DESIGN AND IMPLEMENTATION OF AN EXPERIMENTAL NETWORK DBMS (ENDBMS)

Evangelos Liopiris

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirement
for the degree of Master of Computer Science
at Concordia University
Montreal, Quebec, Canada

September 1981

A B S T R A C T

DESIGN AND IMPLEMENTATION OF AN EXPERIMENTAL NETWORK DBMS (ENDBMS)

Evangelos Liopiris

This thesis describes the development of an Experimental Data Base Management System (ENDBMS) based on the Network approach. All the major features of a modern DBMS are considered and most of them such as schema, subschema, security, concurrency and recovery have been included in ENDBMS. Many concepts are borrowed from the well known CODASYL DBTG proposals.

During the 'installation' of ENDBMS the identity of a special user, the Data Base Administrator (DBA) is defined. All other users are installed by the DBA and are assigned a unique user number along with a password and a USER PROFILE. The later determines the user's data base access capabilities. A user accesses the stored data through his SUBSCHEMA, which is a 'disciplined' subset of the schema. A proper and consistent use of both subschemas and user profiles constitute the cornerstone of the overall security enforcement in ENDBMS.

The ENDBMS provides a Data Manipulation Language, which includes retrieve and update utilities for both record and set type data base operations. The ENDBMS can operate in a concurrent multiuser environment. The design of the concurrency and recovery scheme is affected by the file system supported by the host computer system.

An interactive editor facility is provided for error corrections during the translation period of any ENDBMS command.

To my father and mother.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

DATA-BASE MANAGEMENT SYSTEMS

1.1 INTRODUCTION

One of the most important topics today in computing is data-base technology. It seems to be a very disoriented field; differences in terminology, modeling and in implementation confuse the potential user who is faced with almost unanswerable questions such as whether to use the current data-base technology or wait for new developments. If he chooses to use a data base system, he is faced with the question of which type of system to choose. Such problems are found in any evolving technology, especially when it is associated with a fast developing industry such as computing. Data-base systems have evolved from nothing to becoming major topics of current interest. Top management of major operations have grown to appreciate the importance of their data-bases. Government regulatory agencies are already worrying about the implementation of privacy and freedom of information laws and their relation to data-banks.

During the fifties and the sixties the first data processing systems, using duplication of computerized data and procedures, managed to reduce the overall cost. But the problem of 'availability' of the data to different application programs or users gave rise to questions like:

- Why not integrate the data?
- Can we access this data through the current computer languages?
- Why not allow a high level language for ad hoc use of the integrated data?

The first question gave rise to the concept of generalized data base management systems, the second resulted in modifications to the existing conventional programming languages, and the third lead to the development of special query languages as an interface.

The industry congratulated itself for reducing data redundancy and improving its availability, but it also introduced the potential for disaster. The first problem with integration that arose was that the data now became more vulnerable to destruction through machine malfunction, personal errors or deliberate human tampering. The loss of quality in the integrated data (including total destruction) by any of these means may be considered a threat to the organizations, because data is one of its most valuable assets. Integrity and security techniques were therefore a necessity. [FRY-76] deals with the definitions common to data base technology and traces the evolution of data base management systems.

More and more prospective users of DBMS, even though they may not have studied the subject formally, have already formed a general impression - from articles in data processing magazines, news items, advertisements and similar

sources - as to what constitutes a data base management system. Their impression probably looks something like Figure 1.1

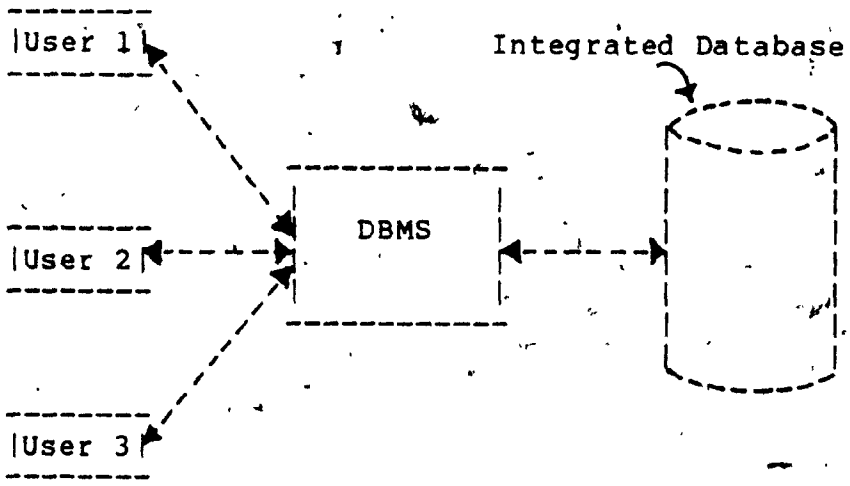


Figure 1.1 A user's general impression of a DBMS.

First of all there is the data base itself - a collection of data stored on disks, drums or other secondary storage media. Second there is a set of ordinary application programs which use this data base for retrieval or updates (update is used to mean Inserting, Deleting or Replacing). Additionally there could be a set of on-line users who interact with the data base from remote terminals and performe the above operations. Third there is 'something' which controls any kind of interaction with the data base, protects it and responds to all program or user requests, provides each application with its own view of the common data, implements various operators for retrieval and updating of data, and resolves interference among concurrent users. The term 'integrated' data base means that the data

base contains the data for many users. Every user is concerned only with just a small portion of it. Individual pieces of data may be shared by many different users. The term data base has been defined variously; two definitions by Date and Martin are given below:

(1). "A data base is a collection of stored operational data used by the application systems of some particular entrprise. Enterprise is simply any reasonably large-scale commercial, scientific, technical or other operation" .... "some examples are: manufacturing company, bank, hospital, university, government department". Any enterprise must necessarily maintain a lot of data about its operations. This is its operational data. The operational data for the enterprises listed above would probably include the following: Product data, Account data, Patient data, Student data or Planning data respectively. "Operational data does not include input or output data or any transient information. Input data refers to information entering the system from the outside world (from cards or terminals) and output data refers to messages and reports emanating from the system" [DATE-77].

(2). A data base may be defined as a "collection of interrelated data stored together without harmful or unnecessary redundancy to serve multiple

applications". The data are stored so that they are independent of programs which use the data; "a common and controlled approach is used in adding new data, modifying and retrieving existing data within the data base. The data is structured so as to provide a foundation for future application development". One system is said to contain a collection of data bases if they are entirely separate in structure [MART-77].

Martin in [MART-77] discusses the primary and secondary objectives of data base organization in Chapter 4 and Fry in [FRY-76] discusses the data availability, data quality, privacy and security, management control and data independence as major objectives of a DBMS and their relation to the overall functional architecture of the DBMS. Another thorough discussion on the objectives of a DBMS is given in [EVER-74].

1.2 DATA MODELS

"A data model is a pattern according to which data are logically organized. It consists of named logical units of data and expresses the relationships among the data as determined by the interpretation of a model of the world" [TSIC-77]. One of several data models can be used to represent the interpretation of a model of the world. The main difference among them is the manner in which they

represent certain relationships among the data. The data model constitutes the heart of a DBMS. The range of data structures supported in the data model critically affects all the other components of the system. In particular it dictates the design of the corresponding data manipulation language (DML). Each DML operation must be defined in terms of its effect on these data structures explicitly in case of the Data Base Administrator (DBA) where each refers to the data model (conceptual model) itself and implicitly in case of other users, where he refers to his own external model (derived from the conceptual model) - provided that external models are supported by the DBMS. For example, if a RELATIONAL system does not provide facilities for a 'join' operation over two relations then its DML cannot include commands related to this operation; similarly the way a NETWORK system implements the logical association between some record types greatly affects the capacity of its DML. There are three approaches to design of the data model and its accompanying data manipulation language:

1. Relational model
2. Hierarchical model
3. Network model

1.2.1 RELATIONAL MODEL

In this model the data is organized in a number of tables. Each table resembles a traditional sequential file, with rows of the table corresponding to records of the file and columns corresponding to fields of the records. Each of

these tables is actually a special case of the construct known in mathematics as a relation. The relational approach to data is based on the realization that files that obey certain constraints may be considered as mathematical relations and hence that elementary relation theory can be used to handle problems dealing with data in such files.

E.F Codd [CODD-70] was the first to give a rigorous definition for n-ary relations in the data base context and to emphasize their advantages for data independence and symmetry of access. Codd's paper introduced concepts which set the direction for research in relational data base management for several years to come. The paper defined a data sublanguage as a set of facilities suitable for embedding in a host programming language. He suggested that a standard logical notation (namely the first order predicate calculus), is appropriate as a data sublanguage for n-ary relations. The paper also explored the properties of redundancy and consistency of relations. An excellent introduction to relational concepts can be found in Date's text book [DATE-77]. An important aspect of the relational model is the normalization theory; it is based on a series of normal forms - first, second and third - which provide successive improvements in the update properties of a data base. A thorough treatment of normalization is given in [CODD-71] , [DATE-77] and [VASS-80]. Chamberlin in [CHAM-76] defines the essential concepts of the relational

data model and also discusses normalization, relational languages based on this model as well as the advantages and implementations of relational systems. At this point we can mention Tymshare's MAGNUM [MAGNUM] and IBM's APL [APL] as commercially available systems. A large number of experimental systems have been and continue to be developed at universities and similar institutions. In particular we mention System R [SYST-R] and INGRES [INGR-76]. A survey of currently available relational database systems and their highlights has been reported in [KIM-79].

1.2.2 HIERARCHICAL MODEL

In this model the data is represented by a simple tree structure. Every node of the tree stands for a record type. The record type at the top of the tree is usually known as the 'root'. In general the root may have any number of dependent record types, each of these may have any number of lower level dependents, and so on, to any number of levels. Every node in the tree, with the exception of the root, has one node related to it at a higher level and this is called its parent. No node can have more than one parent and each node can have one or more nodes related to it at a lower level. The dependent nodes, of the parent node, are called children; no child can exist without a parent node. Nodes at the end of the branches are called leaves. Such trees are used in both logical and physical descriptions. In the logical data descriptions they are used to describe

relations (associations) between record types. In the physical data organization they are used to describe sets of pointers and associations between entries. In [TSLO-76] it is stated that a hierarchical system is a DBMS which presents to the users of the system certain explicit views of the data base that are characteristic of the hierarchical data model and the characteristics of the data model are:

(1). There is a set of record types {R1,R2,...,Rn}

(2). There is a set of relationships connecting all record types, in one data structure diagram.

(3). There is no more than one relatonship between any two record types Ri and Rj. Hence, relationships need not be labeled.

(4). The relationship expressed in the data structure diagram, form a tree with all arcs pointing towards the leaves.

(5). Each relationship is 1:M and it is total - that is for every Rj record occurrence there is exactly one Ri record occurrence connected to it, if Ri is the parent of Rj in the definition tree.

Date in [DATE-77] states that the asymmetry of the data model (some record types are treated as superiors and other as dependents) is a major drawback of the hierarchical approach; he also discusses the undesirable properties of the hierarchical model as far as the storage operations are concerned, for a many-to-many relationship. For example the PARTS and SUPPLIERS, N:M relationship, in a hierarchical data base (with parent type PARTS) is represented by introducing redundancy, i.e., copies of the same SUPLIERS occurrences are stored as leave-occurrences to all PARTS

occurrences supplied by the same supplier. First, it is not possible to insert a new SUPPLIERS occurrence without introducing a special dummy PARTS occurrence until that supplier supplies some part. Second, if a PARTS occurrence is to be deleted we loose all information for all supliers who supply only this part because of the hierarchical nature of the physical organization. Third, in case we want to change some information about a particular supplier, e.g., his address, we must search all PARTS occurrences to find all occurrences of the supplier concerned; otherwise we may end up with a situation where the same supplier is located in Montreal, Vancuver and New York at the same time.

Hierarchical systems have been available and well accepted for a long time. Examples are [BLEI-76], [MARS-VI], and IBM's Information Management System [IMS], which is one of the most widely used of commercially available systems and is at least partially responsible for the importance of the hierarchical data model, on which it is based. J.D. Ullman [ULLM-80] in Chapter 8 of his textbook examines the features of the IMS system.

1.2.3 NETWORK MODEL

In the Network model the data (as in hierarchical) is represented by records and links. However, a network is a more general structure than a hierarchy because a given record occurrence may have any number of superiors, as well

as any number of immediate dependents, as opposed to a maximum of one in the hierarchical model. The network approach thus allows us to model a many to many correspondence more directly than does the hierarchical approach. An N:M association between two record types Ri and Rj is implemented as two 1:M associations by introducing a new record type Rc so that Ri to Rc and Rj to Rc are 1:M associations. We may liken the data model to files of records and links. The internal structure of this file (maintaining the declared associations between different record types) is more complex than in the hierarchical case.

Date in [DATE-77] discusses the symmetry of the model by using two inverse questions using a PARTS-SUPPLIERS network data base. He also examines the ambiguity which exists in an N:M association between Ri and Rj, as to which occurrence will be used as the entry point (an Ri occurrence or an Rj occurrence) for some queries such as: "find out if the part P2 is supplied by the supplier S3". One way to answer this question is to locate the S3 record occurrence and then follow its chain (path) to find out if there is any P2 record occurrence in it, but an alternate way is to locate the P2 record occurrence and then follow its own chain to search for the S3 record occurrence. Date also examines the storage operations Insert , Delete and Update. A deletion of a record occurrence of network data base does not cause any triggeered deletions of other record

occurrences, but implies destruction of its logical (and physical) association with other record occurrences. An insertion of a new record occurrence might require manual or automatic physical association. The most important example of a network system is provided by the proposals of the CODASYL Data Base Task Group (DBTG), [DBTG-69] and [DBTG-71]. Several commercially available systems are based on these proposals, among them UNIVAC's DMS 1100 [DMS-1100] and Honeywell's Integrated Data Store-IDS [IDS]. CODASYL proposals are discussed in section 1.3.

1.2.4 COMPARISONS BETWEEN THE THREE MODELS

In this section we will attempt to point out the most important features, as well as the advantages and disadvantages of the Relational, Hierarchical and Network approach to data base design.

A crucial feature of the relational data structure is that associations between types (rows) are represented solely by data values in columns drawn from a common domain. It is a characteristic of the relational approach, in fact, that all information in the data base (both 'entities' and 'associations') is represented in a single uniform manner, namely, in the form of tables. This charateristic is not shared by the hierarchical and network approach. The relational model seems to be the simplest but simplicity is not the end of the story. In any DBMS the data manipulation

language is more important thing from the user's point of view rather than the data representation. At the user interface level, no particular access path is 'preferred' over any other. However the DBMS must choose a path which may or may not be the best. The user interface is independent of the means by which data is physically stored. The uniformity of the data representation leads to uniformity in the operator set where one operator is needed for each function. This contrasts with the situation, with more complex structures, where information may be represented in several ways and hence several sets of operators are required. In the relational model the basic retrieval operator is "get next where" which will fetch the next row of a table (relation) satisfying some specified condition. 'Next' is interpreted relative to the current position. The 'where' part of the operator may be a boolean combination of conditions that the field values of the requested record (row) must satisfy. However this retrieval operator is not unique to the relational model since the same operator can be implemented in the network model by considering only the path which links all the occurrences of a record type. Let us consider the storage operations and then we will look at the data manipulation language based on relational algebra.

There are three basic storage operations, namely the Insert, Delete and Replace (our terminology); to insert a

new row (record occurrence) in a relation (record type), to delete one row (occurrence) or to replace (change) a portion of a row with something new. These operations, however, need special consideration in case the relational DBMS supports any kind of 'vituality' at either schema (DBA's view of the data base) or subschema (user's view of the data base) level. A virtual row of a relation is one that it is not actually stored in the data base the way it is viewed but it is generated, probably, from more than one other 'stored' row. For example let us assume a relation which is a projection of a join of two relations; obviously the insert operation will attempt to create partially filled stored row(s) when it operates upon such a virtual relation, while the delete operation will attempt to delete more information by deleting entire stored row(s); therefore the operators of Insert and Delete are to be prohibited for 'virtual' relations. The same considerations will apply to network models that allow any virtuality. The question of whether or not we will allow a Replace operation upon a 'virtual' relation, has to be examined too. The insert and delete operations are straightforward as far as the relational model is concerned with no additional requirements, e.g., creation or destruction of chains (links). In the hierarchical model an insertion of a new record occurrence requires immediate storage of any bearing information (e.g., being dependent of some superior occurrence). For insertion in the network model a record

occurrence can be stored in the data base and if the corresponding record type belongs to some sets the bearing 'association' is 'stored' (automatically or manually). The removal of a record occurrence causes the following: In the hierarchical model an occurrence deletion triggers the deletion of all of its dependents (related information). In the network model an occurrence deletion implies deletion of the corresponding bearing information, but does not require the deletion of any other record occurrence. Thus association among record occurrences is removed without any deletion of the record occurrences.

Some specific advantages of the hierarchical approach are: It is a simple data model and the user needs relatively few commands to master it. Because of the constraints on the types of relationships allowed, it can allow an easier implementation than other, more complex structures.

The following are some specific disadvantages [TSI-LOC] related to hierarchical systems.

(1). The imposed restrictions sometimes force an unnatural organization of the data. As an example consider the N:M relationship STATE - COMPANY in a single hierarchical definition with root the STATE, then the COMPANY record occurrences would have to be repeated under each state in which the companies are registered. This might lead to considerable duplication and hence to a storage space problem. Again one can imagine the results of the deletion

of a COMPANY occurrence. The N:M relationship does not create such problems in the network model.

(2). The strict hierarchical ordering make operations such as insertion and deletion quite complex. Users must be very careful with the delete operation because of the 'loss' of information.

(3). Symmetrical queries sometimes cannot be answered easily in a hierarchical system. Therefore the structure of the data base may tend to reflect the needs of the application. Finally we can say that the hierarchical model fits better to naturally hierarchically structured information.

Let us now consider the relational algebraic operations, SELECT, PROJECT and JOIN. The SELECT operation selects all rows of a relation which satisfy some specified condition. The same can be implemented in a hierarchical and network model, too. PROJECT is an operator which constructs a 'vertical' subset of a relation (one or more entire columns). Again this can be implemented in the network model by using the stored record types; this is easier in a network model which supports subschemas. A user, for example, can define such a 'subset' in his subschema provided that he has appropriate access capabilities. The JOIN is based on at least one common domain (field) of two relations (record types). Such an operation can be performed in a network model if a set

exists between the record types to be 'joined'.

The term 'compatible' record types is used in ENDBMS to provide a user view with an ability similar to the join operation of the relational model; a defintion for the term is given in section 2.3.4. We prove that such 'join' operation is implemented in our subschema and we also claim that this can be a feature of any network system which supports subschemas. An extenssive comparison of relational and network approaches is given in [MICH-76].

1.2.5 WHY USE A NETWORK MODEL ?

We do recognize the complexity of the network approach at the data base defintion level, the 'link' or data-set implementation at the physical structure level and at the user interface level. Obviously the user in order to retrieve information based on some complicated queries oust be aware of the declared 'links'. We believe that only the definition of a user subschema might require professional or semi-professional experience. Once this is done a network DML becomes very simple, with commands executed either against the schema or (some) subschema definiton. Also most of the network queries are simplified. Another adavntage is speed with respect to the relational model, since the 'joins' are already made and the 'projections' are predefined. The network model also provides the flexibility and freedom of inserting and deleting 'associations' using

sets. This makes the network model capable of reflecting a dynamic model of the real world. In an environment where new entities or associations are inserted quite frequently it is better to use the network model. A proper record type design may avoid a lot of data redundancy which may be required in a relational model. On the other hand there is a price to pay for the flexibility of the data base which is in the form of the storage needed to implement 'links' of a network model.

1.3 CODASYL DBTG PROPOSALS

The acronym DBTG refers to the Data Base Task Group of the CODASYL Programming Language Commitee (PLC). The PLC is the body responsible for development of the COBOL language; its activities are documented in the COBOL journal of Development which is published every two or three years and which serves as the COBOL language specification.

In this section we will summarize the main features of the CODASYL proposals. A complete description of CODASYL specifications is given in [DBTG-69] and [DBTG-71]. Martin, Date and Tsichritzis describe the DBTG proposals in their text books: [MART-77], [DATE-77] and [TSIC-77]. The DBTG proposals, though controversial to some extent, are extremely important. The DBTG report has been the basis of several commercial systems and the ENDBMS philosophy borrows many concepts from the DBTG proposals.

1.3.1 DBTG SET CONCEPT.

The set concept constitutes the most important single aspect of the DBTG data model. The set concept can be used to construct the hierarchical and the network data models. First we define some DBTG terminology and then we look at the set structure for the network model. A record type is a named collection of some data items related to some attributes of an entity set of the world and expresses an association among those attributes. A data item can be a single data item or a data aggregate. An instance of a record type is called record occurrence. A CODASYL set type is in essence, a named two level tree where the upper level consists of only one node and is called the owner type and the lower level consists of one or more nodes (record types) which are called the member type(s).

In general a CODASYL set type can have one record type declared as its owner type and one or more other record types declared as its member types. There is a one to many (1:M) association between the owner record type and each of the member record type(s).Figure 1.2, Figure 1.3 and Figure 1.4 show set type strucures with one and two member types, and how a set type with multiple member types can be represented by set types with only one member type (the directed lines show the member type(s)). It seems to be very practical to consider set types with only one member type and from now on, the term set type will refer only to

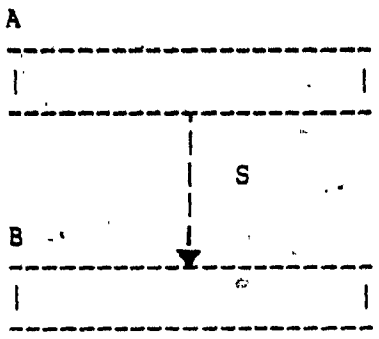


Figure 1.2 Set type structure with one member.

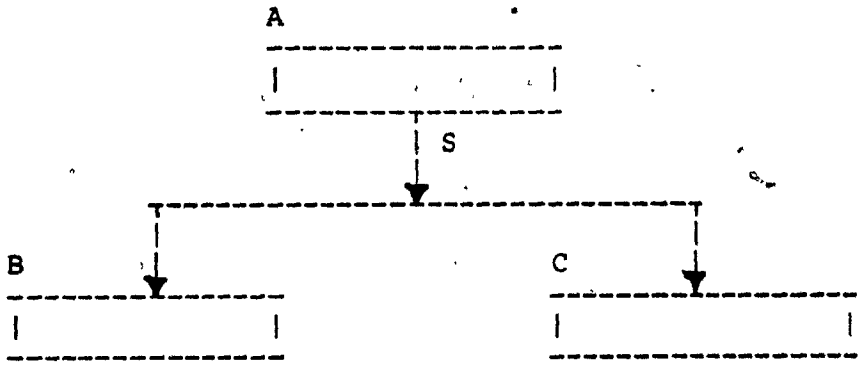


Figure 1.3 Set type structure with two members.

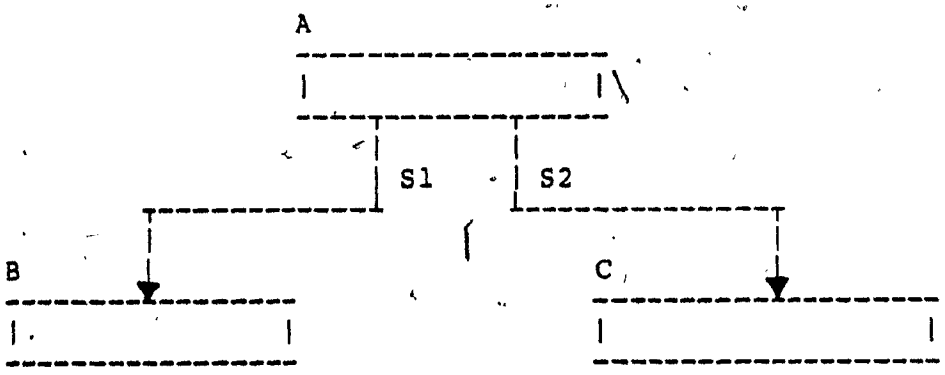


Figure 1.4 Representation of Figure 1.3 using two set types with one member.

types with one member record type. Because a set type is a 1:M association, an N:M association between record types cannot be represented as such directly. This, however, is not a CODASYL dissadvantage because an N:M association can be represented implicitly using types. Let us illustrate this by using N:M association between the entity sets parts and suppliers. One supplier can supply many parts and one part can be supplied by more than one supplier. First of all we are going to have two record types PART and SUPPLIER. Now we introduce a new record type S-P which would contain meaningful data items - as part number, supplier number, part's price, part's quantity or a proper combination of them.

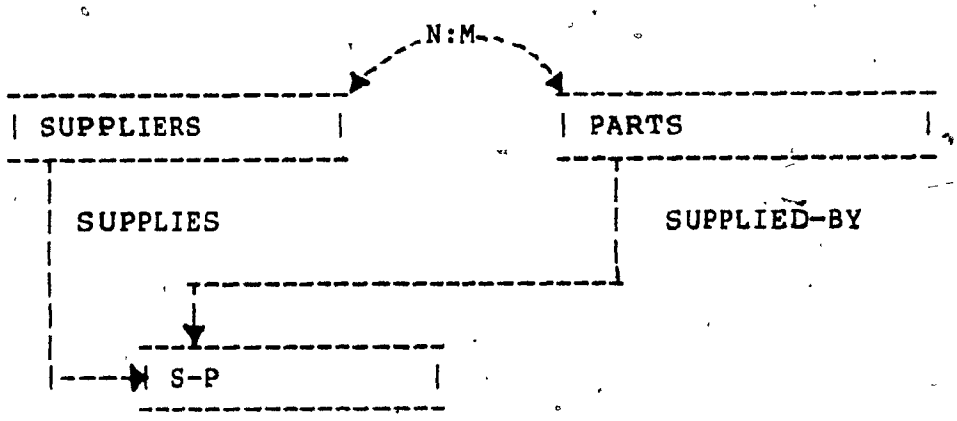


Figure 1.5 Set type representation of an N:M association.

The Figure 1.5 shows the new record type and the two set types (two 1:M associations) used to express the parts - suppliers N:M association. In general a well defined schema should not contain N:M associations between record types or

groups of data items - instead only 1:M associations should be used. A set occurrence is an instance of a set type; it consists of precisely one occurrence of the owner type and zero or more occurrences of the member record type. Each set occurrence represents a hierarchical relationship between the owner occurrence and the corresponding member occurrences. This fact imposes the following restriction: an occurrence of the member type should not belong to more than one set occurrence of a given set type at any given time.

1.3.2 CODASYL SCHEMA

A CODASYL schema defintion describes the logical organization of the data base as a network structured data model; however it also includes features concerned more with the storage level of a data base system. These features raised a lot of criticism and caused a lot of debate. In general a schema definition consists of four types of entries:

1. One schema entry, which identifies the schema.
2. One or more area entries which define the groupping of records (record occurrences) into areas.
3. Record entries, which define record types and specify details of their data items and data aggregates.
4. Set entries which define the grouping of record types into CODASYL set types.

For the schema definition a Data Description Language (DDL) is provided. In the rest of this section we discuss briefly

the most important features of the DDL.

AREAS

The total storage space of a DBTG data base is divided into a number of named subdivisions called areas. For each record type the schema specifies the area or areas into which occurrences of that record type are to be placed when they are entered into the data base. In the case where a record type is assigned to more than one area a decision has to be made, at the time a new occurrence is to be entered in the data base, as to which area should be selected. This decision may be made by the application program concerned. The exact meaning of the area in physical terms is determined by the implementation of a particular system. It can be for example, a disk pack, a cylinder or simply a stored file. The introduction of the area concept gives the DBA the ability to subdivide a data base in ways that may be used to enhance system efficiency without introducing any statements which are device or technique dependent. In this sense storage volumes or devices can be allocated by DBA procedures to store portions of the data base and help optimize the data base access by placing infrequently referenced or archival portions of the data suitably. The concept of area is one of the most controversial of the current DDL. There is a strong anti-area feeling that all physical positioning and data optimization should be carried out quite separately from the data description by people who

know how the data is employed. A frequent change of data employement should not affect the logical description of the data.

SINGULAR SET TYPE

A singular set type may be thought of as a set type where owner is not another record type but is the SYSTEM. Member type can be any of the schema record types. In fact it constitutes only one set occurrence with owner occurrence being the system and member occurrences being all the occurrences of the member type.

DATA BASE KEYS

The data base keys are internally generated unique record occurrence identifiers; when a new record occurrence is generated it is assigned a unique value that identifies the record occurrence and distinguishes it from all other reord occurrences. It can be a logical or a virtual address. During the execution of a DML command, the DBMS finds the data base key (in a known position within the record occurrence), of the required record occurrence and the system can directly access that occurrence by quoting the data base key. An extended notion of the data base keys was to allow the programmer to control the placement of a new record occurrence or let him 'save' a data base key for later use. This expansion caused a good deal of criticism. In general it is impossible for a DBTG system to exixst

without the data base keys, since it does not require the existence of a primary key in any record occurrence (and nor would it be reasonable to do so, provided the existence of the set type construct with its own navigation paths). A user defined primary key is used to generate the address of its corresponding occurrence, which also is used to create various 'chains' for sequential search. In the case where no primary key is supplied, the data base keys are required to create the search 'chains', which in this case will be only sequential.

SET SELECTION

A set type, in general, may have many set occurrences. The set selection clause in the schema set entry provides the means of identifying and locating the required set occurrence when it is to be retrieved, and determining where to place it when it is to be stored. The set selection clause of a schema set entry defines an access strategy for all of its set occurrences. For example, when a new record occurrence is created and it has to be member occurrence in some set occurrences (within different set types) automatically, the system uses the set selection clause of the member subentry, of the concerned set types, to select appropriate set occurrences.

LOCATION MODE

The location mode clause is part of any schema record type entry. Its primary purpose is to control the placement of new occurrences of this record type, when they are first stored in the data base. There are four location modes: DIRECT, CALC, VIA SET and SYSTEM. This clause specifies the means of assigning keys to record occurrences which will be used by the DBMS for retrieval and storage purposes. Particularly the CALC mode defines a hashing procedure (named in this clause); this procedure can operate on a specified key data item to generate the data base key for a record occurrence (it actually locates the record occurrence).

MEMBERSHIP CLASS

The member subentry of a schema set type entry must include a membership class specification for the member record type in the set type concerned. The membership class is specified by means of the INSERTION or RETENTION clause. Insertion refers to membership during storage and retention refers to membership during removal of a member occurrence from a set occurrence. The membership class specified in the insertion clause can be AUTOMATIC or MANUAL; the membership class specified in the retention clause can be: FIXED, MANDATORY or OPTIONAL. Broadly speaking, the

membership class of a record type (which can be different within different set types), affects the maintenance of the set occurrences of these set types. By maintenance is meant the creation, modification, or deletion of instances of the hierarchical relationships, represented by a concerned set type. Examples for each membership class can be found in [DATE-77], Chapter 20.

SOURCE AND RESULT DATA ITEMS

The source clause is used to specify that the value of a data item is to be the same as the value of another specified data item. A virtual source means it does not exist physically, although to the user it appears as if it does. The DBMS is responsible for handling this; an ACTUAL source means that the data item value exists, in the particular occurrence, physically. An ACTUAL or VIRTUAL result data item value is a value which is generated by special procedures by using data item values of specified data item(s), of specified record occurrences. Regardless of whether it is ACTUAL or VIRTUAL, such a data item cannot be the object of a DML MODIFY statement.

1.3.3 CODASYL SUBSCHEMAS

A subschema represents the user's view of the data base; it is drawn from the schema and it must be consistent with it as well as disciplined. The subschema specifies the record types that the user is interested in. The data items

of a subschema record type are exactly those the user wishes to see. The subschema also includes specifications of set types that the user wants to consider in his view. The subschema is defined using the subschema DDL. It is not possible to define a new record type which spans more than one schema record type or to define a new set type relationship which does not exist in the schema.

1.3.4 DBTG DML

The DBTG DBMS, in order to handle most of the DML commands, employs the concept of currency. For every program the DBMS maintains a table of currency status indicators which indeed are data base key values. There are four currency indicators; for area, each record type, each set type, and latest record type.

(1). Current of area A refers to the most recently accessed occurrence within A.

(2). Current of record type R refers to the most recently accessed record occurrence of record type R.

(3). Current of set type S refers to the most recently accessed occurrence - owner or member - within set type S.

(4). Current of run unit refers to the most recently accessed record occurrence, regardless to its record type.

The following is a very brief summary of the DML commands and show why the currency indicators are important.

For retrieval functions :

FIND : Locates an existing record occurrence and establishes it as the current of run unit and also updates the other three indicators appropriately.

GET : Retrieves the current of run unit; fetches data from a record occurrence.

For update functions :

MODIFY - Updates the current of run unit; changes data in this record occurrence.

STORE - Creates a new record occurrence (stores it in the data base); it becomes the current of run unit and updates other indicators.

DELETE - Deletes the current of run unit from the data base.

INSERT - The current of run unit is inserted into some set occurrence(s).

REMOVE - Removes the current of run unit from one or more set occurrences.

For control functions :

OPEN - Prepares parts of the data base for processing.

CLOSE - Makes parts of the data base available for processing.

An extensive bibliography related to DBTG proposals, literature, criticism and implementation is given by R. Taylor and R. Frank in [TAY-FRA].

1.4 SCOPE OF THIS THESIS

The structure of this thesis depicts the natural sequence of events that took place for its completion. The first step was to consider the background of the subject and make decisions as to what features had to be included in the

design of our experimental system. The rest of this thesis is dedicated to the next steps of this research which were the design of the developed system along with the consideration of the computer system which was to be used for its implementation, the detailed implementation data structures and techniques, and finally a sample application on the developed system.

Chapter 2 deals with the overall design philosophy of ENDBMS. It explains the supported data model and then introduces all the components and features of ENDBMS. Emphasis is given on the proposed disciplined subschema and access profile concepts, because it is beleived that their proper use for overall security enforcement in ENDBMS (and as it is suggested to any DBMS) is the most important aspect of this research work. Chapter 3 deals with the host implementation environment. It first explains the file system supported by the host Operating System (O.S.), then describes all the features of the Cyber Record Manager - a major component of the host O.S. - used by the ENDBMS, and then describes a simulation model for 'physical' concurrency. Chapter 4 provides all the details about the data structures and techniques used to implement all the ENDBMS features described in Chapter 2.

Chapter 5 provides the desription of a hypothetical company C, the design of a data model for its data base, the description of the authorized role of some employees of C as

prospective users of its data base. Then the company's data base is implemented on ENDBMS and actual interface sessions from its DBA and various users are to verify some of the system's capabilities with focus on security enforcement. Finally in the conclusion Chapter 6 the ENDBMS is briefly reviewed and possible expansions or modifications for future work are discussed.

CHAPTER 2

ENDBMS DESIGN PHILOSOPHY

In this Chapter the overall design of the ENDBMS is described and the feature-facilities supported by it are explained; these features match those suggested by Tsichritzis in Chapter 4 of [TSIC-77].

2.1 DATA MODEL

The data model of ENDBMS is based upon the Network model's record type and set type concepts. A record type is a named collection of 'field' names, where the record type name corresponds to an 'entity' of the real world, and the field names correspond to some of the attributes of this entity. An instance of a record type is a collection of the corresponding field values and it is called a record occurrence. All record types are declared in the schema. A set type is a named relationship (association), between two record types. This association has an order, that is, one of the two record types is declared as owner and the other as member. An arbitrary number of set types may be declared in the schema. A set type declaration requires both owner and member record type declarations. Any record type may be declared in the schema as the owner of one or more set types. Any record type may be declared in the schema, as the member of one or more set types. A given record type can participate as both owner and member in the same set

type. A set occurrence is an instance of a set type association. It consists of exactly one occurrence of the owner record type - owner occurrence - and zero or more occurrences of the member type - member occurrences. Each set occurrence is a hierarchy with the owner occurrence as superior and all the member occurrences being the dependants. A record occurrence cannot participate, as member, in more than one set occurrence of a given set type at any given time. A set occurrence may have an arbitrary number of member occurrences. A number of set occurrences - within different set types - may have common owner occurrence. An occurrence of the member type may belong to different set occurrences - within different set types. We mentioned above, that a set type is a named and ordered relationship of two record types; it must also be a 1:M relationship. An N:M (many to many) relationship cannot be declared explicitly in the schema. However by using a new record type as a connector, an N:M association is declared as two set types each being a 1:M association and wherein the connector record type is the common member.

A graphical representation of a set type is a directed graph where the owner and member are connected with a named directed arc (direction is from owner to member). The name of the arc is the name of the set type.

EXAMPLE 1

Figure 2.1 is a representation of the set type declaration

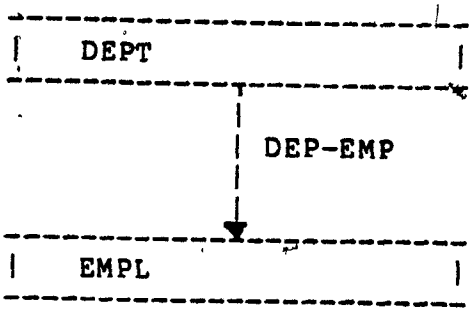


Figure 2.1 Data model for department-employees association.

DEP-EMP with owner and member record types, DEPT and EMPL respectively. This set type declaration corresponds to the 1:M association between the department and employee entities of a company.

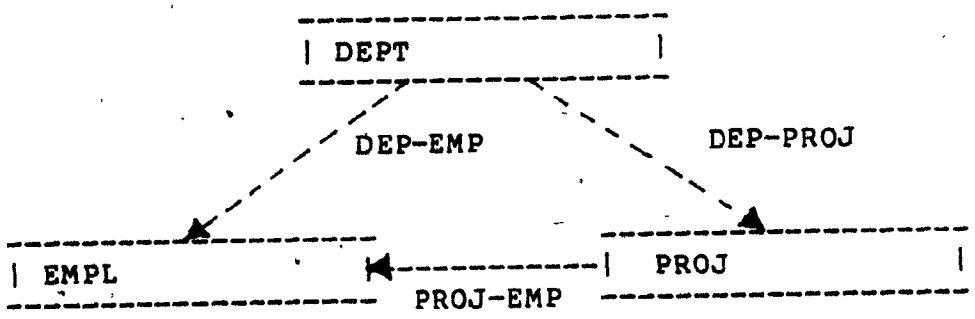


Figure 2.2 Data model for department-employees department-project and project-employees associations.

EXAMPLE 2

This example is a more complex structure than example 1. We

consider an environment of departments, employees and projects, and their 1:M associations between department and employees, department and projects, projects and employees. The names for the corresponding record types and set types, are shown in Figure 2.2

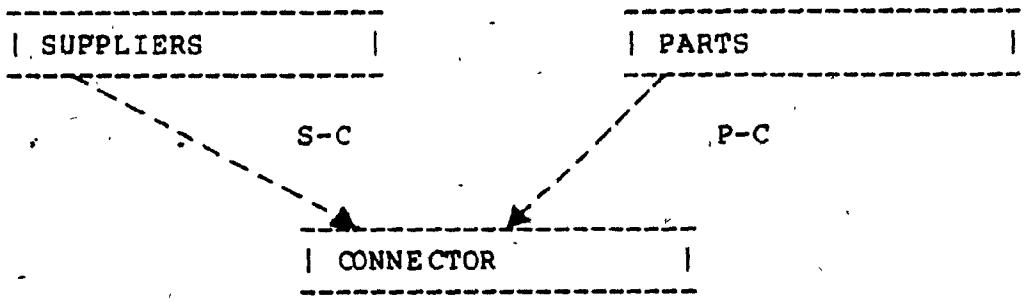


Figure 2.3 Data model for suppliers-parts association.

EXAMPLE 3.

In this example we consider an N:M association. A company uses some parts for its operation, and each part is supplied by some suppliers. We say some to include the possibility that different suppliers may supply the same part. We have the record types SUPPLIERS and PARTS and we introduce the new record type CONNECTOR. Now we can declare two set types; the set type S-C between SUPPLIERS and CONNECTOR, and the set type P-C between PARTS and CONNECTOR. These two set types represent two 1:M associations. The CONNECTOR record type will have as fields the part number, PNO, supplier

number, SNO, and probably the price and quantity. For any SUPPLIER owner occurrence, in a set occurrence within S-C, there are as many CONNECTOR member ocurrenes as the number of different parts supplied by this supplier. For any PARTS owner occurrence, in a set occurrence within P-C, there are as many CONNECTOR member occurrences as the number of different suppliers who supply this part. Figure 2.3 shows the data model for example 3.

SET OCCURRENCE REPRESENTATION

The means by which the owner occurrence is connected to the corresponding member occurrences is irrelevant as far as the user is concerned. One way of making these connections is to use a chain of pointers that originates at the owner occurrence and runs through all the member occurrences, but the owner occurrence can be reached from any member occurrence. This is shown in Figure 2.4. The actual physical connections used in ENDBMS are described later in section 4.9.2.

2.2 SCHEMA : A SIMPLIFIED DBTG SCHEMA

The ENDBMS stores the data required by all applications in one data base. In section 2.1 we described the generalized model, available to describe the data base. The description of the overall view of the data (i.e., the data base administrator's view of the data) is called the SCHEMA. It names all logical units of data in the data

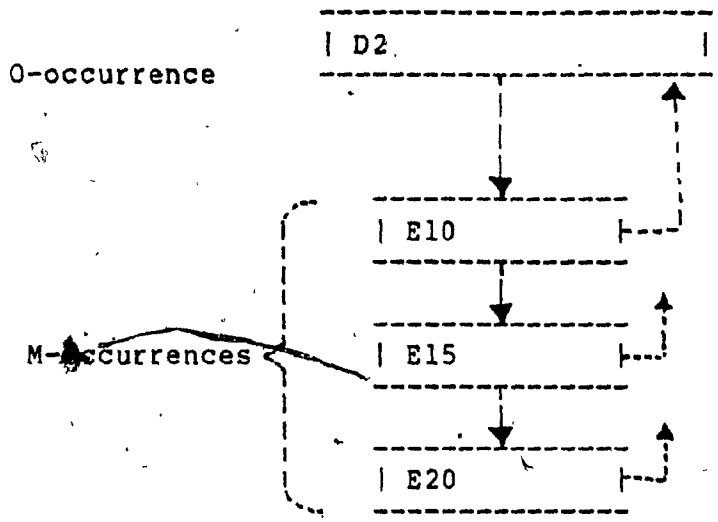


Figure 2.4 Logical representation of set occurrence.

base, e.g., record types and set types; it also names the fields of each record type and defines their type - character or numeric -, and their maximum length in characters. The schema expresses all possible logical associations of the records by means of the set types. The ENDBMS schema does not describe any aspects of the mapping of logical units of data to a physical storage structure. It also does not specify any access restrictions to certain logical units of data (security or integrity measures) because security in ENDBMS is implemented in a separate integrated subsystem.

The ENDBMS schema supports the CODASYL concepts such as: area, singular sets, set selection, location mode, membership class and result data items, but in a different fashion.

(1). The ENDBMS data base (at the storage level) consists

of subdivisions (physical and logical) but they are not defined in the schema, and as such the user sees a virtual unified data base.

(2). The ENDBMS uses the idea of the 'singular set' by considering all record occurrences of a given record type as member occurrences of a set occurrence, where owner occurrence is a system generated head-occurrence, but no such specification is required to be made in the schema.

(3). The ENDBMS selects a set occurrence, by selecting the owner occurrence explicitly (set selection). This occurrence is selected either directly by using the DML command 'get the owner of' or indirectly by using a record type DML command to locate a record occurrence, which furthermore becomes the owner occurrence of some set occurrence by using the 'currency indicators' command (see section 4.9.4). Once the ENDBMS has selected a set occurrence any set type command can be used. The ENDBMS schema does not include any of the SET SELECTION clauses of the DBTG proposals.

(4). The DBTG schema includes the LOCATION MODE clause to specify how a new record occurrence is going to be placed in the data base when it is first created. The ENDBMS schema has no such specification, because a new record occurrence is placed in its own logical unit, corresponding to its record type, and can exist independently. Later it can manually participate in any set type occurrences which have been declared for the record type in the schema. Thus in

ENDBMS a manual membership for any record occurrence, whithin permissible set occurrences, is imposed as well as a manual or automatic dis-association. Manual refers to the explicit 'remove' DML command and automatic refers to the 'delete' DML command, where a record occurrence is deleted from the data base along with all existing associations to different record occurrences.

(5). All data defined in the ENDBMS schema are real. A complete picture of the 'virtuality' supported by ENDBMS is given in 2.3.4.

(6). The ENDBMS uses the idea of the data base key. When a new record occurrence is stored in the data base, it is associated with a unique key; this key is generated by the system, and it not only identifies the record occurrence but it also identifies the logical and physical unit of the data base where it will be stored. The fact that the key of a record occurrence identifies the logical and physical unit of the data base, has to do with possible future modification of ENDBMS using associative storage (see conclusion).

2.3 SUBSCHEMAS

2.3.1 INTRODUCTION

A subschema is defined by almost all authors of text books on data base systems as being one of the following :

(1) User's (interactive or application programmer) view of the data base.

