DATA BASE TASK GROUP REPORT TO THE CODASYL PROGRAMMING LANGUAGE COMMITTEE

In October 1969, the CODASYL Programming Language Committee authored the publication by the Data Base Task Group of its report containing proposals for a Data Description Language and a Data Manipulation Language. Section 2 of that report is reprinted here. It introduces the concepts embodied in the Data Description and Data Manipulation Languages. Extracts from the Preface and the Introduction to the report have also been included.

The complete Data Base Task Group Report contains three additional sections detailing the syntax and semantics of the Data Description Language and of the Data Manipulation Language. The complete report is available at $4 prepaid from the Association of Computing Machinery. Readers are reminded that the Data Base Task Group Report does not reflect approved language specifications.

1. EXTRACT FROM PREFACE AND INTRODUCTION

This report has been prepared by the Data Base Task Group which is an ad hoc committee of the CODASYL Programming Language Committee. The report details the recommendation of the Data Base Task Group to its parent committee. [It consists of a proposal for a Data Description Language and a Data Manipulation Language.]

The Data Description Language is a language for describing a database. The Data Manipulation Language is language which, when associated with the facilities of a host language such as COBOL, PL/1, ALGOL, JOVIAL, FORTRAN. . ., allows manipulation of databases described by the Data Description Language.

The specification of separate Data Description and Data Manipulation Languages is significant in that it allows databases described by the Data Description Language to be independent of the host languages used for processing the data. Of course, for this to be possible, the host language processors must be able to interface with such independent descriptions of data.

The objective of the Data Base Task Group in developing its proposals was to make it easier and more efficient for *programmers* to store and retrieve data in secondary storage by providing features which:

- 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.
- allow more than one run-unit to concurrently retrieve or update the data in the database.
- provide and permit the use of a variety of access methods against an entire database or portions of a database.
- provide protection of the database against unauthorized access of data and from untoward interaction of programs.
- allow the user to plan and implement his system as if he had a virtual memory at his disposal.
- provide the Data Base Manager with the capability to control the physical placement of data.
- allow the declaration of a variety of data structures ranging from those in which no connection exists between data elements to network structures.
- allow the user to interact with the data while being relieved of all of the mechanics of maintaining the structural associations which have been declared.
- allow programs to be as independent of the data as current techniques will permit.

These features, then, provide both generality and flexibility and allow the building and manipulation of structures as complex as necessary for a given application.

Such complexity is not spurious. It allows concise and efficient problem-modeling in a way that no single technique could. In any one application, many different techniques can be used effectively to help in solving the various aspects of the overall problem. The Data Base Task Group approach allows the use of appropriate techniques for each aspect, while providing the discipline for interlocking the various parts into a unified whole.

It is important to note that the Data Base Task Group's proposals are oriented to the programmer. Specifically, the DML is a language for programmers. It is *not* an inquiry language intended for the non-programmer.

The Data Base Task Group recognized the need for an inquiry language, but considers it essential that such a language be capable of interacting with the same databases that are developed for routine, day to day processing. Because of this, specification of an inquiry language has been deferred in favor of providing the tools required by programmers to build and maintain "databases that are available to, and suitable for, processing by multiple applications." The proposals in this report, however, provide a solid foundation for building an inquiry system.

2.1 THE DATA DESCRIPTION LANGUAGE (DDL)

The DDL is the language used to declare a SCHEMA. A SCHEMA is a description of a DATABASE, in terms of the names and characteristics of the DATA-ITEMS, RECORDS, AREAS, and SETS included in the database, and the relationships that exist and must be maintained between occurrences of those elements in the database.

Due to technical problems, Mr. Paul Siegel's article on a similar subject is being delayed to a future publication.

The term *DATA-ITEM* has the same meaning as in COBOL and may be a group data-item or an elementary data-item.

A *RECORD* is essentially a group data-item and may contain zero, one or more specific data-items. There may be an arbitrary number of occurrences in the database of each record-name specified in the schema for that database. For example, there would be one occurrence of the record-name PAYROLL-RECORD for each employee. This distinction between actual occurrences of a record and the "type" or name of the record is an important one.

