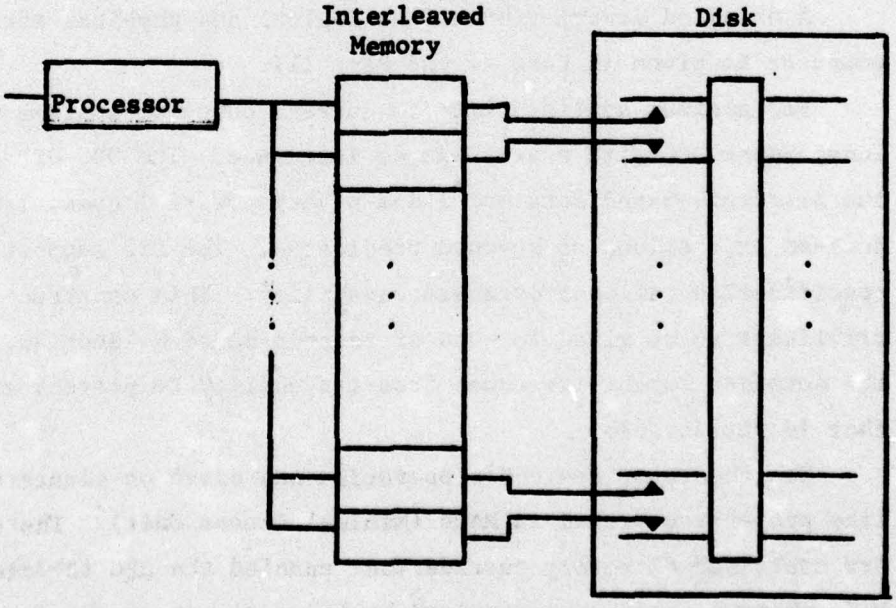


Figure 11. The Architecture of the MM

the need for a high-speed set manipulation processor, a structure memory that could process keyword predicates, and a mass memory that could properly support MAUs. They also showed us the need for a way to manage the MAUs of the database. To supply this last requirement the DBC supports a sophisticated clustering mechanism that allows records to be automatically assigned to MAUs. The other requirements were met by a specialized component (the SMIP) to do set operations, a PCAM-based structure memory with very large-capacity partitions that could use hashing to handle a broad class of keyword predicates and a mass memory in which MAUs were parititioned into clusters to distinguish records with different sets of properties from one another within the mass memory. Security enforcement was realized by two mechanisms. The first mechanism, introduced to enhance performance, utilized the concept of security atoms to form clusters of records that were protected the same way. The second mechanism (the security filter processor) used the actual file sanctions to enforce security. These two types of security mechanisms allow security specifications to be readily processed and thoroughly enforced.

Part II and III of the paper gives detailed specifications of the data and instruction formats of the database computer and its componenents, the structure, speeds and capacities of the components and the technology required to build the machine. It will be shown there that the architectural principles used in the database computer do not require distant technology and so can be realized in the near future. Preliminary study on how the DBC should support higher-level data models (such as the network model) is underway. Early work shows that the propsoed DBC can indeed support high-level data models.

References

1. Coulouris, G.F., Evans, J.M and Mitchell, R.W., "Towards Content Addressing in Data Bases," Computer Journal 15, 2 (February 1972), 95-98.
2. Su, S.Y.W. and Lipovski, G.J., "CASSM: A Cellular System for Very Large Data Bases," Proceedings of Very Large Data Base Conference, Sept. 1975, 456-472.
3. Ozkarahan, E.A., Schuster, S.A. and Smith, K.C., "RAP--Associative Processor for Data Base Management," AFIPS Conference Proceedings, 44 (1975), 379-388.
4. Moulder, R., "An Implementation of a Data Management System on an Associative Processor," Proceedings of the AFIPS National Computer Conference 42 (1973), 171-176.
5. DeFiore, C.R. and Berra, P.B., "A Data Management System Utilizing An Associative Memory," Proceedings of the AFIPS National Computer Conference, 42 (1973), 181-185.
6. Minsky, N., "Rotating Storage Devices as Partially Associative Memories," Proceedings of the AFIPS Fall Joint Computer Conference, 41, (1972), 587-596.
7. Lin, S.C., Smith, D.C.P. and Smith, J.M., "The Design of a Rotating Associative Memory for Relational Database Applications," ACM Transactions on Database Systems, 1, 1 (March 1976), 53-65.
8. Stellhorn, W.H., "A Specialized Computer for Information Retrieval," University of Illinois Department of Computer Science Report No. R-74-637, October 1974.
9. Hollaar, L.A., "A List Merging Processor for Information Retrieval Systems," Presented at the Workshop on Architecture for Non-Numerical Processing, October 1974, Dallas Texas.
10. Berra, P.B. and Singhania, A.K., "A Multiple Associative Memory Organization for Pipelining a Directory to a Very Large Data Base," Digest of Papers COMPCON 76, 109-112.
11. Canaday, R.H., Harrison, R.D., Ivie, E.L., Ryder, J.L. and Wehr, L.A., "A Back-End Computer for Data Management," Communications of the ACM 17, 10 (October 1974), 575-582.

12. Hsiao, D.K. and Harary, F., "A Formal System for Information Retrieval from Files," Communications of the ACM 17, 10 (February 1970), 67-73.

13. Hsiao, D.K., Syatems Programming - Concepts of Operating and Data Base Systems, Chapter 6, Addison-Wesley, 1975.

14. McCauley, E.J. III, "A Model for Data Secure Systems," Ph.D. Dissertation, Department of Computer and Information Science, The Ohio State University (1975). Available as Research Center Report OSU-CISRC-TR-75-2.

15. Hughes, W.C. et. al., "A Semiconductor Nonvolatile Electron-Beam Accessed Mass Memory," Proceedings IEEE, 63, 8 (August 1975), 1230-1240.

16. Hodges, D.A., "A Review and Projection of Semiconductor Components for Digital Storage," Proceedings IEEE, 63, 8 (August 1975), 1136-1147.

17. Bobeck, A.H., Bonyhard, P.I. and Geusic, J.E., "Magnetic Bubbles-- An Emerging New Memory Technology," Proceedings IEEE 63, 8 (August 1975) 1176-1195.

18. Cohen, M.S. and Chang, H., "The Frontier of Magnetic Bubble Technology," Proceedings IEEE 63, 8 (August 1975) 1196-1206.

19. Baum, Richard I., "The Architectural Design of a Secure Data Base Management System." Nov. 1975, Ph.D. Dissertation, The Ohio State University, Tech. Report No. OSU-CISRC-TR-75-8.

20. Baum, Richard I. and Hsiao, David K., "Database Computers -- A Step Towards Data Utilities," IEEE Transactions on Computer, Dec. 1976, Vol. C-25, No. 12.