(2) Subset or logical subset of the data base.

Date in [DATE-77], Chapter 21, makes a 'precise' statement that 'subschema is a subset of the schema' and specifies what entries of the schema may be omitted in a given subschema. Tsichritzis in [TSIC-77], Chapter 4, says that the subschema is a logical subset of the schema and thus a view of the data base. He also indicates that the subschema 'selects' relevant parts of the schema and 'adds necessary modifications' to form an application's view. Martin in [MART-77], Chapter 6, gives the following defintion: 'The term subschema refers to an application programmer's view of the data he uses. Many different subschemas can be derived from one schema'. We consider all of the above definitions as incomplete. The concepts 'user view', 'subset of the schema' or 'logical subset of the schema' are not sufficiently clear without a precise schema to subschema mapping function.

2.3.2 SUBSCHEMA : A DISCIPLINED SUBSET OF SCHEMA

We define the subchema as a disciplined subset of the schema. It consists of a collection of record types and set types defined over the schema and constructed using a given 'access profile' (see section 2.5.2) and a set of rules. The access profile specifies which schema record types can be accessed with probable restrictions on some fields and/or occurrences of the record type - as far as read and update operations are concerned; it also specifies which schema set

types can be accessed with probable restrictions - as far as read, include, remove operations are concerned. The set of rules are the subschema's record and set type construction rules. Thus when a user retrieves a subschema record occurrence, it may result in retrieval of more than one schema record occurrences. The information retrieved from the data base, based upon a user's subschema and profile, is divided into two categories: real and virtual. The set of rules governs the method of deriving this information from the information stored in the data base.

2.3.3 REAL SUBSCHEMA FIELD AND RECORD TYPES

A field or record type, defined in a user's subschema, is said to be real if it is defined in the same way as in the schema. A real record type is composed of real fields and is associated with one schema record type, which is called the schema 'base' record type. The user defined subschema names may or may not be the same as their corresponding schema names, however their attributes must be the same as in the schema.

2.3.4 VIRTUAL SUBSCHEMA FIELD AND RECORD TYPES

Any field or record type, defined in a subschema, in any form different from the corresponding schema definition, is said to be virtual. A virtual field is defined over one schema record type and it is a transformation of the one or more fields of this schema record type. At present, ENDBMS

allows a number of numeric and string type transformations (i.e., the RESULT-OF, SUM-OF, AVERAGE-OF and MERGE-FROM procedure types). The fields of a virtual subschema record type can be defined over one or more COMPATIBLE (defined below) schema record types. Every subschema record type must specify a schema base record type; every real or virtual field is taken or generated from the base record type or a record type which is compatible with the base record type. One important requirement in defining a virtual record type is that, at least one of its fields must be real and defined over the base. A COMPATIBLE record type is defined as follows : let B be the base record type of the subschema record type R and let A be another schema record type over which some field(s) of R is(are) defined. We say that A is compatible with B if and only if A is the same as B or there exists a schema set type S, with A being its owner and B being its member type.

2.3.5 SUBSCHEMA SET TYPES

A subschema set type S is defined over two subschema record types O and M, where O is the owner and M is the member record type. O and M can be real or virtual subschema record types. Every subschema set type S is considered to be real and there is a corresponding schema set type SS which must be declared in the subschema set type definition as the 'base' set type for S. Thus the rule for defining a subschema set type is as follows : let OB be the

base of the subschema owner record type O, and MB be the base of the subschema member record type M; OB and MB must be the owner and member types of the schema set type SS. We do not have virtual set types in the subschema, because the user is not allowed to declare new associations, different from the ones declared in the schema. This rule does not reduce the potential of the system. If an application needs a new set type, the DBA could define (add) the new set type and give appropriate access rights, for this set type, to all concerned users.

2.3.6 SUBSCHEMA CONSTRUCTION AND USE

The construction of a user subschema can begin after the user has been 'installed' in the system and his profile has been defined. As the subschema is defined, the definition is checked for lexical, syntactic and semantic errors with respect to the schema definition and the user's profile. An error free definition is stored in the system's subschema file. Once the subschema is stored, the user's DML commands are checked against his subschema for validity of set, record and field names used by the commands. The check also includes a look-up of the user's profile for record occurrence restrictions (ROR's); if there are any, these are considered during the execution of the command.

2.3.7 SUBSCHEMA CONSISTENCY

The use of subschemas for security enforcement in ENDBMS, requires the maintainance of consistency of a user's subschema with respect to his access profile. We had three options to implement this consistency : (1) permit no updates to the user's profile, (2) permit updates, but only in a positive direction, which would allow expansion and (3) permit any type of updates to a user's profile. The subschema must, at any time, reflect the access capabilities defined in the user's profile. With the first option, the user's profile is defined once and no updates are permitted. The user's subschema definition and any further subschema updates must be consistent with this profile. The only way the DBA can change the user's profile, is to delete the old profile and the user's subschema (if any) and redefine a new profile from which the user has to recreate a new subschema. The second option does not create any consistency problem with an existing subschema. In the third option, which was adapted and implemented in ENDBMS, additions and deletions to user's access profile are permitted with the system automatically checking the consistency of the user's subschema with his updated profile.

2.3.8 SUBSCHEMA AND MULTIUSER CONCURRENT ENVIRONMENT

The ENDBMS can operate in 'batch' or in an 'on-line' concurrent multiuser environment. Any time a user executes a retrieval command he gets what is available to him, in view of his access profile, at the 'stored' data base level at that time. We believe there are three mechanisms for storing the subschema

FIRST: When the subschema defintion is stored, in one of the system's files, the definition is used to derive a 'logical' subset of the data base which is stored separately. This mechanism does not propagate [KIM-79] the effects of DML commands against the subschema, down to source data base level. Instead all DML commands are executed against the stored data base subset. It is obvious that extra storage is required for storing the 'data base subset'. When changes are made to the source data base, the stored result-subset for a given subschema has to be destroyed and has to be reevaluated in order to reflect the recent changes. Any update command, directed to a subschema, changes only the corresponding 'data base subset'. The hazardus here is the following: when the stored subset is destroyed, all the effects of the update DML commands against the subschema which have been

recorded in the stored subset are lost too.

SECOND: The subschema definition as well as the corresponding 'logical' subset of the data base is stored. Only retrieval commands are directed to the stored subset of the data base, without the overhead necessary for evaluating the subschema definition every time. In case of an update statement the subset is destroyed, the DML command is executed against the stored data base, and the subschema definition is reevaluated.

THIRD: This mechanism requires only the storage of the subschema definition. This is the one adapted and implemented in ENDBMS. The user executes all his DML commands, against his subschema and down to the source, stored, data base. It reflects all changes made at the data base level by any other user and can support any concurrency control, like shared read locks and exclusive update locks. One advantage obviously is the storage efficiency. Storing only the subschema definition is very efficient for maintaining consistency between user profiles and user subschemas.

2.4 ENDBMS INTERFACES (USER INTERFACES)

The ENDBMS provides facilities for two type of users, the Data Base Administrator or DBA, and the non-DBA users, who wish to access the data base according to their

application's view of the stored data.

NOTE:

Through out this thesis we use the term 'user' to indicate a non-DBA user, and the term 'system user' or ENDBMS user to indicate both the DBA and non-DBA user. However, some times, we will use the term 'non-DBA user' to avoid confusion.

2.4.1 DBA

As far as the ENDBMS is concerned the DBA is a special user or an individual who has to perform a certain role. This role can be played by a single person or by a number of persons within an enterprise. The DBA, primarily, is responsible for system installation. For ENDBMS there is a separate component which installs the system. At the system's installation, an entry is generated which among other things includes the DBA's identity. Generally the DBA is responsible for the data base operation of an organization and his major function is to determine the information requirements of the various applications and to provide the views they require. In the ENDBMS the DBA has the following duties :

1. To install the system.
2. To define and administer the schema.
3. To install users.
4. To define and administer every user's profile.

5. To help users in constructing their subschemas and explaining all information included in the schema definition.

6. To establish consistency in the data base after a crash, namely, to recover all users which were using the system, at the time the failure occurred.

7. To recover specific users who logged-off abnormally, either intentionally or because of some system - host or ENDBMS - bug.

ENDBMS provides him with all the utilities needed to perform these actions. He optionally, may use any of the DML commands and he is assumed to have no restrictions as far as data base retrieval and update is concerned. Some of his utilities require a special environment for execution and for this reason he may have to do some monitoring; e.g., in order to delete a record or set type entry from a user's profile, the DBA must make sure that the user is not using the system.

2.4.2 NON-DBA USERS

The domain of interaction of a non-DBA user with the data base, is determined by a subschema. A user presents his data requirements to the DBA and the DBA defines an appropriate access profile for him and helps, if necessary, the user to define the subschema that desribes and implements the view. Different users will have different views of the data base, and the access capabilities of every user are contained in his access profile.

Before a user can access the data base, the following steps must have been performed :

a. User installation; this is done by generating a user entry in the 'user list'; this is a list of all validated users of the ENDBMS. This entry among other information contains the user's identity: user number and user password.

b. The user has been given an access profile, where all access capabilities, as far the data base is concerned, are clearly defined.

c. The user must have defined his subschema, which reflects his view of the data base and which is consistent with his profile.

2.4.3 INTERFACE MODE - EDITOR

Every system user must use the ENDBMS in either the 'batch' or in the 'interactive' mode. The system can be accessed either through a card deck or through a terminal. In case of a terminal any of these two modes can be selected. As we already mentioned the interactive, or I, mode activates the special feature of the ENDBMS, the editor facility, which is used to correct errors during the parsing of any command. The parsing or translation of any ENDBMS command uses a parsing stack to check for lexical and/or syntactic errors. The command's handler verifies all the semantics of the command, i.e., whether the specified record, field and/or set type names are used properly from the authorization point of view. Every time a new input line is placed in an internal buffer for analysis, the contents of the parsing stack as well as the stack pointer (a pointer which points to the current top of the stack), is saved if the I-mode was selected. In case of an error during translation time, the line containing the error is

displayed and below the line, an error indication character, i.e., '^', indicates the position of the error within the line, followed by an informative error message. At this point the interactive user can correct the error by entering a new - error free - line, or drop the parsing of the current command, and move to the next command - if any, in the same input session. If a new line is entered, the saved contents of the parsing stack, are loaded again and the stack pointer is set to the value of the saved stack pointer. In other words the parsing status goes back to the status, which existed before the analysis of the erroneous line had started; and then parsing continues from the begining of the new line, as if nothing had ever happened. If the system is used with 'B' mode the editor facility can not be used to correct any error. If the error is such that the parsing of the corresponding command can not continue, the system displays an error message, stops the parsing of the current command and advances to the next command of the same input session; if there is no other command, a new input session is initiated by displaying (printing) "READY".

2.5 SECURITY IN ENDBMS

2.5.1 INTRODUCTION

We use the term 'security' to mean the protection of the access to ENDBMS via a log-in, the protection of the data kept by the system itself (this involves information kept in system directories like schema and subschema

definitions), and finally the protection of the stored data in the data base, from both authorized and unauthorized users of ENDBMS. This definition does not correspond to the one usually given in the literature. It is not enough to say that security includes only protection of the data in the data base, against unauthorized disclosure, alteration or detruction [DATE-77]. In this section we are not concerned with the social aspects of the problem; Martin and Norman discuss this aspect in [MAR-NOR]. The ENDBMS is designed so that it can support: 1. Log-in security 2. System directories security and 3. Data base security, by assigning each of its users a unique user number, a password and an access profile. Any one who attempts to use the system must identify and authenticate himself. Identifying a user consists of entering a code that is recognized by the system; this code is a unique user account number. Authenticating a user means verifying that the user is actually the person who is allowed to use the identification code. The use of passwords, known only by the authorized users, is employed for authentication. More sophisticated and costly methods of authentication do exist - e.g., badge readers, 'formula' passwords and built-in terminal identification. An extensive discussion of various authentication procedures is given by Hoffman in [HOF-69].

2.5.2 USER ACCESS PROFILE

A user access profile consists of a number of record and set type entries. A record type entry refers to a schema record type entry and defines the access capabilities - read, insert, modify, delete - for a particular user, at the type and occurrence level. Using an integer code, the DBA indicates the permissible combinations of retrieve and update operations.

The DBA uses five sets to define the restrictions at record type and ocuurrence level:

1. RFR: Read Field Restrictions.
2. MFR: Modify Field Restrictions.
3. ROR: Read Occurrence Restrictions.
4. MOR: Modify Occurrence Restrictions.
5. DOR: Delete Occurrence Restrictions.

which may be empty, to indicate respectively:

1. Which fields of a schema record type can not be seen by by a user (are not available).
2. Which fields of the accessable ones can not be modified by a user. This implies that the RFR and MFR sets have to refer to different fields.
3. Which record occurrences, of a schema record type, are not accessable. To do this, the DBA uses a restriction criterion - i.e., one field name, one relational operator and a key value for the selected field. In the current version, the ENDBMS supports only one criterion and not a boolean expression of different criteria.
4. Which accessable record occurrences are not

available for modification - i.e., to change some field values. Same as in (3), only one criterion can be specified.

5. Which accessable record occurrences cannot be deleted. Same as in (3), only one criterion can be specified.

The field name used in ROR, MOR and DOR must not be one specified in RFR set, and cases (4) and (5) have meaning only if the record type access code permits Modify or Delete in the first place.

A set type entry of a user access profile refers to a schema set type. Every set type entry is associated with a set access code, to indicate the three set type operations: Read, Include and Remove. Include and Remove access rights presume Read with no field restrictions - i.e., RFR empty - over the member schema record type. A user can include or remove a record occurrence from some set occurrence, only if he can access the entire record. As a consequence a subschema set definiton must have a REAL member record type in the case of include and remove operations. In the case where a profile set type entry defines read access rights, the set type's owner and/or member type may have read field restrictions. Thus a subschema definition based on a such profile set type entry can have VIRTUAL owner and/or member subschema record types. More specifically, a virtual owner or member occurrence can be retrieved using set type operations.

2.5.3 A FIVE LEVEL SECURITY SYSTEM

After a proper log-in the system imposes user identification - user number check - for every system command, to make sure that every user enters the commands that he is supposed to. A 'delete' DML command normally can be used (submitted) by any user, but if the user happens to have restrictions, as far the data base is concerned, the command will never be executed. In order to protect the data base we make use of the user subschema and user access profile. Date [DATE-77] in his Chapter 23.3, states that a user profile alone is not sufficient to determine whether the user should be allowed to perform a given operation; without explaining his perception of the user profile. We strongly dissagree because our profile structure with its access code - record or set type access code - clearly determines which operations (e.g., Read, Insert, Modify, Delete, Include, Remove) or legal combinations of these, can be used by a particular user. Furthermore each of these operations is associated with a schema record or set type definition and its corresponding entry in the user's profile clearly specifies what part(s) of a record type is accessable, by its RFR set. It can also specify what fields are restricted from modification, and at the record occurrence level can impose restrictions for read, modify

and delete respectively. The ENDBMS monitors any data manipulation operation - submitted by a user - and permits only the authorized operations on the authorized portion of the data base. The ENDBMS security subsystem is designed as a five level hierarchy, the levels being:

(1) Identification and Authentication Level
(2) System Availability Level
(3) Record Type Level
(4) Set Type Level
(5) Record Occurrence Level

The security system is divided into two categories; the data independent and data dependent security [CONW-72]. More details and its implementation is given in Chapter 4.

2.6 LANGUAGE FACILITIES

The ENDBMS has four independent subsystems; these being the schema, the subschema, the data manipulation and the security subsystems. The last also includes another subsystem, namely the recovery subsystem. The user communicates with them and uses their facilities, via four independent languages. In this section we consider these four languages and describe their primary features. Implementation details are given in Chapter 4. The formal description of the LL(1) grammars used by these four languages is given in Appendix A.

2.6.1 SML: SCHEMA MANIPULATION LANGUAGE

This language has been designed so that it can be used by the DBA to: (1) define the data base; (2) expand the schema by inserting new record or set type entries; (3) modify the schema by removing one or more record or set type declarations, thus freeing their allocated storage at the data base level; (4) display at any time the schema - the data base description.

It is worthwhile mentioning that removal of a record type or set type is a very powerful operation and require special considerations with respect to their effects on the data base and with respect to their execution environment. For example, removal of a schema record type would cause removal of all schema set types which use it, and furthermore would appropriatly update all user access profiles; this would lead to an update of the user subschemas. It is for these reasons that the DBA should consider carefully such an operation. He should also make sure that there is no user activity on the ENDBMS before executing the operation to remove the record type; special utilities are provided in ENDBMS to perform these operations.

2.6.2 SSML: SUBSCHEMA MANIPULATION LANGUAGE

This sublanguage has been designed to help all authorized users to define their view of the data base or subschemas. The structuring of a subschema represents the user's data needs; it might require professional or semi-professional knowledge of the stored data. The suggested procedure is the following: the user contacts the DBA and explains his application's information needs; then the DBA defines the access capabilities for that user, through an access profile which is stored in the ENDBMS. Having done this, the user can define the entries - record and set types - of his view. The SSML contains commands to perform the following operations: (1) subschema definition; (2) insertion of a new subschema record or set type entry; (3) deletion of one subschema record or set type entry; (4) destruction of the entire subschema; (5) listing of the subschema definition. A user subschema, once defined, can remain in the ENDBMS as long as required by the user - unless the DBA decides to revoke all or a portion of his access capabilities. In the current version of the ENDBMS no subschema sharing is allowed.

2.6.3 SL: SECURITY LANGUAGE

The security language includes commands especialy designed for the DBA, so that they provide him with the means to exercise centralized control over the ENDBMS. First there is a group of commands to 'manipulate' the non-DBA users, e.g., to install, remove, list them, as well as to suspend and restore their access rights. Second there is the profile manipulation group of commands, in order to define, remove and update user access profiles. In addition there are two more commands to lock and unlock the entire ENDBMS, one command to change a user's password (available to non-DBA users as well) and finally one recovery command in order to recover specific (interrupted) users upon their request. Details about each of these commands are given in section 4.7.3.

2.6.4 DML: DATA MANIPULATION LANGUAGE

The DML has been designed so that it can be used effectively by the DBA or any other user. The ENDBMS keeps track of the identity of all system users who are using it, and performs appropriate actions (e.g., security enforcement), at both translation and execution time of any DML command. As we explained before, a record type is assigned to a data base logical unit which contains all of its record occurrences; this logical unit exists

idependently in the data base and every occurrence of the logical unit can be accessed through a 'record type' path. The logical association between two schema record types is expressed by a schema set type definition. This logical association, at the data base level, between the record occurrences of the two corresponding logical units, is implemented by means of 'set type' paths (see section 4.9.2) and they exist independently from the 'record type' paths. These set type paths are used to retrieve information from the data base, using the set type DML commands. Every logical unit must be 'locked' appropriately in order to be processed. In addition if the logical unit is no longer needed, it must be 'unlocked'. Both the lock and unlock operations, must be done with explicit user commands. However the system, during a normal log-off, checks for 'forgotten' locked logical units and unlocks them.

The ENDBMS uses the DBTG concept of currency status indicators, but the number of currency indicators, as well as their use, is quite different. In fact the ENDBMS uses the following three currency indicators:

1. 'current of record type R' refers to the most recently accessed record occurrence within R.

2. 'current owner of set type S' refers to the most recent owner occurrence within set type S; it identifies the most recently accessed set occurrence of S.

3. 'current member of set type S' refers to the most recently accessed member occurrence of an already selected set occurrence; it is set to zero in the case where no member occurrence exists.

Default system settings are explained in section 4.9.4. The difference in use from the DBTG currency indicators, is that the user has full control of the first two. There are two kind of settings, the system and user settings.

SYSTEM SETTINGS. Every time a record occurrence is accessed through a 'record type' DML command, the system updates the currency indicator for this record type. Every time a set occurrence is selected - by accessing its owner occurrence - the system updates the 'current owner' indicator for this set type, and sets the 'current member' indicator to the first member occurrence, for the selected set occurrence - if the set occurrence is not empty. This for example happens when the 'get first owner' or 'get next owner' is executed. The command 'get the owner in S, of the current of R' sets the 'current member' not to the first member in S, but to the member R itself.

USER SETTINGS. The user explicitly can make a selected record occurrence of some record type, become the owner occurrence of some set occurrence and in this way select a set occurrence. Within a specific set type, with consequent updates of the 'current owner' and 'current member' currency indicators of this set type. The 'current owner' indicator of a set type S with owner type R is different, and set independently, from the 'current of R'. The user, also, can make the 'current owner' of some set type, become the 'current of the owner record type'.

Another important difference from the DBTG DML is the presence of conditional and unconditional jumps - IF-GOTO and GOTO commands - included in the ENDBMS DML, which allow a 'loop' structure.

A record occurrence is FETCHED first - i.e., selected by means of a record or a set type path - without enforcing security, using the GET DML command. A fetched record occurrence is retrieved through another DML command, the LIST. Before the occurrence is retrieved, the ENDBMS checks if the user is the DBA or not, and if not then checks for Read Record Occurrence Restrictions. In the rest of this section we give a summary of the DML commands.

2.6.4.1 FETCH COMMAND: GET

There is a record type and a set type GET. The record type GET is used to fetch the 'first', the 'next', and the 'next where' record occurrence within a record type file (schema or subschema, see section 4.9.1) R. The set type GET is used to fetch: the 'first' owner, the 'next' owner (of some set type S), the 'first' member (of the current set occurrence), the 'next' member (of the current set occurrence), and the owner of the 'current of a record type R' (within a set type S).

2.6.4.2 RETRIEVE COMMAND: LIST

There are different formats of LIST, for record type and set type retrieval. The record type LIST is used to retrieve the 'current' of a selected record type R, and to retrieve all accessable occurrences of R starting from the 'current' of R. It can, also, be used with an optional 'where-clause' to select occurrences from R, and/or with an optional 'sort-clause' to sort in ASCENDING or DESCENDING order the retrieved occurrences from R, by specifying ONE field name. The set type LIST is used to retrieve the 'current member' of a selected set occurrence S, or retrieve all member occurrences of S, starting from the 'current member', with an optional 'where' and/or 'sort' clause.
NOTE: all owner occurrences can be retrieved by using the record type LIST format, by considering the owner record type alone.

2.6.4.3 CONDITIONAL JUMP COMMAND: IF

The IF DML command tests if the 'current' of a record type is last and, if it is, the next DML command indicated by the specified 'label' is executed. It also tests if the 'current member' of the current set occurrence (within a set type S) is last, and, if it is, the next DML command indicated by the specified 'label' is executed.

2.6.4.4 UNCONDITIONAL JUMP COMMAND : GOTO

It alters the flow of DML command execution. The 'label' associated with it specifies the DML command with is going to be executed next. The 'label' refers to the current input session and it can cause a forward or backward jump.

2.6.4.5 SETTING COMMAND : SET

The SET DML command is used to set explicitly the three currency indicators and the length of an 'output line' printed by ENDBMS. It can set the current of record type R, to current owner of set type S - where S has R as owner type, the current of record type R to current member of set type S - where S has R as member type, and it can also set the current owner of set type S - with owner type R - to current of record type R.

2.6.4.6 UPDATE COMMANDS

(1) INSERT: it enters one or more record occurrences (of the same record type) to a logical unit.
(2) DELETE: it deletes a selected, the 'current', record occurrence from a specified logical unit.
(3) REPLACE: it changes a selected field value of the 'current' occurrence, of a specified logical unit, with a new one.

(4) INCLUDE: it enters a selected record occurrence as member occurrence to some selected set occurrence.

(5) REMOVE: it does exactly the reverse of INCLUDE.

2.7 DATA INDEPENDENCE

Fundamentally, data independence implies the separation of the user's view of the data base, from everything else - from the data access mechanism, from the storage media, from the stored data base, and from the views of other users. There are two points of view of Data Independence; logical data independence and physical data independence. Physical data independence implies program immunity to changes in the storage structure, while logical data independence implies program immunity to changes in the data model definition. Physical data independence allows application programs or user queries to continue to execute correctly after the storage structure has been changed to maximize overall performance, to implement new standards in the storage structure or to take advantage of some new hardware technology. Logical data independence allows application programs or user queries to continue to execute correctly after the schema has been changed. Schema changes may involve adding or deleting data structure elements of record and/or set types. New record types imply new data in the data base and new set types imply new associations bettween old and/or new data - new access paths. Deletion of schema record and/or set types require, of course, -

tuning (update to retain consistency) of all existing subschemas; a subschema by convention must be a consistent subset of the schema. Updates to user subschemas does not affect the logical independence. For example a query which used to use a certain subschema record type, is executed against an updated subschema, with the record type being deleted; either it is executed correctly or it is not executed at all. The ENDBMS schema, subschemas and DML provide both logical and physical data independence.

2.8 DATA BASE INTEGRITY

Data base integrity refers to the problem of ensuring that the data which is stored in the data base, is accurate at any given time. The smallest logical unit accessable in the ENDBMS is a record occurrence. A 'field' defined in the schema has two characteristics; the type (character or numeric) and its length (in number of characters). Similarly a record type consists of some such fields, and thus has a certain total length too. The degree of integrity provided in the ENDBMS is as follows: at the time a new record occurrence is stored in the data base, the system checks the submitted data for number of fields, their type and their length. The number of fields must be the same as the one declared in the schema record type definition; the same applies for the type of a field, that is a field declared of type numeric must have as value a string of characters representing an integer or a real

number. Finally the length of a particular field value must be less than or equal to the specified length in its field definition in the schema. A field value having a length smaller than its declared one is padded with blanks to the right. The current version of ENDBMS does not support integrity constraints; a user authorized to update the SALARY field of an EMPLOYEE record type with declared field length 6, can submit a SALARY value of '500000', which may be 'unreasonable', but the current version of ENDBMS does not detect this as an error. Similarly a user who has update rights over a PARTS record type with a field COLOR of type CHAR and length 8, can submit the 'xzxzxzxz' as COLOR value, even if the possible colours for a particular part are black, brown and red. However this won't be detected as an error and will be stored as a COLOR value.

2.9 ENDBMS CONCURRENCY

2.9.1 CONCURRENCY AT PHYSICAL UNIT LEVEL

The ENDBMS is designed so that it can operate in a concurrent multiuser environment. As we explain in Chapter 3, the ENDBMS uses 19 NOS direct access files - the so called physical units - 10 of them are used for the data base, 8 are used to store the system directories, and one is used to store the error messages. The number of files is 19 because we decided so. Using 9 as system files is the best choice as far as 'physical sharing' is concerned. On the other hand, every 'stored record type file' (see section

4.9.1) must be entirely (one only NOS file is required for accessing) placed on one NOS file. For our environment the ultimate case will be to have one NOS file assigned to each schema record type. Consider for example the delay time involved in a situation where two users require concurrently two different 'logical units' with 'D-lock', (see section 2.9.2) and both of them reside on the same NOS file. In this particular case a 'D-lock' over a logical unit automatically makes unavailable, for a D-lock, all the other logical units which happened to be on the same NOS file; a D-lock requires the corresponding NOS file attached with M-mode and only one can have at a time. An R-lock does not affect the distribution of the logical units over the NOS files. Every user performs 'read' and/or 'update' operations on these physical units. As it is explained in Chapter 3, the 'availability' of some of these units is controlled either by the host operating system or by the ENDBMS itself. In Chapter 4 we explain how NOS or the ENDBMS controls the 'availability' of these physical units. All of them are available on a 'many read' - 'one updates' basis. As a consequence reading does not cause any consistency problem in a concurrent environment. For an update operation, on a physical unit, the ENDBMS is responsible for providing that unit with the appropriate mode - i.e., attach the unit with M-mode.

Most of the ENDBMS update commands, require more than one physical unit in 'update' mode, for their complete execution. To avoid deadlock situations the ENDBMS makes sure that all required physical units are available, before starting execution of an update operation.

2.9.2 CONCURRENCY AT LOGICAL UNIT LEVEL

The user is not aware - and is not supposed to - of the physical units used by the data base. He thinks of the data base, as a big pool where all the data is stored, and the system can get it from there for him. Our data base is divided in logical units, that is the stored schema record type files, which are distributed uniformly over the physical units. For the DBA, a logical unit is a collection of record occurrences of some schema record type, and they are stored in the way he looks at them. For any other user, a record type file is a collection of record occurrences of his subschema record type. Those record occurrences may be either real or virtual. In any case these subschema record type files are not really stored separately, but they are generated from the stored schema ones, based on the corresponding subschema record type definition. Before a record occurrence, either real or virtual, is accessed the corresponding schema or subschema record type file must be explicitly 'prepared' for appropriate processing by 'opening' or 'locking' it. Explicit opening of a subschema record type file might result in opening of more than one

schema record type file, i.e., logical units. An opened record type file, must be explicitly 'closed', in other words its processing must be terminated. In order to make sure that a record type file is not left opened, the ENDBMS closes all open record type files in a partial action of the normal log-off procedure.

To implement logical concurrency, i.e., concurrency over data base logical units, we used a method of mutual exclusion with 'shared' read locks and 'exclusive' delete locks. Details are given in the implementation Chapter 4, in section 4.11.2. D. Potier and P. Leblanc in [POT-LEB] discuss different locking schemes (division of the data base into locking units or granules) for data base access control, as well as locking policies to ensure integrity of information, and finally they present a framework for quantitative analysis of the impact of these factors on the DBMS's performance.

2.10 ENDBMS RECOVERY: A DBA CENTRALIZED SCHEME

Recovery techniques can be used to restore data in a system to a usable state. Such techniques are widely used in data base systems, in order to cope with failures. A failure [VERH-78] is an event at which the system does not perform according to specifications. Some failures are caused by hardware faults (e.g., a power failure or a disk failure), software faults (e.g., bugs in program or invalid

data), or human errors (e.g., the operator mounts a wrong tape on a drive, or a user does something unintentionally). A failure occurs when an erroneous state of the system is processed by some algorithm of the system. In the ENDBMS we consider only two cases for recovery. The first deals with the case where all running ENDBMS processes are interrupted, because of some failure of the host system. The second deals with the case where a user logged-off abnormally (e.g., he did not terminate his interface with ENDBMS properly).

The ENDBMS keeps track of any user activity (actual or apparent; by apparent we mean that a user is 'marked' active because he never logged-off properly), and additionally monitors every command which requires recovery (see section 4.12.). At any given time, a user is either idle or executes a recovery-free or recovery-needed ENDBMS command. The recovery-free ENDBMS commands are harmless because they use only 'read' operations and they do not require recovery. The recovery-needed commands require special treatment and they are divided into two categories: the INSERT-type and the DELETE-type commands. The insert-type commands require UNDOING of their partial execution, so that a system directory or the data base is brought back to its last consistent status. The delete-type commands require mandatory completion of their execution, no matter where their execution is interrupted. Our scheme is centralized

to the DBA; he recovers himself and all interrupted users after a system crash. A user who intentionally interrupts himself can no longer re-enter the system; he should inform the DBA and the DBA can recover him with the 'RECOVER' security utility.

2.11 USER-ENDBMS-O.S.-DATA BASE COMMUNICATION

Douque in [DOUQ-74] describes the four "levels of access control" which have been implemented in PHOLAS. They establish communication paths between the user and the DBMS on one hand and, between the DBMS and the host Operating System on the other. Each of the four methods is depicted by Figure 2.5, Figure 2.6, Figure 2.7 and Figure 2.8 respectively. In Figure 2.5 every user has his own copy of the Data Base Handler (DBH) and he interfaces with it by means of his own User Work Area (UWA). The DBH interfaces with the host O.S. by means of a local buffer pool. The difference between Figure 2.5 and 2.6 is that in Figure 2.6 the DBH is a shared segment. The O.S. provides the segmenting and there is only one DBH code in core. Logically it appears as two data base handlers and this version cannot be considered as a central DBH facility. The full central DBH version of PHOLAS is shown in Figure 2.8, where several users, each with his own UWA, interface with the DBH which owns one buffer pool and access the data base via the O.S. In Figure 2.7 the 'multitasking' DBH facility of PHOLAS is shown; however this multitasking is implemented

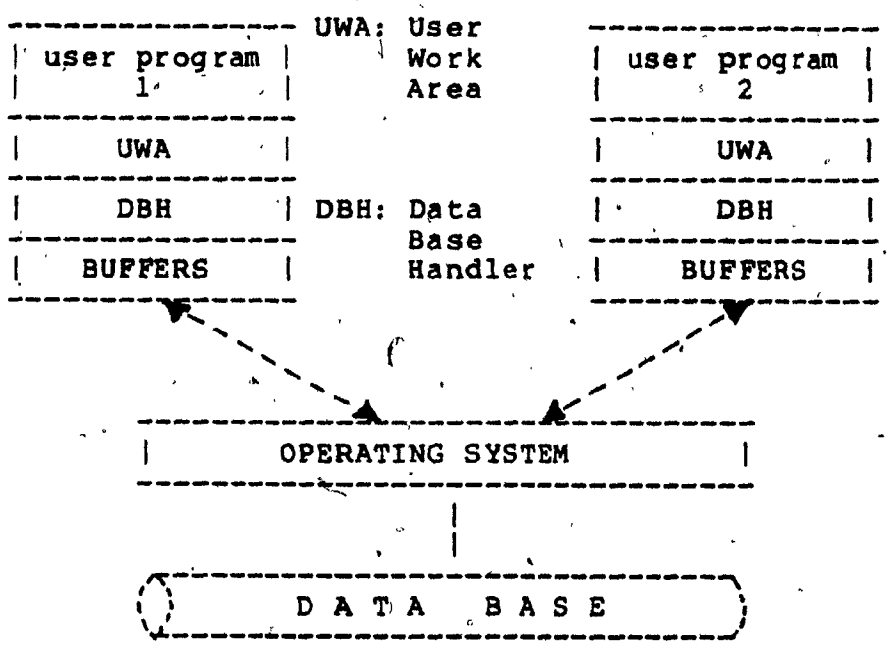


Figure 2.5 Independent DBH facility of PHOLAS.

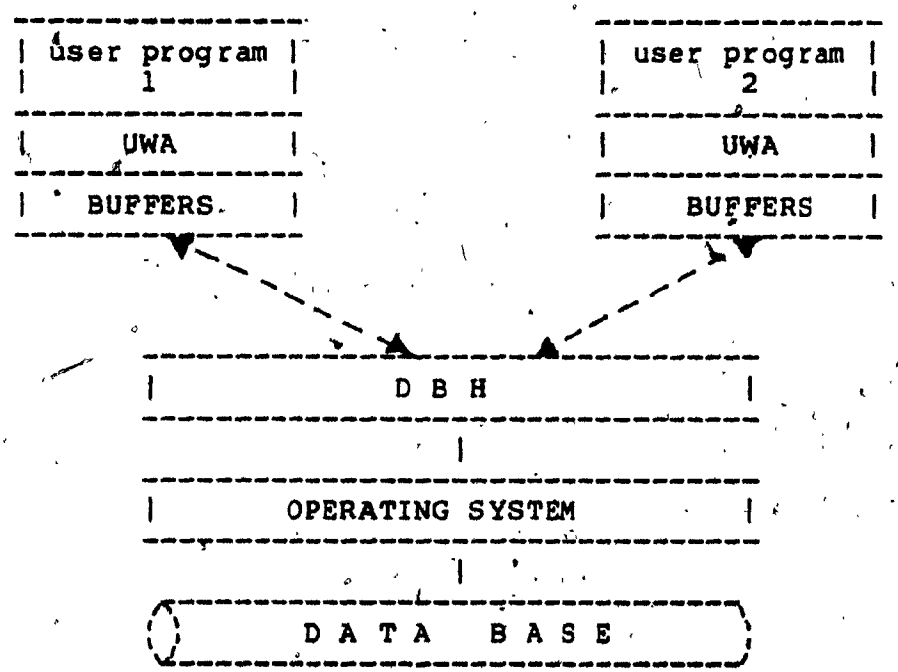


Figure 2.6 Pseudo central DBH facility of PHOLAS.

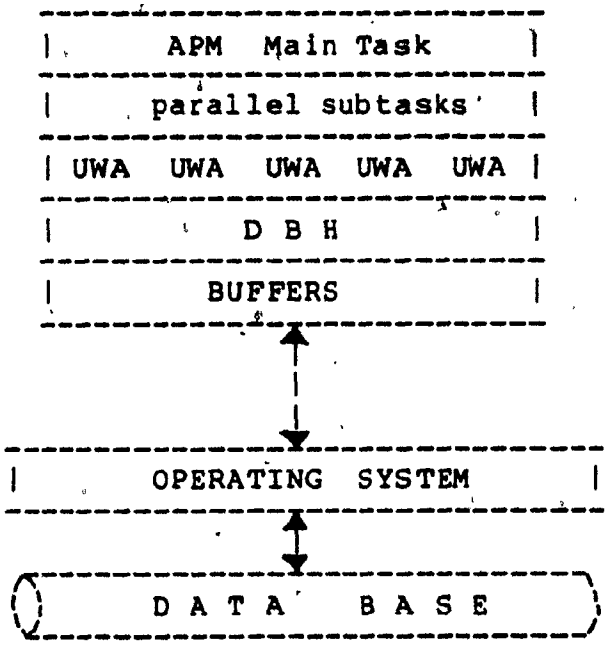


Figure 2.7 Multitasking DBH facility of PHOLAS.

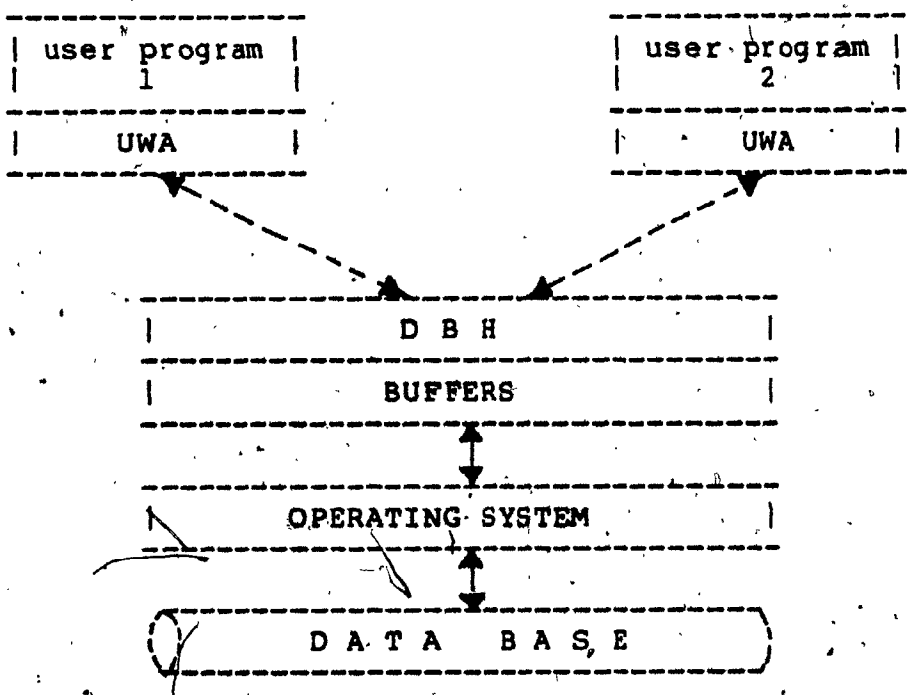


Figure 2.8 Full central DBH facility of PHOLAS.

at the user interface level and not at the O.S. level. Every subtask of a user program is performed by a special package, called the Asynchronous Process Monitor (APM). The subtasks are invisible to the O.S., but because the DBH code is linked to every user program, the COBOL compiler automatically assigns a separate UWA to every subtask.

The first approach was used for the ENDBMS; it seems to be simple in the way that it is implemented as a user program in a high level language, without 'special' treatment of the O.S. Secondly we were not permitted to use the O.S. in such a way, that would have allowed one of the other approaches to be used. Note also that use of one of these alternate approaches would have had great impact on the overall design of the ENDBMS. The ENDBMS has been compiled and then stored as an NOS object nodule. Every one who wants to use the ENDBMS, must submit the following NOS control statements - provided that he has proper NOS permission modes over the specified NOS files, as well as over the 19 system and data base files used by the ENDBMS.

DBA:

```
/ATTACH,DBALIB.
/$LIBRARY,AAMLIB,DBALIB.
/ATTACH,ENDBMS.
/ENDBMS.
```

NON-DBA:

```
/ATTACH,USERLIB/UN=KEGFE62.
/$LIBRARY,AAMLIB,USERLIB.
/ATTACH,ENDBMS/UN=KEGFE62.
/ENDBMS.
```

The first NOS statement attaches to NOS user's job a library which contains 19 object modules corresponding to 19 COMPASS routines used by the ENDBMS. The second statement specifies all the required libraries - AAMLIB is a NOS library used by the CRM - the third 'loads' the ENDBMS's object module and the fourth executes it.

CHAPTER 3

IMPLEMENTATION ENVIRONMENT

The ENDBMS was impemented on a CDC CYBER 172 which is running under the NOS 1 operating system. This Chapter presents the relevant features of the FILE SYSTEM supported by NOS as well as the salient NOS component the Cyber Record Manager (CRM). It also presents the method of providing concurrency in the ENDBMS system.

3.1 NOS FILE SYSTEM

The file system supported by NOS is divided into two categories: (1) files assigned to a user job and (2) files permanently residing on mass storage (see [NOS-1of2] section 1-2-8). A file assigned to a job is known to the NOS system by its entry in the file name table/file status table (FNT/FST), which contains the file name, the device on which the file resides, the file type and its current position and status. A permanent mass storage file is known to the NOS system by its entry in a permanent file catalog associated with a user number. The catalog entry contains the file's name, location, length, permission modes, and access history.

3.1.1 FILES ASSIGNED TO USER JOBS

NOS uses the folowing mnemonics for file classification.

INFT: Input file
PRFT: Print file
PHFT: Punch file
LOFT: Local file
PTFT: Primary terminal file
PMFT: Direct access file
LIFT: Library file
ROFT: Rollout file
TEFT: Timed/event rollout file

The input, print, punch, rollout and timed/event files are queued files. A queue file waits on mass storage until the system resource or peripheral equipment it requires becomes available and its priority is the highest of the files in the queue.

An input file - also called job file because it contains user supplied control statements and data for a job - exists in mass storage in the input queue. It enters (e.g., it is created) the input queue directly when a local or remote job enters the system or indirectly when a user job 'submits' another job via some NOS control statement. The input file of a time-sharing job consists of all terminal input directed to the system during a time-sharing session. A user job refers to its input file by the file name INPUT. A print file contains data to be printed and it is created and placed in the print queue as a result of a job termination, when the system changes the local file OUTPUT (if present) into a print file, or as a result of excution of an OUT, ROUTE, PRINT, or DISPOSE control statement naming a local file to be printed. The system assigns to every print file an identification code which

identifies its origin and by default they are printed at the place of origin. The user can overide the default routing and specify his own printer or printer type. As a print file waits in the print queue its priority increases and the file is printed when its printer becomes available and when its priority is higher than all other files destined for that printer. A punch file contains data to be punched on cards and it is routed from the mass storage punch queue according to the name the user assigns to it or according to parameters specified on a 'punch' control statement.

Local files are temporary files; this file type includes all scratch and working files except the primary file which is also a temporary file and it is designated so by the PRIMARY, NEW or OLD NOS control statements. A user assigns a direct access permanent file to his job by issuing an ATTACH or DEFINE control statement and if the file is attached with a mode permitting file modification, he can write on the permanent file. The library file type includes the USER-NUMBER LIBRARY, the PROGRAM LIBRARY and the USER LIBRARIES. A library file is a read only file that several users can access simultaneously. USER LIBRARIES are the files named in the LIBRARY loader control statement and searched by CYBER LOADER to satisfy external references within the program it is loading; they contain compiled or assembled routines.

If during job processing, the system or the user determines that a job must be TEMPORARILY removed from central memory, the system writes all information concerning the job on a system-defined rollout file. The file is read back into the central memory (CM) when the job is reassigned to a control point, [NOS 1of2, section 1-3-8]. The timed/event rollout file is similar to a rollout file in that it contains all the information concerning a job temporarily removed from the central memory. However it is rolled back into the CM only when a specified event has occurred, e.g., a file required by the job is no longer busy, or a specified time period has expired.

Normally, each executing program is allowed to reside in the CM for a certain period of time before relinquishing its space to another program. The amount of time that a job is allowed to occupy CM is called the central memory time slice and it is a system parameter. When this CM time slice is exceeded the program concerned may be rolled out. A NOS user can explicitly or implicitly submit a request to the system to roll out his job by using the ROLLOUT control statement or the ROLLOUT macro respectively and he can also specify his own event or time period. In case the user doesn't specify any event or time period the 'roll in' is controled by the NOS system.

3.1.2 PERMANENT FILES

Permanent files are retained on mass storage until they are purged. There are two types of permanent files, the Indirect Access Permanent Files (IAPF) and the Direct Access Permanent Files (DAPF). The IAPF are accessed by copying the permanent file to a temporary file (local or primary). A user can retrieve a temporary copy of an indirect access file using the OLD and GET NOS control statements, he can create a new one using the SAVE control statement and he can alter an existing one using the REPLACE control statement. The user accesses a direct access permanent file directly without using a temporary copy. The user can create a new DAPF with the DEFINE control statement and he accesses the file with an ATTACH control statement.

