# IDMS QUERY LANGUAGE

by

William E Shea

B.S., University of Tampa, Tampa Florida, 1972

---

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1977

Approved by:

[illegible]

Major Professor

# TABLE OF CONTENTS

## Section 1.1 Introduction

In this day of credit cards, instant loan processing, and high speed order processing for business , a vast amount of data is required to keep each of these application programs running smoothly. The use of data bases containing the required information is becoming more and more a part of our society. In fact, James Martin commented,'The development of corporate data bases will be one of the most important data-processing activities for the rest of the 1970's.'(1). Data bases take on many forms, but one thing they all have in common is complexity. Record storage and retrieval techniques, set definitions, and data manipulation languages(DML) are but a few of the things that a person must thoroughly understand in order to use the information in the data base.

The purpose of this report is provide a description of a prototype query language(QL1) to be used with Cullinane Corporation's Integrated Database Management System(IDMS). QL1 is designed tc relieve the user of most of the requirements for understanding the structure of the data base and yet interactively manipulate the data base as desired.

The project was prompted by the growing use of on-line systems as well as the need for a less complex means of accessing data base elements. In order to relieve the user of much of the requirement of knowing the structure of the

--------------------

(1)James Martin, <u>COMPUTER DATA-BASE ORGANIZATION</u> ,(PRENTICE-HALL,1975),P. 2.

data base, QL1 uses a computer-prompted method of execution. That is to say that the user supplies information that he is asked for; rather than having to know in advance what information is going to be required next. A sample session is shown in Appendix B. Implementation of this type of man-computer dialog gives the impression that QL1 knows the structure of the data base when , in fact, even that is not required.

## Section 1.2 Design Requirements

In designing the query language for this project, certain requirements were specified; these were:

1. The language had to be interactive.
2. The user should not be required to know the structure of the data base.
3. Manipulation of the data base had to include:
    a. Entering new occurrences of any record type.
    b. Deleting old occurrences of any record type.
    c. Modifying the data items in any record type.
    d. Retrieving information contained in any record type.

The above requirements were satisfied although some assumptions were necessary and some restrictions had to be placed on the design of the data base. The following assumptions were made:

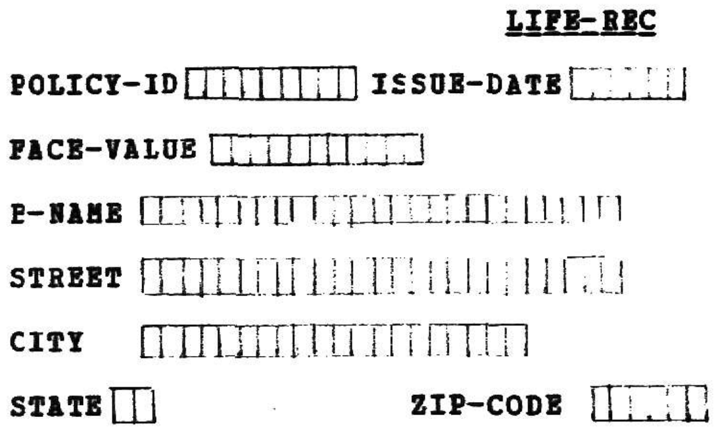
1. Set membership inclusion would be Mandatory Automatic in all cases.
2. Forms would be available to users listing the elementary data item names for each record type.(2)
3. Once compiled, all of the IDMS files as well as the QL1 files would be stored permanently on disk for use by the query language as well as other application programs that might be called by QL1, .i.e. report producing programs.

The only restriction placed on the design of the data

------------------

(2)See figure 1.2.1 for a sample user form.

base was on the type of attribute that could be given an elementary data item. Attributes that are acceptable are: character, and the X, V, and 9 formats of the picture specification. The reason for this restriction is that data-structure mapping is extremely difficult and complicated except when limited to the type of attributes that are byte aligned. There would be considerable difficulty in obtaining addresses for data items whose attribute type was not byte aligned. The reasons for the assumptions and the restriction are discussed in more detail in chapters two and three.



**LIFE-REC**

POLICY-ID ISSUE-DATE

FACE-VALUE

P-NAME

STREET

CITY

STATE ZIP-CODE

fig. 1.2.1 Sample user form.

## Section 1.3 IDMS Terminology

The following definitions are provided to assist the reader in understanding the terminology used throughout the report.(3)

1. SCHEMA. This is a complete description of the data base. It includes the names and definitions of all records, sets, and data items. The schema DATA DESCRIPTION LANGUAGE(DDL) is used to describe the schema.

2. SUBSCHEMA. This is the user's view of the data base. Only those records, sets, and areas that are used within a specific application program are defined in the subschema. While there is only one schema to define the data base, there may be several subschemas to provide different views of that data base to different users. The subschema DDL is used to describe the subschema.

3. RECORD. A record may be thought of as a COBOL or PL/1 like structure, with the level 1 name being the reord type. It is important to note that when referencing a record type, one might have several occurrences of that record type.

4. SET. A set is a logical relationship between two or more record types. Each set consists of one or more record types declared as members of the set, but only one record type declared as the owner of the set.

5. DATA ITEM. This is the smallest unit of data which has

-------------------

(3) A more thorough understanding of IDMS may be obtained by reading, <u>DATA MANIPULATION LANGUAGE PROGRAMMER'S REFERENCE GUIDE</u> (Cullinane Corporation), release 3-1, April, 1975. Hereafter this is cited as DML programmer's guide.

a name. This is analogous to the elementary data-element in a structure.

6. CALC. A method of determining the location to be used when storing a record in the data base. The CALC identifier is the name of an elementary data item within the record. When the record is stored, a procedure within the DBMS uses the value of the CALC identifier to determine the specific page in the data base on which the record will be stored.

7. VIA. This is another method by which to determine the location of a record. VIA specifies a set-name in which the object record participates as a member. When the record is stored VIA, it is placed on the same page, or a nearby page, as the owner of the set-name indicated.

8. MANDATORY AUTOMATIC. This refers to the manner in which a record is established as the member of a set. The 'Mandatory' portion means that once the membership of a record has been established, it's participation as a member of the set is permanent. The 'Automatic' portion means that membership in a set is established automatically by the IDMS system anytime an occurrence of the record type is stored in the data base.

## Section 2.1 Overview of QL1

QL1 was designed as a computer-prompted language. This simply means that the user is asked for specific information and then the various routines use that information to perform the function the user desires. (See appendix B for examples of this type of interaction) A computer-prompted method of implementation was considered most appropriate for this type of query language since it was felt that the user might often be just a casual user rather than a dedicated user(4). For such users the slightest problem communicating with the data base could cause considerable irritation. This can be readily seen if a manager of some type, not being familiar with the system, were to try to access the data base without any help from the computer. Not being used to the required commands, a lack of knowledge of a specific syntax, or a number of other problems could cause the manager to become hopelessly frustrated. With the computer-prompted method these problems are somewhat alleviated and the manager now only has to respond to the questions posed by the computer.

The query language itself has been implemented in PL/1. Interaction between the user at the terminal and the various executing routines is provided by the PL/1 'DISPLAY' and 'REPLY' statements.(5). The reader may wish to consult

--------------------

(4) The term casual is used to describe a user who only occassionaly uses the language, and a dedicated user as one who works constantly with the language. See James Martin, <u>DESIGN OF MAN-COMPUTER DIALOGUES</u>, (Prentice-Hall, 1973), PP. 25-26.

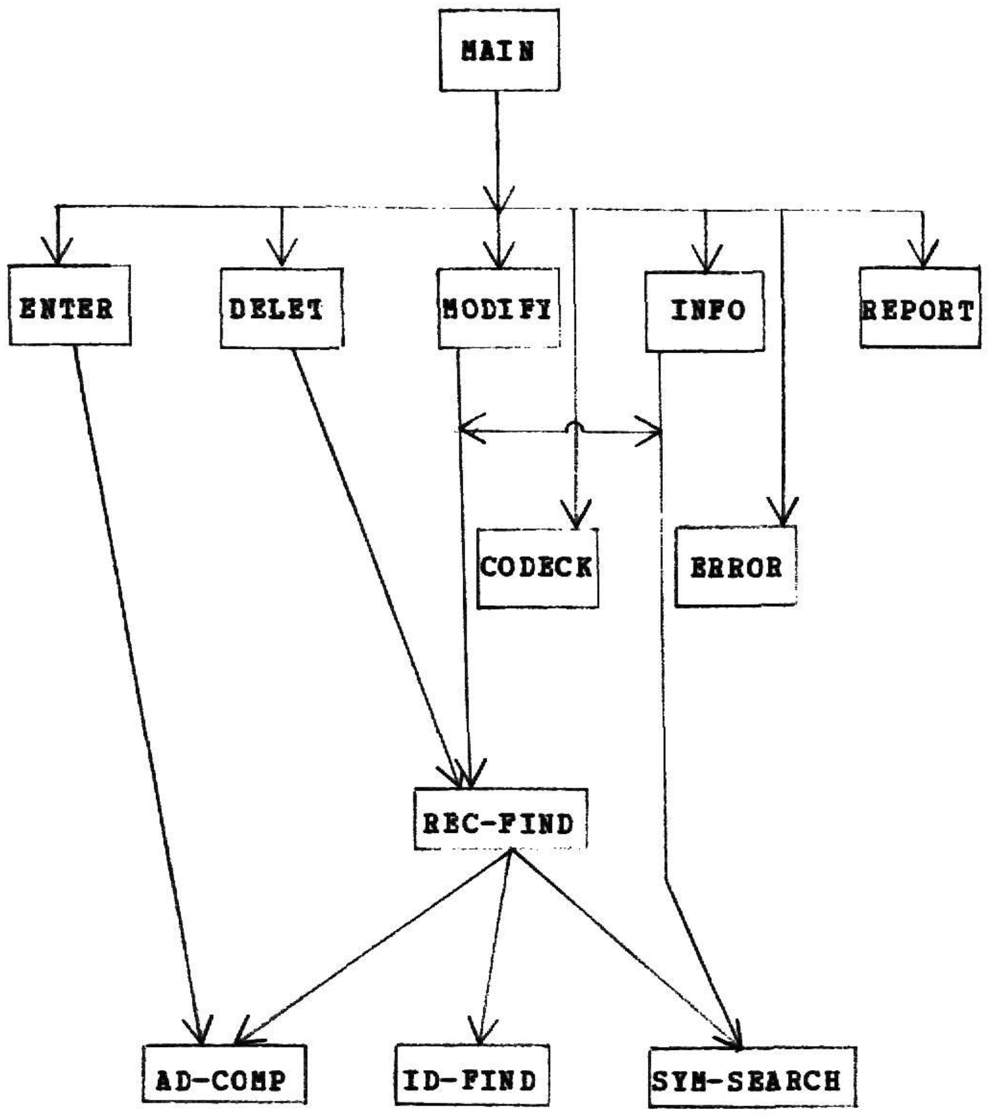
(5) <u>PL/1(F) LANGUAGE REFERENCE MANUAL</u> (IBM Corporation), 5th

appendix A for examples of the display and reply statements used in the source listings. QL1 is modular in design(see fig. 2.1) with the major modules being the driver, ENTER, DELET,MODIFY, and INFO routines. A fifth routine, REPORT, has been provided for but not implemented. This would be used to call specific report generating programs. Should this routine be desired, the user's Data Base Administrator (DBA) would be required to insure the routine was properly coded.

Several other modules and programs, which are described in detail in chapter three, handle such things as address calculation, retrieval of records into working storage, checking the validity of security codes and personnel numbers, and verifing data item names. The modules are called by the major modules and by each other as needed.

--------------------

ed.,December,1972,P.373.



HIERARCHY OF QL1 COMPONENTS

FIG 2.1

## Section 2.2 Description of main routine

The main routine, or driver, of QL1 is activated by typing QL1. All required file definitions are provided at that time. The terminal, after some miscelleneous information will respond with 'PLEASE ENTER YOUR SECURITY CODE.'.

At that time the user should enter his six digit security code that was provided by the Data Base Administrater. The next response by the terminal is for the user to enter his personnel number. After the personnel number has been entered, a call is made to CODECK(6). CODECK verifies the security code and personnel number of the user and returns to the main routine. It should be noted, that for security reasons a mask field has been provided in which the user types his security code and personnel number; however, due to the way in which 'DISPLAY' and 'REPLY' statements work in PL/1, a carraige return and line feed are placed in the data stream and the user must manually roll back the paper in order to type the codes in the field provided. While this is no problem on the 2741 terminal, some other method such as a display erase would have to be used if a CRT were being used.

After verification of the security code and personnel number, a list of functions available to the user is displayed with instructions to enter the line number of the function desired. The desired routine is then called and the

--------------------

(6) See section 3.9 for a complete description of CODECK.

appropriate function performed. Return to the main routine is provided by two means. First, when the user specifies that he is through with a given routine, and second, if any type of error is detected during execution of the routine.

Upon return of control to the main routine, the user is asked if he wishes to continue. This is so he might execute other functions or in the case of an error such as a misspelled record name he might wish to reenter the last routine and enter the correct name. If his reply is yes then the list of functions is again displayed and execution continues. Otherwise, 'END OF JOB' is displayed and the program terminates.

## Section 2.3 Description of the ENTER routine

The purpose of the ENTER routine is to place a new cccurrence of a record into the data base. The routine is called from the main routine by the user entering the line number for the ENTER NEW DATA function.

