COMPLIANCE WITH CODASYL STANDARDS: A CASE STUDY

by

C. LARRY KEELER

B. A., Kansas University, 1963

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

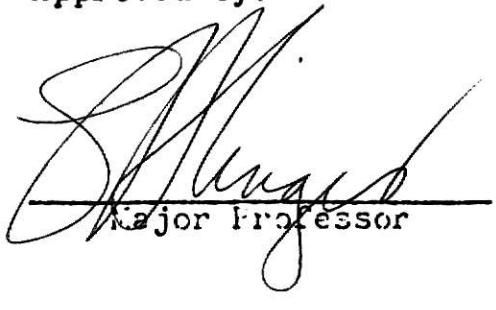
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

KANSAS STATE UNIVERSITY
MANHATTAN, KANSAS

1981

Approved by:



Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES

continued

CHAPTER 1: INTRODUCTION

1.0 Purpose

The objective of this paper is to produce a comprehensive comparison report that will detail the differences and similarities of CODASYL's DBTG 1971 (CODA71) proposal on data base management systems and that of a commercially available data base management system, specifically UNIVAC's DMS 1100. The comparison between the two will exhibit:

A. The degree to which a major vendor has adhered to the 1971 DBTG proposal. This provides one indication of the effect of the DBTG proposal on the data processing community.

B. The extent to which a committee of business, vendor, and government people can create a proposal that is useable and marketable in the data processing industry.

C. The specific areas of data base management systems standardization one vendor indicates of importance by compliance with the standard. This aspect of the comparison portrays the areas given the most research and effort this vendor believes most marketable.

This comparison criteria may provide insight into the future level of influence and the expected future life of the CODASYL's subcommittee, DATA BASE TASK GROUP.

1.1 CODASYL STANDARDIZATION

The DBTG proposal is the one vehicle available for use in a standardization effort. Standardization will have many benefits for data base users including a degree of portability of data base manipulating programs.

If compliance to the DBTG proposal was fairly uniform among major vendors and a standard set of codes for use in manipulating data bases could be developed and shared among users, this would result in many advantages such as:

1. ease of reading code
2. insure a high degree of compatibility among different vendor's languages
3. increase the portability of code
4. ease the transfer of programs between users
5. ease of transfer between different generations of computer hardware
6. encourage the development of standard auxiliary features such as: data dictionary/data directory systems, documentation standards, etc.
7. encourage programmer training aids

Compliance to a standard such as the CODASYL (CODA71) DBTG proposal would result in advantages to potential DBMS users in the data processing community.

One disadvantage of standardization with a fluid field such as

data bases is the rigidity it would give the emerging ideas.

1.2 THE COMMERCIAL DATA BASE MANAGEMENT SYSTEM - UNIVAC DMS 1100

A data base management system has been described as "a software tool that provides for the logical organization, physical organization, access to and maintenance of data base."

Sperry Univac in 1972 announced the release of a data base management system called DMS 1100. This network-oriented data base management system was designed for use on their 1100 series processors. DMS 1100 allows for both one-to-one relationships and for many-to-many relationships.

According to Sperry Univac, the DMS 1100 is "CODASYL-based." Later comparisons with the CODASYL 1971 proposal will clarify this claim.

DMS 1100 provides for:

A. Separation of the data definition function of the DBMS from the data manipulation function thus providing some degree of data independence.

B. Data base protection and integrity is provided for via the features within DMS 1100.

DMS 1100 has three principle components:

A. a data description language, (DDL)

B. a data manipulation language, (DML)

C. a data management routine, (DMR)

1.3 ORGANIZATION OF THIS REPORT

This report is organized into three additional chapters. Chapter 2 reviews the history and objectives of the DBTG and defines the task group's proposal. Chapter 3 describes the data base management system under study, namely Univac's DMS 1100 and then compares this system with the DBTG proposal for a standard DBMS. Chapter 4 summarizes the results of this study.

CHAPTER 2: CODASYL DATA BASE TASK GROUP (DBTG) STANDARDS

2.0 HISTORICAL MOTIVATION

The Conference on Data Systems Languages (CODASYL) began work in 1959. It has existed since its inception by virtue of committees of volunteers who are supported by the organization which employ them. CODASYL's initial emphasis was on the development of the COBOL language.