3.1.3 FILE STRUCTURES

3.1.3.1 NOS FILE STRUCTURE

A NOS file may contain more than one logical file; if it does, it is called a multiple file. A multiple file begins with a Begining Of Information (BOI) and ends with an End Of Information (EOI). The end of a single file is indicated by its own End Of File (EOF). A file within a multiple file begins either at BOI or after an EOF of the preceeding file. Each file consists of one or more records of information and a record is made up of 60 bit CM words. The following is an example of a single file with one

record:

(BOI) data (EOR) (EOF) (EOI)

3.1.3.2 PHYSICAL FILE STRUCTURE

When NOS stores a file it converts it to a structure that conforms to the physical characteristics of the storage medium (disk, tape, cards). The file and record marks are converted to physical BOI, EOR, EOF, EOI indicators. The basis of all physical file structures is the physical record unit (PRU); this being the amount of data that can be read or written in a single device access.

3.1.3.3 CYBER RECORD MANAGER FILE STRUCTURE

Cyber Record Manager (CRM) handles I/O for several CDC program products including FORTRAN EXTENDED 4, FORTRAN 5 and COBOL 5. CRM superimposes its file structure on the NOS file structure. Through CRM the user can specify:

a. File Organization
b. Blocking Type
c. Record Type

for his data. The file organization determines how records are accessed. The blocking type determines how CRM records are grouped on their storage media and the record type determines the smallest unit of data that can be retrieved by the CRM.

3.2 CYBER RECORD MANAGER

CRM is a generic term relating to both Basic Access Methods (BAM) and Advanced Access Methods (AAM) as they run under the NOS and NOS/BE operating systems. BAM is a file manager that processes sequential and word addressable file organizations. AAM is an interface [CRM-AAM] between user programs and system I/O routines; it also provides consistent error processing and maintenance of different file organizations. AAM routines are used by some compilers such as FORTRAN and COBOL, and they recognize calls to AAM routines. Use of AAM by compilers and user programs extends I/O compatibility to both the system and application levels. The primary task of AAM is to provide record input/output for files on supported devices. The various types of records and file organizations must be identified for AAM. These and other file characteristics must be set by the user in the File Information Table (FIT). The FIT (a user defined array of at least 35 words) is divided into a number of fields that describe certain aspects of the file.

3.3 AAM: ADVANCED ACCESS METHODS

3.3.1 FILE ORGANIZATIONS

A user, using the AAM file manager, can specify the following file organizations:

a. INDEXED SEQUENTIAL
b. DIRECT ACCESS
c. ACTUAL KEY

In Indexed Sequential files, records are in order of a primary key and can be accessed sequentially or randomly. In Direct Access files (which should not be confused with the direct access permanent files), records are not in any recognised order and are accessed by key manipulation. In Actual Key files records are accessed by a primary key containing the block and record number within the file. In the next section we consider only the DIRECT ACCESS file organization because this is the one used by the ENDBMS.

3.3.2 DIRECT ACCESS FILE ORGANIZATION

A direct access file contains a file statistics table, a number of home blocks and under certain conditions, overflow blocks; all blocks being of fixed length, specified by the user or by the system. The following terms have specific meaning in relation to direct access files: [CRM-AAM pages 2-5 and 2-6]

PRIMARY KEY

A primary key is a contiguous bit string that always appears in a direct access record. It is hashed to produce the location of the home data block containing the record.

HASHING

Hashing denotes the method of using primary keys to search for relative home block addresses, of direct access records. The CRM uses its own hashing routine but it is possible for a user to define a new hashing

routine.

SYNONYMS

Synonyms are records whose key hash to the same home block.

HOME BLOCKS

A home block is a block whose relative address is computed by hashing primary keys; it contains synonym record keys hashed to that relative address. The number of records within a home block is determined by the size of the block and the type of the records.

OVERFLOW RECORD

An overflow record is a record whose key has been hashed to a home block which is already filled.

OVERFLOW BLOCK

An overflow block is the second or subsequent block in a chain that starts at a home block; it contains overflow records and can contain records belonging to more than one overflow chain.

CHAIN

A chain consists of blocks that are logically connected by forward and/or backward pointers. Home blocks and overflow blocks are chained in both forward and backward directions.

The relative position of records within a direct access file is not important. A record is stored and retrieved by hashing its primary key to produce the relative address of a home block. When a home block is filled, the record can be placed in another home block or in a system generated overflow block. The placement of the overflow record depends on the overflow record storage option selected by the user. The logical structure of a direct

access file is shown in Figure 3.1. FSTT is the file statistics table, H1 to H6 are the home data blocks and OV1 to OV3 are the overflow blocks.

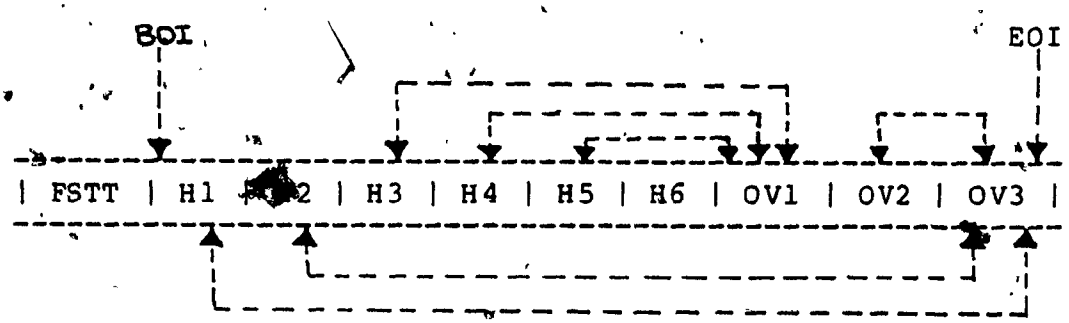


Figure 3.1 Logical structure of a Direct Access File.

FILE STORAGE ALLOCATION

Mass storage space is preallocated when a direct access file is opened for the first time. The size of it is determined by the home block size and the number of home blocks both of which can be defined by the user. The records are grouped in fixed-size home or overflow blocks. The user can supply his own hashing routine for optimum record distribution. ENDBMS uses the hashing routine supplied by the CRM.

FILE BLOCKING

Each direct access block (home or overflow) is an integral number of PRU's less one central memory word and is

treated as a system logical record. Each block has a header and records are stored in the remaining words as they are received, begining with the word following the last word of the header. Each stored record has its own record header (one CM word) which contains the record's deletion flag, the record's length, the number of unused bits of the last word of the record and some other information used by the CRM.

3.3.3 RECORD TYPES

AAM supports eight record types; here we are going to consider only the F and Z type records.

FIXED LENGTH, F TYPE RECORDS

In a file with F type records, all records are of the same length. The number of characters in the F type records is specified by the fixed lenghth (FL) field in the FIT. From all NOS permanent files, used by the ENDBMS, only the 'message' file has F type records.

ZERO BYTE, Z TYPE RECORDS

A zero byte or Z type record is terminated by a 12-bit byte of zeros, in the low order position of the last word in the record. In contrast with the F type, the Z type records are not fixed and they can have different record length. Maximum record size is indicated by the FL field of the FIT and it can be set when the file is created. When a record

is written, the value of the record length (RL) field determines the processing that takes place. If the RL field is set to a value greater than zero, the end of the record is determined by searching backwards from the character position specified by the value of the RL field and removing all full words of blanks. If the RL field is set to zero when a record is being witten (PUT routine) the end of the record is determined by a backwards search for the last non-blank character in the working storage area (i.e., a user defined array which contains the actual data to be stored). The search begins from the character position indicated by the FL field in the FIT and all full words of blanks are removed.

3.3.4 AAM ROUTINES FOR FORTRAN 4 USERS

In this section we describe briefly the AAM routines; detailed descriptions are given in the following CDC reference manuals: [CRM-AAM] and [CRM-F-GUIDE].

FILE

This routine is used to establish the structure of the file and it also specifies a file name. If a file with this name is not ATTACHED to the user's job the CRM creates a local file with the specified name. If such a file is attached as a local file the CRM creates and associates with

it a FIT. The user specifies the name of the FIT, the name of the file and other file parameters, by setting appropriate FIT fields, such as record type, max and min record length, position and length of the key, number of home blocks, size of home blocks, etc.

OPENM

A file can be processed if it has been defined by the FILE routine and has been opened with an OPENM. The very first OPENM is used with processing direction equal NEW. This indicates file creation. The values of the mandatory parameters (because there are some parameters which can be set later) specified in the FILE AAM function, determine some characteristics of the file which cannot be changed later (e.g., once the block size is defined it cannot be redefined). In fact the first OPENM constitutes the creation of the file structure and a file which has been created can be opened for read or update purposes, depending upon the value of the processing direction field specified in the OPENM call.

CLOSEM

An opened file must be explicitly closed with the CLOSEM AAM function. If a job is terminated and a file is left opened there is a possibility that the contents of the file's buffer, kept by the CRM, were not updated at the disk level. The user should at least specify the file to be

closed, by supplying the name of the file's FIT as first parameter and in this case the file is closed and can later be reopened without issuing the FILE call. If the user wants to get rid of the file at the same time, i.e., releasing it, he should supply a second parameter too. Both of these formats are used by the ENDBMS.

IFETCH

With this AAM function the user can 'inspect' the value of some FIT fields, for example an error may occur as a result of a read request. (note: for any error condition the error status field of the file's FIT is set to an octal error code. If the error is FATAL or the number of trivial errors matches the specified number the user job is aborted, otherwise execution continues and the next successful CRM operation clears off the error status field. For details see error processing in [CRM-AAM]). If FIT1 is an array of 35 elements, one word each, and ES is the name of error status field of the FIT1 table and if the statement N=IFETCH(FIT1,2LES) is executed after the read request then the value of N can give the result of the read request. In the case where the N is set to zero then the requested record was found, but if N is set to a non zero value, then an error occurred and the octal value of N is a code for an informative error message.

GET

The GET AAM function is used to read records from a file which has been opened previously. The user should, at least, supply the first three parameters in the GET call which are: the name of the file's FIT; the file's work space, an array defined by the user with appropriate length, wherein the requested record is returned; and the key of the requested record which is used internally by the hashing routine. The ENDBMS uses the following GET format:

CALL GET(FIT, WS, KEY)

FIT: name of the file's FIT

WS : file's work space

KEY: key of the requested record

GET, as well as the following functions PUT, REPLC and DLTE, require the file they operate on to be opened.

PUT

This AAM function is used to write a record on a file. Its call must specify, at least, the name of the file's FIT and the name of the file's work space (array with appropriate length and which contains the actual data of the record to be written). The user can also specify how many words - in characters - are to be written as a third parameter in the PUT format and optionally some other parameters. The ENDBMS uses only the first three. The

format used is:

CALL PUT(FIT, WS, RL)

FIT: FIT name

WS : work space name

RL : length of record to be written, in characters.

REPLC

The REPLC function is used to replace a specific record with a new record, with or without the same length. The user should at least supply the first four parameters in the following order: the name of the file's FIT, the name of the file's work space - which contains the new record - the length of the new record and finally the key of the record to be replaced. The format used by the ENDBMS is:

CALL REPLC(FIT, WS, RL, KEY).

DLTE

The DLTE function is used to delete a specified record from a given file. The CRM 'flags' the record as deleted and its space becomes free so that can be used again. The user should at least specify the first two parameters: the name of the file's FIT and the key of the record to be deleted. The format used by ENDBMS is:

CALL DLTE(FIT, KEY)

3.4 NOS FILE CATEGORIES

Permanent files fall into three categories which specify the accessability of the file, see section 1-8-2 in [NOS-1of2].

The categories are:

a. P or PRIVATE (for private)
b. S or SPRIV (for semi-private)
c. PU or PUBLIC (for public)

Private files are available for access only by the originator or those to whom the originator has explicitly granted permission with the PERMIT NOS control statement. Semi-private files are available for access by all users who know the file name, the originator's user number and the file's password - if any. Public files can be accessed the same way as the semi-private but NOS records different information for S and PU files. The kind of operations a user can perform on S or PU file depends upon the value of the file's permission mode parameter (see next section). One of the three options can be used when the file is defined, changed or saved with the 'CT' parameter; if this parameter is not selected the system's default value of P is assumed.

3.5 PERMANENT FILE PERMISSION MODES

There are eight permission modes for a NOS permanent file:

1. E execute
2. R read
3. RA read-append
4. RM read-modify
5. A append
6. M modify
7. W write
8. N null

here we consider only the RM and M modes for direct access files, more details are given in [NOS-1of2] section 1-8-3.

RM: for direct access files it allows the user to read and/or execute the file while another user is concurrently accessing the file in M or A mode.

M: for direct access files MODIFY permission means that the file can be changed, lengthened but not shortened. Table 3.1 shows all combinations of multiple access to a permanent file. The indication A/R means that one user has attached the file with A mode and one or more other users have attached it in R mode; the indication M/R means that one user has attached the file with M mode and one or more other users have attached it in R mode. The table also shows that if the file is free any request is granted. All response indicators except BUSY mean a grant of the access request.

3.6 SIMULATION MODEL FOR CONCURRENCY

In order to write/read on a NOS file, the user must have the right to access the file (unless he owns the file), second he must have been given proper permission modes and third he must have attached the file appropriately to his job. For example if a user has an M permission mode on some

CURRENT ACCESS	REQUESTED ACCESS						
	W	M	A	R	RM	RA	E
FREE	W	M	A	R	RM	RA	E
W	busy	busy	busy	busy	busy	busy	busy
M	busy	busy	busy	busy	M/R	busy	busy
A	busy	busy	busy	busy	A/R	A/R	busy
R	busy	busy	busy	R	R	R	R
RM	busy	M/R	A/R	R	R	R	R
RA	busy	busy	A/R	R	R	R	R
E	busy	busy	busy	R	R	R	R

Table 3.1 Combinations of multiple access.

file and he attaches the file with R mode, he cannot update the file.

A permanent direct access file becomes known to a user job using the ATTACH control statement [NOS-1of2, 1-8-7]. As a consequence from the access table 3.1 and M, RM mode definition, we can have one user who writes on a file (attached with M-mode) and many other users who read from the same file simultaneously (they have attached the file with RM-mode). The problem arises from the moment where some other user wants to write on a file which is already attached with M-mode. What we did in ENDBMS is the

following. All the files, used by the ENDBMS, can be shared by all validated users. In other words, every user can attach an ENDBMS file with either RM or M-mode.

Every user attaches all files with RM-mode. If a user wants to write on a file, as a result of an ENDBMS command, he must attach all required files with M-mode. This requires the following procedure: first if the files are opened they must be closed, second they must be released and reattached with M-mode, and third they must be opened again for processing. All the M-attached files are kept so as long as the command needs them. After the command is executed the M-attached files are closed, released, reattached with RM-mode and reopened for read processing. It is understood that all this must be done at execution time of a user's job. To handle this problem we designed 19 COMPASS routines, nine for the ENDBMS system files and ten for the ENDBMS data base files. The purpose of these 19 COMPASS routines is to attach 19 direct access files, at execution time, with M or RM-mode. Each routine is called from the ENDBMS with one parameter. When a routine is called with zero, the associated direct access file is attached with RM-mode and when it is called with one the corresponding file (if free) is attached with M-mode.

Each COMPASS routine uses the NOS macros RFILEB, ATTACH and ROLLOUT [NOS-2of2]. The ROLLOUT macro uses the 'COMCMAC' NOS COMMON DECK and it is used only for those

files whose availability is controlled by the NOS operating system, i.e., where NOS determines if the files concerned are free or not. The ATTACH and RFILEB macros use the COMCSYS, COMCPFM and COMSPFM NOS common decks. Each file is attached with the local file name being the same as the permanent file name. The macro RFILEB defines the FET for each file attached to a user's job (i.e., an ENDBMS running process). When the operating system processes an ATTACH request and the file is not available, i.e., the file is busy, the normal action of the O.S. is to abort the user's job. In the ATTACH macro format it is possible to set the error processing bit (bit 44 of the second word, of the file's FET). When the O.S. processes an ATTACH macro - for some file with its error processing bit set - and the file is found busy the user's job is not aborted but the O.S. returns control to it, i.e., the next statement of the user's running program will be executed.

In the case where the O.S. attempts to attach a file with M-mode and the file is busy, it returns an error code 1 in the 'AT' field of the first word, of the file's FET table. Because the bit 44 is set, the job is not terminated. At this time the user can use the ROLLOUT macro to rollout his job until a system default time period has elapsed or the file becomes available. This is implemented as follows: in the COMPASS routines we test the file's 'AT' field and if it is zero the routine returns with the file

attached, otherwise ('AT'=1) we issue the ROLLOUT macro and then the ATTACH macro is executed again until the 'AT' becomes zero.

For some ENDBMS files we do not have to set the error processing bit in an ATTACH macro call, because their M-attachment is controlled by the ENDBMS itself, i.e., the ATTACH macro (for M-mode) is executed after the ENDBMS determines that the file needed is free. More about this in the concurrency implementation in Chapter 4.

CHAPTER 4

IMPLEMENTATION OF THE ENDBMS

The language chosen for implementation of the ENDBMS is FORTRAN 4; the specific version being the CDC FORTRAN EXTENDED VERSION 4 [FORT-4]. The choice had to be made between COBOL and FORTRAN, because only their CDC compilers support direct access with the Cyber Record Manager, through user written programs. We believe that FORTRAN is more flexible and convenient with respect to the structure of a software unit of this size. The implementation of ENDBMS includes the following modules:

1. A NOS 'procedure file' INSTALL of 268 lines; the name of the procedure being GEN. This file consists of a number of NOS control statements and a FORTRAN program. The NOS statements create 19 direct access NOS files, and the FORTRAN program implements the 'creation' run of these files [CRM-AAM] which includes the definition of the file's organization and the generation of some initial entries for the ENDBMS system files such as the 'file head entries' and the DBA's 'security entry', as they are explained in this Chapter. The file organization of a direct access file specifies values for some vital fields of its FIT [CRM-AAM]. These fields are: the FO which determines the type of file organization, the RT which determines the record type, the MNR and MRL which determine the min and max record

length for Z-type records, FL which determines the field length used for accessing, the KL which determines the length of the key (the default position of the key is the first field of the record if the user wants to specify another field, he can do so by setting the RKW field of the FIT), the MBL which determines the size of a home block, and finally the HMB which determines the number of home made blocks or the initial size of the file. All other FIT fields are assumed to have the CRM's default values.

2. A collection of 19 COMPASS routines (537 lines) stored on NOS file DBACOM. Each of these routines is used to perform the task of attaching and releasing a particular file (one of the 19 ENDBMS files) while the ENDBMS run copy is executing. The DBACOM is used (see LIBGEN control statement in [NOS-1of2]) to produce the DBALIB library which must be attached to the DBA's NOS job which executes the ENDBMS's NOS object module (see section 2.11).
3. A collection of 19 COMPASS routines (556 lines), similar to the ones in DBACOM, stored on a NOS file USERCOM. The USERCOM is used to produce the USERLIB library, which must be attached to every non-DBA user's NOS job, which executes the ENDBMS's NOS object module (see section 2.11).
4. A FORTRAN program (50 lines) saved in a NOS file WMFILE, which is used to write all ENDBMS error messages on the

ENDBMS file SF7. This ENDBMS file has fixed length records with length of 6 CDC words; the first word is reserved for the record's key which is an integer code of the corresponding error message stored in that record. The next five words are used to store an error message of up to 50 characters. The execution of the WMFILE uses the MFILE (245 lines) as a data file. Every line of the MFILE contains an error message with its integer error code.

5. A NOS 'procedure file' DELFILE (8 lines), with procedure name DELPROC, which is used to destroy the 19 ENDBMS files; this will be done by the DBA when he wants to destroy an existing data base and later re-install the ENDBMS for further use. This procedure file makes use of the PURGE NOS control statement to purge the 19 NOS direct access files.
6. A FORTRAN program of 16,556 lines, whose compilation produces an object module which will be the 'run copy' of ENDBMS.

4.1 ENDBMS INSTALLATION

The ENDBMS is installed with a completely separate procedure which has to be carried out first, in order to create and organize all the NOS direct access files used by the ENDBMS. The installation is exclussively a function of the DBA. As mentioned earlier the ENDBMS uses 19 direct access NOS files which are also called physical units. This

group of files is subdivided into the system files and the data base files; with the first being storage for system information (i.e., internal directories) and the latter being storage for the data base information. The installation process consists of the following steps:

1. File creation and organization
2. Loading of the DBA's library
3. Writing of all ENDBMS error messages

The first step is completed by executing the procedure file INSTALL. The DBA can decide at this point (or later) whether the files will be private or not - see Chapter 3. If they are to be private the DBA must give to every user explicit permission modes, for every file. As we explained in Chapter 3 'permission modes' refers to the NOS operating system and not to the ENDBMS security control; the ENDBMS security is enforced on top of this NOS security. The second step loads the DBA's library DBALIB which will be needed for the step 3 (note: all ENDBMS files are 'unloaded' after the first step and the system file SF7 must be attached with M-mode at execution time of the third step). The reason we are using two libraries is that every user should specify the 'alternate' NOS account number (i.e., the DBA's NOS account) in each of the COMPASS routines - see ATTACH macro in [NOS-2of2]. The owner of the physical units, the DBA, is not required to specify an alternate NOS account. The third step reads all error messages from the file MFILE, and writes them on the ENDBMS system file SF7

(which is attached with M-mode by referencing the DBALIB) by executing the FORTRAN program kept in WMFILE.

If for any reason the DBA wants to destroy the 19 NOS files, the procedure file - kept in file DELFILE and with procedure name DELPROC - which deletes all ENDBMS files must be executed and the installation process can start all over again. In appendix A is shown the sequence and type of NOS control statements that have to be used in order to install the ENDBMS or destroy all the ENDBMS files.

4.2 A THREE LEVEL STRUCTURE DESCRIPTION OF ENDBMS

Figure 4.1 shows the external and internal interface of ENDBMS. Both interfaces are handled by the host Operating System, NOS. The ENDBMS is running on top of the host Operating System and ENDBMS's system data as well as data base is built on top of and supported by the host O.S. file system. As explained in Chapter 2 the user has his own copy of the ENDBMS with his own work space. The ENDBMS's data - system directories - and data base are common to all users. Figure 4.2 shows a simple architecture of the ENDBMS. The user submits input (usually commands) to the ENDBMS, which is accepted and stored by the external interface handler. The input is passed to the interpreter which analyzes and recognizes valid ENDBMS commands, and accordingly calls the command's handler, which uses one of the four subsystems for further translation and - after

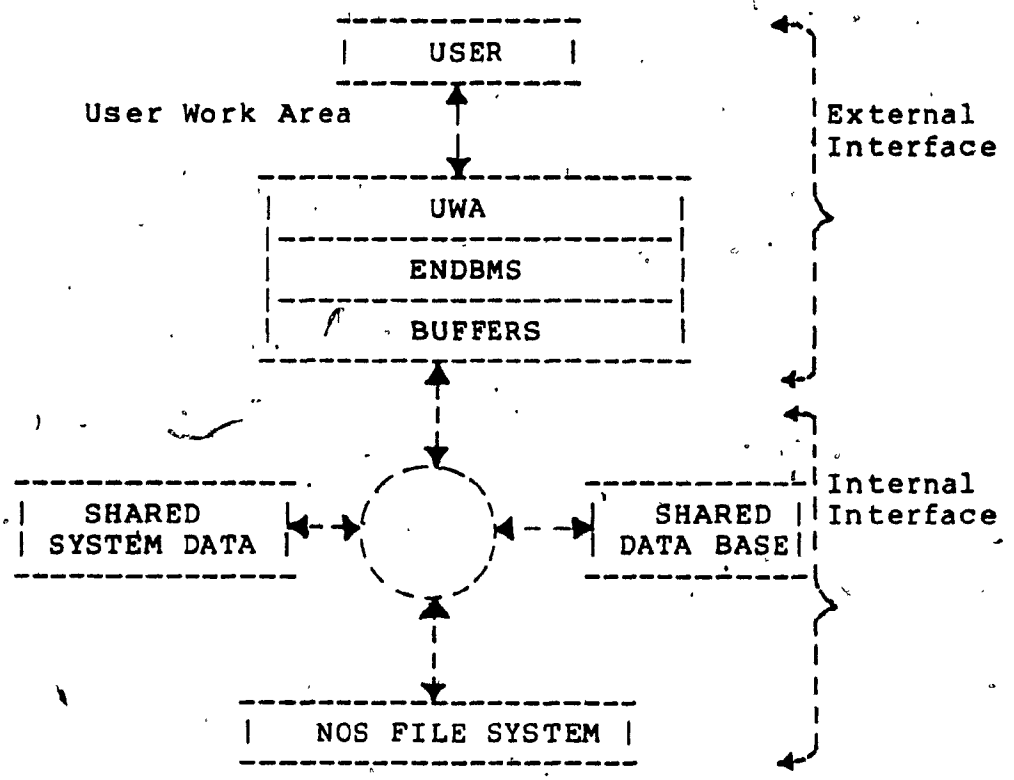


Figure 4.1 External and Internal ENDBMS interface.

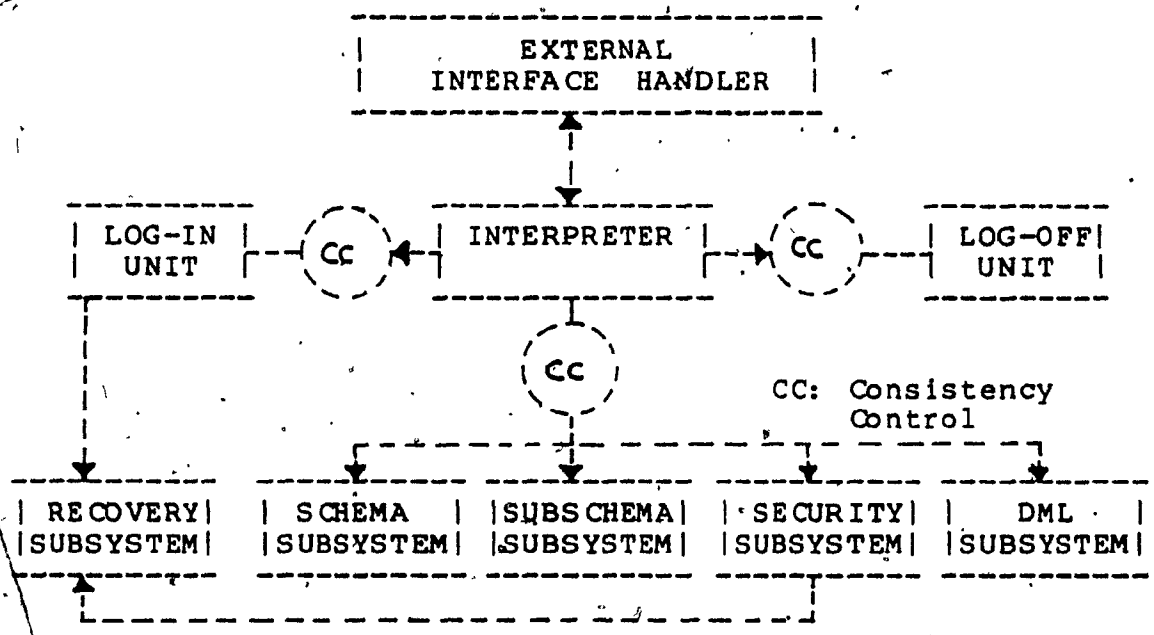


Figure 4.2 Simple architecture of ENDBMS.

successful translation - for further execution. The two log-in and log-off ENDBMS commands are handled by separate units. The recovery subsystem is called either at log-in time or by a security command to recover specific users (the recovery subsystem is called only by the DBA).

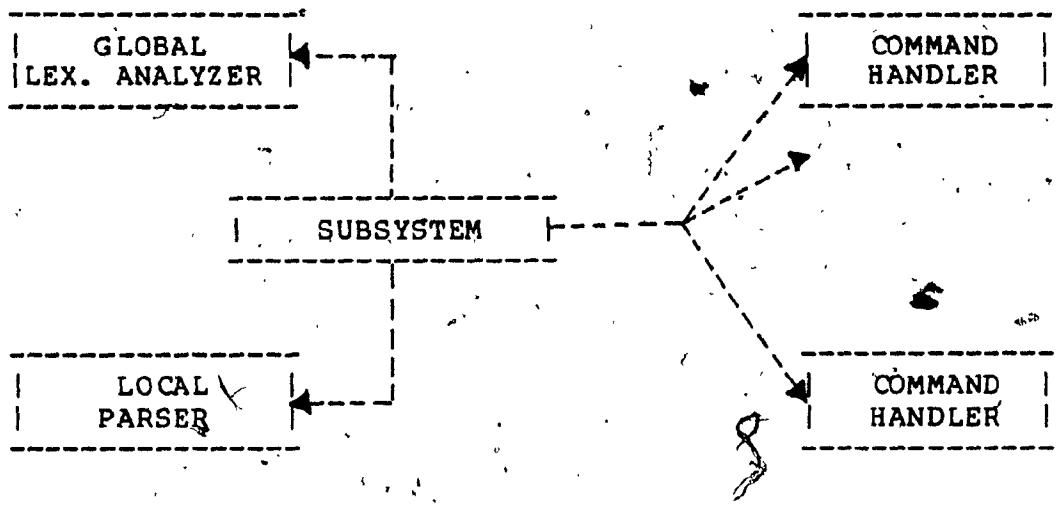


Figure 4.3 Subsystem structure.

Figure 4.3 shows the structure of each subsystem. A global lexical analyzer, used by all subsystems, scans the input and recognizes tokens (words, numbers and special symbols accepted by the ENDBMS). The subsystem consists of a number of command handlers - one for each command that belongs to the subsystem. A local parser is used to check the syntax of each command.

4.3 LL(1) PARSING

The LL(1) parsing technique [AHO-ULM] is used extensively in the implementation of ENDBMS; in this section we explain how the LL(1) parsing is employed in ENDBMS. Each subsystem of ENDBMS was designed and implemented in a logical sequence so that we could test them. For example; first we designed the basic framework (i.e., interface, interpreter, lexical analyzer), second the schema, third the security, fourth the subschema and fifth the DML subsystem. The reason for this is that the security subsystem requires the schema subsystem, the subschema subsystem requires the security subsystem, and finally the DML subsystem requires the subschema subsystem. For example execution of a DML command, let us say: 'list all occurrences of some subschema record type file', submitted by a user, would require the existence of his subschema, which would require the existence of his access profile, and which finally would require the existence of the schema. All the commands of each subsystem constitute a sublanguage expressed by an LL(1) grammar. A grammar is written in LL(1) format if all the first terminals, of all productions derived from the same nonterminal, either explicitly or implicitly, are mutually exclusive. Each of the four LL(1) grammars formally describe all the commands belonging to the corresponding subsystem. All formal grammars are given in

appendix B. Each grammar is described as a production set, a set of nonterminals (one of them will be the 'start' nonterminal) and a set of terminals (word terminals and special terminals). Word terminals are the reserved key words and user defined 'names', both of them start with an alphabetic character, while special terminals are the '(', ')', ';', '=', '<', etc). Details are given in the source code.

The N nonterminals of a grammar are assigned to an integer code from 1 to N and the M terminals to an integer code from 1 to M. Each grammar is implemented by three tables, that is the production table, the word terminal table and the parsing table. The production table is a two dimensional array and each of its rows is used to store one production rule as a string of positive and/or negative integers. For example, if a and b are terminals with codes 5 and 10 respectively, and S, A, B are nonterminals with codes 1, 3, 5, then, the production rule of some grammar:

S --> AaBb

can be stored as: -3 , 5 , -5 , 10

There is a one to one correspondence between the production rules of an LL(1) grammar and the rows of the production table. The word terminal table is a one dimensional array and is used to store the word terminals of a grammar in ascending order. The position of a word terminal within the word terminal table indicates its code. The parsing table

is a two dimensional array, with number of rows equal to number of nonterminals, and number of columns equal to number of terminals of a grammar. Each entry of this table is either zero, or negative integer, or non-zero positive integer.

If during the parsing of a command a zero entry is selected, an error situation is considered to have been encountered, while if a negative entry is selected then an 'empty' production rule is to be used. Finally if a possitive entry is selected, then the positive integer indicates the production rule which is to be used next; the integer is used as row index in the production table to access the corresponding production rule. A stack, common to all subsystems, is used for parsing of any command. The top of the stack is either a terminal or a nonterminal. Terminals are represented on the stack with their integer codes, while the nonterminals are represented by the negative of their respective codes. The start nonterminal of any grammar has code 1 and because any command is derived directly from the start nonterminal, the stack top is initialized to minus one before the command's syntax analysis starts. A global lexical analyzer is used to get the next token with a global code (see section 4.5). This token is looked up in the local word terminal table using a binary search and a local code is assigned to it. The local code will be zero if the token is not a valid one for a

particular subsystem.

The parsing, or syntactic analysis, of any command is carried out by using (1) the parsing table, (2) the parsing stack and (3) the current input token, which was returned upon request to the lexical analyzer. The current input token and the top of the stack determine the next action. If the top of the stack is a positive integer, i.e., a terminal, the current input token should match with it, that is its code must be the same as the the top of the stack, otherwise an error situation is found. In case of an error, a message is generated indicating what caused the error, and a global error condition flag is set to 'ON'. However, if the ENDBMS is used in I-mode (interactive), the editor facility is called to correct or drop the command under examination, and if the ENDBMS is used with B-mode (batch), the error flag remains on, but the current input token is checked if it matches the expected next token; if it does the parsing of the command continues, otherwise it is terminated and the parsing of the next, if any, command is initiated.

Let us now examine the case where the top of the stack is a negative integer, i.e., a nonterminal. Let I be the absolute value of the top of the stack and J be the local code of the input token. The local parsing table is consulted and its entry - with row index I and column index J - is considered. If the selected entry value is zero, the

parsing is terminated in case of the B-mode; or in the case of I-mode the editor is called to allow the user to correct the error. A pop-off operation is performed on the parsing stack in the case where the selected entry value is negative. (Note: this action is taken for an 'empty' production.) Finally there is the case where the entry value is a positive integer, indicating the unique production number of the production rule which must be used next. This production rule is taken from the production table and it is transferred onto the parsing stack. The transfer is accomplished with a POP and some PUSH operations on the parsing stack, bringing the begining of the production rule on top of the stack. The new top is tested with the current input token and the above process is repeated. The lexical analyzer is called any time the current input token matches the top of the stack. All the above described actions take place within the local parser itself, and only if there is a match between the top of the stack and the current input token, control is returned to the command handler. At this point the command handler performs all the semantic actions, if it is necessary, and then calls the parser again, until the stack becomes empty or a forced termination of the parsing is encountered.

Once the end of the stack is reached, the command handler checks the error condition flag, and if it is 'off' then the execution of the command can start, otherwise

control is returned to the interpreter. Each command handler, before the parsing or execution starts, performs some consistency checks, namely, user identification and execution requirements for the command. It also generates error messages during the extensive semantic analysis and uses the editor for possible error correction. After the execution phase is finished, control is returned again to the interpreter which checks if there are any more input commands; if there are none the system asks for more input by dispaying the message 'READY' and immediately after the 'input prompt character': '?'. The user can either start a new 'input session', (see next section), or terminate his interface with ENDBMS by issuing the 'OUT' command.

4.4 INPUT SESSIONS

The ENDBMS can be accessed either through a terminal or through a card deck, in either B-mode (batch) or I-mode (interactive). The difference between B-access and I-access is that B-access cannot use the editor facility which provides the terminal user with the ability to correct errors in a commnd's format, during its translation period, without having to rewrite the entire command. Obviously when accessing the ENDBMS through a card deck, the specified processing mode must be the B-mode, because the user has no way to intervene and correct possible errors. A card deck user should submit all the ENDBMS commands to be executed at once with the first being the 'USER' and the last being the

'OUT'. In addition to the editor option, a terminal user has the option to organize his commands in groups and submit one group at a time. The ENDBMS processes all the commands of a group and then asks for more, if last processed command was not the 'OUT'.

The entering of such a group of commands is called an input session. An input session is initiated after the system displays the prompt word 'READY', and it is terminated by the dollar sign ('$') character. The input submitted during an input session, may include one or more ENDBMS commands; a command separator - the semicolon ';' - should always separate two commands. The first command of the first input session must be 'USER', and the last command of the last input session must be the 'OUT' command, because every ENDBMS command, except 'OUT', requires proper log-in, i.e., execution of the 'USER' command. Every input line could have any number of leading blanks, after the host system's prompt character '?', and every command should start on a new line. The maximum length of an input line is 80 characters, with all the rest ignored. The body of a command can be spread over many lines, with only one restriction which gurantees that the command's semantic analysis will be successful after an error has been found. If a command includes no errors the restriction can be overwritten. Here follows the restriction and its explanation.

If an ENDBMS command contains a 'name' (e.g record, set or field name), the 'name' with its preceding reserved word must be on the same line. The reason is that the reserved word is used to set 'ON' an internal flag, kept by the concerned command handler. If the name is on another line and an error occurs later on this line, the line will be replaced by the editor either by a 'DROP' indication or by a corrected line. Here is a possibility for a problem. Some times the flag which was set by the preceding reserved word is reset immediately after the expected 'name' is found. Suppose a 'name' exists on a certain line and its preceding key word exists on the previous line. The flag which was set by the key word is reset after the name is found in the next line. Let us now assume that an error occurs later on the same line with the 'name', and a new line is submitted with a new 'name'. The new 'name' will never be stored (i.e., will not reflect the change), because the flag is stil reset and the new 'name' will be ignored). This is an extreme case which can happen during a schema or subschema field definition, for example. A user is expected to submit the same correct portion of an erroneous line, while he inserts a new corrected line.

EXAMPLE:

1. (safe) FIELD: SALARY OF NUMERIC (3.55)

corrected line

FIELD: SALARY OF NUMERIC (5)

or

FIELD: EMP-SAL OF NUMERIC (5)

Both of the corrected lines will cause no problem, and the changed field name will be stored as such.

2. (not safe) FIELD:
SALARY OF NUMERIC (abc)
corrected line
EMP-SAL OF NUMERIC (6)

The corrected line will not cause any translation problem, and the user will beleive that he changed the field name SALARY to EMP-SAL. However the ENDBMS will not record this change; because the key word FIELD is not on the same line and because the field name flag is reset (from the initial scan of the line with the error), the new field name will be ignored.

4.5 VALID INPUT TOKENS

The user submitted input consists of a number of tokens - recognizable units by the global lexical analyzer. A token can be one of the following:

1. Reserved word.
2. Name. A name should start with a letter and can be following by 'some' numeric and/or alphabetic characters, including the '-'. The max length can be 80 characters, equal to the max input line length. Only the first 20 characters of a 'name' are stored and only the first 10 are significant

(used for comparisons).

3. Number. A number can be an integer or a real with a maximum of 14 significant digits. The accepted formats for a real are: n.m, n., .m

4. Special Tokens.

' ; ' semicolon (command separator).

' $ ' dollar sign (input terminator).

' . ' period.

' " ' quote.

' + ' arithmetic plus.

' - ' arithmetic minus.

' * ' arithmetic times.

' / ' arithmetic divide.

' (' left parenthesis.

') ' right parenthesis.

' = ' equal.

' < ' less than.

' <= ' less than or equal.

' > ' greater than.

' >= ' greater than or equal.

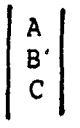
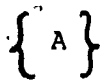
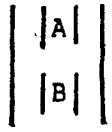
' <> ' not equal.

Blanks, colons and commas are valid characters and they are not tokens. They can be used for 'readability' purposes and they are skipped during line scanning. However they cannot be included within the above tokens. A reserved word, a name and a number must be preceded and followed by either at least one blank, or a comma, or a colon, or a special token.

The first token on a line can start immediately from the first character. Each of the above tokens is assigned an global integer code, which will be changed to a local one, by each subsystem.

4.6 SCHEMA SUBSYSTEM

The schema subsystem consists of six command handlers related to the schema concept supported by the ENDBMS. This group of command-utilities has been designed to be used, solely by the DBA, as tool to describe the logical organization of the data base and furthermore to handle its administration - i.e., making changes to schema definition. An LL(1) schema sublanguage describes formally the schema group of commands and it is given in appendix B. In this section, we first explain the data structures used to represent the schema internally, and then we desribe the schema commands. In order to describe a command in BNF (Bacus Norm Form) form we make use of the following symbols:

	: A or B or C (one of A,B,C)
	: A is repeated ZERO or MORE times.
	: none, or one, or both.

4.6.1 SCHEMA INTERNAL REPRESENTATION

For the internal representation of the schema definition one system file ('SF1') is used. In this file the ENDBMS keeps all the information about the schema and it uses five singly linked lists, as shown in Figure 4.4; with the lists being: (1) the schema record type list (schema RT-list) to link all record type entries, one for every record definition; (2) the field name list, for every record type, to link all field name entries, corresponding to each field definition; (3) the schema set type list (schema ST-list) to link all set type entries, one for every set type definition; (4) the RT-S.O.L., record type - set ownership list, for every record type; (5) the RT-S.M.L., record type - set membership list, for every record type. Each entry of the RT-S.O.L. indicates the set type where the corresponding record type is owner and it also indicates the associated member record type. The number of entries in a RT-S.O.L. is equal to the number of set types where the corresponding record type is owner. Similarly, an entry of the RT-S.M.L. indicates the set type, as well as its owner, where the corresponding record type is the member type and again, the number of entries in a RT-S.M.L. indicates the set types where the corresponding record type is a member. The RT-S.O.L. and RT-S.M.L. are the ownership and membership lists at the 'type-level' associated with a schema record type. We will find later (see section 4.9.2) the

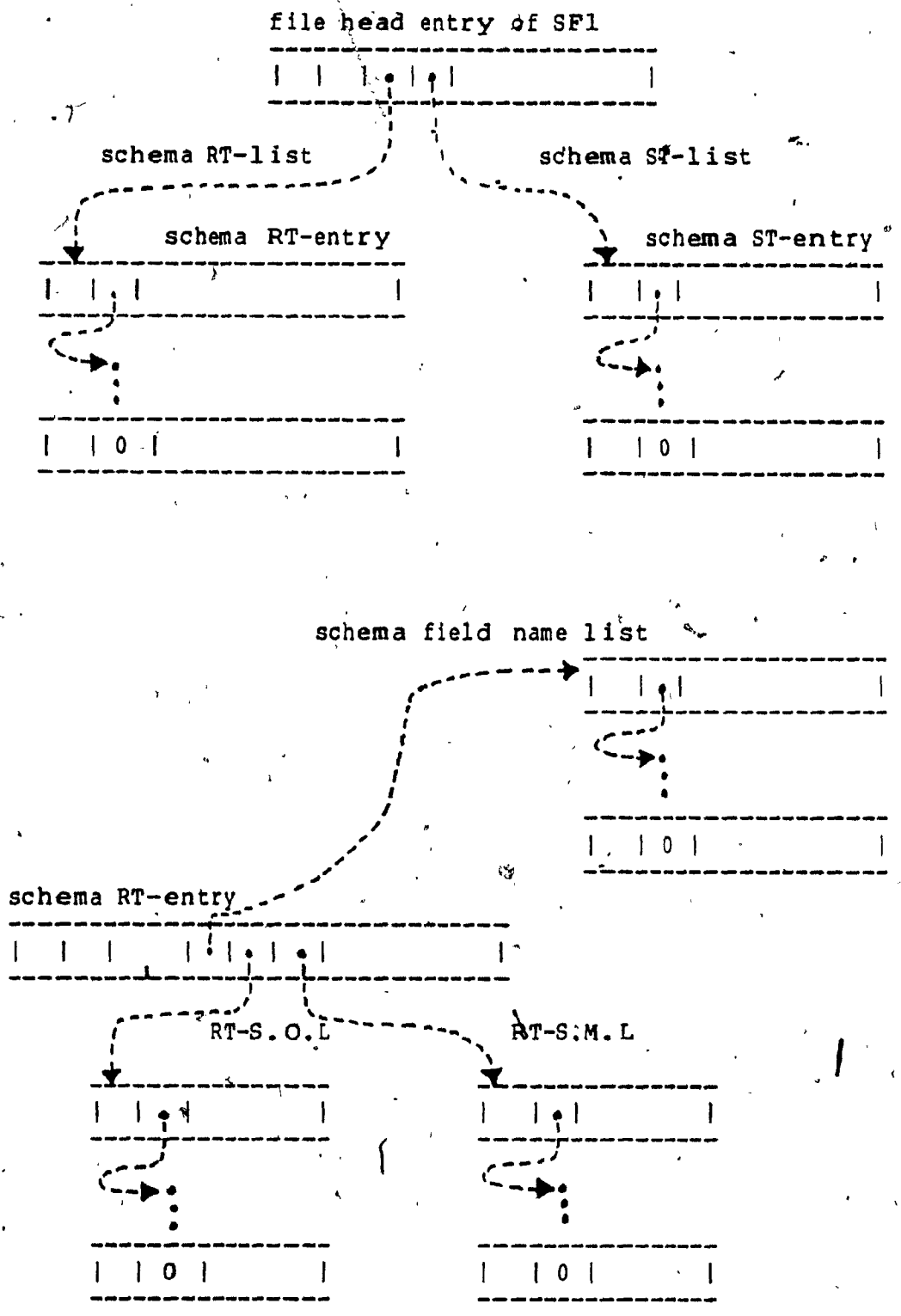


Figure 4.4 Schema internal representation.