When control is passed to this module, the user is asked to enter the record name of the record type to be entered. Under IDMS requirements, current set occurrences for all involved sets must be established prior to attempting to place the record in the data base(7). If the record is to be stored and it is the member of a set, the owner record must have been established to be the current of set before the first member record is stored. If an attempt is made to store a record which is the member of a set and the current of set is null, an IDMS error will occur. QL1 assumes that whenever an entire record is being entered, the owner record, if there is one, will have been entered during the same terminal session. The problem of the user attempting to enter a record, which is the member of a set, without first establishing the proper currency could be prevented by including the set name for all sets in which the record is a member in a file and checking to insure that the current of set was not null. The user could also be asked if the owner record for the set had been entered immediately prior to entering this record or if the owner had been stored during a previous terminal session. If it was stored during

--------------------

(7) <u>DML PROGRAMMER'S REFERENCE GUIDE</u> ,(Cullinane Corporation),release 3-1,April,1975,P. 100.

aprevious terminal session, then the user would be asked to enter a value for the owner's calc identifier. That particular record could then be obtained.

After the record name has been entered by the user, control is passed to AD-COMP(8) and a pointer is set to the beginning address of where the record type is located in working storage. Control then returns to the ENTER routine. At this time the symbol table, SYMFILE, is accessed using the key(9). Each elementary data item name is retrieved from SYMFILE, one at a time, and a pointer is set to the beginning address of the data item. The user is asked to enter the value for each item as it is displayed. When the last data item has received a value the record is stored in the data base using the IDMS routine, IDBMSCOM(42). The DML format is:

STORE record-name;

If the store was successful, the error-status is set to '0000', and the record is made: current of run-unit, current of area, current of it's record type, and current of all sets in which it is specified as an owner or a member. It should be noted that if the object record is the owner of a set, the successful store of the record will establish a new set occurrence. The successful store will also cause 'record-name RECORD STORED...' to be displayed at the

--------------------

(8) See Section 3.4 for a complete description of the AD-COMP routine.
(9) See Section 3.2 for a description of the key.

terminal, and the user will be asked if he desires to enter additional records. If he responds with yes, execution cycles back to the beginning of the ENTER routine. If his response is no or if the store attempt was unsuccessful, control returns to the main routine.

## Section 2.4 Description of DELET routine

The DELET routine is used to remove various records from the data base. It is important to note that the previously mentioned assumption that set membership inclusion by MANDATORY AUTOMATIC in all cases(10) is of great importance in understanding the function of this routine. The DML statement used by the DELET routine to delete a given record is:

```
ERASE RECORD (record-name) PERMANENT;(11)(12)
```

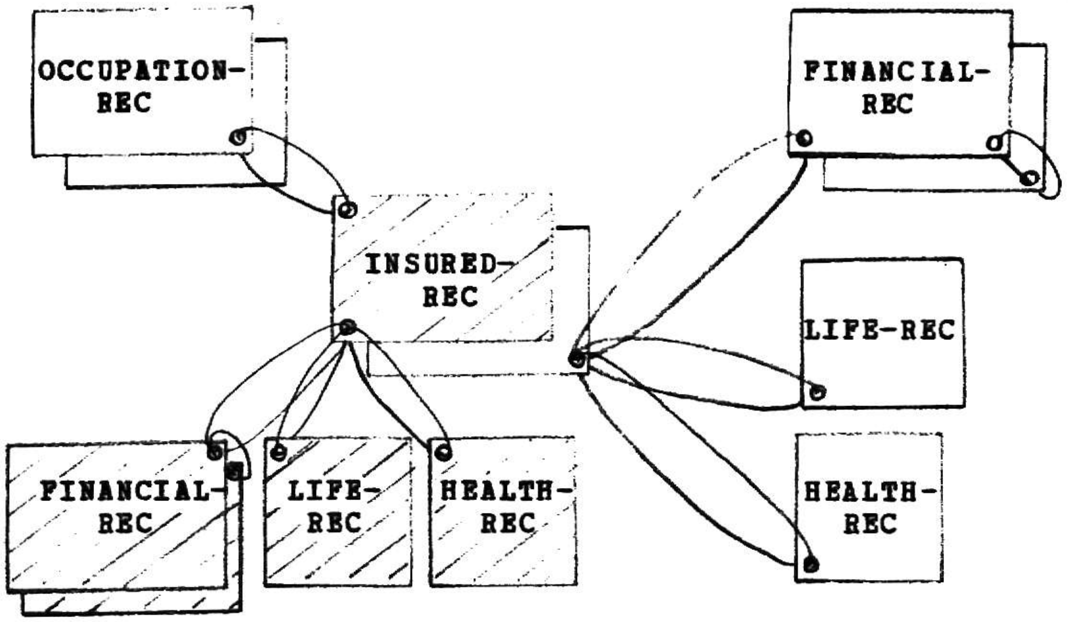
The permanent verb on the end of the statement specifies that the object record as well as all of it's mandatory members are deleted from the data base. Fig 2.2 shows that if an INSURED-REC were being deleted from the test data base(13), the following records:

1. LIFE-REC
2. HEALTH-REC
3. FINANCIAL-REC

would also be deleted since they are MANDATORY AUTOMATIC in the three sets in which INSURED-REC is specified as the owner record.

--------------------

(10) Refer to Section 1.2
(11) _IDMS DML MANUAL SUPPLEMENT_, (Cullinane Corporation), June, 1976, P. 16.
(12) For a complete description of the ERASE statement, refer to the DELETE statement in the IDMS DML PROGRAMMER'S REFERENCE GUIDE.
(13) See Section 4.1 for a complete description of the test data base.



Shaded records indicate those deleted
FIG 2.2

Upon entering the DELET routine, the user is asked for the name of the record type to be deleted. Control is passed to REC-FIND which brings into working storage the appropriate occurrence of the desired record. The record is deleted and the user is asked if there are additional records to delete. If the answer is yes, control cycles back to the beginning of the DELET routine; otherwise control returns to the main routine.

It is not necessary that the object record be placed into working storage prior to deletion. The only requirement is that the object record be made current of the run-unit(14). This could be accomplished by a successful find; however, to prevent having to write a considerable amount of redundant code, REC-FIND was used and that routine always places the object record into working storage.

------------------

(14) DML PROGRAMMER'S REFERENCE GUIDE ,(Cullinane Corporation),release 3-1,April,1975,P. 87.

After an object record has been deleted, the space and the database key are available for reuse. This also applies to any records which were deleted because they were MANDATORY AUTOMATIC members of a set which was owned by the deleted object record.

## Section 2.5 Description of MODIFY routine

The MODIFY routine is used to change the value of a given data item in a record which has been previously stored in the data base. Control is passed to the MODIFY routine from the main routine. Upon entry into the MODIFY routine, the user is asked to enter the name of the record type to be modified. REC-FIND is called which brings the correct object record into working storage.

Once the correct record is in working storage, the user is asked to provide the data item name for that data item which is to be modified. SYM-SEARCH is called and a pointer to the address in working storage for the data item is returned. the user is then asked to input the new value for that data item. The new value is placed into working storage and the user is asked if there are anymore data items to be modified. If there are, These items are modified. After all data items for that record which are to be modified have been changed, the statement

MODIFY record-name;

causes the values of all data items of the object record to be replaced with the values from like-named data items from working storage(15).

If the data item which is to be modified is a CALC identifier then the object record may be found, after

---

(15) Refer to DML PROGRAMMER'S REFERENCE GUIDE.

execution of the modify statement, by refering to the new CALC value. If the data item to be modified is defined as an ascending/decending control item, execution of the modify statement will cause the intra-set occurrence position of the object record to be examined and readjusted if necessary.

## Section 2.6 Description of INFO routine

The INFO routine is used to retrieve information from a given record. The user has the option of displaying a single item of information or an entire record. Current implementation uses a routine named REC-DISPLAY to display entire records. It is assummed that if more than one or two items were going to be displayed, the user would have a report routine which would format the information to be displayed. REC-DISPLAY was included only as a temporary routine to be used during testing and debugging.

As with the other major modules, the first action upon entry into the INFO routine is to ask the user for the record name. REC-FIND is called and the desired record is brought into working storage. The user is then asked if he wants the entire record displayed. If so, control is passed to the REC-DISPLAY routine and each data item along with it's current value is displayed. If the user indicates that only a single item is to be displayed, he is asked for the data item name he wishes displayed. SYM-SEARCH is used to set a pointer to the data item desired, and both the data item name and it's current value are displayed.

After displaying a single data item the user is asked if there is another data item in this record he wishes to have displayed. If there are no more items from the current record to be displayed, the user is asked if there are any other records which contain information to be displayed. If so, control cycles back to the beginning of the routine; otherwise, control returns to the main routine.

## Section 3.1 General

This chapter will explain in detail the various files used by QL1 which are not part of the IDMS file system, the various support routines mentioned in previous chapters. SYMBILD, a program used to build SYMFILE, will be discussed in depth.

## Section 3.2 File Structure and Maintenance

In addition to the normal files provided by the IDMS routines, QL1 has need for several files containing such information as record structures, data item attributes, record names, set relationships, and security codes and personnel numbers.

The major file used is SYMFILE. SYMFILE is basically a symbol table, and is implemented as a direct access, regional 2 (16), keyed record. Having been implemented in IBM's Conversational Monitoring System (CMS)(17) environment, the keys are located in a directory at the end of the file(18). The key is derived from a sixteen character record name concatenated with a fixed binary(15) integer. The integer is converted into character (9), thereby giving a keylength of 25 characters. Each time an access is made into SYMFILE the key must be provided. If that particular key is not found an ON KEY condition will prevail(19). Information contained in SYMFILE is:

1. An elementary data item name.
2. A character, C or F, to designate whether the data item attribute is character or numeric.
3. The number of bytes of storage required to store the item.

--------------------

(16) For further information on this type of record the reader should consult PL/1(F) PROGRAMMER'S GUIDE (IBM CORPORATION),September,1972,Chapter 11.
(17) IBM Virtual Machine Facility/370: CMS User's Guide ,(IBM Corporation),release 3,1st ed.,February, 1976.
(18) Refer to Section 3.3
(19) PL/1(F) LANGUAGE REFERENCE MANUAL ,(IBM Corporation),5th ed.,December,1972,P. 314.

4. The offset, in bytes, from the beginning address of the record structure itself, to the beginning address of the data item. See fig 3.2.1.

| | | | |
|---|---|---|---|
| data item name | C | no of bytes | offset |

fig. 3.2.1 SYMFILE data structure

The program used to build SYMFILE is SYMBILD(20). Input required is the name of the record type, the record structure, and an 'end' card (see fig. 3.2.2). This is required for each record type.

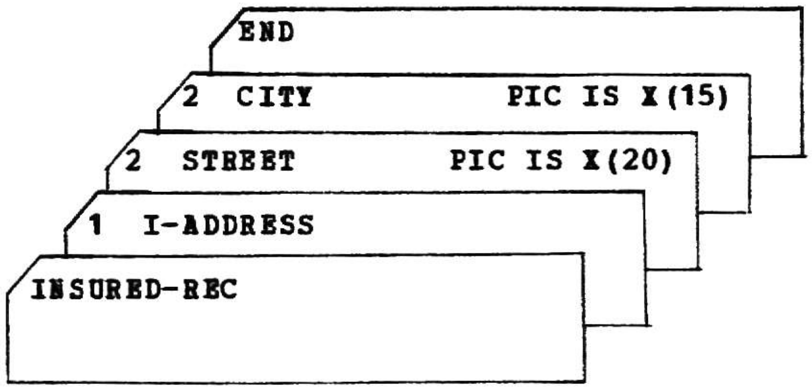


fig. 3.2.2

SYMBILD is required to be run anytime there is a change to the structure of an existing record or a new record is added to the data base. See Appendix A for the source listing for SYMBILD.

-------------------

(20) Refer to section 3.3.

IDLIST is a file which is used by various routines needing to retrieve a specific record. Information contained in IDLIST (see fig's. 3.2.3 and 3.2.4) is the record name, if the record has a location mode of CALC(21) then the CALC identifier is given; otherwise this field is left blank. The next two fields are only used if the record does not have a CALC identifier, i.e. records which are members of a set and stored using the VIA(22) location mode. These two fields would then contain first, the record name of the owner of the set and second, the set name. The reason for this type of information will become apparent as soon as the reader sees the calling formats to the various IDMS routines.

| | | |
|---|---|---|
| INSURED-REC | SSAN-ID | |

fig. 3.2.3 Record with a CALC identifier.

| | | | |
|---|---|---|---|
| LIFE-REC | | INSURED-REC | INSURED-LIFE-SET |

fig. 3.2.4

IDLIST only needs to be updated if a new record is added to

-------------------

(21) <u>DML PROGRAMMER'S REFERENCE GUIDE</u> ,(Cullinane Corporation),release 3-1,April,1975,P.16.
(22) Op. Cit.,P.18.

the data base or a new set is defined. The current method of updating is by using the edit commands in CMS. If this file were to be maintained in an OS dataset, then other means would have to be taken to update the file when necessary.

SECMAT is the file containing the security codes and personnel numbers for all users. See figure 3.2.5. It is a sequential file which contains a six digit personnel code followed by a six digit security code. This file would be updated any time a new user is to be authorized access to the data base or an authorized user were no longer allowed access to the data base. For security reasons, this file should have very limited access. Updating is currently the same for this file as for IDLIST, using the CMS edit commands.

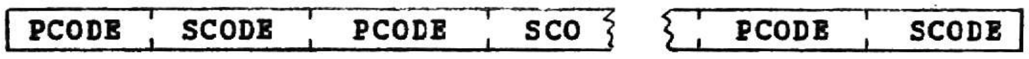


fig. 3.2.5 SECMAT data structure.

The last file required by QL1 is the RECNAMS file. RECNAMS is used by a PL/1 preprocessor routine to generate text for the AD-COMP routine. RECNAMS contains a list of all record names and is updated whenever a record type is added or deleted in the SCHEMA. To be used in the AD-COMP routine, RECNAMS had to be a member of a partitioned dataset. The dataset name is MACLIB and only has one member which is the RECNAMS file.

## Section 3.3 Description of SYMBILD Program

The program SYMBILD is a separate program used to build SYMFILE(23) As with QL1, SYMBILD is implemented in PL/1. The program consists of a main routine and three subroutines. The hierarchy is shown in fig. 3.3.1.