A *SET* is a name collection of records. Each set-name specified in the schema must have one record "type" declared as its OWNER and one or more record "types" declared as its MEMBER records. There may be an arbitrary number of occurrences in the database of each set-name specified. 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."

An *AREA* is a named sub-division of a database and may contain occurrences of one or more record and sets. However, all occurrences, or portions of areas, may be opened by a run-unit with USAGE MODES which permit, or do not permit, concurrent run-units to open the same area or portion of an area. An area may be declared in the schema to be a TEMPORARY AREA. The effect of this is to provide a different occurrence of the temporary area to each run-unit opening it. At the termination of the run-unit, the storage space involved becomes available for re-use.

A *DATABASE* consists of all the record occurrences, set occurrences and areas which exist in secondary storage and are controlled by a specific schema.

2.2 THE DATA MANIPULATION LANGUAGE (DML)

The DML is the language which the programmer uses to cause data to be transferred between his program and the database.

The DML is not a complete language by itself. It relies on a host language to provide a framework for it and to provide the procedural capabilities required to manipulate data in primary storage. The relationship between the DML and its host language is discussed in Section 2.5.

2.3 RELATIONSHIP BETWEEN THE DDL AND DML

The relationship between the DDL and DML is the 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 the host language; that is, they are implicit procedures which may be invoked by the execution of DML statements.

2.4 RELATIONSHIP BETWEEN DDL/DML AND THE DATA BASE MANAGEMENT SYSTEM

Though this report is not a complete specification for a Data Base Management System, it may be helpful to an understanding of the DDL/DML to conceptualize a complete system. The system presented is for pedagogic purposes only and is illustrated by Diagram 1.

The numbered arrows in Diagram 1 trace a call for data by user program 1, and are explained in the following. Calls for data by other user programs are handled concurrently by the Data Base Management System (DBMS), but this is not shown in the diagram.

CONCEPTUAL DATA BASE MANAGEMENT SYSTEM

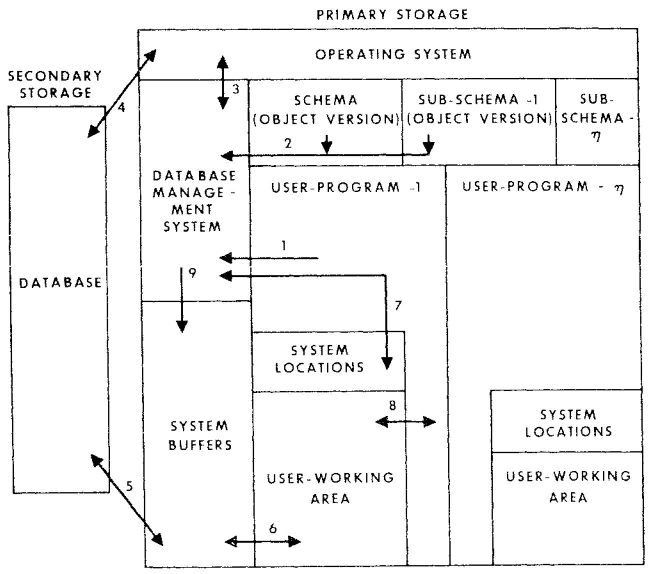


DIAGRAM 1

'1' a call for data by a user program to the DBMS. All calls for the services of the DBMS are made in the DML.

'2' the DBMS analyzes the call and supplements the arguments provided in the call itself with information contained in the object version of the schema for the database, and in the object version of the sub-schema invoked by the user program originating the call. The schema describes the database in terms of the characteristics of the data as it appears in secondary storage and the implicit and explicit relationship between data elements. The sub-schema is a subset of the schema. It describes the data known to the program invoking it in the form in which the DBMS makes it available, and expects to find it, in that program's USER WORKING AREA (UWA). (In this conceptual system it is assumed that the object version of the sub-schema contains only the differences from the schema and is not complete itself. The source form of the schema and sub-schema are written in the DDL. A more detailed discussion of the concept of schema and sub-schema appears in Section 2.6.)

'3' on the basis of the call for its services and information obtained from the object version of the schema and sub-schema, the DBMS requests physical I/O operations, as required to execute the call, from the Operating System.

