THE UNIVERSITY OF MANITOBA

Database Systems

by

Patrick N. Smith

A thesis
submitted to the Faculty of Graduate Studies
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Computer Science

WINNIPEG, MANITOBA

May 1974

TABLE OF CONTENTS

CHAPTER 4

CHAPTER 5

ABSTRACT

The development of Management Information Systems and the subsequent recognition of the need for Generalized Data Base Management Systems (GDBMS) are traced. Data structures and access techniques used in database applications are presented and compared. The work of the CODASYL Data Base Task Group, and in particular their April 1971 report, is discussed and explained. A plan for implementation of the specifications in the report is presented and an outline of work done on this implementation is given. Future systems and their requirements for GDBMS software and hardware are presented to indicate a direction for future research.

ACKNOWLEDGEMENTS

Thanks is due to C. Robert Zarnke, Micheal Doyle and Paul Dirksen my thesis committee for their time spent reading and commenting on this rather long thesis.

Special thanks to Brent D. Beach for the many hours he spent discussing the CODASYL report and my thesis, and for the version of XPL used in the implementation.

1.1 HISTORICAL DEVELOPMENT OF DATABASE

In the past when businesses were small and communication slow, it was possible for an accounting department to take each accounting entry and carry it through by manual means to its final updating. When the bookkeeper closed the ledger on his desk at the end of the day, the last figure he entered was the true balance and position of the business.

As businesses became more intricate - with branch offices scattered about the country - accounting became more complex. Mechanization appeared and the bookkeeper was supplied with calculating equipment, accounting machines, and eventually punched card equipment. In order to make punched card equipment more universal, the manufacturers implemented each function in a separate unit-record machine. For the sake of efficiency, all similar items, such as payroll, accounts receivable, sales analysis, etc., were processed in batches. It became impossible to summarize a business activity at the close of each day. Instead, it had to be done periodically - weekly, semi-monthly, monthly or quarterly. Batches of information pertaining to the reports were accumulated and run in accordance with deadline requirements.

As business grew and batches became larger, more and faster equipment was needed. Eventually, the unit-record machines could not keep up with the pace. The time came for the introduction of the electronic computers. Initially computerized systems used the same concepts as previous unit-record systems. The power of the computer served only to speed up the processing.

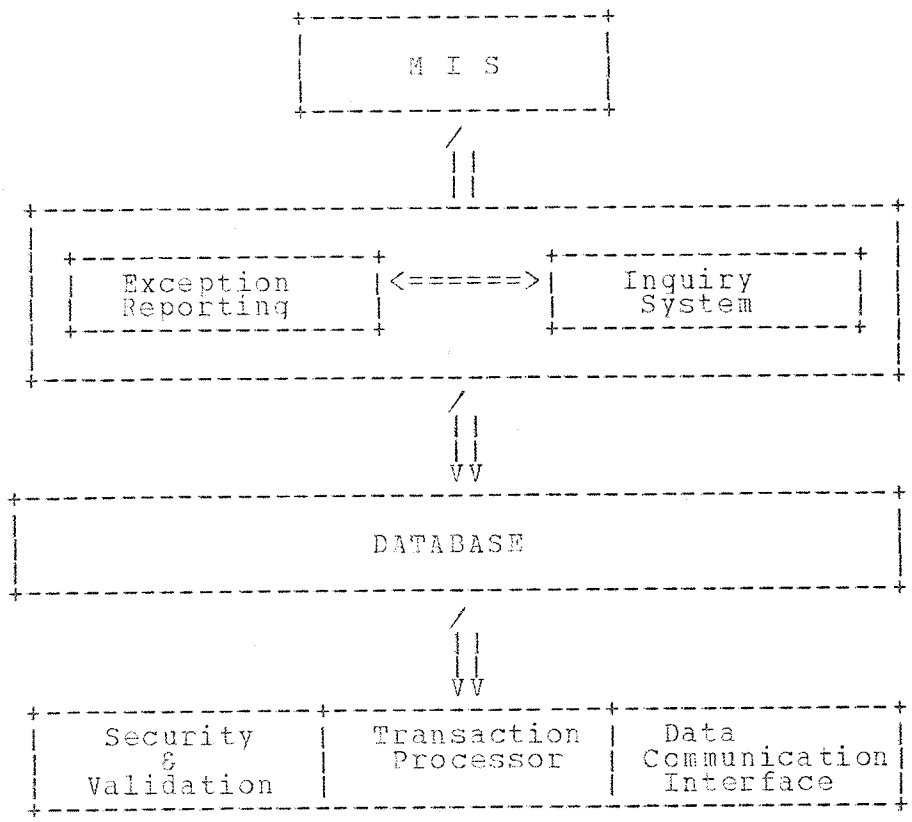
In the early 1960's managers realized that they were being swamped by an ocean of reports, which, while containing all the details of business, failed to provide a conceptual view of the state of the business. Management began to use the computer to sift through the mass of details and to produce summary information pertinent to the managerial function.

Attention was now drawn to a disadvantage of cyclical processing. All management reports were historical; that is, they reflected what happened a week ago, a month ago, or in the last quarter. Frequently

the reports were too late to enable a timely decision on the information they contained. In addition, reports from individual departments using the same original data were inconsistent because the cut-off date for transactions varied throughout the organization. As a result, management called for interim reports, usually generated by hand, in an attempt to provide up-to-date consistent information.

In the mid 1960's management, aided by enterprising systems analysts within their organizations, recognized the need for timely (real time) management reporting. The result of this recognition was the specification of requirements for an Integrated Management Information System (MIS). A MIS is a system designed to provide significant information needed by management in the planning, direction and control in areas of direct responsibility, in sufficient time to take effective action. A MIS incorporates the processing modules of previous batch systems into a generalized information collection, editing, maintenance and reporting facility. The interfacing of the modules is conceptualized in the diagram below.

MAJOR DESIGN ELEMENTS OF AN INFORMATION SYSTEM



If we define information as that part of data which can be used to increase knowledge, then it is reasonable to expect that that a MIS provide information not just data. The heart of a MIS is its Database Management System (DBMS). The usefulness of a MIS is largely dependent on the implementation of its DBMS.

"A typical database system has the appetite of an elephant and the output of a constipated mosquito". This comment, made by a disgruntled user, T. Bolivar Shagnasty in a recent SHARE publication, reflects the attitude of many computer professionals regarding presently implemented DBMS. It is probably also an indication that systems analysts are not database wise. Many systems analysts design with the unit record mentality of previous systems. They have not grasped the potential power of database technology.

New applications require the use of data from more than one system to be manipulated in sequences and formats different from those required by the original processes. The sequential nature of unit-record processing and the binding of processes to data is not suitable. It is the responsibility of a DBMS to remove from the process the physical description of data formats and organization. In its place, a logical view of the data is used by each process. This logical view is mapped by the DBMS to the physical database. Data items are accessed via this mapping. In a MIS the DBMS must provide facilities to maintain logical relationships between data items which allow information processing in response to MIS requests. The following examples indicate the types of requests which might be entered into the MIS.

> Give me a list of the names and employee numbers of all salesmen in the eastern region who have sold in excess of $100,000 of product ABC in the past two months.

or

> Whenever the level of inventory of products ABC and ZZZ falls below 100 and no supplies are in shipping send me a message.

These requests require the use of more than one record type and file. Both may require access to information stored in various sequences and formats. To handle these requests effectively and efficiently a facility must exist which does not require a complete sequential scan of each file to collect the required information. Such a facility is provided by a GDBMS.

The design criteria for a Generalized Database Management System (GDBMS) is that it provide a generalized, common and integrated collection of organizational data to fulfill the data requirements of all applications which access it. Generalized implies an ability to access data in a manner consistent with its data content. Common implies a non-redundant storage of data items. Integrated implies that all data is logically filed together; that is, an artificial structuring of related data into disjoint files is not required. Further, it is possible to create logical structures involving any or all data items in the database.

As yet no formal definition exists for database. However, one can look at the component parts of a database and say what the database does and does not look like. A database is the union of Physical Storage Structure, Data and Control Information, and Logical Relationships. A database is not a collection of data files for several related applications. The data within the database should, perhaps even must, be structured to model the natural data relationships which exist within an organization.

In answer to T. Bolivar Shagnasty's complaint, it is useful to ask and answer the question, " Why use a database?".

1. A database provides consistency of reporting. Whenever a data item is referenced the same information is returned since the data item is stored uniquely.
2. A database may provide more information and less data through the structuring of data. Data may be information only when it relates to other data items in a defined manner.
3. A database may provide data processing cost savings for a user in two areas:
   (a) The collection, editing and maintenance operations are executed only once for each data item since it is

stored uniquely. Non-database systems process data items redundantly for each file in which the data item appears.

(b) System implementation and maintenance costs may be reduced. There is no need to design and implement file systems for each new application. Data structure and content change may be invisible to application programs.

4. Data in a database can be made transparent to the hardware and software using the data. In a dynamic or multi-system environment this can represent considerable savings in conversion costs while making information more readily available throughout the organization.
5. Data security and integrity is enhanced in a database by the controlled access to data via a GDBMS.

The failure of MIS's is partly the result of overly optimistic predictions on the part of system designers and partly the result of a hardware and software immaturity which produces DBMS'S which do not satisfy the aforementioned design criteria. Recently, newer GDBMS's have been developed which attempt to provide the facilities lacking in the older systems.

1.2 BIBLIOGRAPHY

MATTHEWS, D.O., The Design of the Management Information System.
pub by Auerbach 1971 Cat # 77-124629.

MURDICK, R.G, and ROSS, J.E.
Information Systems for Modern Management
pub by Prentice-Hall 1971 Cat # 70-130842.

SHARE Secretary Distribution
"Data Base" A Users View
SSD # 237 June 3.0,1973. pp 78-117.

2.1 DATA BASE CONCEPTS

Recently considerable new literature has appeared describing the function and structure of databases. Unfortunately each author has defined a new terminology to describe his ideas. This requires that the reader cross reference familar and new systems in order to evaluate the new concepts. Often it turns out that the new ideas are only old ideas masked in new terminology. One of the significant advantages to be gained by the publication of the CODASYL DATA BASE TASK GROUP REPORT of APRIL 1971 may be the establishment of a consistent terminology to be used by all authors of articles on database systems. Throughout this thesis descriptions which apply to a class of databases or database management systems will use the terminology of the CODASYL report.

2.2 BACKGROUND AND OBJECTIVES OF CODASYL

On May 28 and 29, 1959 a meeting of major data processing users and American government officials was held for the purpose of considering both the desirability and feasibility of establishing a common language for the programming of electronic computers to solve problems relevant to the business data processing environment. Large users, and particularly the U.S. government, were becoming concerned about the lack of program and data transferability between computers. Systems developed for one set of hardware were quickly made obsolete by new hardware and could not be used on computers produced by other manufacturers. While FORTRAN had become the de-facto standard for scientific applications, by virtue of its being the first reasonably efficient high-level language, no such language standard existed in the data processing area. IBM, SPERRY-RAND, and HONEYWELL were all pushing languages of their own design. The outgrowth of this concern and the May 1959 meeting was the organization of the COnference on Data SYstems Languages (CODASYL). Committee members were selected from the manufacturer and large user communities. Since that time CODASYL has expanded its activities to include the definition of a Data Description Language independent of but common to other high level programming languages, and the modification of COBOL to include data

description statements to interface with this common language. In addition data manipulation statements to access the database described in the common data description language are to be added to COBOL. The CODASYL April 1971 Report is the result of this committee's work.

2.3 OBJECTIVES OF CODASYL re DATABASE SYSTEMS

The specifications set out by the CODASYL committee are intentionally not beyond the range of existing technology. It is their intent that this specification provide the basis for actual implementations of systems using the CODASYL syntax. In the introduction to the April 1971 Report the following objectives for a database management system are specified:

1. Allow data to be structured in the manner most suitable to each application, regardless of the fact that some or all of that data may be used by other applications -- such flexibility to be achieved without requiring data redundancy.
2. Allow more than one run-unit to concurrently retrieve or update the data in the database.
3. Provide and permit the use of a variety of search strategies against an entire database or portions of a database.
4. Provide protection of the database against unauthorized access of data and from untoward interaction of programs.
5. Provide a centralized capability to control the physical placement of data.
6. Provide device independence for programs.
7. Allow the declaration of a variety of data structures ranging from those in which no connection exists between data items to network structures.
8. Allow the user to interact with the data while being relieved of the mechanics of maintaining the structural associations which have been declared.
9. Allow programs to be as independent of the data as current techniques will permit.
10. Provide for separate descriptions of the data in the database and of the data known to a program.

11. Provide for a description of the database which is independent of the language of the processing run-unit.
12. Provide an architecture which permits the description of the database, and the database itself to be interfaced by multiple processing languages.

2.4 DATABASE TERMINOLOGY

Using the terminology of the CODASYL report, a DATABASE can be viewed as a hierarchy of DATA ITEMS and DATA AGGREGATES which are combined to form RECORDS. Records may be involved in SET relationships as OWNER and MEMBER. Records are stored in AREAs allocated within the address space of the database.

A DATA ITEM is the smallest unit of named data within the database.

A DATA AGGREGATE is a named collection of data items within a record. Two types of data aggregates are recognized: vectors and repeating groups. A vector is a one-dimensional, ordered collection of data items, all of which have identical characteristics. A repeating group is a collection of data that occurs an arbitrary number of times within a record occurrence. A collection may consist of data-items, vectors, and repeating groups. A RECORD is a named collection of zero, one or more data-items or data aggregates. It is not necessary for the contents of a database record to be stored physically contiguously. In fact, the data items comprising a record may not all be physically present in the database. They may be calculated when referenced.

Records are addressed by DATABASE KEY. The database key value is a conceptual entity whose value uniquely identifies a database record. The database key is not part of the database record. The database key value remains with the record until the record is removed from the database. At this time the database key may be allocated to a new record. Database keys are allocated for an area. Each area contains a range of database keys.

An AREA is a named sub-division of addressable storage space in the database and may contain occurrences of records and sets, or parts of sets, of various types.

A SET is a named collection of record types. Each set type specified in the SCHEMA (database definition) must have one record type declared as its OWNER and one or more record types declared as its MEMBER records. Each occurrence of a set must contain one occurrence of its owner record and may contain an arbitrary number of occurrences of each of its member record types. The owner record provides an entry point into the set occurrence and access to the member record occurrences.

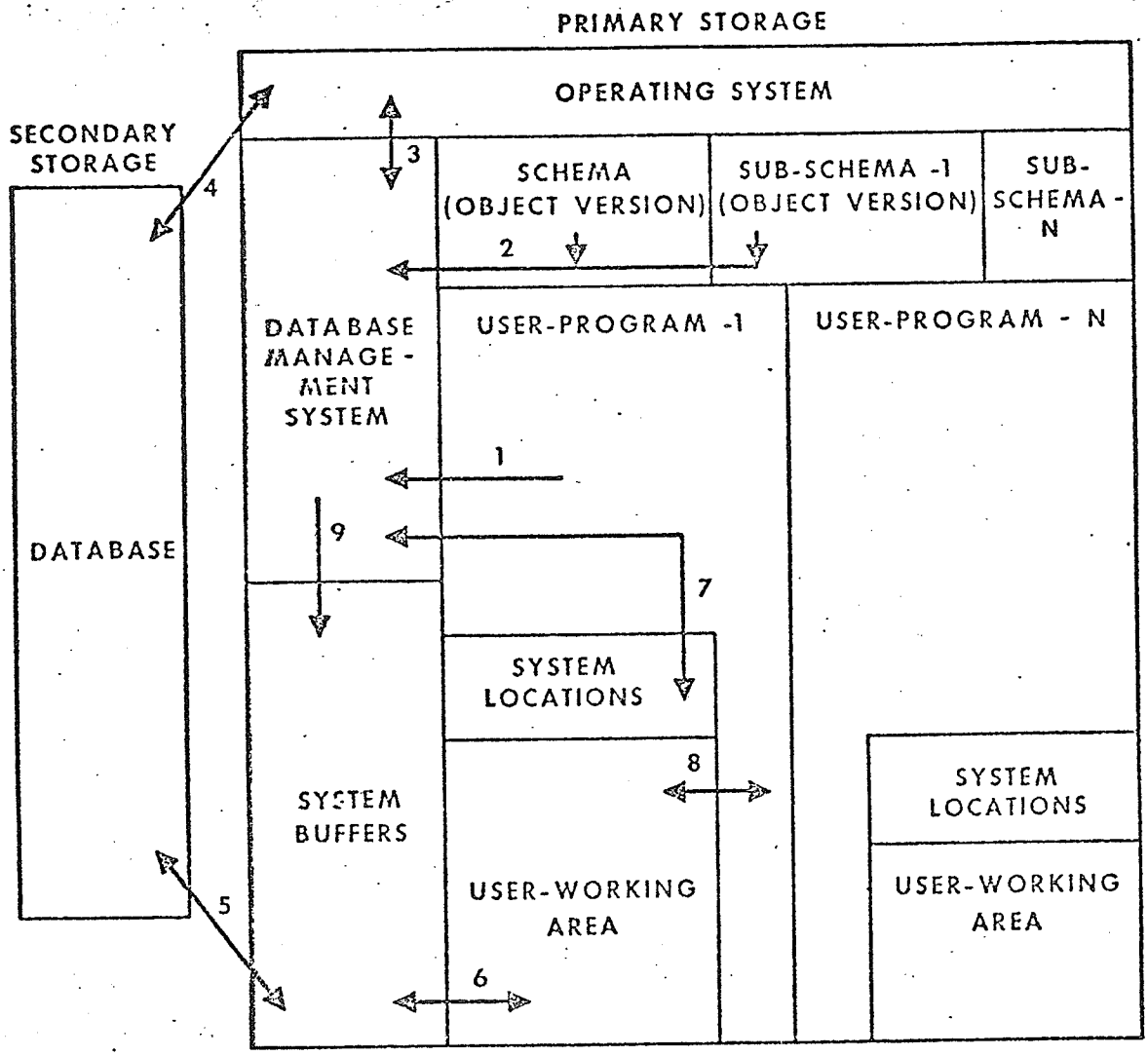
Data Description Languages (DDL's) are used to define the physical description of the database (schema) and the logical description of the database as it is known to a user (sub-schema).

Database Manipulation Languages (DML's) use the logical database relationships to access the database. These DML's may be included in host languages, or in non-procedural self-contained languages. In a host language the DML commands may be issued either as CALL's to the GDBMS or the language may be enhanced to include the DML commands. The CODASYL report includes a specification for the enhancement of COBOL to include the DML commmands. Most of the major DBMS in use today are host language systems. This allows users to convert the file system of existing applications without having to make significant program changes. Special purpose languages are, and will be, designed which allow the specification of requests in a form suitable to a particular user. This is particularly useful when the user is not a trained programmer. The self-contained language interprets the user request and generates appropriate DML commands.

The relationship between DDL and DML is a relationship between declarations and procedure. The declarations impose a discipline over the executable code and are to a large extent substitutes for procedures written in the DML and a host language; that is, they are implicit procedures which may be invoked by the execution of DML commands.

The GENERALIZED DATABASE MANAGEMENT SYSTEM (GDBMS) conceptualized on the following page taken from the CODASYL report illustrates the relationship between DL, DML and DDBMS.

CONCEPTUAL DATA BASE MANAGEMENT SYSTEM



The interaction, as shown, is always from user to DBMS. The DBMS uses the schema and sub-schema to formulate a request to the file system of the Operating System (OS). The OS issues the physical I/O instructions, to the database, which cause data to be transferred to the buffer pool of the DBMS. The information required by the user is copied from the buffer pool to a user work area (UWA). Certain status information relevant to a particular user is stored in a system area in the user region. The user may then manipulate data in his work area using the facilities of his host language.

The GDBMS provides a number of utilities to support the database. There are utilities to:

a) dump, edit and print the database,
b) load the database,
c) precondition the database, (eg. preformat direct files)
d) garbage collect,
e) collect statistics and analysis performance
f) allow the schema to be changed and effect automatic modification of the data base
g) allow areas to be allocated in selected location on selected devices.

The GDBMS provides for database recovery including logging, checkpoint and rollback of the database to a pre-error state.

The responsibility of interfacing between users of the database, the GDBMS, and owners of the data in the database belongs to the Data Base Administrator (DBA). This is a human function and involves the following specific functions:

1. Describing data structures which model the business or problem. This is accomplished through DDL statements which define the schema and sub-schema.
2. Controlling the assignment of names to data items, data aggregates, records, areas, and sets to ensure uniqueness. He may use synonyms to protect uniqueness of names already assigned.
3. Selecting search strategies based on user requirements.
4. Selecting and structuring a proper subset of the database required by an application programmer.

5. Assigning privacy locks and issuing privacy keys at various levels, to users of the database, based on the need to utilize those portions of the database affected by privacy locks.
6. Where applicable altering privacy locks and issuing privacy keys to application programmers.
7. Assigning areas to devices and addresses based on time/space requirements.
8. Loading the database with the aid of DDL, DML and specially written routines. He will usually be responsible to write certain of the sub-schema.
9. Monitoring the database continually for:
   (a) usage
   (b) response
   (c) privacy breaches
   (d) potential reorganization.

   Information to assist in this function is provided by the GDBMS log file.
10. Reorganizing the database by:
   (a) reassigning areas to different devices/media
   (b) changing schema and/or attributes of the schema
   (c) changing the database to reflect changes in the schema
   (d) some form of garbage collection.

A run-unit is a task of the Operating System. The GDBMS considers each run-unit to be a unique user of the database and provides a User Work Area (UWA), a set of communication locations, and a set of currency indicators. A currency indicator is the database key of the last record occurrence accessed by the run-unit of:

(1) each record-type known to the run-unit - current of record-name
(2) each set-type known to the run-unit - current of set-name
(3) each area name known to the run-unit - current area-name
(4) the last record processed by the run-unit - current of run-unit.

A database key is a unique and permanent value assigned to each record occurrence on creation. The database key may be used for direct

retrieval of records from the database.

2.5 BIBLIOGRAPHY

CODASYL DATA BASE TASK GROUP REPORT of APRIL 1971
pub Association for Computing Machinery
Single Copy Department
1133 Avenue of the Americas
New York, N.Y. 10036

FEATURE ANALYSIS OF GENERALIZED DATABASE MANAGEMENT SYSTEMS
A CODASYL Systems Committee Technical Report May 1971
pub Association for Computing Machinery

CODASYL COBOL Data Base Facility Proposal
A CODASYL Data Base Language Task Group Report March 1973.
pub The Technical Services Branch
Department of Supply and Services
5th Floor, 8 Metcalfe Street
Ottawa, Ontario, Canada
K1A0S5

3.1 INTRODUCTION TO DATABASE DATA STRUCTURES

In this chapter we will look at data structures which can be used in a database to satisfy the requirements of the aforementioned design criteria for databases.

A Generalized Database Management System (DBMS) provides a variety of data structures from which the data base administrator (DBA) chooses those suitable to each application. The DBA must weigh the costs of each data structure against the performance criteria established for the application. There are four functions for which costs must be determined:

1. the cost to add a new record,
2. the cost to update a record,
3. the cost to retrieve a record, and
4. the cost to delete a record.

Costs arise in four areas:

1. the amount of processor storage used to perform a function,
2. the amount of external device storage required to support the data structure,
3. the number of I/O operations required to perform a function, and

the amount of processing time required to perform the function.

The DBA requires certain information about the environment in which the database application run-units are to operate. Specifically he must determine:

1. the expected number of record additions, retrievals, updates and deletions,
2. the processing modes (ie are these functions batched or run on line),
3. the required response times for specified functions, and
4. the distribution of functions over time. (ie Are all database modifications batched or they submitted throughout the total time the system is available?)

The tradeoffs between the use of system resources and response time often make the selection of suitable data structures an important and difficult task.

3.2 SEQUENTIAL DATA STRUCTURES

Records and data items in sequential data structures are related by physical adjacency. The key sequence of the data and the physical sequence of the data are the same. The address of the next record is calculated as the address of this record plus the length of this record.

Records are typically accessed in sequence by a key field defined within each record. Sequential organization is used for tape, printer, and card devices, and may be used on direct-access devices.

3.2.1 PURE SEQUENTIAL

Records are stored in physical sequence of a key within each record. Storage is required for the record data and a delete flag. Records must be read sequentially from the beginning of the data base in order to find a particular record. A record to be updated must be retrieved and then rewritten at the same address after internal modification. Records may not physically be changed in length. Records to be inserted must be presented in key sequence. Additions take place after the last record in the data structure. If sequence is maintained, a deleted record of the same size may be replaced by a new record. The data management system will usually allow additions to take place at end of the structure without reading all preceeding records. Records cannot be physically deleted from the file but are flagged with a delete flag and recognized by the run-unit as deleted. The diagram below illustrates the organization of a pure sequential data structure.

Record No.	Relative Byte Addr	Rec Key	Data Items
1	499	8321	D---D
2	513	8419	D---D
3	549	8733	D---D
4	680	8999	D---D
5	723	9001	D---D
6	800	9101	D---D
7	900	9111	D---D
8	1004	9233	D---D

3.2.2 SEQUENTIAL WITH POINTERS

A slight modification of the pure sequential data structure allows records to be inserted logically at any position in the file. A pointer field (usually 4 bytes) is added to each record to point to the next record when logical and physical sequence do not correspond. New records are stored in an overflow area in entry sequence while records in the prime area are maintained in key sequence.

Record No.	Relative Byte Addr	Rec Key	Data Items	Pointer
1	499	8321	D---D	0
2	517	8419	D---D	0
3	557	8733	D---D	0
4	696	8999	D---D	18345
6	743	9101	D---D	0
7	824	9205	D---D	0
8	928	9111	D---D	18392

Overflow Area

Record No.	Relative Byte Addr	Rec Key	Data Items	Pointer
1	18345	9001	D---D	743
2	18392	9233	D---D	824

Records must be read sequentially from the beginning of the data base and chains to the overflow area must be followed to retrieve successive records. Records are updated exactly as in the pure sequential form. Records can be inserted anywhere in the logical sequence of the database. When inserting a new record between two existing records an overflow area is obtained to store the new record. The address of this record is stored in the logically preceeding record. The address of the logically succeeding record is stored in the new record. Record deletion is typically done exactly as in the pure sequential database. It is possible to improve performance by physically deleting records in overflow and modifying pointers to this record. Record updating is done exactly as in the pure sequential data structure.

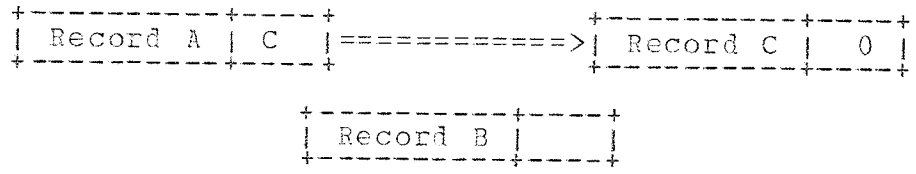
A further extension to this technique stores all record in the overflow area. This technique is known as Embedded Link.

3.3 EMBEDDED LINK STRUCTURES

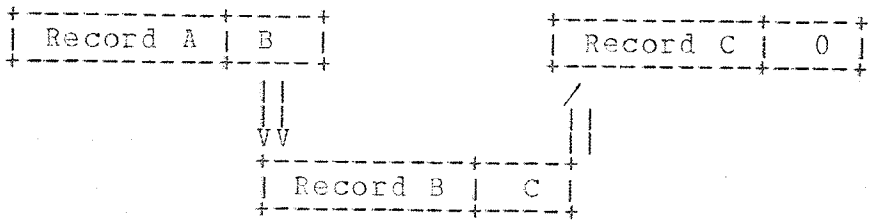
Chaining is a technique for linking logically related records. A CHAIN is a group of records so linked. A pointer is a special reference field incorporated into the physical record. A pointer contains a reference to the next record in logical sequence. Records are linked to their successors by pointers. If it is convenient to access predecessors, two pointers may be included -- one back, one forward. Some method must be found of identifying the start and end of the chain. This is often done by including dummy records whose addresses are known to the access modules. The last record in the chain may contain a pointer to the first record in the chain to form a RING structure.

Chained records need not be related by physical adjacency since the address of the next record is known from a pointer. Addition and deletion of records may take place place throughout the data structure and is accomplished through pointer modification.

Addition of a new record into a one-way forward linked data structure is illustrated below.

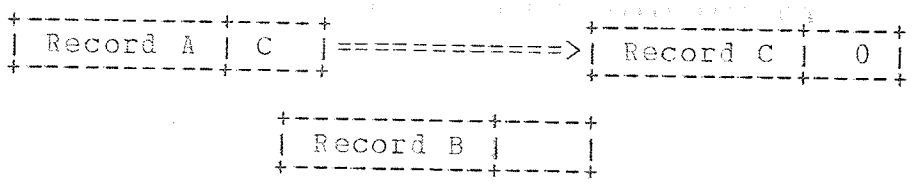


To add record B which logically follows A and preceeds C, set SUC(B) = SUC(A) and SUC(A) = B.



Deletion of record B from the same data structure is illustrated below.

Set SUC(A) = SUC(B).



These examples illustrate the relative ease with which records can be added and deleted from a linked data structure once the required record has been located. If a predecessor pointer is included in each record, then the addition and deletion operations must be modified to update the second set of pointers.

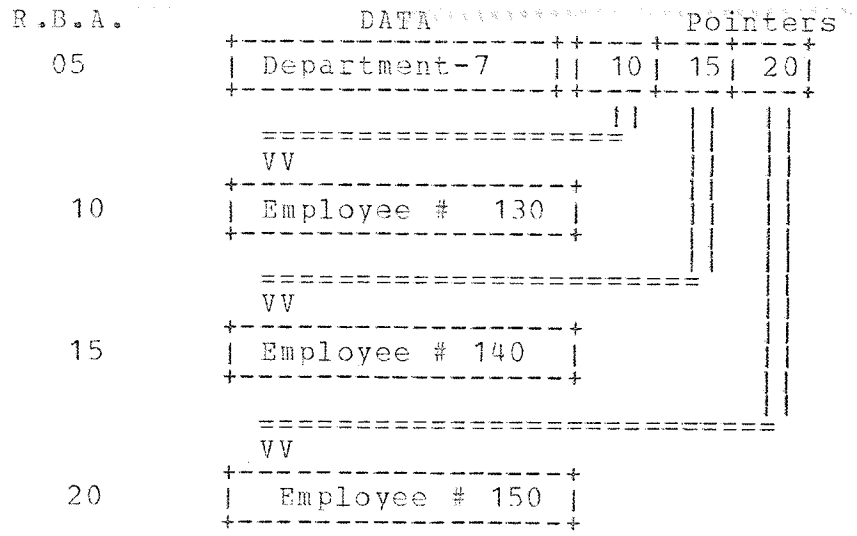
Record retrieval requires searching the chain from a starting point and looking at each record successively until the required record is found.

Record updating is a retrieval, in core update, and rewrite sequence when the record key is not modified. Typical implementations require that the records be fixed in length. When the key is modified the operation is a delete, insert sequence.

