Job. No. 3746
Div. & Sec. 640.02
No. Copies. · 303

NBSIR 74-500

The CODASYL Data Base Approach: A COBOL Example of Design and Use of a Personnel File

Edgar H. Sibley

System and Software Division
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D. C. 20234

February 1974

Final

U. S. DEPARTMENT OF COMMERCE
NATIONAL BUREAU OF STANDARDS

NBSIR 74-500

THE CODASYL DATA BASE APPROACH: A COBOL EXAMPLE OF DESIGN AND USE OF A PERSONNEL FILE

Edgar H. Sibley

System and Software Division
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D. C. 20234

February 1974

Final

Foreword

On April 30 and May 1, 1973, the Institute for Computer Sciences and Technology presented a tutorial symposium on "Data Base Management: The CODASYL Approach" for the benefit of Government agencies, national standards groups, and other interested professionals in the data processing field. The speakers were Dr. E.H. Sibley, now of the University of Maryland, and Mr. M.L. O'Connell of Rockwell International. This report reflects the essential content of Dr. Sibley's presentations.

TABLE OF CONTENTS

THE CODASYL DATA BASE APPROACH: A COBOL EXAMPLE OF DESIGN AND USE OF A PERSONNEL FILE

Edgar H. Sibley

This report introduces examples of the use of the proposed CODASYL Data Description Language and Data Base Language extensions to COBOL. It does this by suggesting the needs and data base elements which can be expected for a set of simple personnel applications. The discussion of the data definitions centers around the decisions that the data administrator makes, and the tools that are provided for him. Then it discusses a few of the processes (programs) which are required by typical personnel departments, and shows their implementation (in outline) in three COBOL programs. The reader is expected to have some knowledge of the CODASYL specifications.

Keywords: Data Base; COBOL; Data Definition; Data Structure Applications; CODASYL; Data Definition Language; Data Description.

1. INTRODUCTION

This technical report is intended to serve three principal purposes:

- to illustrate the thought process of a data base administrator in setting up the data base schema, when provided with a powerful data definition language;
- to show the power of the recommended additions to COBOL for dealing with data bases;
- to show the interrelationship between the schema, subschema and data manipulation verbs, in order to illustrate how the

complexity of the structure and manipulation verbs aid in simplifying the programmer's task, while restricting him to perform according to the dictates of the data administrator.

The illustration deals with the partial design of a personnel system which may be said to be rather simple, but this data base and procedures should be understandable by a wide audience. The three example programs are not intended to be exhaustive, but illustrative. They are not complete, in that there is no intention of putting in all of the checks that would normally be made against abnormal input errors, but rather the skeleton, or that portion of the program that best illustrates the data base concept.

The syntax and semantics of the data base example described in this report are based on two recent documents. The first of these discusses the data description language, and will be published* by the U.S. National Bureau of Standards for the CODASYL DDL Committee. From this point, all reference to this Journal of Development will be abbreviated to JOD-DDL. The second document is the CODASYL COBOL DATA BASE FACILITY PROPOSAL, dated March 1973, published by the Department of Supply and Services of the Canadian Government for the Data Base Language Task Group of the Programming Language Committee of CODASYL. From this point, references to this document will be abbreviated to DBFP.

The following paragraph is inserted to cover references to these documents. Such a statement is requested of all quotations of these CODASYL documents:

* NBS Handbook 113, issued January 1974

The reader is reminded that the document he is about to read neither reflects COBOL nor Data Description Language specifications nor any implied support by members of the Committees or their sponsors, but merely that it is being offered for possible consideration. The information has been made available to establish better communication. Anyone reproducing parts of this document is requested to include such an introduction, where meaningful.

2. DESIGN OF THE DATA BASE AND DEFINITION OF THE SCHEMA

In order to understand much of the work discussed in this document, it is very important to realize that the concept of modern data base design requires some authority in the enterprise which makes decisions involving the total needs of all users of the data base. Such an authority has been given either the name "Data Administrator" or "Data Manager". Because "Data Manager" is a term often associated with operating systems and their access methods, we shall use the former term. Thus the data administrator has the ultimate authority over the design and implementation of the data base.

In their work on a language for data definition, the CODASYL DDLC has provided features which, when implemented, will provide potential for real control. This section deals with the design of a data base and its definition using the JOD-DDL techniques. The emphasis will be on the decisions made by the data administrator as well as the definition of the data base, which could not, of course, be provided without the requisite tools.

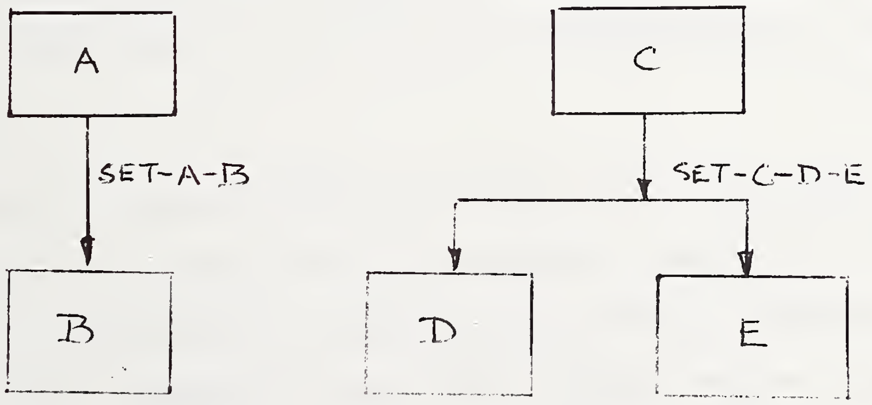
As was suggested in the introduction, the data base will be relatively simple, in that it will be abbreviated; i.e., not contain all information of the sophisticated personnel file. Conceptually, the enterprise about to embark on data base techniques has decided to incorporate most of the information which it previously collected by hand. This involves some essentially static information involving the nucleus of a person's information: his name, identification number, date of birth, present salary, etc. During the development of all previous uses of the personnel files, the enterprise has determined that some educational information is necessary, but that it is seldom worth keeping a large amount. Consequently, the information that will be retained will consist solely of the final degree or diploma, dates during which the employee was taking this educational course, and the location. Furthermore, it has been determined that the most recent three degrees or diplomas are all that will be stored.

The JOD-DDL provides two ways of retaining repeated information. The first of these is by using the SET technique, the other is by use of the concept of a REPEATING GROUP, and while it is assumed that the reader will augment this document by referring to the JOD-DDL and its discussion of concepts, a few basic ideas will be introduced here.

The COBOL file description (FD) statements allow a data description entry to OCCUR several times. This has led to the concept of a <u>vector</u> (a data description entry, or elementary item, which repeats) or of a <u>repeating group</u> (a collection of elementary items, defined at a lower level in the structure, which all repeat).

In this illustration the educational information will be retained as a repeating group within the (principal) information about an employee. The basic idea behind the data administrator's decision is that a fixed, small number of repetitions will be entered for each employee. In fact, even if all educational information were to be retained about each employee, and even if this was to be implemented by a variable number of repetitions, it could still be a reasonable decision.

The second way of storing repeated information is by using the SET concept. In this the record is broken into parts, and each one separately named. The fixed record is termed the OWNER, the variable record or records is termed the MEMBER. Thus a set type consists of one owner record type and one or more member record types. To illustrate the set, records are shown as named boxes, and an arrow goes from the owner to its member(s). The set is named, and this name may be affixed to the arrow as shown:



In this illustration we have a group of information about the medical condition of the personnel. This consists of the history of all of his illnesses, absence dates, and a free format note page for use by the medical department. Conceptually, the records will be used both by the medical department and for any legal or human engineering aspects associated with the illnesses of the person. As will be seen in one of the examples, it is assumed that certain diseases may be caused by certain types of working conditions; even when the employee is retired, or leaves the company for other cause, it may still be necessary to retain this information. An example shows the retention of medical history of certain diseases even after the employee is retired, and his personal information is no longer retained within the working data base. The final group of information in this data base is the job history. It consists of the particular job classification, dates at which the person started and completed his assignments, and the performance rating he received during this time.

Because the medical and job records are likely to be extremely volatile, with an unknown number of each associated with a particular person, they will each be tied to the person records by separate "sets," so that the entire employee base may be represented as shown in figure 1.

Because there is a need for the MEDICAL records after the PERSON records are deleted, the example uses the "singular set" concept. This is a set where the SYSTEM is considered to be the owner, and the member record or records (MEDICAL in this case) is therefore the only real component of the (RETIRED-EMPLOYEE-MEDICAL), as also illustrated in figure 1.

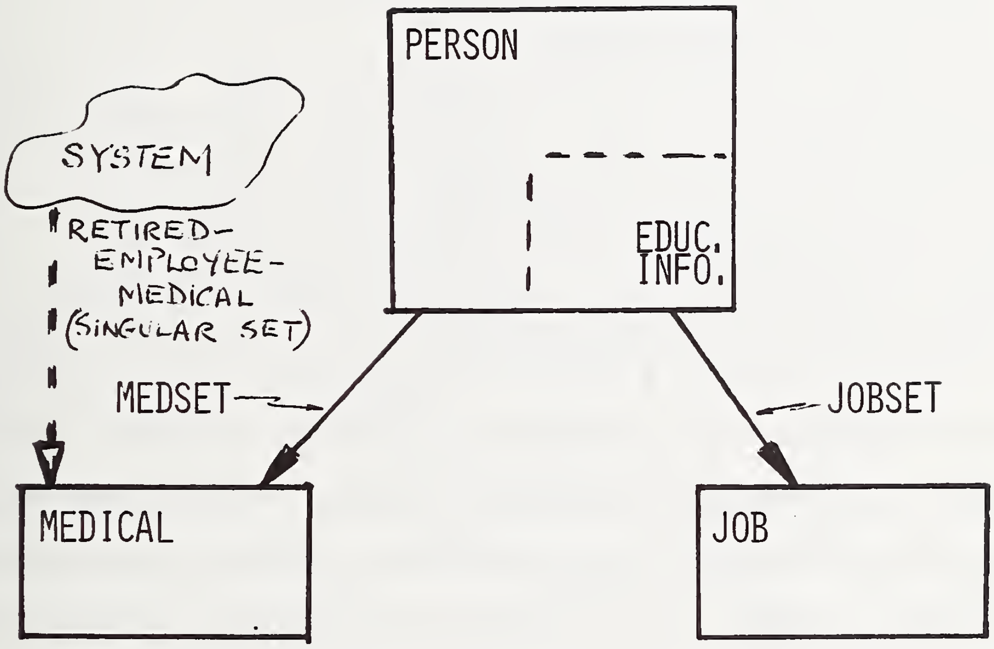


Figure 1 The Schema for the Employee Base

In defining the schema, certain locks and "ON conditions" may be associated with the definition. This is illustrated by the set of code:

```
SCHEMA NAME IS EMPLOYEE-BASE;
PRIVACY LOCK FOR LOCKS IS PROCEDURE E-LOCK;
PRIVACY LOCK FOR COPY IS COMP-LOCK;
PRIVACY LOCK FOR ALTER IS EMP-ALT-LOCK;
PRIVACY LOCK FOR DISPLAY 'SEE EMPLOYEE BASE';
ON ALTER CALL SNOOPER-ROUTINE.
```

This is intended to convey first of all the name of the schema to be EMPLOYEE-BASE, then provide four different locks. The first lock is to allow the data administrator to set those other locks for using, changing or viewing the schema. This may be looked on, therefore, as the lock on the cupboard containing all of the rest of the locks. The lock is to be set, and consequently opened, by a procedure known as E-LOCK. As discussed in the CODASYL document, such a procedure may initiate a communication between a particular terminal in the data administrator's office, allowing keys to be entered in an interactive mode, or initiate any other interrogative routines. Naturally, the procedure is written and inserted by the data base administrator himself, or his staff.

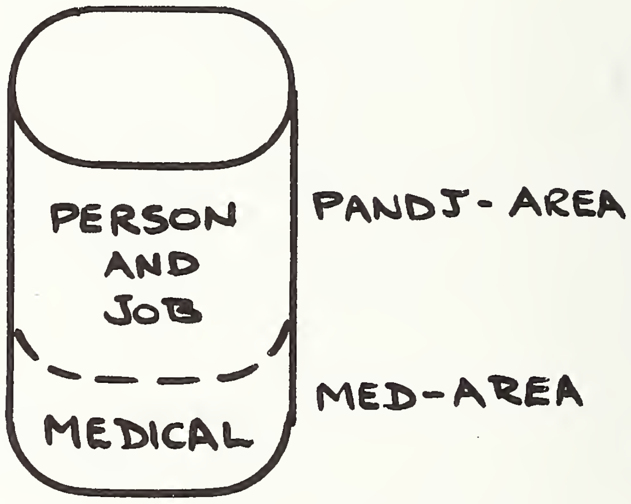
The second lock is on the ability to "copy" (i.e., it is a lock which must be satisfied in order for this schema to be copied, a process which typically occurs when the subschema references the schema); the copy lock is given as a location: COMP-LOCK, which presumably is within the data base management system, and contains a key-value which must be satisfied in order to be able to copy the schema. Naturally one of the operations which must therefore be performed by the data base administrator is to set (by putting a value into) this lock, which presumably he achieves by calling the procedure E-LOCK.