'4' the Operating System interacts with secondary storage.

'5' the Operating System transfers data between secondary storage and the System Buffers.

'6' the DBMS transfers data, as required to fulfill the call, between the System Buffers and the UWA of the program originating the call. Any required data transformations between the description of the data as it appears in secondary storage, and the description of the data as it appears in a program's UWA, are handled by the DBMS.

'7' the DBMS provides status information to the calling program on the outcome of its call. The information provided is: Currency status information, Error Status Condition Codes, area-name, record-name, and area-key.

'8' data in a program's UWA may be manipulated as required, using the facilities of the host language.

'9' the DBMS administers the System Buffers. The System Buffers are shared by all programs serviced by the DBMS. User programs interact with the System Buffers entirely through the DBMS.

2.5 RELATIONSHIP BETWEEN THE HOST LANGUAGE AND THE DML

A users application program is written in a mixture of host language statements and DML statements. The DML provides the ability to interact with the database in that it is the language interface with the DBMS. All calls to and from the database to retrieve data, to add new data, to modify existing data, or data relationships, and to delete existing data or data relationships are written in the DML.

As a result of the successful execution of a call for data included in the database, the data requested is delivered to the UWA of the calling program and may then be referenced and manipulated using the facilities of the host language. To add new data or return modified data to the database, the host language is used to initialize the appropriate values in the UWA and the DML is used to call on the DBMS's services.

The host language, then, is the language used to manipulate data in primary storage. The host language processes or provides the framework in which the DML functions. And the DML is the interface language with the database.

DML and host language statements are intimately mixed in an application program. Indeed, the distinction between them is conceptual. The two languages may be mixed freely and there are no special "enter" or "exit" requirements from one language to the other. Thus, from the programmers point of view, he is using a single language – a language which has the combined capabilities of the host language and DML.

2.6 CONCEPT OF A SCHEMA AND SUB-SCHEMA

A schema consists of DDL statements and is a complete description of a database. It includes descriptions of all of the areas, set occurrences, record occurrences and associated data-items, as they exist in secondary storage.

A sub-schema also consists of DDL statements. It, however, need not describe the entire database but only those areas, sets, records, and data-items which are known to one or more specific programs. Further, it describes them in the form in which they are known to those specific programs.

The concept of separate schema and sub-schema allows the separation of the description of the entire database from the description of portions of the database known to individual programs. The concept is significant from several points of view:

- An individual programmer need not be concerned with the universe of the entire database but only with those portions of the database which are relevant to the program he is writing. Since the database may contain data which is relevant to, and shared by, multiple applications, this may be important to the ease of writing, debugging and maintaining programs.
- A program is limited to the subset of the schema that is known to it via its sub-schema. To a large extent, this automatically ensures the privacy and integrity of the rest of the database from that program.
- A measure of data independence is provided for programs in that certain changes may be made to the schema for the database – and the database adjusted accordingly – without affecting existing programs using that data. This is possible because the sub-schema may vary in certain important aspects from the schema of which it is a subset.

A sub-schema may differ from a schema of which it is a subset in several important respects:

- At the data-item level:
  a. The characteristics of data-items may be different.
  b. Privacy locks may be changed.
  c. Entire entries for specific data-items may be omitted.
  d. New data-items may be included where such data-items are the names of group data-items whose elementary terms are included in the schema; or where they are names of data-items which are defined as part of a group data-item included in the schema.
- At the record level:
  a. Entire entries for specific records may be omitted.
  b. Privacy locks may be changed.
  c. Record occurrences included in specific areas or portions of areas may be omitted, while other record occurrences of that record name are included.
- At the set level:
  a. Entire entries for specific sets may be omitted.

b. Privacy locks may be changed.

- At the area level:

  a. Entire entries for specific areas may be omitted.

  b. Privacy locks may be changed.

A sub-schema must, however, be a consistent and logical subset of the schema from which it is drawn.