In June, 1965, it was suggested to the CODASYL COBOL Language Subcommittee that "List Processing" be added to the agenda as a topic for future development work. As a result a List Processing Task Force was formed and met regularly to solicit the opinions of interested parties, to examine many data base and file systems and to produce working papers at all levels of detail from functional requirements to draft languages specification. In May of 1967, the List Processing Task Force voted to change its name to the Data Base Task Group (DBTG).

Because the membership of the group changes constantly, a major amount of the DBTG's effort is directed toward listening to and examining the views of as many people as possible.

In an effort to solicit endoresment of the DBTG's objectives, and to solicit recommendations and/or guidance for future work, the DBTG presented to the COBOL Language subcommittee in January of 1968, an interim report entitled "COBOL EXTENSIONS TO HANDLE

DATA BASES." At that time the committee had determined that the structure of a data base management system should be included in the COBOL language specifications. Later the DBTG produced functional and language specifications for data base management systems and incorporated its specifications within the COBOL language specifications.

At the tenth anniversary conference of CODASYL held in May 1969, consideration was given to the idea of separating the data description and the data manipulation portions of languages. It was thought by the committee that such a separation would allow data bases described by a data description language to be independent of a user language used for processing the data. This became the basis of the direction of the efforts by the DBTG for the next few years.

Late in 1969 the DBTG presented the recommendations concerning the separation of data description and manipulation to its parent committee, the main CODASYL Committee. These recommendations detailed sample semantics and syntax of a data description language and a data manipulation language. The data description language specified in the report is a language which when associated with facilities of a host language such as COBOL, PL/1, ALGOL, JOVIAL, FORTRAN ..., allows the manipulation of data bases described by the data description language.

It was the hope of the DBTG that the data description language

would ultimately form the basis for an industry standard and that that individual host language would subsequently interface with it.

The semantics and syntax of the data manipulation language detailed in the 1969 DBTG report were proposed not only as an extention to COBOL, but also as a prototype of the manipulative capabilities required in a user language.

The April 1971 DBTG report upon which the comparison herein is based is a revised edition of the 1969 DBTG report.

2.1 DATA BASE TASK GROUP OBJECTIVES

Data base management systems originated in the early 1960's. Some of the earliest developed were the integrated data store (IDS) system introduced by General Electric Company. Among the basic concepts in the early development years were the following two:

A. separation of the structure of the data from the excuting program and thereby institute the beginnings of data independence.

B. creation of data structures which were not strictly a linear physical sequential file organization (one or two dimensional tables). These structures made extensive use of pointers and/or tables to describe the data structure. The data structure was then interpreted by software interposed between the physical data structure and the application program.

As a result of the original concepts, there has been a tendency for control over data to be relinquished by individual programmers in favor of more centralized and coordinated control. This development has paralleled a change in the philosophy of processing away from the traditional approach where data is always accessed by one run-unit (processing module or single program) at a time to one with concurrent access. In such an environment, files are frequently designed to optimize the processing of a run-unit; for other processing to be performed on the same data, the files are re-sorted or new files created which re-

dundantly contain the same data. The traditional approach to data was thus to create process-oriented (one unique file for each specific purpose or function) files. This was adequate in some circumstances, but too costly or impractical in others. It is too costly because today's storage devices permit a more integrated approach to be taken to data. This approach permits multiple applications to use and share the same data without requiring redundancy. Since the quantities of contemporary data are generally large and ever expanding and since data acquisition and maintenance costs are high, data storage with minimal redundance is an important consideration. File replication or exclusive access approaches are becoming increasingly impractical in that they are unsuitable for an environment where message processing is involved, especially when the data must be used by more than one application or process. It is essential to develop data bases that are available to and suitable for processing by multiple applications and that can be interfaced by multiple languages.

In addition, as random access storage devices came into common use data bases started gaining wide acceptance. Random access devices represent complex relationships and still maintain performance and storage efficiencies. The above led to the 1971 data base management system proposal as advanced by the Data Base Task Group of the Conference on Data System Languages (CODASYL). This report, although controversial, is a landmark in the development of data base technology.

The data base tasks group's stated objectives were to:

A. allow data to be structured in a mannger most suitable to each application, regardless of the fact that some or all of that data may be used by other applications - such flexibility to be achieved without requiring data redundancy.

B. allow more than one run-unit (the execution of one or more programs) to concurrently retrieve or update the data base.