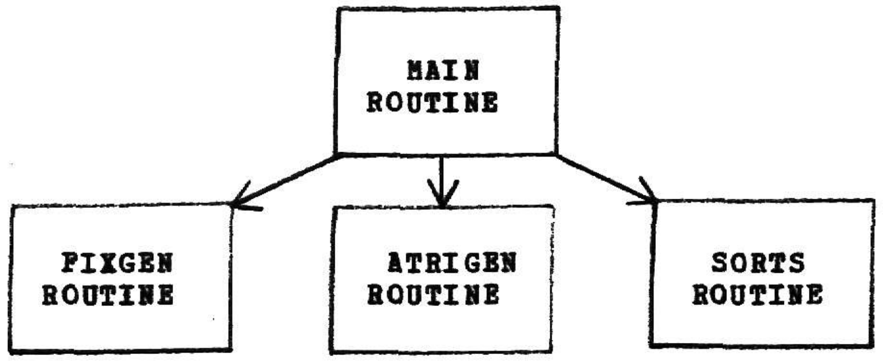


fig. 3.3.1 Hierarchy of SYMBILD Program.

SYMFILE is a regional(2), direct access, keyed record. The key is comprised of two items: the name of the record , and a fixed binary(15) counter. The record name is input by card. As each data item is input, various computations are performed and the information is written into SYMFILE. The counter is incremented for each data item. This gives the unique key for each item. The counter is converted into a character string of length nine(24) and is concatenated onto the end of the record name, which is a character string of length sixteen, to give a keylength of twenty-five. Figure 3.3.2 shows the SYMFILE key form.

--------------------

(23) Refer to Appendix A for the source listing.
(24) <u>PL/1(F) LANGUAGE REFERENCE MANUAL</u> ,(IBM Corporation),5th ed.,December,1972,PP.270-280.

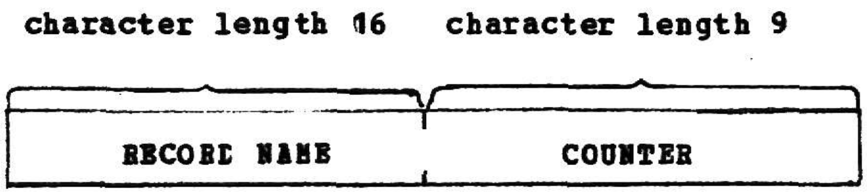


fig. 3.3.2 SYMFILE key description.

SYMBILD reads in a card which contains the name of the record whose data items follow that card. Each data item name is read into the variable 'NAME'. The item is checked to determine whether it is character or numeric, and the variable 'ID' is set to either a 'C' or 'F' respectively. The next variable,'VALUE', is calculated by determining the number of bytes of storage that the data item requires. The last variable, 'OFFSET', is the number of bytes from the beginning address of the structure to the beginning address of the data item.

The routine SORTS is a bubble sort and was included so that a binary search technique could be used in the SYM-SEARCH routine. This technique was not implemented, but by having SYMFILE sorted the capability for a binary search is there.

## Section 3.4 Description of AD-COMP routine

The AD-COMP routine is rather unique in that it allows the beginning address of each record type to be calculated, even if different record structures are implemented, without explicitly re-writing the source code. This is done by means of the PL/1 preprocessor. The source code for QL1 utilizes a preprocessor macro, AD-CALC(25) , to generate the code for AD-COMP.

In order to be able to calculate the address of a given data item, the beginning address of the structure must be known. PL/1 has a builtin function, ADDR, which returns the twenty-four bit address of a variable. To get the beginning address of INSURED-REC, the following code is used:

```
IF RECNAM = 'INSURED-REC' THEN RPTR =
   ADDR(INSURED-REC);
```

RPTR will then contain the twenty-four bit address of INSURED-REC. It should be noted that RPTR must be declared as a pointer variable; otherwise an error will occur when an attemp is made to assign the address of INSURED-REC to it.

AD-COMP, after having statements generated for it by AD-CALC, consists of a series of IF-THEN-ELSE statements for each record name. The list of record names is contained in the RECNAMS file. RECNAMS is INCLUDED text in the AD-COMP routine(26). Input into AD-COMP is the name of the record

--------------------

(25) Refer to section 3.5
(26) For a complete discussion of INCLUDED text, the reader

whose address is to be found. Output from the routine is the address of the record and a return code. The return code is set to one if the record was found; otherwise the return code is set to five.

---

may wish to refer to PL/1-(F) PROGRAMMER'S GUIDE ,(IBM Corporation),9th ed.,September,1972,P.61.

## Section 3.5 Description of AD-CALC routine

The AD-CALC routine is a preprocessor routine used to generate code for the AD-COMP routine. The calling format is:

```
AD-CALC(NAMES);
```

The argument, NAMES, comes from the INCLUDED text RECNAMS which is the file containing a list of all record names to be used. Where %INCIUDE RECNAMS is located in the source code, the following type of code is inserted by the AD-CALC routine:

```
%NAMES='INSURED-REC,OCCUPATION-REC,...,*';
```

Output from AD-CALC is a character string of unspecified length which is the IF-THEN-ELSE statements for AD-COMP.

AD-CALC searches the input string for a record name and then using that name builds the string, 'IF RN='record-name' THEN RPTR=ADDR(record-name);ELSE IF ...;. The asterisk is used to denote the end-of-file. When the asterisk is found the entire output string is returned and placed in the AD-COMP source code. This is the source code for the PL/1 compiler.

## Section 3.6 Description of ID-FIND routine

The ID-FIND routine is used to find the CALC identifier for a given record. If the object record does not have a CALC identifier the owner record is found and its CALC identifier is returned. The calling format is:

```
CALL ID-FIND;
```

ID-FIND takes the current record name and using a sequential search attempts to find the record name in IDLIST. If the record name is not found, the return code is set to three; otherwise, a check is made to see if the object recode has a CALC identifier. If the object record does have a CALC identifier then the variable, ID, is given the value of the CAIC identifier. For example, INSURED-REC has a location mode of CALC and the CALC identifier is SSAN-ID. If ID-FIND were called and the current value of RN were 'INSURED-REC', then AD-COMP would return the variable, ID, with the value of 'SSAN-ID'.

If the object record does not have a CALC identifier, one may assume that it is a member of a set. IDLIST contains the owner record name and set name for all records which do not have CALC identifiers. When it is determined that the object record does not have a CALC identifier a flag is set to indicate that the value of RN has been changed. RN gets the value of the owner record name of the set. SN gets the value of the set name. ID-FIND then begins the sequential search again, this time with the owner record name(27).

The search will now result in ID getting the value of the CALC identifier for the owner record.

REC-FIND is the only routine which calls ID-FIND and the object record name is stored in a temporary variable prior to calling AD-COMP. if the record name is changed by the AD-COMP routine, the original record name will not be lost.

--------------------

(27) The reader is reminded of the assumption that all records designated as owners would have CALC identifiers.

## Section 3.7 Description of SYM-SEARCH routine

The SYM-search routine is an internal procedure used to find a data item in the file SYMFILE and return the address of where the data item is located in working storage. SYM-SEARCH can be called by the MODIFY, INFO, and REC-FIND routines, and the calling format is:

CALL SYM-SEARCH;

Input values for SYM-SEARCH are: RN, ID, and RPTR. RN is is the name of the record which contains the name of the data item being searched for. ID is the name of the data item, and RPTR is the beginning address of the structure for the object record.

Output values for SYM-SEARCH ARE: S, N, P, and RCODE. S will contain either a character 'C' or 'F' depending on the attribute of the data item. N will contain the number of bytes of storage required for the item. P is the beginning address, P=ADDR(id), of the data item, and RCODE returns a value of one if the data item was found or a value of two if the data item was not found.

When control is passed to SYM-SEARCH the value of ID is searched for in SYMFILE, using the key 'RN || CNT'. CNT is simply set to one and then incremented each time the data item name for that specific key does not match ID. If ID is not found for that record name then RCODE is set to two; otherwise, the value of offset is taken from SYMFILE and added to the value of RPTR. The new address is assigned to the base pointer, P. P now points to the beginning address of the data item desired. Fig's 3.7.1 thru 3.7.3 show this pointer assignment for the following structure:

```
1   INSURED-REC
    2   SSAN          CHAR(9)
    2   AGE           CHAR(2)
    2   NAME          CHAR(25)
```

assumming the following input:

```
RN='INSURED-REC'
ID='AGE'
RPTR=ADDR(INSURED-REC)
```

| <u>NAME</u> | <u>ID</u> | <u>VALUE</u> | <u>OFFSET</u> |
|---|---|---|---|
| AGE | C | 2 | 9 |
| NAME | C | 25 | 11 |
| SSAN | C | 9 | 0 |

fig. 3.7.1 Contents of SYMFILE for INSURED-REC

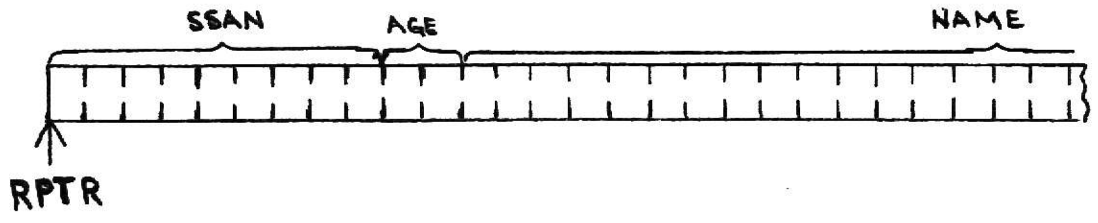


fig. 3.7.2 Position of RPTR upon entry to SYM-SEARCH

Note that P now points to the beginning address of the data item, AGE.

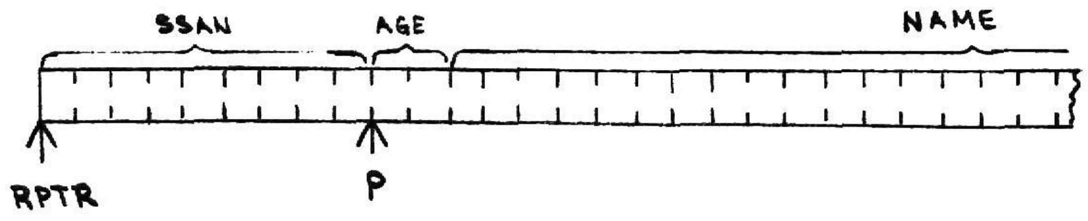


fig. 3.7.3 Position of P after offset added to RPTR

After the base pointer P is set, S is set to the value of ID from SYMFILE and N is set to the value of VALUE from SYMFILE. In the above example, S would be set to 'C', and N would be set to '2'. This would terminate the SYM-SEARCH routine and control would return to the calling routine.

## Section 3.8 Description of REC-FIND routine

The REC-FIND routine is an internal procedure designed to bring into working storage a specific record from the data base. REC-FIND may be called by the DELET, MODIFY, or INFO routines, and the calling format is:

CALL REC-FIND;

The only input to REC-FIND is RN which contains the name of the object record. Upon returning to the calling routine, RN will not have been changed, and a return code of either one, two, or three will have been set. A return code of one indicates the object record was retrieved from the data base and placed into working storage. A return code of two indicates that the unique identifier required to retrieve a record which is the member of a set or a CALC identifier could not be found. The most likely cause for a return code of two would be a misspelled identifier. A return code of three indicates that the object record name could not be found. This could be caused by a misspelled record name or if the particular record is not included in the subschema.

Upon entering REC-FIND the record name is stored in a temporary variable. This is due to the possibility that RN might be changed in the ID-FIND routine. After the record name is stored, calls are made to the ID-FIND, AD-COMP, and SYM-SEARCH routines. Upon returning from SYM-SEARCH the CALC identifier for the object record or the owner of the set if the object record does not have a CALC identfier will have been found, and the based pointer, P, will be pointing to the beginning address of the CALC identfier. The user is then asked to enter a value for the CALC identifier. The value is then stored and the record with that CALC identifier is obtained. If this record is the object record the routine terminates.

If the record placed into working storage was not the object record, the object record must still be obtained from the data base. This is determined by checking the variable, FLAG. If FLAG = 1 then RN was changed. At this point RN is set to the value of TEMP, which is where the object record name was saved. The user is then asked to enter a data item name which may be used to uniquely identify the desired record. AD-COMP and SYM-SEARCH are then called to set the based pointer, P, to the data item name that the user supplied. The user is asked to provide a value for the data item name. Using the value supplied, each record in the set is obtained and checked to see if the data item value matches the one provided by the user. When a match is found, the routine terminates. If a match is not made, the user is so notified and the routine terminates.

## Section 3.9 Description of CODECK routine

The CODECK routine is an internal procedure designed to verify the security code and personnel code entered by the user. The main routine is the only routine which calls CODECK, and the calling format is:

```
CALL CODECK;
```

CODECK uses two variables: SCODE and PCODE. SCODE is the security code for the user, and PCODE is the user's personnel code. Both codes must match those contained in the file SECMAT.

The operation of CODECK is very straight forward. Since SECMAT is a sequential file, CODECK takes PCODE and sequentially searches the file SECMAT looking for a match. If a match is made, then the next six digits in SECMAT must match SCODE.

If either PCODE or SCODE do not match in the file SECMAT, a message is sent to the user telling him which code did not match. The user has three tries to match PCODE and SCODE before the CODECK routine returns to the main routine and execution of QL1 is terminated.

## Section 3.10 Description of ERROR routine

The ERROR routine is an internal procedure designed to produce a specific error message based on the value of the return code, RCODE. The routine consists of several IF-THEN-ELSE statements. The return code is checked and if it matches one of the IF-THEN-ELSE statements the error message is displayed; otherwise the message 'RETURNING TO MAIN ROUTINE...' is displayed and the routine terminates.