The following additional points are also important to an understanding of the concept of the schema and sub-schema:

- An object version of the source code schema may be "compiled" independently of any user program or any sub-schema.
- Object versions of a source code sub-schema may be "compiled" independently of any user program and stored in a library.
- An arbitrary number of sub-schema may be declared on the basis of any given schema.
- The declaration of a sub-schema has had no effect on the declaration of any other sub-schema and sub-schema may overlap one another.
- Each sub-schema must be named.
- A user program invokes a schema or sub-schema by name in its DATA DIVISION.
- The same schema or sub-schema may be named in the DATA DIVISION of an arbitrary number of programs.
- Only the areas, records, data-items, and sets included in the schema or sub-schema invoked by a program may be referenced by that program.

2.7 THE USER WORKING AREA (UWA)

Conceptually, the UWA is a "loading and unloading zone" where all data provided by the DBMS in response to a call for data is delivered and where all data to be picked up by the DBMS must be placed. Each program has its own UWA. The data in the UWA of a program is not disturbed except in response to a DML call or by the user program's host language procedures. There is no implication that UWA locations are contiguous.

The UWA is set up by the DBMS in accordance with the schema or sub-schema invoked in the DATA DIVISION of a program. Each data-item named in the schema or sub-schema invoked will be assigned a location in the invoking program's UWA and may be referenced by its declared name. Where a sub-schema is invoked, the PICTURE AND USAGE of data-items included in the sub-schema and, therefore, in the UWA, may differ from the PICTURE AND USAGE of those items in the schema for the database. The DBMS is responsible for required conversions. Data-items included in the database, but not in the sub-schema invoked, are not assigned space and may not be referenced.

The DBMS must also provide for a number of "System Communication Locations." Such locations are used for run-unit/system interaction and are assigned space by the DBMS. They are: AREA-KEY; AREA-NAME; RECORD-NAME; ERROR-STATUS; ERROR-SET; ERROR-RECORD; and ERROR-AREA.

2.8 THE "DATA MANAGER" FUNCTION

In an environment where a database includes data which is shared by many user programs, it becomes necessary for the schema and sub-schema to be developed centrally. In such a shared environment, a database, is, in a sense, a compromise between the needs of the various user programs; and the proper trade-offs can only be made centrally by a "Data Manager."

On the basis of information as to the data required by individual programs, statistics on usage of data, and response requirements, the "Data Manager" must make decisions; for example, on whether to repeat data redundantly and on what relationships to build into the database; and, based on these decisions, declare the areas, records, data-items, and sets required; and, if necessary, restructure the database.

It is assumed here that the "Data Manager" is a specialist person or group. Ideally, the function of the "Data Manager" would be performed by the System. The Data Base Task Group considered this but concluded that the techniques required are not yet sufficiently well developed for it to propose a standard for the information which would be required as input. Thus the DDL does not include any facility to provide usage and response statistics to the System. This does not preclude the "Data Manager" from using "programmed aids" in doing his job.

2.9 CHARACTERISTICS OF SETS

The following is relevant to an understanding of the concept of a set:

- Sets are ordered logical collections of associations of records.
- An arbitrary number of sets may be declared in a schema.
- Each set must be named and must have one owner record and one or more member records declared for it in the schema. (This does not apply to sets specified as DYNAMIC -- See Section 2.15.)
- Any record may be declared in the schema as a member record of one or more sets.
- Any record may be specified as both an owner record in one or more sets and a member record in one or more different sets.
- Each occurrence of a set includes one occurrence of its owner record. In fact, the existence of the owner record in the database is a condition of the existence of the set occurrence and distinguishes that set occurrence from all other occurrences of that set-name. Where, however, the owner record occurrence is itself not unique, other information is required to uniquely identify the set occurrence.

- A set occurrence which contains only an occurrence of its owner record is known as an "empty set."
- In addition to an occurrence of its owner record, each set occurrence may include an arbitrary number of occurrences of each of the member records declared for it in the schema.
- If a record which participates in a set is addressed, the program is able to find:
  a. its successor record occurrence in that set
  b. its predecessor record occurrence in that set
  c. its owner record occurrence in that set.

2.10 ORDERING OF SETS

Each set named in the schema must also have a SET ORDER specified for it. The effect of this is to cause the DBMS to control in accordance with the "set order" specified, the logical order of the member record occurrences within each set occurrence. The logical order of the member records of a set is completely independent of the physical placement of the records themselves. Thus, the same member record occurrences could participate in occurrences of two different sets and be ordered differently in each of those sets.