C. provide protection of the data base against unauthorized access of data and from untoward interaction of programs.

D. provide for centralized capability to control the physical placement of data.

E. provide device independence for programs.

F. allow the declaration of a variety of data structures ranging from those in which no connection exists between data items to network structures.

G. allow the user to interact with the data while being relieved of the mechanics of maintaining the structural associations that have been declared.

H. allow programs to be as independent of the data as possible.

I. provide for separate descriptions of the data in the data base and of the data known to the program.

J. provide for a description of the data base which is not restricted to a particular processing language.

K. provide an architecture which permits the description of

the data base and/or the data base itself to be interfaced by multiple processing language.

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

2.2 DBTG PROPOSAL - MAJOR CONCEPTS

The Data Description Languages (DDL) are languages used for describing a data base, or that part of a data base known to an application program. These descriptions are given in terms of the names and characteristics of the data-items, data-aggregates, records, areas, and sets included in the data base, and the relationships that exist and must be maintained between occurrence of those elements. A data-item is the smallest unit of named data. A data-aggregate is a named collection of data-items within a record. There are two types of data-aggregates, vectors and repeating groups. A vector is a one-dimensional, ordered collection of data-items, all of which have identical characteristics. A repeating group is a collection of data that occurs an arbitrary number of times within a record occurrence. The collection may consist of data-items, vectors, and repeating groups. A record is a named collection of zero, one, or more data-items or data-aggregates. There may be an arbitrary number of occurrences in the database of each record type specified in the schema.

A set is a named collection of record types. As such, it establishes the characteristics of an arbitrary number of occurrences of the named set.

An area is a named sub-division of the addressable storage space in the data base and may contain occurrences of records and sets or parts of sets of various types. The concept of area allows

the data base to be subdivided instead of considering the data base as a single unit. The system allows a person or the DBMS to control placement of an entire area to provide efficient storage and retrieval. Areas are a convenient unit for recovery, as duplication or backup can be carried out selectively. Areas also provide a convenient natural subdivision for allowing certain unused portions of the database to be saved in archival storage while the remainder of the database is actively accessed.

A data base consists of all the record occurrences, set occurrences and areas which are controlled by a specific schema. If an installation has multiple data bases, there must be a separate schema for each data base.

A schema is described by DDL entries and is a complete description of a data base. It includes the names and descriptions of all of the areas, set occurrences, record occurrences and associated data-items and data-aggregates as they exist in the data base.

A sub-schema is described by DDL entries. It, however, need not describe the entire data base but only those areas, sets, records, data-items and data-aggregates which are known to one or more specific programs. A subschema describes a user view of the database by describing the form in which that part of the data base is known to those specific programs.

The Data Manipulation Language (DML) is the language used to cause data to be transferred between the application program and the data base. The DML is not a complete language; it relies on a host language (primary programming language used by the application programmer) to provide a framework for it and to provide the procedural capabilities required to manipulate data.

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

To aid in the understanding of the relationship between DDL/DML and the data base management system (DBMS) it may be helpful to conceptualize a complete system. The conceptualized system is presented in figure 2.2-1 where the numbered arrows trace a call for data by user program 1. Calls for data by other user programs are handled concurrently by the data base management system (DBMS). The actions in this call sequence are:

1. a call for data by the user program to the DBMS. All calls for the services of the DBMS are made in the DML.
2. the DBMS analyzes the call and supplements the arguments provided in the call itself with information contained in the object version of the schema for the data base, and in the object version of the sub-scheme invoked by