In applications requiring secondary keys several chains can be built and maintained through the database. Inserts, updates and delete operations require that each chain be maintained.

For each chain in which a record participates as owner or member a 4 byte pointer field is maintained. A data structure which contains more than a single chain is a THREADED or MULTI-LIST structure. When the chains throughout a database get very long it may be convenient to use a controlled list length multi-list. In this data structure no list may exceed a given length. Instead a list is constructed which gives several entry points into the chain. This reduces chain searching and thus the number of I/O operations to retrieve records.

The list lengths can vary from one to the number of records in the chain.



When the list length is the number of records in the chain, the structure is an INVERTED LIST. A further method of partitioning a database is known as CELLULAR PARTITIONING. In this data structure the database is partitioned into cells. A list is maintained of entry points into each cell. As the cell size decreases the database becomes inverted until for a cell size of one record the structure is an inverted list. Large cell sizes tend toward either a multi-list if the cell is internally multi-listed, or a sequential structure if the database is not internally list structured. If the cell is internally inverted by an Inverted List then, regardless of cell size, the entire file is inverted but in a multistage hierarchy.

The primary storage requirements of embedded link systems is small. The address of the next record is always available from the record in primary storage. The algorithm for processing an embedded link system is very simple and thus the amount of coding to support the data structure is minimal. In order to improve performance many buffers may be used to store a larger number of records in primary storage at once. In a multi-user environment the probability of finding a record in primary storage will be increased when the number of records in primary storage is increased. Blocking of the physical records and attempting to store logically sequential records

physically sequentially are other ways of improving performance.

3.4 INVERTED DATA STRUCTURE

When an application requires multi-key access to a database a common technique involves the maintaining of inversion lists which give key values and record pointers for all fields defined as key. A fully inverted or 'involute' database supports access based on every field defined in the record. In this data structure the data exists as the concatenation of the inversion lists. In practice only a subset of the record fields are defined as key, and a key value is stored as part of its associated record occurrence. Since the inversion lists are dense, there is no need for records to be stored in blocks any sequence. Typically records are stored in (equivalent to VSAM control intervals, see Section 3.7 for a description of VSAM). Blocks are allocated as required from free space.

Multi-key retrieval is facilitated by the use of list searches to find the data records satisfying the boolean search argument. If lists are searched from smallest to largest the number of comparisons may be minimized at each step. When record existence is the answer to the query, no data record need be retrieved, since existence information is available from the lists.

The GDBMS's implemented using the inverted list technique are all proprietary. The developers of these packages refuse to make available detailed technical specifications about the algorithms used to build and maintain their inverted files. The implementation suggested here is a possible technique.

As each physical record is loaded it is assigned as internal storage number (ISN) which remains with the record until it is deleted. An ISN-BLOCK table is built which associates the ISN with the physical storage address of the record. Records are stored in fixed length blocks. Each block may then contain as many variable length records as will fit. At load time free space is left in each block to allow for record expansion and insertion. All fields are stored in a compressed form. High order zeroes and right blanks are removed from fields and null fields are not stored. A null field is a field defined in the physical record but not present in the logical record. If a record grows and will no longer fit in the block it is removed and stored in another block. The original block recaptures the space as

free space. The ISN-BLOCK table is updated to reflect the new block number of the ISN. There is only one entry in the table for each ISN. ISN's are allocated sequentially and reused when a record is deleted. The relative block number is the only entry in the ISN-BLOCK table and it is 3 or 4 bytes in length depending on the number of blocks which can be stored in the database.

ISN-BLOCK TABLE

11
72
3
70
14
19
12
15

If a record field is defined as key an inversion list is built to cross reference field value occurrences to the ISN of the record in which the value appears. The primary list contains the key value, the ISN of the first record containing the key value, a pointer to the duplicate table, and optionally a count of the number of records containing this value.

PRIMARY LIST

KEY VALUE	ISN	DUP PTR	#ISNS
ABC	18	2	20
ABD	47	0	1
ADDD	19	1	2
AZ	1	0	1
AZZ	9	3	2

The primary list is maintained in key sequence. The duplicate list is a chain of ISN's of other records containing the key value of the primary list entry point. It contains the ISN of a record with a key value in the Primary List and a pointer to the next duplicate.

DUPLICATES TABLE

ISN	NEXT
24	0
4	7
8	0

The third table maintained for each key field cross references

the ISN's and primary list key values. It is maintained in ISN sequence to facilitate insertion and retrieval. Each record containing a key value occurrence has its ISN and a pointer to its entry in the Primary List stored.

ISN-KEY XREF TABLE

ISN	KEY PTR
4	3
8	4
9	4
18	1
19	2
28	2
47	1

Retrieval of all records with a particular key value involves searching the PRIMARY LIST to find the occurrence of the key value. If all that is required is the number of records with this key value, then the information is available as the #ISNS field. The ISN's of all records with this key value are available as the primary list ISN and the ISN occurrences on the duplicate chain pointed to by the DUP-PTR field in the PRIMARY LIST.

Accessing records based on a multi-key search argument is accomplished in the following manner:

1. If possible select the key whose associated primary list is the shortest.
2. Build a temporary HIT-TABLE of the ISN's satisfying this key value.
3. Select from the remaining unprocessed keys that key with the smallest ISN-KEY XREF table.
4. Match the HIT-TABLE to this list and delete from the HIT-TABLE all ISN's not found in the ISN-KEY XREF table.
5. Continue (3) and (4) until all keys have been matched.
6. The resulting HIT-TABLE contains the ISN's of the records satisfying the search criteria. If the request is for existence of records satisfying the search argument or for the number of records then this information is now available. If the records must be retrieved then the ISN's are looked up in the ISN-BLOCK table to find the address of

the desired records.

Note that the algorithm will work regardless of the order of key selection. If an AND/OR boolean search criteria is used, then the keys must be presented in an order which satisfies the boolean algebra. Selecting the short lists is purely an efficiency consideration.

The PRIMARY LIST is maintained in key sequence. Sequential processing may thus take place with little difficulty. The ISN's of successive records are found by sequentially processing the PRIMARY LIST and following DUP-PTR when necessary.

When a record is updated and no key fields are modified only the ISN-BLOCK table may need to be modified. If the record is lengthened and so moves to a new block then the ISN-BLOCK entry must be updated to reflect this change. When a record update modifies key fields then the other tables are affected. All changed key values must be entered in their respective PRIMARY LISTS. The entry for the previous key must be purged. The ISN-KEY XREF table must be updated.

When a record is deleted the ISN of the record must be removed from the PRIMARY LISTS of each key field in the record. The ISN-KEY XREF table entry must be deleted for each key field in the record. The ISN-BLOCK entry is flagged.

The PRIMARY LIST can be searched using a binary search or a hashing algorithm. The ISN-BLOCK table is directly indexable by ISN and the ISN-KEY XREF table can be searched using a binary search. If the table is created using a hash function then the hash function may be an effective retrieval method.

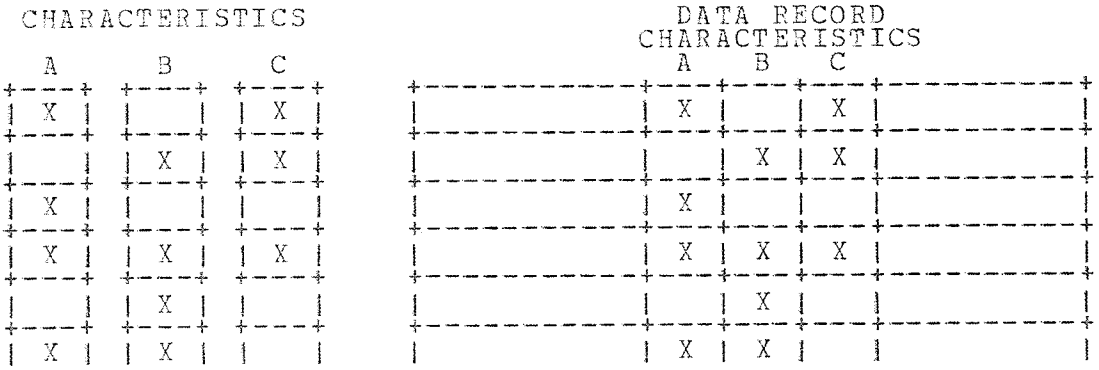
It is possible to reduce the amount of table space required to implement an inverted file system but this will cause a decline in the efficiency of updating and sequential processing. There are applications which batch update requests and seldom process a database in a sequential manner. Police Information Systems are examples of this type of application.

Considerable storage is required for the inversion lists and related tables. If this information is maintained in primary storage the costs may be excessive, particularly when multi-key retrieval operations require the storage of information for several keys. Techniques of key compression and bit coding of ISN-BLOCK

relationships can be used to reduce the storage requirements at the expense of extra processing time.

3.5 BIT VECTORS

When data fields can be coded with a yes/no indicator a bit vector can be built to describe the field value of these fields. This facilitates retrieval of all records satisfying a particular criteria. When multiple fields are so coded then a bit mask can be built and compared across bit vectors to select records satisfying specified criteria.



These bit vectors are often combined intc a bit profile which is appended to each record in a sequential file. The file is then processed sequentially and desired records retrieved. The bit profile reduces comparison time.

In a random access envircnment the bit profile may be maintained as a separate index with the address of each record which satisfies a particular bit mask.

Bit coding can be used effectively when the fields on which enquiries will be made are known in advance, and when the field values can be represented as a single bit. An extension of this technique supports the use of bit strings for data value coding. To be an effective technique, the algorithms tc ccnvert data to its bit representation should be simple. When sufficient core exists to store the bit vectors or bit profiles in core very rapid retrieval on multi-key search arguments is possible. As in the inverted data structure only records satisfying the conditions need be retrieved.

3.6 DIRECT THROUGH KEY TRANSFORMATION

The records within a database may be organized in any manner suitable to the user. Key sequence and physical sequence are typically unrelated. A record location is determined by applying a function to the key value which then returns a record location indicator.

3.6.1 PURE DIRECT

A pure direct data structure can be viewed as a pure sequential data structure in which records are directly addressable.

When a key value assigned to a record is equal to its address on secondary storage the record is directly addressable when its key is known. Storage is required for the record data and a dummy record indicator. The key field need not be stored with the record. This data structure requires that the implementor control the allocation of keys and that every possible key be assigned to a valid or dummy record occurrence; for example, a company allocates keys in the range 0-9999. This allows 10,000 records to be stored and 10,000 locations on external storage are reserved for this database.

3.6.2 HASHED DIRECT

Records are located directly at an address calculated by a key transformation algorithm which maps the supplied key value to a record address. The key field of the record retrieved is compared to determine whether the desired record has been retrieved. Unfortunately it is possible that more than one key value will be transformed to a single record address (see ACM vol 16, #10 for analysis of transformation functions). Subsequent records are called synonyms and must somehow be related to the first record at a particular record address.

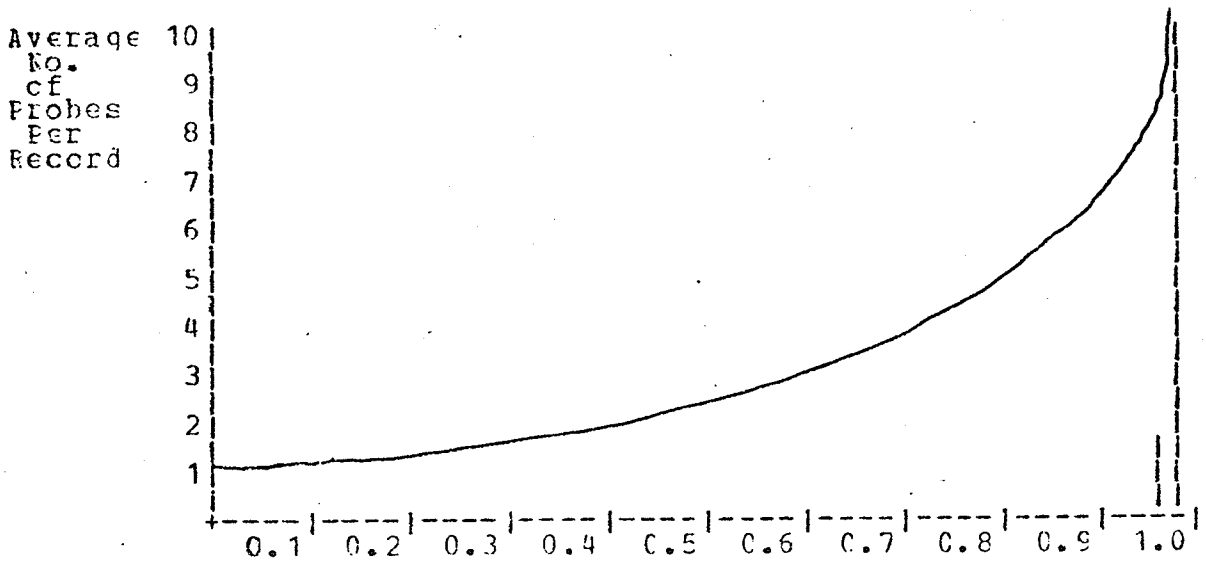
The first measure of effectiveness of a key transformation function is the number of synonyms which occur at varying packing densities (ie. degree of fullness of the data space).

The second and usually more important consideration when evaluating this technique is the handling of synonyms. Space must be found to hold the new record and some method must be included to

relate syncnyms.

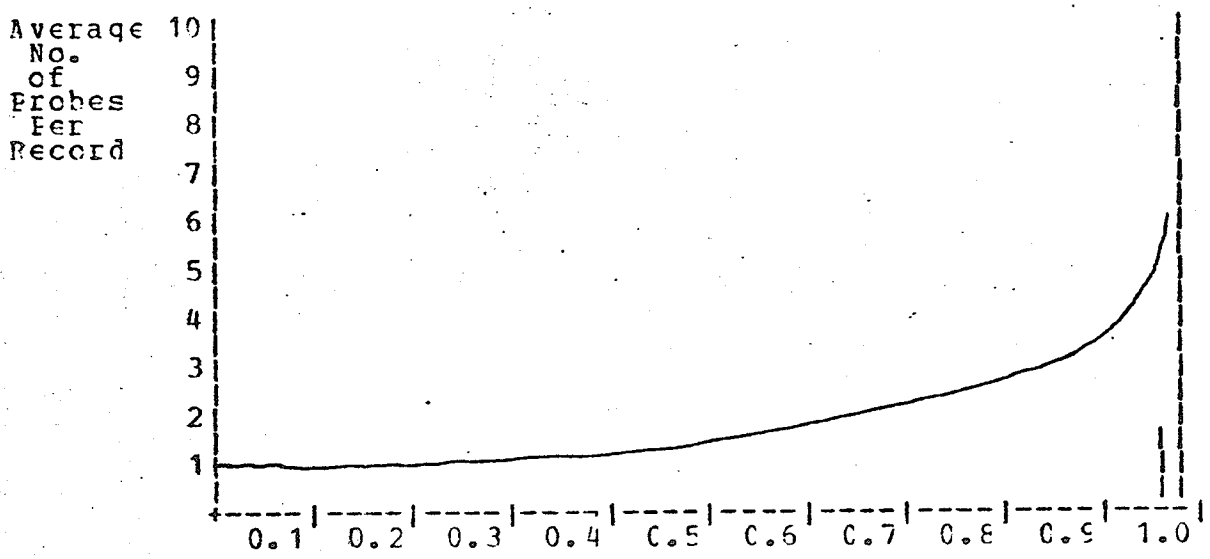
Several methods have been used tc handle this situation. The simplest and usually worst method is tc search fcr the next available open slot in which the new reccrd will fit. A syncnym chain may be ccnstructed tc redtce retrieval time. This methcd decrades very rapidly (see diagram) cnce the file packing exceeds 50%. The expected number cf probes tc retrieve a particular reccrd increases expcnentially until at 95% it exceeds 10 and cver 90% approaches infinity.

DIRECT FILE MAINTENANCE WITHOUT RCINTERS
Average Number cf Prcbes tc Retrieve a Reccrd



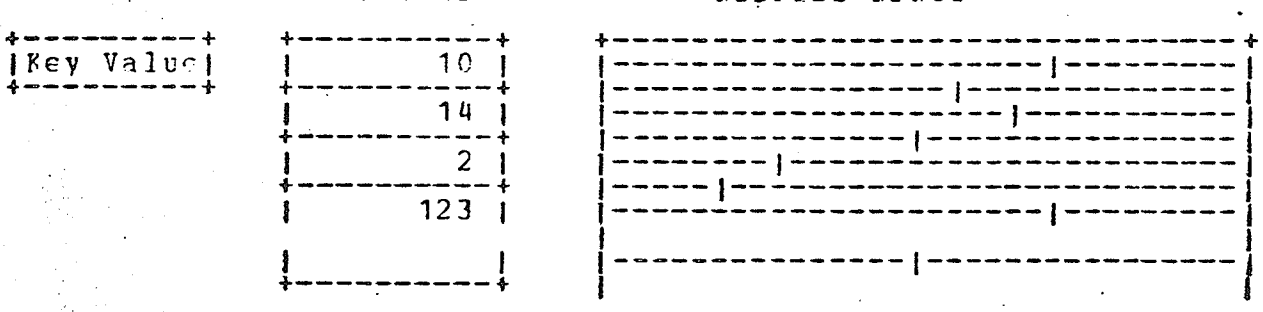
A seccnd technique the VYSSCTZKY METHCD attempts tc improve space utilizatien by applying a second transfcrmatien tc the key and using the new address fcr the syncnym. Stbsequent syncnyms are chained and space is allccated in some way. This methcd appears tc wcrk well for packing up tc 90%, with the expected number cf prcbes abcut 2.5. An extension tc this method which might alsc work is tc allocate 2 or more reccrd types into the same dataspace. If the keys of the subsequent reccrd types fcllow a different distributicn then the primary key, this methcd should tend tc use the tctal space more completely.

VYSSOTZKY METHOD FOR MAINTAINING DIRECT FILES
Average Number of Probes to Retrieve a Record

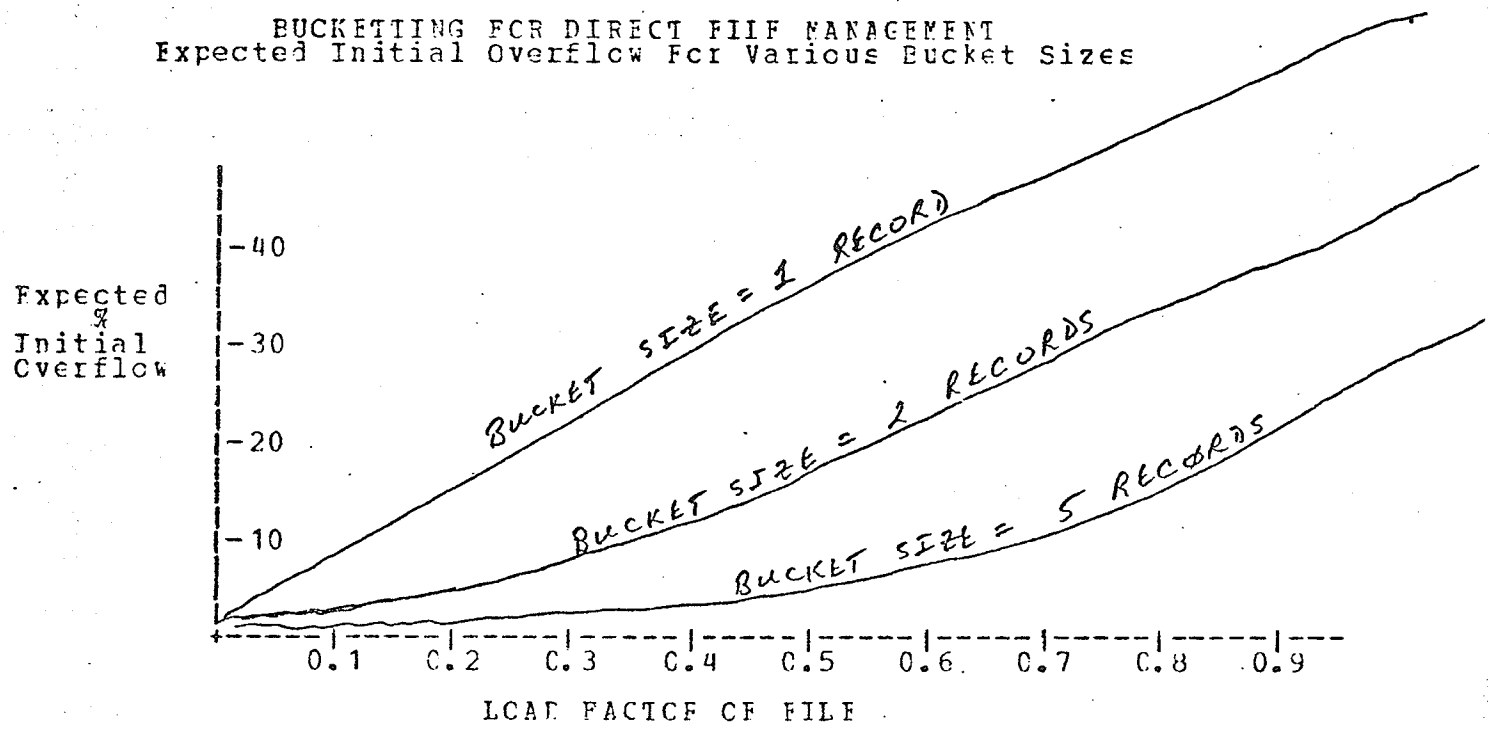


In both these methods available space must be managed in some way so that synonyms can be allocated to unused space efficiently.

A third technique known as BUCKETTING simplifies space management by hashing to a BUCKET which then points to a block containing 'n' records. Free blocks are maintained by an available block chain and allocated as needed. Blocks need not be contiguous and are immediately reusable if deleted. The number of I/O operations required to retrieve a record decreases as the number of records per block increases. However, the amount of unused space also increases, since only records hashing to address 'B' may be stored in BLOCK(BUCKET(B)). The bucket size is picked as a prime number such that the product of the bucket size and the number of records 'n' per block slightly exceeds the expected number of records in the database.



BUCKETTING FOR DIRECT FILE MANAGEMENT
Expected Initial Overflow For Various Bucket Sizes



Records are retrieved from the address determined by the key transformation algorithm on the input key. If the record is a synonym it may be necessary to retrieve several records to find the correct record. Updating involves the retrieval, internal modification and rewritting of the record at the pre-determined address. Record length may not be increased except in the bucket technique. Records are deleted by updating the dummy record flag. In the bucket technique space can be recovered within a block when a record is deleted from the block; a block is recovered when every record in the block is deleted. Records are added by retrieving the record from the

determined address. If the retrieved record is a dummy the operation is an update. If not then a synonym operation must be invoked.

Logical sequential processing of the data structure is not possible since records are not maintained in any logical sequence.

Minimal overhead for the storage of control information is inherent in this method. The address of a desired record is available from the key transformation algorithm. Primary storage is required for the algorithm, and records presently being processed in primary storage. In the bucketting technique further storage is required to store the bucket and blocks containing records presently being processed in primary storage. In a multi-user GDBMS it may be desirable to increase the number of buffers available to store records in primary storage. In this manner the probability of a request for a record being satisfied from primary storage increases. The use of several buffers allows the scheduling of asynchronous I/O operations from several devices, which will improve response in a multi-user environment with distributed requests.

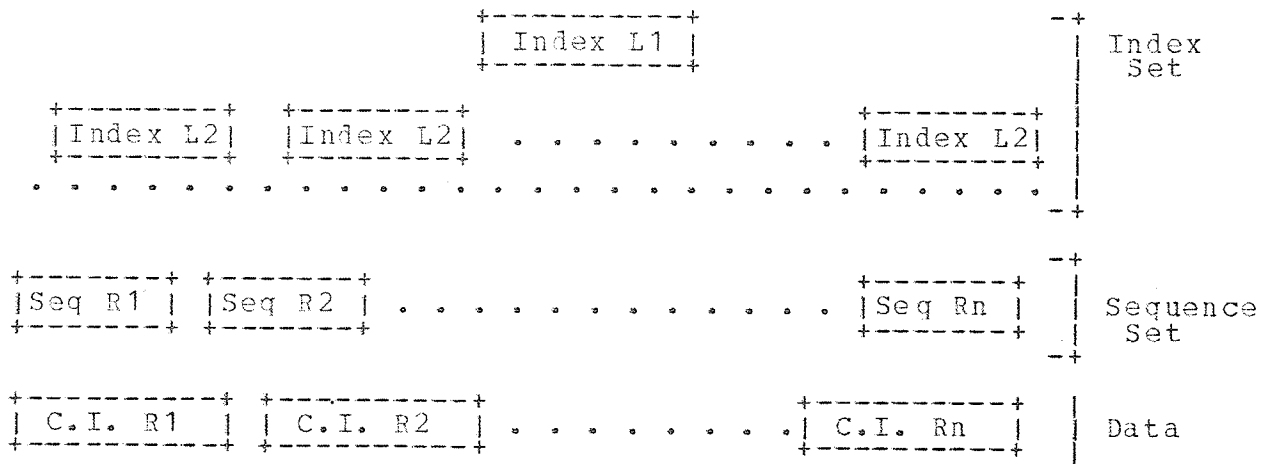
3.7 DIRECTORY-BASED-SYSTEMS

Directory-based systems are designed to provide random access to uniquely keyed data records without the necessity of sequentially processing a large part of the file. Additionally, support is provided for sequential processing of the data. The data record keys are organized into a hierarchy of indices much like the indexing structure used in library card catalogues. Two types of indexing are used. The difference occurs in the lowest level of index. A "dense" index contains an entry for each data record whereas a "nondense" index contains one entry for each physical storage unit (block, control interval).

A dense index contains, in the sequence set, an entry for every data record in the database. Using the dense technique, it is not necessary for records within a physical block to be in key sequence. Each logical record is located from the sequence set as a relative record number, in a physical block, stored at a relative byte address (RBA). A dense index allows records to be randomly or sequentially retrieved from a database on more than a single key. The advantage of a nondense index, when records are retrieved on a single key, is the reduction in index storage and faster retrieval of sequentially processed records from a physical block. Records within a block must be maintained in key sequence while blocks need not be physically sequenced.

VSAM (Virtual Storage Access Method) is an example of a nondense indexing structure. VSAM can also be implemented as a dense indexing scheme. In an indexed access method, levels of indices are built and maintained which contain the highest key in, and a pointer to, each block in the next lower level. Sufficient levels are built such that the highest level is a single block.

VSAM Index Structure



In VSAM the sequence set contains the key of, and a pointer to, each physical block (control interval). The data records within a block are maintained physically in key sequence. Logical sequence of the dataset is maintained by successive sequence set pointers.

There are two basic insertion strategies which can be used in an indexed structure: in-place and out-of-place. In-place insertion is used when the sequence set is nondense and out-of-place when the sequence set is dense.

Out-of-place insertion involves inserting new records into an overflow area and pointing the index to the new record. Space for overflow may be allocated within or external to the prime data area. A "variation" of this method is used in the Index Sequential Access Method (ISAM).

In-place insertion is somewhat more complicated; since the number of insertions and their location is not predetermined, space cannot be reserved in advance within the prime data area. VSAM uses the in-place insertion technique in both its dense and nondense implementations. A physical block (in VSAM a control interval) is fixed in length for all entries in a dataset. Within the control interval records may be variable in length. Record lengths are maintained in control information stored within each control interval. reserved within a control interval to facilitate the insert and update operations.

The unit of allocation in VSAM is the control area. When more

space is required in a VSAM dataset, a control area is obtained which is formatted into control intervals.

The algorithm to insert a record into a VSAM dataset is most easily presented as a sequence of decisions and steps adapted from the IBM SYSTEMS JOURNAL Vol 12, #4:

1. Search the index and locate the control interval into which the data record is to be inserted.
2. If there is space within the control interval, merge the record to be inserted in sequence within the control interval. No update of the nondense index is required. The dense index must be updated for the new record and for each record whose location within the control interval changes.
3. If there is insufficient space within the control interval, check to see if there is a spare control interval in this control area.
4. If there is a spare control interval then:
   (a) Map the first half of the records from the target control interval into the spare control interval along with the new record if it fits with this sequence, and write the new control interval.
   (b) Update the sequence set record for the control area by inserting an entry from the activated control interval and write the updated sequence set record. Update the sequence set entries for each dense index. The sequence set for each record moved to the new control interval requires upating.
   (c) Map the second half of the records of the target control interval into the first half of the target control interval. Merge the record to be inserted if it fits into this sequence. Write the updated target control interval.
5. If there is no spare control interval then:
   (a) Allocate a new control area and a new sequence set record.
   (b) Map the first half of the target control area intervals into the new area and build the sequence set record for

the new area. Write the new control area and sequence set record.

(c) Insert an entry into the high-level index for the new sequence set record.

(d) Update the target sequence set record by making the first half of the control intervals free control intervals, and write the record.

(e) The insertion can now be done via Rule 4.

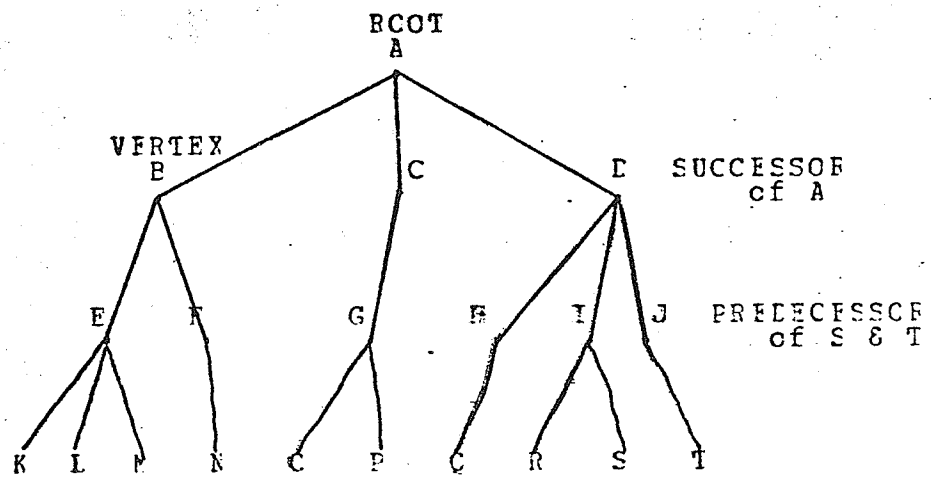
An update operation which does not increase the size of a data record or change the data record key is a simple retrieve, update in core and rewrite. If an update changes the record length or changes the record key then the operation becomes analogous to a delete followed by an insert.

Data records are retrieved randomly by a top-to-bottom left to right search of the index entries. When the desired sequence set entry is retrieved a binary search of the entries in the selected sequence set control interval is used to find the Relative Byte Address (RBA) of the data record and data control interval. Sequential processing uses the sequence set from left to right to recover logically sequential control intervals.