The member record of each occurrence of a given set may be ordered in one of several ways:

- SORTED in ascending or descending sequence based on specified keys. The keys specified may be data-items in each of the member records, the member records names or their relative addresses, or any combination of these.
- In the order resulting from inserting new member record occurrences into the set:
  a. FIRST, that is, immediately after the owner record occurrence. This is equivalent to LIFO.
  b. LAST, that is, immediately before the owner record occurrence. This is equivalent to FIFO.
  c. NEXT, PRIOR, that is, after or before another record which is selected by the program adding or inserting the record in the set.

2.11 AUTOMATIC AND MANUAL MEMBERSHIP IN SETS

The membership of a record in any specific set (that is, in all occurrences of that set) may be declared in the schema as either AUTOMATIC or MANUAL. A record may be an automatic member in some sets and a manual member in other sets.

A record declared to be an automatic member of a set is an unconditional member of an occurrence of that set. Whenever an occurrence of such a record is added to the database, it will be logically inserted into (that is, made a member of) the appropriate occurrences of *all* the sets in which it has been declared as an automatic member. It will remain a member of those sets (but not necessarily in the same occurrences of those sets) until it is deleted from the database.

A record declared to be a manual member of a set is a conditional member. Such a record will not be logically inserted into any occurrence of the sets in which it has been declared as a manual member except as a result of the execution of an explicit INSERT command by a run-unit. It may be logically removed from any of the specific occurrences of sets in which it participates as a member. It remains in the database and is still accessible, though not through the sets in which it no longer participates.

2.12 SET "MODE"

The set concept may be implemented in several different ways, each of which involves different trade-offs between time and space. The particular trade-off which is appropriate depends on the characteristics of the processing to be performed. Since this varies from set to set and cannot be forecast at the time the DBMS is developed, the DDL allows selection, from among the approaches provided by the System, of the approach to be employed for any given set.

The DDL provides for an arbitrary number of set "modes." Two are named and specified in this report. Other set "modes" are at the discretion of the implementor of the DBMS.

The two modes specified are CHAIN and POINTER ARRAY. They correspond to embedded and non-embedded pointers and are described in Sections 2.13 and 2.14, respectively.

2.13 SETS DECLARED AS CHAINS

One method of implementing the set mode CHAIN is by means of lists in which each element in the list points to the next element. Since a set occurrence consists of one owner record occurrence and "n" member record occurrences, the owner record of a chain contains a pointer to the first member record in the set which, in turn, points to the next member record and so on until the last member record points back to the owner record. This is illustrated in Diagram 2 which is a representation of a set occurrence with two member records.

A chain then has the property that given any particular record in the set all other participating records may be accessed by following the pointers or "links" in the chain. A chain is also an efficient "routing device" or "junction box" in that having accessed a given record, it contains pointers to all other chains in which it participates, and any of these pointers may be followed.

Chains are always processable in either direction from any given record in the chain. However, the linkage provided between the records in a chain is only in the NEXT direction unless the optional clause "LINKED TO PRIOR" is used.

When this caluse is used, additional links in the reverse (that is, the PRIOR) direction are also provided. Diagram 3 is a representation of a chain which is "LINKED TO PRIOR."