The next lock is for changing the schema, it also consists of a key stored in the location termed EMP-ALT-LOCK. Next there is a lock for displaying the schema, which is illustrated as the literal "SEE EMPLOYEE BASE". Finally the schema illustrates how to invoke a data base procedure by using an ON clause. Whenever an attempt is made to alter the schema a procedure called SNOOPER-ROUTINE is invoked. Such a routine could immediately advise the data base administrator staff when an attempt is being made to change the schema, thereby making them aware of potential infiltration.

The next decision that must be made is one of defining which areas or realms are needed for the data base. Figure 2 illustrates the concept. Areas may be looked on as logical subsetting of the data base. They may also, by their mapping in the data base management system (for example utilizing a device media control language, DMCL) be used for physical partitioning; e.g., by assigning one of the areas to high-speed, and another to low-speed second storage devices. We should consider however that all the partitioning is purely logical in concept.

We come to a logical partitioning in the following way: it will be assumed that when we wish to reference the person, we will generally also be interested in his job history. However, it is less likely that

ASSUME TWO AREAS, ONE OF WHICH HOLDS MEDICAL INFORMATION, THE OTHER CONTAINS THE REST:



THERE ARE NO SECURITY RESTRICTIONS ON EITHER AREA, EXCEPT FOR UPDATE OF MEDICAL FILES.

Figure 2 Realms or Areas

we shall be dealing with medical records on regular basis, except by regular search within the medical department. We shall, therefore, partition the base into two parts, where person and job records go into one area which will be called the PANDJ-AREA, whereas the medical records will go into a different area, called MED-AREA.

At this time, privacy locks and ON conditions may be defined for each of these areas. In this illustration, no security or ON condition will be enforced on the PANDJ-AREA, but only certain people will be allowed to update the medical records. Such a condition can be achieved by using an ON condition for UPDATE of MED-AREA; application of this condition will invoke a procedure called MED-DEPT. As is shown later, in the examples which require to update the medical area, this may be satisfied by providing a key within the DECLARATIVE section of the procedure division of the COBOL program. Thus the definition of the areas is:

```
              AREA NAME IS PANDJ-AREA.
              AREA NAME IS MED-AREA;
              PRIVACY LOCK FOR UPDATE IS PROCEDURE MED-DEPT.
```

The next consideration in the data base design is that of "file location." This term is used in the sense of <u>method of storing</u> particular records from the data base. Apart from the implementor defined option: LOCATION MODE IS SYSTEM, there are three principle ways: DIRECT, CALC, and VIA. The logical or physical address of a record on storage is termed its "data base ksy". Thus the data base key in some implementations may (though this is <u>not</u> recommended) be the physical address at which the occurrence of the record is stored. In a more sophisticated implementation, the data base key would probably ba a logical address (i.e. a means whereby the physical address can be deduced, such as a conversion table). The DIRECT mode of locating a record uses a data base key; this implies previous storage of this information. The other two methods (CALC and VIA) will also be used in this example:

When endeavoring to find a person's record, it is supposed that the personnel department of the enterprise has assigned a unique number known as IDENTIFICATION-NUM. No person in the organization may be given the same identification number as another. It is therefore possible to use an address calculation method to locate (i.e., position for storage or retrieval of the record) a particular record; i.e., using hashing or content addressability techniques. Now it would be possible to utilize the system hashing algorithm, but we have decided to supply a procedure (which is presumably more efficient than that of the system for this particular expected set of keys): this procedure will be called EMP-HASH. Because we know that the personnel department should not be issuing a duplicate identification number, we shall use a DUPLICATES ARE NOT ALLOWED clause.

The location mode of the other records will be discussed as such time as they are defined:

The person record definition starts off in the following fashion:

```
    RECORD NAME IS PERSON;

    LOCATION MODE IS CALC EMP-HASH USING IDENTIFICATION-NUM
      DUPLICATES ARE NOT ALLOWED;

    WITHIN PANDJ-AREA;
    ON DELETE CALL MICROFILM-RECORDER;
    PRIVACY LOCK FOR DELETE IS PROCEDURE EMP-LEFT.
```

This shows that the record, which has been named PERSON, is placed in the previously defined PANDJ-AREA area.

As with other portions of the definition, it is possible to use privacy and ON conditions for records; it has been decided that it would be useful to insure that there is no total loss of information about the person. For this reason, an ON clause is used to insure that deletion of the record invokes a procedure termed MICROFILM-RECORDER. This procedure may be assumed to be a method whereby a duplicate copy is made of the record. The name is intended to convey the idea that this procedure may provide permanent records on some low cost storage device, such as microfilm. Naturally, however, the effectiveness of such an ON-call depends on the sophistication of the data base procedure called MICROFILM-RECORDER.

The example also illustrates the use of a privacy lock on a particular operation. The privacy lock for DELETE ensures that only those programs which satisfy the procedure termed EMP-LEFT will be able to invoke the deletion verb. As suggested previously, the satisfaction of this procedure is one of executing the DECLARATIVE paragraphs within the beginning of the COBOL procedure section. This concept is illustrated in two of the programs shown later.

The information which is to be contained in the PERSON RECORD is now given. For purposes of illustration, a large number of different data sub-entries are given in this example. The data sub-entries within the person records are:

```
     NAME; PICTURE IS "A(20)".

     IDENTIFICATION-NUM; PICTURE IS "9(6)".

     DATE-OF-BIRTH; PICTURE IS "99X99X99"; CHECK IS PICTURE.

  1  AGE; PICTURE "99V9"; IS VIRTUAL RESULT OF AGE-CALC.

  1  SALARY; TYPE IS FIXED 7.2; CHECK IS VALUE 8000.00
       THRU 45000.00.

  1  EDUCATION-INFO; OCCURS 3 TIMES.

  2  DEGREE; PICTURE "AA".

  2  START-DATE; PICTURE "99X99X99".

  2  COMPLETION-DATE; PICTURE "99X99X99"; CHECK NOT-BEFORE.

  2  DEGREE-RECEIVED-FROM; PIC "A(20)".
```

The first point of importance is in the use (and absence) of "level numbers". If no level number is given, it is automatically implied to be "level number one". Thus NAME, IDENTIFICATION-NUM, and DATE-OF-BIRTH are all at level one.

COBOL users will immediately notice a difference between a COBOL entry (where level one implies the record name, and all subordinate data items are at level two or more). Otherwise, the defintion is quite similar to that of a COBOL entry. There are five elementary items at level 1, a repeating group (EDUCATION-INFO) at level 1, and the four elements (at level 2) which comprise the repeating group EDUCATION-INFO. Moreover, the group repeats exactly three times.

Now an item definition, apart from its level number and name, must contain either a PICTURE or TYPE, or SOURCE clause. The SOURCE clause is illustrated in the MEDICAL record and will not be discussed at this time. The PICTURE and TYPE clauses are intended to represent the physical character representation or implementor defined internal representation respectively. Thus the PICTURE clause defines the exact storage representation of each of the elementary data items, whereas the TYPE clause defines the accuracy and internal representation required for implementor defined storage elements (such as floating point etc.). THE PICTURE clauses are essentially the same as those found in normal COBOL and PL/I data structure definitions, where A stands for alphabetic, 9 stands for numeric character, X stands for any character, V stands for an assumed radix point, etc. Thus the element termed NAME is assigned twenty alphabetical characters, IDENTIFICATION-NUM is six numerics, and DATE-OF-BIRTH is two numerics followed by any character followed by two numerics followed by any character then by two numerics, while AGE is two numerics followed by an assumed decimal point followed by one numeric. For COBOL readers, it should be noted that the PICTURE clause is quoted. The TYPE clause has a general default of DECIMAL/FIXED/REAL. If the declaration is followed by numbers, they are used to specify the precision, where the first integer (seven in the example of SALARY) specifies the minimum number of significant decimal digits, while the second number (two in this example) gives the scaling factor (i.e., the position of the decimal point).

The specification for the PERSON record also contains some additional clauses. The first of these, the CHECK clause, can occur in three different formats. The example illustrates the use of PICTURE for validation purposes. Thus DATE-OF-BIRTH has the PICTURE "99X99X99", and if the input does not conform with this picture, a validation error will result. A second type of error-checking uses a data base procedure. This is illustrated in the item termed COMPLETION-DATE, where a procedure is invoked called NOT-BEFORE (we can imagine that such a procedure would use the START-DATE to check that the date of initiation of the degree was not stated to occur after its completion). The data base procedure is understood to be able to utilize any item within that record as input parameter to check for an error condition. The third type of validity check ensures that the value of the item is between certain set limits. This is illustrated by item SALARY, where the salary is required to be between the values of 8000 and 45,000.

The final clause of interest in this record is that belonging to AGE; which is specified as being the result of a calculation specified as AGE-CALC. Furthermore this is specified as a VIRTUAL result, which means that AGE-CALC is invoked whenever the item AGE is requested by the program. It is understood that the calculation procedure therefore uses the date-of-birth (another item within this record) in conjunction with the system parameters of today's date to generate the present AGE. Obviously, the concept of AGE being a virtual result comes about in two ways. First of all the data administrator decided that AGE should be computed knowing the date of birth and present date. He therefore provided a data base procedure which produces the age. Furthermore in considering the relative merit of storing this permanently within the data base versus computing it on request he decided for the latter. His argument may have gone as follows:

> If the age is stored permanently (ACTUAL OF RESULT) then it will be changed whenever the input parameters change. This occurs if the date-of-birth should be changed, which is unlikely, or when today's date changes, which is (presumably) daily. Thus the <u>entire set</u> of person records would have age recomputed every day, which is expensive. However, since the activity on requests for age are unlikely to be very high for this particular set of applications, it is reasonable to compute it on request.

The next record to be defined is the MEDICAL record. As was previously discussed, it will be separately stored in an area known as MED-AREA. Because this is likely to be utilized substantially by the medical department in searching for records based on the DISEASE rather than through the person record and MED-SET relationship, it is reasonable to use a location mode of CALC where DISEASE is the key, and obviously people may have several illnesses with the same disease during their time with the enterprise, consequently duplicates will be allowed. The entire record definition is therefore:

```
RECORD IS MEDICAL;
  LOCATION MODE CALC USING DISEASE
          DUPLICATES ARE ALLOWED;
  WITHIN MED-AREA.
  1  PERSON-NAME; IS ACTUAL, SOURCE NAME OF OWNER OF MEDSET.
  1  ABSENCE-DATES; PICTURE "99X99X99X99X99X99";
     FOR ENCODING CALL DATE-PROC.
  1  DISEASE;  PICTURE IS "A(30)".
  1  NOTE-PAGE;  PICTURE "A(1500)".
```

As previously, all items have either a PICTURE or TYPE clause with the exception of the PERSON-NAME which is shown as a SOURCE item. The SOURCE statement allows the copying or use of an element from the owner record of a particular set. In this case, the MEDICAL record is a member of the MEDSET, while its owner is the PERSON record. See Figure 1. We would like to incorporate the NAME of the PERSON within the MEDICAL record (giving it a new name of PERSON-NAME). By using the ACTUAL rather than VIRTUAL form of the statement, we are implying that we wish this item to be physically stored within the record (rather than obtained at the time that it is referenced). This means, of course, that when the person is retired, if the medical record is retained, then it still contains information about who actually had this particular disease.

The ENCODING clause used for ABSENCE-DATES permits a further decoupling of data input and data storage. This allows ABSENCE-DATES to be specified according to some particular PICTURE, but the physical storage of the data to be quite different. Naturally, a decoding statement will normally also be required, unless ABSENCE-DATES are never to be output in their previous physical format.

The final record in this data base is that of JOB. It is assumed that a JOB record is located in the same area as its owner (i.e. PERSON), and that it is located through the mechanism of the set of which it is a member (JOBSET), as shown in figure 1. This means that there is one principal mechanism for locating records: the knowledge of the area into which it is to be placed, and the fact that it is coupled, in some fashion, to the PERSON record which owns it. The other method of locating the record is DIRECT. The actual definition of this record is therefore:

```
RECORD NAME JOB;
     LOCATION IS VIA JOBSET SET;
     WITHIN AREA OF OWNER
     1 JOB CODE; PIC "X(4)".
     1 START-DATE; PIC "99X99X99".
     1 FINISH-DATE; PIC "99X99X99".
     1 PERFORMANCE-RATING; PIC "99V9".
```

All that remains to define the data base is now to describe the sets, which involves a statement of the ownership and membership relations, and certain other key information about their organization. The first of these sets we have termed JOBSET, where the owner is the PERSON record and the only member record is JOB.