RTO-S.O.L. and RTO-S.M.L. for the ownership and membership lists at the 'occurrence-level', associated with a record occurrence of a record type file.

In the rest of this section we describe the exact format of every entry, which is stored in 'SF1'.

(1) FILE HEAD ENTRY.

This entry is used by the ENDBMS to generate unique keys for every new entry written on this file and to generate a unique 'record type number' for any new record type definition stored in the schema definition. Both keys and 'record type numbers' are integer numbers which are generated by increasing their 'installation' values. The unique keys are required by the CRM functions which create and retrieve entries of this file. It also contains some other information as it is shown below and it is generated at installation time with initial field values: 1, 1, 0, 0, 0, 0.

KEY	KGEN	RTLH	STLH	RTNG	RF
1	2	3	4	5	6

field description:

1 : key field; it is used by CRM for accessing.

2 : key generator; it generates unique keys for every entry written in this file. Every time a key is needed the value of this field is incremented by one.

3 : record type list head pointer; it is set to zero when the list is empty.

4 : set type list head pointer; it is set to zero when

this list is empty.

5 : record type number generator.

6 : recovery field; it is used for recovery purposes.

(2) PHYSICAL UNIT DISTRIBUTION ENTRY

This entry is generated during the execution of the schema definition command, with initial field values 2,0,0,0,0,0,0,0,0,0. It is a known entry, because its key is the integer 2. The last ten fields correspond to the 10 data base NOS files.

KEY	DF1	DF2	...	DF10
1	2	3	...	11

field description:

1 : key field.

2,11: ten flags corresponding to ten data base physical units; if the n-th flag is set to 1, then the (n-1) physical unit (PU) is allocated for the current 'rotation'. A uniform distribution algorithm is used for the allocation of the data base PU's to the schema record type files. The first 10 record type files are assigned to PU from 1 to 10, one at time by setting the corresponding flag. The 11-th record type file is assigned to PU-1 and causes reseting of all flags except the first. The same happens for the 21-st record type file. The 12-th record type file, is assigned to PU-2, the 13-th to PU-3, etc.

(3) RECORD TYPE ENTRY

This type of entry is used to save all information about a schema record type definition; it is generated by the ENDBMS after a successful translation of a record type definition.

KEY	LINK	r-name	FLH	OLH	MLH	RT#	PU#	RTL
1	2	3 4	5	6	7	8	9	10

field description:

1 : key field.

2 : link pointer for the schema RT-list; it points to the next record type entry if it is not zero.

3,4 : schema record type name (s-r-name).

5 : field list head pointer; it points to the first entry of the field name list.

6 : head pointer for the RT-S.O.L.; it is zero if the list is empty.

7 : head pointer for the RT-S.M.L.; it is zero if the list is empty.

8 : record type number.

9 : physical unit number; it is an integer from 1 to 10 and indicates the NOS file allocated for the storage of the corresponding schema record type file. It is generated during the execution of a record type definition by a 'rotating' algorithm.

10 : record type length; it is the sum of all field lengths and it is updated any time a field definition is stored.

(4) FIELD NAME ENTRY

This entry is generated by the ENDBMS, during the execution of a schema record type defintion, and corresponds to a schema field definition; it is used to store all the information for a schema field definition.

KEY	LINK	f-name	TYPE	LENGTH	PRTL
1	2	3 4	5	6	7

field description:

1 : key field.

2 : link field; it points to the next field name entry if it is not zero.

3,4 : schema field name.

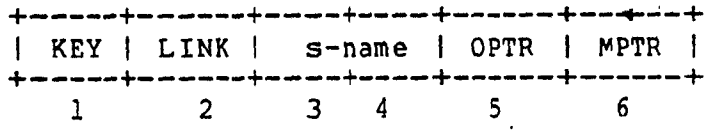
5 : field type; 1 for CHAR and 2 for NUMERIC.

6 : field length; it must be an integer.

7 : previous total field length; it indicates the total length of all previous fields and thus it is zero for the first field.

(5) SET TYPE ENTRY

This entry is generated by the ENDBMS, after a successful translation of a schema set type definition, and it is used to store all the information needed for a schema set type definition.



field description:

1 : key field.

2 : link pointer used by the schema ST-list; if it is not zero, it points to the next schema set type entry.

3,4 : schema set type name (s-s-name).

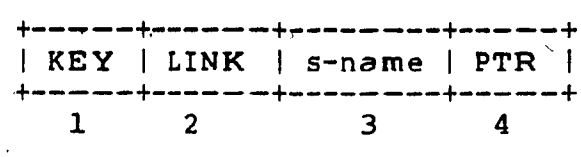
5 : pointer to the OWNER record type entry.

6 : pointer to the MEMBER record type entry.

(6) RT-S.O.L. or RT-S.M.L. ENTRY

Such entries are generated as part of the execution of a set

type definition. The set type definition indicates the OWNER and MEMBER schema record types. An entry is inserted in the owner's RT-S.O.L. and another in the member's RT-S.M.L.



field description:

1 : key field.

2 : link pointer; if it is not zero, it points to the next entry of the same list.

3 : set type name (only the first 10 significant char's).

4 : it points to the member record type entry (in the schema RT-list) for a RT-S.O.L. entry, and it points to the owner record type entry (in the schema RT-list) for a RT-S.M.L. entry.

Note:

Insertions to all mentioned lists take place at the top of each list.

4.6.2 SCHEMA COMMAND GROUP

In this section all the schema commands are described and their syntax in BNF form is given.

DEFINE-SCHEMA

The DEFINE-SCHEMA command is the schema definition command. It may include at least one record type definition, and optionally set type definitions. It is intented to be used only to define the logical organization of the data base.

Once the command has been used to define the schema and the schema contains at least one record type definition; then, it CAN NOT be executed again; there are other commands to update the schema (see below).

Use: for DBA only.

Requirements:

1. DBA identification.
2. the ENDBMS must be 'locked', see security.
3. the schema record type list must be empty (which implies an empty set type list).
4. unique record and set type names, and unique field names within a record type.
5. the owner and member type specified in a set type definition, must have been defined before.
6. the length of a field must be a positive integer.

Syntax: DEFINE-SCHEMA <schema-definition> ;

<schema-definition>:= <record-type-definition>

<record-type-definition>

<set-type-definition>

<record-type-definition>:= RECORD <s-r-name>

<field-definition>

<field-definition>

END

<field-definition>:= FIELD <field-name>

OF | CHAR | NUMERIC | (<integer>)

<set-type-definition>:= SET <s-s-name>

OWNER <s-r-name>
MEMBER <s-r-name>

END

INSERT-SRT

This schema command can be used to 'expand' the schema; it can insert ONE record type definition at a time.

Use: for DBA only.

Requirements:

1. DBA identification.
2. unique record type name and unique field names within the corresponding record type.

Syntax: INSERT-SRT <record-type-definition> ;

Note: the <record-type-definition> is given in the description of the DEFINE-SCHEMA.

INSERT-SST

The 'insert a schema set type' command it can be used to add a new set type in the existing schema definition. It adds only one set type at a time.

Use: for DBA only.

Requirements:

1. DBA identification.
2. the set type name must be a new one; duplicate set type names, in the schema ST-list, are not allowed.
3. the set type must be defined over two schema record types, e.g., owner and member, which must have been predefined.

Syntax: INSERT-SST <set-type-definition> ;

NOTE: the <set-type-definition> is given in the description of the DEFINE-SCHEMA.

DELETE-SRT

The 'delete a schema record type' command can be used to delete a schema record type definition (one at a time). Its deletion causes triggered deletion of all schema set type definitions, which include it as an owner or as a member, and furthermore in order to maintain consistency in schema-profiles-subschemas, it updates the affected profiles and their corresponding subschemas. The corresponding stored record type file is deleted and its data base space is freed.

Use: for DBA only.

Requirements:

1. DBA identification.
2. the ENDBMS must be 'locked'.
3. no user 'activity' at all.
4. the specified record type name must be valid.

Syntax: DELETE-SRT <s-r-name> ;

DELETE-SST

The 'delete schema set type' command can be used to delete a schema set type definition (one at a time), without effecting the involved owner and member types. The independent implementation of the set type association

allows its independent destruction. A deletion of a schema set type updates, as in the case of the DELETE-SRT command, all necessary user profiles and their corresponding subschemas. An important action which takes place during the execution of this command is the update of the RT-S.O.L. and all RTO-S.O.L. for the owner record type, as well as the update of the RT-S.M.L. and all RTO-S.M.L. for the member record type.

Use: for DBA only.

Requirements:

1. DBA identification.
2. the ENDBMS must be 'locked'.
3. no user 'activity' at all.
4. valid set type name.

Syntax: DELETE-SST <s-s-name> ;

Note: both the DELETE-SRT and DELETE-SST schema commands are considered to be powerful operations and they have to be given a lot of thought, before using them.

LIST-SCHEMA

This schema command is designed to give, at any time, a list of the schema definition. It first lists all the record type definitions and then, if any, all the set type definitions.

Use: for DBA and any authorized user.

Requirements:

1. DBA or user identification.

Syntax: LIST-SCHEMA ;

4.7 SECURITY SUBSYSTEM

The security subsystem constitutes a group of 15 commands, carefully selected and designed to handle all the security needs in ENDBMS. The DBA is responsible for the overall security enforcement; he makes all the decisions about which users will have the right to access the ENDBMS and what exactly each of them should be allowed to do. The 15 security commands can be divided into three subgroups. With the first the DBA 'installs' the users - i.e., gives them the right to access the ENDBMS, 'removes' installed users from the system, gets a list of all installed users at any time, 'suspends' some users from accessing the system for certain period of time, and finaly 'restores' their access to the system. With the second subgroup the DBA defines, updates and lists user access profiles of users that have been installed; with the third subgroup the DBA exercizes some system control by locking and unlocking the ENDBMS and explicitly requesting recovery action for specific users.

4.7.1 SECURITY IMPLEMENTATION

Security in ENDBMS is implemented by means of a USER LIST and the USER ACCESS PROFILE; two of the system files, SF5 and SF6 respectively, are used to store the user list and the user access profiles. The user list is represented

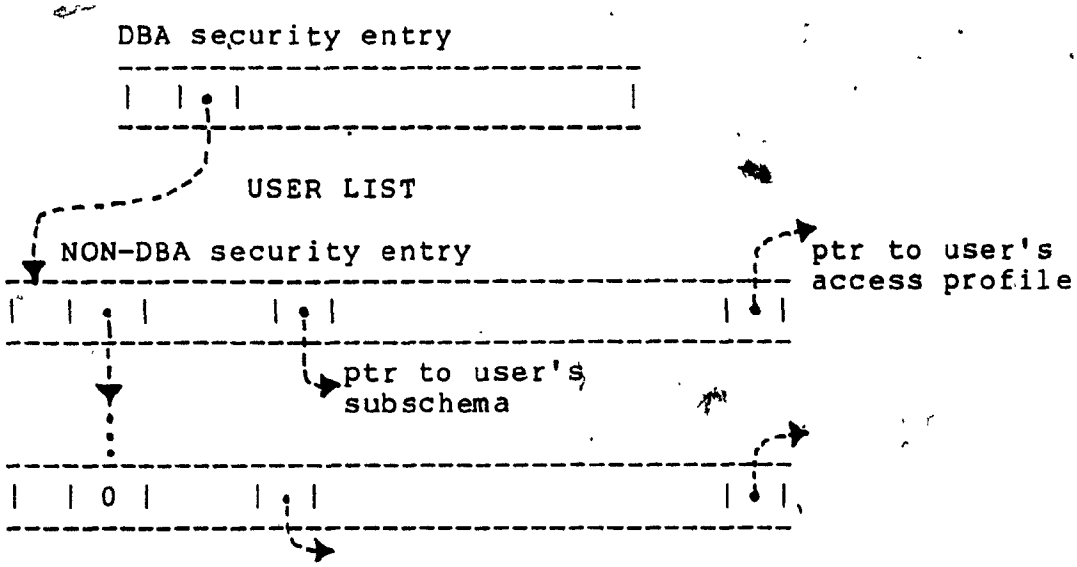


Figure 4.5 Representation of USER LIST.

as a singly linked list (see Figure 4.5) with a head entry (the DBA entry) which is generated at installation time and it is used by the ENDBMS to record the 'status' of the DBA. A user is considered 'installed' if an entry - i.e., a user security entry - has been created for him in the USER LIST. This user security entry is used by the ENDBMS to record the 'status' of the user at any given time and along with the user access profile, is used to enforce all the system provided protection for the particular user. One field value of the user security entry is used as a pointer to the user's access profile. The latter is represented by a null head entry and two singly linked lists (see Figure 4.6), e.g., the profile record type list and the profile set type list. Every entry of the profile record type list has five branch lists, namely the RFR-list, MFR-list, ROR-list, MOR-list and the DOR-list. These lists implement the FIELD and OCCURRENCE RESTRICTIONS with respect to a schema record

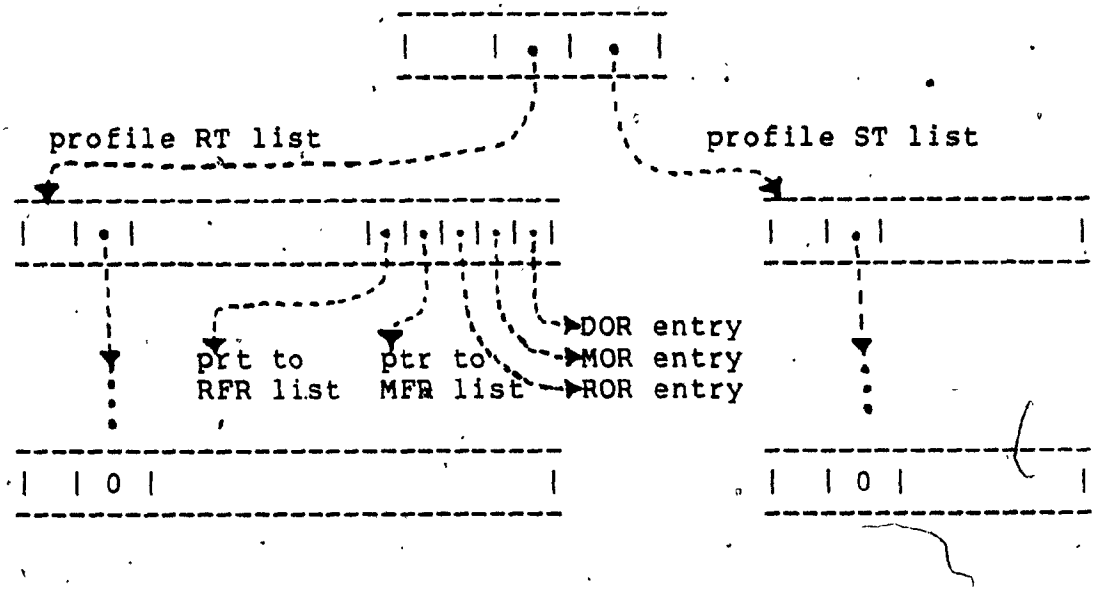


Figure 4.6 Representation of USER ACCESS PROFILE.

type entry. Given below is a format description, of all entries of the system file five and system file six, used to create the user list and the user access profile.

4.7.1.1 STRUCTURE OF SYSTEM FILE FIVE

FILE HEAD ENTRY

This is generated at installation time and has the following format:

KEY	KGEN	SLOCK	SRF	RF
1	2	3	4	5

field description:

1 : key field.

2 : key generator; it is used to generate unique keys for every new entry created on this file. Every time a key is required the field value is incremented by one.

3 : system's lock-flag; when it is set to 1 no one except the DBA can use the ENDBMS.

4 : system's recovery-field; it is not used by the

current version of ENDBMS (see section 4.12.5).

5 : recovery field; used by the recovery subsystem.

The values stored at installation time in the three fields are: 1 , 2 , 0 , 0 , 0 respectively. The key 2 is used for the DBA security entry.

DBA SECURITY ENTRY

It is generated at installation time and has the following format:

KEY	ULH	DBA#	DBAPW	ACTIV	STATUS	RF1	RF2
1	2	3	4	5	6	7	8

field description:

1 : key field.

2 : pointer to the first user security entry in the user list; a zero value indicates that the user list is empty.

3 : DBA user number.

4 : DBA user password.

5 : DBA activity flag.

6 : DBA status field; a non-zero value is a command recovery code and it indicates the recovery-needed command being executed by the DBA at a given time.

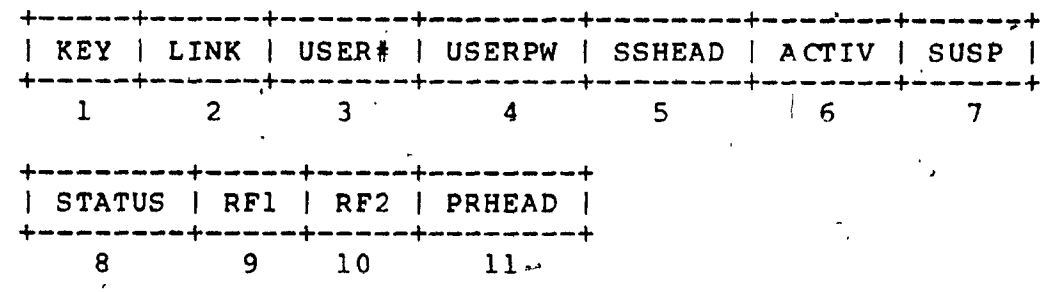
7,8 : recovery fields; used for DBA recovery.

The values stored at installation time are: 2, dbano, dbapw, 0, 0, 0, 0 respectively, where the dbano and dbapw stand for the DBA's user number and password.

USER SECURITY ENTRY

It is generated when the DBA installs a new user, with the

following format:



field description:

1 : key field.

2 : link field; if it is not zero, it points to the next user security entry in the user list.

3 : user's user number.

4 : user's user password.

5 : pointer to user's subschema null head entry, if it is not zero.

6 : user suspension flag; if it is set to 1 the corresponding user is considered suspended.

7 : user activity flag; it is set to 1 when the user logs-in and it is reset to zero when the user logs-off.

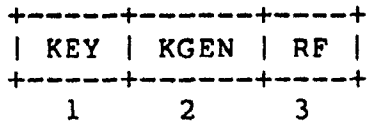
8 : user status field; a non-zero value is a recovery needed command code and it indicates the type of command being executed by the corresponding user.

9,10: user recovery fields; are used to store the pointer(s) of the 'logical start' of any 'insert' or 'delete' type recovery needed command (see recovery subsystem in this Chapter).

11 : if it is not zero, points to the null head entry of the user's access profile.

4.7.1.2 STRUCTURE OF SYSTEM FILE SIX

FILE HEAD ENTRY



field description:

1 : key field.

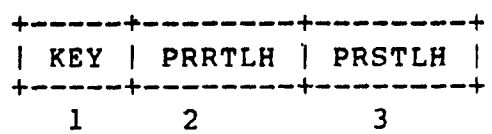
2 : key generator field; same as before.

3 : file's recovery field.

It is generated at installation time with field values 1, 1, 0 respectively.

PROFILE NULL HEAD ENTRY

This entry is generated when the DBA defines the access profile of the corresponding user, and it is destroyed when the user's access profile is deleted. Its format is the following:



field description:

1 : key field.

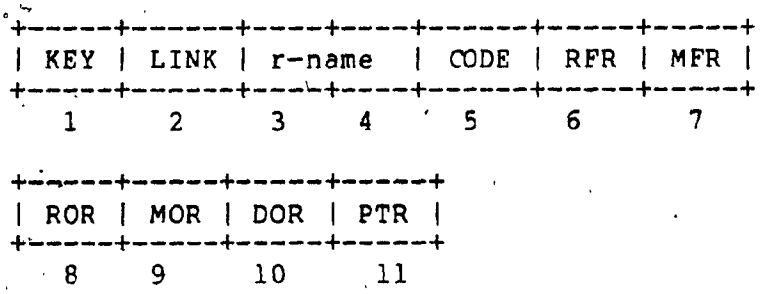
2 : it points to the first entry of the profile record type list; a zero value indicates that this list is empty.

3 : it points to the first entry of the profile set type list; a zero value indicates that this list is empty.

PROFILE RECORD TYPE ENTRY

This entry is generated when the user's access profile is

defined, or access rights to an additional record type are added to the user's profile.



field description:

1 : key field.

2 : link field; if it is not zero it points to the next profile record type entry.

3,4 : schema record type name corresponding to this profile record type entry. The reason we are using two words for the name is that a max of 20 characters can be used for any user defined name and with only the first 10 being significant.

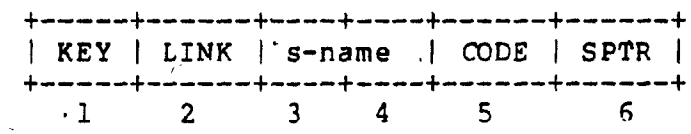
5 : record type access code (see section 4.7.2.3).

6,7,8,9,10: if they are not zero, they point to the tops of the RFR, MFR, ROR, MOR and DOR lists respectively.

11 : pointer to the corresponding schema record type entry of the schema RT-list.

PROFILE SET TYPE ENTRY

This entry is generated either when the user's access profile is defined - provided that it includes set type defintitions - or access rights to an additional schema set type are added to the user's access profile. It is deleted either explicitly by deleting a profile set type entry or implicitly as a triggered action of a deletion of a schema record or set type entry.



field description:

1 : key field.

2 : link field; used by the profile set type list.

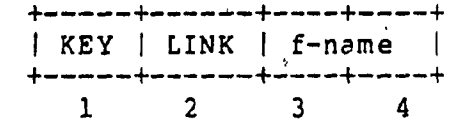
3,4 : the name of the set type (same comments as before).

5 : set type access code (see section 4.7.2.4).

6 : pointer to corresponding schema set type entry of the schema ST-list.

RFR or MFR ENTRY

These entries are generated along with a profile record type entry and each of them is used for one read or modify 'field restriction'. They are deleted along with their parental profile record type entry.



field description:

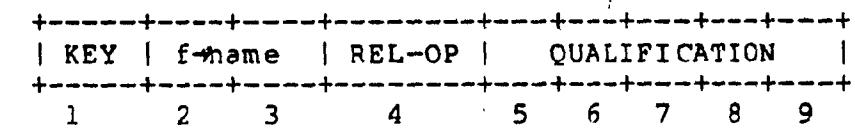
1 : key field.

2 : link field; if it is not zero it points to the next (read or modify) field restriction entry in its own list.

3,4 : the restricted field name.

ROR or MOR or DOR ENTRY

These entries are generated and deleted along with their parental profile record type entry. They are used to implement the Read, Modify and Delete Occurrence Restrictions respectively.



field description:

1 : key field.

2,3 : field name used to define the restriction criterion.

4 : relational operator used in the restriction criterion

5,...,9 : these five fields (5 CDC words) are reserved to store the value specified in the R or M or D, ROR clause of a profile record type definition.

4.7.2 SECURITY ENFORCEMENT

Security in ENDBMS is enforced through a five level hierarchy given below:

1. Identification and Authentication Level.
2. System Availability Level.
3. Record Type Level.
4. Set Type Level.
5. Record Occurrence Level.

Access rights at level (1<i<5) require access rights at all its lower levels. Access rights at the record occurrence level require access rights at levels 1-3 for record type operations. However, if a record occurrence is to be retrieved using set type operations, access rights over the corresponding set type are required. For example let us consider two record types, EMPL and DEPT and the set type DEP-EMP. Let users X and Y have access rights for all occurrences of the two record types and let user Y have, in addition, the access rights over the set type DEP-EMP. Both

users can access any occurrence of the two record types but only user Y can make use of the logical association between them, by using set type operations, or define a virtual subschema record type by selecting fields from EMPL and DEPT record types. Before we describe each level we make an important observation: security enforcement at the first four levels does not require any 'search' operations at the 'stored' data base level, however to enforce security at the fifth level, i.e., to decide whether to allow access to a given record occurrence, we must first access that record occurrence. According to this observation the security enforcement is divided into two categories: the data independent security enforcement and the data dependent security enforcement. The word 'data' refers to the stored information in the data base.

Once a user has been installed and his/her access profile has been defined, he/she can create and administer his/her own subschema. In the data independent security enforcement, a user's request can be granted or denied by consulting the user's subschema and access profile. Any ENDBMS command issued by a user is analyzed for lexical, syntactic or semantic errors during the first pass over the command's text or the so called 'translation' period. If the command is error free then the second phase (execution phase) of the command starts. Some times it is impossible to keep the information, needed for the execution phase,

from the first pass and a second pass over the command's text will be required. The second pass does not require any error checking. The error analysis of any of the DBA's commands is done, at translation period, against the schema. For only retrieval DML commands, the execution phase might be interrupted because of security enforcement with an informative message, which explains why the interrupt took place. The rule to enforce security for retrieval of a subschema virtual record occurrence is the following: if any subschema record occurrence is constructed from more than one schema record occurrence, and if any one of these is not accessable, because of some restrictions, the entire subschema record occurrence is not available.

Generally access to a data base record occurrence requires security enforcement at the record occurrence level, except for the DBA, that is the user's access profile is searched to see if any record occurrence restrictions exist, and if it does then it is determined if the current record occurrence satisfies the restriction criterion. This determination requires the contents of the current record occurrence, which requires 'fetching' and 'reading' of this record occurrence from the data base. Let us consider the following example: a user is permitted to read the 'SALARY' field of the record type 'EMPLOYEE' but for only some EMPLOYEE occurrences. This user's request, such as "list all EMPLOYEE records" cannot be denied nor granted

unquestionably. If a look-up of the user's subschema indicates that the requested record type belongs to his view, and if there are no errors during the translation phase, the execution phase will be initiated. Security enforced up to this point constitutes the data independent security enforcement. At execution phase, before an EMPLOYEE occurrence is listed, a check is made to see if the record satisfies any restriction criterion, and if it does it is not listed, otherwise it is listed; this operation is repeated for each EMPLOYEE record occurrence.

4.7.2.1 IDENTIFICATION AND AUTHENTICATION LEVEL

Every user who wants to use the ENDBMS, including the DBA, must identify himself using the 'USER' command. The DBA, who is installed at system installation time, can perform any operation in the system after his identification. All other users must have been installed by the DBA with appropriate access profiles.

4.7.2.2 SYSTEM AVAILABILITY LEVEL

After the identification and authentication is completed, the ENDBMS checks the status of the system. The entire system may not be available because it is 'locked' by the DBA for system maintenance, or the system may not be available to a particular user. The latter can happen because the user has been suspended by the DBA, for maintenance of his profile, or because the user's activity

flag of his security entry in the USER LIST is set. The setting of the activity flag means either the user attempts to use the ENDBMS from another terminal, which is not permitted, or the user in his last session with the ENDBMS did not clear-off his status by a normal log-off operation - i.e., 'OUT' command; in any case a message for recovery is displayed, the user is not allowed to proceed further, and the ENDBMS is disconnected from the user.

4.7.2.3 RECORD TYPE LEVEL

If a schema record type is included in a user's profile, the user has access to the record type. The access rights can be any combination of Read, Insert, Modify and Delete operations. Each combination is associated with a record type access code and this code is stored in the user's access profile. As indicated earlier, the access rights of a record type may involve field restrictions for read and/or modify, as well as restrictions on occurrences of the record type. We have two groups of legal combination of the above operations, because of the read field restrictions. If the 'RFR' set is not empty the user automatically should not have Insert and Delete access rights for the corresponding record type. Given below is a list of all access codes and legal combinations:

access code	legal combination
1	READ
2	READ + INSERT
3	READ + MODIFY
4	READ + DELETE
5	READ + INSERT + MODIFY
6	READ + INSERT + DELETE
7	READ + MODIFY + DELETE
8	READ + INSERT + MODIFY + DELETE (ALL)
9	READ*
10	READ* + MODIFY

In the first 8 cases, Read is granted with no restrictions at all, and the the entire record occurrence is accessable, if not restricted with the ROR set. In the last two combinations, the asterisk refers to Read with read field restrictions, in which case only a portion of a record type can be accessable. ROR, MOR and DOR can be used appropriately - i.e., with appropriate access codes - to define occurrence restrictions; ROR and MOR can be used with access codes 9 and 10 to restrict occurrences from reading and/or modifying.

4.7.2.4 SET TYPE LEVEL

Security at this level is enforced by means of a set type access code, stored in a profile set type entry, which is associated with a legal combination of all set type operations: Read, Include and Remove. Given below is a list of all legal combinations with their associated set type

access codes:

access code	legal combination
1	READ or READ*
2	READ or READ* + INCLUDE (see note)
3	READ or READ* + REMOVE (see note)
4	READ or READ* + INCLUDE + REMOVE

If a set type is not included in the user's profile set type list, the user has no access at all to this set type.

Note: the READ or READ* set type access rights is assumed on both the OWNER and MEMBER record types, but not necessarily to be the same. It is very important to emphasize here that the set type access codes 2 , 3 , and 4 should be given with only READ (no read field restrictions) for the MEMBER type, and with READ or READ* for the OWNER type. This restriction is obvious because a user cannot associate logically, only a part of a record occurrence (or delete the logical association for only a part of a record occurrence).

4.7.2.5 RECORD OCCURRENCE LEVEL

Security at this level is data dependent and is enforced at the execution phase of any ENDBMS DML command by making use of the ROR, MOR and DOR sets. It is also important to note here that the MOR and DOR refer to the accessible record occurrences, because a record occurrence which is not accessible - because of ROR - cannot be further modified or deleted in any way.

4.7.3 SECURITY COMMAND GROUP

The following commands are available to implement security in ENDBMS:

INSTALL-USER

The command can be used to install one or more users in the ENDBMS. For every user it creates a security entry in the USER LIST, where the supplied user number and password are stored. Both user number and passowrd must start with an alphabetic and can be followed by numeric or alphabetic characters; the maximum number allowed is 10.

Use: for DBA only.

Requirements:

1. DBA identification.
2. the specified user numbers must be unique.

Syntax: INSTALL-USER

(<user-number> , <user-password>)

{ (<user-number> , <user-password>) } ;

REMOVE-USER

This command is used to remove one or more installed users from the ENDBMS. It deletes the corresponding user security entries from the system's USER LIST. The command is executed only if the specified users have their access profiles removed. In other words if a user is to be removed

from the system the DBA must first remove his access profile and then use this command to delete the user's security entry. The reason we did this was to simplify the command's execution phase.

Use: for DBA only.

Requirements:

1. DBA identification.
2. valid user numbers.
3. user's access profile must be removed.

Syntax: REMOVE-USER <user-number>

{<user-number>} ;

LIST-USERS

This command can be used by the DBA to get a list of all installed users; both the user's user number and password are displayed. The generated list of installed users also indicates which of the users were suspended, at the time the command is executed, by putting an asterisk in front of the corresponding user number(s).

Use: for DBA only.

Requirements:

1. DBA identification.

Syntax: LIST-USERS ;

DEFINE-PROFILE

The command can be used by the DBA to define the access

profile of one or more already installed users; at least one profile definition must be supplied. A profile definition should include at least one record type definition and optionally set type definitions.

Use: for DBA only.

Requirements:

1. DBA identification.
2. indicated user(s) must be installed.
3. all names used must be valid schema names.
4. valid record and/or set type access code(s).
5. validity of RFR, MFR, ROR, MOR and DOR with respect to each other, on one hand, and with respect to the corresponding record type access code on the other.

Syntax: DEFINE-PROFILE <profile-definition>

{<profile-definition>} ;

<profile-defintion>:= USER <user-number>

<record-definition>

{<record-definition>}

<set-definition>

<record-definition>:= RECORD: <s-r-name>

CODE: <rec-access-code>

RFR : <f-name> <f-name>

MFR : <f-name> <f-name>

ROR: <f-name><rel-op><quot-value>

MOR: <f-name><rel-op><quot-value>

DOR: <f-name><rel-op><quot-value>

<set-definition>:= SET: <set-name>
CODE: <set-access-code>

REMOVE-PROFILE

This command is used by the DBA to remove an access profile from one or more installed users; at least one user number must be specified in the command.

Use: for DBA only.

Requirements:

1. DBA identification.
2. every indicated user must be installed, suspended and NOT active.

Syntax: REMOVE-PROFILE <user-number>

{<user-number>} ;

LIST-PROFILE

This command can be used by either the DBA to get a list of user access profile(s), corresponding to specified user number(s) in the command (the DBA must specify at least one user number), or by any other installed user to get a list of his own profile (if he has one).

Use: for DBA and any other installed user.

Requirements:

1. DBA or user identification.
2. the indicated user(s) must be installed and in case the user is not the DBA, he must provide his own user number.

Syntax for DBA: LIST-PROFILE <user-number>
{<user-number>};

for any other
installed user: LIST-PROFILE <user-number> ;

ADD-RIGHTS

This command is used to expand the defined access profile of some user by adding ONE record type or set type entry to the corresponding profile record type or set type list respectively.

Use: for DBA only.

Requirements:

1. DBA identification.
2. indicated user must be installed and have a defined access profile, which can also be empty.
3. duplicate record or set names are not allowed in an access profile.

Syntax: ADD-RIGHTS , USER: <user-number>
|<record-definition>
|<set-definition> |;

REMOVE-RIGHTS

This command is used by the DBA to update an access profile, of some specified user, by removing one profile record type or one set type definition from the corresponding access profile. It does the opposite of ADD-RIGHTS. During its execution the corresponding subschema (if any) is checked

for probable consistent update.

Use: for DBA only.

Requirements:

1. DBA identification.
2. indicated user must be installed, suspended, not active and with a profile defined.
3. validity of specified record or set name.

Syntax: REMOVE-RIGHTS , USER: <user-number>

```
| RECORD: <rec-name> |
| SET: <set-name>    | ;
```

SUSPEND-ACCESS

This command is used by the DBA to suspend one or more users for some period of time; a user's access to the ENDBMS is 'restored' later explicitly by the DBA. The command sets the suspension flag of the corresponding user(s) and if the user(s) is(are) already using the ENDBMS, he(they) is(are) not kicked out but he(they) continue his(their) interface and next time his(their) secuirity entry is accessed, a message to prepare for log-off is displayed.

Use: for DBA only

Requirements:

1. DBA identification.
2. indicated user(s) must be installed.

Syntax: SUSPEND-ACCESS: <user-number>

```
{<user-number>} ;
```

RESTORE-ACCESS

This command is used to restore one or more suspended users; it resets the suspension flag(s) of the indicated user(s).

Use: for DBA only.

Requirements:

1. DBA identification.
2. every indicated user must be installed.

Syntax: RESTORE-ACCESS: <user-number>

$$\left\{\text{<user-number>}\right\}\ ;$$

LIST-ACTIVITY

This command is used by the DBA to check what users are using the ENDBMS at any given time.

Use: for DBA only.

Requirements: DBA identification.

Syntax: LIST-ACTIVITY ;

LOCK-SYSTEM and UNLOCK-SYSTEM

These commands are used by the DBA to lock and unlock the ENDBMS. All the active users at the time the ENDBMS is locked, still remain on the system, but no one else can enter the system from that point on.

Use: for DBA only.

Requirements: DBA identification.

Syntax: LOCK-SYSTEM ; and UNLOCK-SYSTEM ;

RECOVER

This command is used by the DBA to recover one or more interrupted users.

Use: for DBA only.

Requirements:

1. DBA identification.
2. all indicated users must be installed and interrupted.

Syntax: RECOVER: <user-number> {<user-number>};

PASSWORD

This command is used by either the DBA or any other user to change their password.

Use: for DBA and any other installed user.

Requirements:

1. DBA or user identification.
2. validity of specified old password.

Syntax: PASSWORD <old-pass> <new-pass> ;

4.7.4 EXPLICIT AND IMPLICIT ACCESS RIGHTS

Let us consider a data base whose schema definition consists of two record types, the DEPARTMENT and EMPLOYEE, and one set type, the DEP-EMP. Suppose that the manager of a department is authorized to access the data base as follows: he is permitted to read and update all the fields of his own DEPARTMENT occurrence and read and modify all the

EMPLOYEE occurrences which 'belong' to his department (we could also specify which fields are restricted from read and/or modify access).

To enforce this security for the manager-user, using our implemented security scheme, we have to introduce an extra DNO - department number or name - field. This field can be used in the ROR clause of the EMPLOYEE profile record type entry (of the manager's access profile) to exclude all EMPLOYEE occurrences which do not belong to the manager's department. This leads us to a 'relational type' of record type. The difference between such a network data base and a similar relational one, is the fact that all the EMPLOYEE member occurrences of a DEPARTMENT owner occurrence can be found directly in the network data base, using the implemented 'set occurrence path' (as a pay-off to the extra DNO field!).

However it is possible to avoid the inclusion of such required fields by introducing another type of 'access rights' for any schema record type. An additional IMPLICIT access clause in a profile record type (X) definition, could specify some set types to be used as follows: occurrences of X can be accessed only as member occurrences of some owner occurrence of the specified set types. This IMPLICIT access can be associated with the implemented EXPLICIT one to enforce further restrictions at the 'type' and 'occurrence' level (i.e., with a proper combination of the explicit

record type access code and the RFR, MFR, ROR, MOR and DOR sets).

A possible expansion to the occurrence restriction specification makes the proposed EXPLICIT and IMPLICIT authorization scheme a powerful combination which could implement very complex data base access rights.

4.8 SUBSCHEMA SUBSYSTEM

The subschema subsystem consists of seven command handlers, one for each of the seven subschema commands available in ENDBMS. With these commands, every installed user with a non-empty access profile, can define (store) his view - in accordance with his profile -- and furthermore administer it. Subschema administration includes commands to (1) totally destroy a subschema definition, (2) list the stored subschema definition, (3) expand the definition by adding a new subschema record type or a new subschema set type, (4) remove from the subschema definition a subschema record or set type. In section 2.3, we explained the subschema concept supported by the ENDBMS along with its consistency with respect to changes to either schema definition or user access profile definition.

4.8.1 SUBSCHEMA IMPLEMENTATION

All subschema definitions are stored in one system file, the system file two ('SF2'). Each subschema

definition is implemented with one null head entry, which is pointed to by the 'susbchema head-field' value of the user's security in the USER LIST, and two singly linked lists as 'branch' lists of the null head entry (see Figure 4.7);

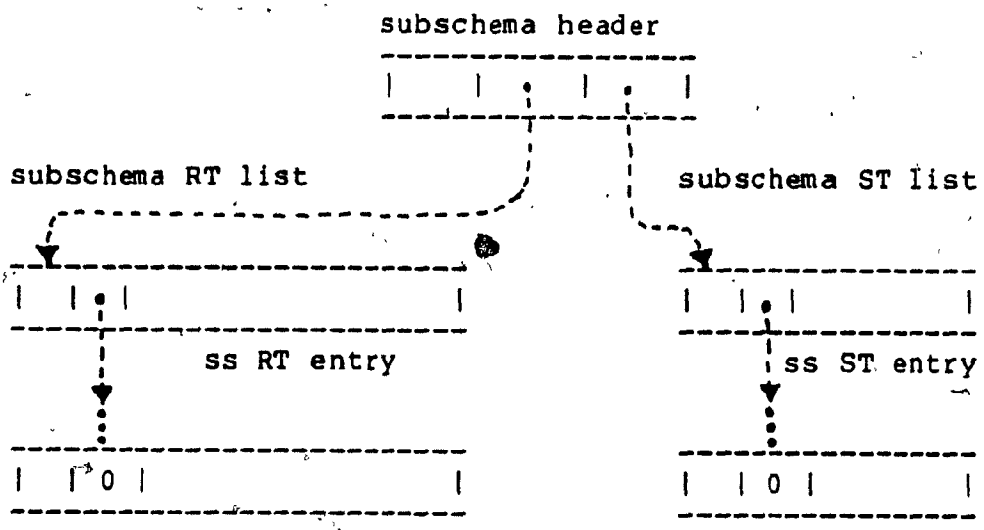


Figure 4.7 Representation of USER SUBSCHEMA.

the two lists being the subschema record type and set type list. Each entry of the subschema RT-list has one branch list - the field name list for the corresponding subschema record type. In the case where the record type entry corresponds to a virtual subschema record type, each field definition is implemented as one 'main' entry, which belongs to the field name list, and exactly one 'auxiliary' entry for the real field definition, or one or more 'auxiliary' entries for the virtual field definition; if a virtual field is constructed by using the procedures SUM-OF, AVERAGE-OF or MERGE-FROM, one 'auxiliary' entry will be required for each schema field name specified in those clauses. Every

subschema set type definition is implemented as one entry, so the entries of the subschema ST-list do not have any 'branches'.

4.8.2 STRUCTURE OF SYSTEM FILE TWO

Given below is the format description of all entries used to store a subschema definition.

FILE HEAD ENTRY

It is generated at system installation time and its format is similar to other file head entries.

KEY	KGEN	RF
1	2	3

field description:

1 : key field.

2 : key generator; it is used to generate unique keys (integers numbers) every time a new entry is written in SF2.

3 : recovery field; it is used by the recovery subsystem to store the key of every entry, before its deletion.

SUBSCHEMA NULL HEAD ENTRY

It is generated when a user's subschema definition is stored and it has the following format:

KEY	SSRTLH	SSSTLH
1	2	3

field description:

1 : key field.

2 : if it is not zero it points to the top of the subschema RT-list.

3 : if it is not zero it points to the top of the subschema ST-list.

SUBSCHEMA RECORD TYPE ENTRY

It is generated either when a subschema defintion is stored or when a new subschema record type is inserted to a stored subschema definition. Its format is the following:

KEY	LINK	r-name	b-name	BPTR	R/V	FLH	TOTL
1	2	3	4	5	6	7	8	9	10

field description:

1 : key field.

2 : link field; if it is not zero it points to the next subschema record type entry.

3,4 : subschema record type name (ss-r-name).

5,6 : the name of the schema record type, used as 'base' type in the subschema record type definition.

7 : pointer to 'base' record type in the schema RT-list.

8 : type of the subschema record type; 1 is used for REAL and 2 is used for VIRTUAL.

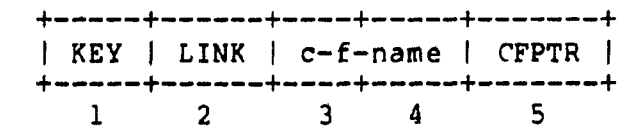
9 : it points to the top of the corresponding field name list.

10 : length of the subschema record type; it is calculated implicitly during storage of the record type definition, from all field definitions.

FIELD ENTRY FOR REAL RECORD TYPE

This type of field entry is used only for storage of a REAL subschema record type. This distinction can be justified by the formal definition of a REAL subschema record type, see section 4.8.3, where only the subschema field names are

specified; the field names of a REAL subschema record type must be in one to one correspondence with the field names of its specified 'base'. The subschema field names may or may not be the same as their corresponding schema ones.



field description:

1 : key field.

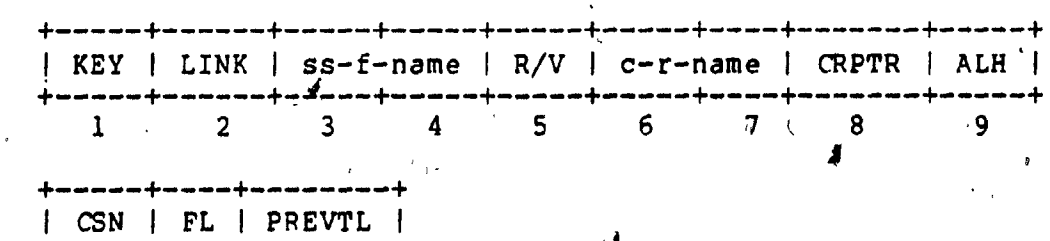
2 : link field; it points to the next field name entry, of the same field name list - if it is not zero.

3,4 : subschema field name.

5 : pointer to corresponding schema field name entry, of the specified 'base' record type.

MAIN FIELD ENTRY FOR VIRTUAL RECORD TYPE

This type of field entry is used by the field name list of a VIRTUAL subschema record type for partial storage of either a REAL or VIRTUAL field definition.



field description:

1 : key field.

2 : link field.

3,4 : subschema field name.

5 : type of the field; 1 for REAL and in case where the field is VIRTUAL -1, -2, -3, -4 are used respectively for the procedures RESULT-OF, SUM-OF, AVERAGE-OF, MERGE-FROM.

6,7 : name of the compatible schema record type, where this field is to be taken from or calculated from.

8 : pointer to schema compatible record type entry in the schema RT-list.

9 : points to first auxiliary field entry, associated with this main field entry.

10 : name of the schema set type which defines the compatibility.