Record deletion involves record retrieval, an in-core deletion and left shift of the control interval content and a rewrite of the control interval. Indices are never updated for a delete operation. When all the records in a control interval are deleted the control interval is added to the free control interval list to be used in case of a control interval split. Similarly when all records in a control area are deleted, the control area is made available for reuse.

3.8 TREE AND NETWORK STRUCTURES

The basic logical structures used in a database are the tree and network. The structures previously examined are used in the building of trees and network structures.

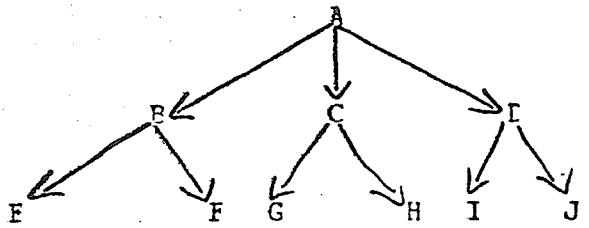


Tree Structure

A rooted tree is a graph consisting of a collection of edges and vertices with each edge containing exactly two vertices. The vertex at the upper end of an edge is the predecessor of the vertex and the vertex at the lower end of an edge is the successor of the vertex. In addition:

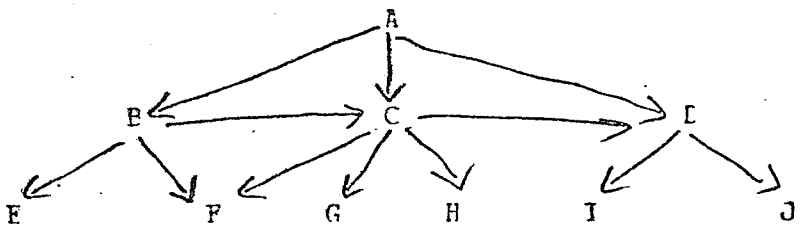
1. There is a unique vertex (root) which has no predecessors.
2. Every vertex other than the root has exactly one predecessor.
3. Every vertex other than the root is connected to the root by a (unique) path such that the path begins at the root and ends at the vertex, and such that each vertex on the path other than the root has exactly one edge entering it.

A directed graph (diagraph) is a set E of ordered pairs (edges) of elements from a set V called vertices of the graph.



Directed Graph

A network is a connected diagraph which contains no circuits. Unlike a tree, there are no restrictions on the number of edges entering a vertex. The vertices of the graph correspond to records in a database structure. A set is the subtree formed by a node and its leaves.



Network Structure

Tree structures in a database are directed. In the hierarchy, records in the structure are owners or owned. A record is owned by its immediate predecessor in the graph and owns its immediate successor. In a tree structure, each record except the root has a single owner but may own any number of records. If a record has more than a single owner, the graph is a network. If a record is owned by itself then the graph contains a circuit. In data structures of interest in a database circuits are not allowed.

CIRCUIT

3.9 DATA MANAGEMENT ACCESS METHODS

The operating systems under which GDBMS run provide several access methods to simplify the data management function. These access methods have evolved over time and are implemented in various ways by the computer manufacturing and user communities. In implementing a data structure within a database, data access is usually accomplished through an access method routine. Requests for records from the database are converted by the GDBMS file system into a sequence of access method requests, which in turn generate requests to the Input Output Supervisor (IOS) through which data transfer is handled. Data items from the physical data records are combined to create the logical data record requested by a user.

The Access Method Services routines exist with the operating system to provide the user with certain data management facilities at a non machine language level. Some of these facilities are:

1. READ/WRITE/REWRITE commands for the handling of logical or physical data records.
2. Blocking and deblocking of physical records to logical records.
3. Asynchronous execution of Input Output operations and processing.
4. Handling of end-of-volume, end-of-file and error special conditions.
5. Access to data by name only.
6. Deferred definition of data characteristics until execution time.
7. Allocation, deletion and renaming of data files.
8. Data integrity in case of accidental system or program failure.
9. Support for fixed, variable and undefined record types.
10. Access to data organized into sequential, indexed, or direct structures.

3.10 BIBLIOGRAPHY

LEFKOVITZ, D - FILE STRUCTURES FOR ONLINE SYSTEMS
pub Spartan Books, 1969 Cat # 68-20673
available in British Commonwealth from
Macmillan and Company, Ltd.
4 Little Essex Street
London, W. C. 2

KNUTH, D.E. - Sorting and Searching
Vol 3 of The Art of Computer Programming
pub Addison Wesley, 1973 Cat # 67-26020

LONDON, K.R. - Techniques for Direct Access
pub Auerbach, 1973 Cat # 72-89103

STONE, H.S. - Introduction to Computer Organization
and Data Structures
pub McGraw-Hill, 1972 Cat # 75-167560

WAGNER, R. E. - Indexing Design Considerations
IBM Systems Journal, Vol 12, #4 1973 pp 351-367.

SENKO, M.E., ALTMAN, E.B., ASTRAHAN, M.M., FEHDER, P.L.
IBM Systems Journal, Vol 12, #1 1973 pp 30-93.

LUM, V.Y., General Performance Analysis of Key-to-Address
Transformation Methods Using an Abstract File Concept
Communications of the ACM, Vol 12, #10.

4.1 INTRODUCTION

Prior to the advent of Generalized Data Base Management System's (GDBMS), the definition of data was included as an integral part of each program. Programs were bound to their data at compile time and a change to the data format caused at least a recompilation of each program referencing the data. Most GDBMS's provide for the definition of data to be input separately from the program operating on the data. The GDBMS builds a directory which is used with the program at execution time. In this way a change in the database can be reflected in the directory without a recompilation of any program not directly affected by the change. In most systems data definition facilities exist at two levels; the physical level, to describe the actual structure of the entire database; the logical level, to describe the logical view of the database known to a particular run-unit. In the CODASYL specification the SCHEMA defines data at the physical level and the SUB-SCHEMA defines data at the logical level. Data Description Languages (DDL'S) exist to define the schema and sub-schema. A DDL for the sub-schema is to be defined for each programming language supported by the GDBMS. A single DDL common to all users is defined for the schema definition.

4.2 Variations between the SCHEMA and the SUB-SCHEMA

A sub-schema must be a consistent and logical subset of the schema. The sub-schema definition can be used to create new record structures in the following ways:

1. At the data-item level-
   (a) Data Items may be declared to have different characteristics.
   (b) Descriptions of specific data items may be omitted.
   (c) The ordering of data items within data aggregates may be changed.
2. At the data aggregate level
   (a) Descriptions of specific data aggregates may be omitted.
   (b) The ordering of data aggregates may be changed.
   (c) Vectors may be redefined as multi-dimensional arrays.
   (d) Data items or data aggregates may be selected and given a group name.
   (e) Additional structure mappings may be provided by the facilities of a particular sub-schema DDL.
3. At the record level
   (a) Descriptions of specific record types may be omitted.
   (b) Record occurrences included in specific areas may be omitted, while other occurrences of that record type are included.
4. At the set level
   (a) Descriptions of specific set types may be omitted.
   (b) Different set selection criteria may be specified.
5. At the area level
   (a) Descriptions of specific areas may be omitted.
6. At all levels the sub-schema allows the definition of a synonym which can be used in place of a schema defined name.

4.3 Data Independence

The schema and sub-schema interface with each other such that:

1. An object version of the source code schema may be compiled independently of any user program or any sub-schema.
2. Object versions of a source code sub-schema may be compiled independently of any user program, and stored in a user or system library.
3. An arbitrary number of sub-schema may be declared on the basis of a given schema.
4. The declaration of a sub-schema has no effect on the declaration of any other sub-schema and sub-schema may overlap one another.
5. Each sub-schema must be named.
6. A user program invokes its sub-schema.
7. The same sub-schema may be invoked by an arbitrary number of programs.
8. Only the areas, records, data items, and sets included in the sub-schema invoked by a program may be referenced by that program.
9. Since sub-schema are host language oriented a program must invoke a sub-schema consistent with its source language.

CODASYL
4.4 DATA DEFINITION LANGUAGE FOR SCHEMA

4.4.1 Syntax Definition Terminology

The syntax of the CODASYL schema consists of four types of ENTRY which:

Identify the schema (Schema Entry).

Define areas (Area Entry).

Define records (Record Entry).

Define sets (Set Entry).

Only one Schema Entry is defined within each schema. One entry is defined for each area, record and set type. The entries must be ordered such that definition dependencies are resolved. The schema entry is first, followed by area, record and set entries. No record can be defined until its associated area entries have been declared. Similarly no set entry can appear until all of the record types which are owner and members in the set have appeared.

An entry consists of one or more clauses which describe its attributes. The Record Entry has a Data Item sub-entry. The Set Entry may have Member sub-entries. More than one occurrence of a sub-entry may appear within a single entry. Entries and sub-entries are terminated by a period. The notation used in all formats and the rules which apply to all formats are:

The elements wich make up a clause consist of upper-case words, lower-case words, special symbols and special characters.

Upper case words which are not underlined are optional words and need not be used.

Lower case words are generic terms which must be replaced by appropriate names or values.

The meaning of enclosing a portion of a general format in special symbols is:

```
+-   -+
|  a  |
|  b  |   0 or 1 occurrences
|  c  |
+-   -+
```

```
< a >
< b >    exactly one occurrence
< c >

|| a ||   at least one occurrence
|| b ||
|| c ||   at most one occurrence of each
```

An ellipsis (that is, '...') indicates repetition is allowed. The portion of the format which may be repeated is determined by the ' ' or ' ' which logically matches the ' ' or ' ' to the immediate left of the '...'.

CHARACTER SET

The character set consists of 51 characters which include letters of the alphabet, digits and symbols. The symbols = < > are reserved for future use. The special symbols are + - * , ; . " ( ) / $.

WORDS

A word is a sequence of not more than 30 characters. Each character is selected from the set of letters and digits and the symbol '-'. The '-' symbol may not appear as the first or last character of a word.

RESERVED WORDS

Reserved words are a list of words which have special significance. They may not be used as user-defined words. The reserved word list is given in Appendix D.

NAMES

A name has two forms. A word beginning with a letter may be used as a name. An escape form provides that any string be a word if it is enclosed within the currency symbol '$'. The string may include the currency symbol if the '$' appears twice in succession.

LITERALS

A literal is a string of characters whose value is implied by the

ordered set of characters of which the literal is composed. Every literal belongs to one of two types, numeric and nonnumeric.

NONNUMERIC LITERALS

A nonnumeric literal is a string of any allowable characters in the computer's character set, of any length, delimited by quotation marks. The quotation mark may be included if it is written twice consecutively for each of its occurrences. Embedded blanks are considered to be part of the nonnumeric literal.

NUMERIC LITERALS

A numeric literal is defined as a string of digits, the symbols '+' and '-' and '.' and the character 'E'. Numeric literals may be written as fixed-point or floating point decimal.

COMMENTS

Comments may appear anywhere that spaces may be used as a delimiter and are delimited by '/*' '*/' pairs.

PUNCTUATION

The following punctuation characters are used:

One or more consecutive spaces, when not contained in a comment or delimited string, is a separator.

The comma is used as a separator.

The semi-colon may be used to separate clauses.

The period is delimiter for an entry or sub entry and is required.

DATA-BASE-DATA-NAMES

A data-base-data-name may not be subscripted or qualified unless specifically permitted by the rules of the format in which it appears.

DATA-BASE-IDENTIFIERS

A data-base-identifier may be both qualified and subscripted or either qualified or subscripted.

QUALIFICATION

Where the same data-base-data-name is declared in more than one Record Entry, its use as a data-base-identifier may have to be qualified to achieve uniqueness. Syntax rules specify when qualification may be necessary. A name can be qualified even though it does not need qualification.

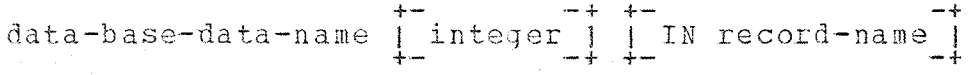
SUBSCRIPTING

Subscripts can be used only when reference is made to a data item within a data aggregate. The subscript must be an integer.

The range of permissible subscript valus is 1 to the maximum number of occurrences specified in the OCCURS clause. Subscripts are written in order of successively less inclusive dimensions of the data orqanizatiоn.

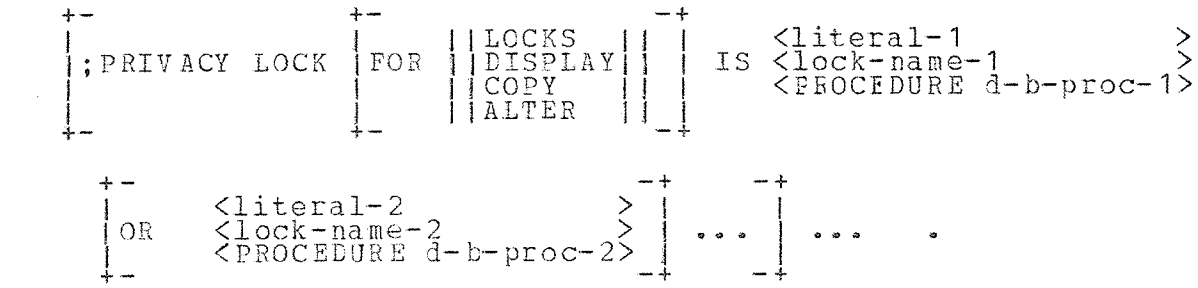
FORMAT

The format of a data-base-identifier in the DDL for the schema is:



4.4.2 SCHEMA ENTRY

Each schema known to the GDBMS is given a unique name. This name is used to find the records in the schema database containing the object version of the schema. The syntax of the Schema Entry is given below.



The schema database is secured against unauthorized access through the use of PRIVACY LOCKS. Access can be restricted for any or all of the functions ALTER, COPY DISPLAY and LOCKS. Alter access permits the modification of the schema with the exception of the privacy lock clauses. Copy is used in the sub-schema definition to copy portions of the schema into the sub-schema definition. Display permits the viewing of the schema with the exception of the privacy locks. Locks permits the viewing, creating, or changing of privacy lock clauses throughout the schema.

When a schema is referenced the GDBMS matches the contents of lock-names and literals and initiates the privacy procedures to determine right of access. Privacy procedures return a yes/no access indicator. Literals, lock-names and data-base-procedures may appear in more than one privacy clause. When the privacy FOR option is omitted all privacy procedures, literals and lock-names apply to all functions. If no privacy is specified for a function then use of that operation is unrestricted.

4.4.3 AREA ENTRY

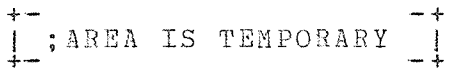
The unit of external storage allocation for the database (equivalent to an OS dataset) is the AREA.

```
AREA NAME is area-name-1
```

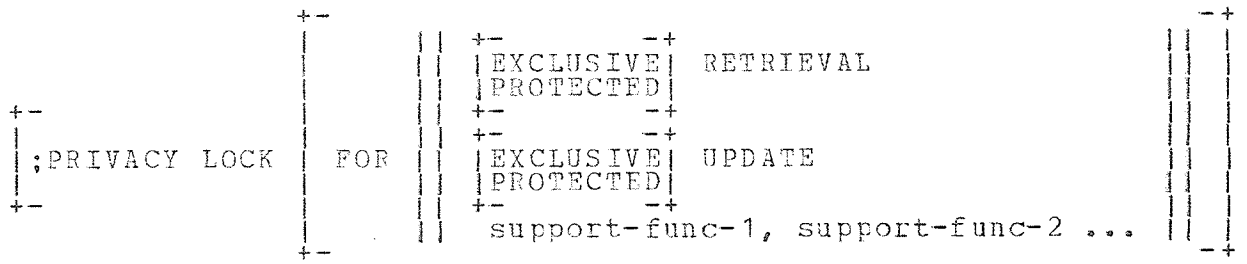
An area is that portion of the database containing all records having a range of database keys. Each area defined within a database is uniquely named.

The physical placement of an area is handled through the use of Operating System data management facilities. The use of areas allows the Data Base Administrator to control placement of data as a means of optimizing performance. This facilitates the allocation of areas to offline storage if the records within an area are not always required. Areas will be separated to minimize contention problems on secondary storage devices.

Areas may be temporary (ie allocated for the duration of the run-unit) or permanent.

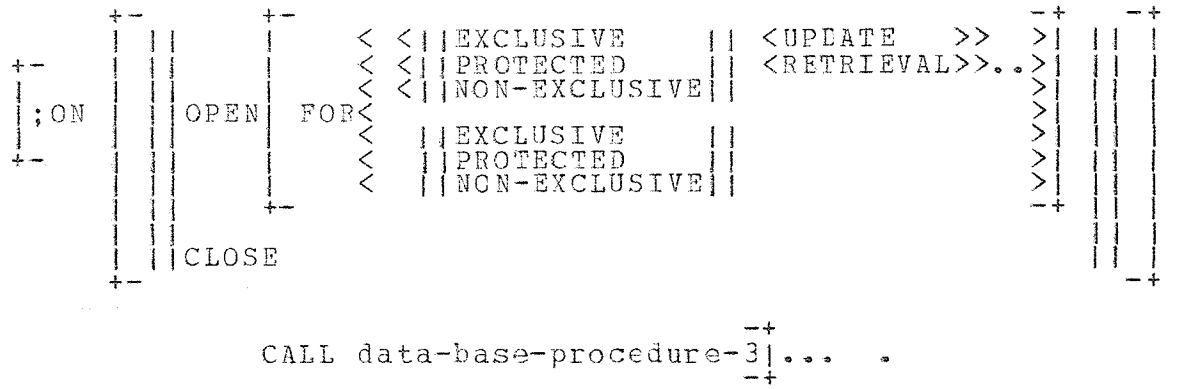


Temporary areas belong to the run-unit which allocates them. They are deleted on termination of the run-unit. Concurrent access to temporary areas is not supported. Record occurrences in a temporary area cannot participate, either as owner or member records, in occurrences of sets which contain records that are not in temporary areas. This prevents set paths from being built which will have discontinuities when the temporary area is deleted. Temporary areas have application as work areas for production programs. Their primary use is as the area assigned to a non-procedural user who works from a terminal and defines a temporary file to produce a report or some statistics. Allocating temporary areas ensures that space will not be retained for the storage of useless data. It also allows the GDBMS and Operating System data management routines to place the areas dynamically on volumes whose present use count is low.



The PRIVACY LOCK clauses apply as in the Schema Entry when the specified type of access is requested.

Support-functions are parts of the GDBMS and include such functions as loading, copying, patching and dumping part of the database. The availability of these functions in the GDBMS allows dynamic maintenance to take place on subsets of the database. Controlled access ensures that a minimum disruption to normal service will be experienced by regular users.



The ON clause is used to provide usage contrcl over an area. When an AREA in OPENed or CLOSEd the procedure specified is executed. The procedure is passed a ccmmand code and returns an ERROR-STATUS. The command code specifies the type of access requested. A non zero error-status indicates that ancther run-unit has control of the area in a manner which precludes control as requested in the On clause. This clause is used to maintain integrity over an area when a run-unit is to update records in the area.

If OPEN or CLOSE is not specified the procedure gets control

whenever the run-unit issues an open or close. If OPEN is specified and the FOR option is not given, the procedure is executed whenever the run-unit issues an OPEN to the area. If the FOR option is stated, but neither UPDATE or RETRIEVAL is specified, then the procedure is executed whenever the run-unit issues the specified type of open, whether for update or retrieval. When UPDATE or RETRIEVAL are specified with no qualifiers then the procedure is executed when an open is issued for update or retrieval. When the FOR option is used and either/or UPDATE and RETRIEVAL are used but only some of the qualifier words are given, then the qualifiers apply only to the closest use of UPDATE or RETRIEVAL. For example FOR EXCLUSIVE RETRIEVAL UPDATE is equivalent to FOR UPDATE EXCLUSIVE RETRIEVAL.

Since opening an area for exclusive control precludes the use of the area by another user, it is critical that the Data Base Administrator (DBA) ensure that the run-unit requests only the level of control required by its processing. Usage statistics should be collected which indicate the type of access requested. These will allow the DBA to investigate unauthorized exclusive or protected control. Controlled access ensures that a minimum disruption to normal service will be experienced by regular users.

4.4.4 RECORD ENTRY

A record type is assigned a name unique to the schema. The record is the GDBMS unit of logical access within the database. Each record is assigned a unique database key at load time.

```
RECORD NAME IS record-name-1
```

The database key is selected from those available to the area in which the record will be stored. Database key values remain invariant for the record until the record is removed from the database. Only then can the GDBMS reassign the database key to a new record. Each data item and data aggregate is logically associated with a record type.

```
                    <DIRECT <data-base-name-1>                    >
+-                  <       <data-base-identifier-1>              >
|                   <    +-          -+                          >
|;LOCATION MODE IS  <CALC|d-b-proc-1| USING d-b-ident-2            >
|                   <    +-          -+                          >
|                   <         +-          -+                      >
+-                  <         |,d-b-ident-3|...                   >
                    <         +-          -+                      >
                    <                        +-  -+               >
                    <       DUPLICATES ARE  |NOT| ALLOWED         >
                                             +-  -+               >
                    <VIA set-name-1 SET                           >
```

The LOCATION clause provides the GDBMS with a record retrieval method.

Three access modes are defined. DIRECT accessing implies that the database key of the record is available when a request is made to retrieve the record.

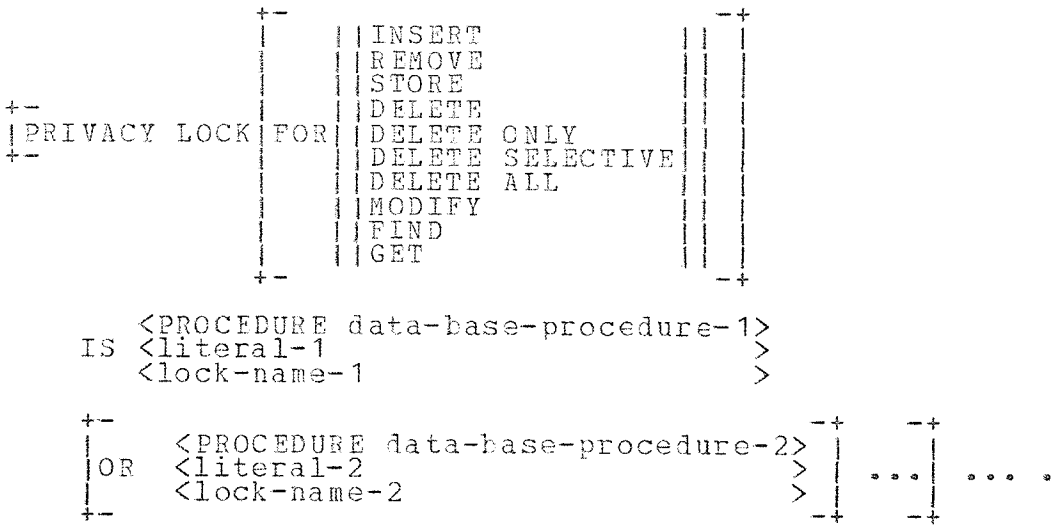
CALC accessing uses a key transformation procedure to calculate the database key. The procedure is supplied by the GDBMS or optionally by the user. A parameter list of identifiers from which to calculate the key is supplied as argument to the procedure. The DUPLICATES clause which must be stated when LOCATION MODE IS CALC is used, refers to the CALC keys. If the DUPLICATES ARE NOT ALLOWED clause is specified no additional occurrence of this record type will be allowed to exist if all of its CALC keys are identical to those of an occurrence already in the same area. This avoids storing records which can never be retrieved, since the calculation will always return the address of the first record.

VIA accessing implies that the record can be retrieved from the SET given. The SET SELECTION clause defined in the Set Entry provides further information on how to select the correct occurrence of the set.

```
;WITHIN area-name-1

     +-                                                    -+
     |<,area-name-2>... AREA-ID IS data-base-data-name-1|
     +-                                                    -+
```

A record occurrence in the WITHIN clause is assigned to a particular area. Record occurrences may be stored in a single area or in one of list of areas. In the second case the storage area is determined dynamically by the run-unit at execution time. Record occurrences of a particular record type may be stored in different areas. This provides the user with control over placement of records to optimize retrieval. A typical use of this feature concentrates the storage of records in a set occurrence.

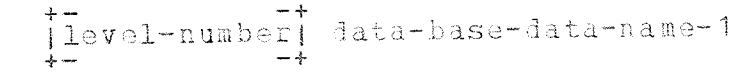
```
          +-                                  -+
          |  ||INSERT                        ||  |
          |  ||REMOVE                        ||  |
          |  ||STORE                         ||  |
+-        |  ||DELETE                        ||  |
|;ON      |  ||DELETE ONLY                   ||  |
+-        |  ||DELETE SELECTIVE              ||  |
          |  ||DELETE ALL                    ||  |
          |  ||MODIFY                        ||  |
          |  ||FIND                          ||  |
          |_ ||GET                           ||_ |
          +-                                  -+

                                                -+
     CALL data-base-procedure-2|... .
                                                -+
```



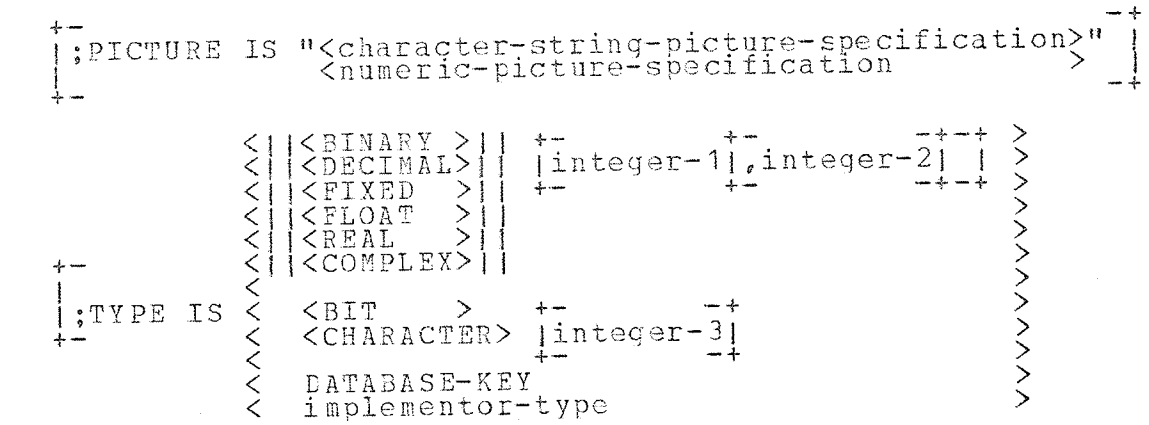
Access to records is controlled through the PRIVACY and ON clauses. In addition the ON clause specifies the type of access that the run-unit requires. Control of access at this level provides record level enqueueing as opposed to the area level enqueueing defined in the area entry On clause. Simultaneous access to a record which may be modified is controlled by the database procedure. The GDBMS provides data integrity through a facility known as monitoring. The current record of the run-unit is always in monitored or extended monitored mode. No other record known to the run-unit can be in monitored mode. Monitored mode is established automatically when a record becomes current of run-unit. A record remains in monitored mode until it ceases to be the current record of a run-unit or a REMONITOR is executed for the record. A record is placed in extended monitor mode by execution of a KEEP. This allows any number of records to be controlled by the run-unit. If a record in monitored mode or extended monitor mode is changed in any way the GDBMS determines whether the record has been modified by a concurrently executing run-unit. If the record has not been concurrently modified the change is executed, otherwise an error status code is returned to the run-unit and the change is not done. The run-unit must GET the record again and reapply the change.

4.4.5 DATA SUB-ENTRY

Data items and data aggregates which are defined to be part of the record type specified in the preceeding Record Entry are defined in a Data Sub-Entry.



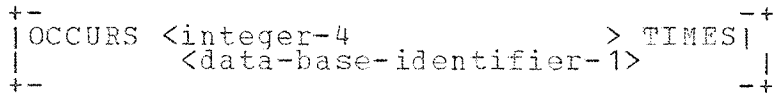
The data-base-data-name assigned to each data item and data aggregate must be unique within the record type. A level number may be assigned to each data-base-data-name to define intra-record structure. The levels represent node levels in the hierarchical intra-record structure.



The PICTURE and TYPE clauses define the internal storage type of the data item and scaling, mode and decimal alignment. The PICTURE clause contains a sequence from the set of characters " 1 2 3 9 V P o A X S T E + - " plus a repetition factor of the form "(n)character", to pictorially describe the format of a data item. A data item gets type string or arithmetic from the occurrence of these characters. The TYPE clause when used in conjunction with the PICTURE clause provides for the storage of the data item value in a form other than character. Valid data types are FIXED BINARY or DECIMAL and FLOAT BINARY or DECIMAL for arithmetic data items and string types CHARACTER or BIT.

The TYPE clause when used without the PICTURE can define a wide

variety of arithmetic and string type data storage formats. For arithmetic data types BINARY or DECIMAL for FIXED, FLOATING, REAL and COMPLEX numbers may be defined. For string, BIT and CHARACTER types may be defined. Integer-1 and integer-2 are used to specify precision and scale of arithmetic items and the length of string-type items. A special implementor defined data type DATABASE KEY is defined to hold the database key value. This data item has special meaning and a data item with this type cannot participate in arithmetic operations or comparisons except for equality with other data items of the same type. The final data type is a potpourri. Any data type the implementor cares to define may be declared. Normally data types will be defined for values such as DATE and TIME. This provides a facility to have the DATE or TIME returned as the value of a data item. Additional types will be peculiar to applications. As an example a data item may be defined to have type CURRENT-PRICE-OF-WHEAT, in an application where this is a useful value.



The OCCURS clause is used to define a vector or repeating-group of fixed or variable dimension. Data-base-identifier-1 must be a data-item in the record described by the corresponding Record Entry. A repeating-group may be defined as an alternative to defining a new record type and a set type which contains the record as member.

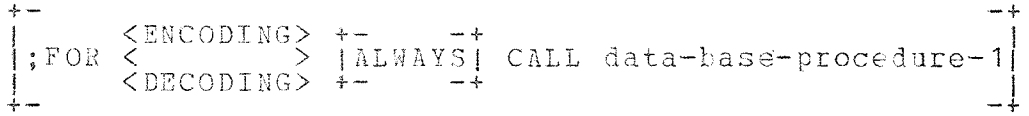
```
+-
|     <ACTUAL > RESULT OF data-base-procedure-1
|;IS <        >
|     <VIRTUAL>
+-

     +-                        +-              -+   -+
     |USING d-b-ident-1 [,d-b-ident-2]...        |
     +-                        +-              -+   -+
     +-          +-                  +-              -+ -+
     |MEMBERS [record-name-1][,record-name-2]... set-name-1|  |
     |          +-                  +-              -+ -+  |
     +-                                                    -+
```

The ACTUAL/VIRTUAL RESULT clause specifies a procedure to be executed to obtain the data item value. A data item defined with the ACTUAL RESULT clause is physically included in the record. Data-Base-Procedure-1 is invoked to update the data item whenever any data items or member records of set-name-1 are altered in any way which affects the result. A data item defined with the VIRTUAL RESULT is not physically included in the record. Data-base-procedure-1 is invoked when a request for a virtual result data item is received. Actual result data items are useful as total data items. They may contain the result of an arithmetic function executed on several other data items in the database. When referenced the result of the function is immediately available. When the result is very dynamic or seldom requested it may be stored as VIRTUAL to save space or maintenance time. In this case the procedure must be executed before the data item can be returned. When very rapid response to a request for the data item is not essential result data items should be defined as virtual.