Because the data administrator sees the necessity to pass in both directions along the jobs which have been located according to their owner (PERSON record) description, the set has been defined as requiring "prior" processing. The definition shows where a particular job should be inserted within the set of all jobs of a particular person; a statement is made that a new job will be inserted last in the string. This implies that the data administrator knows that he wishes to keep a chronological file, and consequently he always inserts the latest job at the end. Equally because he specifies that the order is PERMANENT, he is implying that no reordering of these records is allowed, except during a particular run unit, which means that a particular program may do some local reordering, but this action makes no permanent change to the data base.

The definition of the set is therefore as follows:

```
SET NAME IS JOBSET;
     SET IS PRIOR PROCESSABLE;
     ORDER IS PERMANENT INSERTION IS LAST;
     OWNER IS PERSON;
     MEMBER IS JOB, MANDATORY, AUTOMATIC;
     SET OCCURRENCE SELECTION IS THRU JOBSET OWNER
     IDENTIFIED BY CURRENT OF SET.
```

The deletion and insertion properties of a JOB record are now considered. Because it is assumed that no job will ever be entered unless there is a person to whom this "belongs", the insertion property is stated to be AUTOMATIC, which means that the storage of a particular job record will automatically cause it to be inserted within the current JOBSET. Equally, it is assumed that the JOB records have no relevance unless there is a PERSON. Consequently, whenever the person is deleted from the data base, his job records also disappear. The deletion property is therefore stated to be MANDATORY. Finally, in order to define the method of selecting a set, the simple mechanism of CURRENT is used here.

The "set occurrence selection" clause may take several forms: the simplest is first illustrated here. The selection by CURRENT means that there is no easy way to select a set (as illustrated later) but that a FIND command with JOBSET in its "record selection expression" will return that record which is the <u>current</u> of that record. Currency, as discussed later, means that there is a record occurrence which is <u>marked</u> as current for each area type, record type, set type, run unit, etc.

The next set to be defined is known as the MEDSET. This is quite similar to the definition of the JOBSET except that it is assumed that ordering may be temporary (i.e., a doctor may wish to reorder the medical records in a permanent fashion). For the same reason, the programs are allowed to pick <u>where</u> the medical record would be inserted in a set (by allowing NEXT). The insertion properties are assumed to be AUTOMATIC, as previously. In deletion, however, it is assumed that there is a need for retaining medical records after employees have been deleted from the data base; thus the deletion property is OPTIONAL. Once again, set occurrence selection is using CURRENT. The definition of the MEDSET is therefore:

```
SET NAME IS MEDSET;
     ORDER IS TEMPORARY INSERTION IS NEXT;
     OWNER IS PERSON;
     MEMBER IS MEDICAL, OPTIONAL, AUTOMATIC;
     SET OCCURRENCE SELECTION IS THRU MEDSET OWNER
     IDENTIFIED BY CURRENT OF SET.
```

One extra set must be defined: that which contains all the MEDICAL records of retired employees. This is termed RETIRED-EMPLOYEE-MEDICAL. It is defined as follows:

```
SET NAME IS RETIRED-EMPLOYEE-MEDICAL;

     ORDER IS TEMPORARY INSERTION IS NEXT;
     ON INSERT CALL
         SEE-IF-WE-WANT-THIS-DISEASE;
     OWNER IS SYSTEM;
     MEMBER IS MEDICAL, OPTIONAL, MANUAL;
       SET OCCURRENCE SELECTION IS BY PROCEDURE
       DISEASE-SEARCH.
```

This is a "singular" set, where the owner is the SYSTEM. The order is assumed to be entirely at the discretion of the doctor, and in consequent defined as being TEMPORARY, while insertion is (by program) into the NEXT position. An ON condition during the time of insertion of this record invokes a procedure called SEE-IF-WE-WANT-THIS-DISEASE. It is assumed that this procedure will check to see whether the medical department is interested in a particular disease (presumably being stored in some list of interest) and therefore either complete the insertion or not depending on the current research or requirements.

Because we do not automatically wish to store every medical record on this set during the first insertion into the base, we make its insertion property MANUAL. Because there would be little meaning to an attempt to delete the SYSTEM, the membership of the set has a deletion property of OPTIONAL. Because it is assumed that the medical department may wish to do particular searches, it is stated that set occurrence selection is accomplished by using a procedure known as DISEASE-SEARCH.

The example does not illustrate one major type of set occurrence selection: the VIA LOCATION MODE OF OWNER. In this, the method of location of the owner is examined. This presumably allows the owner of the set to be selected. Then some key value for the member record is compared with the contents of the occurrences of the member records until a match occurs, whence the record has been selected.

3. DEFINITION OF THE SUBSCHEMAS

This section will define two different subschemas of the employee data base schema. However, because a program accesses data through a subschema, the definition of a subschema must be associated with a particular part of, or need for, the data base. Consequently, in order to discuss the needs, as expressed in a subschema, we must consider some particular use of the data base. We therefore first introduce the three problems which will be programmed in the next section.

The first program is essentially to either perform an initial load of the data base, or else add information about a new employee. It therefore must be able to load all information which is initially stored in the person, job and medical records of the data base. Furthermore, it must be able to access both of the areas (which are now called realms, which is the word used for areas in the COBOL subschema and its programs), and finally it must be able to access both the MEDSET and JOBSET (but not the RETIRED-EMPLOYEE-MEDICAL set).

The second program is to determine whether certain diseases are more prevalent for people who have performed certain types of jobs. But because the data administrator, in his definition of the schema, made no provision for a search based on job type, it is necessary to visit each PERSON record. Having found a person, the program must check his job records to determine whether anyone matches the particular job, and if so to check whether he has ever since contracted the disease. Such a program must be able to access all PERSON, MEDICAL, and JOB records. Consequently, it requires exactly the same subschema as the first program.

The third program involves the retirement of all employees who have reached the mandatory retirement age. This involves searching the entire set of employee records to determine who has now reached the age of 65, issuing some sort of report to this effect, and making a deletion of their records from the data base. However, because the medical department has given notice of interest in certain diseases, their medical records are to be retained (under certain circumstances).

Because only certain portions of the records are of interest for this program, and because it is necessary to have the RETIRED-EMPLOYEE-MEDICAL set addressed during the running of the program, a new sub-schema will be necessary for this particular example.

3.1 First Subschema Definition

There are five divisions of a subschema definition. The first of these is the TITLE division, in which the SUBSCHEMA is named, it is linked to its SCHEMA by name, any PRIVACY LOCK to be placed on the use or changing of the subschema is defined, and the PRIVACY KEY for using the schema is provided. The title division for the first two programs is therefore as follows:

```
TITLE DIVISION.
SS   ENTIRE-DATA-BASE WITHIN EMPLOYEE-BASE;
     PRIVACY LOCK FOR INVOKING IS 'OK-TO-COMPILE';
     PRIVACY KEY IS 'COPY OK'.
```

In this, the subschema is named ENTIRE-DATA-BASE, and it is stated to be a subschema of the schema named EMPLOYEE-BASE. On referring to the PRIVACY LOCK of this schema for COPY, we find that the privacy lock is contained within an item called COMP-LOCK. We assume that the data base administrator on "setting" this lock has placed into it the value "COPY OK". Thus providing a privacy key within the SUBSCHEMA allows access to the SCHEMA (for copying those relevant portions defined only in the schema). A lock is also provided against unauthorized use of the subschema. As will be seen later in the programs, they must provide the key, which is the same as the variable string defined to be the lock: viz "OK-TO-COMPILE".

The next division is used to redefine names. It may be used by the data administrator to provide a synonym capability between the data base name and a program which has not used the same name. Also it may be used to bridge differences between multiple language interfaces. In our example, the area or realm was termed PANDJ-AREA, but it would be impossible to use this name in a language such as PL/I or FORTRAN, because the hyphen is not an allowed symbol within a name. We may wish therefore to redefine the name for such a contingency. This is illustrated by the code:

```
    ALIAS DIVISION.
    AD    PANDJ-AREA BECOMES PANDJ.
```

It should be noted that once this change has been made, all references to this area within the subschema or the program which uses it must use the new name. Thus the old name has been essentially expunged or replaced in the definition. The subschema (and consequently the program) can only refer to the new name.

The next division is used for defining the realms. As has already been suggested in the previous discussion, this subschema will require both of the realms that were previously defined in the schema. This is done by the code:

```
   REALM DIVISION.
   RD   ALL.
```

Next we define the necessary sets. As previously discussed, only two of them are necessary and these are defined as follows:

```
    SET DIVISION.
    SD   JOBSET.
    SD   MEDSET.
```

The final division is used for defining the records. In this, there is substantial difference from the definition of the schema. First of all, portions may be omitted. In fact in our case the only element omitted from the PERSON record is AGE. This is because age has been stated to be the result of a calculation, and consequently is never input, but may be referenced as available in the program. Neither of these programs refers to age, and consequently it is entirely unnecessary to include age within the subschema definition. It might be noted that if it was included, then buffer space would be allocated to it, which is totally unnecessary.

One of the principal differences between the schema definition and the subschema is that the level numbers may range from 02 to 49. Consequently, there may be a need to remap the levels. In our case, the schema uses level 1 for elements such as NAME, IDENTIFICATION-NUM, etc. These must be mapped to level 2 or greater for the subschema definition. Apart from this, the PERSON record looks very similar to the schema definition, being as follows:

```
    RECORD DIVISION.
    01  PERSON.
        02  NAME PICTURE A(20).
        02  IDENTIFICATION-NUM PIC 9(6).
        02  DATE-OF-BIRTH PIC 99X99X99.
        02  SALARY PIC 9(5)V99.
        02  EDUCATION-INFO OCCURS 3 TIMES.
            03  DEGREE PIC A(2).
            03  START-DATE PIC 99X99X99.
            03  COMPLETION-DATE PIC 99X99X99.
            03  DEGREE-RECEIVED-FROM PIC A(20).
```

It will be seen that this is exceedingly similar, except for dropping the CHECK clauses, and removing quotations from the PICTURE definitions.

The definition of the MEDICAL record is also quite similar to that of the schema, being as follows:

```
    01  MEDICAL; WITHIN MED-AREA.
        02  ABSENCE-DATES; PIC 999999X999999.
        02  DISEASE; PIC A(30).
        02  NOTE-PAGE; PIC X(1500).
```

It will be seen that the WITHIN statement for the realm was included, although this was unnecessary. It is only totally necessary if the original record could be in more than one area, and we wish to select <u>which</u> of the areas are available to a particular subschema.

The final record is that for a job history, and is almost identical to that defined in the schema. It may be written as follows:

```
    01  JOB.
        02  JOB-CODE: PIC X(4).
        02  START-DATE; PIC 99X99X99.
        02  FINISH-DATE; PIC 99X99X99.
        02  PERFORMANCE-RATING; PIC 99V9.
```

3.2 The Second Subschema Definition

As previously suggested, the final program that will be described involves retiring an employee. The title division is very similar to that of the first example, with the subschema named RETIRE it is:

```
    TITLE DIVISION.
    SS   RETIRE WITHIN EMPLOYEE-BASE;
         PRIVACY LOCK FOR INVOKING IS 'AOK-COMPILE';
         PRIVACY KEY IS 'COPY OK'.
```

No alias will be given in this particular subschema, and consequently the names used in the subschema and within the program will be identical with those selected from the schema. As previously, all realms will be selected, and in this case all sets will also be required. The relevant definition is therefore:

```
    REALM DIVISION.
    RD   ALL.
    SET DIVISION.
    SD   ALL.
```

All records are of interest in this program, however there are few of the elementary items which are referred to by the programs. Thus in the

PERSON records the relevant information is merely the AGE, with presumably the NAME and IDENTIFICATION-NUM in order to be able to output information to the personnel department. Thus the first part of the record division will be as follows:

```
RECORD DIVISION.
01  PERSON.
    02  NAME; PIC A(20).
    02  IDENTIFICATION-NUM; PIC 9(6).
    02  AGE; PIC 99V9.
```

The other two records do not participate in any information transfer to or from the COBOL program. In essence, therefore, they need to have no contents. However due to the requirement that all records contain some data (a requirement of DBFP), the definition is given as:

```
01  MEDICAL.
    02  DISEASE; PIC A(30).
01  JOB.
    02  JOB-CODE; PIC X(4).
```

4. THREE EXAMPLES OF COBOL PROGRAMS

This report introduces three examples of programs selected to show some of the operations of the data manipulation language within COBOL. However, it should be realized that these are not complete programs, since no attempt has been made to tailor them for any particular application, nor are all potential error conditions recognized. The first program is given essentially in its entirety. The second program is only given in outline, while the third program contains everything except the normal input/output statements and file definitions.