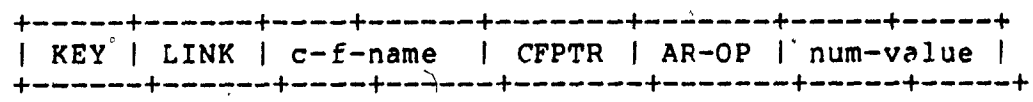
11 : length of the field, REAL or VIRTUAL.

12 : total length of all previous fields.

AUXILIARY FIELD ENTRIES

There are two kinds of auxiliary entries, TYPE-A for the procedure RESULT-OF, and TYPE-B for the procedures SUM-OF, AVERAGE-OF, MERGE-FROM.

TYPE-A



field description:

1 : key field.

2 : link field.

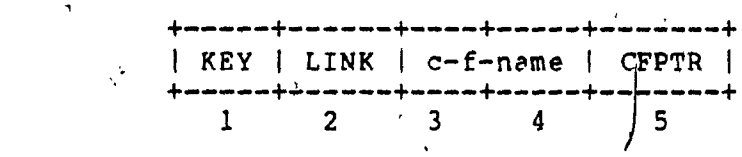
3,4 : name of the compatible schema field.

5 : pointer to corresponding compatible schema field.

6 : arithmetic operator; is stored in character form.

7,8 : two CDC words for the numeric value; it is stored as a character string of numeric digits.

TYPE-B

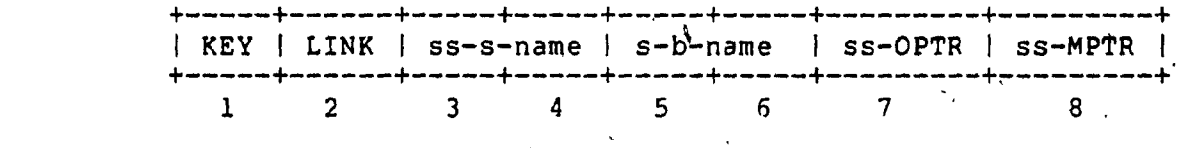


field description:

1,5 : same as in TYPE-A.

SET TYPE ENTRY

This entry is used by the subschema set type list to store the information needed for a subschema set type definition, and it is stored either when a subschema definition is stored or when a new set type definition is added to a stored subschema defintion.



field description:

1 : key field.

2 : link field.

3,4 : subschema set type name.

5,6 : name of the schema 'base' set type.

7 : pointer to the subschema owner in the subschema record type list.

8 : pointer to the subschema member in the subschema set type list.

4.8.3 SUBSCHEMA COMMAND GROUP

In this section all the subschema commands are described and their syntax is given in BNF form.

DEFINE-SUBSCHEMA

This command can be used by any installed user with a defined profile to store his subschema definition. It should include at least one subschema record type definition and optional subschema set type definitions. In a REAL record type definition the field definitions are simple, but a schema 'base' record type must be supplied. The field names must be distinct and they should be in a one-to-one correspondence with the field names of the specified schema 'base'. A REAL or VIRTUAL field definition in a VIRTUAL record type is a bit more complex. In a VIRTUAL field definition the schema record type name - and the corresponding schema field name(s) - where the subschema field is taken or calculated from, must be specified; a VIRTUAL field definition uses one of the four implemented procedure types.

Use: by any installed user with a defined profile.

Requirements:

1. user identification.
2. a defined access profile.
3. validity of all specified schema names.

Syntax: DEFINE-SUBSCHEMA <subschema-definition> ;

<subschema-definition>:= <ss-record-definition>
{<ss-record-definition>}
{<ss-set-definition> }

```
<ss-record-definition>:= RECORD: <s-r-name>
                          |REAL   |
                          |VIRTUAL| WITH-BASE <s-r-name>

                              <field-definition>
                             {<field-definition> }

                          END
```

field definition for REAL subschema RT.

```
<field-definition>:= FIELD  <ss-f-name>
```

field definition for VIRTUAL subschema RT.

```
<field-definition>:= FIELD  <ss-f-name> <field-option>

<field-option>:= |REAL FROM <comp-r-name> . <comp-f-name>|
                 | VIRTUAL  <procedure-type>              |

<procedure-type>:= |RESULT-OF  <comp-r-name> . <comp-f-name>|
                   |                          <ar-op><number>|
                   |SUM-OF     <comp-r-name> (<comp-f-list>)|
                   |AVERAGE-OF     >>             >>      >>|
                   |MERGE-FROM     >>             >>      >>|

<ar-op>:= '*' or '/'

<comp-f-list>:= <comp-f-name> {<comp-f-name>}
```

note

```
  <s-r-name>     : schema record name
  <s-f-name>     : schema field name
  <s-s-name>     : schema set name
  <ss-r-name>    : subschema record name
  <ss-f-name>    : subschema field name
  <ss-s-name>    : subschema set name
  <comp-r-name>  : compatible record name
  <comp-f-name>  : compatible field name
  <comp-f-list>  : compatible field (name) list
```

DELETE-SUBSCHEMA

This command can be used by the owner of a subschema to remove his subschema definition from system file two. It

deletes the two subschema record type and set type lists, the null head entry, and it sets to zero the subschema pointer field of the corresponding user security entry in the USER LIST.

Use: by any installed user.

Requirements:

1. user identification.
2. a defined (even empty) subschema.

Syntax: DELETE-SUBSCHEMA ;

LIST-SUBSCHEMA

This subschema command can be used by any owner of a stored subschema to get a list of its definition. This will happen either because the user wants to verify his view or he wants to find out if his view definition has been modified because of some 'change' to schema or to his access profile. (Note: deletion of a schema RT or ST entry causes consistent update of all profile definitions and their associated subschemas; deletion of a profile RT or ST entry causes consistent update to its associated subschema.)

Use: by any installed user.

Requirements:

1. user identification.
2. a defined subschema.

Syntax: LIST-SUBSCHEMA ;

INSERT-SSRT

This command is used by the owner of a stored subschema to 'expand' its definition by adding a new subschema record type (SSRT). The new subschema record type name must be unique, e.g., not already used in the same subschema, and the overall record type construction must be consistent with the user's access profile. It enters a new entry in the subschema record type list.

Use: by any installed.

Requirements:

1. user identification.
2. a defined (even empty) subschema.
3. unique ss-r-name and consistent use of all specified schema names.

Syntax: INSERT-SSRT <ss-record-definition> ;

INSERT-SSST

This subschema command is similar to INSERT-SSRT, but it is used instead to add a new subschema set type definition, in a stored subschema. The specified subschema record types as owner and member types, should already have been defined.

Use: by any installed user.

Requirements:

1. user identification.
2. a defined (not empty) subschema.
3. valid owner and member subschema names and consistent schema set 'base' name.

Syntax: INSERT-SSST <ss-set-definition> ;

Note

the <ss-record-definition> of the INSERT-SSRT and the <ss-set-definition> of the INSERT-SSST are the same as those of the DEFINE-SUBSCHEMA.

DELETE-SSRT

This subschema command reverses the INSERT-SSRT; it can be used by the owner of a subschema to remove a record type definition from the subschema RT-list. If the specified subschema record type is owner or member to any subschema set type, the command deletes also these subschema set types.

Use: by any installed user.

Requirements:

1. user identification.
2. a defined (not empty) subschema.
3. validity of the supplied name.

Syntax: DELETE-SSRT <ss-r-name> ;

DELETE-SSST

This subschema command does the opposite of the INSERT-SSST, namely removes one subschema set type definition from the corresponding subschema ST-list.

Use: by any installed user.

Requirements:

1. user identification.
2. a defined (not empty) subschema.
3. validity of the supplied name.

Syntax: DELETE-SSST <ss-s-name> ;

4.9 DML SUBSYSTEM

The DML subsystem consists of 15 commnd handlers and it implements the 15 DML commands supported by the ENDBMS. The DML includes: record type commands to retrieve and update record type files, set type commands to retrieve and update information based on the implementation of the set type relationship (e.g., following set type paths), commands for simple control loops using conditional and unconditional jumps, commands for currency indicator manipulation and two commands to prepare and terminate the processing of any record type file. Before we look at the DML commands we explain the implementation of a record type file and the implementation of a set type association at the RECORD OCCURRENCE LEVEL; the implementation of a set type association at the RECORD TYPE LEVEL was explained in Chapter 2, schema section.

4.9.1 RECORD TYPE FILE

It is important to note at this point that the term 'record type file' refers to both schema and subschema record types. A schema record type file consists of a collection of occurrences of its type and because everything defined in the schema is real, a schema record type file is stored in the data base, where each record occurrence is stored exactly the way it is defined in the schema. All the

stored schema record type files divide the data base into logical divisions (the so called data base logical units or simply logical units). Let us now consider the subschema record type file; it again consists of a collection of record occurrences of a subschema record type file, however, a subschema record type file can be REAL or VIRTUAL. A REAL subschema record type matches a schema record type which constitutes its schema BASE, while a VIRTUAL subschema record type is defined OVER one or more schema record types with one being its schema BASE. Thus a virtual subschema record occurrence is not stored the way it is defined but is constructed from the 'base-occurrence' and additionally from some 'compatible' record type occurrences. Therefore it is clear that a subschema record type file is associated with one or more stored schema record type files and hence to one or more logical units. A logical unit is implemented as a doubly linked list, as shown in Figure 4.8,

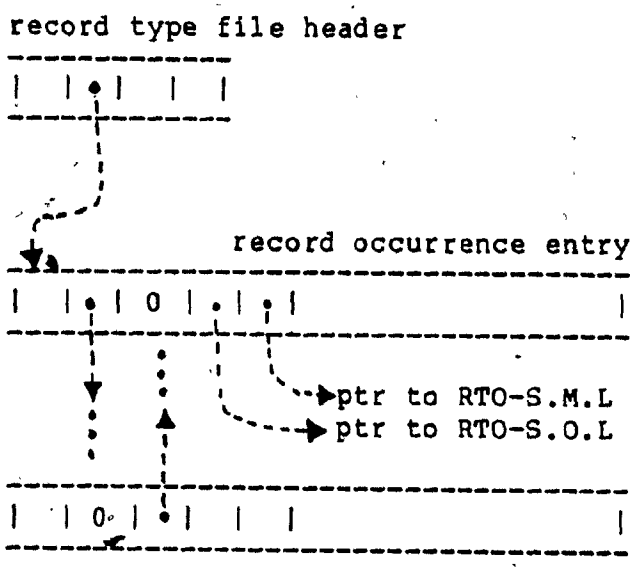
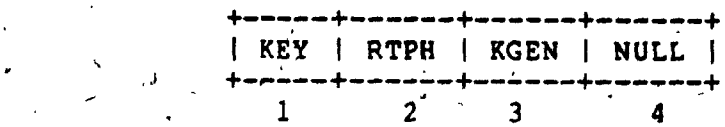


Figure 4.8 Representation of RT-file.

with a uniquely identified head entry. The head entry is generated when the schema record type is defined and its key is the record's record type number. Insertions in the doubly linked list always take place at the top. Each entry of the record type file corresponds to a record occurrence of this type and consists of two parts where the first contains the key, forward and backward pointers (to implement the record type paths), and two pointers to RTO-S.O.L. and RTO-S.M.L. respectively (we recall that S.O.L. and S.M.L. are the set Ownership and Membership lists at the Record Type Occurrence (RTO) level); the second part contains the 'actual' information of the record occurrence. The total length of each entry in the current implementation can be up to 250 characters or 25 CDC words. The length of the entry depends on the defined length of the record type in the schema. Figure 4.8 shows that the implementation of the record type file uses two kind of entries: the head and main entry; given below is the format of these two entries.

HEAD ENTRY FORMAT



field description:

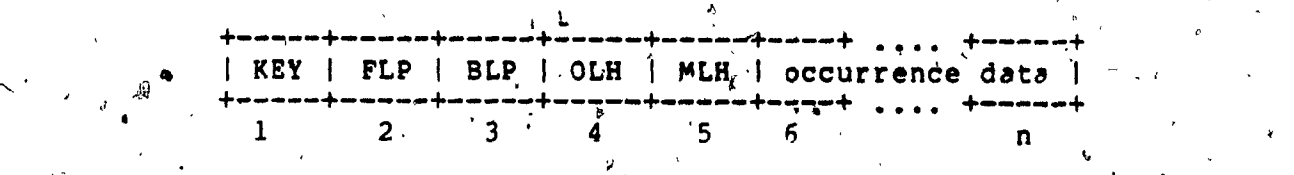
1 : key field; it is equal to the rcord type number of the corresponding schema record type and because the RTNUM is unique every such entry is uniquely identified in the data base.

2 : if it is not zero it points to the first main entry of the associated logical unit.

3 : key generator field; its initial value is set to: (FNUMBER * 100 + RTNUM) * 10**6 where FNUMBER is an integer from 1 to 10 and stands for the index of the physical data base unit which is allocated to the corresponding logical unit, and RTNUM stands for the record type number of the corresponding schema record type. This combination as a key of a main entry of the corresponding logical unit not only identifies the entry itself in the data base but a proper decomposition identifies the logical and physical data base unit as well (see conclusion Chapter).

4 : not used.

MAIN ENTRY FORMAT



field description:

1 : key of the main entry; it identifies the main entry in the data base, the logical and physical data base unit where it is stored.

2 : forward pointer; zero indicates the last entry.

3 : backward pointer; zero indicates the top entry.

4 : points to the head entry of the RTO-S.O.L.

5 : points to the head entry of the RTO-S.M.L.

6,n : this part is used to store the information of a record occurrence. The number of words of this part depends on the length of the corresponding schema record type.

4.9.2 SET TYPE IMPLEMENTATION AT RECORD TYPE OCCURRENCE LEVEL

In Chapter 2 we talked about associations between record types as they are defined in the schema by means of set type definitions, and how these are implemented using the RT-S.O.L. and RT-S.M.L. At the occurrence level we are interested in the representation of associations between record occurrences; such associations are based upon the already defined concept of set occurrence. The following six points give the justification for this implementation:

(1) a record occurrence of record type X which is an owner type of a number of set types can be an owner occurrence in different set occurrences within these set types; we allow it to be the owner of at most one set occurrence within a given set type.

(2) a record occurrence of record type Y which is a member type of a number of set types, can be a member occurrence in different set occurrences within these set types; we allow the record occurrence to be a member of only one set occurrence of a given set type.

(3) a set occurrence can have zero or more member occurrences.

(4) the ability to answer the question: 'find all members of

some owner in a given set occurrence'.

(5) the ability to answer the question: 'find the owner of some member in a given set occurrence'.

(6) the need to have the record type file implemented separately from the set type implementation at the occurrence level and to implement the associations between record occurrences by using existing stored occurrences and not introducing any data redundancy.

The ENDBMS implements the set type association at the record occurrence level by means of three lists: the Record Type Occurrence Set Ownership List (RTO-S.O.L.), the Record Type Occurrence Set Membership List (RTO-S.M.L.), and the Set Occurrence Membership List (SOM-list). Two system files are used to store these lists; system file three (SF3) for the first two lists and system file four (SF4) for the third one.

4.9.2.1 RTO-S.O.L.

This is stored as a singly linked list with a null head entry pointed to by a reserved field of the record occurrence entry. The head entry is created when the record occurrence is created. Each main entry corresponds to a set type where the type of the current record occurrence is owner and one field of this entry points to the head entry of the corresponding SOM-list as it is shown in Figure 4.9. A main entry of the RTO-S.O.L. - along with the head entry

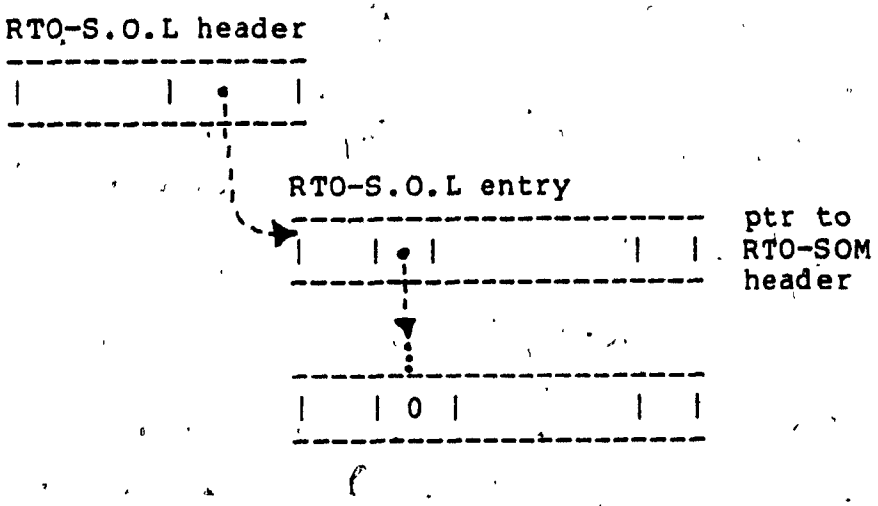


Figure 4.9 Representation of RTO-S.O.L

of its corresponding SOM-list - is generated when the 'owner' record occurrence is created (in this case its record type is declared as owner of some set types in the schema), or during the definition of a new set type in the schema which includes the record type of the current record occrrence, as owner. The RTO-S.O.L. of any record occurrence of type X must indicate at any time all the set types where the X is owner.

DATA STRUCTURES FOR RTO-S.O.L.

Head entry:

KEY	SOLH
1	2

field description:

1 : key field.

2 : link field; if it is not zero, it points to the first set ownership node of the RTO-S.O.L.

Main entry:

KEY	LINK	s-name	SOMH
1	2	3	4

field description:

1 : key field.

2 : link field; if it is not zero, it points to the next entry of the RTO-S.O.L.

3 : name of the set type.

4 : pointer to the head entry of the corresponding SOM-list.

4.9.2.2 RTO-S.M.L.

This list is stored as a singly linked list as shown in Figure 4.10,

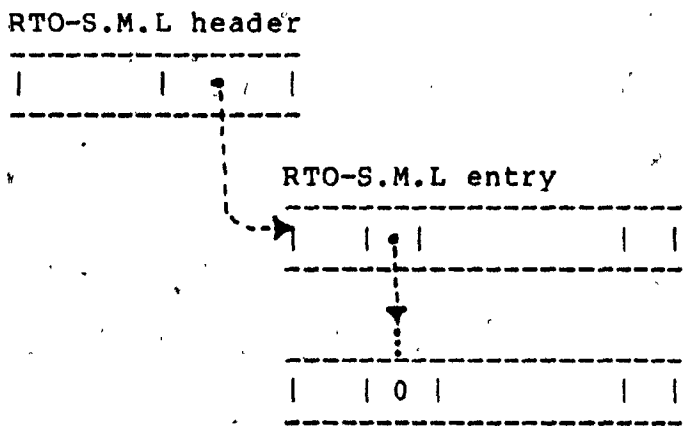


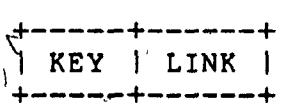
Figure 4.10 Representation of RTO-S.M.L

with a null head entry pointed to by a reserved field of the corresponding record occurrence. The head entry is created when the record occurrence is created. Each main entry corresponds to a set type where the record type of the current record occurrence is member type. A main entry is

generated either when the record occurrence is stored or when a new set type definition is declared in the schema and the record type of the current record occurrence happens to be its member type. At all times the RTO-S.M.L. of a record occurrence of a record type X must indicate all the set types where the X is their member type.

DATA STRUCTURES FOR RTO-S.M.L.

Head entry:

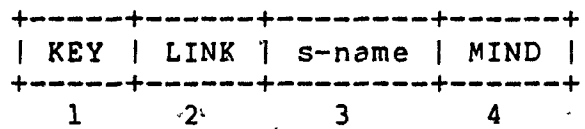


field description:

1 : key field.

2 : link field;if it is not zero it points to the first set membership node of the RTO-S.M.L.

Main entry:



field description:

1 : key field.

2 : link field; if it is not zero, it points to the next set membership node of the RTO-S.M.L.

3 : set type name.

4 : if the record occurrence - where the

RTO-S.M.L. belongs to - is a member occurrence of a set occurrence, of the set type indicated by the third field, then the value of this field points to its owner record occurrence within the owner record type file, otherwise is set to zero, i.e., the current record occurrence it is not a member of any set occurrence within the set type indicated by the third field.

4.9.2.3 SOM-LIST

This list is stored as a doubly linked list, as is shown in Figure 4.11.

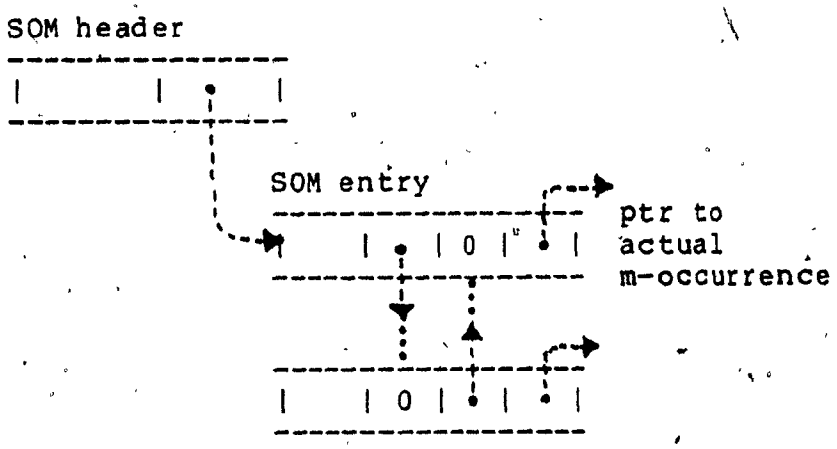


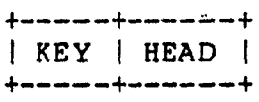
Figure 4.11 Representation of SOM-list

It has two types of entries: the head entry and the main entry. Let us call: (1) S the set type indicated in the main entry of the RTO-S.O.L. from where the head entry of this SOM-list is pointed to, (2) X the corresponding record occurrence and R1 its record type, and (3) R2 the member record type of S. A main entry of the SOM-list is created

when an R2 occurrence becomes a member of the set occurrence whose owner is X; the same main entry is deleted from this SOM-list when the R2 occurrence is removed from the set occurrence - in S - with X as the owner occurrence. By creating and destroying SOM main entries without touching the involved owner and member occurrences, we implement the idependence of the set occurrence association at the record occurrence level.

DATA STRUCTURES FOR SOM-LIST

Head entry:

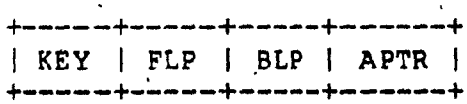


field description:

1 : key field.

2 : link field; if it is not zero it points to the top main entry of the SOM-list.

Main entry:



field description:

1 : key field.

2 : forward link field; zero for last main entry.

3 : backward link field; zero for top main entry.

4 : pointer to actual member occurrence.

4.9.3 RECORD AND SET TYPE LOCAL TABLES

Every ENDBMS user has to specify the logical unit (record type file) of the data base that he wants to access; furthermore before the access attempt is made the logical unit must be opened appropriately (i.e., either with R-mode or D-mode). After completion of the access, the user has to explicitly close the corresponding unit - see the RELEASE DML command in section 4.9.4. In Chapter 2 we talked about the 'currency indicators' and we indicated that there are three such indicators in ENDBMS: (1) CI: the current of a record type (current of 'base' record type, for a subschema record type), (2) OCI: current owner of a set type (current of the owner type of the 'base' set type, for a subschema set type), and (3) MCI: the current member of a set type (current member refers to member type of the 'base' set type, in the case of a subschema set type). The first two are pointers to actual record occurrences, while the third is a pointer to a main entry of the SOM-list, which corresponds to the current set occurrence. This SOM main entry holds the pointer to the actual member occurrence.

The ENDBMS keeps track of the logical units and currency indicators via two local tables; these tables are set up for every user during the execution of the 'USER' command. The first table is the record type table; each entry correspond to a record type definition (schema or

subschema) and it has the following structure:

rec-name	CI	PM	PTR	ss-RPTR
1	2	3	4	5

field description:

1 : record type name (schema or subschema).

2 : currency indicator for the record type file corresponding to schema record type file or to the 'base' record type file of a subschema record type file. By default the system makes 'current' the first occurrence of the associated logical unit; this can be changed explicitly by either the GET or SET DML commands.

3 : processing mode; 1 for Read, -1 for Delete and 0 otherwise.

4 : points to corresponding entry of the schema or subschema record type list.

5 : it is used only in the case of a subschema RT name and it points to its schema 'base' entry, in the schema RT-list.

The second table is the set type (schema or subschema) table; each entry of this table corresponds to a schema or subschema set type definition, and it has the following structure:

set-name	OCI	MCI	ORTI	MRTI	PTR
1	2	3	4	5	6

field description:

1 : name of the schema/subschema set type.

2 : Ownership Currency Indicator for the set type indicated in field 1; it is set explicitly by DML commands and every time a current owner is selected (i.e., a set occurrence) the MCI field is set to the first main entry of the corresponding SOM-list of the

current set occurrence (i.e., implicitly to first member occurrence).

3 : Membership Currency Indicator; it is set by default (see description of field 2) and explicitly by DML commands.

4 : points to an entry of the record type table, which corresponds to the owner type.

5 : points to an entry of the record type table, which corresponds to the member type.

6 : points to the corresponding set type entry of the schema or subschema ST-list.

4.9.4 DML COMMAND GROUP

In this section we describe, in detail, each of the DML commands and we give the format and execution requirements for each of them. The given BNF format does not specify the optional 'label', which can be put at the begining of any DML command. A label is a positive integer number and must be unique within an input session. The label used in an IF or GOTO command must refer to a forward or backward label, but within the same input session as the IF or GOTO command.

GET

In order to retrieve a record occurrence from the data base, it has to be located first within its logical unit and one of the three currency indicators has to be set to point to it. This operation is called GET. We do not enforce data dependent security for any GET format without the WHERE-clause since no actual data is made available to the

user by the GET command. All GET formats set the currency indicators (except the GET-NEXT-WHERE which some times does not, see below) unless there are no occurrences at all, or there is no next occurrence within the selected 'path', in which case the currency indicators do not change. The GET-NEXT-WHERE format searches to find the next AVAILABLE ocurrence which satisfies the WHERE-clause and then sets the corresponding currency indicator. If no such occurrence is found the corresponding currency indicator does not change.

Use: by any installed user.

Requirements:

1. user or DBA identification.
2. for record type formats, the specified record type file must be opened, at least with R-mode for processing.
3. for the set type formats, both the owner and member record type files must be opened with at least R-mode for processing.
4. each specified record or set type name must match one of the names stored in the user's local record or set type tables respectively.

Syntax:

RECORD TYPE FORMATS

1. GET FIRST OF <rec-name> ;
2. GET NEXT OF <rec-name> ;
3. GET NEXT OF <rec-name>
   WHERE <field-name> <rel-op> "value" ;

the <rel-op> can be one of: = , > , < , <> only.

SET TYPE FORMATS

1. GET FIRST OWNER IN <set-name> ;
2. GET FIRST MEMBER IN <set-name> ;
3. GET NEXT OWNER IN <set-name> ;
4. GET NEXT MEMBER IN <set-name> ;
5. GET OWNER OF CURRENT OF <rec-name> IN <set-name> ;

The name which follows the reserved word OF, is the name of the referenced schema or subschema record type, and the name which follows the reserved word IN, is the referenced schema or subschema set type. All GET-MEMBER formats refer to currently selected set occurrence, within the specified set type.

LIST

The LIST command is used to retrieve (if it is available, i.e., no ROR restrictions) the current occurrence of a record type file, or to get a list of all retrievable occurrences - with optional WHERE and/or SORT clause - of a record type file or the SOM-list of the current set occurrence. The listing, within the indicated record or set type path, starts from the occurrence which is pointed to by either the CI or MCI indicators. We did not implement the LIST CURRENT OWNER and LIST CURRENT MEMBER commands, but they can be easily implemented by adding two more production rules in the DML LL(1) grammar and modifying appropriately

the LIST command handler. However it is possible to retrieve the current owner or current member by making it current of its record type using the SET DML command, and then use the LIST current format for this record type. The first two LIST formats are record type commands and the third one is a set type command.

Use: by DBA and any installed user.

Requirements:

1. user or DBA identification.
2. the user must have a defined subschema.
3. the LIST CURRENT OF and LIST ALL OF format require the specified record type file to be opened properly.
4. the LIST ALL MEMBERS format requires owner and member record type files of the specified set type, to be opened properly (at least with R-mode).
5. the field name specified in the WHERE and/or SORT clause must be a REAL one if it belongs to a subschema record type.

Syntax:

1. LIST CURRENT OF <rec-name> ;
2. LIST ALL OF <rec-name>
   [<where-clause>]
   [<sort-clause>]
   ;
3. LIST ALL MEMBERS IN <set-name>
   [<where-clause>]
   [<sort-clause>]
   ;

<where-clause>: WHERE <field-name> <rel-op> "value"

<sort-clause>: SORT IN | ASCENDING | USING <s-f-name>
| DESCENDING |

<rel-op>:= = or < or > or <>

IF

The IF DML command is available in three formats which allow conditional jumps; three specific conditions are tested: (a) if the current of a record type file is last; (b) if the current of a record type file has a specified field value which satisfies a relationship with a supplied test value; and (c) if the current member within a set occurrence is last. If the tested condition is true then the command that will be executed next is the one indicated by the specified label, otherwise the command following the IF command will be executed. The second type of IF may be used to construct a complex 'search' criterion. The first two types are record type commands and the third type is a set type command.

Use: by DBA and any installed user.

Requirements:

1. user or DBA identification.
2. a user must have a defined subschema.
3. validity of all specified names.
4. all involved record type files must be opened properly (at least with R-mode).
5. the label must an integer defined in the same input session, either forward or backward.

Syntax:

1. IF CURRENT OF <rec-name> LAST
   GOTO <label> ;

2. IF CURRENT OF <rec-name>
   HAS <field-name> <rel op> "value"
   GOTO <label> ;

3. IF CURRENT MEMBER IN <set-name> LAST
   GOTO <label> ;

GOTO

This command is used to change the sequence of command execution unconditionally.

Use: by DBA and any installed user.

Requirements:

1. user or DBA identification.
2. valid label.

Syntax: GOTO <label> ;

SET

The SET DML is available in four formats. Let us assume a set type S with owner type O and member M to explain the 'currency' SET formats. These can be used to set the CI of O to OCI of S, the CI of M to MCI of S or to set the OCI of S to CI of O. Independent settings of the CI of O or M do not affect the OCI or MCI of S. The fourth SET format is used to define the length of the 'output line' which by default is equal to 80. A user or the DBA can change this default length in the case where a schema or subschema record occurrence is bigger than 80 characters.

The maximum output length is 136. The same command can be used many times in one or more input sessions.

Use: by DBA and any installed user.

Requirements:

1. user or DBA identification.
2. valid and consistent record and/or set names.
3. all involved record type files must be opened before, with at least R-mode.
4. valid length for 'output line'.

Syntax:

1. SET CI OF <rec-name>
   TO OCI IN <set-name> ;
2. SET CI OF <rec-name>
   TO MCI IN <set-name> ;
3. SET OCI IN <set-name>
   TO CI OF <rec-name> ;
4. SET LINELIMIT TO <integer> ;

LOCK and RELEASE

These two DML commands are designed to prepare a record type file for processing and terminate the processing by explicit user or DBA control. If the user is the DBA, one data base logical unit, for every schema record type, is opened or released; with other user things are different. In order to open one subschema record type file, it might be required to open more than one data base logical units, because in the case where the subschema record type is defined over more than one schema record types, all

corresponding data base logical units must be opened. (Note: a VIRTUAL subschema record type file will never be opened with D-mode.) As we explain in logical concurrency later in section 4.11.2, opening a logical unit means creation of a 'user stamp' in the data base logical unit monitor file (SF8), and termination of a logical unit's processing means deletion of this user stamp. In both LOCK and RELEASE commands at least one schema or subschema record type name must be supplied.

Use: by DBA and any other installed user.

Requirements:

1. user identification.
2. a non-DBA user must have a defined subschema.
3. validity of supplied record type names.
4. the specified record type files must be closed for the LOCK, and opened for the RELEASE.

Syntax:

LOCK FOR [READ | DELETE] <rec-name> {<rec-name>};

RELEASE <rec-name> {<rec-name>};

INSERT

The INSERT DML command is a record type command and is used to store new occurrences in a specified data base logical unit. (Note: a non-DBA user can insert only in a REAL subschema record type and with appropriate record type

access code.) A new occurrence must be quoted and all its fields (as many declared in the schema) must be separated with '/'. The supplied field values must match their declared type, and their length must be equal or less than their declared ones.

Use: by DBA and any non-DBA installed user.

Requirements:

1. DBA or non-DBA user identification.
2. the specified record type file must be opened with at least R-mode.

Syntax:

INSERT IN <rec-name>

"record-occurrence"

{"record-occurrence"} ;

DELETE

The DELETE DML command is a record type command and is used to delete the 'current' of some logical unit, which must be locked, i.e., opened, with D-mode. Because the D-mode forbids even reading from this logical unit, it should be released as soon as possible. Deletion of a record occurrence causes automatic removal from all set occurrences that it belongs to and destruction of both RTO-S.O.L. and RTO-S.M.L.

Use: by DBA and any non-DBA installed user.

Requirements:

1. user identification.

2. validity of the supplied name.

3. the specified record type file must be opened with D-mode.

Syntax:

```
DELETE CURRENT OF <rec-name> ;
```

REPLACE

The REPLACE DML command is a record type command and is used to change, i.e., replace, a field value (one at a time) of the 'current' occurrence of a logical unit. The type of the new field value must match its declared type, and its length must not exceed its declared length. The logical unit must be opened with at least R-mode - it does not have to be with D-mode.

Use: by DBA and any non-DBA installed user.

Requirements:

1. user identification.

2. validity of supplied field and record names.

3. consistent new field value in quotes.

4. the specified logical unit must be opened with at least R-mode.

Syntax:

```
REPLACE <field-name> IN CURRENT OF <rec-name>
WITH "value" ;
```

INCLUDE and REMOVE

These two DML commands are set type commnds and are used to perform the set type update operations include and

remove. Each of them reverses the other. The INCLUDE includes the 'current' occurrence of some record type B, which is a member type in some set type X, into the 'current' set occurrence of X and at the same time becomes the 'current' member within X. The REMOVE command, removes the 'current' member of the current set occurrence of some set type. Both of them update the corresponding SOM-list of the owner occurrence and RTO-S.M.L. of the member occurrence. These two operations require the member type of the specified set type, to be opened with D-mode.

Use: by DBA and any non-DBA installed user.

Requirements:

1. user identification.
2. validity of supplied names, e.g., the specified record type name must be the member type name of the specified set type.
3. member must be opened with D-mode.
4. owner must be opened with at least R-mode.

Syntax:

```
INCLUDE CURRENT OF <rec-name>
IN  <set-name> ;

REMOVE CURRENT MEMBER FROM <set-name> ;
```

4.9.5 LIMITATIONS OF ENDBMS DML

In this section we will attempt to expore the capacity of the ENDBMS DML sublanguage. For this we consider two logical associations of the real world; one 1:M and one M:M. The 1:M association being the association of departments and

employees of an enterprise and the M:M being the association between parts and suppliers of the same enterprise. The creation of a data base in which we will store information about those entities and the associations between them, will require a schema definition with five record types: DEPT, EMPL, PARTS, SUPPLIERS and PRIQUA, as well as three set types: DEP-EMP (owner DEPT and member EMPL), PART-PRIQUA (owner PARTS and member PRIQUA) and SUP-PRIQUA (owner SUPPLIERS and member PRI-QUA). The record type PRIQUA is used to implement the M:M parts suppliers association, and it would have two fields PRICE and QUANTITY, to indicate the price and quantity of a part supplied by a particular supplier. The DML sublanguage is capable of answering any question as far the department employees association is concerned but this is not the case for the parts suppliers association. In fact questions which will require UNION or INTERSECTION of the 'information set' cannot be answered. We define the notion of 'information set' as follows: it is a set of occurrences (of the same type) which is generated by accessing a single set occurrence and probably visiting other single set occurrences within other set types. To make all this clear let us consider Figure 4.14 which shows a data base instance of the SUPPLIERS - PARTS association. Figure 4.14 indicates that:

supplier S1 supplies parts P1 , P3

supplier S2 supplies parts P1 , P3

supplier S3 supplies parts P2

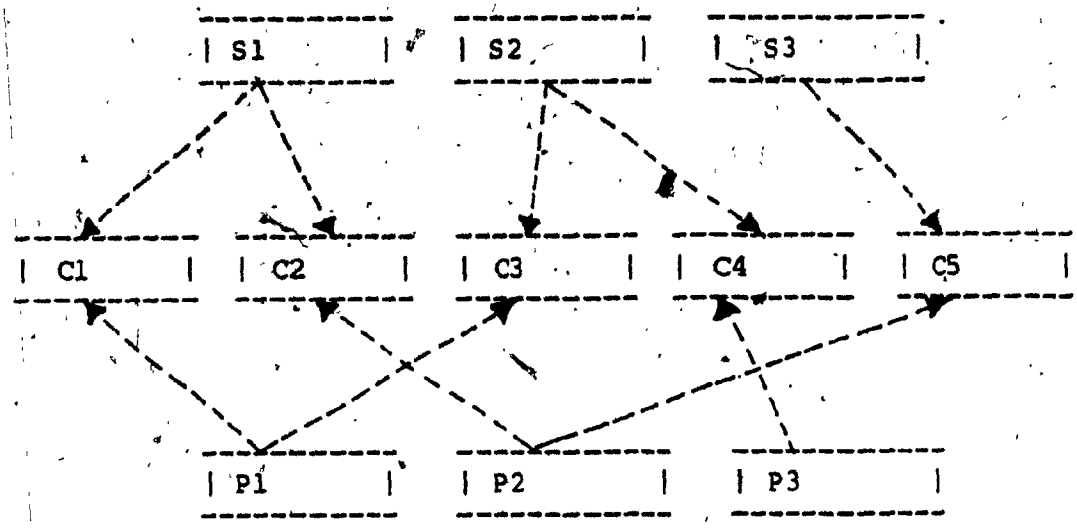


Figure 4.12 An instance of suppliers-parts association.

QUESTION 1: requires UNION

"list all the parts supplied by suppliers S1 and S3". This query cannot be answered at once, because it requires the following: find all the parts supplied by supplier S1 and save them internally (in a local file A), then find all the parts supplied by supplier S3 and save them in a local file B. The answer to above query can be found by performing a UNION operation on files A and B; the answer can be stored on a separate file C, or on one of A or B.

QUESTION 2: requires INTRSECTION

"list all common parts supplied by S1 and S2" (or another same type question could be: "list all suppliers who supply parts P1 and P3). This query will require the generation of the information set A and B separately; information set A being the parts supplied by S1, and information set B being

the parts supplied by S2. An INTERSECTION operation on these two information sets, will give the answer to question 2 and which can be stored in a local file. Again question types as question 1 and 2 cannot be answered by the ENDBMS DML sublanguage as implemented currently. Of course a UNION or INTERSECTION type question may include more than two information sets. However it is possible to expand the current DML by introducing: (1) two more local files in the user's work space, (2) a SAVE command which will create the information sets, (3) two operators UNION and INTRSECT which will always operate on two information sets A and B and leaving the result, let us say, always to A; moreover the result set A can be operated with a new information set kept again in B, and so on.

4.10 THE LOG-IN and LOG-OFF COMMANDS

A user can log-in and log-off using the USER and OUT commands respectively. These two commands do not belong to any subsystem and therefore are implemented as separate ENDBMS command handlers.

USER

This command implements the security at the Identification and Authentication Level; the supplied user number identifies the user as either DBA or any non-DBA valid/installed user, and the supplied user password authenticates the user, the specified processing mode tells

the system in which way it is going to be accessed. Both user number and password must start with a letter which can be followed by a maximum of 9 letters or digits, including the character '-'. They should match either the DBA's user number and password, or the ones of some other installed user. Proper log-in prepares the ENDBMS for further authorized access. During the execution of the USER command the ENDBMS performs the following actions:

for DBA

1. checks the activity field in his security entry in the USER LIST; if it set to 1 then DBA recovery is called, see section 4.12.6.
2. sets up the RECORD and SET TYPE LOCAL TABLES as they were described in section 4.9.3.

for non-DBA user

1. checks the system recovery field (section 4.12.5).
2. checks if the ENDBMS is locked.
3. checks if the user is suspended.
4. checks the user's activity field; if it is set to 1 the user is logged-off immediately, otherwise the system sets up his RECORD and SET TYPE LOCAL TABLES and the ENDBMS becomes available for authorized access.

Successful execution of the USER command causes setting of a global flag which is checked as a requirement for any other ENDBMS command except the OUT. A successful execution is indicated by displaying the message READY.

Use: by any user who whishes to use the ENDBMS.

Requirements:

1. the user must have NOS permissions for all NOS files used by ENDBMS.

2. valid user number, password and processing mode.

Syntax:

USER <user-number> <user-password> <proc-mode> ;

<proc-mode>: B or I

OUT

This command causes the normal termination of the system's interface. Its execution consists of the following operations: (1) the ENDBMS checks the two record and set type local tables to find out if any record type file has been left inadvertedly unlocked and releases it, (2) sets to zero the activity field for the user logging-off and (3) displays the log-off time.

Syntax:

OUT ; or OUT$

4.11 CONCURRENCY

This section deals with the implementation of the concurrency mechanism, which is supported by the ENDBMS. The mechanism is implemented as two concurrency levels, with one - the logical level - build up on top of the other - the physical level. The logical concurrency level handles the accessability of the data base logical units, which requires accessability of the corresponding data base physical unit. This is handled by the physical concurrency level. (Note:

the physical concurrency level also handles the accessability of the system physical units.)

4.11.1 PHYSICAL CONCURRENCY

The attachment of any of the 19 NOS files used by the ENDBMS, in RM-mode does not cause any 'sharing' problem. However there are some ENDBMS commands (see section 4.12.2), which will require more than one of these ENDBMS physical units attached in M-mode. As mentioned in section 3.6., the M-attachment ~~of system~~ files 2 to 6 is controlled by the NOS Operating System. To avoid 'deadlock' situations we introduce a 'dispatcher' file: the system file nine (SF9), which will be used by the ENDBMS to monitor the 'activity' of all system files from 1 to 6. This file consists of a single entry with 7 fields; the first 6 for the system physical units 1 to 6 and the the last for the entry's key. If the execution phase of a command requires some physical units, the dispatcher file must be attached with M-mode and then the activity status of all required units must be inspected. If all of them are free, i.e., their corresponding field values are zero, then they are reserved for the command - which needs them - by marking them as busy, i.e., their corresponding field values are set to user's - who is executing the command - user number and finally the dispatcher file is released. The reserved system physical units are closed, released and then attached with M-mode. The ENDBMS uses similarly the dispatcher file

to free the units after the command is executed. This process includes: releasing of M-attached units and attaching them with RM-mode, then reseting their corresponding field values in the dispatcher file entry, and finally releasing the dispatcher file itself. This kind of simulation uses the specially designed COMPASS routines to achieve the RM and M attachments. In addition every time we switch from one mode to another, the files must be closed and reopened again with the proper mode. These operations, imposed by the host Operating System, cause a great programming overhead and add a considerable delay in the system's 'response time'. No attempt has been made to evaluate the delay time involved; the number of active users and the number of concurrent commands requiring extensive use of the files, will affect degradation of the response time.

4.11.2 LOGICAL CONCURRENCY

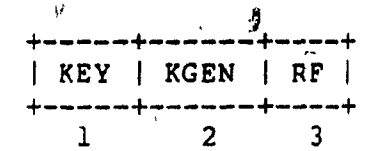
The logical concurrency is build on top of the ENDBMS physical concurrency and it is controlled exclusively by the ENDBMS; it refers to data base logical units, i.e., stored schema record type files. We mentioned in section 4.9.1 that every schema or subschema record type file must be opened properly in order to be processed. We have seen two types of locks, the R and D lock, i.e., the R and D opening modes. However it is possible that an R-lock on a subschema record type file, might require R-locks on more than one

data base logical units (recall that a D-lock on a subschema record type file will always be applied to a REAL one). The run-copy of ENDBMS, for every user, keeps track of what schema/subschema record type files are R or D-locked by using the RECORD TYPE LOCAL TABLE, see section 4.9.3. In order to prevent deadlock situations at the data base logical unit level, we employed a locking mechanism with 'shared' read locks and 'exclusive' delete locks; the locked unit being a data base logical unit itself. Suppose that a user executes a command which has to delete a record occurrence from some data base logical unit, and some other users are allowed to read the same unit at the same time; it is possible that one command C gets the key of the occurrence R, then the user U deletes R and then command C attempts to access the occurrence R. This situation would cause unpredictable problems and for this reason deletion of some record occurrence will require D-lock on its record type file, which forbids concurrent reading from the corresponding logical unit.

The implementation of logical concurrency is achieved by using the system file eight, which will also be called: logical unit monitor file. This file has three types of entries, the file head entry, the logical unit entry and the read stamp entry. Given below is a description of the structure of each of them.