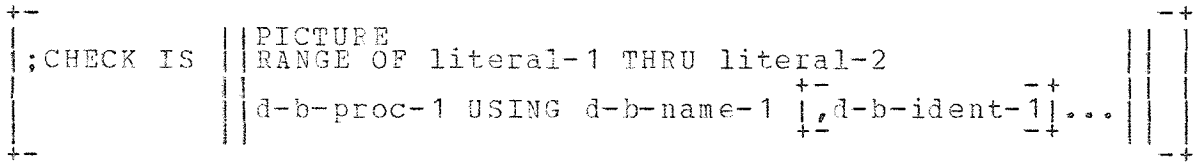
```
+-                                                          -+
|     <ACTUAL >                                             |
|;IS <        > SOURCE IS d-b-ident-1 OF OWNER OF set-name-1 |
|     <VIRTUAL>                                             |
+-                                                          -+
```

The ACTUAL/VIRTUAL SOURCE clause specifies the origin of the contents of a data item and whether or not that data item is to exist physically within the data base. A data item defined to be actual source is physically included in the member record. The GDBMS is responsible for keeping the values of the data item equal to the value of database-identifier-4. The clause is used to store data items

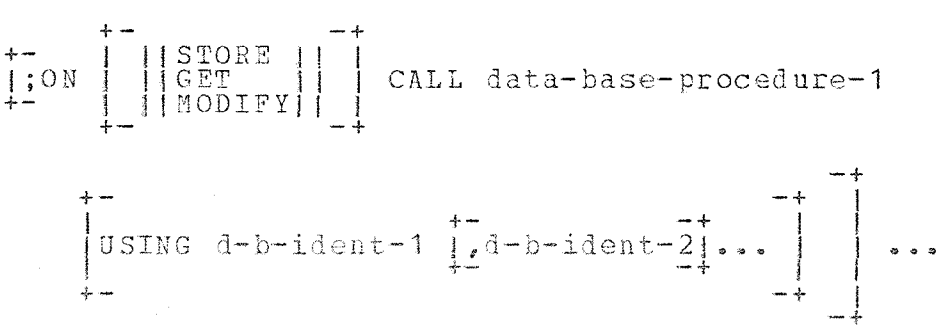
physically redundantly. This can expedite the retrieval function at the expense of redundant storage and maintenance. A single logical update to any of the actual data items causes all data items to be updated. The virtual source data item is not physically included in the member record. The GDBMS retrieves the data item from the record in which it is stored or returns a implementator defined null value if the record occurrence containing data-base-identifier-4 does not exist.

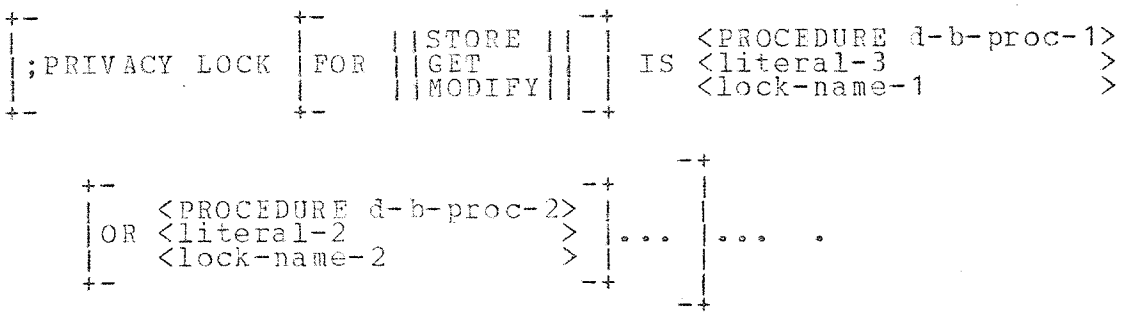


The ENCODING/DECODING clause is used to cause data items to be converted according to the rules of data-base-procedure-4. This facility is used when non-standard data conversions are to be executed. It also can be used to encrypt data either for security or data compression purposes.



The CHECK clause is used to inhibit data conversion or to cause validity checking of the data item value. When a data item value is stored it can be checked for reasonableness. The value can be checked to ensure that the value satisfies the constraints of its PICTURE definition. The GDBMS can verify that the value lies between a RANGE of two values. The GDBMS can call a user supplied procedure to verify the value using a list of other values in the record.





The PRIVACY and ON clauses provide functional control of access at the data item level. These clauses will typically be used at this level only on very sensitive data. It is very expensive to execute these procedures for each access to a data item.

4.4.6 SET ENTRY

The set entry defines inter-record structure.

```
SET NAME IS set-name-1
```

The set name clause defines a name for a set type which is unique within the database.

```
                <          +-                  -+>
;MODE IS        <CHAIN     |LINKED TO PRIOR|  >
                <          +-                  -+>
                <                +-          -+>
                <POINTER-ARRAY   |DYNAMIC|    >
                <                +-          -+>
                <implementator-name            >
```

The MODE clause defines the manner in which record occurrences in a set type are acssociated. Records participating in a set occurrence whose mode is CHAIN are linked to successor records. The chain may optionally be two-way (ie LINKED TO PRIOR). Chains may be implemented as embedded links or non-embedded lists. The record pointer is a database key.

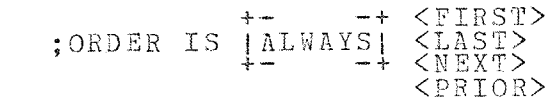
A POINTER-ARRAY is functionally equivalent to a non-embedded chain. The database keys of all member record occurrences are contained within the occurrence of the owner record. The NEXT record is the record whose database key occurs next in the list. The use of lists may provide an efficient method for enquiries using Boolean AND/OR conditions. Records satisfying the conditions can be found from the lists without retrieving unnecessary records. Pointer-array mode should be used when only a subset of the records in a record type will be members of the declared set type. This avoids the overhead of storing a pointer in each occurrence of a record of the record type.

A DYNAMIC set is one in which no record types are declared to be members of the set type. At run time any record type in the database can dynamically be associated with a dynamic set occurrence. The dynamic option can only be used with a POINTER-ARRAY since it is not possible to anticipate when a record type or occurrence will become a member of the set type. It is therefore not possible to identify the

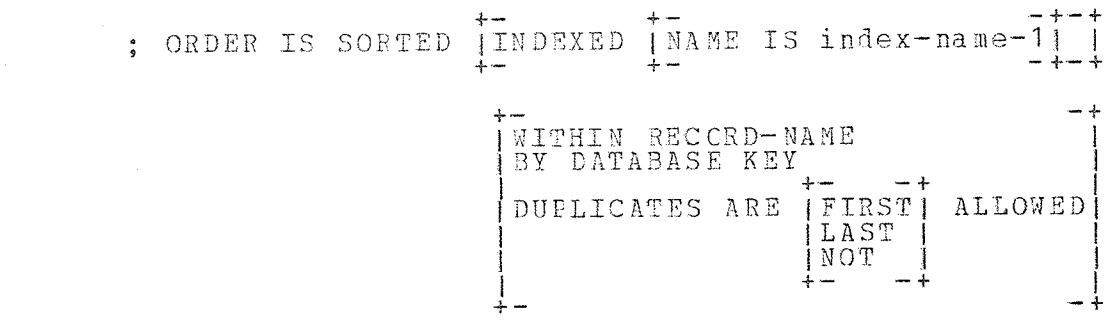
location of a dynamic set chain in a record occurrence. For the same reason member records of dynamic sets cannot point to their owners.

The implementor-name mode provides for the implementation of further set relationships. INDEXING modes will almost certainly be made available where access method data management services exist to support this data structure. In conjunction with the POINTER-ARRAY, bit vector information structures may be implemented to allow for more complex multi-key search arguments. The keys stored in a POINTER-ARRAY might be implemented as data item key values to allow indirect addressing to records. Information about record content can then be gathered from the list.

format-1



format-2



The ORDER clause controls the logical ordering of member record occurrences within each set occurrence. In the first format new records are inserted:

FIRST - as the immediate successor to the owner record occurrence.

LAST - as the immediate predecessor to the owner record occurrence.

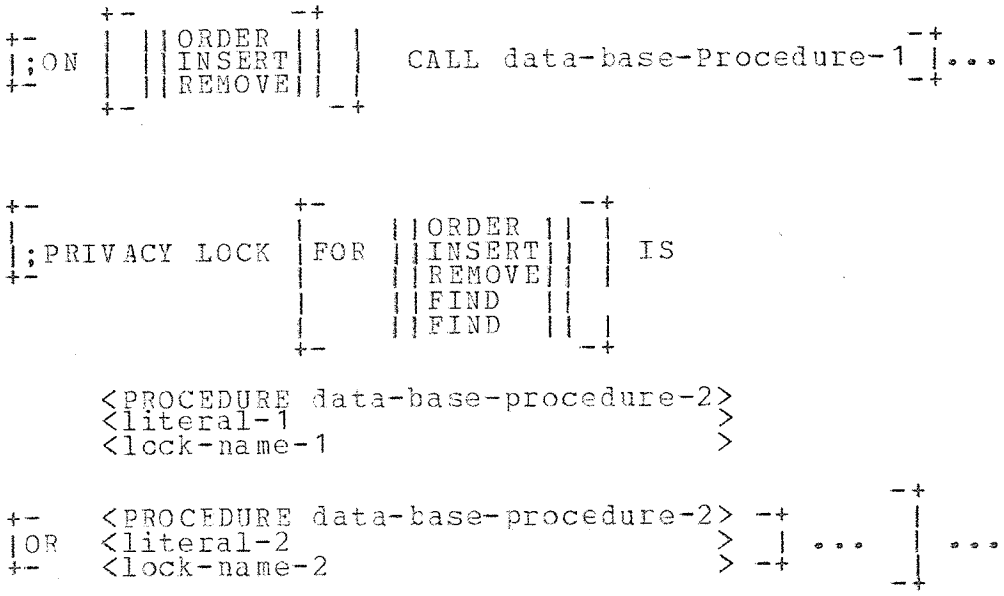
NEXT, PRIOR - after or before another record occurrence which is current to the run-unit.

Use of ALWAYS indicates that the run-unit cannot override the

schema DDL order clause.

Member records can be stored in sequence of a key defined in the Member sub-Entry ASCENDING/DESCENDING clause. Specifying only ORDER IS SORTED causes member record occurrences to be sorted on the regular key field of each record within record type. When the set type is DYNAMIC no ordering can be defined for potential member record types since a key field cannot be defined at schema definition time. Records can optionally be stored in sequence of database key within record name or strictly by database key. Member records can be associated by an implementor defined index structure. The GDBMS interfaces the schema with the Operating System data management routines via an index-name. Duplicates can be not allowed or allowed. New duplicate occurrences can optionally be stored last or first in the sequence of duplicates.

Privacy and On clauses provide protection and control of the set relationships. Monitored control is available for set modification. Access can be restricted for any set function.



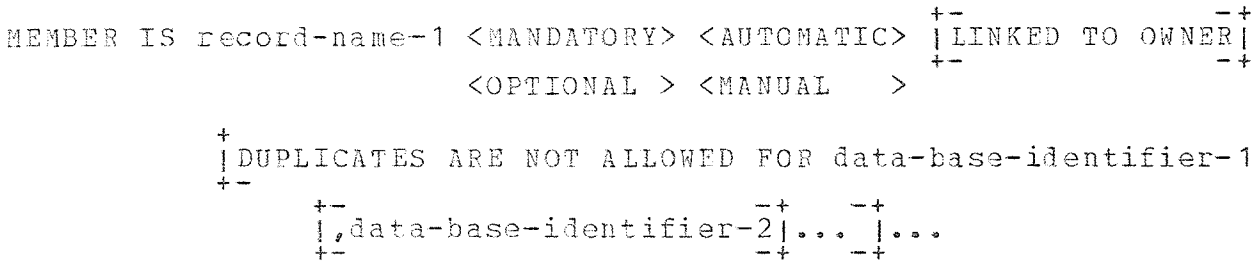
Each set type must have a single record type declared as owner.

```
; OWNER IS  <record-name-1>    .
           <SYSTEM>
```

Each occurrence of the record type establishes the existence of a set occurrence. A special type of set, the SINGULAR set has only one occurrence and has as owner the SYSTEM (GDBMS).

Member Sub-Entry

The member sub-entry names record types which may be member record occurrences in the occurrence of the set type of the set entry.



A record which is declared as a member of a set type may also appear as member or owner of other set types. No record may be declared as owner and member in the same set type.

AUTOMATIC membership implies that each occurrence of the record named will become a member of a selected set occurrence. MANUAL membership implies that the run-unit must issue a DML command to affect membership. An OPTIONAL member may be removed from a set occurrence into which it has automatically or manually been placed. A MANDATORY member may never be removed from its set occurrence, without being deleted from the database, once it is established as a member. RING structures are created with the LINKED TO OWNER clause. If it is not desirable to insert a record into a set occurrence as member when key fields are completely duplicated within an existing occurrence, then the DUPLICATES clause instructs the GDBMS to disallow an attempted insertion. Since the GDBMS allows modification of key fields and the logical reshuffling of records to maintain set order, it is possible to generate a duplicate by changing an existing key. Such an attempt, when the DUPLICATES clause is used, will fail and the database will remain unchanged. The run-unit will be notified of the failure via a status code.

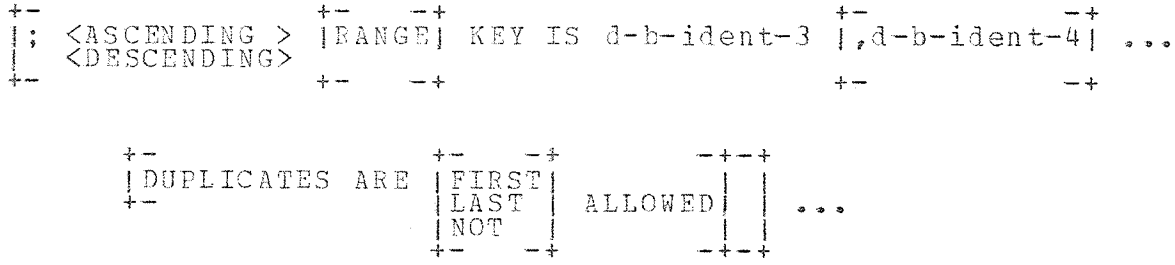
Manual membership may be defined when it is desirable to have record insertion done during prime time, but the load on the system is too great to allow the updating of all set relationships to be done at store time. In this case set updating may be done when time is available or batched and run in non-prime time. This allows new

records to be stored in the database when they are received but defers the updating of non-critical relationships.

Optional membership is used when it is desired to remove a record from a set relationship but retain the record in the database. Historical data records are likely to use this facility. When a record is first entered it is current and belongs in the current set relationship. After some period of time the record is no longer current but it still has historical value. The record can be deleted from the current set and stored in the historical set. The DML has a delete selective facility which allows records to drop membership in sets without being removed physically from the database.

If one or more member records of a set are declared as MANUAL it is possible to generate a cycle. Since automatic membership in a set implies that an occurrence of a member record is unconditionally inserted into an occurrence of its associated set type, when the record is added, at least one record type in a cycle must be a manual member. In order to insert an automatic member record occurrence into a set occurrence its owner must exist. If all members are automatic then no record can be the first stored. If however one is manual and its LOCATION MODE is other than VIA then the record for which no owner exists can be stored independently of its owner existence. Once the member is stored subsequent records, automatic and manual, can be stored based on it as owner. Eventually a record may be stored which has as member the original record. For example if A is a manual member and it is stored. A is the owner of B and C which are stored. B is the owner of D, E, and F which are stored. D is the owner of A which has been stored and a cycle now exists ie ABDA.

Cycle



When an ORDER IS SORTED clause, with other than the DATABASE KEY option, appears in a set entry the ASCENDING/DESCENDING clause must be used to specify the sort contrcl keys for the member record. When the ORDER IS SORTED clause does not indicate what to do with duplicates the DUPLICATES CLAUSE may be specified in the ASCENDING/DESCENDING clause. If no sequencing is defined then database key is used as sort control key. The RANGE option specifies that the named identifier implies a range of values to be used in comparisen for set occurrence selection. Equality between the range key and the input argument value in the User Work Area (UWA) is not required for a record to be selected as owner of the sought set occurrence. A match will occur regardless of whether range key is specified as ascending or descending when:

The input argument value in the UWA equals the value of any specific range key.

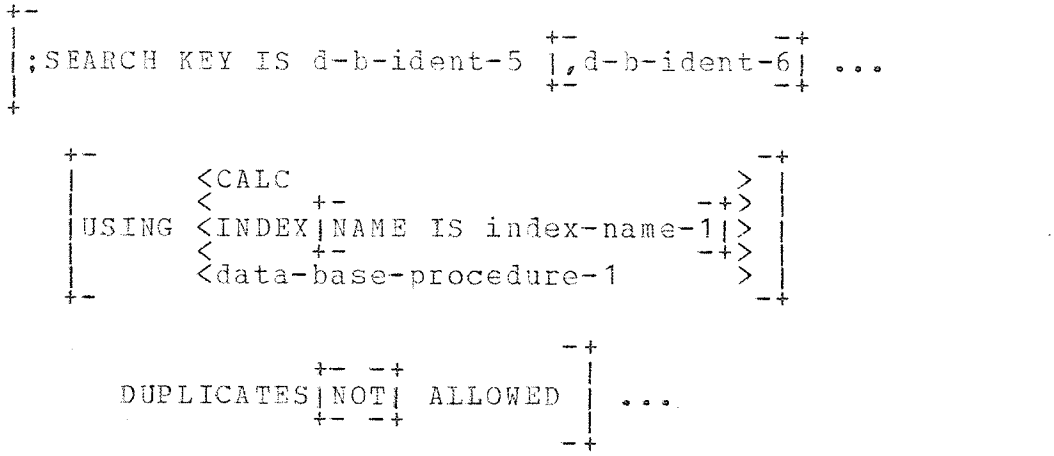
The UWA value is less than the lowest value of any range key; then the match will occur with the lowest range key.

The UWA value lies between two adjacent range key values; then the match occurs with the higher range key values.

A match will not occur if the UWA is greater than the largest value of any RANGE key.

RANGE keys can significantly reduce search time to recover a record. As an example, suppose an alphabetic key is defined as a RANGE key. Twenty-six ranges can be used to subdivide record occurrences into 26 subsets of about 38 records each. Where previously the search of a linked set of 1000 records takes an average of 500 record accesses, the search now takes only an average of 13 + 19 = 32 record accesses. A ring structure and more range keys can be used to further reduce searching. This indexing scheme is very useful when the number of records in a set occurrence can be large and when the distribution

of key values can be approximated.

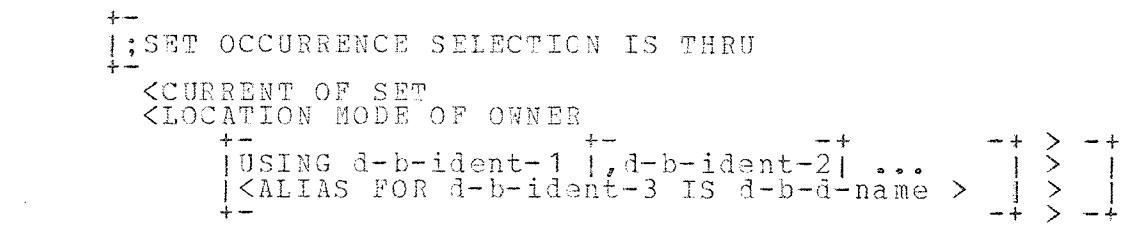


The SEARCH clause provides the GDBMS with a technique for finding particular member record occurrences within a selected set occurrence. Three search strategies are provided. The CALC technique uses a GDBMS supplied key transformation on the identifier list to find the database key of the desired record occurrence. A system defined index is created when no USING information is given. The INDEX technique implies the existence of an Operating System supplied data management indexing access method. The index-name relates the OS index with the schema description. Optionally a user may provide a procedure to operate on the identifier list and produce a database key for the desired record occurrence.

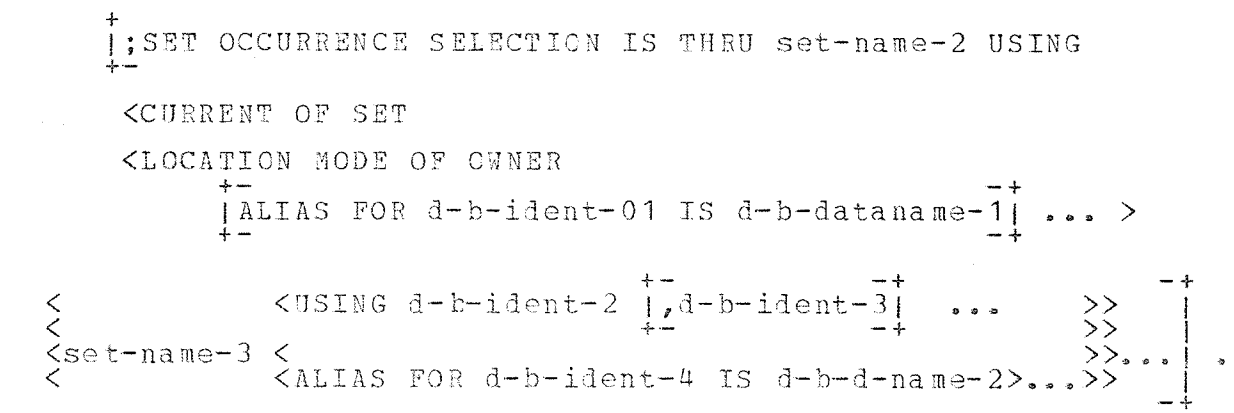
The SEARCH clause is used when it is expected than many member records will occur for particular set occurrences.

The DUPLICATES clause is used to specify whether record occurrences with completely duplicate search keys will be inserted into the set occurrence. When records are retrieved by key values subsequent occurrences of records with identical keys will not be retrieved.

format-1



format-2



The SELECTION clause is used to define rules for the selection of a set occurrence containing a particular member record occurrence, whenever the set is non singular. A sufficient condition to select the set occurrence is to find the owner record occurrence. An owner record may be the current record of set to the run-unit, in which case it is immediately available. It may be possible to locate the owner record using the LOCATION MODE information given in its Record Entry. The arguments used in locating the record are those of the Record Entry LOCATION MODE clause and may be provided in the USING and ALIAS clauses. The run-unit must ensure that they are available. If the LOCATION MODE is not VIA a set then the occurrence is immediately accessible whenever the necessary identifiers are available. When the LOCATICN MODE is VIA then the GDBMS must trace back through a chain of set relations until a set whose owner record is directly accessible is located.

The two formats of the SELECTION clause differ in that the second form is used when a set path must be defined. That is, when the immediate owner record of the set occurrence to be selected cannot itself be uniquely selected except in terms of its membership in

another set and the criteria for selecting that set are included in a SELECTION clause. This condition may occur to an arbitrary number of levels creating a set path. Eventually an owner record must be reached which is CURRENT, or can be selected uniquely because its LOCATION MODE is DIRECT or CALC.

An ALIAS clause may be required when a particular record is defined as a member of more than one set type and each such set type has the same owner record type. In this case it may be necessary to provide more than one argument value for an identifier in subsequent retrievals of the record. An ALIAS provides a facility for the run-unit to establish alternate values for the identifier in his UWA.

When the LOCATION MODE of the Record Entry is VIA and the set occurrence is being selected using LOCATION MODE OF OWNER then it is necessary to use a USING or ALIAS clause to specify information to be used in the selection of the VIA set occurrence.

Specification of the set selection criteria requires that the DBA be aware of all set and record descriptions in the database. Only with a complete knowledge of these can he specify efficient set selection criteria. Much of the complexity of the schema DDL lies in this declaration, and in order to simplify the definition the DDL analyzer may have to produce set, record and repeating-group relational information.

4.5 BIBLIOGRAPHY

CODASYL DATA BASE TASK GROUP REPORT of APRIL 1971
pub Association fcr Computing Machinery
Single Copy Department
1133 Avenue of the Americas
New York, N.Y. 10036

5.1 INTRODUCTION

The Data Manipulation Commands provide, in conjunction with a host language, facilities to access and manipulate records defined in a sub-schema consistent with the host language. CODASYL envisions a time when each host language will be extended to include a sub-schema and a DML for use with databases defined in the common schema DDL. The CODASYL DBTG Report of April defines a sub-schema and DML for COBOL.

The DML requires that the host language make available fixed data areas for the communication of information between the GDBMS and run-unit. Six special registers contain status information maintained by the Generalized Data Base Management System (GDBMS) but available to the run-unit for interrogation.

DATABASE-STATUS is a special register capable of containing five alphanumeric characters. The execution of every DML command causes a value to be stored in DATABASE-STATUS. This value known as the Data Base Status Indicator, is made up of a Statement Code in the leftmost character positions, and a Status Code in the rightmost character positions. When the execution of a DML command results in a Data Base Exception Condition it is posted in the Status Code and the command code is posted in the Statement Code. Valid executions result in a zero indicator. The run-unit is responsible for the interrogation of DATABASE-STATUS. Execution logic should be based on the posted status after each DML command is executed.

The data-names DATABASE-AREA-NAME, DATABASE-RECORD-NAME, and DATABASE-SET-NAME define special registers capable of holding 30 alphanumeric characters each. When the execution of a DML commands results in a Database Exception the associated area, set and record names are stored in these registers. The run-unit can examine these registers and take appropriate action based on their contents.

DATABASE-CONFLICT is a special register capable of containing the values 00 to 15. The GDBMS alters DATABASE-CONFLICT when the current run-unit attempts to alter a record or record relationship that has been altered by a concurrent run-unit.

DATABASE-PRIVACY-KEY is a special register capable of holding at least ten alphanumeric characters. It is established by the run-unit before an attempt is made to execute a command on protected data. The

DATABASE-PRIVACY-KEY can be changed only by the run-unit for the current record.

Currency status information is also kept to identify the last record occurrence accessed by the run-unit. The current database key is maintained for:

1. each record-type known to the run-unit (ie "current of record-name"),
2. each set-type known to the run-unit (ie "current of set-name"), (This may be any record which participates in the set type, either as owner or member.)
3. each area-name known to the run-unit (ie "current of area-name"), and
4. any record-type known to the run-unit (ie "current record of run-unit").

The GDBMS maintains the currency indicators and the run-unit may access them at any time. Many of the DML commands use the currency indicators to control execution. Unless otherwise qualified the "current record" is assumed to current of run-unit.

The DML commands are of three types: those which operate on records, those which operate on sets, and those operating on areas.

Only those data items, records, areas and sets defined in the sub-schema may be explicitly referenced by DML commands.

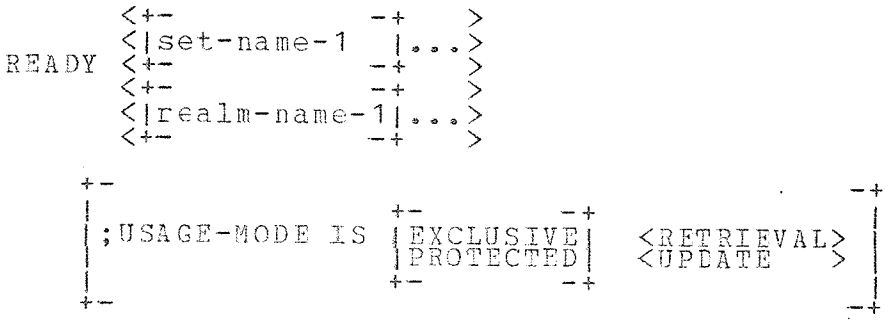
When a record is being referenced by a run-unit it is placed in monitored or extended monitored mode (by a KEEP command). Commands which modify records or record relationships enqueue on all records which may be affected by change. Concurrently executing run-units may not access a record which is enqueued upon by another run-unit. In addition all areas containing records or relationships to be modified must be readied for update before a command which causes update can be executed. Failure to ready an area properly will cause a database exception condition when the DML command is executed.

5.2 Area Integrity Commands

The two commands READY and FINISH ensure database integrity at the area level. Throughout this section the terms area and realm are considered to be synonymous

5.2.1 READY Command

The READY command specifies the access intent of a run-unit for an area or areas.



Temporary areas belong to a single run-unit and must therefore be readied as EXCLUSIVE UPDATE. Any sets defined in temporary areas must be readied with the same intent.

The READY command is designed to avoid 'interlock' (ie two run-units requesting the same record, and at least one requesting update, at the same time). In conjuction with the exclusive and protected operands the ready command helps to ensure that conflicting access to a realm is not initiated.

Specification of an area-name causes the entire contents of the area to be readied for the given intent. Similarly, readying a set-name causes all areas which may contain record occurrences for the set specified to be readied. If all required areas cannot be readied none is.

RETRIEVAL permits access to database records and is the default value. UPDATE permits RETRIEVAL and modification of the contents of database content.

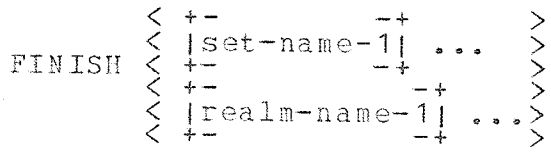
PROTECTED prevents other run-units from executing a READY UPDATE

or EXCLUSIVE intent for the area. EXCLUSIVE prevents other run-units from executing a READY for the area.

Execution of a READY for an area which is readied with a conflicting intent causes the run-unit to set a database status indicator and wait for availability of the required areas.

5.2.2 FINISH Command

The FINISH command indicates to the GDBMS that the run-unit has completed its processing on the specified areas.



Only areas which were readied by the run-unit may be finished by the run-unit.

If a set-list is specified then all areas which may contain record occurrences in sets of the named types are removed from ready mode. If neither a set-list nor area-list is specified then all ready areas are removed from ready mode.

On completion all currency indicators for the selected areas are null and an implied FREE is executed for all records which are in extended monitored mode in the selected areas.

An implied FINISH is executed to remove all ready areas of a run-unit on run-unit termination.

5.3 Set Manipulation Commands

The four set oriented commands are: IF, CONNECT, DISCONNECT, and ORDER.

5.3.1 IF Command

Data Base Conditions enable the run-unit to test for current record involvement in set relationships.

5.3.1.1 Tenancy Condition

The run-unit can determine whether the current record is a member, or owner, or either (tenant) in one or more specified set occurrences.

```
       +-  -+ +-        -+ <OWNER >
 IF    |NOT| |set-name| <MEMBER>
       +-  -+ +-        -+ <TENANT>
```

5.3.1.2 Member Condition

The run-unit can determine whether or not a specified set occurrence has any members.

```
                   +-  -+
 IF   set-name IS |NOT| EMPTY
                   +-  -+
```

The set occurrence is the one selected by the current record of set-type 'set-name'.