## Section 3.11 Description of PTR-ADJUST routine

The PTR-ADJUST routine is very siailiar to the pointer setting used in the SYM-SEARCH routine. The main difference is that PTR-ADJUST is used to right-justify numeric values while SYM-SEARCH always left-justifies the values it points to. PTR-ADJUST may be called by the ENTER, MODIFY, and REC-FIND routines.

When character data is entered into working storage it is left-justified. By using the SYM-SEARCH routine the based pointer, P, points to the appropriate byte to enter the first character. However, when entering numeric data it must be entered right-justified. This means the length of the data being entered must be determined, the amount of storage for the item must be determined, and if the number of bytes of storage is greater than the number of bytes the data consists of, then the based pointer, P, must be adjusted. Fig. 3.11.1 shows the storage location for a data item, AMOUNT-DUE. The picture attribute for this item is 9(3)V99.

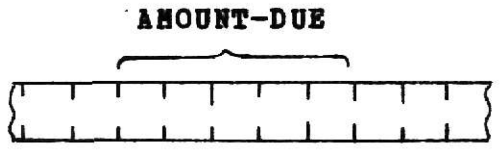


fig. 3.11.1 Storage location for AMOUNT-DUE

The picture character, V, indicates that an assumned decimal point belongs between the third and fourth digits. If the value 1000 were entered left-justified as shown in figure 3.11.2, it's numeric value would be 100.0;

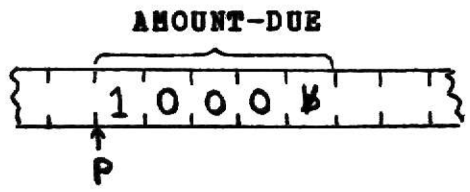


fig. 3.11.2 AMOUNT-DUE left-justified

However, it's proper value, 10.00 is obtained when it is placed into storage right-justified as shown in figure 3.11.3.

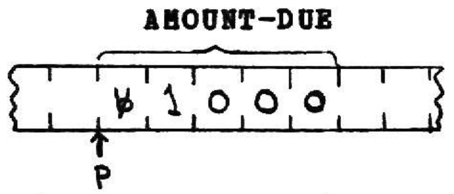


fig.3.11.3 AMOUNT-DUE right-justified.

The adjustment of P is made by subtracting the number of bytes the input value contains from the amount of storage reserved for the item. This difference is added to the value of P. In the above example, there were five bytes reserved for AMOUNT-DUE, but the input value only consisted of four bytes. The difference of one byte was added to P so that it would point to the appropriate first byte of storage.

# CHAPTER FOUR TEST PROCEDURES

## Section 4.1 Data Base Description

The data base, DATATEST, represents some of the information that might be contained in the data base of a small insurance corporation. Designed only for testing purposes, DATATEST lacks many of the features and design concepts that might be expected in a data base of this type. Figure 4.1.1 shows a graphic representation of the various records and set relationships of DATATEST.

INSURED-REC is the main record of the data base: the record on which all others are based. It contains information directly related to the insured, i.e. name, address, social security number, age,.... INSURED-REC is the owner of four sets:

1. FIN-DATA-SET.
2. INSURED-LIFE-SET.
3. INSUR-HEALTH-SET.
4. INSURED-MED-SET.

and a member of the OCC-DATA-SET. Current implementation requires, for currency purposes, that the user know if a record is both the owner of a set and a member of one or more sets. At the time the data base was designed it was not realized the amount of difficulty that would be involved in implementing a system that would properly store and retrieve records which are both owners and members of various sets. Consequently, when an INSURED-REC is placed in the data base, a new OCCUPATION-REC must also be placed in the data base. Figure 4.1.2 shows one OCCUPATION-REC for each INSURED-REC. This is the way QL1 is currently implemented. Figure 4.1.3 shows one OCCUPATION-REC for several INSURED-RECs. This is the way the records should be stored if the record is both an owner and a member.

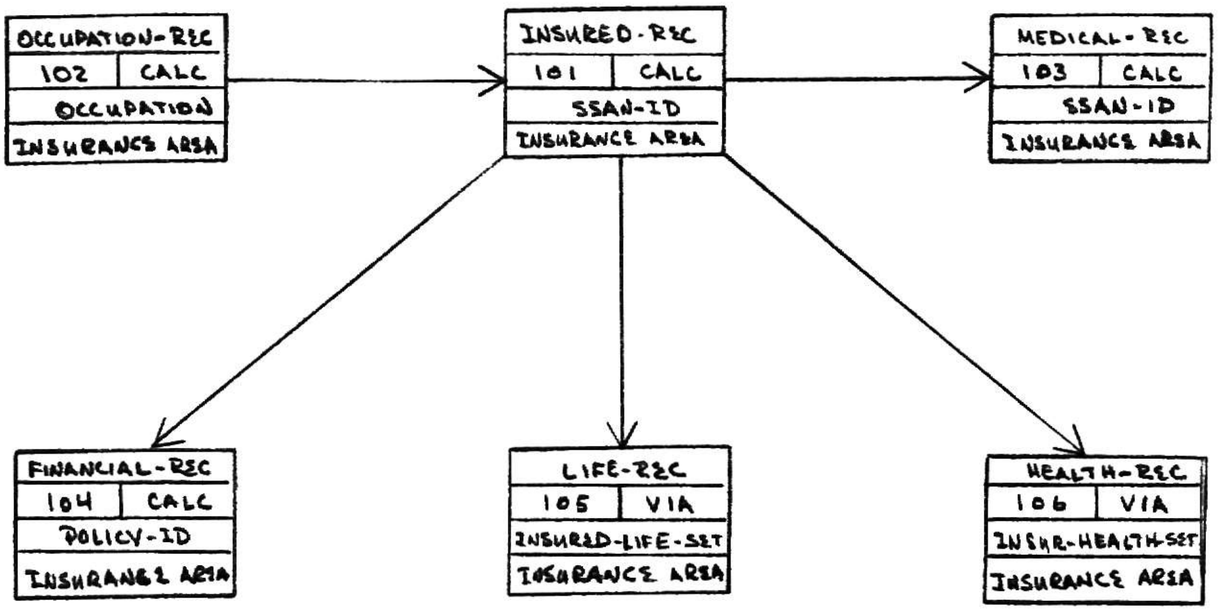


Figure 4.1.1 DATATEST Data Base

The reason for making OCCUPATION-REC the owner of the OCC-DATA-SET was so the user could have a report generated which would be able to efficiently search the data base and list all of the insured by occupation.

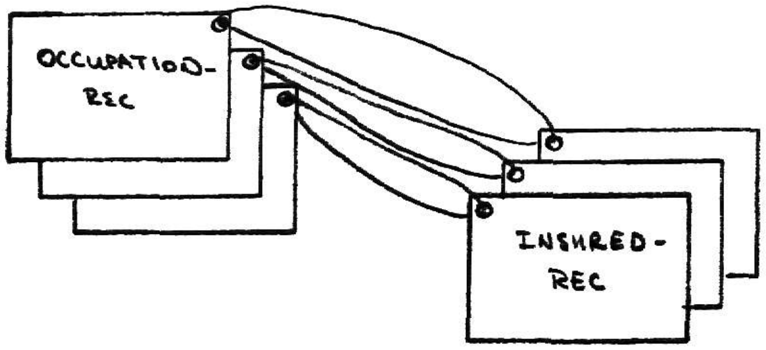


fig. 4.1.2 Current implementation of QL1.

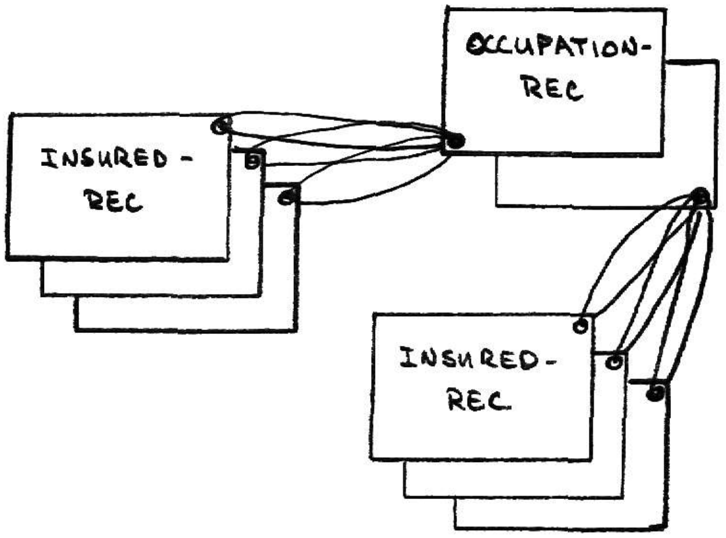


fig. 4.1.3 Possible implementation of QL1.

OCCUPATION-REC should only contain the occupation, but currently shows the place of employment also.

The INSURED-LIFE-SET consists of INSURED-REC and LIFE-REC. LIFE-REC contains information relating to a specific life insurance policy. Figure 4.1.4 shows how the members of the INSURED-LIFE-SET are stored.

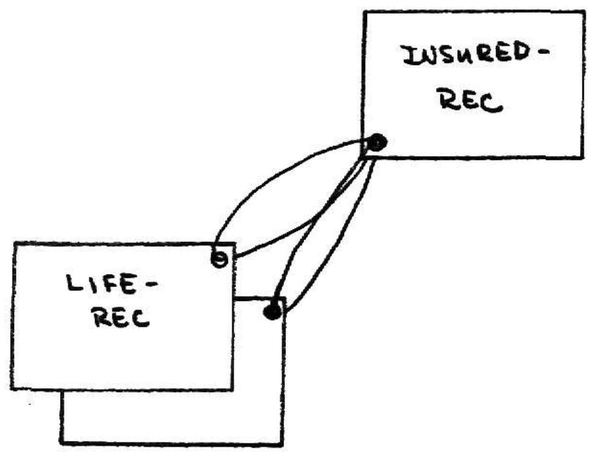


fig. 4.1.4 Implementation of INSURED-LIFE-SET.

One insured may own several life policies. The policy number is the unique identifier to a specific LIFE-REC record.

The INSUR-HEALTH-SET is similiar to the INSURED-LIFE-SET except it is for health insurance policies. The policy number is the unique identifier and the set is implemented the same as shown in figure 4.1.4.

The INSURED-MED-SET contains information relating to the medical status of the insured. The record is stored CALC using the SSAN-ID as the CALC identifier. The reason for not storing this record VIA was because it was not expected to be utilized as much as LIFE-REC and HEALTH-REC.

The FIN-DATA-SET contains the FINANCIAL-REC as it's member and shows the financial information for each life policy and each health policy. FINANCIAL-REC is stored CALC on a policy number so that it might be easily retrieved when financial information for a specific policy is desired.

## Section 4.2 Test Senario

Testing of QL1 consisted of entering and deleting entire records, modifing specific data items, and retrieving data from various records.

Appendix B shows a sample of the final testing session. The test was conducted as follows:

1. All of the records for a given insured were entered into the data base.
2. A member of a set was deleted then all of the records for a given insured were deleted.
3. Specific data on a member of a set was modified.
4. Information was displayed throughout the test to show that data was properly entered, deleted, or modified as the case may be.

Considerable other testing was conducted as each individual module was implemented. This testing is not shown in the report.

## Section 5.1 Objectives

The objective of this project was to design and implement a prototype query language which could be used with the Integrated Database Management System(IDMS). This chapter discusses how well the design and implementation of QL1 met that objective.

The evaluation will consider such points as portability, extensibility, and efficiency. Since this is a truly subjective evaluation, I will attempt to cover both the good and bad features of the language. Numerous observations made during both the implemantation and and testing phases caused thought to be given to major design changes. However, since this project was limited mainly by a time factor the changes were not implemented. Rather, a discussion of the possible changes is included in the evaluation.

## Section 5.2 Evaluation of the Prototype Language

As computer networks and distributed data bases come more and more in to use, the portability of software which interfaces with the network or data base becomes a greater concern. In the initial design phase of QL1 it was felt that the language should be portable to as many systems as possible. This decision immediately ruled out using an assembler language as the implementation language. Of the high level languages suitable for use, COBOL and PL/1 appeared to be the two most likely candidates. COBOL, because of it's wide spread usage throughout the business world would be the most likely choice, but because of the time limitation and the fact that I was not familiar with COBOL, it was decide to go with PL/1. In addition to the fact that PL/1 is not as widely used as COBOL, another major drawback is that some of the PL/1 features used in QL1 are only available on versions of the PL/1 compiler that have a preprocessor available.

Being a computer-prompted language and modular in design allows considerable room for expanding and improving the features of QL1. As previously mentioned, numerous ideas for improvment came during the testing and implementation phases. One such improvment could be made in the ENTER routine. Since IDMS requires that the owner of a set be the current of set when the object record is stored, the sequence in which the user enters records becomes extremely important. The potential problem of a user attempting to enter a record without first having the proper currency status could be alleviated by including the owner and set names for all records, not just those without CALC identifiers, in the file IDLIST. At the time the user entered the name of the record to be entered, IDLIST could be checked to see if the record was a member of a set or not. If it was, the owner record's name could be retrieved and the user asked to provide the value of the CALC identifier for the owner. The owner could then be obtained and proper placement of the object record insured. If the owner had not yet been stored, the IDMS error-status could be checked and the user advised to store the owner record prior to storing the object record. This procedure would cause considerable cverhead for the storing of multiple records; however, the security provided in storing individual records would be greatly enhanced. Also, the probability of multiple records being stored through the facilities of QL1 is less than the probability of individual records being stored. If an entire series of records were being stored it would more likely be done in a batch mode rather than interactively.

In considering the efficiency of QL1, it can be considered far more efficient, even in its present form, than a batch operation, since it allows for immediate access to the data base. However, a major modification that should be made would be to convert the sequential search technique used in the SYM-SEARCH routine into a binary search technique. SYMFILE is sorted alpabetically within each record type. This was done with the idea of implementing the binary search.