FILE HEAD ENTRY

It is generated at system installation time and it is similar to other file head entries; its structure is:



field description:

1 : key field.

2 : key generator field; it generates unique keys for any new entry written on SF8.

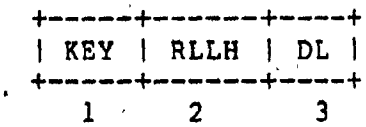
3 : recovery field.

LOGICAL UNIT ENTRY

This entry is associated with a schema record type and it is generated at the time the record type definition is stored. Every schema record type has its own logical unit entry, in the logical unit monitor file, uniquely identified by its record type number. In fact the key of a such entry is calculated from the corresponding record type number as follows:

```
key = RTNUM * 10**9
```

the structure of the logical unit entry is:



field description:

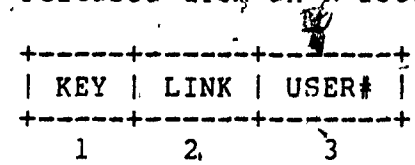
1 : key field.

2 : read lock field; if it is not zero it points to the top entry of the read stamp list.

3 : delete lock; it can be zero or one.

READ STAMP ENTRY

This entry is used by the read stamp list of every logical unit entry. The read stamp list is created to indicate all the users concurrently reading from the corresponding logical unit. The ENDBMS creates or deletes such an entry when the corresponding logical unit is locked-with or released-from an R-lock respectively. Its structure is:



field description:

1 : key field; its value is generated by the KGEN field of the file's file head entry.

2 : link field; if it is not zero, it points to the next entry within its read stamp list.

3 : user number field; every time a user wants to place an R-lock on a logical unit (and he can do so) a read stamp entry is created and his unique user number is stored on this field, so that the system can determine to whom such an entry belongs. Insertions on the read stamp list take place at the top of the list.

RULE: a placement of a D-lock requires both values of the read and delete lock field of the corresponding logical unit entry, to be zero, while a placement of an R-lock requires the value of the delete lock field to be zero.

4.12 RECOVERY

We have seen that every ENDBMS command requires for its execution, some of the ENDBMS physical units to be attached in either RM or M mode. The smallest addressable unit of information which can be accessed at a time by the CRM, from a physical unit, is a record occurrence. There are four kind of operations performed on a record occurrence: the Read, Write, Replace and Delete corresponding to CRM functions GET, PUT, REPLC and DLTE respectively. There is a small probability that any of these operations may be interrupted, but in this ENDBMS version we do not consider this case and we assume all these four CRM functions as ATOMIC operations, i.e., they are either completed or not.

4.12.1 RECOVERY-NEEDED AND RECOVERY-FREE COMMANDS.

The entire ENDBMS set of commands is divided into two groups: the Read group and the Update group. The Read group includes all those commands whose execution requires only the use of the GET CRM function and we call them Read Type commands; the Update group includes those commands whose execution requires, in addition to GET, the use of the PUT and/or REPLC and/or DLTE CRM functions. The Update group is farther subdivided into: Insert Type commands using only the GET, PUT and REPLC, Replace Type commands using only the GET,

and REPLC, and the Delete Type commands using only the GET, REPLC and DLTE CRM functions. Now, based upon the four types of commands, we divide the ENDBMS command set into two categories: the RECOVERY-FREE and RECOVERY-NEEDED category. The recovery-free category includes the Read and Replace Type commands. Interruption of these commands does not cause any problem as far as the consistency of the system or data base files is concerned; they are not assigned to any recovery code and the STATUS field of the user's security entry in the USER LIST is not set for this category, and they can be repeated safely later. The remaining two types belong to the recovery-needed category. Each command of this category has a unique recovery code, which is stored in the STATUS field of the user's security entry, at the begining of the execution phase of the corresponding command.

4.12.2 EXECUTION OF RECOVERY-NEEDED COMMANDS.

The following steps must be performed in the given order, during the execution phase of every recovery-needed command, and this sequence is used to retrace execution of an intrrupted command.

step 1. attach all necessary physical units (files) with M-mode, using the dispatcher file, and mark them as 'busy' by setting their corresponding fields in the dispatcher file entry, to user's user number.

(Note: not all M-attachments require the dispatcher file.)

step 2. get or find the start or root pointer(s) needed to trace the interrupted insert or delete process of an Insert or Delete Type command. (Note: the root pointer is provided implicitly by the command itself and is determined during the translation phase of the command; for example the name supplied by the delete a subschema record type command, corresponds to an entry in the associated subschema RT-list and its key is the root pointer which will be saved for recovery.)

step 3. set the STATUS and recovery fields RF1 and RF2 (see section 4.12.5) of the corresponding user security entry.

step 4. start and complete the Insert or Delete process of the Insert or Delete Type command.

step 5. if the command is a multistep command, then repeat steps 2, 3, 4 and 5, otherwise continue. A multistep command has the format: <com-key-word>, A, A,...,A; where the A's represent identical steps with same insert or delete process. In these cases insertion or deletion is done step by step and recovery is applied to the interrupted step and not to the previously completed ones. Every step has

its own start pointer for recovery.

step 6. mark free all files used by the command, release them from M-mode properly and reattach them with RM-mode.

step 7. Reset STATUS and recovery fields RF1 and RF2.

It is important to note that all files are marked busy or free (steps 1 and 6) by atomic operations; steps 3 and 7 are performed also with atomic operations. Our recovery scheme is concerned with interrupts between any two steps and within any step. The ENDBMS initiates 'command recovery action', see section 4.12.6, only if the user's STATUS field is not zero, and for that reason an interrupt upto step 3 is handled without calling command recovery. Every command-recovery routine, first checks if the insert/delete process (step 4) was started or not; if it did then starts an UNDO or CONTINUE recovery process is started followed by the execution of steps 6 and 7, otherwise steps 6 and 7 are performed. The UNDO recovery process must remove all results of the partially executed insert process and bring the ENDBMS back to the status which it had before the interrupted command had started. The CONTINUE recovery process refers to an interrupted delete process of a Delete Type command, and its goal is to complete the execution phase of the interrupted command. Before we describe the recovery scheme for Insert and Delete Type commands we

explain how an entry is inserted (written) or deleted to/from an ENDBMS file.

4.12.3 INSERTIONS AND DELETIONS IN ENDBMS FILES

Every Insert Type command creates (generates) an 'insert tree'. The insert tree can be: (1) a single entry with or without branches, (2) a singly or doubly linked list of single entries with or without branches. A branch itself can be a single entry or a linked list of single entries with or without branches. If an Insert Type command is to enter (add) a new entry in a linked list, the new entry is always entered at the top of the list. The ENDBMS uses two insertion rules:

for single list

1. get the key of the new entry by incrementing the KGEN field of the head entry of the corresponding file.
2. write the new entry.
3. link it to its 'parent' entry.
4. create all required branches.

for double list

1. get the key for the new entry.
2. write the new entry.
3. update the forward pointer of the previous entry (update PREV).
4. update the backward pointer of the next entry (update NEXT).
5. create all required branches.

Every Delete Type command deletes a 'delete tree', which can be: (1) a single entry with or without branches, (2) a singly or doubly linked list of single entries with or without branches. If a single entry has to be deleted all of its branches, if any, are deleted first and then the entry itself is deleted. The ENDBMS uses two rules to delete a single entry with no branches:

for single list

1. save the key of the under deletion entry in the recovery field of the concerned file's head entry.
2. remove the entry from its own list by updating the previous entry.
3. delete the entry.

for double list

1. save the key (as before).
2. update backward pointer of next entry.
3. update forward pointer of previous entry.
4. delete the entry.

Extreme cases, such as insertion in an empty list, or deletion of the top or bottom entry of a list, are taken into consideration, by simply not executing all the steps of the above rules.

4.12.4 RECOVERY FROM INSERT/DELETE TYPE INTERRUPT

Step 4, see section 4.12.2, of the execution phase of every recovery needed command, i.e., the insert/delete process, has a logical start and end; a logical action which

indicates the start/end of the insert/delete process and is known for every Insert or Delete Type command. Let us consider three ENDBMS commands to clarify the logical start/end concept.

EXAMPLE 1: consider the security command ADD-RIGHTS which adds a new profile record type definition in some user's access profile. The logical start of the insert process of this command would be the writing of the profile record type entry and the logical end would be the linking of this entry in its own list.

EXAMPLE 2: consider the security command REMOVE-RIGHTS which removes an entry, i.e., a profile record type definition, from some profile's record type list. The logical start of the delete process of this command would be the removal of the DOR branch entry corresponding to under removal record type entry, and the logical end would be the removal of the record type entry itself from its own list.

EXAMPLE 3: consider the schema command INSERT-SRT which inserts a new record type defintion in the schema definition. The logical start of the insert process of this command would be the writing of the schema record type entry and the logical end would be the linking of this entry into the schema's RT-list (other steps of the insert process would be the creation of the corresponding field name list and the insertion of the corresponding logical unit entry in the logical unit monitor file SF8).

When a recovery routine is called to recover from an Insert or Delete Type interrupted command, we know definitely that step 7 of its execution phase was never completed. There is a possibility to continue the interrupted execution phase if the interrupt took place after completion of the logical end action. Therefore what every recovery command handler, has to do as very first thing, is to test for the logical end action; if it was performed we complete step 6 (if it is required) and finally step 7, otherwise the ENDBMS is virtually unable to locate the exact interrupt point within the insert/delete process, but it is an indication that all M-attached files - for some user U1 - were left marked busy and they can be reattached with M-mode with no problem, when recovery is called for user U1. Suppose another user U2 was interrupted and he left busy another set of files, which will be mutually exclusive with the set of user U1. If another user U3 attempts to execute a command which requires some files from the U1's set and some from the U2's set he will never get those files unless the U1 and U2 are recovered. Because the exact interrupt point, mentioned above, cannot be determined, all the recovery routines must be able to handle this fact, and act as if the interrupt happened just at the begining of the insert/delete process of the corresponding execution phase. All recovery routines employ most of the subroutines used by a normal (not recovery) ENDBMS run, in order to reduce the size and complexity of the recovery

subsystem. Each of these subroutines perform specific simple tasks - i.e., delete a single entry with or without branches - and they are designed so that they do not cause any problem in the case they are called to perform their subtask, which has already been performed. Imagine for example a subroutine which deletes the RTO-S.M.L. of some record occurrence, after all necessary precautions have been taken, and assume that the RTO-S.M.L. was deleted during the normal run and that a recovery routine calls upon execution of this subroutine. We know that the head entry of the RTO-S.M.L. is pointed to from some reserved field of the corresponding record occurrence, and previous deletion of the RTO-S.M.L. would have made that field zero, so we can test for this zero before the subtask is started. Generally all these delete subroutines check first if an entry is a member of a list and then delete it. Both UNDO and CONTINUE processes of every recovery needed command, make use of the same delete subroutines.

If an interrupt took place during an insert process, one of the following happened:

1. a key for a new entry was generated.
2. the new entry was written but never linked.
3. the new entry was written but never linked completely in the case of a double list (the partial linking means that the new entry can be found in the forward path of the double list).
4. the new entry was written and properly linked.

In any case the value of the KGEN field of the physical unit

concerned stands for the entry which was written last or was to be written. Some times it might be necessary to increase the KGEN fields of 'some' files, before the insert process starts, if it is not possible to determine whether or not at least one entry was written on an ENDBMS file (during the insert process of some command's execution phase).

If the interrupt took place during a delete process (or even during an UNDO or CONTINUE process) we have to look for the following possibilities:

1. the key of an under deletion entry, with no branches, was only saved in the RF field of the corresponding file's head entry.
2. the under deletion entry was unlinked entirely but never deleted.
3. the under deletion entry was partially unlinked from a doubly linked list (the partial unlinking means that the entry still belongs to the forward path of the doubly linked list).
4. the entry was unlinked and deleted.

It is obvious that the problem causing cases, are the ones where an entry was written but never linked at all, or an entry was completely unlinked but never deleted. These kind of entries do not belong to any path and therefore cannot be detected by the delete subroutines, but they can be found because their keys are saved in the KGEN or RF fields of the corresponding file(s). In order to takle these two cases the ENDBMS does this: (1) each UNDO process saves the values of KGEN and RF fields of the involved M-attached files at the begining, and then takes all the UNDO actions (deleting

the partially generated insert tree), (2) each CONTINUE process saves at the begining the values of the RF field(s) of the involved M-attached files and then takes all the CONTINUE actions to delete entirely the delete tree. At the end of the UNDO or CONTINUE process, the saved keys are used to delete the corresponding entries; the DLTE CRM function does not cause any problem in the case where they were never written.

4.12.5 RECOVERY FIELDS

The security entry in the USER LIST, for every user, has four fields: ACTIVITY, STATUS, RF1, RF2 used by the recovery subsystem. In addition, we have seen another security field in the head entry of system file five, the System Recovery Field (SRF). Given below is a description of each recovery field.

ACTIVITY FIELD

This field is set to 1 after a user logs-in properly. A proper log-off resets thid field to zero. If at log-in time the ACTIVITY field is found to be 1, the system allows the DBA to procede and forbids any other user (see details in DBA and NON-DBA recovery in the next sections).

STATUS FIELD

This field is used to store the recovery code of any recovery needed command, see section 4.12.1. At the begining of the execution phase of every command requiring

recovery, its recovery code is stored in the STATUS field of the user's security entry, and after the execution phase is completed the STATUS field is set to zero. In this way the ENDBMS knows what every active user is doing or what any interrupted user was doing when the interrupt occurred.

RF1 and RF2 FIELDS

One or both of these fields are set at the same time as the STATUS field and they determine the logical start for the UNDO, and the logical end for the CONTINUE recovery processes of an Insert or Delete Type command respectively. The command handler routines in the source code indicate which routines make use of both recovery fields. The root pointer of the insert or delete tree is saved in RF1 field; however, some times an additional pointer is required for recovery, and this is stored in the RF2 field (e.g., in INCLUDE or REMOVE DML commands both pointers of the owner and member occurrence are saved).

SRF FIELD

This field is not used by the current version of ENDBMS, but it can be used as follows: in the case of a 'crash' of the host computer all active users will be interrupted; it is possible to instruct the host O.S to set this field when the system comes back, and once this is set only the DBA can log-in and activate the recovery routines in order to recover all interrupted users. Further modifications to the host O.S to allow automatic recovery

may also be introduced.

4.12.6 DBA RECOVERY

When the ENDBMS executes the USER command and it finds the ACTIVITY field of the DBA security entry in the USER LIST set to 1, then: (1) it closes all the data base logical units that the DBA may have left open, in order to establish consistency at the logical level (see section 4.11.2), (2) it looks at the DBA STATUS field and if it is not zero then the command recovery handler indicated by the value of the STATUS field is called; if it is zero the dispatcher file entry is checked to see if there are any files held busy and frees them. If the DBA logs-in after a system hardware failure, which left a number of users interrupted, he should procede as follows: (a) the ENDBMS must be locked, (b) the security command LIST-ACTIVITY must be executed to find out what users were recorded as active, (c) make sure that these users are not actually using the system, and (d) use the security command RECOVER, to recover all the interrupted users.

4.12.7 USER RECOVERY

When a non-DBA user executes the USER command and the ENDBMS finds that the value of the ACTIVITY field in his security entry is set to 1, it stops the log-in process and sends a message saying that the DBA should be informed so

that explicit call for recovery could take place. Thus it is impossible for the same user to access the ENDBMS from more than one terminal (or by more than one card deck) at the same time. When the DBA requests recovery for a specific user the ENDBMS repeats the two steps of the DBA recovery, and additionally clears-off the ACTIVITY and STATUS fields of the user concerned.

CHAPTER 5

SAMPLE DATA BASE APPLICATION ON ENDBMS

5.1 INTRODUCTION

In this Chapter we explain how a sample data base for a hypothetical company C, is implemented on ENDBMS, and in Appendix D we provide actual interface sessions with ENDBMS, which show how features of ENDBMS can be used to define, retrieve and update the data base of company C in a secure environment. First we describe the information environment of company C; second we design a Network data model which reflects the logical organization of the comapany's data base; third we describe the specific functions of some of the company's employees as users of its data base, and finaly we show the 'original' data stored in the data base by the DBA before they become 'accessible'.

5.2 DESCRIPTION OF THE HYPOTHETICAL COMPANY C

Our company C consists of a number of departments and is administered by a chief executive officer. Every department has a unique name, a number of employees and its own manager who is responsible for its operation. An employee can belong to one department only. The company keeps information about its employees but in addition is interested in keeping track of the medical history of its employees. Each department is functioning in groups and a

group has a unique name, its own group leader and a number of employees taken from the department it belongs to. An employee can be assigned to only one group of his/her department.

The company needs for its operation a number of parts which may be supplied by different suppliers. Since different suppliers may supply a part at a different prices and lead times - a period of time a supplier needs to supply a part after he has received an order for it - the company wants to know who supplies what, at what price and at what lead time. If a part is required then its available prices and lead times are considered and one or many orders are issued to its (selected) supplier. The company must be always in a position to check the total quantity, which has been ordered, for any part; namely all the issued orders must be available.

Finally the parts are used by the company's various groups and it is possible that different groups use the same parts. The company wants to record the total quantity of a part, received by a group, at any time for inspection purposes. For example if part P1 is used by groups G1 and G3 then the total quantity of P1 received by G1 and G3 plus the quantity in stock, must be equal to the total quantity of P1 bought by the company. This inspection obviously requires all the past orders of P1.

5.3 A NETWORK DATA MODEL FOR COMPANY C

Figure 5.1 shows the various record types and set types required by the data base of the above described company C. Figure 5.2 and 5.3 show not only the names of the record and set types but the field names of each record type. In order to keep a level of abstraction in our company we are going to use unique symbolic names instead of real names for entities such as departments, employees, groups, parts and supplier; the symbolic names will start with the first letter of the corresponding entity, following by an integer, e.g., E5 and S4 for some employee and supplier respectively. It can be seen that natural N:M associations such as groups-parts and parts-suppliers are not shown directly but they are implied by existing set types over these record types and some carefully introduced 'connector' record types.

Let us now make some observations about some record and set types shown in Figure 5.2 and Figure 5.3. First of all we see some 'redundant' fields such as DNO and GNO in EMPLOYEES record type and DNO in GROUPS; the inclusion of these fields is necessary in order to impose Read Occurrence Restrictions over these record types for some users. We mentioned in section 4.7.4 the use of 'implicit' access rights as an alternative to avoid this redundancy. However this feature is not implemented in this ENDBMS version. The RPARTS (received) and GPARTS (given) indicate what parts are

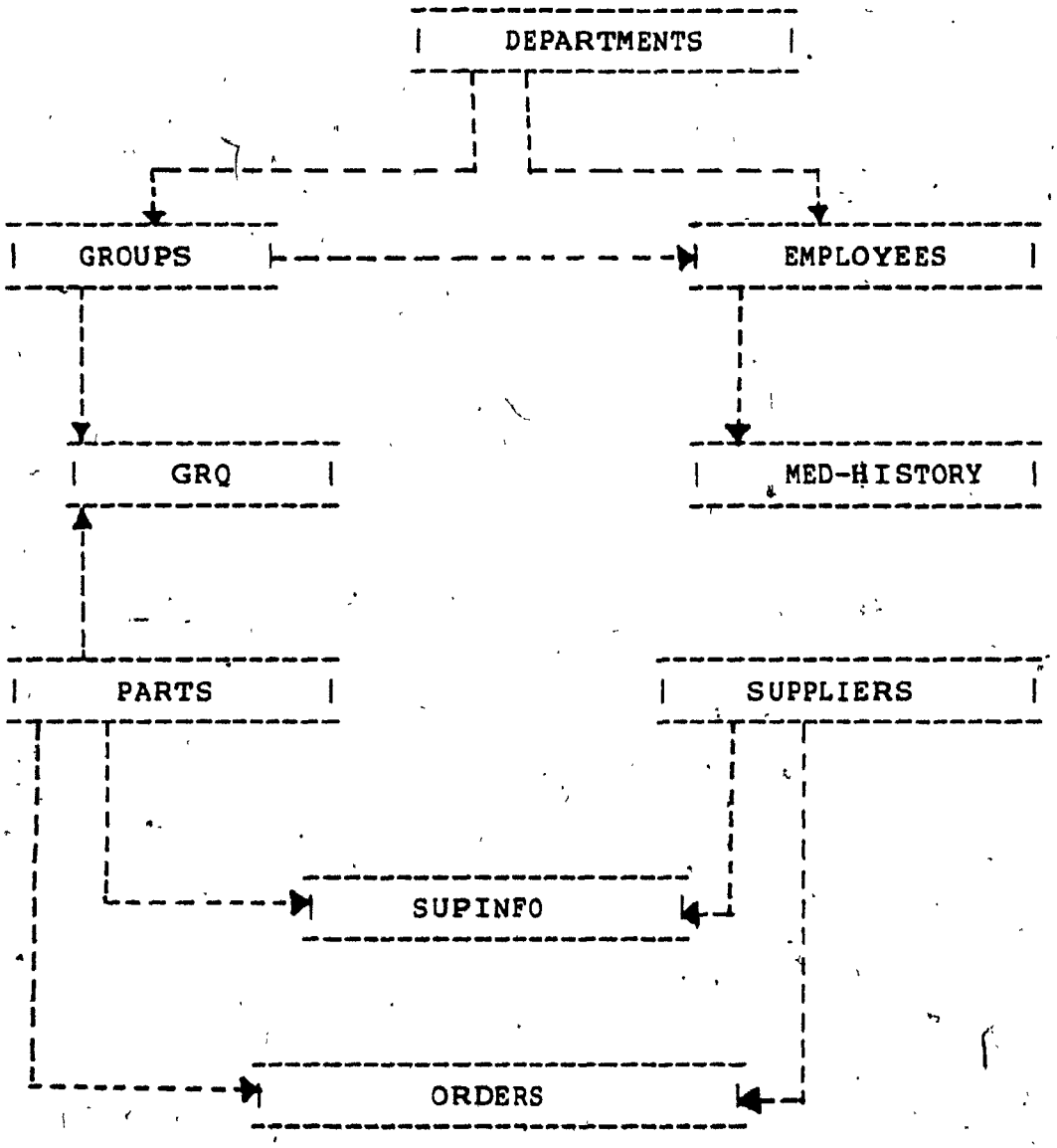


Figure 5.1 A Network model for company C.

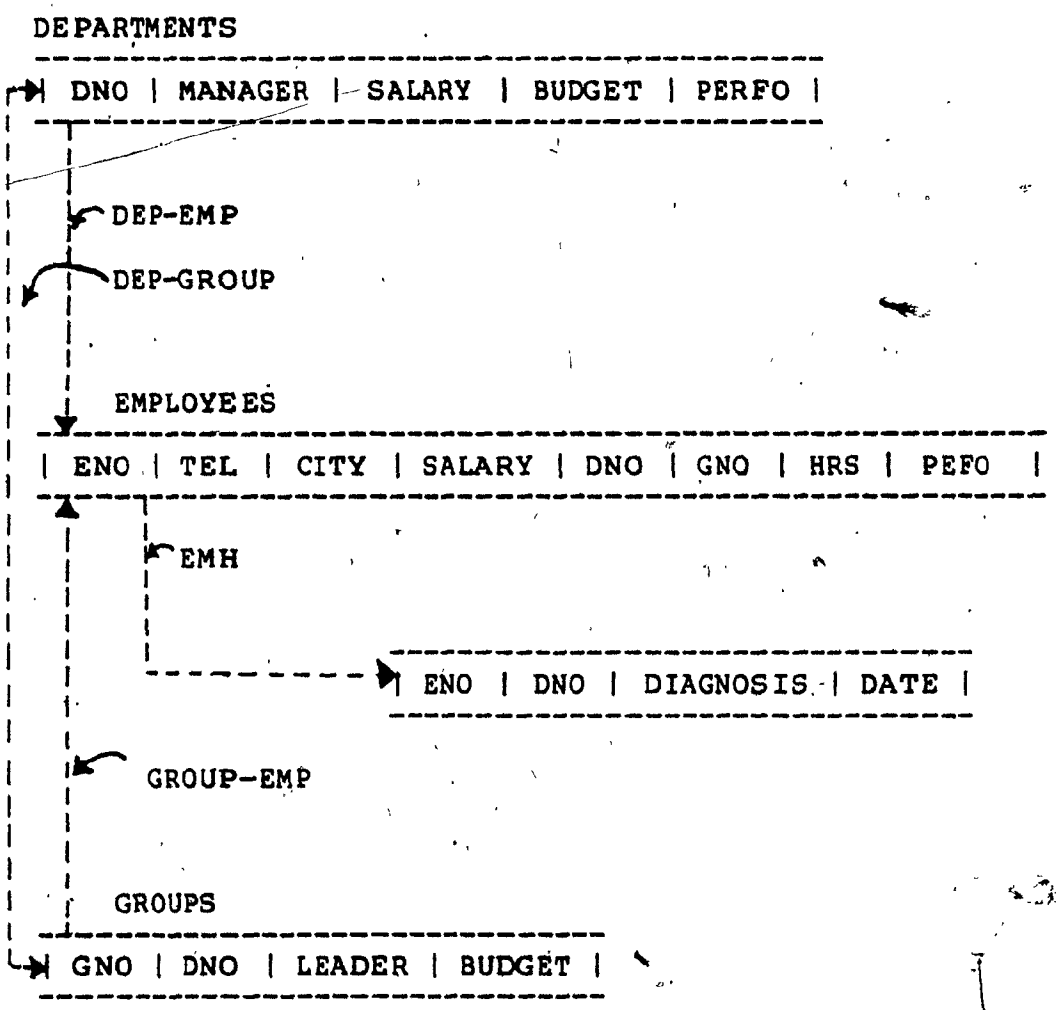


Figure 5.2 Detailed representation of Figure 5.1

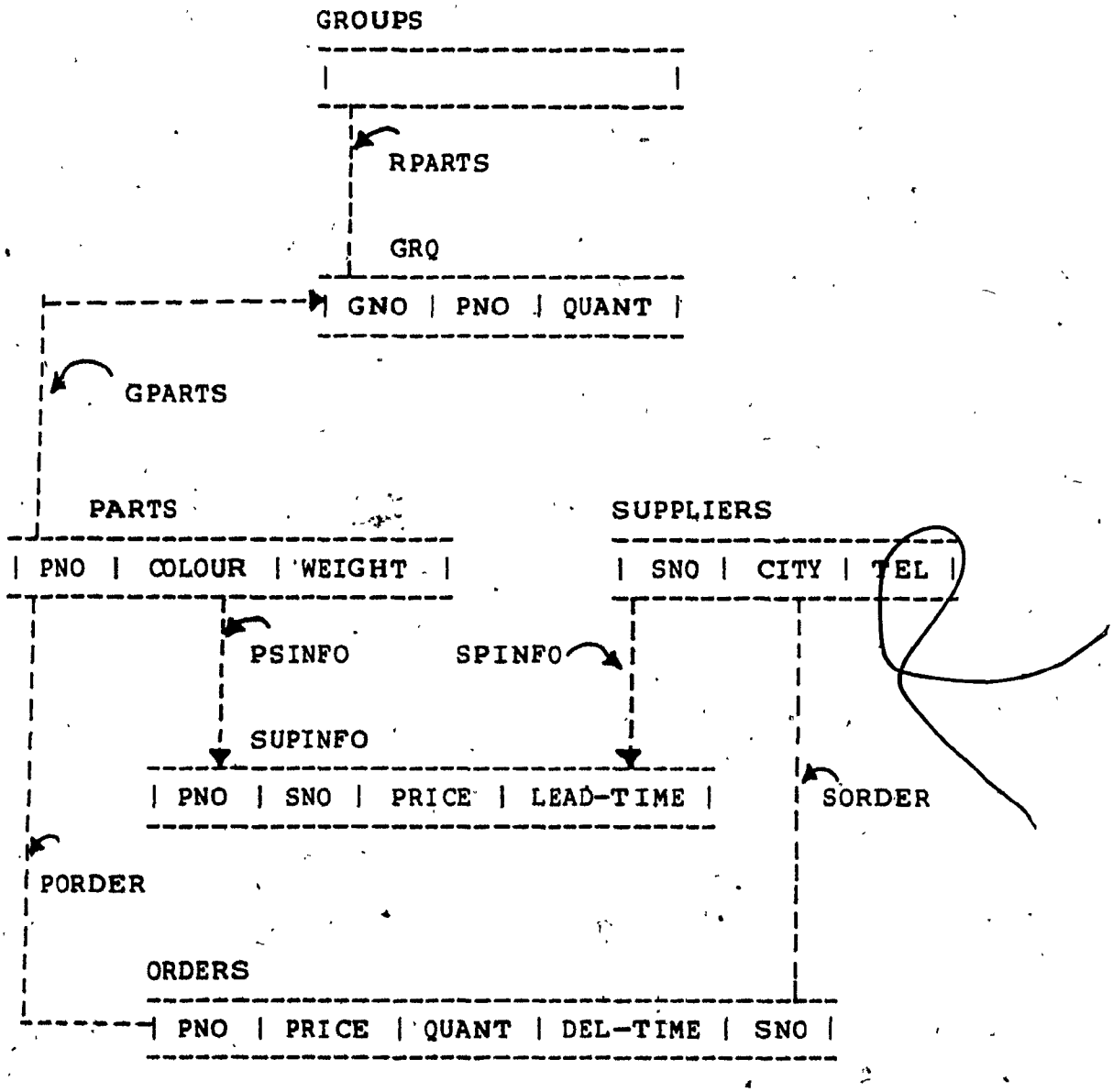


Figure 5.3 Detailed representation of Figure 5.1.

used by each group and what groups receive a particular part. The record type GRQ is the connector type required for the set type representation of the groups-parts association. The fact that part P3 is used by group G2 is 'stored' by creating a GRQ occurrence which will be a member of G2 within RPARTS and member of P3 within GPARTS. This 'membership' can be 'removed' when the group G2 no longer needs the part P3, but no user can delete GRQ occurrences. The GNO and PNO fields help to 'identify' the stored quantity the case a part is not used any more by a particular group. The record type SUPINFO and set types PSINFO and SPINFO are intended to be used for answering questions related to "supplies" and "supplied-by" associations between parts and suppliers. Similarly the record type ORDERS and the set types PORDER and SORDER will be used for answering queries such as what and how many orders have been issued for a particular part or what orders have been issued to a particular supplier. Authorized users will create ORDERS occurrences and include (remove) them in (from) set occurrences of PORDER or SORDER set types, but they will be deleted by the DBA only, for the 'inspection' reasons we mentioned before. Finally the EMH set type will be used to find the 'medical history records' of some employee. The ENO field of the MED-HISTORY record type is used to enforce ocuurrence restrictions.

5.4 TYPES OF USERS WITHIN COMPANY C

In this section we describe the role and responsibilities of some of the company's employees as far its data base is concerned. The DBA will assign a unique user-number and user password to each of these employees, and based upon the descrption of the role of each prospective user he will assign an access profile to each of them. The definition of the access profile will provide the ENDBMS the ability to determine if its users access the data base in the way they are supposed to.

A. DBA

According to the description of ENDBMS the DBA has no restrictions at all, as far as the system or data base access is concerned. He is responsible for the overall administration of the company's data base.

B. CHIEF EXECUTIVE OFFICER

The company's administrator will have any access rights, with no restrictions, to all department information, i.e., read, insert, replace and delete with no field restrictions (FR) or occurrence restrictions (OR), over the DEPARTMENT record type. In addition he can access in read only mode (with

no FR or OR) the rest of the company's data base.

C. DEPARTMENT MANAGER

A manager of a department will have the right to access his department entry in the DEARTMENT record type file in read only mode except its PERFO field; the value of this field is a code used by the chief officer to classify the perfomance of all department managers. He can insert new EMPLOYEE occurrences in the data base (e.g., hiring new employees in his department) but he can read, modify or delete EMMPLOYEE occurrences corresponding to his department, with no FR or OR restrictions. A manager can read only all of his employees medical information and he can read, modify or delete the GROUPS occurrences belonging to his department and he can insert a new GROUPS occurrence when he forms a new group in his department. Finally a manager will have read, include and remove access rights over the set types DEP-EMP, DEP-GROUP and GROUP-EMP, namely he will be responsible for hiring (firing) employees for (from) his department, for the group organization of his department and for the assignment of his employees into his department groups.

D. DEPARTMENT SECRETARY

The secretary of a department will have access to all EMPLOYEES occurrences, corresponding to her department, but she won't be able to see their salary and performance fields or change their ENO, DNO, GNO and HRS fields. She can also read only her DEPARTMENT occurrence except the manager's salary and performance, and she will have read only access over the set type DEP-EMP.

E. GROUP LEADER

The leader of a group will have read access with no FR over his GROUPS occurrence and he will be able to see all the EMPLOYEES occurrences corresponding to the employees of his group except their SALARY and PERFO fields; the group leader will have the right to modify the HRS field of his employees, the right to read only with no FR or OR, the PARTS occurrences and read with no FR or OR, and insert GRQ occurrences. Finally a leader of a group will have read, include and remove access rights over the set types RPARTS and GPARTS, and read only over the set type GROUP-EMP. He will be responsible for keeping up to date the association between his group and the parts it uses.

F. PURCHASING DIRECTOR

He will have the following access rights: (1) read with no FR or OR and insert over the PARTS record type, (2) read, insert and modify with MFR=SNO (i.e., the supplier number cannot be changed) over the SUPPLIERS record type (3) no restrictions at all over the SUPINFO record type (4) read with no FR or OR and insert over ORDERS record type; he cannot modify or delete an ORDERS occurrence (5) read with no FR or OR, insert and modify with MFR=(GNO, PNO) over the GRQ record type (6) read only with RFR=BUDGET over GROUPS record type (7) read, include and remove over the set types PORDER, PSINFO, SPINFO and SORDER (8) read only over the set types RPARTS and GPARTS.

G. MEDICAL PERSONNEL DIRECTOR

He will be responsible for storing in the data base any new medical information about any employee but after that, he will not be able to change or delete stored information. This will require access to EMPLOYEES and MED-HISTORY record type and EMH set type, which will be: read only with RFR=(SALARY, DNO, GNO, HRS, PERFO) over EMPLOYEES, read with no FR or OR and insert over MED-HISTORY and read and

include over the EMH.

5.5 SAMPLE DATA BASE

In this section we use six tables to show the sample data stored in the company's data base by the DBA at its 'creation time'; the tables do not show all the fields of the corresponding record occurrences but they indicate their logical associations. Table 5.1 shows the grouping of the company's employees in its three departments. A name with two stars indicate the manager of the corresponding department and a name with one star indicates the secretary of the department. Table 5.2 shows the group organization of two departments and the emlployees assigned to each of them. A name with a star indicates the group leader of the corresponding group. Table 5.3 shows the company's suppliers and the parts supplied each of them (price and lead time not shown). Table 5.4 shows some orders issued to some suppliers. Table 5.5 shows the kind of parts required by every group an Table 5.6 shows the medical history of some employees.

These tables will help to understand and compare the answers given by the ENDBMS to some user-queries, as it is shown in the Appendix D.

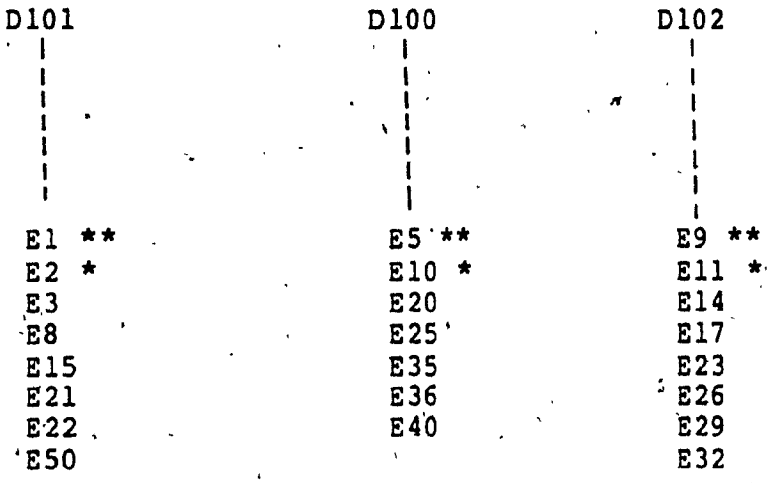


Table 5.1 Sample of department-employees association.

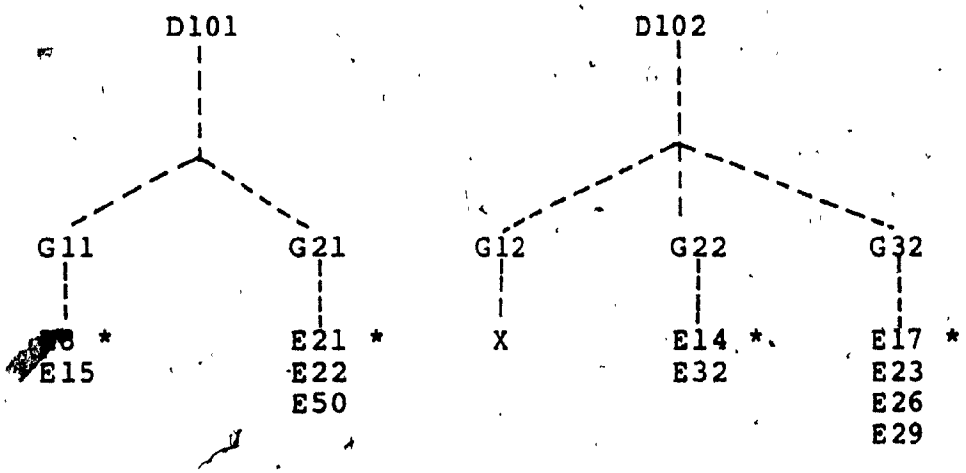


Table 5.2 Sample of department-groups and group-employees associations.

S1	S2	S3	S4	S5
P1	P1	P2	P3	P4
P5	P6	P10	P7	P8
P4	P9		P9	P2
	P10			

Table 5.3 Sample of supplier-parts association.

part		supplier
P1	from	S2
P2	>>	S3
P2	>>	S5
P7	>>	S4
P9	>>	S2
P9	>>	S4
P10	>>	S3
P10	>>	S2

Table 5.4 Sample of currently issued orders.

G11	G21	G22	G32
P1	P1	P4	P7
P3	P2	P5	P8
P5	P10	P6	P9
P10		P8	P10
		P9	
		P10	

Table 5.5 Sample of group-parts association.

E5	E10	E15	E20
broken leg	cold	eye operation	heart attack
alergy		cold	

Table 5.6 Sample of employee-med-history association.

CHAPTER 6

CONCLUSION

The goal of the current research was to design and implement an experimental data base management system. The Network approach - one of the three widely known approaches to DBMS - was chosen for the design of the ENDBMS, which is an acronym for Experimental Network Data Base Management System. The ENDBMS supports most of the suggested modern features of a DBMS and everything described at the design level has been implemented. The facility supported by SCHEMA is designed so that it can be used only by a 'declared' DBA to define the logical organization of the data base, using the Record and Set Type Network concepts. The ENDBMS also supports SUBSCHEMAS or user-views of the data base, providing a disciplined model for subschema construction and clearly determines the REAL and VIRTUAL subschema elements. The 'virtual' field-generation employes only four procedure types, but the subschema language can be changed to include more. Every user of ENDBMS must have his own subschema and his own access profile. In a real environment, even in our hypothetical company C, many users will require the same access profile. It is therefore efficient to have a common access profile for some users which may have different subschemas (derived from the common access profile). Another modification to ENDBMS subschema design would be the 'sharing' of a subschema by different

installed users with the same authorization. ENDBMS can operate in a 'read' and/or 'update' concurrent, multiuser environment, by using a two level concurrency mechanism. The lower - physical - level simulates the 'availability' of the used NOS files based on the fact that every file can be read by many users at the same time but can be updated by only one user; moreover updating of a NOS file requires that the file is attached to a NOS user job with a NOS exclusive 'write interlock'. The concurrency mechanism at the physical level provides all the required NOS files with the appropriate attach modes and at the same time avoids deadlock situations. The upper - logical - level determines which parts of the data base can be shared by some DML commands and which can be accessed only in a 'one at a time' basis. It employes shared locks for read, insert and replace DML type commands and exclusive locks for delete type DML commands. The data base 'lock-unit' is a stored schema record type file which must be 'opened' and 'closed' with explicit DML commands. A schema record type file can be accessed only after it has been opened with an appropriate access mode. The ENDBMS supports a five level security system which gurantees that only authorized users can use the ENDBMS and furthermore access only the authorized portion of the data base. The most important aspects of the security enforcement are the subschema and the USER ACCESS PROFILE. An authorized user is given an access profile which defines the schema elements (record and

set types) which can be accessed and determines the permitted operations on them; the access profile specifies restrictions on 'type' and 'occurrence' level. The read and modify field restrictions can be multiple (i.e., specify more than one field name), but the record occurrence restrictions must be single. In other words occurrences of a record type can be restricted from accessing, using only one field name to specify the restriction criterion. A future version of ENDBMS can expand this feature by allowing a combination of such single restriction criteria by using AND and/or OR operators. We already discussed the possible addition of the IMPLICIT access rights to an access profile, in (or not) association to all provided restrictions clauses. This would allow occurrences of a record type to be accessed only through their 'owner' within a specified set type. A subschema can be defined after the corresponding user access profile has been defined; it must be consistent at any time with respect to schema and associated profile definition. The subschema and the user access profile are used to enforce data independent security at the translation phase of any ENDBMS DML command; while the user profile is further used to enforce data dependent security at the execution phase. For the execution of any other ENDBMS command, the system checks their specific execution requirements to ensure consistency at all times. The cornerstone of the supported RECOVERY mechanism is the fact that the host O.S. does not allow more than one NOS

user to 'write' on a file at the same time. The ENDBMS assigns a unique 'recovery code' to those of its commands which require recovery in the case of an interrupt. This code is stored in the user's SECURITY ENTRY - in the system's USER LIST - when the command starts executing, and is used after an interrupt to identify the interrupted command and call the appropriate recovery routine to 'recover' the corresponding user.

The DML command set supported by ENDBMS is divided into two types of commands: the record type and the set type commands. The former uses the path implemented for a schema record type file and its record type currency indicator to retrieve information based on the stored occurrences of a particular record type. The later type uses the path implemented for the schema set occurrence and the two set type currency indicators to retrieve information based on occurrences of the owner and member record type files as well as on their 'stored' logical association. Both types include update commands to update a record type file or a set occurrence of some set type. Finally the ENDBMS provides an EDITOR facility to interactively correct lexical, syntactic or semantic errors in a user submitted command, by replacing only the line which contains the error.

The overall design of the ENDBMS was not affected by any particular application and thus any application that can

be described with the system's network data model, can be handled. It is evident from the schema definition that there is no 'primary' key specification in any record type definition. As a consequence there is no need of 'searching' techniques and no overhead of keeping indexes and inverted file lists and as we indicated the 'searching' for the "GET NEXT WHERE" is done sequentially. It is obvious that such a search mechanism to locate a record occurrence, based on some search criterion, in a medium or large data base is neither the best nor the suggested one. However the design of the ENDBMS schema was not accidental; the design started with the assumption that a Content Addressable Memory (CAM) will be used for the data base storage at some future time. In such a memory a record occurrence, of some record type, can be retrieved by specifying only a field value. It is also believed that when such memory becomes cheap, the design of a DBMS will be oriented to this direction and a great deal of addressing and searching programming as well as processing overhead will be eliminated.

The present implementation of the ENDBMS cannot be considered as host computer system independent or 'portable'. It relies upon the Cyber Record Manager (CRM); a component of the host NOS operating system. The FORTRAN 4 implementation of ENDBMS uses some CRM functions to: (1) define the file organization of the ENDBMS system and data

base files (2) prepare a file for processing and (3) read and update a file. If these CRM function 'calls' can be excluded and instead FORTRAN 4 code be used, to perform the same actions the ENDBMS can be implemented in any computer system which supports a FORTRAN 4 compiler. It is also possible that these CRM functions can be changed by equivalent ones in another computer system in order to make possible the ENDBMS implementation on this system. In the current version the CRM is used to carry out all the I/O operations; however the CRM functions could be assigned to a number of microprocessors. One of them could serve as a supervisor and receive any I/O transfer-request from the ENDBMS. Each of the other microprocessors could be assigned to the disk unit to add intelligence to the secondary storage. When the supervisor receives a request to retrieve or store a record occurrence, it looks at its 'data base key', derives the 'logical' and 'physical' unit number and then assigns the necessary operation to the disk microprocessor of the known 'physical' unit. Another future modification would be the expansion of the DML. We have explained the limitations of the current DML design and we suggested that inclusion of three more operators such as: UNION, INTRESECT and SAVE would provide the ability to answer 'union-type' and 'intersect-type' queries related to a many to many logical association between two record types. Queries whose answer is a group of occurrences which is logically a union or intersection of some groups of

occurrences generated by visiting different 'set occurrences' within the same set type, can not be answered. An example for this would be: "find the parts used by the assemblies A1, A3 and A5" related to ASSEMBLY-PARTS m:m association where a part may be used by different assemblies. It seems that the answer group, is the union of the answer groups of the simple queries - which can be answered by the current DML - "find all parts used by A1", "find all parts used by A3" and "find all parts used by A5".