5.3.2 CONNECT Command

The CONNECT command causes the current record, which was previously stored as a MANUAL or OPTIONAL member, to become a member of one or more sets for which it's record type has been declared an eligible member.

```
                                              +-           -+      >
          +-           -+      <set-name-1   |,set-name-2|  ...>
 CONNECT |record-name| TO   <              +-           -+      >
          +-           -+      <ALL                                >
```

Record-name must be the current record of run-unit. When ALL is specified the record must be a MANUAL or OPTIONAL member of at least

one set type. When the sets are named the record type specified must a MANUAL or OPTIONAL member of each set named. The record is connected in accordance with the ordering defined in the set entry. The record becomes current of set type for each set in which it becomes a member. If the record cannot be connected to every set specified it is not connected to any. Execution of a connect causes the current record to be placed in monitored mode.

5.3.3 DISCONNECT Command

The DISCONNECT statement logically removes the current record from one or more sets.

```
                                              +-              -+
            +-           -+        <set-name-1 |,set-name-2| ...>
DISCONNECT |record-name| FROM <               +-              -+   >
            +-           -+        <ALL                            >
```

The record type of the current record must be defined to be an OPTIONAL member of all specified set types. ALL indicates that the DISCONNECT applies to all sets for which the current record type is declared as an OPTIONAL member. Currency indicators are not affected. The current record is placed in monitored mode for the duration of execution of the DISCONNECT.

5.3.4 ORDER Command

The ORDER statement is used to logically reorder members of a given set occurrence.

```
ORDER set-name +-       -+
               |LOCALLY|
               +-       -+

    <    <DESCENDING>
    <ON  <          > KEY
    <    <ASCENDING >

      <| |+-              +-              -+    -+              ||>    >
      <| ||record-name-1|,record-name-2|...  |  RECORD-NAME  ||>    >
      <| |+-              +-              -+    -+              ||>    >
      <| |+-              +-              -+    -+              ||>    >
      <| ||record-name-1|,record-name-2|...  |DATABASE-KEY  ||>...>...
      <| |+-              +-              -+    -+              ||>    >
      <| | identifier-1 <,identifier-2> ...                    ||>    >
```

A set occurrence can be permanently reordered only when it is not

defined in the schema to be logically fixed in order. LOCALLY is used when such an ordering is predefined and implies that a temporary area must be created tc contain the newly ordered record occurrencess for the duration of the run-unit.

All record types referenced must be member of the set type "set-name". If the record occurrences exist in non-temporary areas then the area must be opened for exclusive or protected retrieval if LOCALLY reordering or exclusive update when recrdering is permanent.

Records which are members of the set occurrence but not part of the reordering are logically placed at the end of the set occurrence.

RECORD-NAME implies that records are to be ordered by record-name. DATABASE-KEY implies that records are to be ordered by database-key. The identifier list causes records to be ordered using the values of the identifiers in each record. Only one identifier may be specified for each record type.

5.4 Record Accessing Commands

The record manipulation commands are divided into two types: those used in the accessing of records and those used to maintain record integrity. In the first category are the ERASE, GET, MODIFY and STORE commands. The MONITOR, REMONITOR, KEEP, and FREE commands provide features to ensure record integrity. The FIND command is used to ensure record integrity and to make a record available for further processing.

5.4.1 Record Integrity Commands

5.4.1.1 Record Selection Expressions

A record selection expression is used to indicate to the GDBMS the criteria to be used in selecting the record to become current record of run-unit. Record selection expressions appear only in FIND statements. The seven formats of the record selection expression permit absolute and relative record selection criteria. A record may be selected based on its content, database key value, or present currency indicator. When records are selected using currency indicators either a current record or a record relative to a current record can be selected.

Format-1

```
 +-           -+
 |record-name-1|; DATABASE-KEY identifier-1
 +-           -+
```

Format-2

```
 <ANY       >
 <          > record-name-2
 <DUPLICATE>
```

Format-3

```
                                          +-             -+
  set-name-1 DUPLICATE USING identifier-2 |,identifier-3|...
                                          +-             -+
```

Format-4

```
<NEXT          >
<PRIOR         >
<FIRST         >   record-name-3
<LAST          >
<integer-1     >
<identifier-4>
```

Format-5

```
+-              -+
|area-name-1     |
|set-name-2      |   CURRENT
|record-name-4   |
+-              -+
```

Format-6

```
set-name-3  OWNER
```

Format-7

```
                +-                -+
record-name-5   |WITHIN CURRENT|
                +-                -+
   +-                  +-                -+  -+
   |USING identifier-5|,|identifier-6|...|
   +-                  +-                -+  -+
```

All records addressed by a record selection expression must exist in areas which are opened and available (ie not under exclusive control of another run-unit). There are several considerations when using the various formats of the record selection expressions. Whenever record-name-i is used it is implicitly assumed that the record-name may be qualified with a RECORD OF SET set-name or RECORD OF AREA area-name. It may also be replaced with the word RECORD, in which case all record types are valid for this expression.

In format-1:

If specified, record-name-1 must have a database key equal to the value of identifier-1. Identifier-1 must be declared as type database-key.

In format-2:

Record-name-2 must be declared with LOCATION MODE CALC. The USING values must have been initialized by the run-unit. When records exist whose key values are duplicated, the ANY option returns the record whose database key is lowest. The DUPLICATES option returns the next record following the current record.

In format-3:

The selected record has the same type as the current record of set-name-1 and the contents of all named identifiers are equal in the selected record and the current record of set-name-1.

In format-4:

When record-name-3 is used in its unqualified form the selection is executed using the current of record type database key value. In each of the optional forms records are related by database key. The NEXT record is the record with the next highest database key. The PRIOR record is the record with the next lowest database key.

In format-5:

The record selected is current of set, area, record-name or run-unit. When record-name-4 is qualified then the record selected will be current of the qualified set or area. When none of the options is presented the record selected is current of run-unit.

In format-6:

Set-name-3 must not be singular.

In format-7:

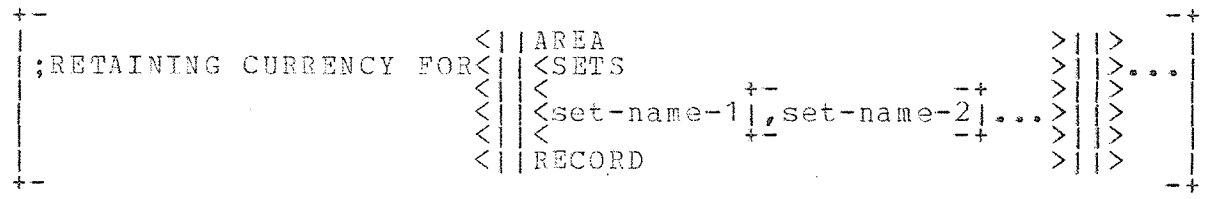
The identifiers must be defined to be part of record-name-5 which must be qualified by a set-name. If WITHIN CURRENT is specified then record-name-5 is a member of the set occurrence in which the current record of the set type is a member. When WITHIN CURRENT is not specified then the set occurrence containing record-name-5 as a member must be selected using the set selection criteria specified for the qualifying set type. When USING is specified the identifiers must be intialized in the UWA record area by the run-unit. They are matched to the record selected by the GDBMS to locate the correct record occurrence. The search and ordering information declared in the schema is used in selecting the appropriate record occurrence. When no USING clause appears the logically ordered first member record

occurrence of the selected set occurrence is returned.

5.4.1.2 FIND Command

The FIND command is used to establish a specific record occurrence as available for subsequent processing. The selected record occurrence becomes current of run-unit.

FIND record-selection-expression

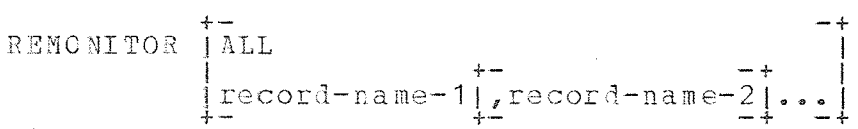


When RETAINING is not specified the record selected becomes current of run-unit, record type, area name, and set type. The RETAINING option allows the run-unit to suppress the modification of currency locations. The special registers DATABASE-AREA-NAME and DATABASE-RECORD-NAME are loaded with the associated area and record name of the selected record occurrence.

The FIND command ensures that the selected record occurrence is in monitored mode. A record may have previously been put into extended monitored mode by execution of a KEEP; otherwise it is put into monitored mode.

5.4.1.3 REMONITOR command

The REMONITOR command is used by a run-unit to ensure that all records required are in monitored or extended monitored mode. If the operation is unsucessful a status indicator is set. REMONITORing conceptually involves the ending of monitored or extended monitor mode for each record selected and the immediate re-establishment of monitored or extended monitored mode for the same record.



When ALL is specified then all records which are in monitored or extended monitored mode except the current record are selected. If a record-name list is given then these records with the exception of the

current record are selected. When neither ALL or a record list appears the current record is selected.

There must be at least one record occurrence available for each selected record. If all selected records are not in monitored or extended monitored mode then the REMONITOR fails and a status indicator is set.

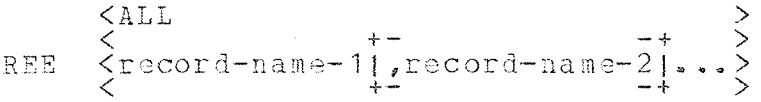
5.4.1.4 KEEP command

The KEEP command causes the current record to be put into extended monitored mode.

KEEP

5.4.1.5 FREE command

The FREE command is used to remove one or more records from extended monitored mode.



If ALL is used then all records in extended monitored mode except the current record are selected and freed. When a record-name list is specified then only the specified records and not the current record are selected. There must be at least one occurrence of each selected record or no record is freed.

5.4.2 Record Manipulation Commands

The four record manipulation commands ERASE, GET, MODIFY, and STORE provide the capability to delete, retrieve, update and store individual records.

5.4.2.1 ERASE Command

The ERASE command is used to logically remove selected records from the database.

```
                                  +-         -+
          +-            -+        |PERMANENT|
    ERASE |record-name|           |SELECTIVE|  MEMBERS
          +-            -+        |ALL      |
                                  +-         -+
```

Record-name must be the current record of run-unit. It is pre-selected with a FIND command. The areas containing all selected records must be readied for update. Areas containing identifiers used in the calculation of ACTUAL RESULT data items must also be opened for update. It is the responsibility of the run-unit to ensure that the sub-schema specifies the correct access control.

The current record is logically removed from the database and is not longer available in a FIND. If neither PERMANENT, SELECTIVE nor ALL is specified then the current record may not be the owner of a set occurrence containing any member record occurrences. If PERMANENT is specified then each MANDATORY member of a set occurrence for which the current record is owner recursively becomes current and is ERASEd from the database. OPTIONAL member record occurrences are DISCONNECTed from the set occurrence but not ERASEd. If SELECTIVE is specified then each OPTIONAL member record occurrence of the set occurrence which is not a member record occurrence in any other set occurrence is ERASEd. ALL specifies that the current record is logically removed from the database and all member record occurrences of set occurrences for which the current record is owner are logically removed from the database.

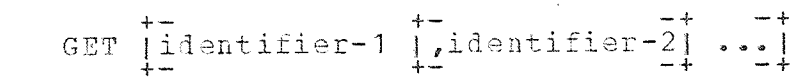
If all selected records cannot be deleted or disconnected then no records are affected. The current of run-unit is null on completion of the ERASE.

During execution of the ERASE command the subsequently current

records are not available to concurrently executing run-units.

5.4.2.2 GET command

The GET command causes data items from a database record, or an entire database record, to be retrieved into its associate record area in the UWA.



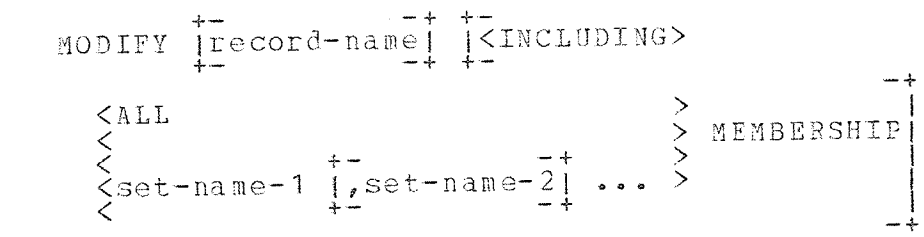
The record to be accessed must first have been the object of a FIND and be the current record.

When no identifier list is given, or the identifier list is a record-name, then the entire current record is made available in the UWA. When the identifier list is specified all identifiers must be defined as part of the current record. Only that data explicitly named is transferred, all other data items remain unchanged.

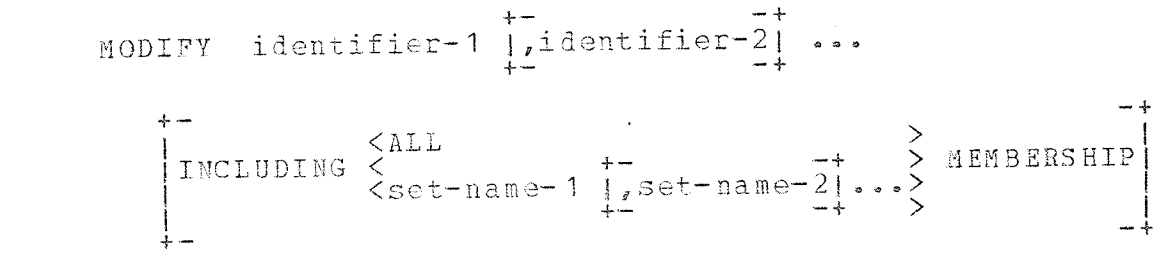
5.4.2.3 MODIFY command

The MODIFY command is used to alter the content of one or more data items in a record and/or to change the set membership of the record.

Format-1



Format-2



The record must have been the object of a FIND and be the current record. Only identifiers which physically exist within the current record occurrence may be modified. Areas containing all records and relationships addressed must be opened for update.

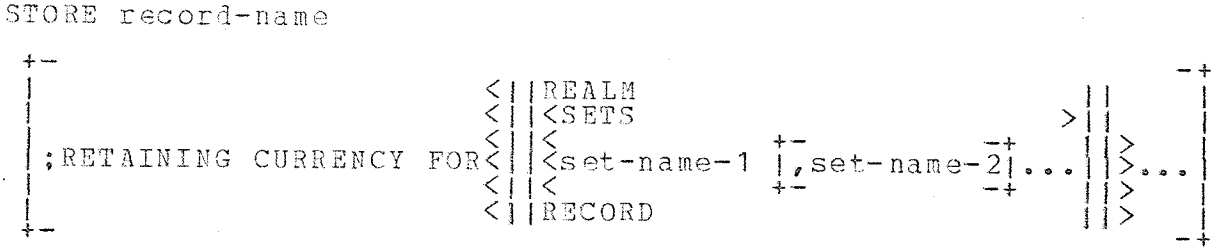
MODIFY and MODIFY INCLUDING cause the content of the current record to be physically updated in the database with the content of the current record (format-1) or identifiers (format-2) stored in the run-unit UWA. MODIFY ONLY does not affect the content of any data item.

MODIFY ONLY and MODIFY INCLUDING cause changes in the set membership of the current record. Membership is affected for ALL, or only the named sets, based on the SET OCCURRENCE SELECTION criteria specified in the schema and sub-schema. The value of appropriate currency locations and identifier and data name values is used in the selection of a set occurrence. The current record becomes current of set for all modified sets.

MODIFY causes the GDBMS to enqueue on the current record and make it unavailable for access by other concurrently executing run-units.

5.4.2.4 STORE Command

The STORE command causes a new record occurrence to be stored in the database from the associated record area in the UWA.



The record is incorporated into all sets for which it is an automatic member or owner. Set membership is determined on the basis of set selection and ordering criterion specified in the schema and sub-schema.

The record becomes the current record. Unless the RETAINING option is used the record beccmes current of each affected currency location. The record remains available to the run-unit in the UWA.

A database key is allocated to the new record on the basis of information supplied in the WITHIN and LOCATION MODE clauses of the schema.

5.5 BIBLIOGRAPHY

CODASYL DATA BASE TASK GROUP REPORT of APRIL 1971
pub Association for Computing Machinery
Single Copy Department
1133 Avenue of the Americas
New York, N.Y. 10036

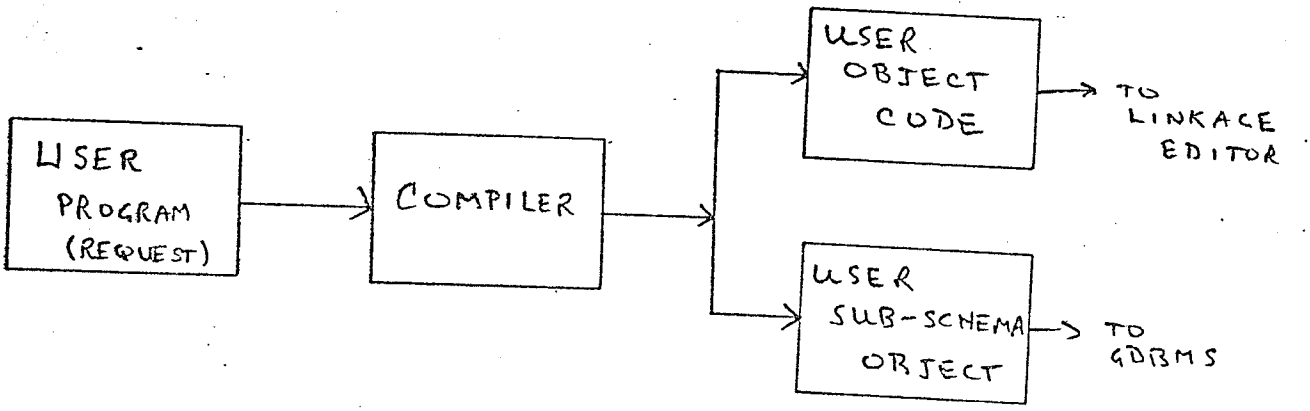
FEATURE ANALYSIS OF GENERALIZED DATABASE MANAGEMENT SYSTEMS
A CODASYL Systems Committee Technical Report May 1971
pub Association for Computing Machinery

CODASYL COBOL Data Base Facility Proposal
A CODASYL Data Base Language Task Group Report March 1973
pub The Technical Services Branch
Department of Supply and Services
5th Floor, 8 Metcalfe Street
Ottawa, Ontario, Canada
K1A0S5

6.1 INTRODUCTION

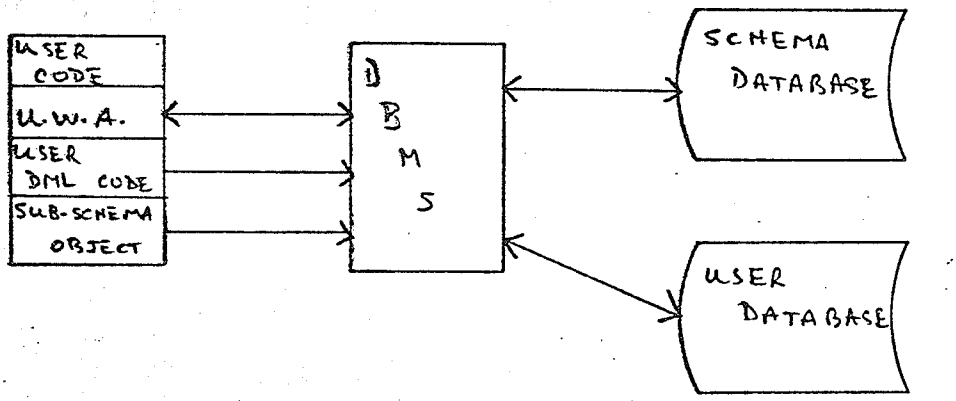
A GDBMS is the interface between a user and his data. The CODASYL GDBMS acts in response to DML commands issued in respect to logical records, described in a sub-schema DDL, to affect associated physical records, described in the schema DDL, in the database. Each of the components of the GDBMS relates to and depends upon other components of the GDBMS. In implementing the GDBMS it is important to create the components in an order which provides each component access to the components required for its operation.

The development of the GDBMS is best illustrated by a sequence of schematics illustrating the interaction of components. A user submits his program with its embedded DML commands and logical record descriptions (sub-schema) to a compiler which in turn produces two object modules. The sub-schema object is passed as is to the GDBMS at execution time. The user program object module, with its DML commands compiled as calls to the GDBMS, is passed to a linkage editor to be converted to load module form before execution.

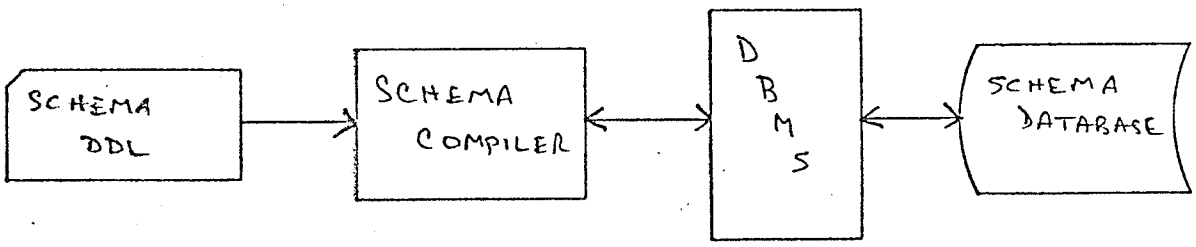


At execution time the GDBMS has access to an object schema describing the user database, the user object sub-schema and the user database. In response to DML requests from the user program, logical records are created and maintained in the User Work Area. The mapping

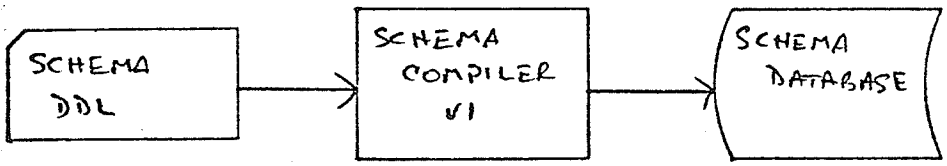
of sub-schema and schema descriptions determines the placement of physical records in the database.



The object schema is created by a schema DDL compiler frcm input DDL statements describing the database. The object schema is maintained in a database of schemas, the schema database. Internal structure of the object schema is described by the schema schema.



The object schema in this case is merely another entry in the schema database. The object schema describing the schema database is created in a similar manner to all other object schema. Before the original object schema is created it is necessary to hand code the object schema structure into the schema compiler.



When an object schema exists is it possible to use the GDBMS and DML commands. Physical and logical organization of the schema database is identical.

6.2 Implementation Plan

As part of this thesis a plan for the implementation of the CODASYL specification is developed. The implementation uses the Translator Writing System (TWS) developed and implemented by William M. McKeeman, James J. Horning and David B. Wortman and modified and implemented at the University of Manitoba by Brent D. Beach.

A brief description of the TWS is included here to provide background for the implementation mechanism. The following excerpt from A Compiler Generator, a text authored by the developers of the TWS, describes the components of the system:

> Our TWS utilizes the language XPL (a dialect of PL/I), for the description of programs, and the metalanguage Backus-Naur Form (BNF), for the description of languages. It consists of three principal components:
>
> 1. ANALYZER, a translator from BNF into syntax tables, which are used in
> 2. SKELETON, a table-driven proto-compiler written in XPL, which when supplied with syntax tables and suitable semantic routines may be compiled by
> 3. XCOM, a translator from XPL to System/360 machine code.

The TWS has two major advantages over direct implementation of the syntax. First, the syntax can be debugged - any inconsistencies in the syntax are detected by ANALYZER - before coding any semantic routines. Changes in the grammer are readily incorporated into the BNF description of the grammer, often with minimal changes to the semantic routines of SKELETON. Second, writing the syntax in BNF breaks the syntax into logical units. In ANALYZER the semantic routines which a programmer writes correspond to the production rules of the grammer defined in BNF. Syntatic units are presented to the programmer in a stack. Only a small part of the total syntax need be considered at any time. The system forces modular programming, since the production rules define the syntax as a sequence of logical modules.

The implementation plan details in bottom up fashion the steps in implementing the schematic outline presented in the introduction.

1. Define in BNF the syntax of the schema Data Description Language (DDL).

2. Define a data structure to contain the information in a schema description.
3. Implement the semantic routines and modify SKELETON to accept the schema DDL as input. Build the data structure of (2) as output. The set relationships in the database must be hand coded in the structure definitions. Name the program DDLV1.
4. Define, using the schema DDL, a database which has the structure of (2). Process this schema and name the output database SCHEMA1. SCHEMA1 is a database containing an object description of the schema database.
5. Define in BNF the syntax of a form of the CODASYL Data Manipulation Language (DML).
6. Implement the semantic routines and modify SKELETON to accept the DML commands as input. The DML commands operate on the schema database using SCHEMA1 as the object version of the schema and sub-schema.
7. Modify DDLV1 to use DMLV1 and produce a new object version of the schema from a new schema written in the schema DDL. The set relationships are established from those defined in SCHEMA1. Name the new program DDLV2, and the new database SCHEMA2. SCHEMA2 contains the object schema for the schema database. DDLV2 uses this object schema to store object schema for user databases in the schema database.
8. Define in BNF a sub-schema which allows the declaration of data item sensitivity (ie data items from the schema known to the run-unit), synonyms (ie new names for data items in the schema), and characteristics (ie all data items are assumed character but length, scaling and output formats are definable).
9. Write semantic routines and modify SKELETON to generate an object version of the sub-schema using the schema and sub-schema inter-relationships.
10. Define in BNF a non-procedural language which allows access to a database defined with a user sub-schema.
11. Write semantic routines and modify SKELETON to generate DML commands, from the non-procedural language, which can be

input to DMLV1.

12. Implement a GDBMS to support the concurrent execution of multiple run-units using the non-procedural language statements defined in (10).

6.3 Summary of Work Completed

The schema DDL is defined in BNF and the data structure to contain the object schema is defined. This data structure is hand coded into DDLV1 and a schema description in the schema DDL is written. This description has been processed by DDLV1 and loaded into SCHEMA1. DDLV1 checks for syntax errors in the DDL and for missing identifiers, but many of the possible inconsistencies which can occur in the definition are not detected. Justification for this is the fact that extensive validity checking is a very tedious function and the coding must be redone when DMLV1 is written. For this reason basically correct input is assumed for the first schema definition. When the DML commands are implemented most of the checking is simplified through the use of DML commands. The DML is defined in BNF. Throughout the course of the thesis the CODASYL definition of the DML commands underwent two major revisions. The present description appears to be almost the final version. The BNF description applies to the current description.

Appendix B contain the BNF descriptions of the schema DDL. The data structure defined for the schema database and the associated DDL statements are included in Appendix B.

The database is treated as a virtual address space. All data records are referenced by relative byte address (RBA). I/O routines convert the RBA into a block and displacement address. The file is accessed using the Basic Direct Access Method which allows direct access to a relative block number. To simplify space management all blocks are fixed in length. Logical records may be any length, including greater than the physical block length.

The database key is a sequential number allocated to records as they are loaded. A table, maintained with the database schema, relates database keys to RBA. This allows records to be moved within the database with minimal difficulty. Only the DATABASE KEY-RBA table need

be modified. All requests for records must be resolvable to a list of database keys.

Indexing within the database is done using an inverted list of database keys. No facility is provided for use of the indexed access methods of the operating system.

A debugging program has been written which processes the schema database and recreates the DDL statements used to create the object schema. This program will eventually be modified to produce a graphic view of the data structures defined by the schema. An online version of this program, which accesses user databases, is very useful to the Data Base Administrator (DBA) attempting to define optimal SET OCCURRENCE SELECTION criteria.

6.4 Implementation Considerations

The CODASYL specification is designed to be implemented on current hardware using current software. A primary intent of the task group is the specification of a system which can now be accepted, implemented, and provide a standard for data definition and access. One topic that the report fails to address in sufficient detail, to direct implementors, is the area of data integrity.

The integrity of data can be affected directly through data loss or more subtly through the interaction of concurrently executing run-units. Data can be lost:

1. as the result of an external destruction of some or all of the data on a volume (I/O error on volume), or
2. as the result of an externally caused termination of a run-unit executing an update operation, or
3. as the result of errors in the input data or run-unit procedures, which cause incorrect update operations to be executed against the database.

The GDBMS mechanism for recovering from data loss involves the periodic dumping of the entire database to backup storage volumes and the maintenance of a run-time log of all system activities. The log contains before and after images of each database record involved in an update operation. In a data communications environment all input transactions are also logged. Checkpoint information is logged to

allow warm starting of run-units active at the time of a system failure. The run-units are warm started at a point in time prior to any data loss. The before images allow the database to be restructured.

Present hardware is always able to complete active I/O operations before the system is totally quiesced. This implies that abnormal termination of a run-unit or the GDBMS will not cause a partially updated physical record to be stored in the database. Therefore, I/O errors should not be generated as a result of a system failure.

Concurrently executing run-units can potentially cause the integrity of data to be violated. There are two situations in which integrity may be affected:

1. Retrieval with concurrent update and
2. Update with concurrent update.

We define two types of integrity violations: logical and physical. Logical violations occur when a run-unit A processes a set over time and as yet unprocessed records are modified by a concurrent run-unit B. The information derived by A can be said to be logically incorrect, since it does not reflect the status of the set at any instant in time. Suppose run-unit A is accessing a set containing production information for individual employees and producing a report showing the productivity to date of selected employees. Run-unit B is concurrently active in update mode modifying the production data with quality control reject information. If run-unit A is producing salary and promotion recommendations based on the information in the set, then employees whose records have not been affected by run-unit B have an unfair advantage.

Physical integrity violations may occur when the database is being accessed concurrently by more than one run-unit, one or more of which are update processes. Suppose run-unit A retrieves record R1 and modifies it by adding $100 to the salary. After the retrieval but before the record is rewritten run-unit B retrieves the same record R1 and decreases the salary by $50. Before run-unit B rewrites the record run-unit A rewrites the updated R1. When run-unit B subsequently rewrites R1 the salary value has been decremented by $50 rather than the correct $50 increment.

Both of these integrity situations occur frequently when multi-user access to a database is allowed. The simplest and least acceptable solution to both integrity problems is to disallow the simultaneous execution of an update run-unit with any other run-unit. The CODASYL GDBMS provides for the specification of exclusive control over a number of sets or areas. Exclusive control can significantly reduce the availability of data when sets are stored over several areas in the database. A single run-unit may effectively block access to the entire database.

To avoid the deadlock situation where two run-units request exclusive control of different areas and then one requests access to an area under exclusive control of another run-unit, all usage mode requests must be given as the first commands of the run-unit.

A better solution to the logical integrity violation problem is possible if a time stamp is included with each physical record. When a run-unit begins execution it specifies whether all records retrieved should reflect the state at run-unit initiation or should be the most current versions. If logical integrity is not a concern then only physical integrity must be considered.