One problem peculiar to IDMS was the use of READY and FINISH statements. A READY statement must be issued before any DML command can be given. After the READY statement is issued data is passed back and forth between the user's working storage and the Data Base Management System (DBMS) buffers. However, the journal, which maintains changes to the data base, is not closed until a FINISH statement is encountered. Therefore, in an interactive environment abnormal termination of the program could cause a loss of all the records which had been entered, deleted, or modified. It was considered that for the sake of security that a READY and FINISH would be placed in the three routines which performed the previously mentioned functions. This would insure that in the event of an adnormal termination of the program, only those records being manipulated by the current routine would be affected. This was, however, considered to have too high an overhead for the small amount of security which it provided. Therefore, only one READY statement is issued at the beginning of the main routine and one FINISH statement just prior to the termination of the program.

## Section 5.3 Conclusions

The purpose of this project was to design and implement a prototype query language which would interface with the Integrated Database Management System. The design requirements were specified in chapter one. Several assumptions were made concerning the design of the data base; mainly due to the fact that the language is a prototype. Most of the assumptions could be changed through expansion and modification of the prototype.

As the features of the language began to emerge, the need for considerable information not readily available became apparent. Most of this information can be obtained and placed in files which may be accessed by QL1 anytime. This includes information required when data items are added or deleted and new record types are created. A major area of further consideration is the problem of data-structure mapping. It was this area that caused considerable consternation when attempting to develop address pointer algorithms.

The problems encountered during the development of QL1 were many and varied. However, it is hoped that the preceeding pages have shown the benefit of such a language to potential users of *IDMS*. Although limited in sophistication, QL1 does meet it's basic design requirements and hopefully provides the framework for further research in the area of query language design.