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

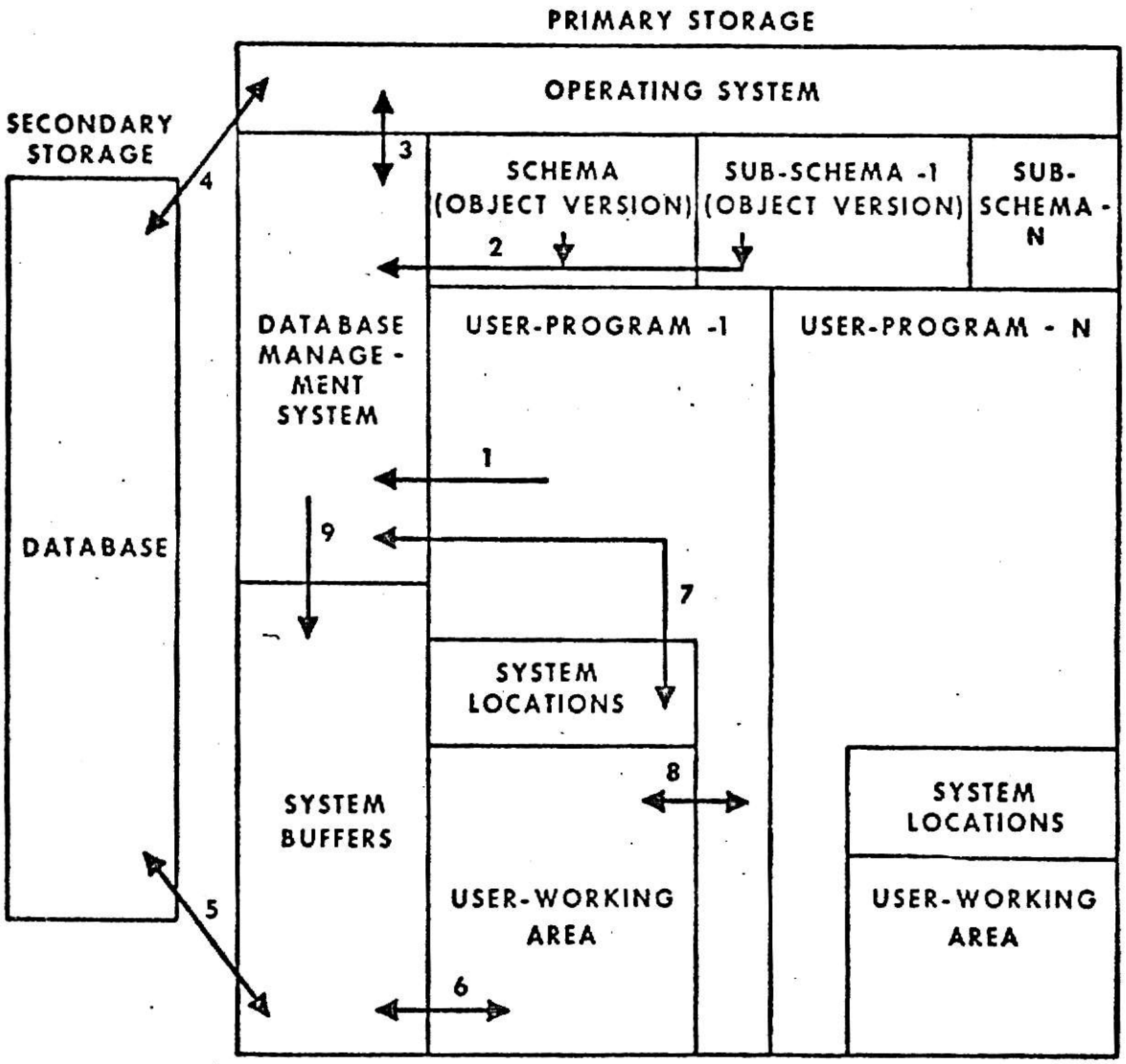


Figure 2.2-1 CONCEPTUAL DATA BASE MANAGEMENT SYSTEM

the user program originating the call. The schema describes the database in terms of the characteristics of the data and the implicit and explicit relationship between data items. The sub-schema is a subset of the schema. It describes the data known to a user program including the forms in which the DBMS makes the data available in which it appears in the program's USER WORKING AREA (UWA). In this conceptual system it is assumed that the object version of the sub-schema contains only the differences from the schema and is not complete in itself. The source form of the schema is written in the schema DDL, and the source form of the sub-schema is written in the sub-schema DDL.

3. on the basis of the call for its services and information obtained from the object versions of the schema and sub-schema, the DBMS requests physical I/O operations, as required to execute the call, from the operating system.
4. the operating system interacts with secondary storage.
5. the operating system transfers data between secondary storage and the system buffers.
6. the DBMS transfers data, as required to fulfill the call, between the system buffers and the UWA of the program originating the call. Any required data transformations between the representation of the data as it appears in secondary storage and the representation of the data as it appears in a program's UWA are handled by the DBMS.
7. the DBMS provides the status information to the calling

program on the outcome of its call. The information provided is: currency status information, error status condition codes, area-name and record name.