Records inserted or deleted are marked with an insert/delete time stamp. Records added after run-unit initiation will not be retrieved. Records deleted after run-unit initiation will be retrieved. Record modification is treated as a delete, insert sequence.

A garbage collection run-unit, which executes concurrently with user run-units, physically removes deleted records from set occurrences if no active run-unit sensitive to the set, was initiated prior to the time stamp in the deleted record.

There are several advantages to this scheme in addition to the prevention of integrity violations. The garbage collection run-unit executes as a low priority task using cpu cycles not required by user run-units. Maintenance of set relationships may be a time consuming process when a record participates in many sets. Productive work is not delayed to affect the set maintenance required for deleted records. Database recovery is speeded since the time stamp provides an easy mechanism to recognize records current at a time prior to system failure. A disadvantage of this scheme is the overhead introduced in

the processing of deleted records. In a dynamic environment a significant percentage of the database may consist of logically deleted records. Development of hardware or remotely executed software to recognize and not retrieve deleted records would reduce this problem.

Physical integrity is more difficult to ensure. The protected usage mode of the CODASYL GDBMS prevents two run-units from being active with update rights to the same area. Violations of the data of a read-only run-unit by an update run-unit are avoided when the read-only run-unit requests that all records be current to the start time of the run-unit.

Read-update physical-integrity violations can occur if time stamping is not used when an update run-unit modifies the linkages between records being processed by a read run-unit. A very blatant example occurs when read-only run-unit A is processing a set in sequence and update run-unit B issues a non-local ORDER command for the set. Suddenly the sequence in which run-unit A gets his records is changed. By default the GDBMS assumes a run-unit will retrieve records current to its initiation time.

The monitor facility of the CODASYL GDBMS is used to allow recognition of a potential physical integrity violation. In our example when run-unit A issues a FIND for R1, then R1 is monitored for A. A subsequent FIND by B for R1 causes R1 to become monitored for B as well. When A issues a MODIFY for R1 the GDBMS flags all monitored occurrences of A as non-updatable. The subsequent MODIFY by B causes a monitor exception to be returned in the status indicatior of B and R1 is not updated. Run-unit B must re-get R1 and redo the update operation. This is unacceptable since it requires that all updating run-units include code to recover from monitor exceptions and it may not be possible to recover. If the update operation performed by B is the result of many input transactions which are no longer available it is not possible for B to redo the update. It may be necessary for the operations of B to be backed out and the process restarted using the logged information. Since the log is a sequential history of events this requires that the GDBMS be quiesced.

If concurrent updating is allowed then control of integrity

becomes the responsibility of the data base administrator (DBA). He must ensure that applications are not designed in which integrity problems can occur. Where possible he will encourage the design of transaction processors in which all database updating is done in a single run-unit. The use of actual and virtual fields increases the probability of integrity violations since more records must be simultaneously monitored or controlled.

In non-host languages systems the DBA will encourage the use of temporary areas in which no integrity violation can exist. In a temporary area the user may update and retrieve information created from data in the whole database secure in the knowledge that no other run-unit may access his area.

The area of data integrity is a major unexplored area in the field of database technology. Most implemented systems use the exclusive control method to ensure integrity. If we hope to utilize large integrated databases in a multi-user distributed environment this problem must be solved.

6.5 BIBLIOGRAPHY

McKEEMAN, W.M., HORNING, J. J., WORTMAN, D.B.,
'A Compiler Generator'
pub. Prentice-Hall, 1970.

7.1 Introduction

We have looked at a rationale for the development of database, database structures, some terminology, database description and manipulation languages and the implementation of a Generalized Data Base Management System. It seems relevent at this time to look to the future. First, we look at some of the future requirements for the use of computer systems and particularly the use of database in these systems. Second, we look at the new hardware and software which will be developed to support these systems.

On-line access to databases will certainly be the norm. Future systems rely on the availablity of on-line access. In addition the development of and implementation of non-procedural languages will make the information in databases available to a larger percentage of the population.

7.2 FUTURE SYSTEMS

Management Information Systems

Future systems will use real-time data collection and terminal enquiry and update facilities. Sensor-based distributed computing systems will collect both analog and digital inputs at the unit (operational) level for transmission to plant computers. At the unit level mini-computers monitor activities, supervise the detailing of activities and directly control minute by minute operations. At the plant level the input from operations is used to maintain databases used in optimizing production, scheduling and accounting. At the corporate level plant databases are manipulated to provide data for simulation, forecasting and budgeting.

In this environment we can expect to see a significantly greater sharing of data by users with a right to know than is presently the case. A fully integrated data collection, database maintenance and database enquiry system will make management more of a science and less of an art. As the jolly green giant of computers is fond of saying "not just data but reality" will be available to management.

Integrated Financial Accounting

The cashless and chequeless society has been a dream of financial institutions for several years. The availability of suitable GDBMS and data-entry hardware is all that holds up the implementation. For financial institutions an on-line banking facility eliminates the float of outstanding cheques; that is, money which has been paid to an account but not received from the buyer. For commercial sellers a sale produces immediate revenue and reduces the amount of capital tied up in the non-productive float. For an individual it may produce instant bankruptcy. Those of us who live from month to month on the delay between buying and paying for an item will suddenly discover that in one month we must pay two month's bills.

There are, however, several advantages to be derived for an individual or a company. Each transaction is recorded and can therefore be retrieved at some later date. Expense recording is automatic. No more hunting for bills to make up travel expenses or to pay income tax. Budgeting is less of a problem. Fixed expenses can automatically be applied and at any time present worth can be calculated. The full usefulness, to an individual, of this system awaits the availability of terminal access to the database information for which the individual has a right of access.

From a corporate point of view the monthly batch billing operations are no longer necessary. The billing process is a machine to machine operation taking place at the time of transaction processing.

Information Dissemination

An area where vast amounts of money is now being spent is advertising. How effective is the present communication of advertising information? Advertising attempts to explcit an existing market for old products and to create markets for new products. A database detailing available products appeals to those buyers who have a need and require a source. Eatons catalogue provides ordering information and prices. Pictures and text are used to market the product. This form of advertising is static. It reflects information at the date of

publication. Market-creating advertising requires that the advertiser get the attention of potential buyers. For this function the cataloque is not really effective. Interruption advertising is used on television, radio and to a lesser extent in newspapers and magazines to present an advertisement to potential buyers. The medium gets the users attention and the advertiser attempts to sell. A database of product information, in computer readable form, is a well indexed cataloque. It is dynamic, reflecting the current description, pricing and availability of products. Further, all sellers of a product are grouped. It is not necessary to read several catalogues to do comparative shopping. Video-tape and graphics presentations provide pictorial descriptions of products. Promotional advertising will be done using the interruption technique. For the privilege of interrupting a session the advertiser will pay part of the usage costs. An on-line order entry system, connected to the product database, reduces the need for retail outlets. The cost of advertising is borne by the seller and interested buyers. For each access to the data the user and seller are charged a fee.

Another commodity which relies on advertising to market its product is the entertainment industry. Once again a database of available entertainment allows an interested individual to quickly locate an item of interest to him. Forms of entertainment with limited interest can be advertised in this way. Only those persons interested will be notified. If the enquiry facility exists in the home then, we might define criteria specifying products, entertainment, and news information about which we wish to be automatically informed.

When data is stored in a database it can be interrelated and extracted as meaningful information. When the data in newspapers, magazines and books is available in a database it is possible to locate specific information on selected topics. Each individual has access to an entire library with an indexing scheme designed for his needs. It is possible to retrieve other articles referenced or related to the subject of interest. In this way more complete information is available to the interested user. This facility reduces the tedious process of literature searching which each researcher must do when starting a new project.

Personal communications are improved using this facility. Messages which are presently conveyed over telephone calls can now be sent in hard copy, not necessarily paper, form. The message can be sent and received at a time convenient to the sender and receiver. The use of video-tape will personalize the message switching operation. The Postal Service in the U.S. presently offers a facsimile mail service. Mail is sent from Xerox-Xerox via the telephone lines. This service offers guaranteed delivery between any large city in the U.S. within four hours. The cost of this service is still excessive partly because of the human intervention involved in carrying the mail from the sender to the post office and from the post office to the receiver. Direct linkups from each individual and company to a postal database eliminates the need for personal delivery and the production of unnecessary paper.

Personal Databases

In addition to the communication advantage for individual use, these systems will provide mechanisms for the storage of personal information. We will be reminded of special dates (eg anniversaries) and be provided with mechanisms for the storage and cross-referencing of personal information required to be secure but available (eg diaries).

Computer Assisted Instruction (CAI)

One result of the widespread implementation of these GDBMS will be a significant increase in available leisure time. The GDBMS will be obligated to provide facilities, in addition to those already mentioned, to teach and entertain.

While the use of CAI will not likely replace educational institutions as the primary source of basic education, it should supplement them in providing specialized training in selected areas.

A database of games will also be available. The use of an integrated facility provides for the introduction of competition into the games. Each individual may, if he wishes, compete against the results of all other individuals.

Paper Shortages

With the increasing concern for and expense of non-renewable and non-reusable resources, the use of on-line information dissemenation and financial accounting systems using video terminals will become morally and financially more justified.

7.3 New Hardware and Software Requirements

Some of the applications mentioned previously could be implemented with existing technology but it is not yet cost effective to do so. Further advances in technology can be expected to provide cost justification for the implementation of these systems.

The introduction of a large number of users into a GDBMS requires the availability of sufficient internal storage to store current records. Recent advances in bubble memory, super-cool memories and field-effect transistor technology, point towards a continuing decline in the price and and an increase in the speed and size of directly addressable memories. Virtual memory operating systems logically expand memory through the use of a hierarchy of storage classes.

The storage of large databases requires the availability of cheap large capacity mass storage devices with a random access capability. Recent advances indicate that rapid access to moveable head disk devices containing 10**9 bytes of data is now feasible. The cost per byte of data storage on these devices is .0004 cents for an average access time per record of 27ms. (Computerworld, October 31, 1973. page 1) Other techniques requiring minor modification to these devices seem likely to double and even quadruple the capacity with very little increase in cost. The most obvious of these techniques involves the storage and retrieval of information in parallel across the plate to allow use of a greater surface area without increasing data density.

New technology has produced random access devices with slower access times but much larger storage capacities. Present drives have capacities of more than 10**12 bytes (Datamation, Oct 1973. p 52). Laser Computer Corp, optimistically predicts 10**39 byte capacities by 1977. More realistic projections of 10**17 bytes in 1990 and 10**21 by the year 2000 are being made, based on past experience. These devices require between 10 and 100 seconds to access a block of data. While this is slow in comparison to fixed head devices, it may be quite acceptable in archival storage situations. Data migrates from slow to faster devices depending on the frequency of its access. Special purpose CPU's are used to move data between storage classes. These

operations can proceed asynchronously depending on the number of data paths available. Ampex (Datamation, Oct 1973. p54) has presently available a dismountable hypertape with a 362*10**12 byte capacity and access time of less than 100 seconds. The use of devices such as this make available a virtually unlimited amount of data storage. Access may be restricted for data not stored on an on-line device.

Virtual Operating Systems are self-tuning, in that they assign paged data to storage classes based on its frequency of access. An obvious extension to this requires that the virtual operating system be extended to allow truly virtual addressing. (Present systems allow a maximum of 16,777,216 byte address space.) The database will be considered as permanently resident and most of it will be paged out because of inactivity. Thus data addressing is done by relative byte addressing (RBA) and handled in the operating system. This would require improvements in system recovery capabilities, already a prime requirement for future systems.

Distributed computer networks accessing common databases present several problems in controlling and optimizing database access. A possible solution involves the development of a specialized computing facility. A database computer system designed to contain the GDBMS will schedule non-conflicting accesses to the database. Maintenance of the buffer pool is simplified when a single processor controls data access. Device controllers, with the ability to be programmed, accept requests for data from relative byte addresses. The conversion to actual addresses is done in the controller. This provides a level of device independence required for this paging environment and not easily obtainable in present systems. The controllers optimize search requests to each device and support parallel searching on all devices under their control.

Specification of well defined interface requirements will be necessary in order to allow the GDBMS processor and device controllers to communicate with all types of hardware. The GDBMS must return data to the host processor in the form requested. This may require that extensive data conversions be done on stored data. Special purpose processors designed to perform this function will be designed and integrated with the GDBMS and device controller processors. This may

be a suitable operation to have done in a device ccntrcller.

Recent advances in field-effect transistor technology and the development of low cost power supplies point the way towards inexpensive and reliable video display terminals. Over the last two years; for example, the prices have decreased by about 500%. Further developments and mass production of these devices can be expected to provide even greater reductions in price. The availability of television sets in almost every hcme provides a readily available display terminal. Although the resolution of these terminals is less than desirable, manufacturers have developed keyboards and interfaces that allcw the television to be used as a transmitter of data as well as a receiver.

Caxton Foster (Communications of the ACM, Vol 15, #7) predicts that by 1975 micro-computers will require cnly one chip and cost less than $100. Further by 1980 the cost will be in the range $1-$10. This is for ccmputers with 65k words of 32 bit memory and an instruction rate of 2*10**6 instructions per second. These micro-computers will be used for a single specialized function and will not have an I/O capability. One possible application, with great potential for our terminal requirements, is the development of smart terminals. The memory and CPU power of the micro-computers are used to store and refresh the display on the terminal. Error detection and some correction is done by the terminal. Much less data will need to be transmitted to the terminal, since only changes in the screen are required. Foster predicts that the present television transmission requirements could be handled by a single channel. Thus the remaining frequencies are available for other applicaticns. Terminals will become part time televisions.

The use of satellites for data transmission means that data can be transmitted thousands of miles for little more than the cost of a hundred mile transmission. In addition the use of electronic switching equipment in the toll central offices of communication companies promises to reduce the switching costs for all tolls. These two developments open the door tc increased use of data transmission. Canada is in the unique position of having excess capacity in her satellites waiting to be used. The availability of cable television

hookups in many homes provides a convenient and already installed linkup between a computer system, its database and each individual. Software developments to account for usage and the participation of financial and commercial institutions can be expected to accelerate the widespread use of database technology.

Relatively inexpensive data collection systems now exist. These are primarily used by retailers and credit card companies for data enquiry. It is expected that they will soon be expanded to include data inputting capabilities. In Canada the Bank of Montreal is installing a country wide terminal oriented system which will introduce on-line banking to each branch of the bank. Simpson-sears has an order entry system for catalogue shopping installed in Toronto using touch-tone telephones. Certain users are testing out this facility and if successful its availability will be expanded to all users of touch-tone phones. IBM has a System 7 computer system which can handle up to 255 analog and digital inputs. The processor is compatible with the larger 360 and 370 series and software exists to support cpu-cpu communication. The System 7 is used now in process control applications particularly in the communication and petro-chemical industries. In most applications it is connected directly to a host 370 and thus to the database of the 370.

Other companies such as DEC, Data General and a company making a product known as the Naked Mini have produced special purpose mini-computers which can be connected tc larger systems and their associated databases. These systems are becoming very cost effective as the cost of production control and data collection increases with rising wage costs.

The use of micro-film and videotape provide existing means of entering data into a database (IBM System Journal, Vol 11, 1972. p 56). Future systems will provide for the conversion of micro-film and the printed word into machine readable form. Voice input is another requirement for future systems and work now being done indicates that this is quite possible, particularly in regions where a high degree of speech consistency exists (Telephony, reference not available).

The integrity and privacy of user data must be guaranteed in such general use systems. Present work has not yet solved this problem in a

manner which will allow the type of response required in this environment. Further research in this area will develop systems which can satisfy the requirements of users.

A major task for future designers of general use GDBMS lies in the education of users. Very high-level languages will be developed to allow the user to make requests in a form suited to his experience. Help facilities will be developed to allow the user to be guided along as he learns to use the system and take advantage of its facilities. The first CAI courses will be instruction on the use of the system.

Considerable research remains to be done to develop GDBMS's which can satisfy the requirements of future systems. Increasing pressure from non-technical users for systems responsive to their needs is enhancing the probability that the near future will see truly GDBMS's used in the areas previously mentioned. It is an exciting area providing an opportunity for research in a wide range of computer related areas.

BIBLIOGRAPHY

HOUSTON, G.B., Trillion Bit Memories
Datamation, Vol 19, #10 1973 pp 52-57.

COMPUTERWORLD, STC Announces 3330 Compatibles
October 31, 1973 page 1.

FOSTER, C., A View of Computer Architecture
Communications of the ACM
Vol 15, # 7, pp 557-565.

SAMMET, J., Programming Languages: History & Future
Communications of the ACM
Vol 15, # 7, pp 601-610.

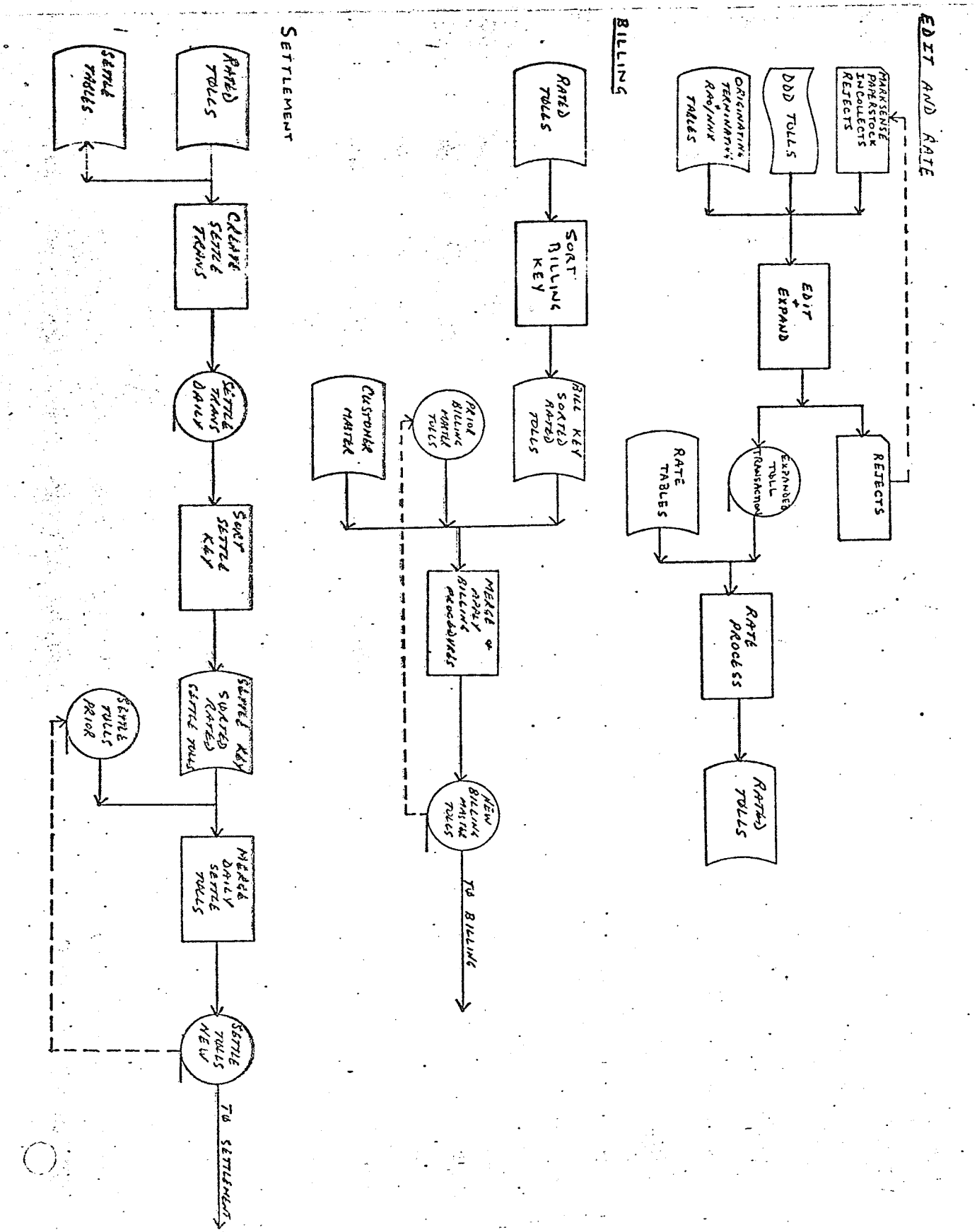
APPENDIX A

In this section we look at a transaction driven system much like that presently being used by all the large telephone companies in Canada. At present the system is batch oriented. There are three major reasons for this:

1. the files are very large. Bell Canada has approximately 13,000,000 transactions on each of two files at any given time,
2. the same transactions are used in two toll processing sub-system
in completely different sequences,
3. the system is implemented on second generation hardware which does not support large volumes on random access devices.

The batch processing system is outlined here with a short description of each processing function. Following this a database organization is outlined which supports the same functions and provides considerable flexibility in the implementation of new applications.

FLOWCHART OF THE TOLL SUB-SYSTEMS



EDIT AND EXPAND PROCESS:

This process handles all input toll transactions. Extensive validation is done. The toll input transaction is expanded into the toll master transaction by the addition of information from the Originating and Terminating Master Files and the RAO/NNX table. Information required for the billing and settlement functions is carried with each toll master record. Each toll input record is assigned a unique serial number. The input toll is a source document and must be retained in a library for 6 months. The tolls are pre-serialized with a surface-stamped serial number. All rejected tolls are pulled using this serial number. Rejected tolls are manually corrected and re-submitted.

INPUTS:

Marksense toll cards are prepared by telephone operators for most calls which are not dialed directly. These cards are pre-processed on a 519 Marksense reader punch.

Paperstock toll cards are handwritten by telephone operators in manual offices. These must be keypunched before submission.

Incollect toll cards are received from other companies. They are calls originated in another company's area but billed to this company.

DDD tolls are received on a 20-channel paper tape. This tape is punched in the Central Office by paper tape punches hooked directly into the switching equipment. No prior processing takes place for these tolls.

The Terminating Point File is an index sequential file containing all points in North America to which it is possible to dial. The place name, a vertical and a horizontal coordinate, and settlement information are stored with each terminating point record.

The Originating Point File is an index sequential file containing all points from which a user in this telephone company may place a call. The place name, a vertical and a horizontal coordinate, settlement and billing information are stored for each record.

The RAO/NNX table is a binary coded table which is maintained in core and contains billing information for each originating NNX and RAO from which a user of this telephone company may place a call.

OUTPUTS:

The Toll Master File contains the expanded tcll record for each valid input toll transaction.

Three report files containing toll reject infcrmation and Central Office statistics are produced.

TOLL RATING PROCESS:

This process calculates the cost of each toll call using information in the Toll Master Record and the Toll Rate Table.

A reasonability check is made of each toll and tolls exceeding 100 minutes in length or $100 in value are reported.

Operator-rated calls which differ from the computer rating by more than 10% are reported. These tolls are pulled from the library and verified. Statistics are maintained of each operator's misrated calls.

A statistical sampling of the tolls is reported, based on parameters specified by company auditors and varied over time.

INPUTS:

The Toll Master File output from the Edit and Expand process.

The Toll Rate Table which is read into core and processed using date, time, call type and class, and V & H information as indices into the table. This table is very dynamic subject to a minor change at least once per month and a major change once or twice a year. Occasionally the structure of the table changes necessitating changes to the rating process.

OUTPUTS:

An updated Toll Master Transaction File which contains the rating information for each toll.

MERGE AND APPLY BILLING PROCEDURES:

This is primarily a merge function. The Unbilled Toll Master File is maintained in billing-number sequence. Each billing cycle the tolls for that cycle are removed from the file and the remaining file becomes the Unbilled Toll Master File.

Each Customer Master Record is matched to the Unbilled Toll File. When tolls are found for which no master exists they are reported. Each customer master contains a maximum toll value. When the accumulated toll value exceeds this amount by more than 10% this is reported. Final customers (ie customers who have asked to have their phones removed) are reported for all tolls and a special indication is given when tolls are found with a date after the customers end of service date.

Tolls on the input Toll Master File which are not billed to customers in this company are not written to the unbilled file.

INPUTS:

Toll Master File output from the Toll Rating process in sequence by date of toll within billing information.

OUTPUTS:

New Unbilled Toll File in sequence by date within billing information.

Reports showing unbilled toll values by billing cycle and statistics on toll quantities and dollar value.

SETTLEMENT PROCESSES:

An operating company owns and operates the central office. A billing company handles the billing and collection of toll revenues. Each company is entitled to a percentage of the call value for services performed. In addition, when a toll call is handled by more than one operating company, each company is entitled to a share of the revenues. The determination of these percentages and the allocation of revenues is a function of the Settlement Process.

CREATE SETTLEMENT RECORDS:

Records are assigned settlement and traffic breakdown codes. Based on the to/from information. A breakdown of the portion of the call revenue due each company on the call route is determined. The settlement records include only marksense and DDd tolls. Paperstock are not settled since they are intra-company calls. Incollects are received from other billing companies and have already been settled.

INPUTS:

The Toll Master File output from the rating process and in sequence of date within terminating point within originating point.

OUTPUTS:

Toll Settlement File with settlement and traffic breakdown information.

Reporting of revenue breakdown and statistics on call volumes by quantity and time.

SETTLEMENT MERGE:

The Toll Settlement File is accumulated after each rating process. The file is maintained after each rating process with new settlement tolls being added in sequence to the existing accumulation. Each settlement period the settlement reports are run and all tolls involved are removed from the file.

INPUTS:

Toll Settlement File new transactions which have been created from the Create program.

OUTPUTS:

New Settlement Master File containing only tolls after the toll settlement period.

Outcollect file in toll image form (ie 80 character record of information required to bill and not including place names etc.).

Report showing the Settlement Revenue Breakdowns by operating company.

Wats Billing Information and statements for customers in sequence by Wats associated telephone number.

Problems With The Batch System

The major problem with the batch system is the duplication of data. With such a large volume of data it is not possible to sort the files when a billing or settlement process is to run. For this reason two copies of the file are maintained in appropriate sequences.

As a result of the data duplication several problems arise. It is difficult to balance the toll quantities on the two files since they are purged at different dates. The auditors and operations staff have difficulty ensuring that input tolls are not missed from one sub-system. This is particularly critical when a rerun must be done because of a hardware or software error.

The existing system is strictly sequential. Enquiries against particular tolls or telephone numbers require a sequential scan of the file.

Toll edit reject tolls cannot be re-inserted into the billing and settlement files quickly. The cost of running the whole system for a small number of tolls is prohibitive. Because of the sequential nature of the system every toll on file must be passed even when inputting only a small number of new tolls.

The Originating and Terminating Point files are frequently updated to reflect information received from other companies correcting previous errors. Changes to these files cannot be reflected in the batched toll files after the edit stage without a complete rescan of the files. Frequently this causes problems in toll correction when tolls are returned because of a customer complaint.

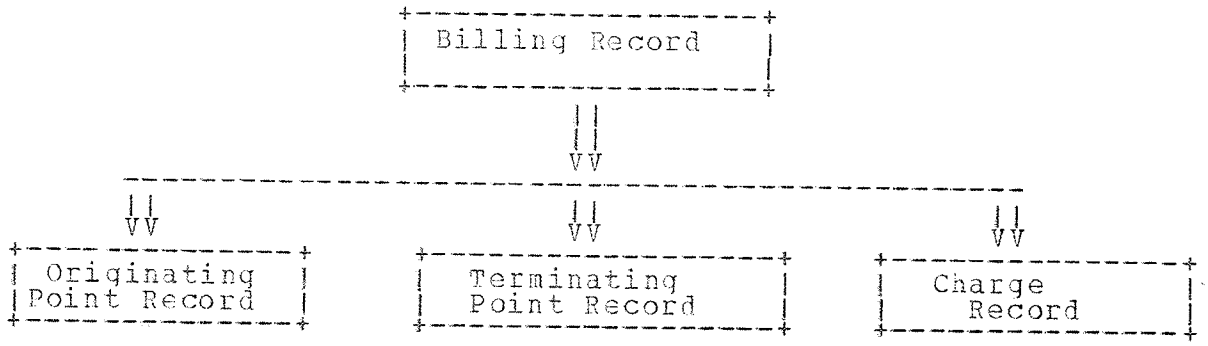
New applications which require that the toll files be processed in different sequences are expensive to operate. With an increasing need for statistics as justification for rate applications, the toll files are being processed to produce breakdowns on many catagories not anticipated when the systems were designed. Future requirements dictate that the entire file must be used in producing statistical analysis, rather than a selected sampling as is now done. The problem of sorting such a volume of records into many and unforseen sequences requires that a solution be found which allows dynamic enquiries without the necessity of constantly sorting the files. Such a facililty may be provided by a database.

A Database Solution

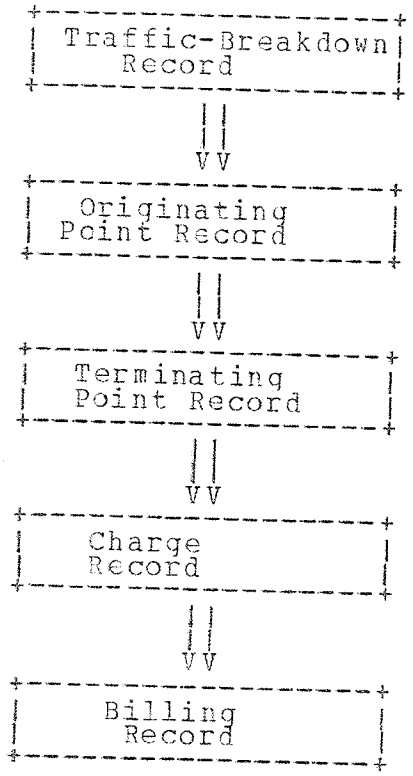
Using the facilities of the CODASYL Data Description Language (DDL) it is possible to design a schema which encorporates into a single occurrence of the data, the structures required to satisfy present and and future requirements.

If we look at the existing applications, two major hierarchical data structures are being utilized. These two structures relate the same information but in different relationships. In the toll billing sub-system records are related by billing-number. In the settlement sub-system records are related by traffic-breakdown, originating point, terminating-point and billing-number.

Billing Data Structure

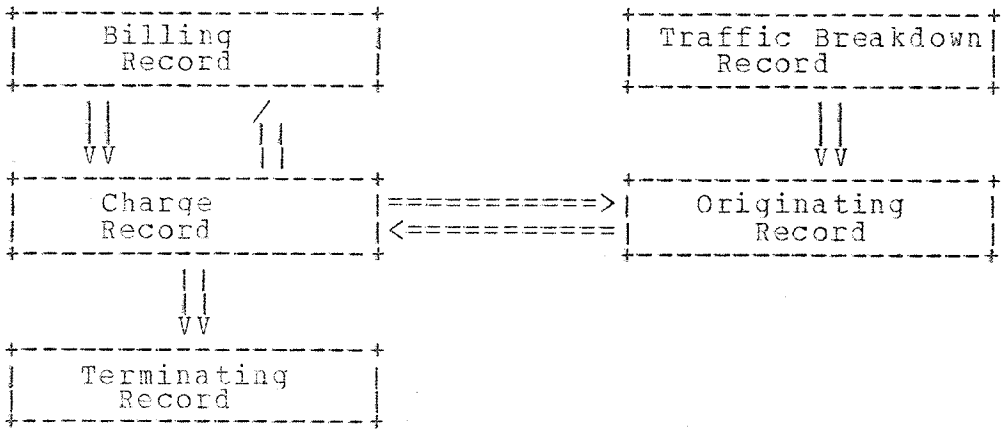


SETTLEMENT DATA STRUCTURE



The combination of these two data structures into a single structure produces a network structure which can be implemented and used in a database environment.