## APPENDIX A

```
SYMBILD:  PROC OPTIONS (MAIN):
          DECLARE (NUMVAL, TOTVAL, TEMP) FIXED BIN (15),
          CHARVAL CHAR (16) VARYING:
          DECLARE 1 SYMTAB (50),
                    2 ID CHAR (1),
                    2 VALUE FIXED BIN,
                    2 OFFSET FIXED BIN;
          DECLARE (I, L, P) FIXED BIN,
                    COUNT FIXED BIN (15) INIT (1),
                    RECNAM CHAR (16),
                    CARD CHAR (80):  /* INPUT VARIABLE*/
          DECLARE SYMFILE RECORD DIRECT KEYED
          ENV (REGIONAL (2)):
          OPEN FILE (SYMFILE) OUTPUT:

START:    GET FILE (SYSIN) EDIT (RECNAM)  (A(16)):
          ON END FILE (SYSIN) BEGIN:
                    CLOSE FILE (SYMFILE);
                    GO TO TFRM;
                    END:
          COUNT = 1:
          TOTVAL, NUMVAL = 0:
          GET FILE (SYSIN) EDIT (CARD)  (COL (1), A(80)),
          /* THE CHARACTERS 'END' INDICATE THE END OF A
          RECORD TYPE */

COMP:     DO WHILE (SUBSTR(CARD, 1, 3)  ='END');
          CHARVAL = '';
                    I = 1;
                    L = 0;
                    P = 0;

          /* CONT1 TO LAB1 SEARCHES ACROSS THE INPUT TO
          FIND */
          /* THE DATA ITEM NAME. AT THE START OF LAB1,
          THE I */
          /* POINTER POINTS TO THE FIRST CHARACTER IN THE
          DATA */
          /* ITEM NAME. P POINTS TO THE FIRST BLANK FOL-
          LOWING THE */
          /*THE DATA ITEM NAME. */

CONT1:    DO WHILE (L   2):
          DO WHILE (SUBSTR(CARD, I, 1) =' ');
                    I = I + 1;
                    END;
          P = I:
```

```
CK:  DO WHILE (SUBSTR(CARD, P, 1)¬= '  '     );
     P = P + ];
     END;
     IF L = 0 THEN DO;
          I = P;
          L = 1;
          END;
          ELSE L = 2;
          END CONT1;
LAB1:     NAME(COUNT) = SUBSTR(CARD,I,P-I);
    .R = VERIFY (SUBSTR(CARD,P), ' ');

     /* IF R = 0 THEN THIS IS NOT AN ELEMENTARY DATA ITEM.*/

     IF R = O THEN DO;
          OFFSET(COUNT) = TOTVAL;
          ID(COUNT) = ' ';
          VALUE(COUNT) = O;
          END;
          ELSE DO;
               P = INDEX(CARD,'(');
               IF P¬= O THEN DO;
                   I = INDEX(CARD,'FIXED');
                   IF I ¬= O THEN DO;
                   ID(COUNT) = 'F';
                   CALL FIXGEN;
                   VALUE(COUNT) = NUMVAL;
                   END;
                   ELSE DO;
                        CALL ATRIGEN;
                        IF SUBSTR(CARD,I+1) ¬= ' ' &
                           SUBSTR(CARD,I+1,1) ¬= '.'
                           THEN DO;
                             TEMP = NUMVAL;
                             NUMVAL = O;
                            I = I + 1;
                            IF SUBSTR(CARD,I,1) = 'V' THEN
                               I = I + 1;
                            DO WHILE ((SUBSTR(CARD,I,1)¬=' ') &
                                      (SUBSTR(CARD,I,1)¬= '.'));
                                 NUMVAL = NUMVAL + 1;
                                 I = I + 1;
                                 END;
                            NUMVAL = NUMVAL + TEMP;
                            ID(COUNT) = 'F';
                            VALUE(COUNT) = NUMVAL;
                            TOTVAL = TOTVAL + NUMVAL;
                            END;
                       ELSE DO;
                         ID(COUNT) = 'C';
                         VALUE(COUNT) = NUMVAL;
                         END;
                       END;
```

```
                                    END;
                                    ELSE DO;
                                        I = INDEX(CARD,'9');
                                        NUMVAL = 0;
                                        DO WHILE((SUBSTR(CARD,I,1)¬=' ') &
                                               (SUBSTR(CARD,I,1)¬='.'));
                                            NUMVAL = NUMVAL + 1;
                                            I = I+1;
                                            END;
                                        ID(COUNT) = 'F';
                                        VALUE(COUNT) = NUMVAL;
                                        OFFSET(COUNT) = TOTVAL;
                                        TOTVAL = TOTVAL + NUMVAL;
                                        END;
                                    END;
                                COUNT = COUNT + 1;
                                GET FILE(SYSIN) EDIT (CARD) (COL(1),A(80));
                                END COMP;
                            CALL SORTS;
                            GO TO START;

FIXGEN: PROC;
    P = P+1;
    DO WHILE((SUBSTR(CARD,P,1)¬=',')&(SUBSTR(CARD,P,1)¬=')'));
        CHARVAL = CHARVAL || SUBSTR(CARD,P,1);
        P =  P+1;
        END;
    NUMVAL = CHARVAL;
    OFFSET(COUNT) = TOTVAL;
    TOTVAL = TOTVAL + CEIL((NUMVAL+1)/2);
    END FIXGEN;
```

```
ATIRGEN: PROC;
     I = P + 1;
     DO WHILE (SUBSTR(CARD,I,1)¬=')');
          CHARVAL = CHARVAL   SUBSTR(CARD,I,1);
          I = I + 1;
          END;
     NUMVAL = CHARVAL;
     OFFSET(COUNT) = TOTVAL;
     TOTVAL =  TOTVAL + NUMVAL;
     END ATRIGEN;

SORTS: PROC;
     DECLARE 1 TEMP,
               2 NAME CHAR (16),
               2 ID CHAR(1),
               2 VALUE FIXED BIN,
               2 OFFSET FIXED BIN;
          SORTED = 0;
          J,K = COUNT - 1;
BUBBLE:   DO WHILE((SORTED=0)&(J>=2));
               SORTED = 1;
               DO I = 2 TO J;
                    IF SYMTAB.NAME(I-1) > SYMTAB.NAME(I)
                         THEN DO;
                         TEMP = SYMTAB(I-1);
                         SYMTAB(I-1) = SYMTAB(I);
                         SYMTAB(I) = TEMP;
                         SORTED = 0;
                         END;
                    END;
               J = J - 1;
               END BUBBLE;
RITE:     DO COUNT = 1 TO K;
               TEMP = SYMTAB(COUNT);
               WRITE FILE (SYMFILE) FROM (TEMP)
                              KEYFROM (RECNAM || COUNT);
               PUT SKIP FILE (SYSPRINT) LIST (TEMP);
               END RITE;
          END SORTS;
TERM:     END SYMBILD;
```

```
DATATST:  PROC OPTIONS (MAIN);
          DCL SYMFILE RECORD DIRECT KEYED UPDATE
               ENV(REGIONAL(2));
          DECLARE (I,J,CNT,CONT,TEST1,TEST2,FLAG,RCODE)
                                                  FIXED BIN,
                  (S,NUM) CHAR (1),
                  (PCODE,SCODE) CHAR (6),
                  ANS CHAR (3) VARYING,
                  (K,L) FIXED BIN (31),
                  (RN,SN,ID,TEMP) CHAR (16),
                  (INPUT,OUTPUT) CHAR (50) VARYING,
                  (ADJ,LEN) FIXED BIN (15),
                  CVAL CHAR (50) BASED (P),
                  NVAL FIXED BIN (31) BASED (P),
                  (T, RPTR) POINTER,
                  MASK BIT (32) INIT('000000001111111
                                     11111111111111111'B),
                  1 TABLE (50),
                    2 NAME CHAR(16),
                    2 ID CHAR(1),
                    2 VALUE FIXED BIN,
                    2 PTR FIXED BIN,
                  1 BUFF,
                    2 NAME CHAR(16),
                    2 ID CHAR(1),
                    2 VALUE FIXED BIN,
                    2 PTR FIXED BIN,
                  FUNCTION(5) CHAR (25) VARYING
                    INIT('1. ENTER NEW DATA.',
                         '2. DELETE DATA.',
                         '3. MODIFY DATA.',
                         '4. INFORMATION RETRIVAL.',
                         '5. REPORT GENERATION.');

%DCL AD_CALC ENTRY (CHAR) RETURNS (CHAR);
%AD_CALC: PROC (NAMES) RETURNS (CHAR);
          DECLARE (NAMES, RSTR,C,TEMP1,TEMP2) CHAR,
                  (I,J,K) FIXED;
          I,K = 1;
          J = 0;
          RSTR,TEMP1, TEMP2 = '';
START:    DO I = 2 TO 500;
               C = SUBSTR(NAMES,I,1);
               IF J = 1 THEN DO;
                    IF C¬= '*' THEN RSTR + RSTR || ';';
                         ELSE I = 500;
                    J = 0;
                    END;
               IF C = '-' THEN DO:
                    TEMP1 = TEMP1 || C;
                    C = '_';
                    ELSE IF C = '.' THEN DO;
                         J = 1;
                         IF K = 1 THEN DO;
                              RSTR = 'IF RN = ''' || TEMP1 ||
                              ''' THEN RPTR = ADDR(' ||
                              TEMP2 || ')';
                              K = 0;
                              END;
                              ELSE RSTR = RSTR || 'ELSE
IF RN = ''' || TEMP1 || ''' THEN RPTR = ADDR(' ||
TEMP2 || ')';
                         TEMP1,TEMP2 = '';
                         END;
                         ELSE DO;
                              TEMP1 = TEMP1 || C;
                              TEMP2 = TEMP2 || C;
                              END;
                    END;
               RETURN (RSTR);
          %END AD_CALC;

          DCL (CUSTOMER SUBSCHEMA.DATATEST SCHEMA.DATATST
               PROGRAM) MODE (KSU) DEBUG;
          INCLU E IDMS (GENERIC);
          INCLUDE IDMS (SUBSCHEMA_DESCRIPTION);
 INCLUDE IDMS (SUB_SCHEMA BINDS);

          READY UPDATE;

IDMS_STATUS: PROC;
          IF ERROR_STATUS = '0000' THEN RETURN;
          DISPLAY ('****IDMS ERROR DETECTED****');
          DISPLAY ('PROGRAM NAME = ' || PROGRAM);
          DISPLAY ('ERROR STATUS = ' || ERROR_STATUS);
          DISPLAY ('ERROR RECORD = ' || ERROR_RECORD);
          DISPLAY ('ERROR SET = ' || ERROR_SET);
          DISPLAY ('ERROR AREA = ' || ERROR_AREA);
          DISPLAY ('LAST GOOD REORD WAS ' || RECORD_NAME);
          DISPLAY ('LAST GOOD ARE WAS ' || AREA_NAME);
          DISPLAY ('DML SEQUENCE NUMBER IS ' || DML_SEQUENCE);
          ERROR_STATUS = '1400';
          RCODE = 4;
          END IDMS_STATUS;

LAB1:     RCODE = 2;
          I = 1;
          DO WHILE (RCODE = 2);
               DISPLAY('PLEASE ENTER YOUR SECURITY CODE.');
               DISPLAY('MMMMMM       SSSSSS       KKKKKK        ')
                    REPLY(SCODE);
               DISPLAY('PLEASE ENTER YOUR PERSONNEL NUMBER.');
               DISPLAY('MMMMMM       SSSSSS       KKKKKK        ');
                    REPLY(PCODE);
               CALL CODECK;
               I = I + 1;
               END;
```

```
/* RCODE = 5 INDICATES THAT THREE ATTEMPTS WERE MADE TO
   ENTER THE SYSTEM WITH THE WRONG SECURITY CODE OR PER-
   SONNEL NUMBER*/

          IF RCODE = 5 THEN GO TO TERM;
LAB2:     DISPLAY ('ENTER NUMBER FOR FUNCTION TO BE
               PERFORMED');
          DO I = 1 TO 5;
          DISPLAY(FUNCTION(I));
          END;
LAB3:     DISPLAY(' ') REPLY(NUM);
          IF NUM = '1' THEN CALL ENTER;
             ELSE IF NUM = '2' THEN CALL DELET;
                ELSE IF NUM = '3' THEN CALL MODIFY;
                  ELSE IF NUM = '4' THEN CALL INFO;
                     ELSE IF NUM = '5' THEN CALL REPORT;
                        ELSE DO;
                         DISPLAY('INVALID COMMAND');
                         DISPLAY('RE-ENTER FUNCTION
                                        NUMBER');
                         GO TO LAB3;
                         END:
          IF RCODE > 1 THEN CALL ERROR;

/* THE ABOVE IS AFTER RETURN FROM ONE OF THE MODULES */

          DISPLAY('DO YOU WISH TO CONTINUE? (YES/NO)');

          DISPLAY(' ') REPLY(ANS);
          IF ANS = 'YES' THEN GOTO LAB2;

/*******************************************************/
/*                                                     */
/*                  ENTER ROUTINE                      */
/*                                                     */
/*******************************************************/

ENTER: PROCEDURE;
          DECLARE TEST BIT(1);
          OPEN FILE (SYMFILE);

/* ON KEY CONDITION WILL INDICATE THE LAST DATA ITEM */
/* FOR THAT PARTICULAR RECORD HAS BEEN ACCESSED.     */

          ON KEY (SYMFILE) GOTO E4;
          TEST = '1'B;
E1:       DO WHILE (TEST = '1'B);
               CNT = 1;
               RCODE = 1;
               DISPLAY('ENTER RECORD NAME') REPLY(RN);

/* AD_COMP RETURNS THE BEGINNING ADDRESS FOR THE RECORD */
/* IN STORAGE.                                          */
```

```
          CALL AD.COMP(RN,RPTR,RCODE);
          IF RCODE = 1 THEN DO;
               ALLOCATE CVAL;
               I = P;
E3:            DISPLAY ('PLEASE ENTER THE FOLLOWING INFORMATION');
               DO WHILE ('1'B);
                    READ FILE (SYMFILE) INTO (BUFF)
                                        KEY (RN || CNT);
               IF BUFF.NAME ¬= 'FILLER' & BUFF.ID ¬= ' '  THEN DO;
/* THIS SECTION OF CODE SETS THE POINTER FOR CVAL AND NVAL TO THE     */
/* THE BEGINNING ADDRESS OF THE DATA ITEM TO BE ENTERED.              */

                         UNSPEC(RPTR) = UNSPEC(RPTR) + MASK;
                         K = BIN(UNSPEC(RPTR),31,0);
                         L = BIN(UNSPEC(BUFF.PTR),31,0);
                         UNSPEC(P) = UNSPEC(BIN(K+L,31,0));
                         N = BUFF.VALUE;
LOOP:                    DISPLAY ('ENTER' || BUFF.NAME) REPLY(INPUT);
                         IF BUFF.ID = 'C' THEN SUBSTR(CVAL,1,LEN) =
                                                    INPUT;
                         ELSE DO;
                              CALL PTR_ADJUST;
                              IF RCODE = 1 THEN SUBSTR (CVAL,1,LEN) =
                                                    INPUT;
                              ELSE DO;
                                   DISPLAY('LENGTH OF  '|| ID ||
                                   'CONTAINS' || ADJ || 'CHARACTERS'
                                   'TOO MANY.');
                                   GO TO LOOP
                                   END;
                              END;
                         END;
                    CNT = CNT + 1;
                    END;
E4:            P = T:
               FREE CVAL;

          /* STORE RN  */
               SUBSCHEMA_CTRL.DML_SEQUENCE = 0008;
               CALL IDMSP1F(ADDR(IDBMSCOM(42)),RN);
               CALL IDMS_STATUS;
               IF RCODE = 1 THEN DO;
                    DISPLAY (RN||'RECORD STORED...');
                    DISPLAY ('DO YOU WISH TO ENTER ANYMORE RECORDS?');
                                        REPLY (ANS);
                    IF ANS = 'NO' THEN TEST = '0'B;
                    END;
                    ELSE TEST = '0'B;
          END E1;
E6:  CLOSE FILE (SYMFILE);
     END ENTER;
```

```
/*****************************************************************/
/*                                                               */
/*                       DELETE ROUTINE                          */
/*                                                               */
/*****************************************************************/

DELET: PROCEDURE;
     RCODE = 1;
     ALLOCATE CVAL;
     T = P;
D1:  DISPLAY('ENTER THE NAME OF THE RECORD TO BE DELETED')
                                                 REPLY (RN);

/* REC_FIND PLACES THE RECORD TO BE DELETED INTO WORKING          */
/* STORAGE.                                                       */

     CALL REC_FIND;
     IF RCODE = 1 THEN DO;
          SUBSCHEMA_CTRL.DML_SEQUENCE = 0013;
          CALL IDMSP1F (ADDR(IDBMSCOM(03)),RN);
          CALL IDMS_STATUS;
          IF RCODE = 1 THEN DO;
               DISPLAY('ARE THERE ANYMORE RECORDS TO DELETE?')
                                                 REPLY(ANS);
               IF ANS = 'YES' THEN GO TO D1;
               END;
          END;
     P = 1;
     FREE CVAL;
     END DELETE;

/*****************************************************************/
/*                                                               */
/*                       MODIFY ROUTINE                          */
/*                                                               */
/*****************************************************************/

MODIFY: PROCEDURE;
     RCODE = 1;
     TEST1,TEST2 = 1;
     ALLOCATE CVAL;
     T = P;
START:    DO WHILE(TEST2 = 1);
          DISPLAY('ENTER NAME OF RECORD TO BE MODIFIED')REPLY(RN);

/* REC_FIND PLACES THE RECORD TO BE MODIFIED INTO WORKING          */
/* STORAGE.                                                        */

CALL REC_FIND;
          IF RCODE = 1 THEN DO;
```

```
                DO WHILE(TEST1 = 1);
                     DISPLAY('ENTER NAME OF ELEMENT TO');
                     DISPLAY('BE MODIFIED.') REPLY(ID);
/*SYM_SEARCH SET A POINTER TO THE ADDRESS OF THE DATA ELEMANT */
/* TO BE MODIFIED.                                              */

                     CALL SYM_SEARCH;
                     IF RCODE ¬= 1 THEN DO;
                          Z = 0;

LOOP:                     DISPLAY('ENTER NEW VALUE FOR ' || ID)
                                              REPLY(INPUT);
                          Z = Z + 1;
                          IF S = 'C' THEN SUBSTR(CVAL,1,N) = INPUT;
                          ELSE DO;
                               CALL PTR_ADJUST;
                               IF RCODE ¬= 1 THEN SUBSTR(CVAL,1,LEN) =
                                                        INPUT;
                               ELSE DO;
                                    DISPLAY('LENGTH OF ' || ID ||'CONTAINS');
                                    DISPLAY(ADJ || 'TOO MANY CHARACTERS.');
                                    IF Z <= 2 THEN GOTO LOOP;
                                         ELSE GOTO M1;
                                    END;
                               END;

/*                  MODIFY (RECORD-NAME)                  */

                          SUBSCHEMA_CTRL.DML_SEQUENCE = '0017';
                          CALL IDMSP1F(ADDR(IDBMSCOM(35)),RN);
                          DISPLAY('ARE THERE ANYMORE DATA_ELEMENTS');
                          DISPLAY('TO BE MODIFIED? (YES/NO)')
                                              REPLY(ANS);
                          IF ANS = 'NO' THEN TEST1 = 0;
                          END;
                     END;
                DISPLAY('ARE THERE ANYMORE RECORDS TO BE MODIFIED?');
                DISPLAY(' (YES/NO)') REPLY(ANS);
                IF ANS = 'NO' THEN TEST2 = 0;
                END;
             ELSE TEST2 = 0;
     END START;
M1:  P = T;
     FREE CVAL;
     END MODIFY;
```

```
/*****************************************************************/
/*                                                               */
/*                INFORMATION RETRIEVAL ROUTINE                  */
/*                                                               */
/*****************************************************************/

INFO: PROCEDURE;
      ALLOCATE CVAL;
      T = P;
      RCODE = 1;
      TEST1 = 1;
I1:   DO WHILE (TEST1 =1);
            TEST2 = 1;
            DISPLAY('ENTER THE NAME OF THE RECORD WHICH ');
            DISPLAY('CONTAINS THE INFORMATION TO BE DISPLAYED')
                                                  REPLY(RN);

/*  REC_FIND PLACES THE RECORD TO BE DISPLAYED INTO WORKING     */
/*  STORAGE.                                                    */

            CALL REC_FIND;
            IF RCODE ¬= 1 THEN TEST1 = 0;
            ELSE DO;
                  DISPLAY('DO YOU WANT THE ENTIRE RECORD DISPLAYED?')
                                                       REPLY(ANS);
                  IF ANS = 'YES' THEN CALL REC_DISPLAY;
                  ELSE DO;
                        DO WHILE(TEST2 = 1);
                        DISPLAY('ENTER THE NAME OF THE DATA_ELEMENT');
                        DISPLAY(' TO BE DISPLAYED.');
                        DISPLAY('ELEMENT =') REPLY(ID);

/*  SYM_SEARCH SET A POINTER TO THE BEGINNING ADDRESS OF THE */
/*  DATA_ITEM TO BE DISPLAYED.                               */

                        CALL SYM_SEARCH;
                        IF RCODE ¬= 1 THEN GOTO I2;
                        ELSE DO;
                              OUTPUT = SUBSTR(CVAL,1,N);
                              DISPLAY(ID || ' = ' || OUTPUT);
                              DISPLAY('IS THERE ANYMORE INFORMATION');
                              DISPLAY('IN THIS RECORD TO BE DISPLAYED?');
                              DISPLAY('(YES/NO)') REPLY(ANS);
                              IF ANS = 'NO' THEN TEST2 = 0;
                              END;
                        END;
                  DISPLAY('IS THERE INFORMATION IN ANOTHER');
                  DISPLAY('RECORD YOU WISH DISPLAYED? (YES/NO)')
                                                 REPLY (ANS);
```

```
          IF ANS = 'NO' THEN TEST1 = 0;
     END I1;
I2:  P = T;
     FREE CVAL;
     END INFO;

/*****************************************************************/
/*                                                               */
/*                    REC_FIND ROUTINE                           */
/*                                                               */
/*****************************************************************/

REC_FIND: PROCEDURE;
          RCODE = 1;

/* TEMP = RN IS USED SO THAT IF RN IS THE MEMBER OF A SET AND    */
/* DOES NOT HAVE A CALC IDENTIFIER, THE OWNER OF THE SET IS      */
/* RETRIEVED AND THEN RN = TEMP IS USED TO FIND THE MEMBER THAT */
/* IS DESIRED. ID_FIND IS THE ROUTINE THAT DETERMINES WHETHER OR*/
/* NOT THE RECORD DESIRED HAS A CALC IDENTIFIER AND IF NOT IT    */
/* LOCATES THE OWNER RECORD AND ITS CALC IDENTIFIER.             */

          TEMP = RN;
          CALL ID_FIND;
          IF RCODE = 1 THEN DO;
               CALL AD_COMP(RN,RPTR,RCODE);
               IF RCODE = 1 THEN DO;

/* SYM_SEARCH SETS A POINTER TO THE BEGINNING ADDRESSS OF THE    */
/* CALC IDENTIFIER FOUND BY ID_FIND.                             */

                    CALL SYM_SEARCH;
                    IF RCODE = 1 THEN DO;
LOOP:                    DISPLAY('ENTER VALUE FOR' || ID) REPLY
(INPUT);
                         IF S = 'C' THEN SUBSTR(CVAL,1,N) = INPUT;
                         ELSE DO;
                              CALL PTR_ADJUST;
                              IF RCODE = 1 THEN SUBSTR(CVAL,1,LEN) =
                                                            INPUT;
                              ELSE DO;
                                   DISPLAY('LENGTH OF ' || ID ||
                                           'CONTAINS   '||ADJ ||
                                           '   TOO MANY CHARACTERS.');
                                   GOTO LOOP;
                                   END;
                              END;

                     /* OBTAIN CALC(RN)    */

                         SUBSCHEMA_CTRL.DML_SEQUENCE = '0014';
                         CALL IDMSP1F(ADDR(IDBMSCOM(32)),RN
                                   ,ADDR(IDBMSCOM(43)));
                         CALL IDMS_STATUS;
                         IF RCODE ¬= 1 THEN GOTO RF1;
                    IF FLAG = 1 THEN DO;
                         RN = TEMP;

/* THIS IS WHERE THE DESIRED RECORD (WHICH IS A SET MEMBER) IS    */
/* IDENTIFIED. FLAG = 1 INDICATES THAT THE ORIGINAL RN WAS THE    */
/* MEMBER OF A SET.                                                */

                         DISPLAY('ENTER A UNIQUE NAME TO');
                         DISPLAY('IDENTIFY THE DESIRED RECORD.')
                                   REPLY(ID);
                         CALL AD_COMP;
                         IF RCODE = 1 DO;
                              CALL SYM_SEARCH;
                              IF RCODE = 1 THEN DO;
                                   DISPLAY('ENTER' \\ ID)
                                        REPLY(INPUT);

/*  OBTAIN FIRST RECORD-NAME WITHIN SET-NAME   */

                                   SUBSCHEMA_CTRL.DML_SEQUENCE =
                                                       '0015';
                                   CALL IDMSP1F(ADDR(IDBMSCOM(18))
                                             ,RN,SN
                                             ,ADDR(IDBMSCOM(43)));
                                   IF ERROR_STATUS = '0307' THEN
                                   DISPLAY(ID \\ SUBSTR(INPUT,1,N) \\
                                        'NOT FOUND.');
                                   CALL IDMS_STATUS;
                                   DO WHILE
                    (SUBSTR(CVAL,1,N) ¬=INPUT & RCODE = 1);

/*  OBTIAN EACH RECORD IN THE SET AS LONG AS THE UNIQUE IDENTIFIER */
/*  THAT THE USER GAVE IS NOT EQUAL TO THAT IDENTIFIER IN THE      */
/*  RECORD OBTAINED.                                                */

                    SUBSCHEMA_CTRL.DML_SEQUENCE = '0016';
                    CALL IDMSP1F(ADDR(IDBMSCOM(10)),RN,SN
                              ,ADDR(IDBMSCOM(43)));
                    IF ERROR_STATUS = '0307' THEN
                    DISPLAY(ID \\ SUBSTR(INPUT,1,N) \\ 'NOT FOUND.');
                    CALL IDMS_STATUS;
                    END;
RF1:      END REC_FIND;

/**************************************************************/
/*                                                            */
/*              REPORT  ROUTINE                               */
/*                                                            */
/**************************************************************/

REPORT: PROCEDURE;

/* THIS ROUTINE WOULD NORMALLY MAKE CALLS TO OTHER ROUTINES      */
/* WHICH WOULD BE SEPERATE APPLICATION PROGRAMS THAT CAUSE       */
/* CERTAIN REPORTS TO BE WRITTEN.                                 */
```

```
          DISPLAY('REPORT PROC CALLED...');
          RCODE = 1;
          END REPORT;

/****************************************************************/
/*                                                              */
/*              SECURITY CODE CHECK ROUTINE                     */
/*                                                              */
/****************************************************************/

CODECK: PROCEDURE;
          DECLARE CODE CHAR(6);
          OPEN FILE (SECMAT);
          GET FILE (SECMAT) EDIT (CODE) (A(6));
          DO WHILE (CODE ¬= PCODE);
               IF CODE = '*     ' THEN DO; /* THE * INDICATES EOF */
                    J = 1;
                    GO TO ERR;
                    END;
               GET FILE (SECMAT) EDIT (CODE)(X(6),A(6));
               END;
          GET FILE (SECMAT) EDIT (CODE) (A(6));
          IF CODE ¬= SCODE THEN DO;
               J = 2;
               GO TO ERR;
               END;
          ELSE DO;
               RCODE = 1;
               GO TO LAST;
               END;

ERR:      IF J = 1 THEN DISPLAY('INVALID PERSONNEL CODE.');
               ELSE DISPLAY('INVALID SECURITY CODE.');
          IF I = 3 THEN DO;
               DISPLAY('PLEASE CHECK YOUR PERSONNEL NUMBER AND');
               DISPLAY('SECURITY CODE PRIOR TO ATTEMPTING RE-ENTRY');
               DISPLAY('INTO THE SYSTEM.');
               DISPLAY('PROGRAM TERMINATING.');
               RCODE = 5;
               END;
LAST:     CLOSE FILE(SECMAT);
          END CODECK;
```

```
/*****************************************************************/
/*                                                               */
/*                   ERROR MESSAGE ROUTINE                       */
/*                                                               */
/*****************************************************************/

ERROR:    PROCEDURE;
          IF RCODE = 2   THEN
               DISPLAY('DATA_ELEMENT' || ID || 'NOT FOUND');
               ELSE IF RCODE = 3 THEN
                    DISPLAY('RECORD NAME' || RN || 'NOT FOUND');
          DISPLAY('RETURNING TO MAIN ROUTINE.......');
          END ERROR;

/*****************************************************************/
/*                                                               */
/*                      ID_FIND ROUTINE                          */
/*                                                               */
/*****************************************************************/

ID_FIND: PROCEDURE;
          DCL IDLIST FILE EXTERNAL,
               RECORD CHAR(80);

/*  ON ENDFILE CONDITION WILL EXIST IF THE RECORD NAME IS INVALID*/

          ON ENDFILE (IDLIST) BEGIN;
                              RCODE = 3;
                              GO TO LAST;
                              END;
          RCODE = 1;
          FLAG = 0;
          DO WHILE('1'B);

/*     GET FIRST RECORD IN THE FILE  */

               OPEN FILE (IDLIST);
               GET FILE (IDLIST) EDIT (RECORD) (A(80));

/*   FIND RECORD THAT MATCHES RECORD NAME    */

               DO WHILE(RN ¬= SUBSTR(RECORD,1,16));
                    GET FILE (IDLIST) EDIT (RECORD) (A(80));
                    END;

/*  WHEN RN IS FOUND, THE RECORD IS CHECKED FOR THE CALC IDENTIFIER*/
/*  IF IT HAS NONE THEN FLAG = 1, RN = THE OWNER RECORD, AND       */
/*  SN = THE SET NAME. THE CODE THEN LOOPS BACK TO  FIND THE  NEW  */
/*  RECORD.                                                        */
```

```
          IF SUBSTR(RECORD,17,16) ¬= ' ' THEN DO;
               ID = SUBSTR(RECORD,17,16);
               GO TO LAST;
               END;
          FLAG = 1;
          RN = SUBSTR(RECORD,33,16);
          SN = SUBSTR(RECORD,49,16);
          CLOSE FILE (IDLIST);
          END;
LAST:     CLOSE FILE (IDLIST);
          END ID_FIND;

/******************************************************************/
/*                                                                */
/*                ADDRESS COMPUTATION ROUTINE                     */
/*                                                                */
/******************************************************************/

AD_COMP:  PROCEDURE;
          %DECLARE NAMES CHARACTER;
          %INCLUDE RECNAMS;
          RCODE = 1;
          AD_CALC(NAMES);
          ELSE RCODE = 5;
          END AD_COMP;

/******************************************************************/
/*                                                                */
/*                 RECORD DISPLAY ROUTINE                         */
/*                                                                */
/******************************************************************/

REC_DISPLAY:   PROCEDURE:
          RCODE = 1;
          IF RN = 'INSURED-REC' THEN DO;
               PUT FILE (SYSPRINT) DATA (INSURED_REC);
               PUT SKIP;
               END;
               ELSE IF RN = 'FINANCIAL-REC' THEN DO;
                    PUT FILE (SYSPRINT) DATA (FINANCIAL_REC);
                    PUT SKIP;
                    END;
                    ELSE IF RN = 'OCCUPATION-REC' THEN DO;
                         PUT SKIP;
                         END;
                         ELSE IF RN = 'LIFE-REC' THEN DO;
                              PUT FILE (SYSPRINT) DATA (LIFE_REC);
                              PUT SKIP;
                              END;
                         ELSE IF RN = 'HEALTH-REC' THEN DO;
                              PUT FILE (SYSPRINT) DATA (HEALTH_REC);
                              PUT SKIP;
                              END;
```

```
                                ELSE IF RN = 'MEDICAL-REC' THEN DO;
                                     PUT FILE (SYSPRINT) DATA (MEDICAL_REC);
                                     PUT SKIP;
                                     END;
                                ELSE RCODE = 3;
          END REC_DISPLAY;

/*******************************************************************/
/*                                                                 */
/*                     SYMBOL SEARCH ROUTINE                       */
/*                                                                 */
/*******************************************************************/

SYM_SEARCH: PROCEDURE;

/*  THE ON KEY CONDITION WILL HOLD IF THE SYMBOL DOES NOT EXIST  */

          ON KEY (SYMFILE) BEGIN;
                                RCODE = 2;
                                GO TO LAST;
                                END;
          RCODE = 1;
          CNT = 1;
          READ FILE (SYMFILE) INTO (BUFF) KEY (RN || CNT);

/*   SEARCH THROUGH SYMFILE FOR ID      */

          DO WHILE (BUFF.NAME ¬= ID);
               CNT = CNT + 1;
               READ FILE (SYMFILE) INTO (BUFF) KEY (RN || CNT);
               END;

/*  WHEN ID IS FOUND, SET THE BASE POINTER TO THE ADDRESS OF ID. */
/*  S = EITHER C OR F DEPENDING ON WHETHER THE ATTRIBUTE OF ID   */
/*  IS CHARACTER OR FIXED. N = THE NUMBER OF BYTES OF STORAGE    */
/*  REQUIRED FOR ID.                                             */


          UNSPEC(RPTR) = UNSPEC(RPTR) + MASK;
          K = BIN(UNSPEC(RPTR),31,0);
          L = BIN(UNSPEC(BUFF.PTR),31,0);
          UNSPEC(P) = UNSPEC(BIN(K+L,31,0));
          S = BUFF.ID;
          N = BUFF.VALUE;
LAST:     END SYM_SEARCH;
```

```
/*******************************************************************/
/*                                                                 */
/*                   POINTER ADJUST ROUTINE                        */
/*                                                                 */
/*******************************************************************/

PTR_ADJUST:    PROCEDURE;
               RCODE = 1;
               LEN = INDEX(INPUT,' ')-1;
               ADJ = N - LEN;
               IF ADJ < 0 THEN RCODE = 4;
                    ELSE DO:
                         K = BIN(UNSPEC(P),31,0);
                         L = BIN(UNSPEC(ADJ),31,0);
                         UNSPEC(P) = UNSPEC(BIN(K+L,31,0));
                         END;
               END PTR_ADJUST;

TERM:     FINISH;
          DISPLAY('END OF JOB');
          END DATATST;
```