8. data in a program's UWA may be manipulated as required, using the facilities of the host language.
9. the DBMS administers the system buffers. The system buffers are shared by all programs serviced by the DBMS. User programs interact with the system buffers entirely through the DBMS.

RELATIONSHIP BETWEEN THE HOST LANGUAGE AND THE DML

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

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

CONCEPT OF SCHEMA AND SUB-SCHEMA

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

A. an individual programmer need not be concerned with or even have knowledge of the universe of the entire data base but only with those portions of the data base which are relevant to his programming needs.

B. a program is limited to the subset of the schema that is known to it via its sub-schema.

C. a measure of data independence is provided for programs in that certain changes may be made to the schema for the data base and the data base adjusted accordingly without affecting existing programs using that data. This is possible because the sub-schema may vary in certain important aspects from the schema of which it is a subset.

D. a common language is allowed to be specified for defining a data base while allowing that part of the data base known to a program to be described in a manner which is oriented towards the conventions of the language in which that program is written.

VARIATIONS BETWEEN THE SCHEMA AND THE SUB-SCHEMA

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

A. at the data item level:

1. the characteristics of data items may be different.
2. privacy locks may be changed.
3. descriptions of specific data items may be omitted.
4. the ordering of data-items may be changed.

B. at the data-aggregate level

1. descriptions of specific data-aggregate may be omitted.
2. privacy locks may be changed.
3. the ordering of data-aggregate may be changed.
4. vectors may be redefined as multi-dimensional arrays.
5. data-items or data-aggregates may be selected and given a group name.
6. additional structure mapping may be provided by the facilities of a particular sub-schema DDL.

C. at the record level

1. descriptions of specific record types may be omitted.
2. privacy locks may be changed.
3. record occurrences included in specific areas may be omitted while other occurrences of that record type are included.

D. at the set level

1. descriptions of specific set types may be omitted.
2. privacy locks may be changed.
3. different set selection criteria may be specified.

E. at the area level

1. descriptions of specified areas may be omitted.
2. privacy locks may be changed.

2.3 CODASYL LANGUAGE COMMANDS

This section describes the CODASYL DBTG defined language commands. These commands are defined in figures 2.3-1 to 2.3-14.

Function

- To make the object record occurrence unavailable for further processing by the imperative-statements of the DML.
- To remove the object record from all set occurrences in which it is a member.
- To delete all record occurrences which are mandatory members of set occurrences owned by the object record.
- To remove or optionally delete all record occurrences which are optional members of set occurrences owned by the object record.
- To optionally prevent deletion of the object record if the database contains any non-empty set occurrences of which the object record is the owner.

General Format

DELETE [record-name] [ONLY / SELECTIVE / ALL]

Syntax Rules

1. Record-name must refer to a record which has been defined in the sub-schema named in the Data Division of the program.

Figure 2.3-1: THE DELETE COMMAND

Function

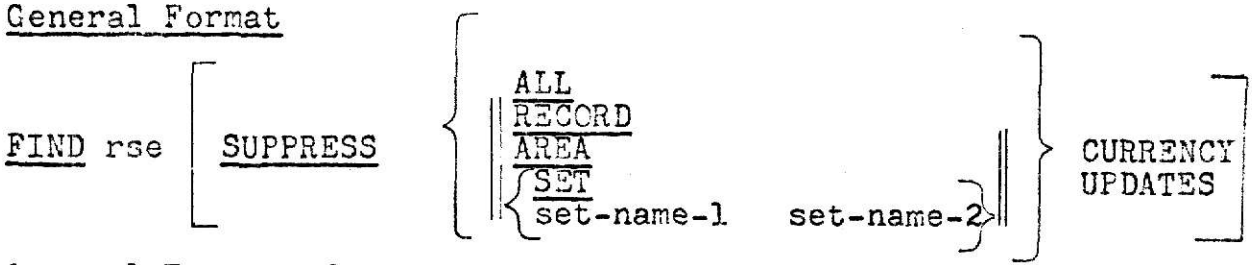
To establish a record occurrence specified by a record-selection-expression (rse) as:

- the current record of the run-unit
- the current record of the area in which it is stored
- the current record of record-name
- the current record of set for all set-names in which it currently participates as an owner or member.