BILLING AND SETTLEMENT DATA STRUCTURE



In the batch system each record of each file contains an occurrence of each of the records defined in the database structure. Some of the database records occur redundantly through many of the batch system records. Assuming an average file size of 13 * 10**6 records on each of the transaction files and a record length (excluding space allocated for expansion) of 137 bytes, the sequential and index sequential files occupy 17.83 * 10**8 bytes of external storage.

In order to calculate the space required for the database data structure it is convenient to define the following notation:

N1 = a unique occurrence of the Billing Record.

n1 = expected number of occurrences of N1 in the database, approximately 2,500,000.

l1 = length of N1 in bytes excluding control information. l1 = 14.

N2 = a unique occurrence of the Originating Record.

n2 = expected number of occurrences of N2 in the database, number of unique originating points in the Bell System. n2 = 4,000.

l2 = length of N2 in bytes excluding control Information. l2 = 25.

N3 = a unique occurrence of the Terminating Record.

n3 = expected number of occurrences of N3 in the database. n3 = number of unique dialable places in North America and the world, = 40,000.

l3 = length of N3 in bytes excluding control information. l3 = 27.

N4 = a unique occurrence of the Traffic Breakdown Record.

n4 = expected number of occurrences of N4 in the database. n4 = average number of other companies settled with in a normal month = 8.

l4 = length of N4 in bytes excluding control information. l4 = 1.

N5 = an occurrence of the Charge Record.

n5 = expected number of occurrences of N5 in the database. n5 = 1 per toll call made = 13,000,000.

15 = length of N5 in bytes excluding control information. 15 = 34.

The total space required to contain the database using the network structure specified here, is the sum of the products of the data record lengths and their respective expected frequencies plus the pointer overhead. The data storage requirement is 7.92 * 10**8 bytes. The pointer overhead requires .52 * 10**8 bytes. External storage required to store the database is 8.44 * 10**8 bytes. This is just under 50% of the batch total. A saving of 9.39 * 10**8 bytes. While the space saved is impressive, similar savings could also be obtained by other means. For example using an assembler routine to code the data items and eliminate null fields would likely produce a 30-40% saving in storage required. More importantly the data is stored non-redundantly in the database. Toll quantities can be controlled by operations and audit staff. A single processing of the tolls is sufficient for both sub-systems. In addition all other applications using the toll information use exactly the same copy of the data.

The Originating and Terminating Point records can be directly retrieved and updated. An update is immediately reflected in all tolls referencing these records.

Settlement and Billing tolls are available for random or sequential processing based on their defined key fields.

New inter-record structures can be added to the database without impacting existing applications.

Future savings and flexibility can be anticipated as the Billing Master and Equipment master files are integrated with the toll database. Billing and equipment files duplicate all billing information. The integrated database allows customer service representatives to access all information about a particular customer. Management can expect to have answers to many and varied requests. What is the relationship between equipment rental and toll revenue? What is the projected toll revenues for this month in a particular or all billing cycles? What is the percentage of bad debt ratio caused by tolls? What would toll revenues have been if a different toll rating algorithm had been used? As new questions are developed new set relationships may have to be developed. These should be evaluated as

to their impact on the total system. It may be necessary to implement non-procedural facilities to allow the creation of temporary areas to contain data relevant for dynamic requests. In this way the database remains unchanged and seldom used set relationships need not be maintained in the production database.

A schema DDL to specify this database is included below. Many variations of this schema can be envisioned and only an ongoing measurement of the database performance will finally determine how records and sets should be declared.

```
    /* a schema DDL to support the toll systems  */
    /* as per the data structure given previously */
    /*  random and sequental access to data in each sub-system */
    /*  is provided for.                                       */
    /* In addition provisions are included in anticipation     */
    /*  of future requirements including the integration of    */
    /*  billing, equipment and toll data                       */

SCHEMA NAME telephone-schema
  ;PRIVACY PROCEDURE schema-access.

AREA NAME settlement-area
                /* in all cases check for authority on open */
  ; ON OPEN FOR EXCLUSIVE PROTECTED CALL enqueue-area.

     /* data for other billing sub-systems also stored here */
AREA NAME billing-area
  ; ON OPEN FOR EXCLUSIVE PROTECTED CALL enqueue-area.

AREA NAME terminating-area
  ; ON OPEN FOR EXCLUSIVE PROTECTED CALL enqueue-area.

RECORD NAME billing-record
  ; LOCATION VIA billing-set
  ; WITHIN billing-area.

 01  billing-key.
     05  cycle        TYPE      BINARY 1.
     05  exch         TYPE      BINARY 2.
     05  nnx          TYPE      BINARY 2.
     05  line         TYPE      BINARY 2.
     05  prty-rngdn   TYPE      CHARACTER 3.
     05  rao          TYPE      BINARY 2
     05  npa          TYPE      BINARY 2.
/* other information is included in this record
    but it is not of interest to this application */

RECORD NAME traffic-breakdown-record
  ; LOCATION CALC USING traffic-breakdown-code
  ; WITHIN settlement-area.

 01  t-b-code     TYPE    BIT 8.

RECORD NAME originating-record
  ; LOCATION CALC orig-hash USING  npa, nnx
       DUPLICATES NOT ALLOWED
  ; WITHIN settlement-area.

 01  orig-key.
     05  npa      TYPE     BIT  16.
     05  nnx      TYPE     BIT  16.
 01  orig-tc      TYPE     BIT  16.
 01  toll-state-sub  TYPE BIT 8.
 01  toll-center-sub TYPE    BIT 8.
 01  switchboard     TYPE    BIT 8.
 01  place           TYPE    CHARACTER 10.
 01  v-coord         TYPE    BIT 24.
 01  h-coord         TYPE    BIT 24.


RECORD NAME terminating-record
  ; LOCATION CALC term-hash USING npa, nnx
  ; WITHIN terminating-area.

 01  term-key.
     05  npa      TYPE     BIT 16.
     05  nnx      TYPE     BIT 16.
 01  tc-code      TYPE     BIT  16.
 01  term-place   TYPE     CHARACTER 14.
 01  other-place  TYPE     BIT 1.
 01  settlement   TYPE     BIT 1.
 01  other-line   TYPE     BIT 1.
```

```
01  v-coord      TYPE     BIT 24.
01  h-coord      TYPE     BIT 24.

RECORD NAME charge-record
    ; LOCATION MODE DIRECT charge-key
    ; WITHIN charge-area.

01  date         TYPE     BIT 24.
01  time-day     TYPE     DECIMAL 2 1.
01  discount1    TYPE     CHARACTER 2.
01  discount2    TYPE     CHARACTER 2.
01  charges      TYPE     DECIMAL 3 2   OCCURS 4.
01  surcharge    TYPE     DECIMAL 0 3.
01  minutes      TYPE     BIT 8.
01  minutes-zoned TYPE    CHARACTER 2.
01  serial-no    TYPE     BIT 16     OCCURS 2.
01  orig-line    TYPE     BIT 16.
01  term-no.
    05  npa       TYPE     BIT 16.
    05  nnx       TYPE     BIT 16.
    05  line      TYPE     BIT 16.
01  flags.
   05  day-indic           TYPE     BIT 1
         ACTUAL RESULT  calc-day.
    05  record-type        TYPE     BIT 1.
    05  tcll-type          TYPE     BIT 1.
    05  class              TYPE     BIT 1
         ACTUAL RESULT  rating-process.
    05  special-condition TYPE   BIT 2.
    05  time-charges       TYPE     BIT 1.
    05  credit-type        TYPE     BIT 4
         ACTUAL RESULT  rating-process.
    05  min-credited       TYPE     BIT 8
         ACTUAL RESULT  rating-process.
    05  misc-codes         TYPE     BIT 2.
    05  ddd-indic          TYPE     BIT 2.
    05  billable-code      TYPE     BIT 1
         ACTUAL RESULT  rating-process.
    05  toll-reject        TYPE     BIT 1.
    05  tcll-fixed         TYPE     BIT 1.

SET NAME  billing-set
    ;MODE CHAIN
    ;ORDER SORTED WITHIN RECORD-NAME
    ;OWNER SYSTEM.

MEMBER billing-record  OPTIONAL AUTOMATIC
          /* provide a generic key facility */
    ;ASCENDING RANGE KEY billing-key
        DUPLICATES NOT ALLOWED
    ;SEARCH KEY billing-key
        USING INDEX NAME billing-name.

SET NAME  billing-charges-set
    ;MODE CHAIN
    ;ORDER SORTED WITHIN RECORD-NAME
    ;OWNER billing-record.

MEMBER charges-record  OPTIONAL AUTOMATIC LINKED OWNER
    ;ASCENDING KEY date
        DUPLICATES ARE FIRST ALLOWED
    ;SET SELECTION CURRENT.    /* retrieve billing record 1st */

  /* other records are also members of this set but
      they are part of other billing subsystems and not
      of interest to this application   */

SET NAME terminating-set
    ;MODE CHAIN
    ;ORDER FIRST
    ;OWNER charges-record.
```

```
MEMBER terminating-record  OPTIONAL AUTOMATIC
    DUPLICATES NOT ALLOWED FOR term-key
    ;SET SELECTION CURRENT.   /* retrieve charges record 1st */

SET NAME settlement-set
    ;MODE CHAIN
    ;ORDER SORTED WITHIN RECORD-NAME
    ;OWNER SYSTEM.

MEMBER  traffic-breakdown-record  OPTIONAL MANUAL
    ;ASCENDING KEY t-b-code
        DUPLICATES NOT ALLOWED.

SET NAME  orig-settlement-set
  ;MODE POINTER-ARRAY
  ;ORDER SORTED WITHIN RECORD-NAME
   ;OWNER traffic-breakdown-record.

MEMBER originating-record  MANUAL OPTIONAL
   ;ASCENDING KEY orig-key
       DUPLICATES NOT ALLOWED
   ;SEARCH CALC
       USING binary-search   /* user routine */
   ;SET SELECTION LOCATION MODE OWNER.

 SET NAME settlement-charges-set
   ;MODE CHAIN
   ;ORDER SORTED WITHIN RECORD-NAME
   ;OWNER  originating-record.

MEMBER charges-record  OPTIONAL AUTOMATIC LINKED OWNER
  ;ASCENDING KEY orig-line, term-line, date
      DUPLICATES FIRST
  ;SET SELECTION IS THRU settlement-set USING
      CURRENT OF SET
   orig-settlement-set USING orig-key.
```

DDL PRODUCTION RULES

PRODUCTIONS
NUMBER OF PRODUCTIONS 263 RIGHT HAND SIDES 468

```
    1 <DDL> ::= _|_ <SCHEMA NAME> <AREAS> <RECORDS> <SETS> _|_

    2 <SCHEMA NAME> ::= <SCHEMA> .
    3                 | <SCHEMA> <PRIVACY CLAUSES> .

    4 <AREAS> ::= <AREA NTPO> .
    5           | <AREAS> <AREA NTPO> .

    6 <RECORDS> ::= <RECORD DEFN> <DATA DESCRIPTORS>
    7             | <RECORDS> <RECORD DEFN> <DATA DESCRIPTORS>

    8 <SETS> ::= <SET DESCRIPTOR> <MEMBER DESCRIPTORS>
    9          | <SETS> <SET DESCRIPTOR> <MEMBER DESCRIPTORS>

   10 <SCHEMA> ::= SCHEMA-NAME <SIMPLE-NAME>

   11 <SIMPLE-NAME> ::= <IDENTIFIER>

   12 <NAME> ::= <IDENTIFIER>
   13          | <IDENTIFIER> <QUALIFICATION>
   14          | <IDENTIFIER> <SUBSCRIPT>

   15 <QUALIFICATION> ::= IN <IDENTIFIER>
   16                   | <SUBSCRIPT> IN <IDENTIFIER>

   17 <SUBSCRIPT> ::= ( <NUMBER> )

   18 <PRIVACY CLAUSES> ::= <PRIVACY CLAUSE>
   19                     | <PRIVACY CLAUSES> <PRIVACY CLAUSE>

   20 <PRIVACY CLAUSE> ::= PRIVACY <PRIVACY TYPE1> <KEYLIST>
   21                    | PRIVACY <PRIVACY TYPE2> <KEYLIST>
   22                    | PRIVACY <OPTS> <KEYLIST>
   23                    | PRIVACY <KEYLIST>

   24 <KEYLIST> ::= <KEY>
   25            | <KEYLIST> OR <KEY>

   26 <PRIVACY TYPE1> ::= <PRIVACY TYPES1>
   27                   | <PRIVACY TYPE1> <PRIVACY TYPES1>

   28 <PRIVACY TYPES1> ::= LOCKS
   29                    | DISPLAY
   30                    | COPY
   31                    | ALTER

   32 <KEY> ::= <STRING>
   33        | <SIMPLE-NAME>
   34        | PROCEDURE <SIMPLE-NAME>

   35 <AREA NTPO> ::= <AREA NTP>
   36               | <AREA NTP> <ON CLAUSES>

   37 <AREA NTP> ::= <AREA NT>
   38              | <AREA NT> <PRIVACY CLAUSES>

   39 <AREA NT> ::= <AREA N>
   40             | <AREA N> TEMPORARY

   41 <AREA N> ::= AREA-NAME <SIMPLE-NAME>

   42 <PRIVACY TYPE2> ::= <PRIVACY TYPES2>
   43                   | <PRIVACY TYPE2> <PRIVACY TYPES2>

   44 <PRIVACY TYPES2> ::= <MODES>
```

```
45                          | <RESTRICTED> <MODES>
46                          | <NAMELIST>

47 <MODES> ::= RETRIEVAL
48              | UPDATE

49 <RESTRICTED> ::= EXCLUSIVE
50                   | PROTECTED

51 <NAMELIST> ::= <NAME>
52                 | <NAMELIST> , <NAME>

53 <ON CLAUSES> ::= <ON CLAUSE>
54                   | <ON CLAUSES> <CN CLAUSE>

55 <ON CLAUSE> ::= ON <PROCEDURE>
56                  | ON <TRIGGER> <PROCEDURE>
57                  | ON <OPTS> <PROCEDURE>
58                  | ON <OPTS> <PROCEDURE> USING <NAMELIST>

59 <OPTS> ::= <OPT>
60             | <OPTS> <OPT>

61 <OPT> ::= INSERT
62            | REMOVE
63            | FIND
64            | GET
65            | MODIFY
66            | ORDER
67            | STORE
68            | DELETE
69            | DELETE ONLY
70            | DELETE SELECTIVE
71            | DELETE ALL

72 <TRIGGER> ::= <TRIGGERS>
73                | <TRIGGER> <TRIGGERS>

74 <TRIGGERS> ::= CLOSE
75                 | OPEN
76                 | OPEN <USAGES>

77 <USAGES> ::= <USAGE>
78               | <USAGES> <USAGE>

79 <USAGE> ::= <RESTRICTION>
80             | <RESTRICTION> <MODES>

81 <RESTRICTION> ::= <RESTRICTED>
82                    | NON-EXCLUSIVE

83 <RECORD DEFN> ::= <RECORD ON> .
84                    | <RECORD ON> <PRIVACY CLAUSES> .

85 <RECORD ON> ::= <RECORD NAME>
86                  | <RECORD NAME> <ON CLAUSES>

87 <RECORD NAME> ::= <RECORD> <WITHIN CLAUSE>
88                    | <RECORD> <LOCATION CLAUSE> <WITHIN CLAUSE>

89 <RECORD> ::= RECORD-NAME <SIMPLE-NAME>

90 <LOCATION CLAUSE> ::= <LOCATION> DIRECT <NAME>
91                       | <LOCATION> VIA <SIMPLE-NAME>
92                       | <LOCATION> CALC <CALC CLAUSE>

93 <LOCATION> ::= LOCATION
94                | LOCATION MODE

95 <CALC CLAUSE> ::= <SIMPLE-NAME> USING <NAMELIST> <DUPS>
96                    | USING <NAMELIST> <DUPS>
```

```
148 <ACTUAL VIRTUAL CLAUSE1> ::= <AVDEFN>
149                           | <AVDEFN> USING <NAMELIST>
150                           | <AVDEFN> <NAMELIST> <SIMPLE-NAME>

151 <ACTUAL VIRTUAL CLAUSE2> ::= <AVKEYWORD> SOURCE <NAME> OWNER
                                                  <SIMPLE-NAME>

152 <AVKEYWORD> ::= ACTUAL
153               | VIRTUAL

154 <CHECK CLAUSES> ::= <CHECK CLAUSE>
155                   | <CHECK CLAUSES> <CHECK CLAUSE>

156 <CHECK CLAUSE> ::= PICTURE
157                 | RANGE <STRING> THRU <STRING>
158                 | RANGE <NUMLIT> THRU <NUMLIT>
159                 | <SIMPLE-NAME> USING <NAMELIST>

160 <ENCODE DECODE CLAUSE> ::= <EDOPTS> <PROCEDURE>
161                            | <EDOPTS> ALWAYS <PROCEDURE>

162 <EDOPTS> ::= ENCODING
163             | DECODING

164 <SET DESCRIPTOR> ::= <SET NAME> <SET MODE> <SET ORDER> <SET OWNER> .
165           | <SET NAME> <SET MODE> <SET ORDER> <SET DATA> <SET OWNER> .

166 <SET NAME> ::= SET-NAME <SIMPLE-NAME>

167 <SET MODE> ::= MODE <MODE TYPES>

168 <MODE TYPES> ::= <CHAINED>
169                | <POINTER ARRAY>
170                | <SIMPLE-NAME>

171 <CHAINED> ::= CHAIN
172             | CHAIN PRIOR

173 <POINTER ARRAY> ::= POINTER-ARRAY
174                   | POINTER-ARRAY DYNAMIC

175 <SET OWNER> ::= OWNER <SIMPLE-NAME>
176               | OWNER SYSTEM

177 <SET ORDER> ::= ORDER <ORDER OPTIONS>

178 <ORDER OPTIONS> ::= ALWAYS <ORDERING>
179                   | <ORDERING>
180                   | SORTED <SORT HOW>

181 <ORDERING> ::= FIRST
182             | LAST
183             | NEXT
184             | PRIOR

185 <SORT HOW> ::=
186             | <INDEXING>
187             | <INDEXING> <SORT WHERE>
188             | <SORT WHERE>

189 <INDEXING> ::= INDEXED
190             | INDEXED <SIMPLE-NAME>

191 <SORT WHERE> ::= RECORD-NAME
192               | WITHIN RECORD-NAME
193               | DATABASE-KEY
194               | DUPLICATES
195               | DUPLICATES <DUPLICATE ACTION>

196 <DUPLICATE ACTION> ::= FIRST
197                       | LAST
```

```
198                          | NOT

199 <SET DATA> ::= <ON CLAUSES>
200                | <ON CLAUSES> <PRIVACY CLAUSES>
201                | <PRIVACY CLAUSES>

202 <MEMBER DESCRIPTORS> ::= <MEMBER DESCRIPTOR>
203                           | <MEMBER DESCRIPTORS> <MEMBER DESCRIPTOR>

204 <MEMBER DESCRIPTOR> ::= <MEMBER NAME> <MEMBER OPTIONS> .
205                        | <MEMBER NAME> <MEMBER OPTIONS> <MEMBER DATA> .

206 <MEMBER NAME> ::= MEMBER <SIMPLE-NAME>

207 <MEMBER OPTIONS> ::= <MOCCUR> <MTYPE>
208                        | <MOCCUR> <MTYPE> <MOPT>

209 <MOPT> ::= OWNER
210           | OWNER <MLINKS>
211           | <MLINKS>

212 <MOCCUR> ::= MANDATORY
213             | OPTIONAL

214 <MTYPE> ::= AUTOMATIC
215             | MANUAL

216 <MLINKS> ::= <MLINK>
217              | <MLINKS> <MLINK>

218 <MLINK> ::= DUPLICATES NOT <NAMELIST>

219 <MEMBER DATA> ::= <MEMBER SEARCH SELECTION>
220                  | <MEMBER SEQ CLAUSES> <MEMBER SEARCH SELECTION>

221 <MEMBER SEARCH SELECTION> ::= <SET SELECTION CLAUSE>
222                                | <MEMBER SEARCH CLAUSES>
223                             | <MEMBER SEARCH CLAUSES> <SET SELECTION CLAUSE>

224 <MEMBER SEQ CLAUSES> ::= <MEMBER SEQ CLAUSE>
225                          | <MEMBER SEQ CLAUSES> <MEMBER SEQ CLAUSE>

226 <MEMBER SEQ CLAUSE> ::= <MORDER> <KEYSEQ>
227                         | <MORDER> RANGE <KEYSEQ>

228 <KEYSEQ> ::= <NAMELIST>
229             | <NAMELIST> <DUPLICATE OPTION>

230 <DUPLICATE OPTION> ::= DUPLICATES
231                        | DUPLICATES <DUPLICATE ACTION>

232 <MORDER> ::= ASCENDING
233             | DESCENDING

234 <MEMBER SEARCH CLAUSES> ::= <MEMBER SEARCH CLAUSE>
235                            <MEMBER SEARCH CLAUSES> <MEMBER SEARCH CLAUSE>

236 <MEMBER SEARCH CLAUSE> ::= SEARCH <NAMELIST>
237                           | SEARCH <NAMELIST> USING <KEYCALC> <DUPOPTN>

238 <KEYCALC> ::= CALC
239              | INDEX
240              | INDEX <SIMPLE-NAME>
241              | <SIMPLE-NAME>

242 <DUPOPTN> ::= DUPLICATES
243              | DUPLICATES NOT

244 <SET SELECTION CLAUSE> ::= SELECTION <HOW SELECT2>
245                           | SELECTION <NAME> USING <HOW SELECT3> <THRU SETS>

246 <HOW SELECT2> ::= <HOW SELECT1>
```

```
247                     | LOCATION <LOC CPT1>

248 <HOW SELECT1> ::= CURRENT
249                     | LOCATION

250 <HOW SELECT3> ::= <HOW SELECT1>
251                     | LOCATION <ALIAS CLAUSES>

252 <LOC OPT1> ::= USING <NAMELIST> <ALIAS CLAUSES>

253 <ALIAS CLAUSES> ::= <ALIAS CLAUSE>
254                       | <ALIAS CLAUSES> <ALIAS CLAUSE>

255 <ALIAS CLAUSE> ::= ALIAS <NAME> <SIMPLE-NAME>

256 <THRU SETS> ::= <THRU SET>
257                  | <THRU SETS> <THRU SET>

258 <THRU SET> ::= <SIMPLE-NAME> <LOC OPT1>

259 <PROCEDURE> ::= CALL <SIMPLE-NAME>

260 <NUMLIT> ::= <NUMBER> .
261               | <NUMBER> . <NUMBER>
262               | . <NUMBER>
263               | . <NUMBER> E+ <NUMBER>
```

SORTED VOCABULARY (RESERVED WORDS)
TERMINAL SYMBOLS

1 .
2 (
3)
4 ,
5 E+
6 IN
7 CN
8 OR
9 |
10 ALL
11 BIT
12 GET
13 NOT
14 VIA
15 CALC
16 CALL
17 COPY
18 FIND
19 LAST
20 MODE
21 NEXT
22 ONLY
23 OPEN
24 REAL
25 THRU
26 TYPE
27 ALIAS
28 ALTER
29 CHAIN
30 CHECK
31 CLOSE
32 FIRST
33 FIXED
34 FLOAT
35 INDEX
36 LOCKS
37 ORDER
38 OWNER
39 PRIOR
40 RANGE
41 STORE
42 USING
43 ACTUAL
44 ALWAYS
45 BINARY
46 DELETE
47 DIRECT
48 INSERT
49 MANUAL
50 MEMBER
51 MODIFY
52 OCCURS
53 REMOVE
54 RESULT
55 SEARCH
56 SORTED
57 SOURCE
58 SYSTEM
59 UPDATE
60 WITHIN
61 AREA-ID
62 COMPLEX
63 CURRENT
64 DECIMAL
65 DISPLAY
66 DYNAMIC
67 INDEXED
68 PICTURE
69 PRIVACY
70 VIRTUAL

72 <STRING>
73 DECODING
74 ENCODING
75 LOCATION
76 OPTIONAL
77 SET-NAME
78 AREA-NAME
79 ASCENDING
80 AUTOMATIC
81 CHARACTER
82 EXCLUSIVE
83 MANDATORY
84 PROCEDURE
85 PROTECTED
86 RETRIEVAL
87 SELECTION
88 SELECTIVE
89 TEMPORARY
90 DESCENDING
92 RECORD-NAME
93 SCHEMA-NAME
94 <IDENTIFIER>
95 DATABASE-KEY
96 NON-EXCLUSIVE
97 POINTER-ARRAY
98 <IMPLEMENTOR TYPE>

THE SCHEMA FOR THE SCHEMA DATABASE

```
SCHEMA-NAME SCHEMA-DDL
   PRIVACY 'SYSTEM-LOCK'.

AREA-NAME SCHEMA-AREA.
AREA-NAME SCHEMA-PRIVACIES-AREA.

RECORD-NAME
   SCHEMA-RECORD
   LOCATION VIA SCHEMA-DDL-SET
   WITHIN SCHEMA-AREA.
01 %SCHEMA-NAME     TYPE     CHARACTER 30.

RECORD-NAME
   PRIVACY-RECORD
   WITHIN SCHEMA-PRIVACIES-AREA.
01 PRIVACY-FOR  TYPE  BIT 16.
01 NUMB-KEYS   TYPE  FIXED.
01 KEY-TYPE    TYPE  BIT 8  OCCURS NUMB-KEYS.
01 KEY-TYPE  TYPE  BIT 8.

RECORD-NAME
   NAME-RECORD
   WITHIN SCHEMA-PRIVACIES-AREA.
01 NAME-VALUE  TYPE  CHARACTER 30.

RECORD-NAME
   PROC-RECORD
   WITHIN SCHEMA-PRIVACIES-AREA.
01 PROC-NAME  TYPE  CHARACTER 30.

RECORD-NAME
   LITERAL-RECORD
   WITHIN SCHEMA-PRIVACIES-AREA.
01 LITERAL-VALUE  TYPE  CHARACTER 30.

RECORD-NAME
   AREA-RECORD
   LOCATION VIA SCHEMA-SET
   WITHIN SCHEMA-AREA.
01 %AREA-NAME  TYPE  CHARACTER.
01 TEMP-FLAG  TYPE  BIT.

RECORD-NAME
   ON-RECORD
   WITHIN SCHEMA-PRIVACIES-AREA.
01 ON-TYPE    TYPE  BIT.
01 ON-FOR     TYPE  BIT 16.

RECORD-NAME
   RECORD-RECORD
   LOCATION VIA SCHEMA-SET
   WITHIN SCHEMA-AREA.
01 %RECORD-NAME  TYPE  CHARACTER 30.

RECORD-NAME
   LOCATION-RECORD
   LOCATION VIA RECORD-SET
   WITHIN SCHEMA-AREA.
01 LOCATION-MODE TYPE BIT 8.

RECORD-NAME
   DIRECT-RECORD
   LOCATION VIA LOCATION-SET
   WITHIN SCHEMA-AREA.
01 KEY-TYPE TYPE BIT.

RECORD-NAME
   CALC-RECORD
```

```
    LOCATION VIA LOCATION-SET
    WITHIN SCHEMA-AREA.
01 CALC-PROC-OPT  TYPE  BIT.
01  CALC-DUPS-OPT  TYPE  BIT.

RECORD-NAME
    WITHIN-RECORD
    LOCATION VIA RECORD-SET
    WITHIN SCHEMA-AREA.
01 %AREA-ID  TYPE  CHARACTER 30.

RECORD-NAME
    SET-RECORD
    LOCATION VIA SCHEMA-SET
    WITHIN SCHEMA-AREA.
01 %SET-NAME  TYPE  CHARACTER 30.
01 MODE-DEFN  TYPE  BIT 8.

RECORD-NAME
    ORDER-RECORD
    LOCATION VIA SET-SET
    WITHIN SCHEMA-AREA.
01 ORDER-DEFN-FLAGS  TYPE  BIT 16.

RECORD-NAME
    OWNER-RECORD
    LOCATION VIA SET-SET
    WITHIN SCHEMA-AREA.
01  OWNER-TYPE  TYPE  BIT.

RECORD-NAME
    IDENTIFIER-RECORD
    LOCATION VIA RECORD-SET
    WITHIN SCHEMA-AREA.
01 IDENTIFIER-NAME  TYPE  CHARACTER 30.
01  LEVEL-NO  TYPE  BIT 8.

RECORD-NAME
    PICTURE-RECORD
    LOCATION VIA RECORD-SET
    WITHIN SCHEMA-AREA.
01 PICTURE-TYPE  TYPE BIT.
01 PICTURE-VALUE  TYPE  CHARACTER 30.

RECORD-NAME
    TYPE-RECORD
    LOCATION VIA RECORD-SET
    WITHIN SCHEMA-AREA.
01 TYPE-FLAGS  TYPE  BIT 16.
01 INT1 TYPE BIT 8.
01 INT2 TYPE BIT 8.

RECORD-NAME
    OCCURS-RECORD
    LOCATION VIA RECORD-SET
    WITHIN SCHEMA-AREA.
01 OCCURS-TYPE  TYPE  BIT.
01 INT4  TYPE  BIT 16.

RECORD-NAME
    ACTUAL-VIRTUAL-RECORD
    LOCATION VIA RECORD-SET
    WITHIN SCHEMA-AREA.
01 AV-FLAGS  TYPE  BIT 8.

RECORD-NAME
    CHECK-RECORD
    LOCATION VIA RECORD-SET
    WITHIN SCHEMA-AREA.
01 CHECK-TYPE  TYPE BIT 8.

RECORD-NAME
```

```
    ENCODE-DECODE-RECORD
    LOCATION VIA RECORD-SET
    WITHIN SCHEMA-AREA.
01   E-D-FLAGS   TYPE BIT 8.
01 PROC-NAME   TYPE   CHARACTER 30.

RECORD-NAME
    MEMBER-RECORD
    LOCATION VIA SET-SET
    WITHIN SCHEMA-AREA.
01 MEMBER-NAME   TYPE   CHARACTER 30.
01 OPTIONS   TYPE   BIT 8.

RECORD-NAME
    ASCENDING-DESCENDING-RECORD
    LOCATION VIA MEMBER-SET
    WITHIN SCHEMA-AREA.
01 A-D-FLAGS   TYPE   BIT 8.

RECORD-NAME
    SEARCH-RECORD
    LOCATION VIA MEMBER-SET
    WITHIN SCHEMA-AREA.
01 SEARCH-OPTS TYPE BIT 8.

RECORD-NAME
    SELECTION-RECORD
    LOCATION VIA MEMBER-SET
    WITHIN SCHEMA-AREA.
01 SELECTION-OPTS   TYPE   BIT 8.

RECORD-NAME
    USING-RECORD
    LOCATION   VIA   SELECTION-SET
    WITHIN   SCHEMA-AREA.
01   DUMMY   TYPE   BIT.

RECORD-NAME
    ALIAS-RECORD
    LOCATION VIA SELECTION-SET
    WITHIN SCHEMA-AREA.
01 DUMMY   TYPE   BIT.


SET-NAME
    SCHEMA-DDL-SET       /* SINGULAR SET NO SELECTION CLAUSES */
    MODE CHAIN
    ORDER SORTED
    OWNER SYSTEM.

MEMBER
    SCHEMA-RECORD   MANDATORY AUTOMATIC
    ASCENDING KEY %SCHEMA-NAME   DUPLICATES NOT ALLOWED
    SEARCH KEY %SCHEMA-NAME.


SET-NAME
    SCHEMA-SET
    MODE CHAIN
    ORDER SORTED WITHIN RECORD-NAME
    OWNER SCHEMA-RECORD.

MEMBER
    AREA-RECORD MANDATORY AUTOMATIC
    ASCENDING KEY %AREA-NAME   DUPLICATES NOT ALLOWED
    SEARCH KEY %AREA-NAME
    SELECTION CURRENT.

MEMBER
    RECORD-RECORD   MANDATORY AUTOMATIC
    ASCENDING KEY %RECORD-NAME   DUPLICATES NOT ALLOWED
    SEARCH KEY %RECORD-NAME
```

```
    SELECTION CURRENT.

MEMBER
    SET-RECORD  MANDATORY AUTOMATIC
    ASCENDING KEY %SET-NAME DUPLICATES NOT ALLOWED
    SEARCH KEY %SET-NAME
    SELECTION CURRENT.

MEMBER
    PRIVACY-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    PRIVACY-SET
    MODE CHAIN
    ORDER LAST
    OWNER PRIVACY-RECORD.


MEMBER
    NAME-RECORD  OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    PROC-RECORD  OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    LITERAL-RECORD  OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    AREA-SET
    MODE CHAIN
    ORDER SORTED WITHIN RECORD-NAME
    OWNER AREA-RECORD.

MEMBER
    RECORD-RECORD  OPTIONAL MANUAL  OWNER
    ASCENDING KEY %RECORD-NAME DUPLICATES NOT ALLOWED
    SEARCH KEY %RECORD-NAME
    SELECTION CURRENT.

MEMBER
    PRIVACY-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    ON-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    ON-SET
    MODE POINTER-ARRAY
    ORDER LAST
    OWNER ON-RECORD.

MEMBER
    PROC-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    IDENTIFIER-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    RECORD-SET
    MODE CHAIN
```

```
    ORDER SORTED WITHIN RECORD-NAME
    OWNER RECORD-RECORD.

MEMBER
    IDENTIFIER-RECORD MANDATORY AUTOMATIC CWNER
    ASCENDING KEY IDENTIFIER-NAME DUPLICATES NOT
    SEARCH KEY IDENTIFIER-NAME USING INDEX DUPLICATES NOT
    SELECTION CURRENT.

MEMBER
    PRIVACY-RECORD OPTICNAL MANUAL
    SELECTION CURRENT.

MEMBER
    ON-RECORD  OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    LOCATION-RECORD MANDATORY AUTCMATIC
    SELECTION CURRENT.

MEMBER
    WITHIN-RECORD MANDATORY AUTOMATIC
    SELECTION CURRENT.


SET-NAME
    LOCATION-SET
    MODE POINTER-ARRAY
    ORDER LAST
    OWNER LOCATION-RECORD.

MEMBER
    DIRECT-RECORD MANDATORY AUTCMATIC
    SELECTION CURRENT.

MEMBER
    CALC-RECORD  MANDATORY AUTOMATIC
    SELECTION CURRENT.

MEMBER
    SET-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    WITHIN-SET
    MODE POINTER-ARRAY
    ORDER LAST
    OWNER WITHIN-RECORD.

MEMBER
    AREA-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    DIRECT-SET
    MODE POINTER-ARRAY
    ORDER LAST
    OWNER DIRECT-RECORD.

MEMBER
    NAME-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    IDENTIFIER-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
```