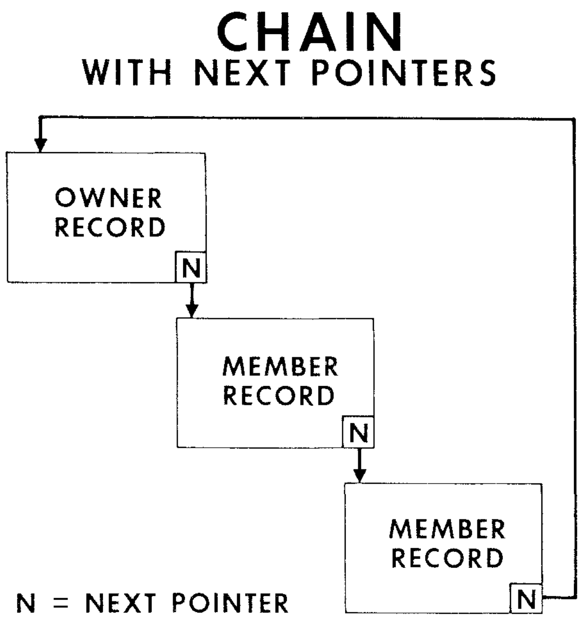


DIAGRAM 2

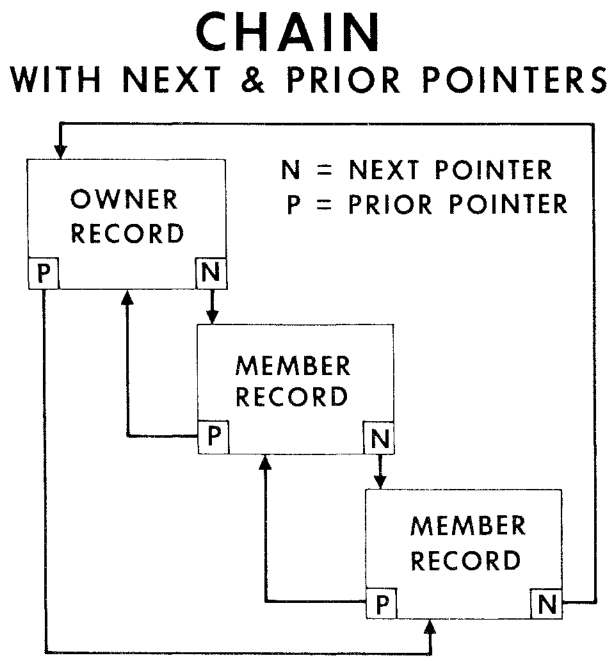


DIAGRAM 3

In addition, the occurrences of any of the member record "types" specified for a set may be declared to be "LINKED TO OWNER." This causes each of the member record occurrence, enabling the owner record to be accessed directly. Diagram 4 illustrates this.

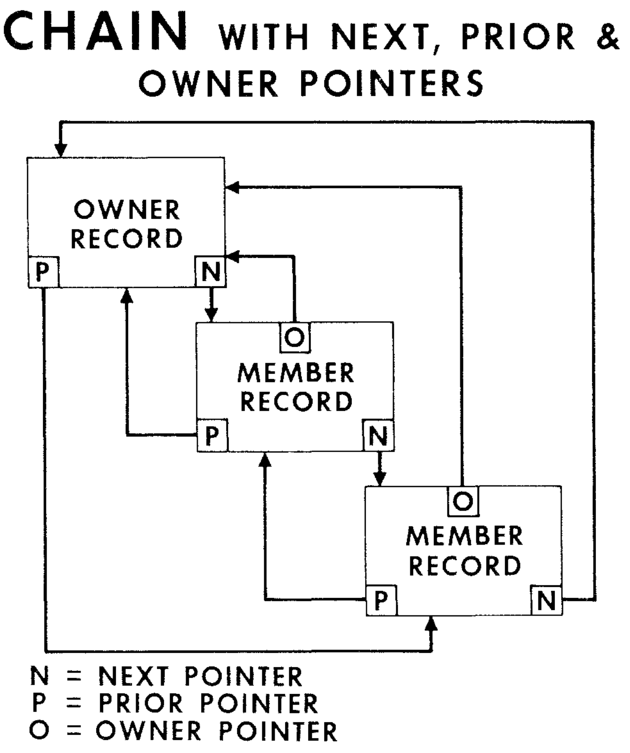


DIAGRAM 4

The unique identifiers assigned by the System to every record occurrence in the database are used as pointers. Space for a minimum of one pointer (the "NEXT" pointer) is required in each record and must be assigned by the System for each chain in which a record participates as owner or member. Additional pointers and space are required if the chain is PRIOR processable or members are "LINKED TO OWNER."

The unique identifiers mentioned here are known as DATABASE-KEYS and are discussed in Section 2.30.