## APPENDIX B

```
qL1
DMSL107401 EXECUTION BEGINS...
PLEASE ENTER YOUR SECURITY CODE.

PLEASE ENTER YOUR PERSONNEL NUMBER.

ENTER NUMBER FOR FUNCTION TO BE PERFORMED
1.  ENTER NEW DATA.
2.  DELETE DATA.
3.  MODIFY DATA.
4.  INFORMATION RETRIEVAL.
5.  REPORT GENERATION.

1
ENTER RECORD NAME
occupation-rec
PLEASE ENTER THE FOLLOWING INFORMATION
ENTER C-AREA-CODE
913
ENTER C-CITY
manhattan
ENTER C-PHONE-NUMBER
776-4100
ENTER C-STATE
ks
ENTER C-STREET
913 w 5th ave
ENTER C-ZIP-CODE
66502
ENTER COMPANY-NAME
u.s. pipe co.
ENTER OCCUPATION
welder
OCCUPATION-REC  RECORD STORED...
DO YOU WISH TO ENTER ANYMORE RECORDS?
yes
ENTER RECORD NAME
insured-rec
PLEASE ENTER THE FOLLOWING INFORMATION
ENTER AGE
34
ENTER ANN-NET-INCOME
$18,000.00
ENTER CITY
clifton
```

```
ENTER DATE-OF-BIRTH
100342
ENTER 1-AREA-CODE
913
ENTER 1-NAME
robert smith
ENTER MARITAL-STATUS
2
ENTER PHONE-NUMBER
883-9911
ENTER POLICY-TYPE
lh
ENTER SSAN-ID
173926515
ENTER STATE
ks
ENTER STREET-ADDRESS
111 steel ave.
ENTER YEARS-EMPLOYED
5
ENTER ZIP-CODE
66507
INSURED-REC     RECORD STORED...
DO YOU WISH TO ENTER ANYMORE RECORDS?
yes
ENTER RECORD NAME
medical-rec
PLEASE ENTER THE FOLLOWING INFORMATION
ENTER CONDITION
1
ENTER D-AREA-CODE
913
ENTER D-CITY
clifton
ENTER D-PHONE-NO
883-0000
ENTER D-STATE
ks
ENTER D-STREET
123 main st.
ENTER D-ZIP-CODE
66507
```

```
ENTER DOCTORS-NAME
o. roberts
ENTER LAST-PHYS-DATE
061074
ENTER MAJ-AILMENT
none
ENTER SSAN-ID
173926515
MEDICAL-REC     RECORD STORED...
DO YOU WISH TO ENTER ANYMORE RECORDS?
yes
ENTER RECORD NAME
life-rec
PLEASE ENTER THE FOLLOWING INFORMATION
ENTER B-NAME
betty smith
ENTER CITY
clifton
ENTER FACE-VALUE
$10,000.00
ENTER ISSUE-DATE
050173
ENTER POLICY-ID
100000005
ENTER STATE
ks
ENTER STREET
111 steel ave.
ENTER ZIP-CODE
66507
LIFE-REC            RECORD STORED...
DO YOU WISH TO ENTER ANYMORE RECORDS?
yes
ENTER RECORD NAME
health-rec
PLEASE ENTER THE FOLLOWING INFORMATION
ENTER EXPIRATION-DATE
050177
ENTER ISSUE-DATE
050173
ENTER POLICY-CLASS
general health
ENTER POLICY-ID
h00000005
HEALTH-REC            RECORD STORED...
DO YOU WISH TO ENTER ANYMORE INFORMATION?
yes
```

```
ENTER RECORD NAME
financial-rec
PLEASE ENTER THE FOLLOWING INFORMATION
ENTER AMOUNT-OF-LOAN

ENTER AMOUNT-REPAID

ENTER CURRENT-OWED

ENTER D-A-AMOUNT

ENTER DATE-APPROVED

ENTER FREQUENCY
mo
ENTER L-P-REC-AMOUNT
4362
ENTER L-P-DATE
110376
ENTER N-P-DUE-DATE
1200576
ENTER NO-OF-PAYMENTS

ENTER P-AMOUNT
$43.62
ENTER POLICY-ID
L00000005
FINANCIAL-REC     RECORD STORED...
DO YOU WISH TO ENTER ANYMORE RECORDS?
yes
ENTER RECORD NAME
financial-rec
PLEASE ENTER THE FOLLOWING INFORMATION
ENTER AMOUNT-OF-LOAN

ENTER AMOUNT-REPAID

ENTER CURRENT-OWED

ENTER D-A-AMOUNT

ENTER DATE-APPROVED

ENTER FREQUENCY
sa
ENTER L-P-REC-AMOUNT
10000
```

```
ENTER L-P-DATE
010376
ENTER N-P-DUE-DATE
070576
ENTER NO-OF-PAYMENTS

ENTER P-AMOUNT
$100.00
ENTER POLICY-ID
h00000005
FINANCIAL-REC     RECORD STORED...
DO YOU WISH TO ENTER ANYMORE RECORDS?
no
DO YOU WISH TO CONTINUE?   (YES/NO)

no
END OF JOB
R;

qL1
DMSL107401 EXECUTION BEGINS...
PLEASE ENTER YOUR SECURITY CODE.

PLEASE ENTER YOUR PERSONNEL NUMBER.

ENTER NUMBER FOR FUNCTION TO BE PERFORMED
1.  ENTER NEW DATA.
2.  DELETE DATA.
3.  MODIFY DATA.
4.  INFORMATION RETRIEVAL.
5.  REPORT GENERATION.
3

ENTER NAME OF RECORD TO BE MODIFIED
financial-rec
ENTER VALUE FOR POLICY-ID
L00000001
ENTER NAME OF ELEMENT TO
BE MODIFIED.
no-of-payments
ENTER NEW VALUE FOR NO-OF-PAYMENTS
1
ARE THERE ANYMORE DATA-ELEMENTS
TO BE MODIFIED?  (YES/NO)
no
ARE THERE ANYMORE RECORDS TO
BE MODIFIED?  (YES/NO)
no
DO YOU WISH TO CONTINUE?  (YES/NO)

yes
```

```
ENTER NUMBER FOR FUNCTION TO BE PERFORMED
1.  ENTER NEW DATA.
2.  DELETE DATA.
3.  MODIFY DATA.
4.  INFORMATION RETRIEVAL.
5.  REPORT GENERATION.

4
ENTER THE NAME OF  THE RECORD WHICH
CONTAINS THE INFORMATION TO BE DISPLAYED
financial-rec
ENTER VALUE FOR POLICY-ID
L00000005
DO YOU WANT THE ENTIRE RECORD DISPLAYED?
yes

FINANCIAL_REC.POLICY_ID='L00000005'
FINANCIAL_REC.PREMIUM_DATA.P_AMOUNT='$43.62 '
FINANCIAL_REC.PAYMENT_DATA.LAST_PAYMENT.L_P_REC AMOUNT=
'4362'
FINANCIAL_REC.DELINQUENT_ACCT.NO_OF_PAYMENTS='   '
FINANCIAL_REC.LOAN_DATA.DATE_APPROVED='      '
FINANCIAL_REC.LOAN_DATA.AMOUNT_REPAID='        '
FINANCIAL_REC.LOAN_DATA.CURRENT_OWED='         '
FINANCIAL_REC.FILLER0007='                       ';
IS THERE INFORMATION IN ANOTHER
RECORD YOU WISH DISPLAYED?  (YES/NO)
no
DO YOU WISH TO CONTINUE?  (YES/NO)

yes
ENTER NUMBER FOR FUNCTION TO BE PERFORMED
1.  ENTER NEW DATA.
2.  DELETE DATA.
3.  MODIFY DATA.
4.  INFORMATION RETRIEVAL.
5.  REPORT GENERATION.

3
ENTER NAME OF RECORD TO BE MODIFIED
financial-rec
ENTER VALUE FOR POLICY-ID
L00000005
ENTER NAME OF ELEMENT TO
BE MODIFIED.
no-of-payments
ENTER NEW VALUE FOR NO-OF-PAYMENTS
1
ARE THERE ANYMORE DATA_ELEMENTS
TO BE MODIFIED?  (YES/NO)
```

```
yes
ENTER NAME OF ELEMENT TO
BE MODIFIED.
d-a-amount
ENTER NEW VALUE FOR D-A-AMOUNT
4362
ARE THERE ANYMORE DATA_ELEMENTS
TO BE MODIFIED?  (YES/NO)
no
ARE THERE ANYMORE RECORDS TO
BE MODIFIED?  (YES/NO)
no
DO YOU WISH TO CONTINUE?  (YES/NO)

yes
ENTER NUMBER FOR FUNCTION TO BE PERFORMED
1.  ENTER NEW DATA.
2.  DELETE DATA.
3.  MODIFY DATA.
4.  INFORMATION RETRIEVAL.
5.  REPORT GENERATION.

4
ENTER THE NAME OF THE RECORD WHICH
CONTAINS THE INFORMATION TO BE DISPLAYED
financial-rec
ENTER VALUE FOR POLICY-ID
L00000005
DO YOU WANT THE ENTIRE RECORD DISPLAYED?
no
ENTER THE NAME OF THE DATA_
ELEMENT TO BE DISPLAYED.
ELEMENT =
no-of-payments
NO-OF-PAYMENTS    = 1
IS THERE ANYMORE INFORMATION IN THIS RECORD
TO BE DISPLAYED?  (YES/NO)
yes
ENTER THE NAME OF THE DATA_
ELEMENT TO BE DISPLAYED.
ELEMENT =
d-a-amount
D-A-AMOUNT        =   4362
IS THERE ANYMORE INFORMATION IN THIS RECORD
TO BE DISPLAYED? (YES/NO)
no
IS THERE INFORMATION IN ANOTHER
RECORD YOU WISH DISPLAYED (YES/NO)
no
DO YOU WISH TO CONTINUE?  (YES/NO)
```

```
yes
ENTER NUMBER FOR FUNCTION TO BE PERFORMED
1.  ENTER NEW DATA.
2.  DELETE DATA.
3.  MODIFY DATA.
4.  INFORMATION RETRIEVAL.
5.  REPORT GENERATION.

4
ENTER THE NAME OF THE RECORD WHICH
CONTAINS THE INFORMATION TO BE DISPLAYED
life-rec
ENTER VALUE FOR SSAN-ID
173926515
ENTER A UNIQUE NAME TO IDENTIFY THE DESIRED RECORD
policy-id
ENTER POLICY-ID
L00000005
DO YOU WANT THE ENTIRE RECORD DISPLAYED?
yes
LIFE_REC.POLICY_ID='L00000005'
LIFE_REC.ISSUE-DATE='050173'
LIFE_REC.BENEFICIARY.B_NAME='BETTY SMITH                      '
LIFE_REC.ADDRESS.STREET='STEEL AVE.                '
LIFE_REC.BENEFICIARY.B_ADDRESS.STATE='KS'
LIFE_REC.FILLER0003='      ';
IS THERE INFORMATION IN ANOTHER
RECORD YOU WISH DISPLAYED? (YES/NO)
yes
ENTER THE NAME OF THE RECORD WHICH
CONTAINS THE INFORMATION TO BE DISPLAYED
insured-rec
ENTER VALUE FOR SSAN-ID
173926515
DO YOU WANT THE ENTIRE RECORD DISPLAYED?
yes
INSURED_REC.SSAN_ID='173926515'
INSURED_REC.I-NAME='NAME=ROBERT SMITH
INSURED_REC.CURRENT_ADDRESS.STREET_ADDRESS='111 STEEL AVE.
INSURED_REC.CURRENT_ADDRESS.CITY='CLINTON
INSURED_REC.CURRENT_ADDRESS.STATE='KS  '
INSURED_REC.CURRENT_ADDRESS.ZIP_CODE='66507'
INSURED_REC.TELEPHONE.AREA_CODE='913'
INSURED_REC.TELEPHONE.PHONE_NUMBER='883-9911'
INSURED_REC.AGE='34'
INSURED_REC.MARITAL_STATUS='2'
INSURED_REC.ANN_NET_INCOME='$18,000.00 '
INSURED_REC.POLICY_TYPE='LH'
INSURED_REC.FILLER0002='   '
IS THERE INFORMATION IN ANOTHER
RECORD YOU WISH DISPLAYED?  (YES/NO)
no
```

```
DO YOU WISH TO CONTINUE?  (YES/NO)

yes
ENTER NUMBER FOR FUNCTION TO BE PERFORMED
1.  ENTER NEW DATA.
2.  DELETE DATA.
3.  MODIFY DATA.
4.  INFORMATION RETRIEVAL.
5.  REPORT GENERATION.

2
ENTER THE NAME OF THE RECORD  TO BE DELETED
life-rec
ENTER VALUE FOR SSAN-ID
173926515
ENTER A UNIQUE NAME TO IDENTIFY THE DISPLAYED RECORD
policy-id
ENTER POLICY-ID
L00000005
ARE THERE ANYMORE RECORDS TO DELETE?
no
DO YOU WISH TO CONTINUE?  (YES/NO)

yes
ENTER NUMBER FOR FUNCTION TO BE PERFORMED
1.  ENTER NEW DATA.
2.  DELETE DATA.
3.  MODIFY DATA.
4.  INFORMATION RETRIEVAL.
5.  REPORT GENERATION.

4
ENTER THE NAME OF THE RECORD WHICH
CONTAINS THE INFORMATION TO BE DISPLAYED
life-rec
ENTER VALUE FOR SSAN-ID
173926515
ENTER A UNIQUE NAME TO IDENTIFY THE DESIRED RECORD
policy-id
ENTER POLICY-ID
L00000005
POLICY-ID     L00000005 NOT FOUND.
****IDMS ERROR DETECTED****
PROGRAM NAME = DATATST
ERROR STATUS = 0307
ERROR RECORD = LIFE-REC
ERROR SET = INSURED-LIFE-SET
ERROR AREA = INSURANCE-AREA
LAST GOOD RECORD WAS INSURED-REC
LAST GOOD AREA WAS INSURANCE-AREA
DML SEQUENCE NUMBER IS              15
****IDMS ERROR DETECTED****
RETURNING TO MAIN ROUTINE........
DO YOU WISH TO CONTINUE?  (YES/NO)

yes
ENTER NUMBER FOR FUNCTION TO BE PERFORMED
1.  ENTER NEW DATA.
2.  DELETE DATA.
3.  MODIFY DATA.
4.  INFORMATION RETRIEVAL.
5.  REPORT GENERATION.

4
ENTER THE NAME OF THE RECORD WHICH
CONTAINS THE INFORMATION TO BE DISPLAYED
insured-rec
ENTER VALUE FOR SSAN-ID
173926515
DO YOU WANT THE ENTIRE RECORD DISPLAYED?
no
ENTER THE NAME OF THE DATA
ELEMENT TO BE DISPLAYED.
ELEMENT =
i-name
I-NAME           =ROBERT SMITH
IS THERE ANYMORE INFORMATION IN THIS RECORD
TO BE DISPLAYED?  (YES/NO)
no
IS THERE INFORMATION IN ANOTHER
RECORD YOU  WISH DISPLAYED?  (YES/NO)
no
DO YOU WISH TO CONTINUE?  (YES/NO)

yes
ENTER NUMBER FOR FUNCTION TO BE PERFORMED
1.  ENTER NEW DATA.
2.  DELETE DATA.
3.  MODIFY DATA.
4.  INFORMATION RETRIEVAL.
5.  REPORT GENERATION.

2
ENTER THE NAME OF THE RECORD TO BE DELETED
```

```
occupation-rec
ENTER VALUE FOR OCCUPATION
welder
ARE THERE ANYMORE RECORDS TO DELETE?
no
DO YOU WISH TO CONTINUED?  (YES/NO)

yes
ENTER NUMBER FOR FUNCTION TO BE PERFORMED
1.  ENTER NEW DATA.
2.  DELETE DATA.
3.  MODIFY DATA.
4.  INFORMATION RETRIEVAL.
5.  REPORT GENERATION.

4
ENTER THE NAME OF THE RECORD WHICH
CONTAINS THE INFORMATION TO BE DISPLAYED
insured-rec
ENTER VALUE FOR SSAN-ID
173926515
****IDMS ERROR DETECTED****
PROGRAM NAME = DATATST
ERROR STATUS = 0326
ERROR RECORD = INSURED-REC
ERROR SET = CALC
ERROR AREA = INSURANCE-AREA
LAST GOOD RECORD WAS OCCUPATION-REC
LAST GOOD AREA WAS INSURANCE-AREA
DML SEQUENCE NUMBER 1S             14
****IDMS ERROR DETECTED****
RETURNING TO MAIN ROUTINE........
DO YOU WISH TO CONTINUE? (YES/NO)

no
END OF JOB

R;

CHECKPOINTING...
```