4.1 Program to Add a New Employee.

The normal identification and environment division for the program called ADDPERS is provided as follows:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ADDPERS.
DATE-WRITTEN.  APRIL 1973.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
    SOURCE-COMPUTER.  HAL-9000.
    OBJECT-COMPUTER.  HAL-9000.
INPUT-OUT SECTION.
  FILE-CONTROL.
     SELECT NEW-EMPLOYEES ASSIGN TO TAPE.
     SELECT EXCEPTION-REPORT ASSIGN TO PRINTER.
```

The first reference to the data base occurs in the data division. In this, the subschema is stated to be the one that was first defined in section 3 (i.e., ENTIRE-DATA-BASE, which belongs to the schema termed EMPLOYEE-BASE). It will be remembered that a lock was placed on this subschema, which is now satisfied by the identical key string "OK-TO-COMPILE". Thus the data division starts as follows:

```
DATA DIVISION.
SUB-SCHEMA SECTION.
DB   ENTIRE-DATA-BASE
     WITHIN EMPLOYEE-BASE;
     PRIVACY KEY IS 'OK-TO-COMPILE'.
```

In the file section, we first define the input file called NEW-EMPLOYEES. The file heading is therefore:

```
FILE SECTION.
FD NEW-EMPLOYEES;
   LABEL RECORDS ARE STANDARD.
```

We shall consider input of a new employee's information to consist of one logical record. This will be in three principal parts, the PERSON-INPUT, the MEDICAL-INPUT, and the JOB-INPUT. The file structure is illustrated in figure 3.

The program will be reading one whole employee's input, and storing the information out onto the data base one record at a time. Thus the input buffer will contain one record of the NEW-EMPLOYEE-RECORD. We wish first to move the information about the person, as given in

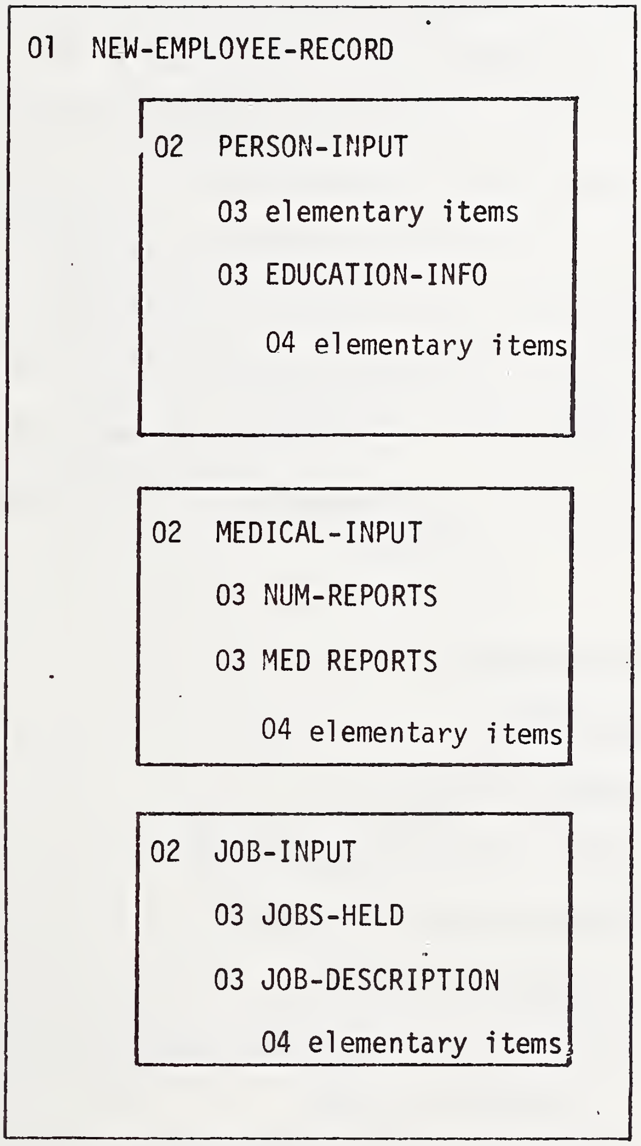


Figure 3 NEW-EMPLOYEES Input File Logical Record Structure

PERSON-INPUT, into the work area for the PERSON data base record, and then store it.

This could involve a large amount of MOVE statements, unless we use the MOVE CORRESPONDING verb. Thus, although the MOVE CORRESPONDING option is seldom used by good COBOL programmers, it is illustrated in this example. Therefore, in order to simplify the process, the names of the elementary items within the NEW-EMPLOYEES input file will be identical to those used for the data base definition. Also the medical reports will each exist at the 03 level of the input file, and the number of reports will vary according to need with a count (NUM-REPORTS) for each medical input record. Essentially the same methods are provided for the job inputs. Thus the employee input structure is as follows:

```
01  NEW-EMPLOYEE-RECORD.
    02  PERSON-INPUT.
        03  NAME; PIC A(20).
        03  IDENTIFICATION-NUM; PIC 9(6).
        03  DATE-OF-BIRTH; PIC 99X99X99.
        03  SALARY; PIC 9(7)V99.
        03  EDUCATION-INFO; OCCURS 3 TIMES.
            04  DEGREE; PIC A(2).
            04  START-DATE; PIC 99X99X99.
            04  COMPLETION-DATE; PIC 99X99X99.
            04  DEGREE-RECEIVED-FROM; PIC A(20).
```

```
02  MEDICAL-INPUT.
    03  NUM-REPORTS; PIC 999.
    03  MED-REPORTS; OCCURS 0 TO 999 TIMES DEPENDING
           ON NUM-REPORTS.
        04  ABSENCE-DATES; PIC 99X99X99X99X99X99.
        04  DISEASE; PIC A(30).
        04  NOTE-PAGE; PIC X(1500).

02  JOB-INPUT.
    03  JOBS-HELD; PIC 999.
    03  JOB-DESCRIPTION; OCCURS 0 TO 999 TIMES
          DEPENDING ON JOBS-HELD IN JOB-INPUT.
        04  JOB-CODE; PIC X(4).
        04  START-DATE; PIC 99X99X99.
        04  FINISH-DATE; PIC 99X99X99.
        04  PERFORMANCE-RATING; PIC 99V9.
```

When errors are found in the input, some exception report information will be printed out. The only error checked in this program is one of duplication of an identification number: i.e., finding that someone is about to be added who has the same identification number as a person whose information is already stored in the data base. The definition of the exception report is:

```
FD EXCEPTION-REPORT;
   LABEL RECORDS ARE OMITTED;
   BLOCK CONTAINS 1 RECORDS;
   DATA RECORDS ARE HEADING-RECORD,
                    DETAIL-RECORD,  SUM-RECORD.
                01  HEADING-RECORD;
                        PIC X(70).
                01  DETAIL RECORD;
                        PIC X(69).
                01  SUM-RECORD;
                        PIC X(40).
```

The working storage section must now define two indices which are used for moving records from the input tape to the storage buffers. Furthermore, the working storage section contains the necessary formats for the exception report. It is defined as follows:

```
    WORKING-STORAGE SECTION.
    77 MED-COUNT; USAGE COMP; PIC 9999.
    77 MOVE-COUNT; USAGE COMP; PIC 9999.
    01 HEADING-RECORD-VALUE.
       02 FILLER; PIC X(10); VALUE SPACES.
       02 FILLER; PIC X(60);
  -       VALUE 'INPUT RECORDS REJECTED BECAUSE OF DUPLICATE
  -       'ID NUMBER'.


    01 DETAIL-RECORD-VALUE.
       02 FILLER; PIC X(5); VALUE SPACES.
       02 FILLER; PIC X(16); VALUE IS 'ID NUMBER IS'.
       02 ID-NUM; PIC 9(6).
       02 FILLER; PIC X(22); VALUE IS 'NAME OF PERSON IS'.
```

```
      02 NAME-REJ; PIC X(20).
   01 SUM-RECORD-VALUE.
      02 FILLER; PIC X(34); VALUE IS 'TOTAL # OF REJECTED
-         'RECORDS IS'.
      02 TOTAL-REJECTED; PIC Z(5)9; VALUE 0.
```

This concludes the file section, and therefore the program continues with the procedure division. The first part of the procedure division consists of the declaratives, which are used to satisfy privacy, and deals with exception conditions during the running of the program. The first declarative is that one which generates a privacy key which allows updating of the realm called MED-AREA. This is needed because the definition of the schema included a privacy lock on UPDATE of the realm called MED-AREA, as defined in the schema to be a procedure called MED-DEPT; we assume that this is met by having the key "MEDICAL UPDATING BY E.H.S.". This process is illustrated in the following code:

```
PROCEDURE DIVISION.

DECLARATIVES.
PRIVATE-REALM SECTION.
     USE FOR PRIVACY ON UPDATE FOR MED-AREA
MOVE-PRIVACY.
        MOVE 'MEDICAL UPDATING BY E.H.S.' TO DATABASE-PRIVACY-KEY.
```

The second use of declaratives is in dealing with <u>exception conditions</u>. These occur when a data base request is invalid, and are quite similar to the detection of an end-of-file when reading a logical record. The intent of the declarative is to allow the program to branch or otherwise deal with the condition. However, in order to describe the program more easily, the exception condition <u>here</u> will merely return control to the program. In the appendix, a better version of the program is given. <u>There</u>, the discussion will show how the programmer may effectively use a declarative for dealing with exception conditions.

Thus the <u>simplified</u> declarative section assumes that the program checks conditions and performs the branching, and the declarative merely returns control as follows:

```
OUT-OF-REALM SECTION.
    USE FOR DATABASE-EXCEPTION.

EXCEPTION-PARAGRAPH.
    EXIT.
END DECLARATIVES.
```

To reiterate, in this simplified example, the effect of the detection of a data base exception is to pass control to the declaratives section, determine that no operation is to be performed, and therefore return control to the statement immediately following that which caused the exception.

The next operation is to open and make ready all of the input, output, and data base files. This is done by:

```
    HOUSEKEEPING.
        OPEN INPUT NEW-EMPLOYEES,
            OUTPUT EXCEPTION-REPORT.
        READY; USAGE MODE IS EXCLUSIVE UPDATE.
        WRITE HEADING-RECORD FROM HEADING-RECORD-VALUE.
```

In this, the NEW-EMPLOYEES file is made ready for input, and the EXCEPTION-REPORT is made ready for output. The areas (realms) are made ready for processing using the verb READY, with the usage being "update in exclusive mode." This implies that no other run unit may expect to reference these realms (either for updating or retrieving them) during the operation of this program; i.e., until such time as the realms are "finished." A heading is put out for exception reports at the end of this piece of code.

The principal loop of the program is now entered. This involves reading a record of the input file and then starting to process it. After this record has been processed, control is returned again to the BEGIN-UPDATE paragraph until such time as an end-of-file condition is detected at which time the program is ended by passing control to an ending paragraph (FINISH-UPDATE, defined later). First, however, the input record of a person must be moved to the person record; as previously discussed, this program is simplified by using corresponding names. After that, the educational information is passed into the user work area. This is achieved by the code:

```
    BEGIN-UPDATE.
        READ NEW-EMPLOYEES; AT END GO TO FINISH-UPDATE.
    MOVE-PERSON-AND-EDUCATION.
        MOVE CORRESPONDING PERSON-INPUT TO PERSON.
        PERFORM MOVE-EDUCATION-INFO VARYING MOVE-COUNT
            FROM 1 BY 1 UNTIL MOVE-COUNT IS GREATER THAN 3.
        GO TO STORE-EMPLOYEE.
    MOVE-EDUCATION-INFO.
        MOVE CORR EDUCATION-INFO IN PERSON-INPUT
             (MOVE-COUNT) TO EDUCATION-INFO IN PERSON
             (MOVE-COUNT).
```

This operation is illustrated in figure 4. Thus the MOVE-PERSON-AND-EDUCATION and MOVE-EDUCATION-INFO paragraphs perform the moving of the PERSON-INPUT to the PERSON record.

We are now ready to perform the storage of the data using the paragraph:

```
    STORE-EMPLOYEE.
         STORE PERSON.
         IF DATABASE-STATUS = 14051;
         GO TO DUPLICATE-RECORD.
```

The STORE command now refers to the location mode of its record (PERSON) and determines that this was defined (in the schema) as calculation mode, using IDENTIFICATION-NUM as a unique key. Within the input record, the identification number exists. The system therefore computes an address for storing the record, and proceeds to store it.

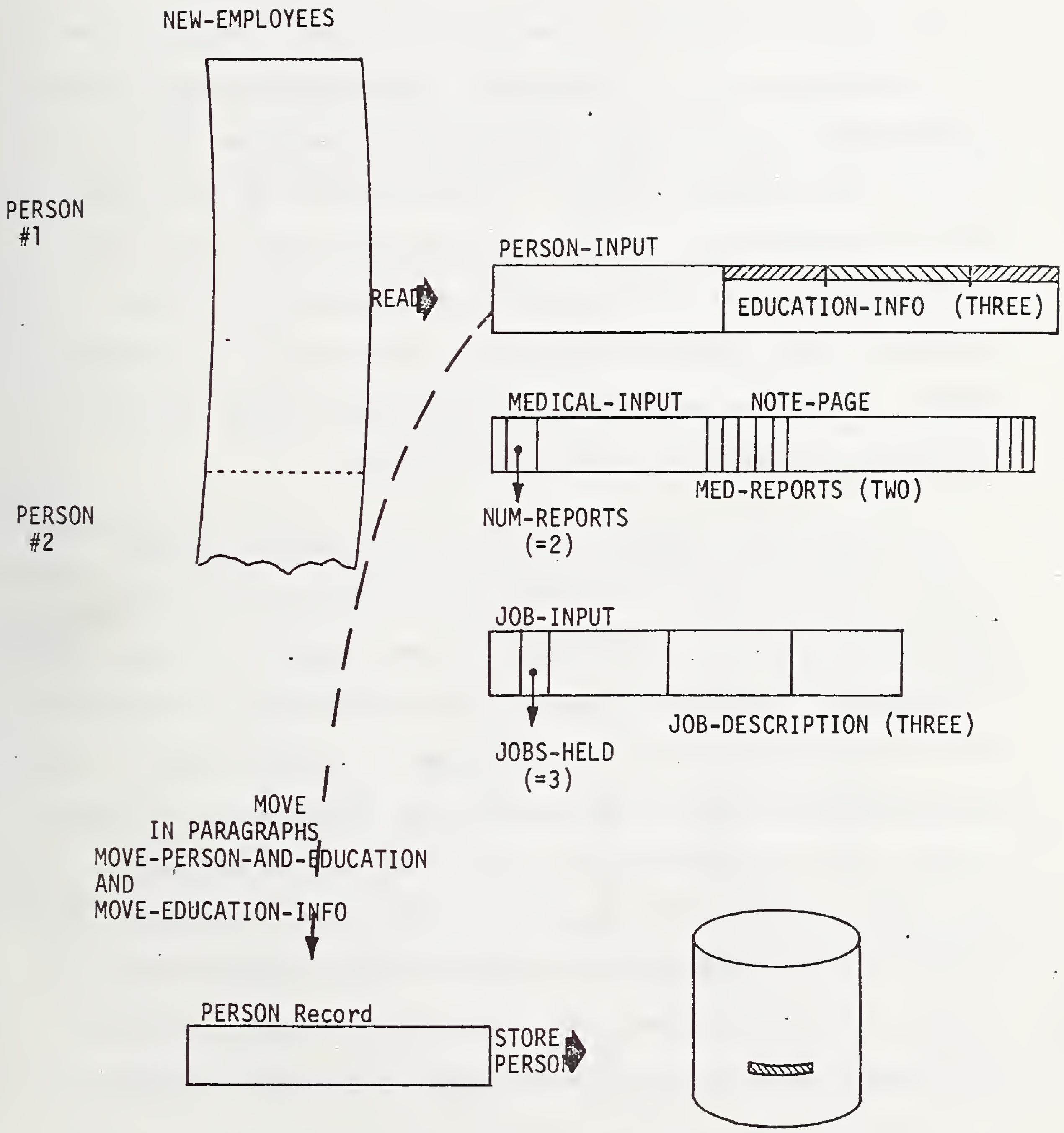


Figure 4 Move & Store PERSON Record

However, in the event that the input value should duplicate a previous identification number, a data base exception condition will occur.

An exception condition, as previously discussed, transfers control to the declarative section, which merely returns control to the program. In consequence, a check (in the program) is now made to see the condition of the DATABASE-STATUS word. In the event that the input contains a duplication of a previously stored identification number, the data base status condition will be 14051, consequently this would immediately switch control to the exception paragraph called DUPLICATE-RECORD. This will be shown later; we shall assume a normal condition, and continue with the main portion of the program.

It is important to realize that the act of storing the PERSON record has set up a number of currency status indicators for various realms, sets, and records. In fact the current condition of these indicators is illustrated in figure 5. At the beginning, all indicators are null (#), whereas after the first storage of a person record, P_1 has become the current of the run unit, the PERSON record, the realm in which it is stored (PANDJ), and as P_1 is the owner of both sets used in this subschema, it becomes the current record of these sets also. This last fact is <u>extremely</u> important when it comes to understanding the storage of the member records, and the fact that the schema specified that member records are to be linked to their owners <u>automatically</u>.

The next set of code performs the movement of the MEDICAL records, one at a time, from the input area to the work area, and hence to store

	RUN UNIT	REALMS		SETS		RECORDS		
		MED-AREA	PANDJ	MEDSET	JOBSET	PERSON	MEDICAL	JOB
START	#	#	#	#	#	#	#	#
AFTER STORE PERSON	P_1	#	P_1	P_1	P_1	P_1	#	#
AFTER STORE MEDICAL	M_1	M_1	P_1	M_1	P_1	P_1	M_1	#
ANOTHER STORE MEDICAL	M_2	M_2	P_1	M_2	P_1	P_1	M_2	#
AFTER STORE JOB	J_1	M_2	J_1	M_2	J_1	P_1	M_2	J_1
ANOTHER STORE JOB	J_2	M_2	J_2	M_2	J_2	P_1	M_2	J_2

Figure 5 Currency Status Indicators

the new MEDICAL record in the data base:

```
    MOVE-MEDICAL.
        PERFORM MOVE-MED-REC THRU STORE-MEDICAL VARYING MED-
           COUNT FROM 1 BY 1 UNTIL MED-COUNT IS GREATER THAN
           NUM-REPORTS.
        GO TO MOVE-JOBS.
    MOVE-MED-REC.
        MOVE CORR MED-REPORTS (MED-COUNT) TO MEDICAL.
    STORE-MEDICAL.
       STORE MEDICAL.
```

Figure 6 illustrates the moving of the first medical input record into the medical work area. After this, the medical record is stored. The data base management system references the location mode of the medical records and determines it is by calculation of the data base key from the item DISEASE, which is contained within the medical record. Furthermore, it is allowable to duplicate this item. However, the insertion property of the MEDICAL record is stated to be automatic; i.e. it is always automatically included as a member of the MEDSET set, as it is stored; thus, M_1 is made an automatic member of the set which was previously current; i.e. by P_1. The currency status indicators are again illustrated in figure 5.

Figure 6 shows the condition where there is another medical record. Let us therefore consider a second iteration through this code. The second medical report is moved to the medical work area. Then M_2 is stored according to the calculation of its position according to the disease. However, once again, it is an automatic member of the MEDSET

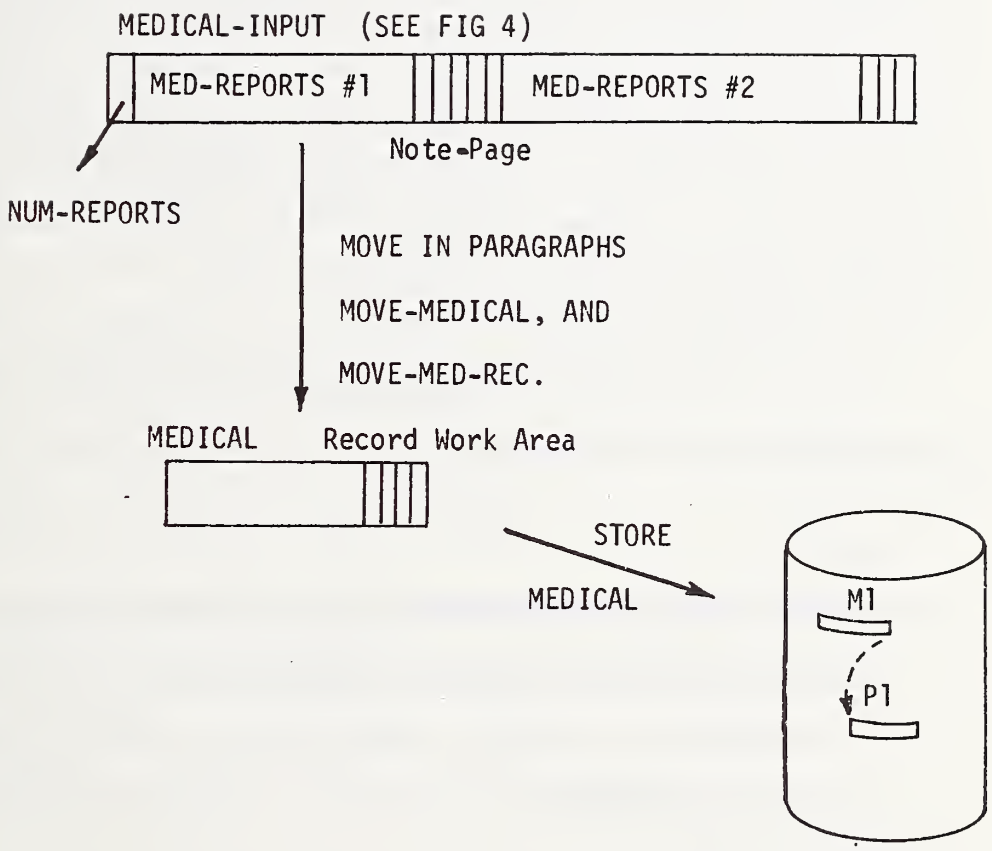


Figure 6 Move and Store MEDICAL Record

set, and consequently <u>it</u> becomes the current member of that set. However there is now another decision that is made: whether the medical appears <u>before</u> or <u>after</u> the previous medical record.

In order to determine the ordering of the medical records within the MEDSET we must again refer to the original definition of this set in the schema. The order is stated to be <u>temporary</u> with <u>insertion last.</u> Thus the latest medical record input will be placed at the end of all of the records (which happens to mean that it is in the same position as if it was placed NEXT). Once again, the condition of the currency status indicators are shown in figure 5 (in the fourth line, where the current record of the run unit is M_2).

We are now ready to transfer and store the JOB records. This is accomplished by the following code:

```
MOVE-JOBS.
   PERFORM MOVE-JOBS-INFO VARYING MOVE-COUNT FROM 1 BY 1
     UNTIL MOVE-JOBS IS GREATER THAN JOBS-HELD IN
     JOB-INPUT.
   GO TO BEGIN-UPDATE.
MOVE-JOBS-INFO.
   MOVE JOB-DESCRIPTION (MOVE-COUNT) TO JOB.
   STORE JOB.
```

Once again each job is separately taken from the input and placed into the user work area for JOB records. At the end of each transfer JOB is stored. This is, once again, illustrated by the currency status indicator change in figure 5. Because JOB records are part of the JOBSET set, the current of JOBSET becomes the latest JOB record input, and because JOB records are in the PANDJ area, the latest JOB record also becomes the current of PANDJ.

Because the JOB record is again an automatic member of the JOBSET set it automatically is linked into the set which currently has P_1 as owner. At the time of storage of the second job, the ordering clause must be invoked, determining in which position it is to be placed. The schema states that order is permanent, which means that no changes in order can be made once this record has been inserted; it is placed in the NEXT position. Because the method of insertion is chronological, the next position happens to be the end position, and consequently this accomplishes the same effect as using LAST; it would, of course, be possible for a different program to make an insertion in a <u>different</u> position because of the use of the word NEXT.

After successful completion of moving and storing job information, the program reverts to BEGIN-UPDATE. There it continues reading and updating the data base with more people records, until either a duplication record is found, or the input is exhausted.

The next portion of the program therefore deals with an error (in the form of a duplicate record). It is essentially a simple piece of COBOL code as follows:

```
     DUPLICATE-RECORD.
        ADD 1 TO TOTAL-REJECTED.
        MOVE NAME IN PERSON-INPUT TO NAME-REJ.
        MOVE IDENTIFICATION-NUM IN PERSON-INPUT TO ID-NUM.
        WRITE DETAIL-RECORD FROM DETAIL-RECORD-VALUE.
        GO TO BEGIN-UPDATE.
```

Finally, with the determination of an end-of-file condition on input, we wish to write out the trailer for the exception report, close our input and output files, and finally FINISH the realms, which makes them unavailable for the processing by this program. The code is:

```
     FINISH-UPDATE.
        WRITE SUM-RECORD FROM SUM-RECORD-VALUE,
        CLOSE NEW-EMPLOYEES, EXCEPTION-REPORT.
        FINISH.
        STOP RUN.
```

4.2 The Job/Disease Program.

As was previously discussed, this program is intended to list all people who work at certain jobs, and check to see whether they have since contracted a given disease. From this survey, a list of names is to be generated, which has flags against their entry if they have had the disease. The subschema for this example is identical to that used in the first program.

To simplify the discussion, this program is only in skeleton form; e.g., it contains no input-output statements, or file defintions. We start with the procedure division, but will not provide the declaratives.

We shall assume that the input has provided an item named TYPE-JOB which contains the interesting job code, and that the item named DIS-TYPE contains the disease of interest. Since we are only dealing with retrieval of information, there would be no reason why other people should not also be able to utilize the same realms, provided that they do not do concurrent updating. The code for opening processing on the realms is therefore:

```
     HOUSEKEEP.
        READY; USAGE MODE IS
              PROTECTED RETRIEVAL.
```

The strategy for this program is somewhat constrained by the decisions of the data base administrator. He has not provided a quick way to pass through the data base, visiting all records of a particular job; e.g. he could have defined another singular set with JOB records as members sorted or indexed on JOB-CODE. Consequently we must resort to the method given here: to pass through the entire data base, examining each person's job until one of interest is found, then examining his medical records and issuing lists accordingly.

No index was provided for the PERSON records, and consequently we need a way of visiting each one throughout the data base. This can be achieved by using the internal ordering within a realm. Thus the principal method will be to find the _first_ person of the PANDJ realm, then the _next_ etc., until some time as the data base has been exhausted. It should be noted that the _next_ person _at the start_ is, indeed the _first_ person. Hence there is no need to differentiate between these two cases.

The effect of exhausting the data will be to produce a data base status of 04021, where 04 indicates an error during finding, and 021 is the condition for exhaustion. Thus the PERSON record is found by the code:

```
    GET-NEXT-PERSON.
        FIND NEXT PERSON OF
            PANDJ.
        IF DATABASE-STATUS = 04021;
            GO TO DONE, ELSE GO TO FIND-AND-GET-JOBS.
```

It should be noted that there is no interest yet in actually retrieving the PERSON record (by a GET statement), because we have not determined whether this is a person of interest. We therefore first look at his JOB history by examining, one at the time, the jobs within his JOBSET. Whenever we FIND a JOB, we GET it. We then check to see whether it has the JOB-CODE of interest. If it does not, we can continue examing the rest of the jobs belonging to this particular JOBSET, until we have exhausted the base, or else we find that we have a match, and go to the relevant code. This is illustrated in the following:

```
    FIND-AND-GET-JOBS.
       FIND NEXT JOB OF JOBSET.
       IF DATABASE-STATUS = 04021;
           GO TO GET-NEXT-PERSON, ELSE GET.

    CHECK-JOB.
       IF JOB-CODE = TYPE-JOB; GO TO
         HAVE-MATCH, ELSE GO TO
         FIND-AND-GET-JOBS.
```

Let us suppose that the second job belonging to this person is of interest, the currency status indicators are therefore as shown on figure 7. Now we have an "interesting" person, we wish to determine his name, and then examine his medical records; i.e., we wish to return to the PERSON record. However if we do this without careful consideration, we shall change the currency status indicators of JOBSET. Then we shall never be able to examine past his job of the JOBSET.

What we do therefore is to retain a currency indicator (a "book mark") pointing at this particular JOB as the current record of the JOBSET, while still going back and getting the PERSON record. This may be achieved in one of two different ways: the first asks for the current value of the PERSON record, whereas the other requests the owner record of the current JOBSET. But whichever way this is achieved we must insure that the JOBSET is not changed, and therefore we retain its currency. This is illustrated in the following piece of code:

```
     HAVE-MATCH.
        FIND PERSON CURRENT; RETAINING JOBSET.
        GET.
```

or:

```
        FIND JOBSET OWNER; RETAINING CURRENCY FOR JOBSET.
        GET.
```

The medical records are now examined one at a time by passing through the MEDSET. Each record is examined to determine whether it has the disease of interest, and after exhausting all the medical information

	RUN UNIT	REALMS		SETS		RECORDS			
		MED-AREA	PANDJ	MEDSET	JOBSET	PERSON	MEDICAL	JOB	
FIND FIRST PERSON	P_1	#	P_1	P_1	P_1	P_1	#	#	
FIND NEXT JOB	J_{11}	#	J_{11}	P_1	J_{11}	P_1	#	J_{11}	
FIND NEXT JOB	J_{12}	#	J_{12}	P_1	J_{12}	P_1	#	J_{12}	
* JOB MATCH .. FIND PERSON RETAINING JOBSET	P_1	#	P_1	P_1	$\underline{J_{12}}$ *	P_1	#	J_{12}	
FIND/NEXT MEDICAL	M_{11}	M_{11}	P_1	M_{11}	$\underline{J_{12}}$	P_1	M_{11}	J_{12}	
FIND NEXT MEDICAL	M_{11}	M_{11}	P_1	M_{11}	$\underline{J_{12}}$	P_1	M_{12}	J_{12}	
GO TO FIND-AND-GET-JOBS & FIND NEXT JOB OF JOBSET	M_{12}	M_{12}	J_{13}	M_{12}	J_{13}	P_1	M_{12}	J_{13}	with retaining
	M_{12}	M_{12}	J_{11}	M_{12}	J_{11}	P_1	M_{12}	J_{11}	without retaining

Figure 7 Currency for Program #2

a report is generated. This is achieved by the following paragraph:

```
     SEEK-MEDICAL.
        MOVE Ø TO FLAG.
        FIND NEXT MEDICAL OF MEDSET.
        IF DATABASE-STATUS=04021; GO TO REPORTING-PARA.
        GET.
        IF DISEASE = DIS-TYPE; GO TO YES-DISEASE,
            ELSE GO TO SEEK-MEDICAL.
```

Each time that a disease is found which corresponds to that of interest a check is made to see whether the absence dates in the medical record are later than the starting dates of the job. If it is, we will go to the reporting routine (REPORTING-PARA) otherwise we will go back to look at the other reports i.e., through the SEEK-MEDICAL paragraph. Thus the skeleton code is:

```
     YES-DISEASE.
         MOVE 1 TO FLAG.
    *NOTE
    *NOW CHECK TO SEE IF ABSENCE-DATES IN MEDICAL ARE
    *LATER THAN START-DATE IN JOB.  IF SO, GO TO
    *REPORTING-PARA, OTHERWISE GO TO SEEK-MEDICAL.
```

Reporting takes cognizance of whether the disease has been flagged or not, and produces output statements, after which it returns to look for another job. The reason for going to find a new job is to report all cases where the person worked at this particular type of job. The skeleton code is therefore:

```
     REPORTING-PARA.
    *NOTE
    *NO OUTPUT STATEMENTS GIVEN HERE.
         GO TO FIND-AND-GET-JOBS.
```

It is important to realize that the reason for retaining the currency indicators of JOBSET was to enable a future seek to continue passing along the particular set (i.e., without starting over). This is illustrated by the last two lines of figure 7. If the currency status of JOBSET is not retained on line 4, then this column remains at P_1 until the last operations. Thus instead of the next being J_{13} (following J_{12}) it is J_{11} (which follows P_1). Thus the error in not retaining currency would cause the program to go into a loop. Finally, on completion of the entire program, the realms must be closed which may be achieved by:

```
     DONE.
         FINISH.
         STOP RUN.
```

4.3 Program to Retire an Employee

As has previously been discussed, this program identified all persons who have reached the mandatory retirement age of 65. When a retiree is found, a report is to be made, and his records are deleted from the system; with the exception that any medical records which contain information of "interest to the medical department" are retained. The medically interesting diseases are defined through a program termed "SEE-IF-WE-WANT-THIS-DISEASE", and if some one of these is found, then the medical record is retained.

The data base administrator, in setting up the schema has, in fact, incorporated the procedure in the schema with an ON condition, which ensures that the insertion of a medical record into the RETIRED-EMPLOYEE-MEDICAL set will immediately invoke this procedure. It is assumed that the procedure will either insert the employee's medical records into this set, or not, depending on the relevance of the disease to the present studies of the medical department.

Thus the essence of the program is to find out <u>who</u> is to be retired, and attempt to insert the medical records into the singular set for retired employees, and then use a deletion (ERASE) on the person so that all of his records including JOB and MEDICAL, disappear (unless the medical records are required by the medical department). The mechanism for this will be discussed briefly during presentation of the skeleton program.

As previously, no input/output statements will be included.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  RETIRE-AND-SAVE-MEDICAL.
DATE-WRITTEN.  APRIL 1973.
```

The data division contains the sub-schema paragraph. This states which subschema is to be used, and how the privacy key for compile is passed.

```
DATA DIVISION.
SUB-SCHEMA SECTION.
```

```
     DB   RETIRE
          WITHIN EMPLOYEE-BASE;
          PRIVACY KEY IS 'AOK-COMPILE'.
          FILE SECTION.
        .
        .
        .
        .
        .
```

The procedure division, as in the previous programs starts off with the declaratives, which provide privacy keys for update (i.e. satisfying the data base privacy for updating, and the return of data base exceptions back to the program). The declaratives are used in an unsophisticated way, as in the first program. Better use of declaratives is shown in the appendix.

```
PROCEDURE DIVISION.
DECLARATIVES.
PRIVACY-SATISFACTION SECTION.
     USE FOR PRIVACY ON UPDATE FOR MED-AREA.
PRIVACY-CLAUSE.
     MOVE 'MD UPDATE FOR RETIRE' TO DATABASE-PRIVACY-KEY.
EXCEPT-CLAUSE.
     USE FOR DATABASE-EXCEPTION.
EXCEPT-DO.
     EXIT.
END DECLARATIVES.
```

We begin by readying the realm for exclusive use, and each person is examined to determine AGE. The process of finding each person is essentially the same as that discussed in program #2. It starts by finding the 'next' person in the realm PANDJ-AREA. It is interesting to note that the subschema has not redefined the name of the realm, and therefore it is the same as in the original schema. The code is:

```
BEGIN-PROGRAM.
    READY; USAGE MODE IS EXCLUSIVE UPDATE.
GET-NEXT-PERSON.
    FIND NEXT PERSON OF PANDJ-AREA.
    IF DATABASE-STATUS = 04021; GO TO DONE.
```

In order to check the age, we now obtain the record, treating AGE as though it was stored within the data base (it is, in fact, a virtual result, and therefore computed at the time of access). The code would be:

```
CHECK-AGE.
    GET.
    IF AGE IS GREATER THAN 65 OR IS EQUAL TO
        65; GO TO RETIRE-PERSON, ELSE GO TO
        GET-NEXT-PERSON.
```

Whenever a person is found who has reached retirement age, a pass is made through MEDSET, and "connection" of each medical record is made into the RETIRED-EMPLOYEE-MEDICAL set. As has been described, the CONNECT (INSERT is the same verb as CONNECT when referenced in the DDL) will possibly not be performed due to the data base procedure which is invoked.

An IF statement is now used to determine whether MEDSET is an empty set. The code is:

```
     RETIRE-PERSON.
       IF MEDSET SET IS EMPTY; GO TO NO-MEDICAL-RECORDS.
     HAVE-MEDICAL.
       FIND NEXT MEDICAL OF MEDSET.
       IF DATABASE-STATUS = 04021; GO TO NO-MEDICAL-RECORDS.
       CONNECT MEDICAL TO RETIRED-EMPLOYEE-MEDICAL.
       GO TO HAVE-MEDICAL.
```

After a connection of the MEDICAL record to the RETIRED-EMPLOYEE-MEDICAL set, an erasure of the person record is performed. If the verb takes on the particular form of SELECTIVE ERASE, then the erasure not only takes away that particular PERSON record, but also all records which are members of any set in which the person record is the owner, with the exception that any record which is defined in the schema as transient (OPTIONAL) will not be erased if it is a member of some other set.