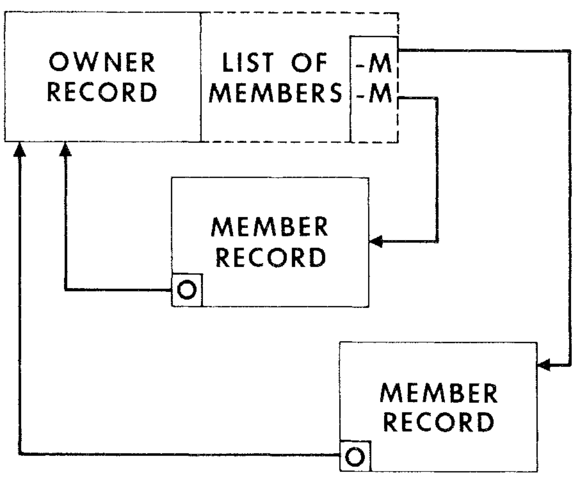
2.14 SETS DECLARED AS POINTER ARRAYS

A pointer array is the functional equivalent of a chain.

In sets declared as pointer arrays, member records do not contain pointers to each other but only to their owner record occurrences. Each owner record occurrence, however, is associated with a list of all of its member record occurrences. In effect, the "NEXT" pointers contained in the member records of a chain are collected together in a list. Since this list of pointers may be logically ordered, any "SET ORDER" declared for a set is equally applicable to chains and pointer arrays. In addition, since the list of pointers may be processed in either direction and the member records point to their owners, a pointer array is always NEXT and PRIOR processable and it is also "LINKED TO OWNER." Diagram 5 is a representation of a pointer array set.

The list of member records developed for pointer arrays allows some logical operations to be performed on the mem-

POINTER ARRAY



O = OWNER POINTERS
M = MEMBER POINTERS

DIAGRAM 5

ber records listed, without the necessity of accessing the records themselves. Thus, for example, where membership in a set has a known meaning, logical AND and OR operations can be performed on the members of two or more sets or set occurrences without accessing any of the member records themselves.

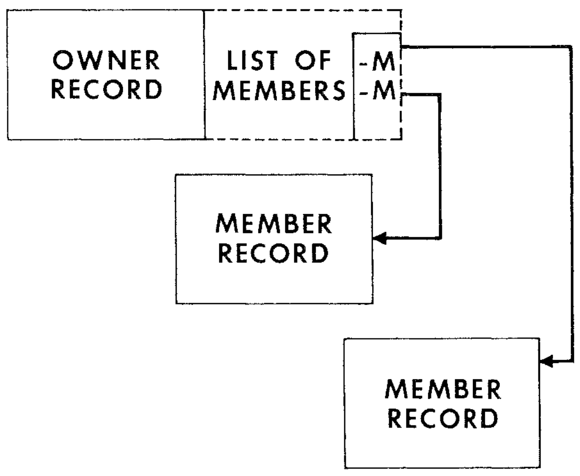
2.15 SETS DECLARED AS DYNAMIC POINTER ARRAYS

A pointer array may optionally be declared DYNAMIC. A dynamic pointer array differs from an ordinary pointer array in several important respects.

- It may not have any member records predeclared for it.
- It is dynamic in the sense that any record occurrence in the database, other than occurrences of the owner record specified for a given set, is a potential manual member of such a set. It may be made an actual member by means of executing an explicit command inserting it; or may be removed from membership by executing an explicit command removing it.
- It is temporary in that all such set occurrences created by a run-unit are destroyed at the termination of the run-unit. That is, all pointers to the member records associated with the owner record are nulled, but the participating records themselves (members and owners) remain in the database and remain accessable. If, however, any of the records involved were assigned to "temporary areas," they would be deleted from the database. This latter point follows the rules for areas declared to be temporary.
- The System does not provide a pointer to their owner in member record occurrences.

Diagram 6 is a representation of a pointer array set declared to be "dynamic."

DYNAMIC POINTER ARRAY



M = MEMBER POINTERS

DIAGRAM 6

2.16 MAINTENANCE OF SET RELATIONSHIPS

The establishment and maintenance of all relationships between records, specified by means of declaring sets in the schema, is a System responsibility.

Such maintenance is required whenever:

- A record which has been declared as an owner or member in one or more sets is added to or deleted from the database.
- A record is explicitly inserted or removed from a set.
- A record is modified in a way which changes its logical position in the set.
- A record is modified in a way which changes the set occurrence in which it participates.