REFERENCES

[APL] IBM Corporation. APL Data Language Program Description/Operations Manual. Form No. SB21-1805.

[AHO-ULL] Aho A.V., Ullman J.D. "Principles of Compiler Design", Addison-Wesley Publishing Company, 1977.

[BLEI-67] Bleier R.E. "Treating hierarchical data structures in the SDC Time-Shared Data Management System (TDMS)", Proceedings of ACM National conference 1967, ACM, New York 1967, pp 41-49.

[CHAM-76] Chamberlin D.D. "Relational Data-Base Management Systems", Computing Surveys, Vol. 8, No. 1, March 1976.

[CODD-70] Codd E.F. "A Relational model of data for large shared data banks", Communications ACM, June 1970, pp 377-397.

[CODD-71] Codd E.F. "Further normalization of the data-base relational model", Courant Computer Science Symbosia 6, "data base systems", New York, May 1971, Prentice-Hall, pp 33-64.

[CONW-72] Conway R.W., et al. "On the Implementation of Security Measures in Information Systems", CACM 15-4, April 1972, pp 211-220.

[DATE-77] Date C.J. "An Introduction to Database Systems", Second Edition, Addison-Wesley Publishing Company, Inc. 1977.

[DBTG-69] Data Base Task Group of CODASYL Programming Language Committe Report, October 1969. Available from ACM, BCS, IAG.

[DBTG-71] Data Base Task Group of CODASYL Programming Language Committee Report, April 1971. Available from ACM, BCS, IAG.

[DMS] Sperry-Univac. Univac 1100 Series Data Management System (DMS-1100): 1. Schema Definition. Order

No. UP-7907, 2. Data Manipulation Language Programmer Reference Manual. Order No. UP-7908.

[DOUQUE] Douque B.C.M. "PHOLAS: A modular implementation of the DBTG proposals", Infotech, State of the art report on Database Systems, Infotech Int. Ltd., Maidenhead, Uk, 1975, pp 331-348.

[EVER-74] Everest G.C. "The Objectives of Data Base Management", Information Systems COINS IV (Tou), Plenum Press, New York, 1974, pp 1-35, also MISRC-WP-71-04.

[FRY-76] Fry C.J., Sibley E.H. "Evolution of Data Base Management Systems", Computing Surveys, v.8 No.1, March 1976.

[HOF-69] Hoffman L.J. "Computers and Privacy: A Survey", Computing Surveys, Vol.1, No.2, June 1969.

[IDS] Honeywell Information Systems. "Series 600/6000 Integrated Data Store Reference Manual. Order No. CPB-1565.

[INGRES-76] Stonebraker M., Wong E., Kreps P. "The Design and Implementation of INGRES", ACM Transactions on Database Systems, Vol. 1, No. 3, September 1976, pp 189-222.

[KIM-79] Kim Won. "Relational Database Systems", Computing Surveys, Vol.11, No.3, September 1979.

[MAGNUM] Tymshare Inc. MAGNUM Reference Manual, Nov, 1975.

[MARS-VI] Control Data Corporation. "MARS VI multi-access retrieval system reference manual", 44625500,1970.

[MAR-NOR] Martin J., Norman A.R.D. "The computerized Society", Englewood Cliffs, N.J, Prentice-Hall (1970).

[MART-77] Martin J. "Computer Data-Base Organization", Second Edition, Prentice-Hall, Inc., Englewood Cliffs, N.J, 1977.

[MARTIN] Martin J. "Security, Accuracy and Privacy in computer systems", Prentice-Hall Inc., Englewood Cliffs, N.J.

[MICH-76] Michaels A.S., Mittman B., Carlson C.R. "A Comparison of the Relational and CODASYL Approaches to Data Base Management", Computing Surveys, Vol.8, No.1, March 1976.

[POT-LEB] Potier D., Leblanc P. "Analysis of Locking Policies in Database Systems", Communications of ACM, October 1980, Vol.23, No.10.

[SYST-R] Astrahan M.M., et al, "System R: A relational approach to data base management", ACM Transactions on Database Systems, No.6, June 1976.

[TSLO-76] Tsichritzis D.C., Lochovsky F.H. "Hierarchical Data Base Management: A Survey", Computing Surveys, Vol.8, No.1, March 1976.

[TSLO-77] Tsichritzis D.C., Lochovsky F.H. "Data Base Management Systems", Accademic Press, New York, N.Y, 1977.

[TAY-FRA] Taylor R.W., Frank R.L. "CODASYL Data-Base Management Systems", Computing Surveys, Vol.8, No.1, March 1976.

[ULLM-80] Ullman J.D. "Principles of Database Systems", Computer Science Press, Inc., 1980.

[VASS-80] Vassiliou Y. "Functional dependencies and incomplete information", Proceedings on VLDB, 6th International Conference on Very Large Data Bases, Montreal, October 1980.

[VERH-78] Verhofstad J.S.M. "Recovery Techniques For Database Systems", Computing Surveys, Vol.10, No.2, June 1978.

CDC REFERENCE MANUALS

[NOS-V1] Control Data Corporation. NOS VERSION 1 REFERENCE MANUAL. Volume 1 of 2, Revision J.

[NOS-V2] Control Data Corporation. NOS VERSION 1 REFERENCE MANUAL. Volume 2 of 2, Revision J.

[CRM-AAM] Control Data Corporation. CYBER RECORD MANAGER ADVANCED ACCESS METHODS VERSION 2 REFERENCE MANUAL, Revision A.

[CRM-F-G] Control Data Corporation. CYBER RECORD MANAGER VERSION 1 GUIDE FOR USERS OF FORTRAN EXTENDED VERSION 4, Revision B.

[FORT-4] Control Data Corporation. FORTRAN EXTENDED VERSION 4 REFERENCE MANUAL, Revision D.

APPENDIX A

A.1 INSTALLATION PROCEDURE

```
/get,install
/begin,gen,install
REVERT.   NO ERROR(S) OCCURED
/get,wmfile
/attach,dbalib
/$library,aamlib,dbalib
$LIBRARY,AAMLIB,DBALIB.
/ftn,i=wmfile,l=0
      .054 CP SECONDS COMPILATION TIME
/lgo,mfile
      .544 CP SECONDS EXECUTION TIME.
```

A.2 DBA ACCESS PROCEDURE

```
/attach,dbalib
/attach,endbms
/$library,aamlib,dbalib
$LIBRARY,AAMLIB,DBALIB.
/endbms
```

A.3 USER ACCESS PROCEDURE

```
/attach,userlib/un=kegfe62
/attach,endbms/un=kegfe62
/$library,aamlib,userlib
$LIBRARY,AAMLIB,USERLIB.
/endbms
```

A.4 ENDBMS DELETION PROCEDURE

```
/get,delfile
/begin,delproc,delfile
```

APPENDIX B

LIST OF THE SCHEMA-SECURITY-SUBSCHEMA-DML LL(1) GRAMMARS

NOTE
---
the integer in front of any production is the production number which uniquely identifies the corresponding production within each grammar.

ABREVIATIONS

s-r : schema record
s-f : schema field
s-s : schema set
ss-r: subschema record
ss-f: susbchema field
ss-s: subschema set

A. LL(1) SCHEMA GRAMMAR

```
 1  <command>:= DEFINE-SCHEMA <schema-definition>  ;
 2           := INSERT-SRT   s-r-definition>  ;
 3           := INSERT-SST  <s-s-definition>  ;
 4           := DELETE-SRT  <s-r-name>  ;
 5           := DELETE-SST  <s-s-name>  ;
 6           := LIST-SCHEMA  ;
 7  <schema-definition>:= <s-r-list> <s-s-list>
 8  <s-r-list>         := <s-r-definition> <s-r-tail>
 9  <s-r-list>         := <s-r-definition> <s-r-tail>
10                     := <empty>
11  <s-r-definition>:= RECORD <s-r-name>
                                <s-f-list>
                              END
12  <s-f-list>       := <s-f-definition> <s-f-tail>
13  <s-f-tail>       := <s-f-definition> <s-f-tail>
14                   := <empty>
15  <s-f-definition>:= FIELD <s-f-name> OF <type>
16  <type>           := NUMERIC ( <integer> )
17                   := CHAR    ( <integer> )
18  <s-s-list>       := <empty>
19                   := <s-s-definition> <s-s-list>
20  <s-s-definition>:= SET <s-s-name>
                                <set-part>
                            END
21  <set-part>       := OWNER  <s-r-name>
```

MEMBER <s-r-name>

B. LL(1) SECURITY GRAMMAR

```
 1  <command>:= INSTALL-USERS <user-list>. ;
 2           := REMOVE-USERS <user-number-list>  ;
 3           := LIST-USERS  ;
 4           := DEFINE-PROFILES <profile-list>  ;
 5           := REMOVE-PROFILES <user-number-list>  ;
 6           := LIST-PROFILES  <user-number-list>  ;
 7           := ADD-RIGHTS  <add-spec>  ;
 8           := REMOVE-RIGHTS <remove-spec>  ;
 9           := SUSPEND-ACCESS <user-number-list>  ;
10           := RESTORE-ACCESS <user-number-list>  ;
11           := LIST-ACTIVITY  ;
12           := LOCK-SYSTEM  ;
13           := UNLOCK-SYSTEM  ;
14           := RECOVER <user-number-list>  ;
15           := PASSWORD <old-pasw> <new-passw>  ;
16  <user-list> := ( <user#> <user-passw> ) <user-tail>
17  <user-tail> := ( <user#> <user-passw> ) <user-tail>
18              := <empty>
19  <user-number-list>:= <user#> <user-number-tail>
20  <user-number-tail>:= <user#> <user-number-tail>
21                    := <empty>
22  <profile-list>    := <profile> <profile-tail>
```

```
23  <profile-tail>     := <profile> <profile-tail>
24                     := <empty>
25  <profile>          := USER <user#> <rec-rights-list>
                                       <set-rights-list>
26  <rec-rights-list>  := <rec-rights> <rec-rights-tail>
27  <rec-rights-tail>  := <rec-rights> <rec-rights-tail>
28                     := <empty>
29  <rec-rights>       := RECORD <s-r-name>
                          CODE   <rt-access-code>
                          <restrictions>

30  <restrictions>  := RFR <option-1>
                       MFR <option-1> <tail-1>
31  <tail-1>        := RROR <option-2>
                    := MROR <option-2> <tail-2>
32  <tail-2>        := DROR <option-2>
33  <option-1>      := NIL
34                  := <s-f-name> <s-f-name-tail>
35  <option-2>      := NIL
36                  := <s-f-name> <srel-op> <quoted-value>
37  <rel-op>        :=  =
38                  :=  <
39                  :=  >
40                  :=  <>
41  <s-f-name-tail>:= <s-f-name> <s-f-name-tail>
42                  := <empty>
43  <set-rights-list>:= <empty>
```

```
44                      := <set-rights> <set-rights-list>
45  <set-rights>        := SET <s-s-name>
                           CODE <st-access-code>o
46  <add-spec>          := USER <user#> <add-tail>
47  <add-tail>          := <record-rights>
48                      := <set-rights>
49  <remove-spec>       := USER <user#> <remove-tail>
50  <remove-tail>       := RECORD <s-r-name>
51                      := SET <s-s-name>
```

C. LL(1) SUBSCHEMA GRAMMAR

```
 1  <command>:= DEFINE-SUBSCHEMA <subsch-definition>  ;
 2           := DELETE-SUBSCHEMA  ;
 3           := LIST-SUBSCHEMA  ;
 4           := INSERT-SSRT <ss-r-definition>  ;
 5           := INSERT-SSST <ss-s-definition>  ;
 6           := DELETE-SSRT <ss-r-name>  ;
 7           := DELETE-SSST <ss-s-name>  ;
 8  <subsch-definition>:= <ss-rec-list> <ss-set-list>
 9  <ss-rec-list>      := <ss-r-definition> <ss-rec-tail>
10  <ss-rec-tail>      := <ss-r-definition> <ss-rec-tail>
11                     := <empty>
12  <ss-rec-def>       := RECORD <ss-r-name> <r-option>
13  <r-option>         := REAL <WITH-BASE <s-r-name>
                                <field-list-1>
                           END
```

```
14                          := VIRTUAL WITH-BASE <s-r-name>
                                    <field-list-2>
                               END

15  <field-list-1>  := <ss-f-def-1>  <field-tail-1>
16  <field-tail-1>  := <ss-f-def-1>  <field-tail-1>
17                  := <empty>

18  <field-list-2>  := <ss-f-def-2>  <field-tail-2>
19  <field-tail-2>  := <ss-f-def-2>  <field-tail-2>
20                  := <empty>

21  <ss-f-def-1>    := FIELD <ss-f-name>
22  <ss-f-def-2>    := FIELD <ss-f-name> <f-option>
23  <f-option>      := REAL FROM <s-r-name> .<s-f-name>
24                  := VIRTUAL <procedure-type>

25  <procedure-type>:= RESULT-OF <s-r-name>.<s-f-name>
                                 <ar-oper> <number>
26                  := SUM-OF <s-r-name>  (
                                   <s-f-name-list>
                                        )
27                  := AVERAGE-OF <s-r-name> (
                                      <s-f-name-list>
                                           )
28                  := MERGE-FROM   <s-r-name> (
                                      <s-f-name-list>
                                           )
29  <ar-oper>       := *
30                  := /
31  <s-f-name-list> := <s-f-name> <s-f-tail>
32  <s-f-tail>      := <s-f-name> <s-f-tail>
```

```
33                      := <empty>.

34  <ss-set-list>       := <ss-s-definition> <ss-s-tail>

35  <ss-s-tail>         := <ss-s-definition> <ss-s-tail>

36  <ss--s-definition>:= SET <ss-s-name>
                            REAL WITH-BASE <s-r-name>
                            <ss-s-part>

37  <ss-s-part>           := OWNER <ss-r-name>
                             MEMBER <ss-r-name>
                             END
```

D. LL(1) DML GRAMMAR

```
 1  <command>       := <label-field> <command-type> ;

 4  <command-type>:= GET <get-part>

 5                  := LIST <list-part>

 6                  := IF CURRENT <if-part>

 7                  := GOTO <label>

 8                  := SET <set-part>

 9                  := LOCK FOR <lock-part>

10                  := RELEASE <rec-name-list>

11                  := INSERT IN <rec-name> <occurrence-list>

12                  := DELETE CURRENT OF <rec-name>

13                  := REPLACE <field-name> <replace-part>

14                  := INCLUDE CURRENT OF <rec-name>
                         IN <set-name>

15                  := REMOVE CURRENT MEMBER FROM <set-name>

16  <get-part>      := FIRST <first-option>

17                  := NEXT  <next-option>
```

```
18                  := OWNER OF CURRENT OF <get-owner-part>
19  <first-option>:= OF <rec-name>
20                  := OWNER  IN <set-name>
21                  := MEMBER IN <set-name>
22  <next-option> := OF <rec-name> <where-clause>
23                  := OWNER  IN <set-name>
24                  := MEMBER IN <set-name>

25  <where-clause>:= WHERE <field-name> <relop> "value"
26                  := <empty>
27  <relop>       :=  =
28                  :=  <
29                  :=  >
30                  :=  <>
31  <get-owner-part>:= <rec-name> IN <set-name>

32  <list-part>   := CURRENT OF <rec-name>
33                  := ALL <list-option>
34  <list-option> := OF <rec-name>
                        <where-clause>
                        <sort-clause>
35                  := MEMBERS IN <set-name>
                        <where-clause>
                        <sort-clause>
36  <sort-clause> := IN <sort-option> USING <field-name>
37                  := <empty>
 2  <label-field> := <empty>
 3                  := <label>
38  <sort-option> := ASCENDING
39                  := DESCENDING
```

```
40  <if-part>       := OF <rec-name> <if-option>

41                  := MEMBER IN <set-name> LAST
                       GOTO <label>

42  <if-option>     := LAST GOTO <label>

43                  := HAS <field-name> <relop> "value"
                       GOTO <label>

44  <set-part>      := LINELIMIT TO <integer>

45                  := CI OF <rec-name> TO <set-option>

46                  := OCI IN <set-name> TO <set-tail>

47  <set-option>    := OCI IN <set-name>

48                  := MCI IN <set-name>

49  <lock-part>     := READ <rec-name-list>

50                  := DELETE <rec-name-list>

51  <rec-name-list>:= <rec-name> <rec-name-tail>

52  <rec-name-tail>:= <rec-name> <rec-name-tail>

53                  := <empty>

54  <occurrence-list>:= "occurrence" <occurrence-tail>

55  <occurrence-tail>:= "occurrence" <occurrence-tail>

56                      := <empty>

57  <replace-part>  := IN CURRENT OF <rec-name>
                       WITH "value"

58  <set-tail>      := CI OF <rec-name>
```

NOTE

1. The <label> must be an integer.
2. The "value" must be a string of characters or digits within quotes.
3. A record occurrence must be within quotes

and its field values must be separated with the character '/'.

4. The LINELIMIT must be < 136 and > 80.

APPENDIX C

LIST OF THE ENDBMS ERROR MESSAGES

NOTE: the ENDBMS error messages are divided in five groups. The first group contains the error messages with error codes from 1 to 118. These messages are generated during the translation or execution phase of any ENDBMS command by all the 'command handlers'. The next four groups contain the error messages with error codes 151 to 170, 201 to 235, 301 to 329 and 401 to 444 and correspond to SML, SL, SSML and DML respectively. These error messages are generated by the four automatic LL(1) parsers during the parsing of a command which belongs to their subsystem. Each parser uses the stack and its local parsing table to test the syntax of a command and in the case of an error situation uses the error codes to generate error messages without testing separately for each case. The repetition of some error messages in different groups is due to the fact that all subsystems have been implemented independently.

001 ENDBMS NOT AVAILABLE.

002 COMMAND KEY-WORD EXPECTED.

003 INVALID COMMAND KEY-WORD.

005 USER NUMBER EXPECTED ; ACCESS IMPOSSIBLE.

006 USER NUMBER AND PASSW START WITH ALPHABETIC.

007 INVALID USER NUMBER.

008 USER PASSWORD EXPECTED.

009 INVALID USER PASSWORD.

010 PROCESSING MODE EXPECTED (I OR B).

011 INVALID PROCESSING MODE.

012 ACCESS IMPOSSIBLE WITHOUT IDENTIFICATION.

013 ONLY DBA CAN EXECUTE THIS COMMAND.

014 USER NUMBER MUST BE UNIQUE.

015 USER LIST IS EMPTY.

016 TEMPORARILY SUSPENDED.

017 TRY AGAIN AND CHECK YOUR PROFILE.

018 COMMAND REQUIRES ENDBMS TO BE LOCKED.

019 COMMAND REQUIRES NO USER ACTIVITY.

020 RUN ONLY ONE JOB AT A TIME.

021 RECOVERY MUST BE CALLED-INFORM DBA.

022 SCHEMA RECORD-TYPE LIST EMPTY.

023 SCHEMA SET-TYPE LIST EMPTY.

024 INVALID SCHEMA RECORD-TYPE NAME.

025 INVALID SCHEMA SET-TYPE NAME.

026 INVALID SCHEMA FIELD NAME.

027 DESTROY ENDBMS AND START INSTALLATION OVER.

028 SCHEMA IS ALREADY DEFINED.

029 FIELD LENGTH MUST BE INTEGER.

030 DUPLICATE SCHEMA RECORD-TYPE NAME.

031 DUPLICATE SCHEMA SET-TYPE NAME.

032 DUPLICATE SCHEMA FIELD NAME.

033 SCHEMA OWNER TYPE,NOT DEFINED BEFORE.

034 SCHEMA MEMBER TYPE,NOT DEFINED BEFORE.

035 USER-PROFILE MUST BE REMOVED FIRST.

036 USER-PROFILE ALREADY DEFINED.

037 ACCESS CODE MUST BE INTEGER NUMBER.

038 INVALID ACCESS CODE.

039 'RFR' AND 'MFR' SETS MUST BE EXCLUSSIVE.

040 CONFLICT BETWEEN 'ROR'-FIELD NAME AND 'RFR'-SET.

041 TYPE CONFLICT IN RECORD OCCURRENCE RESTRICTION.

042 SET TYPE REQUIRES ACCESS OVER OWNER AND MEMBER.

043 A LABEL MUST BE AN INTEGER.

044 CODE CONFLICT BETWEEN SET AND OWNER(OR MEMBER).

045 USER MUST BE SUSPENDED AND NOT ACTIVE.

046 INVALID LABEL--NOT FOUND IN THIS INPUT SESSION.

047 USER'S PROFILE IS NOT DEFINED.

048 CONFLICT IN ACCESS CODE.

049 WARNING % USER IS ALREADY SUSPENDED.

050 WARNING % USER IS NOT SUSPENDED.

051 PLEASE LOG OFF AS SOON AS POSSIBLE.

052 DBA CANOT EXECUTE THIS COMMAND.

053 SUBSCHEMA IS ALREADY DEFINED.

054 DUPLICATE SUBSCHEMA RECORD NAME.

055 DUPLICATE SUBSCHEMA FIELD NAME

056 DUPLICATE SUBSCHEMA SET NAME.

057 RECORD BASE-NAME OUT OF USER'S PROFILE.

058 SET BASE-NAME OUT OF USER'S PROFILE.

059 CONFLICT WITH READ FIELD RESTRICTIONS.

060 CONFLICT IN NUMBER OF FIELD NAMES.

061 INVALID COMPATIBLE R-NAME IN F-DEFINTION.

062 INVALID COMPATIBLE F-NAME IN F-DEFINTION.

063 TYPE CONFLICT IN VIRTUAL F-DEFINITION.

064 SUBSCHEMA OWNER NOT DEFINED BEFORE.

065 SUBSCHEMA MEMBER NOT DEFINED BEFORE.

066 INVALID SET DEFINITION-INCOMPATIBLE BASES.

067 R.T-FILE HAS A READ LOCK.

068 SUBSCHEMA HAS NOT BEEN DEFINED.

069 AT LEAST ONE F-DEFINTION MUST BE FROM BASE.

070 INVALID SUBSCHEMA RECORD NAME.

071 INVALID SUBSCHEMA SET NAME.

072 INVALID SUBSCHEMA FIELD NAME.

073 A REAL SUSBSCHEMA RECORD CANOT BE EXPANDED.

074 NUMBER IN VIRTUAL FIELD TOO BIG.

075 SUBSCHEMA IS EMPTY.

076 FIELD VALUE CANOT BE EMPTY.

077 INVALID RECORD TYPE NAME.

078 RECORD TYPE FILE MUST BE LOCKED PROPERLY.

079 INVALID SET TYPE NAME.

080 OWNER AND MEMBER TYPE MUST BE LOCKED PROPERLY.

081 INVALID FIELD NAME IN SEARCH CLAUSE.

082 FIELD NAME MUST BE REAL.

083 MATCHING QUOTE MISSING.

084 RECORD TYPE FILE IS EMPTY.

085 OWNER TYPE FILE IS EMPTY.

086 OWNER'S MEMBERSHIP LIST IS EMPTY.

087 END OF RECORD TYPE FILE REACHED.

088 END OF MEMBERSHIP LIS REACHED.

089 INCOMPATIBILITY IN SET AND RECORD TYPE NAMES.

090 MEMBER RECORD TYPE FILE IS EMPTY.

091 CURRENT IS NOT MEMBER OF ANY SET OCCURRENCE.

092 ERROR IN NUMERIC VALUE.

093 CURRENT OCCURRENCE IS ALREADY INCLUDED.

094 INVALID FIELD NAME IN SORT CLAUSE.

095 A LABEL MUST BE UNIQUE.(IN EACH SESSION)

096 'ROR' SECURITY ENFORCED.

097 'MOR' SECURITY ENFORCED.

098 'DOR' SECURITY ENFORCED.

099 OUTPUT LINE LIMIT MUST BE >80 AND < 136

100 COMMAND DOES NOT ACCEPT LABEL.

101 ATTEMPT TO VIOLATE THE GIVEN ACCESS RIGHTS.

102 RECORD TYPE FILE IS ALREADY RELEASED.

103 RECORD TYPE FILE MUST BE LOCKED FOR 'DELETE'.

104 PREVIOUS FIELD NAME WAS INVALID.

105 'MFR' SECURITY ENFORCED.

106 LABEL OR

107 OWNER AND MEMBER OCCURRENCE MUST BE 'ROR' FREE.

108 'REMOVE' COMMAND REQUIRES REAL MEMBER.

109 'INCLUDE' COMMAND REQUIRES REAL MEMBER.

110 RECORD TYPE MUST BE REAL.

111 ILLEGAL OCCURRENCE FORMAT FOR 'INSERT' COM.

112 FIELD VALUE MISSING.

113 R.T-FILE HAS A DELETE LOCK.

114 FIELD TERMINATOR MISSING.

115 VALUE LENGTH EXCEEDS FIELD LENGTH.

116 NO SET OCCURRENCE IS SELECTED.

118 NOT FOUND.

151 'CHAR' EXPECTED.

152 'DEFINE-SCHEMA' EXPECTED.

153 'DELETE-SRT' EXPECTED.

154 'DELETE-SST' EXPECTED.

155 'END' EXPECTED.

156 'FIELD' EXPECTED.

157 'INSERT-SRT' EXPECTED.

158 'INSERT-SST' EXPECTED.

159 'LIST-SCHEMA' EXPECTED.

160 'MEMBER' EXPECTED.

161 'NUMERIC' EXPECTED.

162 'OF' EXPECTED.

163 'OWNER' EXPECTED.

164 'RECORD' EXPECTED.

165 'SET' EXPECTED.

166 'IDENTIFIER' EXPECTED.

167 'NUMBER' EXPECTED.

168 '(' EXPECTED.

169 ')' EXPECTED.

170 'COMMAND TERMINATOR' EXPECTED.

201 'ADD-RIGHTS' EXPECTED.

202 'CODE' EXPECTED.

203 'DEFINE-PROFILE(S)' EXPECTED.

204 'DROR' EXPECTED.

205 'INSTALL-USER(S)' EXPECTED.

206 'LIST-ACTIVITY' EXPECTED.

207 'LIST-PROFILE(S)' EXPECTED.

208 'LIST-USERS' EXPECTED.

209 'LOCK-SYSTEM' EXPECTED.

210 'MFR' EXPECTED.

211 'MROR' EXPECTED.

212 'NIL' EXPECTED.

213 'PASSWORD' EXPECTED.

214 'RECORD' EXPECTED.

215 'RECOVER' EXPECTED.

216 'REMOVE-PROFILE(S)' EXPECTED.

217 'REMOVE-RIGHTS' EXPECTED.

218 'REMOVE-USER(S)' EXPECETED.

219 'RESTORE-ACCESS' EXPECTED.

220 'RFR' EXPECTED.

221 'RROR' EXPECTED.

222 'SET' EXPECTED.

223 'SUSPEND-ACCESS' EXPECTED.

224 'UNLOCK-SYSTEM' EXPECTED.

225 'USER' EXPECTED.

226 'IDENTIFIER' EXPECTED.

227 'NUMBER' EXPECTED.

228 '=' EXPECTED.

229 '<' EXPECTED.

230 '>' EXPECTED.

231 '<>' EXPECTED.

232 '(' EXPECTED.

233 ')' EXPECTED.

234 'QUOTED VALUE' EXPECTED.

235 'COMMAND TERMINATOR' EXPECTED.

301 'AVERAGE-OF' EXPECTED.

302 'DEFINE-SUBSCHEMA' EXPECTED.

303 'DELETE-SSRT' EXPECTED.

304 'DELETE-SSST' EXPECTED.

305 'DELETE-SUBSCHEMA' EXPECTED.

306 'END' EXPECTED.

307 'FIELD' EXPECTED.

308 'FROM' EXPECTED.

309 'INSERT-SSRT' EXPECETD.

310 'INSERT-SSST' EXPECTED.

311 'LIST-SUBSCHEMA' EXPECTED.

312 'MEMBER' EXPECTED.

313 'MERGE-FROM' EXPECTED.

314 'OWNER' EXPECTED.

315 'REAL' EXPECTED.

316 'RECORD' EXPECTED.

317 'RESULT-OF' EXPECTED.

318 'SET' EXPECTED.

319 'SUM-OF' EXPECTED.

320 'VIRTUAL' EXPECTED.

321 'WITH-BASE' EXPECTED.

322 'IDENTIFIER' EXPECTED.

323 'NUMBER' EXPECTED.

324 '(' EXPECTED.

325 ')' EXPECTED.

326 '*' EXPECTED.

327 '/' EXPECTED.

328 '.' EXPECTED.

329 'COMMAND TERMINATOR' EXPECTED.

401 'ALL' EXPECTED.

402 'ASCENDING' EXPECTED.

403 'CI' EXPECTED.

404 'CURRENT' EXPECTED.

405 'DELETE' EXPECTED.

406 'DESCENDING' EXPECTED.

407 'FIRST' EXPECTED.

408 'FOR' EXPECTED.

409 'FROM' EXPECTED.

410 'GET' EXPECTED.

411 'GOTO' EXPECTED.

412 'HAS' EXPECTED.

413 'IF' EXPECTED.

414 'INCLUDE' EXPECTED.

415 'INSERT' EXPECTED.

416 'IN' EXPECTED.

417 'LAST' EXPECTED.

418 'LINELIMIT' EXPECTED.

419 'LIST' EXPECTED.

420 'LOCK' EXPECTED.

421 'MCI' EXPECETD.

422 'MEMBERS' EXPECTED.

423 'MEMBER' EXPECTED.

424 'NEXT' EXPECTED.

425 'OCI' EXPECTED.

426 'OF' EXPECTED.

427 'OWNER' EXPECTED.

428 'READ' EXPECTED.

429 'RELEASE' EXPECTED.

430 'REMOVE' EXPECTED.

431 'REPLACE' EXPECTED.

432 'SET' EXPECTED.

433 'TO' EXPECTED.

434 'USING' EXPECTED.

435 'WHERE' EXPECTED.

436 'WITH' EXPECTED.

437 'IDENTIFIER' EXPECTED.

438 'NUMBER' EXPECTED.

439 '=' EXPECTED

440 '<' EXPECTED.

441 '>' EXPECTED.

442 '<>' EXPECTED.

443 ' " ' EXPECTED.

444 ';' OR '$' EXPECTED.

APPENDIX D

NOTE:

The following list shows the user number and passwords of all users that will be installed on ENDBMS and authorized to access the data base of the hypothetical company C.

TYPE OF USER	USER #	USER PW
DBA	Z01	Z02
CHIEF EXECUTIVE OFFICER	CEO	CEO1
MANAGER OF DEPARTMENT 102	DM102	DMD2
SECRETARY OF DEPARTMENT 102	DS102	DSD2
LEADER OF GROUP G22	D2GL22	GRL22
LEADER OF GROUP G32	D2GL32	GRL32
ADMINISTRATOR OF MEDICAL PERSONNEL	MPA	MPA00
ADMINISTRATOR OF PURCHASING DEPARTMENT	PDA	PDA00

The next pages of this appendix show samples of actual interface sessions with the ENDBMS.

D.1 SAMPLE INTERFACE OF DBA

The interface samples of the DBA's interaction with ENDBMS are divided into three groups as follows :

(1). the first group shows how security is enforced by ENDBMS at log-in time when the DBA or any other user attempts to access the ENDBMS. This group also shows how the ENDBMS records the 'user activity' by keeping track of any log-in and log-off activity.

(2). in the second group the DBA defines the schema for the data base of the hypothetical company C, lists the 'stored' schema definition and finaly using various DML commands shows the contents of the data base as they are exactly shown in chapter 5. The actual insertion of record occurrences into different record type files and their inclusion into different set occurrences is not shown, but it has been done.

(3). the third group shows how the access profiles of the authorized users, described in chapter 5, are defined by the DBA so that they define exactly their given access capabilities.

```
/attach,dbalib
/$library,aamlib,dbalib
$LIBRARY,AAMLIB,DBALIB.
/attach,endbms
/endbms


 = CONCORDIA UNIVERSITY ======= ENDBMS =
 = DATE : 81/08/17.
 = TIME : 16.44.20.


  *READY*


? user , 4abc , z02 ,i$
  USER , 4ABC , Z02 ,I$

  *USER NUMBER EXPECTED ; ACCESS IMPOSSIBLE.
  *USER NUMBER AND PASSW START WITH ALPHABETIC.

 LOG OFF TIME 16.44.47.

      .233 CP SECONDS EXECUTION TIME.
/


 endbms


 = CONCORDIA UNIVERSITY ======= ENDBMS =
 = DATE : 81/08/17.
 = TIME : 16.45.25.


  *READY*


? user , z04 , z02 ,i$
  USER , Z04 , Z02 ,I$

  *INVALID USER NUMBER.

 LOG OFF TIME 16.46.03.

      .217 CP SECONDS EXECUTION TIME.
/
```

```
endbms


= CONCORDIA UNIVERSITY ======= ENDBMS =
= DATE : 81/08/17.
= TIME : 16.46.37.


 *READY*


? user , z01 , zabc , i$
  USER , Z01 , ZABC , I$

  *INVALID USER PASSWORD.

 LOG OFF TIME 16.49.59.

      .220 CP SECONDS EXECUTION TIME.


endbms


 = CONCORDIA UNIVERSITY ======= ENDBMS =
 = DATE : 81/08/17.
 = TIME : 16.48.08.


  *READY*


? user , z01 , z02 , xxx $
  USER , Z01 , Z02 , XXX $

  *INVALID PROCESSING MODE.

 LOG OFF TIME 16.48.35.

      .209 CP SECONDS EXECUTION TIME.
```

```
endbms


= CONCORDIA UNIVERSITY ======= ENDBMS =
= DATE : 81/08/17.
= TIME : 18.07.55.


 *READY*


? list-schema$
  *ACCESS IMPOSSIBLE WITHOUT IDENTIFICATION.

  *READY*

? user z01 z02 i$

  *READY*

? list-users$
  *USER LIST EMPTY.

  *READY*

? install-users : ( rda , rda00 )
                  ( mra , mra00 )
                  ( ceo , ceo1 )
                  (ds102 , dsd2 )
                  (dm102 , dmd2 )
  (d2s122,sr122), (d2s132,sr132 )$

  *READY*

? suspend-access : ceo $

  *READY*

? list-activity$
  *ACTIVITY LIST EMPTY.

  *READY*

?
```

```
list-users$

.....
    LIST OF INSTALLED USERS
.....

    USER-NUM        USER-PW
    --------        -------

    D2GL32          GRL32

    D2GL22          GRL22

    DM102           DMD2

    DS102           DSD2

   *CEO             CEO1

    MPA             MPA00

    PDA             PDA

    END OF USER LIST
.....
  *READY*

? suspend-access : ceo$
  SUAPEND-ACCESS : CEO$

  *WARNING % USER IS ALREADY SUSPENDED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? drop

  *READY*

? remove-users : dm102,ds102 $
  REMOVE-USERS : DM102 DS102 $

  *USER MUST BE SUSPENDED AND NOT ACTIVE.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.
```

```
? drop
  *READY*
? suspend-access : dm102 , ds102 $
  *READY*
? remove-users : ds102 , dm102 $
  *READY*
? list-users$

.....
    LIST OF INSTALLED USERS
.....
    USER-NUM        USER-PW
    --------        -------

    D2GL32          GRL32

    D2GL22          GRL22

    MPA             MPA00

    PDA             PDA

    END OF USER LIST
.....
  *READY*
```

```
? installll-users : (ds102,dsd2) , (dm102,dmd2)
? (    ceo   ,
?              ceo1   $
  INSTALLLL-USERS : (DS102,DSD2) , (DM102,DMD2)

  *INVALID COMMAND KEY-WORD.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? install-users : (ds102,dsd2) , (dm102,dmd2)
             CEO1   $

  *')' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

?       CEO1 )$

  *READY*

? list-users$


.....................................
    LIST OF INSTALLED USERS
.....................................

    USER-NUM        USER-PW
    --------        -------

    CEO             CEO1

    DM102           DMD2

    DS102           DSD2

    D2GL32          GRL32

    D2GL22          GRL22

    MPA             MPA00

    PDA             PDA

    END OF USER LIST
.....................................
```

```
  *READY*

? list-activity$
  *ACTIVITY LIST EMPTY.

  *READY*

? list-activity$


.....................................
     LIST OF ACTIVE USERS
.....................................

     1.  DM102      DMD2

.....................................

  *READY*

? list-activity$


.....................................
     LIST OF ACTIVE USERS
.....................................

     1.  DM102      DMD2

     2.  MPA        MPA00

.....................................

  *READY*

? list-activity$


.....................................
     LIST OF ACTIVE USERS
.....................................

     1.  DM102      DMD2

     2.  DS102      DSD2

.....................................

  *READY*

?
```

```
? list-activity$

.....
     LIST OF ACTIVE USERS
.....

     1.  DM102     DMD2

.....

  *READY*

? out$

 LOG OFF TIME 18.44.47.

      .406 CP SECONDS EXECUTION TIME.
/endbms


 = CONCORDIA UNIVERSITY ======= ENDBMS =
 = DATE : 81/08/17.
 = TIME : 18.45.04.


  *READY*


? user : dm102 , dmd2 , i$
  *RECOVERY MUST BE CALLED-INFORM DBA.

 LOG OFF TIME  18.45.23.

      .208 CP SECONDS EXECUTION TIME.
```

```
= CONCORDIA UNIVERSITY ======= ENDBMS =
= DATE : 81/08/17.
= TIME : 18.46.29.

  *READY*

? user z01 z02 i$

  *READY*

? list-activity$

.....
     LIST OF ACTIVE USERS
.....

     1.  DM102     DMD2

.....

  *READY*

? recover : dm102 $

  *READY*

? user : dm102 , dmd2 , i $

  *READY*

? user : z01 z02 i $

  *READY*

? list-activity$
 *ACTIVITY LIST EMPTY.

  *READY*

?
```

```
 = CONCORDIA UNIVERSITY ======= ENDBMS =
 = DATE : 81/08/24.
 = TIME : 10.41.03.


  *READY*


? user : dm102 , dmd2 , i$
  *ENDBMS NOT AVAILABLE.

 LOG OFF  TIME 10.42.43.

      .227 CP SECONDS EXECUTION TIME.
/endbms


 = CONCORDIA UNIVERSITY ======= ENDBMS =
 = DATE : 81/08/24.
 = TIME : 10.43.19.


  *READY*


? user : z01 z02 i;
? unlock-system; ? out$

 LOG OFF  TIME 10.44.02.

      .607 CP SECONDS EXECUTION TIME.
/
```

```
 = CONCORDIA UNIVERSITY ======= ENDBMS =
 = DATE : 81/08/24.
 = TIME : 10.44.32.


  *READY*


? user : dm102 , dmd2 i$
  *TEMPORARILY SUSPENDED.

 LOG OFF TIME  10.45.28.

       .224 CP SECONDS EXECUTION TIME.
/endbms


 = CONCORDIA UNIVERSITY ======= ENDBMS =
 = DATE : 81/08/24.
 = TIME : 10.46.09.


  *READY*


? user : z01  z02 i;
? restore-access : dm102;
? out$

 LOG OFF TIME 10.47.04.

      .631 CP SECONDS EXECUTION TIME.
/
```

```
endbms


= CONCORDIA UNIVERSITY ====== ENDBMS =
= DATE : 81/08/18.
= TIME : 08.51.34.


  *READY*

? user , z01 , z02 , i$
  *READY*

? list-schema$
  *DATA BASE IS EMPTY.

  *READY*

? define-schema
? record,: orders
?    field: pno of char(3)
?    field: price of numeric (5)
?    field: quant of numeric (3)
?    field: del-date of char (8)
?    field: sno of charss (3)
? end
? record : supinfo
?    field: pno of char (3)
?    field: sno of char (3)
?    field: price of numeric (5)
?    field: lead-time of numeric (2)
? end
? record : parts
?    field: pno of char (3)
?    field: colour of char (6)
?    field: weight of numeric (3)
? end
? record : suppliers
?    field: sno of char (3)
?    field: city of char (10)
?    field: tel  of char (8)
? end
```

```
? record : sra
?     field: sno of char (3)
?     field: pno of char (3)
?     field: quant of numeric (3)
? end
? record : med-history
?     field: eno of char (3)
?     field: dno  of char (4)
?     field: diagnosis of char (15)
?     field: date of char (8)
? end
? record : sroups
?     field: sno of char (3)
?     field: dno of char (4)
?     field: leadrer of char (3)
?     field: budget  of char (5)
? end
?
? set : porder
?     owner : parts    member : orders
? end
? set : psinfo
?     owner : parts     member : supinfo
? end
? set : sorder
?     owner : suppliers  member : orderss
? end
? set : spinfo
?     owner : suppliers  member : supinfo
? end
? set : rparts
?     owner : sroups     member : sra
? end
? set : sparts
?     owner : parts      member : sra
? end
? $
    FIELD: SNO OF CHARSS (3)
                ^
  *'CHAR' EXPECTED.
  *'NUMERIC' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.
```

```
?     field: sno  of char (3)
      OWNER : PARTS      MEMBER : SUPINFO

  *'OWNER' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

?     owner: parts     member: supinfo
      OWNER : SUPPLIERS  MEMBER : ORDERSS

  *SCHEMA MEMBER TYPE,NOT DEFINED BEFORE.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

?     owner : suppliers   member : orders

  *READY*

? insert-srt
? record : employees
?    field: eno of char (3)
?    field: tel of char (8)
?    field: city of char (10)
?    field: salary of numericc (5)
?    field: dno of char (4)
?    field: sno of char (3)
?    field: hrs of char (5)
?    field: city of char (1)
? end;
? insert-srt
? record : departments
?    field: dno of char (4)
?    field: manager of cahr (3)
?    field: budget  of numeric (6)
?    field: perfo of char (1)
? end;
? insert-sst
? set : emh
?     owner : employees
?     member: med-history
? end;
? insert-sst
? set : group-emp
?     owner : groups
?     member: emplov
? end;
? insert-sst
? set : dep-group
?     owner : departments  member : groups
? end;
```

```
? insert-sst
? set : dep-emp$
?     owner : departments  member : employees
? end $
     FIELD: SALARY OF NUMERICO (5)
                           ^

  *'CHAR' EXPECTED.
  *'NUMERIC' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

?    field: salary of numeric (5)
     FIELD: CITY OF CHAR (1)
                 ^

  *DUPLICATE SCHEMA FIELD NAME.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

?     field: perfo of char (1)
     FIELD: MANAGER OF CAHR (3)
                         ^

  *'CHAR' EXPECTED.
  *'NUMERIC' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

?    field: manager of char (3)
      MEMBER: EMPLOY
                 ^

  *SCHEMA MEMBER TYPE,NOT DEFINED BEFORE.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

?    owner : groups   member : employees
  SET : DEP-EMP$
            ^

  *'OWNER' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? set : dep-emp

  *READY*

?
```

```
list-schema$

.....
  .SCHEMA  DEFINITION.
.....

   RECORD : DEPARTMENTS

        FIELD : DNO                    OF : CHAR.....( 4 ).

        FIELD : MANAGER                OF : CHAR.....( 3 ).

        FIELD : BUDGET                 OF : NUMERIC...( 6 ).

        FIELD : PERFO                  OF : CHAR.....( 1 ).

   END

   RECORD : EMPLOYEES

        FIELD : ENO                    OF : CHAR.....( 3 ).

        FIELD : TEL                    OF : CHAR.....( 8 ).

        FIELD : CITY                   OF : CHAR.....( 10 ).

        FIELD : SALARY                 OF : NUMERIC...( 5 ).

        FIELD : DNO                    OF : CHAR.....( 4 ).

        FIELD : GNO                    OF : CHAR.....( 3 ).

        FIELD : HRS                    OF : NUMERIC...( 5 ).

        FIELD : PERFO                  OF : CHAR.....( 1 ).

   END

   RECORD : GROUPS

        FIELD : GNO                    OF : CHAR.....( 3 ).

        FIELD : DNO                    OF : CHAR.....( 4 ).

        FIELD : LEADER                 OF : CHAR.....( 3 ).

        FIELD : BUDGET                 OF : NUMERIC...( 5 ).

   END
```

```
RECORD : MED-HISTORY
     FIELD : ENO              OF : CHAR.....( 3 ).
     FIELD : DNO              OF : CHAR.....( 4 ).
     FIELD : DIAGNOSIS        OF : CHAR.....( 15 ).
     FIELD : DATE             OF : CHAR.....( 8 ).
END
RECORD : GRQ
     FIELD : GNO              OF : CHAR.....( 3 ).
     FIELD : PNO              OF : CHAR.....( 3 ).
     FIELD : QUANT            OF : NUMERIC...( 3 ).
END
RECORD : SUPPLIERS
     FIELD : SNO              OF : CHAR.....( 3 ).
     FIELD : CITY             OF : CHAR.....( 10 ).
     FIELD : TEL              OF : CHAR.....( 8 ).
END
RECORD : PARTS
     FIELD : PNO              OF : CHAR.....( 3 ).
     FIELD : COLOUR           OF : CHAR.....( 6 ).
     FIELD : WEIGHT           OF : NUMERIC...( 3 ).
END
RECORD : SUPINFO
     FIELD : PNO              OF : CHAR.....( 3 ).
     FIELD : SNO              OF : CHAR.....( 3 ).
     FIELD : PRICE            OF : NUMERIC...
```

```
RECORD : ORDERS
     FIELD : PNO                    OF : CHAR.....( 3 ).
     FIELD : PRICE                  OF : NUMERIC...( 5 ).
     FIELD : QUANT                  OF : NUMERIC...( 3 ).
     FIELD : DEL-DATE               OF : CHAR.....( 8 ).
     FIELD : SNO                    OF : CHAR.....( 3 ).
END
..... SET TYPE DEFINITION(S) .....
SET : DEP-EMP
     OWNER : DEPARTMENTS
     MEMBER : EMPLOYEES
END
SET : DEP-GROUP
     OWNER : DEPARTMENTS
     MEMBER : GROUPS
END
SET : GROUP-EMP
     OWNER : GROUPS
     MEMBER : EMPLOYEES
END
SET : EMH
     OWNER : EMPLOYEES
     MEMBER : MED-HISTORY
END
SET : PORDER
     OWNER : PARTS
     MEMBER : ORDERS
END
```

```
SET : PSINFO
      OWNER : PARTS
      MEMBER : SUPINFO
END
SET : SORDER
      OWNER : SUPPLIERS
      MEMBER : ORDERS
END
SET : SPINFO
      OWNER : SUPPLIERS
      MEMBER : SUPINFO
END
SET : RPARTS
      OWNER : GROUPS
      MEMBER : GRQ
END
SET : GPARTS
      OWNER : PARTS
      MEMBER : GRQ
END
```