Let us suppose that the condition for a PERSON record which is being erased is shown in figure 8. On erasing P_{23}, because JOBs are permanent (or mandatory) members of JOBSET, $J_{23,1}$ and $J_{23,2}$ immediately are erased. However the medical records are optional (transient) members of MEDSET, therefore they are only deleted in the event that they are not connected to any other set. Thus $M_{23,1}$ and $M_{23,3}$ will be erased, whereas $M_{23,2}$ will merely be removed from MEDSET, and left as a member of the RETIRED-EMPLOYEE-MEDICAL set. The code to complete this program to fill out the form is therefore:

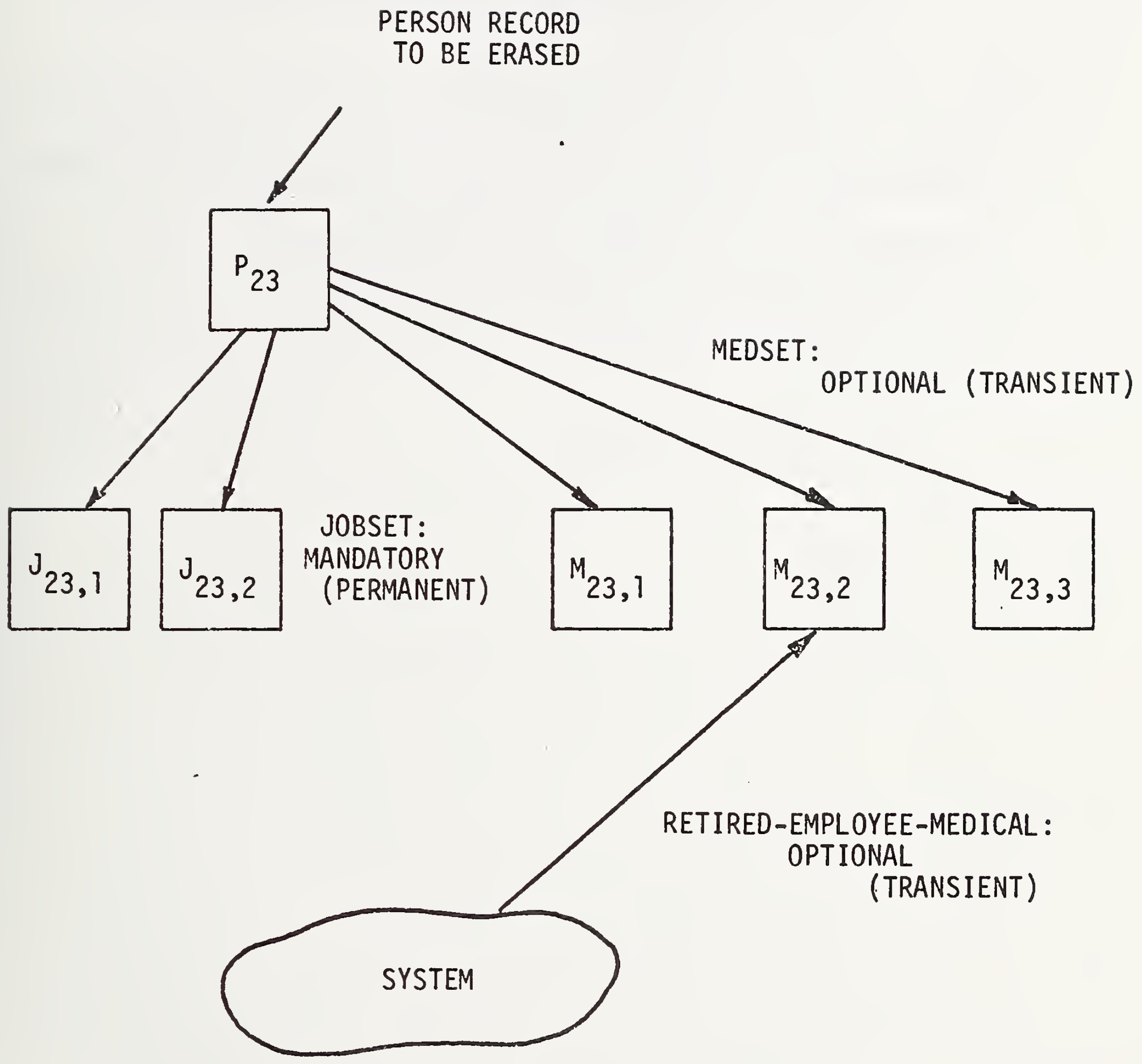

PERSON RECORD TO BE ERASED
P_{23}
MEDSET: OPTIONAL (TRANSIENT)
$J_{23,1}$
$J_{23,2}$
JOBSET: MANDATORY (PERMANENT)
$M_{23,1}$
$M_{23,2}$
$M_{23,3}$
RETIRED-EMPLOYEE-MEDICAL: OPTIONAL (TRANSIENT)
SYSTEM

Figure 8. ERASE PERSON SELECTIVE MEMBERS

```
 NO-MEDICAL-RECORDS.

      PERFORM RETIRED-EMPLOYEE-REPORT.

      ERASE PERSON SELECTIVE MEMBERS.

*NOTE   WE JUST DELETED THE PERSON RECORD AND ALL OF ITS

*JOB RECORDS, AND REMOVED THE MEDICAL RECORD FROM

*THE MEDSET IF IT WAS CONNECTED TO THE RETIRED-

*EMPLOYEE-MEDICAL SET, OTHERWISE IT, TOO, WAS

*DELETED.

      GO TO GET-NEXT-PERSON.

 DONE.

      FINISH.

      STOP RUN.
```

Appendix: Outline of Schema, Subschema, and Programs with More Effective use of Exception Conditions

The use of exception conditions and declarative statements was simplified in the description of the programs to aid in presentation of the tutorial examples. In this appendix, the examples are presented still in their outline form, but with the declaratives illustrated as they might be used by a more sophisticated programmer. A brief explanation is given of the difference at the end.

1. The Schema

```
SCHEMA NAME IS EMPLOYEE-BASE;

PRIVACY LOCK FOR LOCKS IS PROCEDURE E-LOCK;

PRIVACY LOCK FOR COPY IS COMP-LOCK;

PRIVACY LOCK FOR ALTER IS EMP-ALT-LOCK;

PRIVACY LOCK FOR DISPLAY 'SEE EMPLOYEE BASE';

ON ALTER CALL SNOOPER-ROUTINE.

AREA NAME IS PANDJ-AREA.

AREA NAME IS MED-AREA;

PRIVACY LOCK FOR UPDATE IS PROCEDURE MED-DEPT.

RECORD NAME IS PERSON;


LOCATION MODE IS CALC EMP-HASH USING IDENTIFICATION-NUM

  DUPLICATES ARE NOT ALLOWED;


WITHIN PANDJ-AREA;


ON DELETE CALL MICROFILM-RECORDER;


PRIVACY LOCK FOR DELETE IS PROCEDURE EMP-LEFT.

     NAME; PICTURE IS "A(20)".

     IDENTIFICATION-NUM; PICTURE IS "9(6)".

     DATE-OF-BIRTH; PICTURE IS "99X99X99"; CHECK IS PICTURE.

  1  AGE:  PICTURE "99V9"; IS VIRTUAL RESULT OF AGE-CALC.

  1  SALARY:  TYPE IS FIXED 7.2; CHECK IS VALUE 8000.00

       THRU 45000.00;
  1  EDUCATION-INFO; OCCURS 3 TIMES.

  2  DEGREE; PICTURE "AA".

  2  START-DATE; PICTURE "99X99X99"; CHECK NOT-BEFORE.

  2  DEGREE-RECEIVED-FROM; PIC "A(20)".

RECORD IS MEDICAL;

  LOCATION MODE CALC USING DISEASE

          DUPLICATES ARE ALLOWED;

  WITHIN MED-AREA  .

      1  PERSON-NAME; IS ACTUAL, SOURCE NAME OF OWNER OF MEDSET.

  1  ABSENCE-DATES; PICTURE "99X99X99X99X99X99";

     FOR ENCODING CALL DATE-PROC.

  1  DISEASE;  PICTURE IS "A(30)".

  1  NOTE-PAGE; PICTURE "A(1500)".

RECORD NAME JOB;

     LOCATION IS VIA JOBSET SET;

     WITHIN AREA OF OWNER  .

     1 JOB CODE; PIC "X(4)",

     1 START-DATE; PIC "99X99X99".

     1 FINISH-DATE; PIC "99X99X99".

     1 PERFORMANCE-RATING; PIC "99V9".

SET NAME IS JOBSET;

     SET IS PRIOR PROCESSABLE;

     ORDER IS PERMANENT INSERTION IS LAST;

     OWNER IS PERSON;

     MEMBER IS JOB, MANDATORY, AUTOMATIC;

     SET OCCURRENCE SELECTION IS THRU JOBSET OWNER
     IDENTIFIED BY CURRENT OF SET.
SET NAME IS MEDSET;
     ORDER IS TEMPORARY INSERTION IS NEXT;
     OWNER IS PERSON;
     MEMBER IS MEDICAL, OPTIONAL, AUTOMATIC;
     SET OCCURRENCE SELECTION IS THRU MEDSET OWNER
     IDENTIFIED BY CURRENT OF SET.
SET NAME IS RETIRED-EMPLOYEE-MEDICAL;

     ORDER IS TEMPORARY INSERTION IS NEXT;
     ON INSERT CALL
         SEE-IF-WE-WANT-THIS-DISEASE;
     OWNER IS SYSTEM;
     MEMBER IS MEDICAL, OPTIONAL, MANUAL;

       SET OCCURRENCE SELECTION IS BY PROCEDURE
       DISEASE-SEARCH.
```

2. The Two Subschema

2.1 ENTIRE-DATA-BASE Subschema

```
TITLE DIVISION.

SS   ENTIRE-DATA-BASE WITHIN EMPLOYEE-BASE;

     PRIVACY LOCK FOR INVOKING IS 'OK-TO-COMPILE';

     PRIVACY KEY IS 'COPY OK'.

ALIAS DIVISION.

AD   PANDJ-AREA BECOMES PANDJ.

REALM DIVISION.

RD   ALL.

SET DIVISION.

SD   JOBSET.

SD   MEDSET.

RECORD DIVISION.

01   PERSON.

     02  NAME PICTURE A(20).

     02  IDENTIFICATION-NUM PIC 9(6).

     02  DATE-OF-BIRTH PIC 99X99X99.

     02  SALARY PIC 9(5)V99.

     02  EDUCATION-INFO OCCURS 3 TIMES.

         03  DEGREE PIC A(2).

         03  START-DATE PIC 99X99X99.

         03  COMPLETION-DATE PIC 99X99X99.

         03  DEGREE-RECEIVED-FROM PIC A(20).

01   MEDICAL; WITHIN MED-AREA.
```

```
    02  ABSENCE-DATES; PIC 999999X999999.

    02  DISEASE; PIC A(30).

    02  NOTE-PAGE; PIC X(1500).

01  JOB.

    02  JOB-CODE; PIC X(4).

    02  START-DATE; PIC 99X99X99.

    02  FINISH-DATE; PIC 99X99X99.

    02  PERFORMANCE-RATING; PIC 99V9.
```

2.2 RETIRE Subschema

```
TITLE DIVISION.

SS   RETIRE WITHIN EMPLOYEE-BASE;

     PRIVACY LOCK FOR INVOKING IS 'AOK-COMPILE';

     PRIVACY KEY IS 'COPY OF'.

REALM DIVISION.

RD   ALL.

SET DIVISION.

SD   ALL.

RECORD DIVISION.

01  PERSON.

    02  NAME; PIC A(20).

    02  IDENTIFICATION-NUM; PIC 9(6).

    02  AGE; PIC 99V9.

01  MEDICAL.

    02  DISEASE; PIC A(30).

01  JOB.

    02  JOB-CODE; PIC X(4).
```