Programmers are not involved in the mechanics of this process but must initialize with appropriate values those data terms which are required by the System to perform its functions. Such data-items are declared in the schema.

2.17 INDEXED SETS AND SEARCH KEYS

Any set declared to be SORTED may also be declared to be INDEXED. This causes the System to build an index on the basis of the sort keys specified for each occurrence of that set. No control is provided in the DDL over the index developed; however, the index may be named. It is assumed that implementors of the System will provide for such control.

An arbitrary number of SEARCH KEYS may be declared for a set regardless of whether it is sorted or not. The arguments for such SEARCH KEYS must be data-items included in the member records of the set. The declaration of a SEARCH KEY causes the System to develop some form of indexing for each set occurrence in which member records, for which SEARCH KEYS have been specified, participate. The term "indexing" as used in this paragraph means any technique which does not involve a complete scan of the member records involved. It is not restricted to an "index" in the usual sense. Optional control over the type of indexing developed is provided for in the DDL.

Where a set has been declared to be INDEXED, or SEARCH KEYS have been specified for its members, functions or procedures which require a search to be performed on the basis of any argument for which indexing exists will automatically employ the available indexing.

2.18 THE REPRESENTATION OF STRUCTURES

As stated in Section 1.2, one of the objectives of the Data Base Task Group is "to allow data to be structured in the manner most suitable to each application. . .without requiring data redundancy."

To achieve this, it must be possible to represent in secondary storage the associations between data elements which logically exist relative to the performance of particular functions; that is, to represent data structures of varying complexity as storage structures.

The DDL, as described in Section 3, provides the facility to declare such structures through the medium of the set. The set is, in effect, a building block which allows various data structures to be built. Sections 2.19, 2.20, and 2.21 show how the following data structure may be represented by sets.

- Sequential Structures
- Tree Structures
- Network Structures

In addition, the absence of structure may be represented by declaring records in the schema which do not participate in sets.

2.19 REPRESENTATION OF A SEQUENTIAL DATA STRUCTURE

A sequential data structure is one in which each element in the structure, except the first and last, is related to the element proceding it and the element following it. A list is an example of a sequential data structure. As shown in Diagram 7, a list may be a one-way list, where each element points only to the next, a two-way list or a circular list.

Any single set is, in fact, a representation of a sequential data structure or list. It is always circular and may be specified as a one-way or two-way list.

SEQUENTIAL STRUCTURES

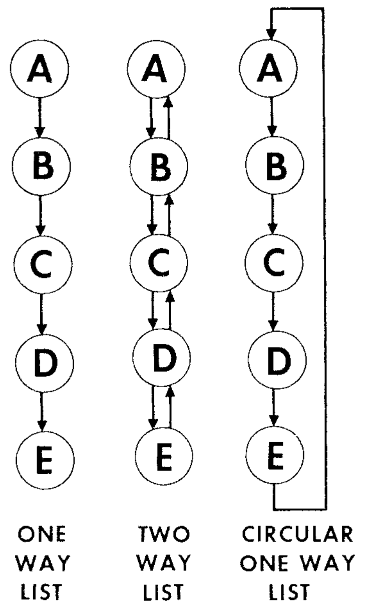


DIAGRAM 7

Diagram 8 is a set representation of a sequential data structure. It is also a diagrammatic representation for a set. In such a diagram of a set:

SET REPRESENTATION OF A SEQUENTIAL STRUCTURE

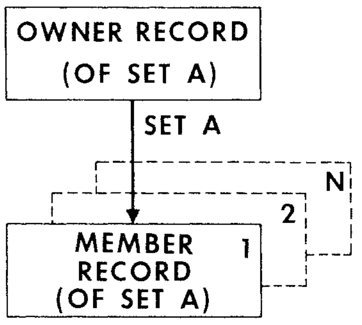


DIAGRAM 8

- The arrow points from the owner record "type" to the member record "type."
- There may be "n" occurrences of the owner record "type."
- There is a "one-to-n" relationship between the owner records and the member records. Thus, for each occurrence of the owner record, there may be "n" occurrences of the member record.