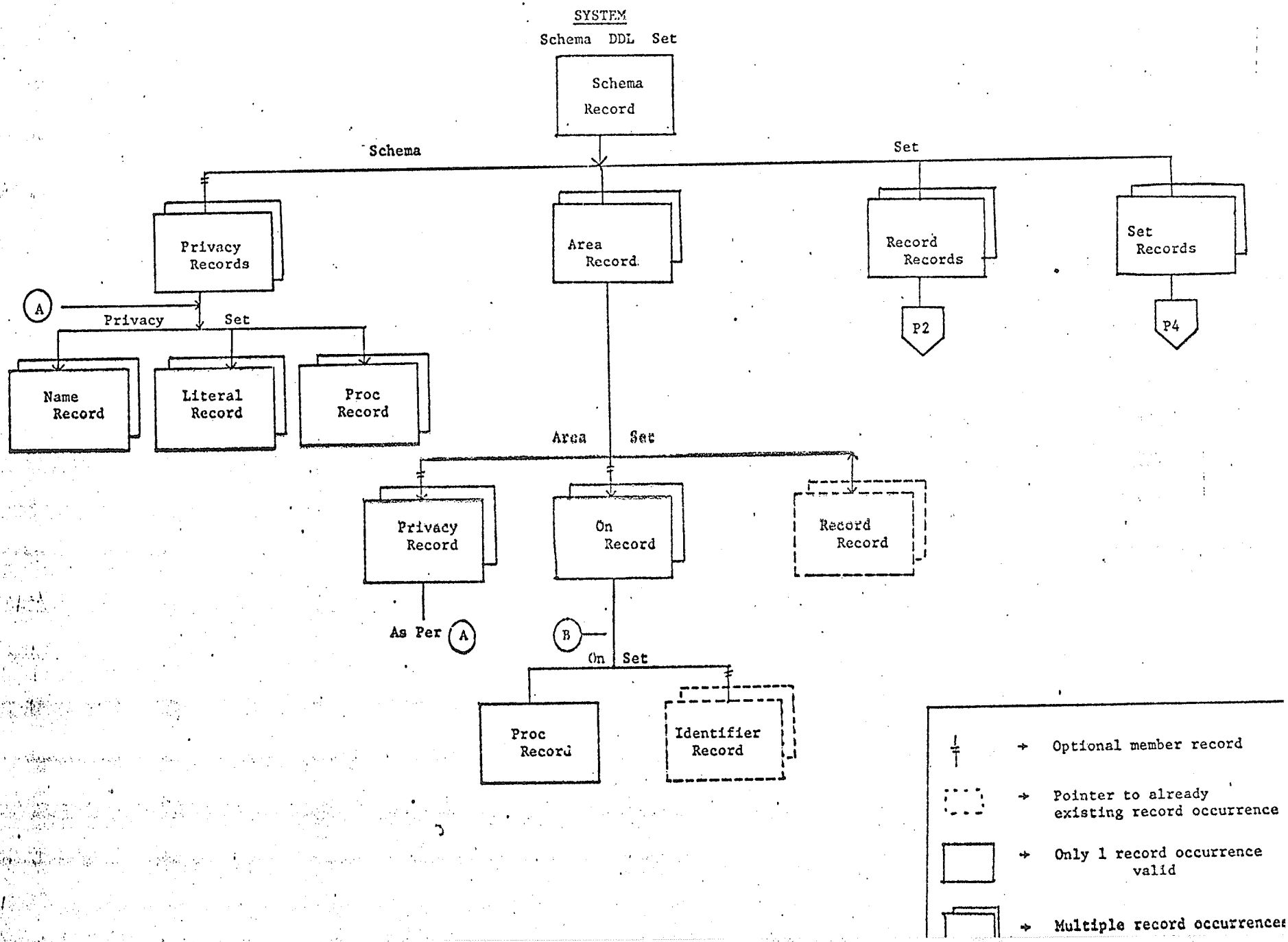
```
    CALC-SET
    MODE POINTER-ARRAY
    ORDER LAST
    OWNER CALC-RECORD.

MEMBER
    PROC-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    IDENTIFIER-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    SET-SET
    MODE CHAIN
    ORDER SORTED WITHIN RECORD-NAME
    OWNER SET-RECORD.

MEMBER
    MEMBER-RECORD MANDATORY AUTOMATIC OWNER
    ASCENDING KEY MEMBER-NAME DUPLICATES NOT
    SEARCH KEY MEMBER-NAME
    SELECTION CURRENT.

MEMBER
    PRIVACY-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    ON-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    ORDER-SET
    MODE CHAIN
    ORDER LAST
    OWNER SET-RECORD.

MEMBER
    NAME-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    OWNER-SET
    MODE CHAIN
    ORDER LAST
    OWNER SET-RECORD.

MEMBER
    RECORD-RECORD OPTIONAL MANUAL OWNER
    SELECTION CURRENT.


SET-NAME
    IDENTIFIER-SET
    MODE CHAIN
    ORDER SORTED WITHIN RECORD-NAME
    OWNER IDENTIFIER-RECORD.

MEMBER
    PICTURE-RECORD MANDATORY AUTOMATIC
    SELECTION CURRENT.

MEMBER
    TYPE-RECORD MANDATORY AUTOMATIC
    SELECTION CURRENT.

MEMBER
```

```
    OCCURS-RECORD MANDATORY AUTOMATIC
    SELECTION CURRENT.

MEMBER
    ACTUAL-VIRTUAL-RECORD  MANDATORY AUTOMATIC
    SELECTION CURRENT.

MEMBER
    CHECK-RECORD MANDATORY AUTOMATIC
    SELECTION CURRENT.

MEMBER
    ON-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    ENCODE-DECODE-RECORD MANDATORY AUTOMATIC
    SELECTION CURRENT.

MEMBER
    PRIVACY-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    OCCURS-SET
    MODE POINTER-ARRAY
    ORDER LAST
    OWNER OCCURS-RECORD.

MEMBER
    IDENTIFIER-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    ACTUAL-VIRTUAL-SET
    MODE POINTER-ARRAY
    ORDER LAST
    OWNER ACTUAL-VIRTUAL-RECORD.

MEMBER
    PROC-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    IDENTIFIER-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    RECORD-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    SET-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    CHECK-SET
    MODE POINTER-ARRAY
    ORDER LAST
    OWNER CHECK-RECORD.

MEMBER
    LITERAL-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    PROC-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.
```

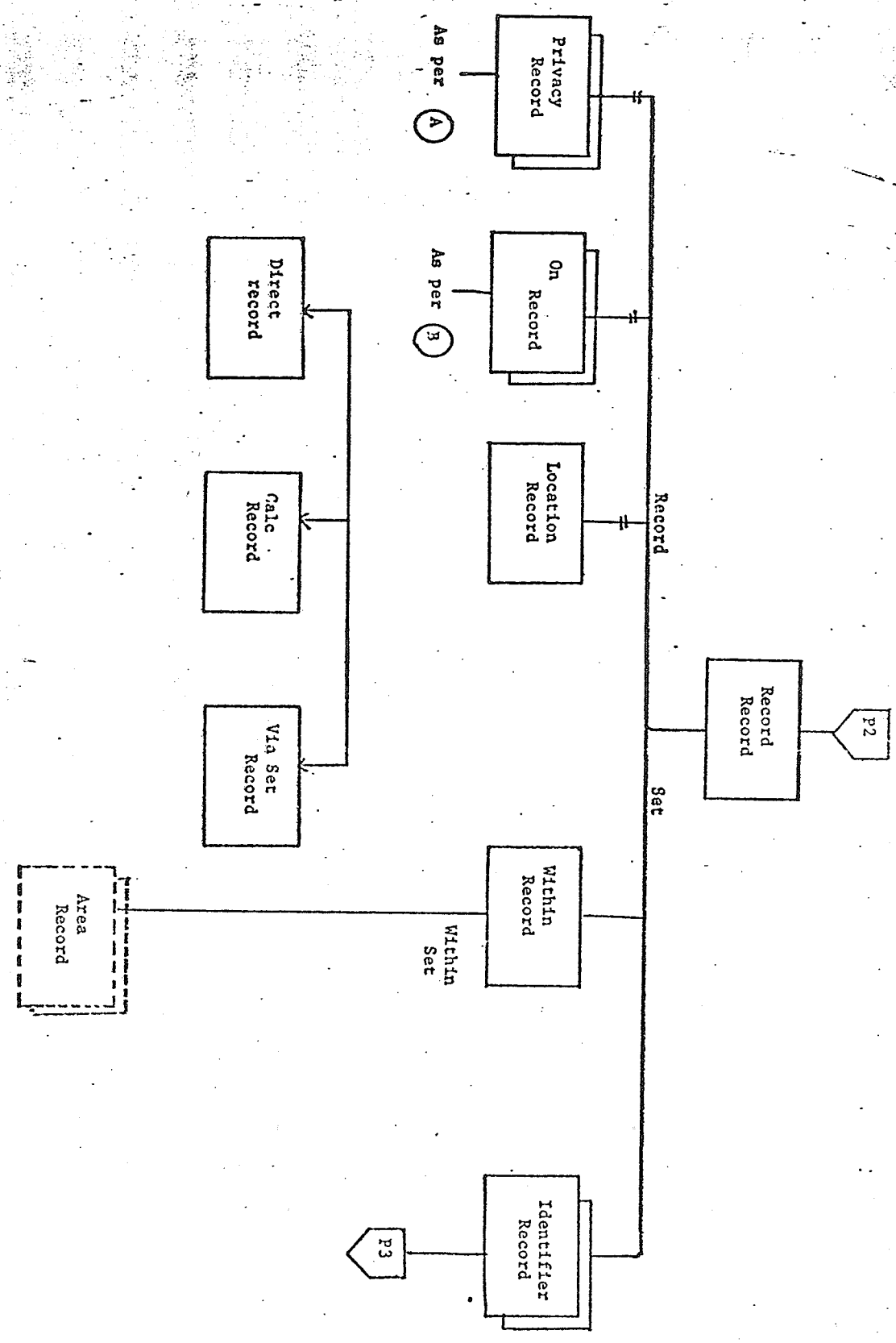
```
MEMBER
    IDENTIFIER-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    MEMBER-SET
    MODE POINTER-ARRAY
    ORDER SORTED WITHIN RECORD-NAME
    OWNER MEMBER-RECORD.

MEMBER
    ASCENDING-DESCENDING-RECORD MANDATORY AUTOMATIC
    SELECTION CURRENT.

MEMBER
    SEARCH-RECORD MANDATORY AUTOMATIC
    SELECTION CURRENT.

MEMBER
    SELECTION-RECORD MANDATORY AUTOMATIC
    SELECTION CURRENT.

MEMBER  /*FOR DUPLICATES CLAUSE IN MEMBER */
    IDENTIFIER-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    ASCENDING-DESCENDING-SET
    MODE POINTER-ARRAY
    ORDER LAST
    OWNER ASCENDING-DESCENDING-RECORD.

MEMBER
    IDENTIFIER-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    SEARCH-SET
    MODE POINTER-ARRAY
    ORDER LAST
    OWNER SEARCH-RECORD.

MEMBER
    IDENTIFIER-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    NAME-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    PROC-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    SELECTION-SET
    MODE POINTER-ARRAY
    ORDER LAST
    OWNER SELECTION-RECORD.

MEMBER
    SET-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    IDENTIFIER-RECORD OPTICNAL MANUAL
    SELECTION CURRENT.
```

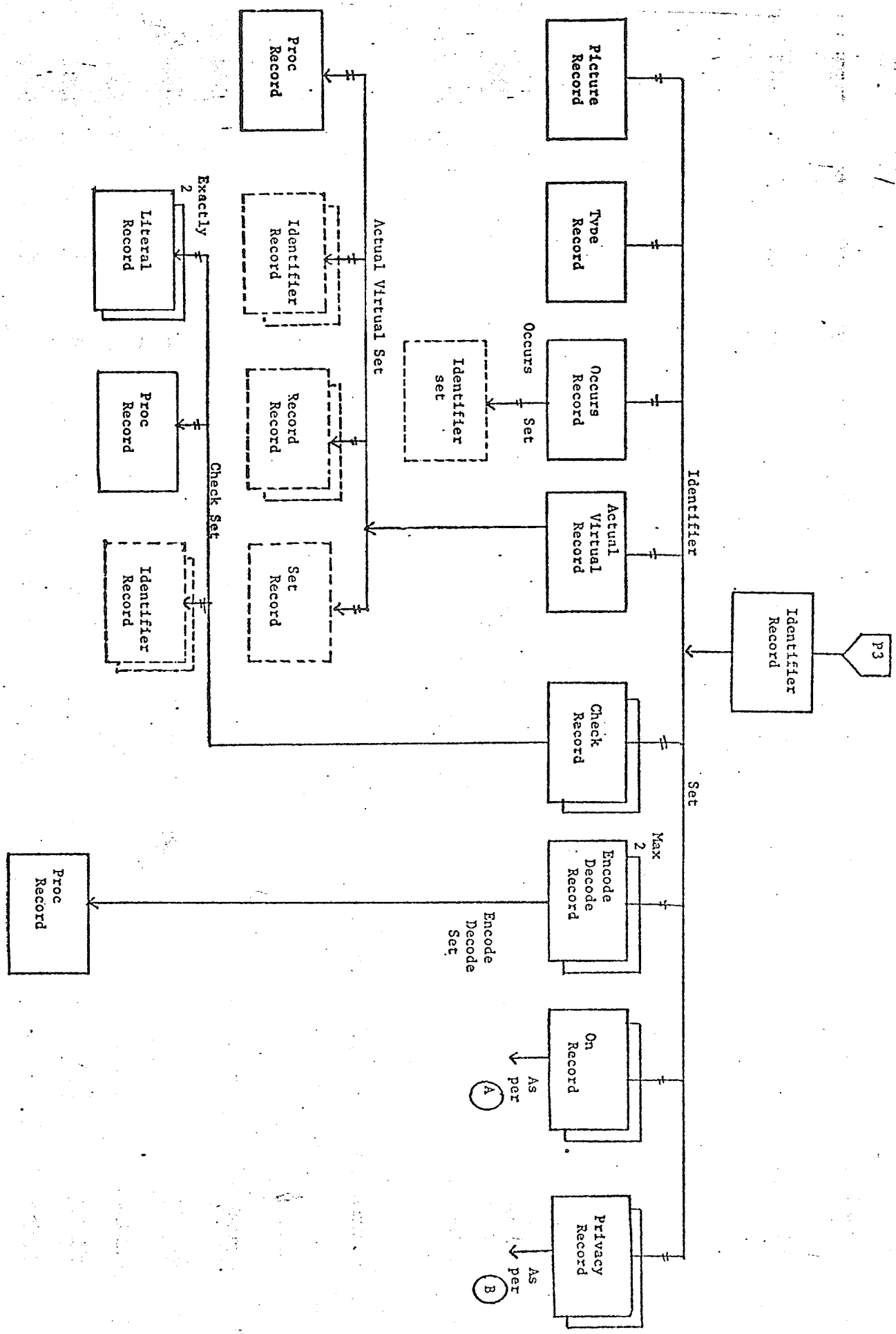
```
MEMBER
    NAME-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    SELECTION-SET
    MODE POINTER-ARRAY
    ORDER LAST
    OWNER SELECTION-RECORD.

MEMBER
    SET-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    USING-RECORD MANDATORY AUTOMATIC
    SELECTION CURRENT.

MEMBER
    ALIAS-RECORD MANDATORY AUTOMATIC
    SELECTION CURRENT.


SET-NAME
    USING-SET
    MODE POINTER-ARRAY
    ORDER LAST
    OWNER USING-RECORD.

MEMBER
    IDENTIFIER-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.


SET-NAME
    ALIAS-SET
    MODE POINTER-ARRAY
    ORDER LAST
    OWNER ALIAS-RECORD.

MEMBER
    IDENTIFIER-RECORD OPTIONAL MANUAL
    SELECTION CURRENT.

MEMBER
    NAME-RECORD  OPTIONAL MANUAL
    SELECTION CURRENT.
/*   */
EOF EOF EOF
```

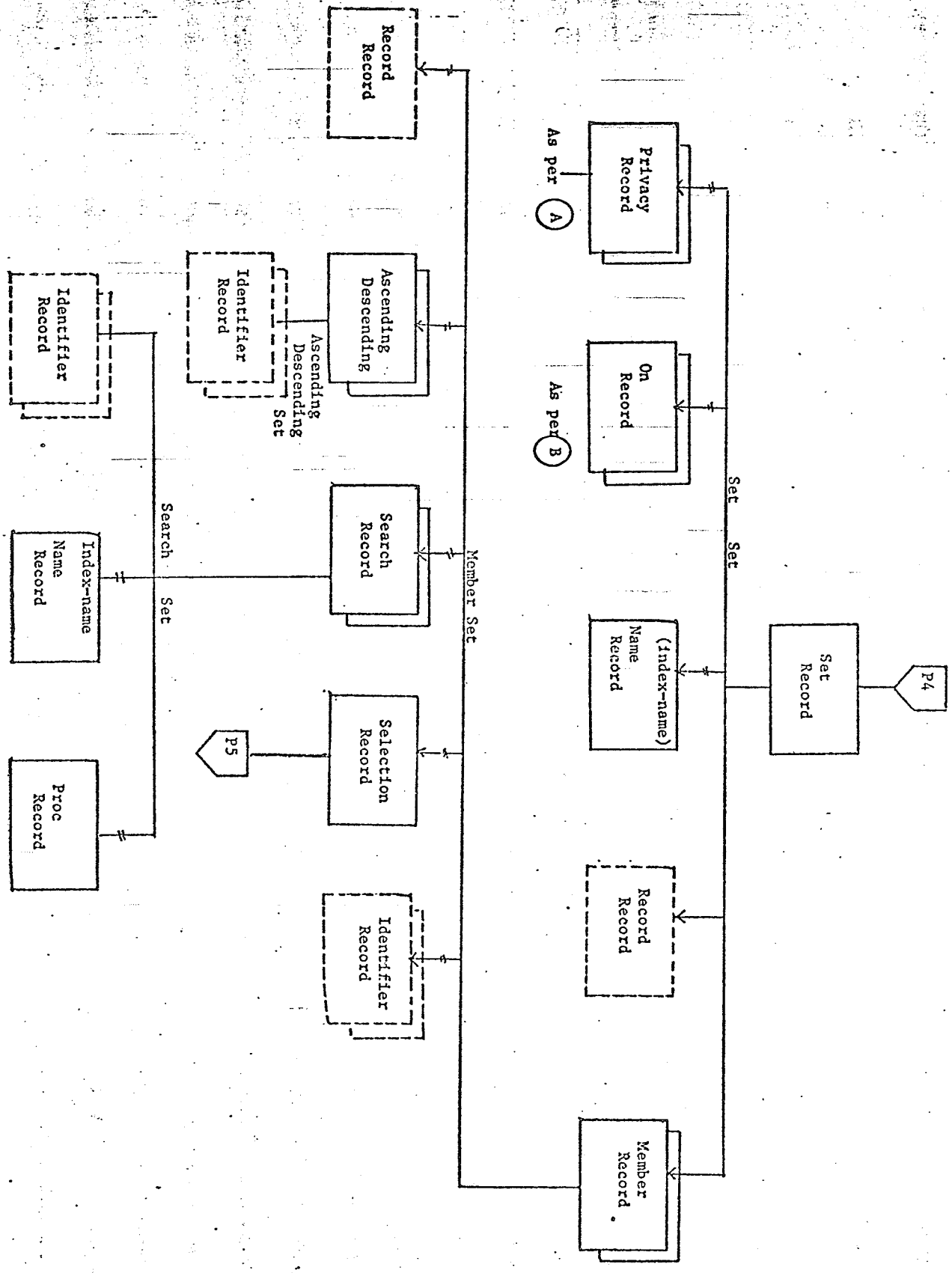
STRUCTURE OF SCHEMA DDL DATABASE

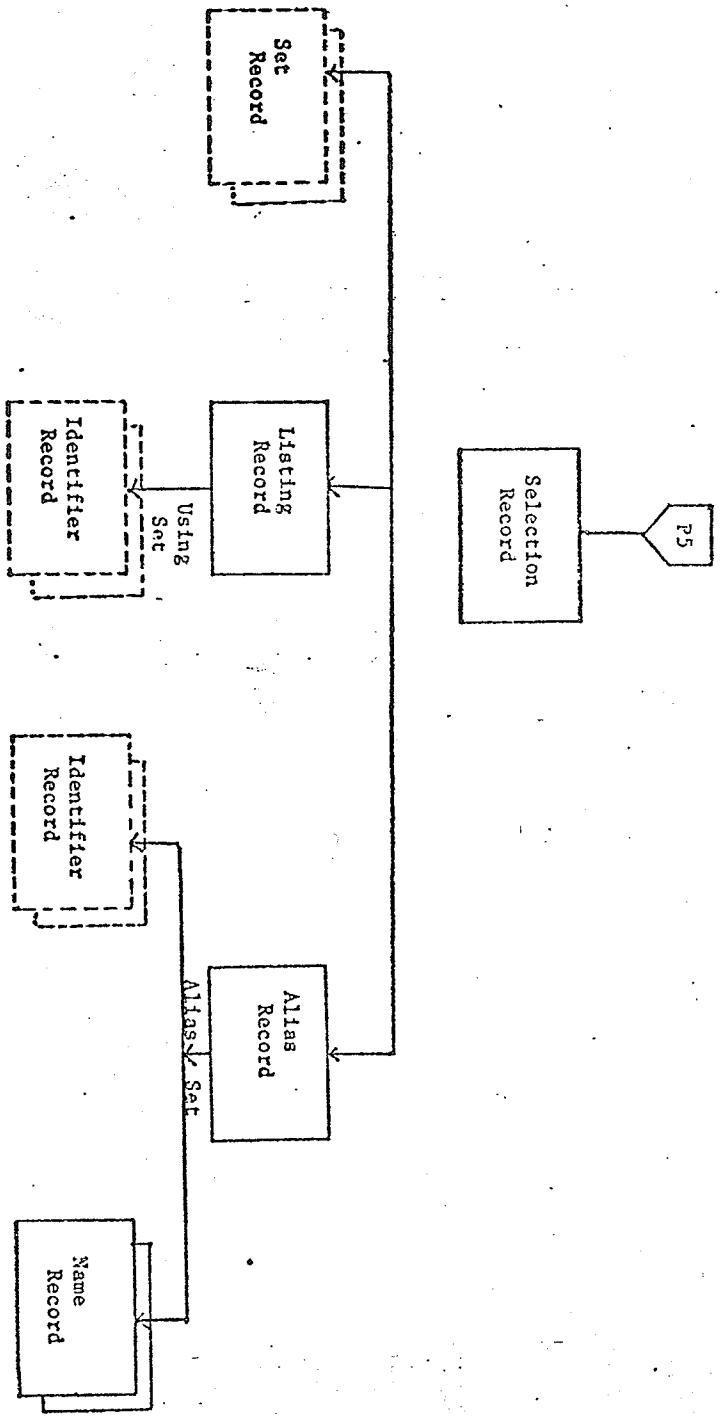


SCHEMA DATABASE DATA STRUCTURE


Privacy Record
As per A
On Record
As per B
Location Record
Record
Record Record
P2
Set
Direct record
Calc Record
Via Set Record
Within Record
Within Set
Area Record
Identifier Record
P3


Picture Record
Type Record
Occurs Record
Occurs
Set
Identifier set
Actual Virtual Record
Actual Virtual Set
Proc Record
Identifier Record
Literal Record
Exactly 2
Record Record
Set Record
Proc Record
Identifier Record
Check Set
Identifier
Identifier Record
P3
Check Record
Set
Max 2
Encode Decode Record
Encode Decode Set
Proc Record
On Record
As per A
Privacy Record
As per B


Record Record
Privacy Record
As per A
Ascending Descending
Identifier Record
Ascending Descending Set
Identifier Record
On Record
As per B
Search Record
Search Set
Index-name Name Record
Member Set
Set Set
Set Record
P4
(Index-name) Name Record
Selection Record
P5
Record Record
Identifier Record
Proc Record
Member Record


Selection Record
P5
Set Record
Listing Record
Using Set
Identifier Record
Alias Record
Alias Set
Identifier Record
Name Record

BIBLIOGRAPHY

MATTHEWS, D.O., The Design of the Management Information System.
pub by Auerbach 1971 Cat # 77-124629.

MURDICK, R.G., and ROSS, J.E.
Information and Systems for Modern Management
pub by Prentice-Hall 1971 Cat # 70-130842.

CODASYL DATA BASE TASK GROUP REPORT of April 1971
pub by Association for Computing Machinery
Single Copy Department
1133 Avenue of the Americas
New York, N.Y. 10036.

CODASYL COBOL Data Base Facility Proposal
A CODASYL Data Base Task Group Report March 1973
pub by The Technical Services Branch
Department Supply and Services Branch
5th Floor, 8 Metcalfe Street
Ottawa, Ontario Canada
K1A0S5.

FEATURE ANALYSIS OF GENERALIZED DATABASE MANAGEMENT SYSTEMS
A CODASYL Data Base Task Group Report May 1971
pub by Association for Computing Machinery

LEFKOVITZ, D - File Structures for Online Systems
pub by Spartan Books, 1969 Cat # 68-20673
available in the British Commonwealth from
Macmillan and Company, Ltd.
4 Little Essex Street
London, W.C. 2.

KNUTH, D.E. - Sorting and Searching
Vol 3 of The Art of Computer Programming
pub by Addison Wesley, 1973 Cat # 67-26020.

LONDON, K.R. - Techniques for Direct Access
pub Auerbach, 1973 Cat # 72-89103.

STONE, H.S. - Introduction to Computer Organization
and Data Structures
pub McGraw-Hill, 1972 Cat # 75-167560.

WAGNER, R.E. - Indexing Design Considerations
IBM Systems Journal, Vol 12, #4 1973 pp 351-367.

SENKO, M.E., ALTMAN, E.B., ASTRAHAN, M.M., FEHDER, P.L.
IBM Systems Journal, Vol 12, #1 1973 pp 30-93.

McKEEMAN, W.M., HORNING, J.J., WORTMAN, D.B.,
A Compiler Generator
pub by Prentice-Hall, 1970 Cat # 76-117205.

SHARE Secretary Distribution
"Data Base" A Users View
SSD # 237 June 30, 1973 pp 78-117.

SIBLEY, E.H., TAYLOR, R.W.
A Data Definition and Mapping Language
Communications of the ACM, Vol 16, #12 pp 750-759.

HOUSTON, G.B., Trillion Bit Memories
Datamation, Vol 19, #10 1973 pp 52-57.

COMPUTERWORLD, STC Announces 3330 Compatibles
October 31, 1973 page 1.