To optionally suppress the establishment of the object record occurrence as:

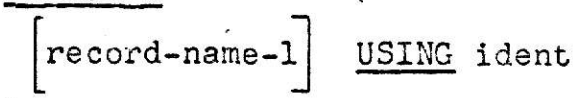
- the current record of the area in which it is stored
- the current record of *record-name*
- the current record of set for all, or specified sets in which it currently participates as an owner or member.

General Format

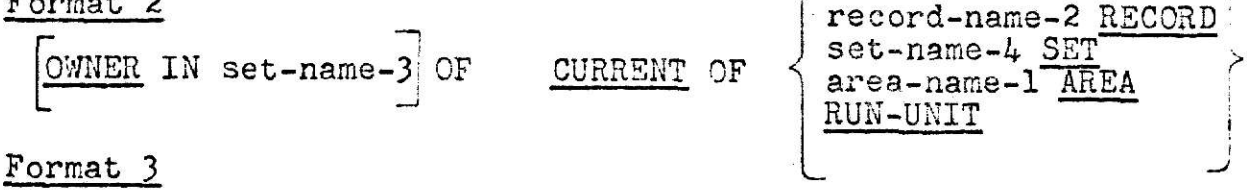


General Format of record-selection-expression (rse)

Format 1



Format 2



Format 3

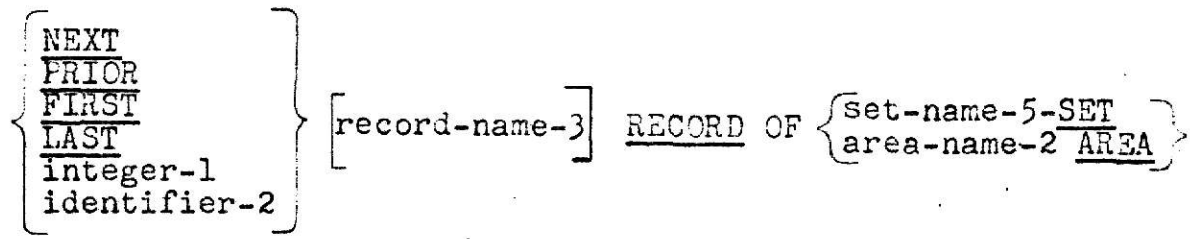


Figure 2.3-2: THE FIND COMMAND

Format 4

OWNER RECORD OF set-name-6 SET

Format 5

```
[NEXT DUPLICATE WITHIN]   record-name-4 RECORD
```

Format 6

```
record-name-5 VIA [CURRENT OF] set-name-7
    [USING data-base-identifier-3   [,data-base-identifier-4] ...]
```

Format 7

```
NEXT DUPLICATE WITHIN set-name-8
    USING data-base-identifier-5   [,data-base-identifier-6] ...
```

Syntax Rules

1. In Format 1 identifier-1 must be defined with a USAGE IS DATABASE-KEY clause.

2. In format 2, record-name-2 must be defined as a member of set-name-3. Set-name-3 and set-name-4 may be the same set-name or different set-names. If set-name-3 is a singular set, a FIND OWNER IN set-name-3 statement must not be used.

3. In format 3, if record-name-3 and area-name-2 are stated, the WITHIN clause for record-name-3 must include area-name-2. If record-name-3 and set-name-5 are stated, record-name-3 must be defined as a member of set-name-5. Integer-1 must be a signed integer and cannot be zero. Identifier-2 must be defined as an integer.

4. Format 4 must not be used if set-name-6 is a singular set.

5. In format 5, the LOCATION MODE IS CALC clause must have been used in the description of record-name-4. All search arguments specified in the CALC clause must be included in the sub-schema named in the Data Division of the program.

6. In Format 6, data-base-identifier-3 and data-base-identifier-4 are implicitly qualified by record-name-5 and must be the names of data items included in record-name-5. Record-name-5 must be defined as a member of set-name-7 and unless the CURRENT phrase is used, all search arguments specified in its SET OCCURRENCE SELECTION clause in set-name-7, must be included in the sub-schema named in the Data Division of the

Figure 2.3-2: THE FIND COMMAND (cont)

program.

7. In Format 7, data-base-identifier-5 and data-base-identifier-6 must be names of data items included in a record type defined as a member of set-name-8.

8. All data items, records, sets, and areas specified must be defined in the sub-schema named in the Data Division of the program.

Figure 2.3-2: THE FIND COMMAND (cont)

Function

To advise the DBMS to cancel for the object record occurrence any explicit keep which is in effect and to reset any 03 Error Status Condition which applies to the object record.