## REFERENCES CONSULTED

Cullinane Corporation, Data Manipulation Language Programmer's Reference Guide, release 3-1, April, 1974, Boston, Mass.

Cullinane Corporation, IDMS DML Manual Supplement, June, 1976, Boston, Mass.

International Business Machines, IBM Virtual Machine Facility/370: CMS User's Guide, release 3, 1st ed., February 1976.

International Business Machines, PL/1(F) Language Reference Manual, 5th ed., December, 1972.

International Business Machines, PL/1(F) Programmer's Reference Guide, 9th ed., September, 1972.

MARTIN, J., Computer data-base organization, Prentice-Hall, Englewood Cliffs, N. J., 1975.

MARTIN, J., Design of man-computer dialogues, Prentice-Hall, Englewood Cliffs, N. J., 1973.

TAYLOR, Robert W.; FRANK, Randall L., "CODASYL Data-Base Management Systems", Computing Surveys 8, 1 (March, 1976), 67-103.

TSKHRITZIS, D. C; and LOCHOVSKY, F. H., "Hierarchical Data-Base Management: A Survey", Computing Surveys 8, 1 (March, 1976), 105-123.

# IDMS QUERY LANGUAGE

by

William E Shea

B.S., University of Tampa, Tampa Florida, 1972

---

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1977

The purpose of this project is to design an interactive QUERY LANGUAGE to interface with Cullinane Corporation's Integrated Database Management System (IDMS).

The query language wil be generalized and designed to perform the following:

1. Provide additional protection for the system by using security codes and personnel numbers for each active user.
2. Provide the capability of retrieving, storing, deleting, or modifing information in the database with only a limited amount of knowledge of the database structure.
3. A report generator module will be included which will place calls to the actual report module provided by the user.

A prototype version of the language will be implemented.

The language will be written in PL/1 and will be a machine prompt lanquage, i.e. the user will be asked to provide information and assisted (to a limited extent) in determining what to do next.