. . . . . .

```
*READY*
?
```

```
  lock for read  :   departments
?                    employees
?                    med-history
?                    groups
?                    grq
?                    parts
?                    suppliers
?                    orders
?                    supinfo   $

  *READY*

? list all of  departments$

  D100  E5   35000   C
  D101  E1   50000   A
  D102  E9   65000   E

  *READY*

? list all of departments
? in descending using budget$

  D102  E9   65000   E
  D101  E1   50000   A
  D100  E5   35000   C

  *READY*

? list all of groups$

  G32  D102  E17  2000
  G22  D102  E14  1800
  G12  D102  XXX  0000
  G21  D101  E21  1500
  G11  D101  E8   1200

  *READY*

? list all of groups
? in ascending using gno$

  G11  D101  E8   1200
  G12  D102  XXX  0000
  G21  D101  E21  1500
  G22  D102  E14  1800
  G32  D102  E17  2000

  *READY*

?
```

```
list all of employees$

E32   273-4576   OUTREMONT     16600   D102   G22   860     B
E29   935-6679   MONTREAL      17000   D102   G32   2450    C
E26   476-3388   LAVAL         17500   D102   G32   3400    B
E23   944-5566   ROSEMOUNT     18500   D102   G32   3400    A
E17   568-4577   LONGEIL       17800   D102   G32   2450    E
E14   877-3367   N.D.G         16700   D102   G22   850     A
E11   277-4466   T.M.R         17500   D102   X     1100    A
E9    931-3131   MONTREAL      15600   D102   X     15600   X
E50   933-5678   MONTREAL      15500   D101   G21   1350    D
E22   674-1235   LONGEIL       16700   D101   G21   450     B
E21   676-4455   BROSSARD      16400   D101   G21   3120    A
E15   476-3355   LAVAL         12800   D101   G11   2600    E
E8    272-4545   OUTREMONT     17000   D101   G11   3400    A
E3    788-9933   ROSEMOUNT     1650    D101   X     1250    B
E2    931-2235   MONTREAL      17000   D101   X     650     A
E1    872-2233   MONTREAL      18000   D101   X     3500    X
E40   935-3636   MONTREAL      18900   D100   X     1680    E
E36   272-7789   MONTREAL      19700   D100   X     12500   A
E35   277-4456   T.M.R         16500   D100   X     2450    A
E25   273-4466   OUTREMONT     3000    D100   X     0       C
E20   678-7788   ST-HUBERT     15000   D100   X     1200    A
E10   933-5555   MONTREL       18000   D100   X     800     E
E5    935-3344   MONTREAL      28000   D100   X     0       U

*READY*

? list all  of employees in ascending using eno $

E1    872-2233   MONTREAL      18000   D101   X     3500    X
E10   933-5555   MONTREL       18000   D100   X     800     E
E11   277-4466   T.M.R         17500   D102   X     1100    A
E14   877-3367   N.D.G         16700   D102   G22   850     A
E15   476-3355   LAVAL         12800   D101   G11   2600    E
E17   568-4577   LONGEIL       17800   D102   G32   2450    E
E2    931-2235   MONTREAL      17000   D101   X     650     A
E20   678-7788   ST-HUBERT     15000   D100   X     1200    A
E21   676-4455   BROSSARD      16400   D101   G21   3120    A
E22   674-1235   LONGEIL       16700   D101   G21   450     B
E23   944-5566   ROSEMOUNT     18500   D102   G32   3400    A
E25   273-4466   OUTREMONT     3000    D100   X     0       C
E26   476-3388   LAVAL         17500   D102   G32   3400    B
E29   935-6679   MONTREAL      17000   D102   G32   2450    C
E3    788-9933   ROSEMOUNT     1650    D101   X     1250    B
E32   273-4576   OUTREMONT     16600   D102   G22   860     B
E35   277-4456   T.M.R         16500   D100   X     2450    A
E36   272-7789   MONTREAL      19700   D100   X     12500   A
E40   935-3636   MONTREAL      18900   D100   X     1680    E
E5    935-3344   MONTREAL      28000   D100   X     0       U
E50   933-5678   MONTREAL      15500   D101   G21   1350    D
E8    272-4545   OUTREMONT     17000   D101   G11   3400    A
E9    931-3131   MONTREAL      15600   D102   X     15600   X

*READY*

?
```

```
list all of employees in descending using salary$

E5    935-3344   MONTREAL     28000   D100  X     0       U
E36   272-7789   MONTREAL     19700   D100  X     12500   A
E40   935-3636   MONTREAL     18900   D100  X     1680    E
E23   944-5566   ROSEMOUNT    18500   D102  G32   3400    A
E10   933-5555   MONTREL      18000   D100  X     800     E
E1    872-2233   MONTREAL     18000   D101  X     3500    X
E17   568-4577   LONGEIL      17900   D102  G32   2450    E
E11   277-4466   T.M.R        17500   D102  X     1100    A
E26   476-3388   LAVAL        17500   D102  G32   3400    B
E2    931-2235   MONTREAL     17000   D101  X     650     A
E29   935-6679   MONTREAL     17000   D102  G32   2450    C
E8    272-4545   OUTREMONT    17000   D101  G11   3400    A
E22   674-1235   LONGEIL      16700   D101  G21   450     B
E14   877-3367   N.D.G        16700   D102  G22   850     A
E32   273-4576   OUTREMONT    16600   D102  G22   860     B
E35   277-4456   T.M.R        16500   D100  X     2450    A
E21   676-4455   BROSSARD     16400   D101  G21   3120    A
E9    931-3131   MONTREAL     15600   D102  X     15600   X
E50   933-5678   MONTREAL     15500   D101  G21   1350    D
E20   678-7788   ST-HUBERT    15000   D100  X     1200    A
E15   476-3355   LAVAL        12800   D101  G11   2600    E
E25   273-4466   OUTREMONT    3000    D100  X     0       C
E3    788-9933   ROSEMOUNT    1650    D101  X     1250    B

*READY*

? list all of employees
? where perfo = "a"
? in ascending using eno $

E11   277-4466   T.M.R        17500   D102  X     1100    A
E14   877-3367   N.D.G        16700   D102  G22   850     A
E2    931-2235   MONTREAL     17000   D101  X     650     A
E20   678-7788   ST-HUBERT    15000   D100  X     1200    A
E21   676-4455   BROSSARD     16400   D101  G21   3120    A
E23   944-5566   ROSEMOUNT    18500   D102  G32   3400    A
E35   277-4456   T.M.R        16500   D100  X     2450    A
E36   272-7789   MONTREAL     19700   D100  X     12500   A
E8    272-4545   OUTREMONT    17000   D101  G11   3400    A

*READY*

?
```

```
list all of med-history$

E10   D100   COLD              20-3-80
E20   D100   HEART ATTACK      19-8-80
E15   D101   COLD              12-10-80
E15   D101   EYE OPERATION     10-7-80
E5    D100   BROKEN LEG        15-5-80
E5    D100   ALERGY            22-4-80

*READY*

? list all of parts$

P7    BROWN    70
P6    BLACK    25
P5    WHITE    50
P4    RED      100
P3    BLACK    20
P8    RED      65
P2    WHITE    80
P9    BROWN    25
P1    BLACK    55
P10   RED      40

*READY*

? list all of parts where weisht > "50" $

P7    BROWN    70
P4    RED      100
P8    RED      65
P2    WHITE    80
P1    BLACK    55

*READY*

? list all of suppliers$

S3    QUEBEC      375-7788
S4    QUEBEC      375-5599
S1    MONTREAL    273-8899
S2    MONTREAL    856-4433
S5    OTTAWA      345-6655

*READY*

?
```

```
list all of suppliers where city = "montreal" $

S1    MONTREAL    273-8899
S2    MONTREAL    856-4433

*READY*

? list all of supinfo$

P10   S3    700     2
P10   S2    680     2
P9    S4    400     5
P9    S2    350     4
P8    S5    1100    4
P7    S4    450     2
P6    S2    50      5
P5    S1    3500    10
P4    S5    600     3
P4    S1    650     3
P3    S4    500     5
P2    S5    1800    14
P2    S3    2000    10
P1    S2    500     25
P1    S1    1200    15

*READY*

? list all of orders$

P10   700     15    10-10-80   S2
P10   680     20    5-9-80     S3
P9    400     8     20-3-81    S4
P9    350     30    18-6-81    S2
P2    1800    10    12-5-82    S5
P7    450     20    25-9-80    S4
P2    2000    3     5-2-82     S3
P1    500     5     12-5-82    S2

*READY*

? list all of orders where pno = "P10"
? in ascending using price$

P10   680     20    5-9-80     S3
P10   700     15    10-10-80   S2

*READY*

?
```

```
  set first owner in dep-emp;
? list all members in dep-emp$

  E5    935-3344   MONTREAL     28000   D100   X     0      U
  E10   933-5555   MONTREL      18000   D100   X     800    E
  E20   678-7788   ST-HUBERT    15000   D100   X     1200   A
  E25   273-4466   OUTREMONT    3000    D100   X     0      C
  E35   277-4456   T.M.R        16500   D100   X     2450   A
  E36   272-7789   MONTREAL     19700   D100   X     12500  A
  E40   935-3636   MONTREAL     18900   D100   X     1680   E

  *READY*

? get next owner in dep-emp;
? list all  members in dep-emp$

  E1    872-2233   MONTREAL     18000   D101   X     3500   X
  E2    931-2235   MONTREAL     17000   D101   X     650    A
  E3    788-9933   ROSEMOUNT    1650    D101   X     1250   B
  E8    272-4545   OUTREMONT    17000   D101   G11   3400   A
  E15   476-3355   LAVAL        12800   D101   G11   2600   E
  E21   676-4455   BROSSARD     16400   D101   G21   3120   A
  E22   674-1235   LONGEIL      16700   D101   G21   450    B
  E50   933-5678   MONTREAL     15500   D101   G21   1350   D

  *READY*

? get next owner in dep-emp;
? list all members in dep-emp$

  E9    931-3131   MONTREAL     15600   D102   X     15600  X
  E11   277-4466   T.M.R        17500   D102   X     1100   A
  E14   877-3367   N.D.G        16700   D102   G22   850    A
  E17   568-4577   LONGEIL      17800   D102   G32   2450   E
  E23   944-5566   ROSEMOUNT    18500   D102   G32   3400   A
  E26   476-3388   LAVAL        17500   D102   G32   3400   B
  E29   935-6679   MONTREAL     17000   D102   G32   2450   C
  E32   273-4576   OUTREMONT    16600   D102   G22   860    B

  *READY*

?
```

```
   set first owner in group-emp;
? list all members in  group-emp$

   E17  568-4577  LONGEIL      17800  D102  G32  2450   E
   E23  944-5566  ROSEMOUNT    18500  D102  G32  3400   A
   E26  476-3388  LAVAL        17500  D102  G32  3400   B
   E29  435-6679  MONTREAL     17000  D102  G32  2450   C

   *READY*

? set next owner in group-emp;
? list all members in group-emp$

   E14  877-3367  N.D.G        16700  D102  G22  850    A
   E32  273-4576  OUTREMONT    16600  D102  G22  860    B

   *READY*

? set next owner in  group-emp;
? list all members in group-emp$

   *OWNER'S MEMBERSHIP LIST IS EMPTY.

   *READY*

? set next owner in group-emp;
? list all members in group-emp$

   E21  676-4455  BROSSARD    16400  D101  G21  3120   A
   E22  674-1235  LONGEIL     16700  D101  G21  450    B
   E50  933-5678  MONTREAL    15500  D101  G21  1350   D

   *READY*

? set next owner in group-emp ;
? list all members in group-emp$

   E8   272-4545  OUTREMONT   17000  D101  G11  3400   A
   E15  476-3355  LAVAL       12800  D101  G11  2600   E

   *READY*

?
```

```
  set first owner in spinfo;
? list all members in spinfo$

  P2    S3    2000    10
  P10   S3    700     2

  *READY*

? set next owner in spinfo;
? list all members in spinfo$

  P3    S4    500     5
  P7    S4    450     2
  P9    S4    400     5

  *READY*

? set next owner in spinfo;
? list all members in spinfo$

  P1    S1    1200    15
  P4    S1    650     3
  P5    S1    3500    10

  *READY*

? set next owner in spinfo;
? list all members in spinfo$

  P1    S2    500     25
  P6    S2    50      5
  P9    S2    350     4
  P10   S2    680     2

  *READY*

? set next owner in spinfo;
? list all mwmbers in spinfo$

  LIST ALL MWMBERS IN SPINFO$

  *'MEMBERS' EXPECTED.
  *'OF' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? list all members in spinfo

  P2    S5    1800    14
  P4    S5    600     3
  P8    S5    1100    4

  *READY*

?
```

```
   set first owner in sorder;
? list all members in sorder$

   P2   2000   3     5-2-82    S3
   P10  680    20    5-9-80    S3

   *READY*

? set next owner in sorder;
? list all members in sorder$

   P7   450    20    25-9-80   S4
   P9   400    8     20-3-81   S4

   *READY*

? set next owner in sorder;
? list all members in sorder$

   *OWNER'S MEMBERSHIP LIST IS EMPTY.

   *READY*

? set next owner in sorder ;
? list all members in sorder$

   P1   500    6     12-5-82   S2
   P10  700    15    10-10-80  S2

   *READY*

? set next owner in sorder;
? list all members in sorder$

   P2   1800   10    12-5-82   S5

   *READY*

? set next owner in sorder $

   *END OF RECORD TYPE FILE REACHED.

   *READY*

?
```

```
  set first owner in psinfo;
? list all members in psinfo$

  P7    S4    450     2

  *READY*

? set next owner in psinfo;
? list all members in psinfo$

  P6    S2    50      5

  *READY*

? set next owner inb psinfo;
? list all members in psinfo$

  GET NEXT OWNER INB PSINFO;

  *'IN' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? set next owner in psinfo
  P5    S1    3500    10

  *READY*

? set next owner in psinfo;
? list all members in psinfo$

  P4    S1    650     3
  P4    S5    600     3

  *READY*

? set next owner in psinfo;
? list all members in psinfo$

  P3    S4    500     5

  *READY*

? set next owner in psinfo;
? list all members in psinfo$

  P8    S5    1100    4

  *READY*

?
```

```
  set next owner in psinfo;
? list all members in psinfo$

  P2   S3   2000   10
  P2   S5   1800   14

  *READY*

? set next owner in psinfo;
? list all membersi in psinfo$

  LIST ALL MEMBERSI IN PSINFO$

  *'MEMBERS' EXPECTED.
  *'OF' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? list  all members in psinfo$

  P9   S2   350   4
  P9   S4   400   5

  *READY*

?


  set first owner in porder;
? list all members in porder$

  P7   450   20   25-9-80   S4

  *READY*

? set next owner in  porder;
? list all members in porder$

  *OWNER'S MEMBERSHIP LIST IS EMPTY.

  *READY*

? set next  owner in porder;
? list all members in porder$

  *OWNER'S MEMBERSHIP LIST IS EMPTY.

  *READY*

? set next owner in porder;
? list all members in porder$

  *OWNER'S MEMBERSHIP LIST IS EMPTY.

  *READY*

?
```

```
? set next owner in porder;
? list all members in porder$

*OWNER'S MEMBERSHIP LIST IS EMPTY.

*READY*

? set next owner in porder;
? list all members in porder$

*OWNER'S MEMBERSHIP LIST IS EMPTY.

*READY*

? set next owner in porder;
? list all members in porder$

P2   2000   3    5-2-82    S3
P2   1800   10   12-5-82   S5

*READY*

? set next owner  in porder;
? list all members in porder$

P9   350    30   18-6-81   S2
P9   400    8    20-3-81   S4

*READY*

? set next owner in porder;
? list all members in porder$

P1   500    5    12-5-82   S2

*READY*

? set next owner in porder;
? list all members in porder$

P10  680    20   5-9-80    S3
P10  700    15   10-10-80  S2

*READY*

?
```

```
? set first owner in rparts;
? list all members in rparts%

? $
  LIST ALL MEMBERS IN RPARTS%

  *'IN' EXPECTED.
  *'WHERE' EXPECTED.
  *';' OR '$' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? list all members in rparts

  G32  P7   2
  G32  P8   5
  G32  P9   3
  G32  P10  2

  *READY*

? set next owner in rparts;
? list all members in rparts$

  G22  P4   3
  G22  P5   5
  G22  P6   3
  G22  P8   1
  G22  P9   5
  G22  P10  3

  *READY*

? set next owner in rparts;
? list all members in rparts$

  *OWNER'S MEMBERSHIP LIST IS EMPTY.

  *READY*

? set next owner in rparts;
? list all members in rparts$

  G21  P1   4
  G21  P2   2
  G21  P10  5

  *READY*

?
```

```
  set next owner in rparts;
? list all members in rparts$

  G11   P1    2
  G11   P3    1
  G11   P5    2
  G11   P10   3

  *READY*

?
```

```
  set first owner in sparts;
? list all members in sparts$

  G32   P7    2

  *READY*

? set next owner in sparts;
? list all members in sparts$

  G22   P6    3

  *READY*

? set next owner in sparts;
? list all members in  sparts$

  G11   P5    2
  G22   P5    5

  *READY*

? set next owner in sparts;
? list all members in sparts$

  G22   P4    3

  *READY*

?
```

```
  set next owner in sparts;
? list all members in sparts$

  G11  P3   1

  *READY*

? set next owner in sparts;
? list all  members in sparts$

  G22  P8   1

  *READY*

? set next owner in sparts;
? list all members in sparts$

  G21  P2   2

  *READY*

? set next owner in sparts;
? list all members in sparts$

  G22  P9   5
  G32  P9   3

  *READY*

? set next owner in sparts;
? list all members in sparts$

  G11  P1   2
  G21  P1   4

  *READY*

? set next owner in sparts;
? list all members in  sparts$

  G11  P10  3
  G21  P10  5
  G22  P10  3
  G32  P10  2

  *READY*

? set next owner in sparts$
  *END OF RECORD TYPE FILE REACHED.

  *READY*

?
```

```
? define-profile : user: d102
? code : 25
? rfrr : perfor
? mfrr : budget
? rror : dnum <> '102'
? mror : nul
? drro : aaa$
  DEFINE-PROFILE : USER: D102

  *INVALID USER NUMBER.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? define-profile  user: dm102
  CODE : 25

  *'RECORD' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? record departm  code:25
  RECORD DEPARTM  CODE:25

  *INVALID SCHEMA RECORD-TYPE NAME.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? record : departments   code:25
  RECORD : DEPARTMENTS   CODE:25

  *INVALID ACCESS CODE.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? RECORD DEPARTMENTS   CODE: 9
  RFRR : PERFOR.

  *'RFR' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? rfr : perfor
  RFR : PERFOR

  *INVALID SCHEMA FIELD NAME.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.
?
```

```
    rfr : perfo
    MFRR : BUDGET

    *INVALID'SCHEMA FIELD NAME.
    *ENTER A NEW LINE -- OR TYPE 'DROP'.

  ? mfr : budget
    MFR : BUDGET

    *CONFLICT IN ACCESS CODE.
    *ENTER A NEW LINE -- OR TYPE 'DROP'.

  ? mfr : nil
    RORR : DNUM <> "102"

    *'ROR' EXPECTED.
    *ENTER A NEW LINE -- OR TYPE 'DROP'.

  ? ror : dnum <> "d102"
    ROR : DNUM <> "D102"

    *INVALID SCHEMA FIELD NAME.
    *ENTER A NEW LINE -- OR TYPE 'DROP'.

    MROR : NUL

    *'MOR' EXPECTED.
    *ENTER A NEW LINE -- OR TYPE 'DROP'.

  ? mor : nil
    DRRO : AAA$

    *'DOR' EXPECTED.
    *ENTER A NEW LINE -- OR TYPE 'DROP'.

  ? dor : aaa$
    DOR : AAA $

    *CONFLICT IN ACCESS CODE.
    *ENTER A NEW LINE -- OR TYPE 'DROP'.

  ? dor : nil

    *READY*

  ?
```

```
list-profile : dm102 $


.....................................
    PROFILE DEFINITION
    USER..  DM102
.....................................

    RECORD : DEPARTMENTS
    CODE   :  9
    RFR : PERFO
    MFR : NIL
    ROR : DNO                    <> D102
    MOR : NIL
    DOR : NIL

.....................................

 *READY*

? restore-access : dm102$

  *READY*

? remove-profile : dm102$
  REMOVE-PROFILE : DM102$

  *USER MUST BE SUSPENDED AND NOT ACTIVE.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? drop

  *READY*

? suspend-access : dm102;
? remove-profile : dm102;
? list-profile   : dm102$
  LIST-PROFILE   : DM102$

  *USER'S PROFILE IS NOT DEFINED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? drop

  *READY*

? restore-access : dm102$
```

```
  *READY*

?
? define-profile
? user : ds102
? record : employees
? code   : 10
? rfr : salary , perfo
? mfr : dno , sno , hrs
? ror : dno <> "d102"
? mor : nil
? dor : nil
?
? record: departments
? code: 9
? rfr : budget , perfo
? ror : dno <> "d102"
? mor : nil   dor : nil
?
? set : dep-emp
? code : 1 $

  *READY*

? define-profile  user: mra
? record : employees
? code : 9
? rfr : salary , dno , sno , hrs , perfo
? mfr : nil
? ror : nil  mor : nil  dor : nil
?
? record : med-history
? code : 2
? rfr:nil   mfr:nil
? ror:nil
? mor:nil
? dor:nil
?
? set : emh
? code : 2
$

  *READY*

?
```

```
  define-profile
? user : d2s132
?
? record : groups
? code : 1
? rfr:nil
? mfr:nil
? ror:nil
? mor:nil
? dor:nil
?
? record : employees
? code : 10
? rfr: salary , perfo
? mfr: eno , tel , city , dno , sno
? ror:  sno <> 's32'
? mor: nil
? dor: nil
? record : srq
? code : 2
? rfr: nil  mfr: nil
? ror:nil
? mor:nil
? dor:nil
?
? set : group-emp
? code : 1
? set : rparts
? code: 4
? set : sparts
? code: 4
$

  *READY*

?
```

```
  define-profile user: pda
? record: parts
? code:2
? rfr:nil   mfr:nil
? ror:nil   morz:nil  dor:nil
?
? record : sra
? code : 3
? rfr:nil
? mfr : sno , pno
? ror: nil  mor:nil;  dor: nil,
?
? record : groups
? code : 9
? rfr : budget    mfr:nil
? rorz:nil   mor:nil   dor:nil
?
? record : orders
? code : 2
? rfr:nil   mfr:nil
? ror:nil   mor:nil   dor:nil
?
? record : supinfo
? code : 8
? rfr:nil   mfr:nil
? ror:nil   mor:nil   dor:nil
? record : suppliers
?
? code : 5
? rfr:nil   mfr : sno
? ror:nil   mor:nil   dor:nil

? set : sparts
? code : 1
? set  : rparts
? code : 1
? set  : porder
? code : 4
? set  : psinfo
? code : 4
? set : soper
? code: 4
? set  : spinfo
? code : 4
$
  ROR:NIL   MORZ:NIL  DOR:NIL

  *'MOR' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.
```

```
? ror:nil   mor:nil   dor:nil
  ROR: NIL  MOR:NIL;  DOR: NIL

  *'DOR' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? ror:nil   mor:nil   dor:nil
  RORZ:NIL   MOR:NIL   DOR:NIL

  *'ROR' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? ror:nil   mor:nil   dor:nil
  SET : SOPER

  *INVALID SCHEMA SET-TYPE NAME.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? set : sorder

  *READY*

?
```

```
list profiles  :  dm102
?                 ds102
?                 d2s132
?                  mpa
?                   pda  $


****************************************
   PROFILE DEFINITION
   USER.. DM102
****************************************

     RECORD : GROUPS
     CODE   :  8
     RFR : NIL
     MFR : NIL
     ROR : DNO                    <> D102
     MOR : NIL
     DOR : NIL

     RECORD : MED-HISTORY
     CODE   :  1
     RFR : NIL
     MFR : NIL
     ROR  DNO                     <> D102
     MOR : NIL
     DOR : NIL

     RECORD : EMPLOYEES
     CODE   :  8
     RFR : NIL
     MFR : NIL
     ROR : DNO                    <> D102
     MOR  NIL
     DOR : NIL

     RECORD : DEPARTMENTS
     CODE   :  9
     RFR : PERFO
     MFR : NIL
     ROR : DNO                    <> D102
     MOR : NIL
     DOR : NIL

     SET    : DEP-GROUP
     CODE   :  4

     SET    : DEP-EMP
     CODE   :  4

     SET    : GROUP-EMP
     CODE   :  4

****************************************
```

```
*****************************************
PROFILE DEFINITION
USER.. DS102
*****************************************

RECORD : DEPARTMENTS
CODE   :  9
RFR : PERFO
      BUDGET
MFR : NIL
ROR : DNO                          <> D102
MOR : NIL
DOR : NIL

RECORD : EMPLOYEES
CODE   : 10
RFR : PERFO
      SALARY
MFR  HRS
      GNO
      DNO
ROR : DNO                          <> D102
MOR : NIL
DOR : NIL

SET    : DEP-EMP
CODE   :  1

*****************************************
```

```
.....................................
  PROFILE DEFINITION
  USER.. D2GL32
.....................................

  RECORD : GRO
  CODE   :  2
  RFR : NIL
  MFR : NIL
  ROR : NIL
  MOR  NIL
  DOR : NIL

  RECORD : PARTS
  CODE   :  1
  RFR : NIL
  MFR : NIL
  ROR : NIL
  MOR  NIL
  DOR : NIL

  RECORD : EMPLOYEES
  CODE   : 10
  RFR : PERFO
        SALARY
  MFR : GNO
        DNO
        CITY
        TEL
        ENO
  ROR  GNO                    <> G32
  MOR : NIL
  DOR : NIL

  RECORD : GROUPS
  CODE   :  1
  RFR : NIL
  MFR  NIL
  ROR : NIL
  MOR : NIL
  DOR : NIL

  SET    : GPARTS
  CODE   :  4

  SET    : RPARTS
  CODE   :  4

  SET    : GROUP-EMP
  CODE   :  1

.....................................
```

```
.....................................
  PROFILE DEFINITION
  USER.. MPA
.....................................

    RECORD : MED-HISTORY
    CODE   :  2
    RFR  NIL
    MFR : NIL
    ROR : NIL
    MOR : NIL
    DOR  NIL

    RECORD : EMPLOYEES
    CODE   :  9
    RFR  PERFO
         HRS
         GNO
         DNO
         SALARY
    MFR : NIL
    ROR : NIL
    MOR : NIL
    DOR  NIL

    SET    : EMH
    CODE   :  2

.....................................


.....................................
    PROFILE DEFINITION
    USER.. PDA
.....................................

    RECORD : SUPPLIERS
    CODE   :  5
    RFR : NIL
    MFR : SNO
    ROR : NIL
    MOR : NIL
    DOR : NIL

    RECORD : SUPINFO
    CODE   :  8
    RFR : NIL
    MFR : NIL
    ROR : NIL
    MOR : NIL
    DOR : NIL
```

```
RECORD : ORDERS
CODE   :  2
RFR : NIL
MFR : NIL
ROR : NIL
MOR : NIL
DOR : NIL

RECORD : GROUPS
CODE   :  9
RFR : BUDGET
MFR : NIL
ROR : NIL
MOR : NIL
DOR : NIL

RECORD : GRQ
CODE   :  3
RFR : NIL
MFR : GNO
      PNO
ROR : NIL
MOR : NIL
DOR : NIL

RECORD : PARTS
CODE   :  2
RFR : NIL
MFR  NIL
ROR : NIL
MOR : NIL
DOR : NIL

SET    : SPINFO
CODE   :  4

SET    : SORDER
CODE   :  4

SET    : PSINFO
CODE   :  4

SET    : PORDER
CODE   :  4

SET    : RPARTS
CODE   :  1

SET    : GPARTS
CODE   :  1
```

.....

READY

D.2 SAMPLE INTERFACE OF THE DM102-USER

This user is the manager of the department with symbolic name 'd102'. His shown subschema does not have to be exactly the view specified by his acess profile. It may be a subset of the view defined by his access profile. If the user wants to change or delete and redefine his subschema, he can do so by using the available ENDBMS subschema commands. After his subschema is defined he is using some DML commands to 'see' some of the view defined by his subschema. It can be seen clear from the results the enforcement of the 'occurrence' restrictions.

```
user , dm102 , dmd2 , i$

*READY*

? list-subschema$

.....................................................
    SUBSCHEMA DEFINITION
    USER  DM102
.....................................................

    RECORD: DEPARTMENTS
    VIRTUAL  WITH-BASE: DEPARTMENTS

         FIELD: DNAME
         REAL  FROM: DEPARTMENTS          .DNO

         FIELD: BUDGET
         REAL  FROM: DEPARTMENTS          .BUDGET

    END

    RECORD: EMPLOYEES
    VIRTUAL  WITH-BASE: EMPLOYEES

         FIELD: ENO
         REAL  FROM: EMPLOYEES            .ENO

         FIELD: DNO
         REAL  FROM: EMPLOYEES            .DNO

         FIELD: PERFO
         REAL  FROM: EMPLOYEES            .PERFO

    END

    RECORD: GROUPS
    REAL  WITH-BASE: GROUPS
         FIELD: GNAME
         FIELD: DNAME
         FIELD: LEADER
         FIELD: BUDGET
    END

    END OF SUBSCHEMA DEFINITION
.....................................................

*READY*

?
```

```
  lock for read  :  departm
?                   employees
?                   groups $
  LOCK FOR READ  :  DEPARTM

  *INVALID RECORD TYPE NAME.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? lock for  read  :  departments

  *READY*

? list all of departments$

  D102  65000

  *READY*

? list all of employees$

  E32   D102   B
  E29   D102   C
  E26   D102   B
  E23   D102   A
  E17   D102   E
  E14   D102   A
  E11   D102   A
  E9    D102   X

  *READY*

? list all of groups$

  G32   D102   E17   2000
  G22   D102   E14   1800
  G12   D102   XXX   0000

  *READY*

? list all of groups in ascending
? using gname$

  G12   D102   XXX   0000
  G22   D102   E14   1800
  G32   D102   E17   2000

  *READY*

?
```

```
  list all of groups
? in descending  using budset$

  G32  D102  E17  2000
  G22  D102  E14  1800
  G12  D102  XXX  0000

  *READY*

? list  all of emoloyees
? in ascending using perfo $.

  LIST  ALL OF EMOLOYEES

  *INVALID RECORD TYPE NAME.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? list all of employees

  E23  D102  A
  E14  D102  A
  E11  D102  A
  E26  D102  B
  E32  D102  B
  E29  D102  C
  E17  D102  E
  E9   D102  X

  *READY*

? list all  employees
? where  perfo = "a"
? in ascending using eno $

  LIST ALL  EMPLOYEES

  *'MEMBERS' EXPECTED.
  *'OF' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? list all of employees

  E11  D102  A
  E14  D102  A
  E23  D102  A

  *READY*

? out$

 LOG OFF TIME  21.51.48.

     3.116 CP SECONDS EXECUTION TIME.
```

D.3 SAMPLE INTERFACE OF THE DS102-USER.

This user is the secretary of the department with symbolic name 'd102'. She defines her subschema and uses some DML commands to 'see' the data she is authorized to. Interesting is the new employees record type file. The listing of all occurrences of this record type file shows the security enforcement at the occurrence level and the appearance of a 'compatible' field.

```
  user   , ds102 , dsd2 , i$

  *READY*

? define-subschema
? record : dep
? virtual with-base : departments
?    field: dname  real from  departments.dno
?    field: mname  real from  departments.manager
?    field:  budget  real from departments.budget
? end
? record  emp
? virtual  with-base : employees
? field: ename  real  from employees.eno
? field: dname  real from ° employees.dno
? field: sname   real from employees:sno
? field:  mname   real from depart.manager
? end
?
? set  : d-e
? real   with-base : dep-emppl
?      owner : dep
?      member  : emp
? end$

     FIELD:  BUDGET  REAL FROM DEPARTMENTS.BUDGET

  *CONFLICT WITH READ FIELD RESTRICTIONS.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? :
  FIELD:  MNAME   REAL FROM DEPART.MANAGER

  *INVALID COMPATIBLE R-NAME IN F-DEFINTION.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? field: man#
  FIELD: MAN#

  *'REAL' EXPECTED.
  *'VIRTUAL' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? field: mname  real from  departments.manager


  REAL   WITH-BASE : DEP-EMPPL

  *SET  BASE-NAME OUT OF USER'S PROFILE.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? real   with-base  : dep-emp

  *READY*

?
```

```
list-subschema$

**************************************************
    SUBSCHEMA DEFINITION
    USER DS102
**************************************************

    RECORD: EMP
    VIRTUAL  WITH-BASE: EMPLOYEES

         FIELD: ENAME
         REAL   FROM: EMPLOYEES          .ENO

         FIELD: DNAME
         REAL   FROM: EMPLOYEES          .DNO

         FIELD: GNAME
         REAL   FROM: EMPLOYEES          .GNO

         FIELD: MNAME
         REAL   FROM: DEPARTMENTS        .MANAGER

    END

    RECORD: DEP
    VIRTUAL  WITH-BASE: DEPARTMENTS

         FIELD: DNAME
         REAL   FROM: DEPARTMENTS        .DNO

         FIELD: MNAME
         REAL   FROM: DEPARTMENTS        .MANAGER

    END

    SET: D-E
    REAL  WITH-BASE: DEP-EMP
         OWNER: DEP
         MEMBER: EMP
    END

    END OF SUBSCHEMA DEFINITION
**************************************************

 *READY*

?
```

```
lock for read  dep , emp $

LOCK FOR READ  DEP , EMP $

*INVALID RECORD TYPE NAME.
*ENTER A NEW LINE -- OR TYPE 'DROP'.

? out$
OUT$
   ^

*LABEL OR
*'LOCK' EXPECTED.
*ENTER A NEW LINE -- OR TYPE 'DROP'.

? drop

*READY*

? user  ds102 , dsd2 , i$

*READY*

? lock for read  dep  ,  emp $

*READY*

? list all of dep $

D102  E9

*READY*

? list all of emp$

E32   D102   G22   E9
E29   D102   G32   E9
E26   D102   G32   E9
E23   D102   G32   E9
E17   D102   G32   E9
E14   D102   G22   E9
E11   D102   X     E9
E9    D102   X     E9

*READY*

? list current of  dep$
*'ROR' SECURITY ENFORCED.

*READY*

? set next of dep where dname='d102';
? list current of dep$

D102  E9

*READY*
```

```
  set  oci in d-e to ci of dep$

EXECUTE
stop
 *TERMINATED*
/rt,lf1
/endbms


 = CONCORDIA UNIVERSITY ====== ENDBMS =
 = DATE :  81/08/24.
 = TIME :  19.21.23.


  *READY*

? user z01 z02 i;
? recover ds102;
? out$

 LOG OFF TIME  19.22.16.

      .977 CP SECONDS EXECUTION TIME.
/endbms


 = CONCORDIA UNIVERSITY ====== ENDBMS =
 = DATE :  81/08/24.
 = TIME :  19.22.49.


  *READY*

? user ds102 ,dsd2 ,i$

  *READY*

? out$

 LOG OFF TIME  19.23.16.

      .352 CP SECONDS EXECUTION TIME.
/
```

D.4 SAMPLE INTERFACE OF THE PDA-USER.

This user is the director of the purchasing department. First he defines his subschema which includes only the parts and suppliers record type files and their 'supplies' and 'supplied-by' associations. The used DML commands show how well information can be retieved based on these record type files and their implemented logical associations. More complicated queries can be defined using the DML 'if' and 'goto' commands.

```
user , pda , pda00 i$

 *READY*

? define-subschema
? record : supl
? virtual  with-base  : suppliers
?       fiekd: sname   real from suppliers . sno
?       field:  city    real   from suplier  . city
? end
? record  : perinfo
? real  with-base : supinfo
?       field : pname
?       field : sname
?       field : price
?       field : lead-time
? end
? record parts
? real  with-base : parts
?      field  : pname
?      field  : colour
?      field  : weight
? end
? set  : part-perinfo
? real  with-base  : psinfo
?       owner  : parts
?       member : perinfo
? end
? set  : sup-perinfo
? real   with-base    : info
?     owner: supl
?     member:   perinfo
? end
? $
          FIEKD: SNAME   REAL FROM SUPPLIERS . SNO

  *'FIELD' EXPECTED.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

?    field: sname  real  from  suppliers . sno
        FIELD:   CITY    REAL   FROM SUPLIER  . CITY

  *INVALID COMPATIBLE R-NAME IN F-DEFINTION.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

?    field: city  real from  suppliers. city
  REAL   WITH-BASE   : INFO

  *SET BASE-NAME OUT OF USER'S PROFILE.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? real with-base  : spinfo

  *READY*

?
```

```
 stop
 *TERMINATED*
/rt,lf1
/endbms


 = CONCORDIA UNIVERSITY ====== ENDBMS =
 = DATE :  81/08/24.
 = TIME :  21.28.36.


  *READY*


? user  pda  , pda00 , i$
  *RECOVERY MUST BE CALLED-INFORM DBA.

 LOG OFF TIME  21.28.53.

      .222 CP SECONDS EXECUTION TIME.
/endbms


 = CONCORDIA UNIVERSITY ====== ENDBMS =
 = DATE :  81/08/24.
 = TIME :  21.29.14.


  *READY*


? user , z01 , z02 i$

  *READY*

? recover : pda$

  *READY*

? user pda , pda00 , i$

  *READY*

? out$

 LOG OFF TIME  21.30.30.

    1.183 CP SECONDS EXECUTION TIME.
/
```

```
list-subschema$

.....................................................
    SUBSCHEMA DEFINITION
    USER  PDA
.....................................................


    RECORD: PARTS
    REAL  WITH-BASE: PARTS
          FIELD: PNAME
          FIELD: COLOUR
          FIELD: WEIGHT
    END

    RECORD: PERINFO
    REAL  WITH-BASE: SUPINFO
          FIELD: PNAME
          FIELD: SNAME
          FIELD: PRICE
          FIELD: LEAD-TIME
    END

    RECORD: SUPL
    VIRTUAL  WITH-BASE: SUPPLIERS

          FIELD: SNAME
          REAL  FROM: SUPPLIERS          .SNO

          FIELD: CITY
          REAL  FROM: SUPPLIERS          .CITY

    END

    SET: SUP-PERINFO
    REAL  WITH-BASE: SPINFO
          OWNER: SUPL
          MEMBER: PERINFO
    END

    SET: PART-PERINFO
    REAL  WITH-BASE: PSINFO
          OWNER: PARTS
          MEMBER: PERINFO
    END


    END OF SUBSCHEMA DEFINITION
.....................................................

  *READY*
```

```
  list all of  parts$

  LIST ALL OF  PARTS$

  *RECORD-TYPE FILE MUST BE LOCKED PROPERLY.

  *READY*

? lock for read :   parts
?                   supl
? lock for delete : perinfo
? $
  LOCK FOR DELETE : PERINFO

  *ATTEMPT TO VIOLATE THE GIVEN ACCESS RIGHTS.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? lock for read  : perinfo

  *READY*

? list all of  parts$

  P7    BROWN    70
  P6    BLACK    25
  P5    WHITE    50
  P4    RED      100
  P3    BLACK    20
  P8    RED      65
  P2    WHITE    80
  P9    BROWN    25
  P1    BLACK    55
  P10   RED      40

  *READY*

? list all of supl$

  S3    QUEBEC
  S4    QUEBEC
  S1    MONTREAL
  S2    MONTREAL
  S5    OTTAWA

  *READY*
?
```

```
  list  all of parts
? in ascending  using pname$

  P1    BLACK    55
  P10   RED      40
  P2    WHITE    80
  P3    BLACK    20
  P4    RED      100
  P5    WHITE    50
  P6    BLACK    25
  P7    BROWN    70
  P8    RED      65
  P9    BROWN    25

  *READY*

? list all of parts
? where  colour = 'black'
? in  descending  using   pname$

  P6    BLACK    25
  P3    BLACK    20
  P1    BLACK    55

  *READY*

? list all of parts
? where   colour = 'red'
? in ascending  using pname$

  P10   RED      40
  P4    RED      100
  P8    RED      65

  *READY*

? list all of supl
? where .  city = 'montreal' $

  S1    MONTREAL
  S2    MONTREAL

  *READY*

?
```

```
list all of perinfo $

P10   S3   700    2
P10   S2   680    2
P9    S4   400    5
P9    S2   350    4
P8    S5   1100   4
P7    S4   450    2
P6    S2   50     5
P5    S1   3500   10
P4    S5   600    3
P4    S1   650    3
P3    S4   500    5
P2    S5   1800   14
P2    S3   2000   10
P1    S2   500    25
P1    S1   1200   15

*READY*

? list all of perinfo
? in ascendins  usins  price$

P6    S2   50     5
P9    S2   350    4
P9    S4   400    5
P7    S4   450    2
P3    S4   500    5
P1    S2   500    25
P4    S5   600    3
P4    S1   650    3
P10   S2   680    2
P10   S3   700    2
P8    S5   1100   4
P1    S1   1200   15
P2    S5   1800   14
P2    S3   2000   10
P5    S1   3500   10

*READY*

? list all of perinfo  where sname='s2'$

P10   S2   680    2
P9    S2   350    4
P6    S2   50     5
P1    S2   500    25

*READY*

?
```

```
  set next of parts where pname="P10";
? list current of parts$

  P10  RED     40

  *READY*

? set oci in part-perinfo to ci of parts;
? list all members in  perinfo$

  LIST ALL MEMBERS IN  PERINFO$

  *INVALID SET TYPE NAME.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? list all members in part-perinfo$

  P10  S2   680    2
  P10  S3   700    2

  *READY*

? set next of parts where  pname="P2";
? set oci in part-perinfo to ci of parts;
? list all members in part-perinfo$

  *END OF RECORD TYPE FILE REACHED.
  P10  S2   680    2
  P10  S3   700    2

  *READY*

? set first of parts;
? set next of parts where  pname="P2";
? set oci in part-perinfo to ci of parts;
? list all members in  part-perinfo$

  P2   S3   2000   10
  P2   S5   1800   14

  *READY*

? out$

 LOG OFF TIME  20.16.41.

     1.214 CP SECONDS EXECUTION TIME.
```

```
  set first owner in  sup-perinfo;
? list  all of sup1$

  S3   QUEBEC
  S4   QUEBEC
  S1   MONTREAL
  S2   MONTREAL
  S5   OTTAWA

  *READY*

? set oci in sup-perinfo to ci of sup;
? list all members in sup-prinfo$

  SET OCI IN SUP-PERINFO TO CI OF SUP;

  *INVALID RECORD TYPE NAME.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? set oci in sup-perinfo to ci of sup1
  LIST ALL MEMBERS IN SUP-PRINFO$

  *INVALID SET TYPE NAME.
  *ENTER A NEW LINE -- OR TYPE 'DROP'.

? list all members  in sup-perinfo$

  P2   S3   2000   10
  P10  S3   700    2

  *READY*

? set next owner in  sup-perinfo;
? list all members in  sup-perinfo$

  P3   S4   500    5
  P7   S4   450    2
  P9   S4   400    5

  *READY*

? set next owner in sup-perinfo;
? list all members in  sup-perinfo$

  P1   S1   1200   15
  P4   S1   650    3
  P5   S1   3500   10

  *READY*

? out$

 LOG OFF TIME  21.38.13.
```