General Format

FREE [ALL]

Syntax Rules

None

Figure 2.3-3: THE FREE COMMAND

Function

To transfer the contents of the specified data items of the object record occurrence into the User Working Area.

General Format

Format 1

GET [record-name]

Format 2

GET record-name; data-base-identifier-1 [,data-base-identifier-2 ...]

Syntax Rules

1. Record-name must refer to a record which has been defined in the sub-schema named in the INVOKE clause in the Data Division of the program.

2. Data-base-identifier-1, data-base-identifier-2 must be defined as part of the named record. The record-name serves as an implicit major qualifier for the data-base-identifiers.

Figure 2.3-4: THE GET COMMAND

Function

The IF statement causes a condition to be evaluated. The subsequent action of the run-unit depends on whether the value of the condition is true or false.

General Format

Format 1

```
IF set-name-1 SET [NOT] EMPTY; {statement-1 / NEXT SENTENCE} [ELSE {statement-2 / NEXT SENTENCE}]
```

Format 2

```
IF RECORD NOT [MEMBER / OWNER] OF {set-name-2 / ANY} SET; {statement-3 / NEXT SENTENCE}

    [; ELSE {statement-4 / NEXT SENTENCE}]
```

Syntax Rules

1. Set-name-1, set-name-2 and record-name must be defined in the sub-schema named in the INVOKE clause in the Data Division of the program.

Figure 2.3-5: THE IF COMMAND

Function

To make the object record a member of occurrences of the specified set-names, provided that it is defined as an optional automatic, optional manual, or mandatory manual member of those sets.

General Format

Format 1

```
INSERT [record-name] INTO set-name-1 [,set-name-2] ...
```

Format 2

```
INSERT [record-name] INTO ALL SETS
```

Syntax Rules

1. Record-name and all set-names must be defined in the sub-schema named in the Data Division of the program.

2. If record-name is specified in format 1, the named record must be defined as an optional or mandatory manual member of the specified sets. If record-name is specified in format 2, the named record must be defined as an optional or mandatory manual member of at least one set included in the invoked sub-schema.

Figure 2.3-6: THE INSERT COMMAND

Function

To advise the DBMS of the run-unit's intent to re-access the object record occurrence.

General Format

KEEP

Syntax Rules

None.

Figure 2.3-7: THE KEEP COMMAND

Function

To save the contents of the specified currency status indicators and to provide a means for deriving the area-name which corresponds to a database-key.

General Format

Format 1

```
MOVE CURRENCY STATUS FOR { RUN-UNIT
                         { record-name RECORD
                         { area-name AREA
                         { set-name SET          } TO identifier-1
```

Format 2

```
MOVE AREA-NAME FOR { RUN-UNIT
                   { record-name RECORD
                   { area-name AREA
                   { set-name SET          } TO identifier-3
                   { identifier-2
```

Syntax Rules

1. The specified record-name, area-name or set-name must be included in the sub-schema named in the Data Division of the program.

2. Identifier-1 and identifier-2 must refer to data items defined with a USAGE IS DATABASE-KEY clause. Identifier-3 must refer to an alphanumeric elementary data item.

Figure 2.3-8: THE MOVE COMMAND

Function

To specify an area's usage-mode and to delay further run-unit execution until such usage can be permitted.

To call for the checking of the run-unit's variable PRIVACY clauses.

General Format

Format 1

```
                ┌ FOR SET set-name-1 [,set-name-2] ... ┐
OPEN ALL        │                                      │
                └ USAGE-MODE IS [EXCLUSIVE] {RETRIEVAL} ┘
                                [PROTECTED] {UPDATE   }
```

Format 2

```
OPEN AREA area-name-1 [, area-name-2] ...
    [USAGE-MODE IS [EXCLUSIVE] {RETRIEVAL}]
                   [PROTECTED] {UPDATE   }
```

Syntax Rules

1. Set-name-1, set-name-2, must be the names of sets included in the sub-schema named in the Data Division of the program.

Figure 2.3-9: THE OPEN COMMAND

Function

- To replace the values of all, or of specific data items of the object record occurrence in the database, with values from the User Working Area.
- To alter set occurrence membership and intraset position, so as to maintain the database in accordance with the set occurrence selection and ordering criteria specified in the schema.