3. Three Programs

3.1 The ADDPERS Program

```
IDENTIFICATION DIVISION.

PROGRAM-ID. ADDPERS.

DATE-WRITTEN.  SEPT 1973.


ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

    SOURCE-COMPUTER.  HAL-9000.

    OBJECT-COMPUTER.  HAL-9000.

INPUT-OUTPUT SECTION.

  FILE-CONTROL.

     SELECT NEW-EMPLOYEES ASSIGN TO TAPE.

     SELECT EXCEPTION-REPORT ASSIGN TO PRINTER.


DATA DIVISION.

SUB-SCHEMA SECTION.

DB   ENTIRE-DATA-BASE

     WITHIN EMPLOYEE-BASE;

     PRIVACY KEY IS 'OK-TO-COMPILE'.

FILE SECTION.

FD NEW-EMPLOYEES;

   LABEL RECORDS ARE STANDARD.

01  NEW-EMPLOYEE-RECORD.

    02  PERSON-INPUT.

        03  NAME; PIC A(20).

        03  IDENTIFICATION-NUM; PIC 9(6).
```

```
          03  DATE-OF-BIRTH; PIC 99X99X99.

          03  SALARY; PIC 9(7)V99.

          03  EDUCATION-INFO; OCCURS 3 TIMES.

              04  DEGREE; PIC A(2).

              04  START-DATE; PIC 99X99X99.

              04  COMPLETION-DATE; PIC 99X99X99.

              04  DEGREE-RECEIVED-FROM; PIC A(20).

      02  MEDICAL-INPUT.

          03  NUM-REPORTS; PIC 999.

          03  MED-REPORTS; OCCURS 0 TO 999 TIMES DEPENDING

                 ON NUM-REPORTS.

              04  ABSENCE-DATES; PIC 99X99X99X99X99X99.

              04  DISEASE; PIC A(30).

              04  NOTE-PAGE; PIC X(1500).

      02  JOB-INPUT.

          03  JOBS-HELD; PIC 999

          03  JOB-DESCRIPTION; OCCURS 0 TO 999 TIMES

                DEPENDING ON JOBS-HELD IN JOB-INPUT.

              04  JOB-CODE; PIC X(4).

              04  START-DATE; PIC 99X99X99.

              04  FINISH-DATE; PIC 99X99X99.

              04  PERFORMANCE-RATING; PIC 99V9.

  FD EXCEPTION-REPORT;

     LABEL RECORDS ARE OMITTED;

     BLOCK CONTAINS 1 RECORDS;

     DATA RECORDS ARE HEADING-RECORD,
          DETAIL-RECORD,  SUM-RECORD.
01 HEADING-RECORD;
          PIC X(70).
01 DETAIL RECORD;
          PIC X(69).
01 SUM-RECORD;
          PIC X(40).
WORKING-STORAGE SECTION.
77 MED-COUNT; USAGE COMP; PIC 9999.
77 MOVE-COUNT; USAGE COMP; PIC 9999.
01 HEADING-RECORD-VALUE.
   02 FILLER; PIC X(10); VALUE SPACES.
   02 FILLER; PIC X(60);
      VALUE 'INPUT RECORDS
-     'REJECTED BECAUSE OF DUPLICATE
-     'ID NUMBER'.
01  DETAIL-RECORD-VALUE.
    02  FILLER; PIC X(5); VALUE SPACES.
    02  FILLER: PIC X(16); VALUE IS 'ID NUMBER IS'.
    02  ID-NUM; PIC 9(6).
    02  FILLER; PIC X(22); VALUE IS 'NAME OF PERSON IS'.
    02  NAME-REJ; PIC X(20).
01  SUM-RECORD-VALUE.
    02  FILLER; PIC X(34); VALUE IS 'TOTAL # OF REJECTED
-          'RECORDS IS'.
    02  TOTAL-REJECTED; PIC Z(5)9; VALUE 0.
```

```
PROCEDURE DIVISION.

DECLARATIVES.
PRIVATE-REALM SECTION.
   USE FOR PRIVACY ON UPDATE FOR MED-AREA.
MOVE-PRIVACY.
        MOVE 'MEDICAL UPDATING BY E.H.S.' TO DATABASE-PRIVACY-KEY.
OUT-OF-REALM SECTION.
   USE FOR DATA BASE-EXCEPTION.
EXCEPTION-PARAGRAPH.
   IF DATABASE-STATUS = 14051;
   GO TO DUPLICATE-RECORD;
   ELSE GO TO FINISH-UPDATE.
END DECLARATIVES.
HOUSEKEEPING.
   OPEN INPUT NEW-EMPLOYEES,
       OUTPUT EXCEPTION-REPORT.
   READY; USAGE MODE IS EXCLUSIVE UPDATE.
   WRITE HEADING-RECORD FROM HEADING-RECORD-VALUE.
BEGIN-UPDATE.
   READ NEW-EMPLOYEES; AT END GO TO FINISH-UPDATE.
MOVE-PERSON-AND-EDUCATION.
   MOVE CORRESPONDING PERSON-INPUT TO PERSON.
   PERFORM MOVE-EDUCATION-INFO VARYING MOVE-COUNT
       FROM 1 BY 1 UNTIL MOVE-COUNT IS GREATER THAN 3.
   GO TO STORE-EMPLOYEE.
```

```
MOVE-EDUCATION-INFO.

    MOVE CORR EDUCATION-INFO IN PERSON-INPUT

         (MOVE-COUNT) TO EDUCATION-INFO IN PERSON

         (MOVE-COUNT).

STORE-EMPLOYEE.

     STORE PERSON.

MOVE-MEDICAL.

     PERFORM MOVE-MED-REC THRU STORE-MEDICAL VARYING MED-

         COUNT FROM 1 BY 1 UNTIL MED-COUNT IS GREATER THAN

         NUM-REPORTS.

     GO TO MOVE-JOBS.

MOVE-MED-REC.

     MOVE CORR MED-REPORTS (MED-COUNT) TO MEDICAL.

STORE-MEDICAL.
     STORE MEDICAL.
MOVE-JOBS.

     PERFORM MOVE-JOBS-INFO VARYING MOVE-COUNT FROM 1 BY 1

        UNTIL MOVE-JOBS IS GREATER THAN JOBS-HELD IN

        JOB-INPUT.

     GO TO BEGIN-UPDATE.

MOVE-JOBS-INFO.

     MOVE JOB-DESCRIPTION (MOVE-COUNT) TO JOB.

     STORE JOB.

DUPLICATE-RECORD.

     ADD 1 TO TOTAL-REJECTED.

     MOVE NAME IN PERSON-INPUT TO NAME-REJ.

     MOVE IDENTIFICATION-NUM IN PERSON-INPUT TO ID-NUM.
    WRITE DETAIL-RECORD FROM DETAIL-RECORD-VALUE.
    GO TO BEGIN-UPDATE.
FINISH-UPDATE.
    WRITE SUM-RECORD FROM SUM-RECORD-VALUE,
    CLOSE NEW-EMPLOYEES, EXCEPTION-REPORT.
    FINISH.
    STOP RUN.
```

3.2 The JOB-DISEASE Program

```
PROCEDURE DIVISION.

DECLARATIVES.

    .

    .

    .

EXCEPTION-REQUIREMENTS SECTION.

     USE FOR DATABASE-EXCEPTION.

EXCEPT-FOR-ALL.

     IF DATABASE-RECORD-NAME = 'PERSON';

        GO TO DONE.

     IF DATABASE-RECORD-NAME = 'JOB';

        GO TO GET-NEXT-PERSON.

     IF DATABASE-RECORD-NAME = 'MEDICAL';

        GO TO REPORTING-PARA.

     EXIT.

END DECLARATIVES.

     HOUSEKEEP.

        READY; USAGE MODE IS

              PROTECTED RETRIEVAL.

     GET-NEXT-PERSON.

        FIND NEXT PERSON OF PANDJ.

     FIND-AND-GET-JOBS.

        FIND NEXT JOB OF JOBSET.

        GET.
```

```
 CHECK-JOB.
    IF JOB-CODE = TYPE-JOB; GO TO
      HAVE-MATCH, ELSE GO TO FIND-AND-GET-JOBS.
 HAVE-MATCH.
    FIND PERSON CURRENT;  RETAINING JOBSET.
    GET.
 SEEK-MEDICAL.
    MOVE Ø TO FLAG.
    FIND NEXT MEDICAL OF MEDSET.
    GET.
    IF DISEASE = DIS-TYPE; GO TO YES-DISEASE,
        ELSE GO TO SEEK-MEDICAL.
 YES-DISEASE.
    MOVE 1 TO FLAG.
*NOTE
*NOW CHECK TO SEE IF ABSENCE-DATES IN MEDICAL ARE
*LATER THAN START-DATE IN JOB.  IF SO, GO TO
*REPORTING-PARA, OTHERWISE GO TO SEEK-MEDICAL.
 REPORTING-PARA.
*NOTE
*NO OUTPUT STATEMENTS GIVEN HERE.
        GO TO FIND-AND-GET-JOBS.
        DONE.
            FINISH.
            STOP RUN.
```

3.3 The RETIRE-AND-SAVE-MEDICAL Program

```
    IDENTIFICATION DIVISION.
    PROGRAM-ID.  RETIRE-AND-SAVE-MEDICAL.
    DATE-WRITTEN.  SEPT 1973.
    DATA DIVISION.
    SUB-SCHEMA SECTION.
    DB      RETIRE
              WITHIN EMPLOYEE-BASE;
              PRIVACY KEY IS 'AOK-COMPILE'.
    FILE SECTION.
         .
         .
         .
    PROCEDURE DIVISION.
    DECLARATIVES.
    PRIVACY-SATISFACTION SECTION.
        USE FOR PRIVACY ON UPDATE FOR MED-AREA.
    PRIVACY-CLAUSE.
        MOVE 'MD UPDATE FOR RETIRE' TO DATABASE-PRIVACY-KEY.
    EXCEPT-CLAUSE.
        USE FOR DATABASE-EXCEPTION.
    EXCEPT-DO-THIS.
        IF DATABASE-RECORD-NAME = 'PERSON';
```

```
          GO TO DONE.
      IF DATABASE-RECORD-NAME = 'MEDICAL';
          GO TO NO-MEDICAL-RECORDS.
      EXIT.
   END DECLARATIVES.
   BEGIN-PROGRAM.
       READY; USAGE MODE IS EXCLUSIVE UPDATE.
   GET-NEXT-PERSON.
       FIND NEXT PERSON OF PANDJ-AREA.
       GET.
       IF AGE IS GREATER THAN 65 OR IS EQUAL TO
           65; GO TO RETIRE-PERSON, ELSE GO
           TO GET-NEXT-PERSON.
   RETIRE-PERSON.
     IF MEDSET SET IS EMPTY; GO TO NO-MEDICAL-RECORDS.
   HAVE-MEDICAL.
     FIND NEXT MEDICAL OF MEDSET.
     CONNECT MEDICAL TO RETIRED-EMPLOYEE-MEDICAL.
     GO TO HAVE-MEDICAL.
   NO-MEDICAL-RECORDS.
        PERFORM RETIRED-EMPLOYEE-REPORT.
        ERASE PERSON SELECTIVE MEMBERS.
  *NOTE  WE JUST DELETED THE PERSON RECORD AND ALL OF ITS
  *JOB RECORDS, AND REMOVED THE MEDICAL RECORD FROM
  *THE MEDSET IF IT WAS CONNECTED TO THE RETIRED-
  *EMPLOYEE-MEDICAL SET, OTHERWISE IT, TOO, WAS DELETED.
```

```
               GO TO GET-NEXT-PERSON.
          DONE.
               FINISH.
               STOP RUN.
```

4. <u>An Explanation of Declarative Usage</u>

The simplified examples in the body of this report merely return control to the program, where a test is made to determine where to switch the flow for each condition. The appendix examples have taken these tests out of the <u>procedure</u>, and put them in the <u>declarative</u> section.

Thus, in the first program the test checks to make sure that the data base status was 14051 (to show that a store had been issued on a record whose key was the same as that previously put into the data base). If it <u>is</u> a duplicate, the control passes to the DUPLICATE-RECORD paragraph, but if it is not, some other (previously unchecked in the simplified program) error condition has caused the transfer to the declarative section, and consequently control is passed to the finishing (FINISH-UPDATE) paragraph to publish the report and stop further processing.

In the second and third programs, the special database location called DATABASE-RECORD-NAME is examined to see which type of record caused the exception condition. In each case, the program branches in exactly the same way as it would for the unsophisticated program, but the condition is tested in the declarative section, which makes the program less cluttered with tests.

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET	1. PUBLICATION OR REPORT NO. NBSIR 74-500	2. Gov't Accession No.	3. Recipient's Accession No.
4. TITLE AND SUBTITLE THE CODASYL DATA BASE APPROACH: A COBOL EXAMPLE AND USE OF A PERSONNEL FILE			5. Publication Date February 1974
			6. Performing Organization Code
7. AUTHOR(S) Edgar H. Sibley			8. Performing Organ. Report No. NBSIR 74-500
9. PERFORMING ORGANIZATION NAME AND ADDRESS NATIONAL BUREAU OF STANDARDS DEPARTMENT OF COMMERCE WASHINGTON, D.C. 20234			10. Project/Task/Work Unit No. 640-1122
			11. Contract/Grant No.
12. Sponsoring Organization Name and Complete Address *(Street, City, State, ZIP)* Same as above.			13. Type of Report & Period Covered Final
			14. Sponsoring Agency Code

15. SUPPLEMENTARY NOTES

16. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.)*

This report introduces examples of the use of the proposed CODASYL Data Definition Language and Data Base Language extensions to COBOL. It does this by suggesting the needs and data base elements which can be expected for a set of simple personnel applications. The discussion of the data definitions centers around the decisions that the data administrator makes, and the tools that are provided for him. Then it discusses a few of the processes (programs) which are required by typical personnel departments, and shows their implementation (in outline) in three COBOL programs. The reader is expected to have some knowledge of the CODASYL specifications.

17. KEY WORDS *(six to twelve entries; alphabetical order; capitalize only the first letter of the first key word unless a proper name; separated by semicolons)*

COBOL; CODASYL; Data Base; Data Definition; Data Definition Language; Data Structure Applications.

18. AVAILABILITY [ ] Unlimited	19. SECURITY CLASS (THIS REPORT) UNCLASSIFIED	21. NO. OF PAGES 84 ~~78~~
[X] For Official Distribution. Do Not Release to NTIS		
[ ] Order From Sup. of Doc., U.S. Government Printing Office Washington, D.C. 20402, SD Cat. No. C13	20. SECURITY CLASS (THIS PAGE) UNCLASSIFIED	22. Price
[ ] Order From National Technical Information Service (NTIS) Springfield, Virginia 22151		