General Format

Format 1

```
MODIFY [record-name]
    [ USING data-base-identifier-1 [,data-base-identifier-2] ... ]
```

Format 2

```
MODIFY record-name; data-base-identifier-3 [,data-base-identifier-4] ...
    [ USING data-base-identifier-5 [,data-base-identifier-6] ... ]
```

Syntax Rules

1. Data-base-identifier-1, data-base-identifier-2, and, data-base-identifier-5, data-base-identifier-6 must refer to control data items specified in SET OCCURRENCE SELECTION clauses. In addition, if a record-name is specified they must refer to all the control data items specified for the immediate owner records in the SET OCCURRENCE SELECTION clauses of one or more of the sets in which the named record is defined as a member.
2. Data-base-identifier-3, data-base-identifier-4 must not be defined as actual or virtual result data items nor may they refer to data items defined as actual or virtual source data items unless the related data item in the owner record is a control data item explicitly or implicitly referenced in the appropriate SET OCCURRENCE SELECTION clause.
3. In format 2 record-name serves as an implicit major qualifier for data-base-identifier-3, data-base-identifier-4.

Figure 2.3-10: THE MODIFY COMMAND

Function

To cancel the membership of the object record in the occurrences of the specified set-names in which it currently participates as a member, provided that the object record is defined as an optional member of the sets named.

General Format

Format 1

```
REMOVE  [record-name]  FROM set-name-1  [, set-name-2]...
```

Format 2

```
REMOVE  [record-name]  FROM ALL SETS
```

Syntax Rules

1. Record-name and all set-names must be defined in the sub-schema named in the Data Division of the program.

2. If record-name is specified in format 1, the named record must be defined as an optional member of the specified sets. If record-name is specified in format 2, the named record must be defined as an optional member of at least one set included in the sub-schema named in the Data Division of the program.

Figure 2.3-11: THE REMOVE COMMAND

Function

- To acquire space and a database-key for a new record occurrence in the database.
- To cause the values of the appropriate data items in the User Working Area to be included in the occurrence of the object record in the database.
- To insert the object record into all sets for which it is defined as an automatic member in the schema.
- To establish a new set occurrence for each set for which the object record is defined as owner in the schema.
- To establish the object record as
  - the current record of the run-unit
  - the current record of the area in which it is stored
  - the current record of record-name
  - the current record of set for all set-names in which it is specified as an owner or automatic member.
- To optionally suppress the establishment of the object record as
  - the current record of the area in which it is stored
  - the current record of record-name
  - the current record of set for all, or specified sets, in which it is described as an owner or automatic member.

General Format

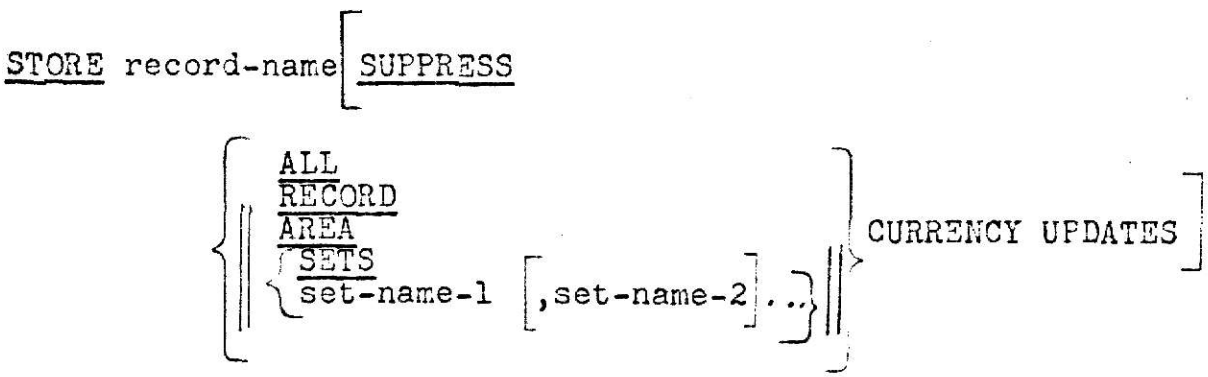


Figure 2.3-12: THE STORE COMMAND

Syntax Rules

1. The sub-schema named in the Data Division of the program must include the named record; the data items or set specified in the LOCATION MODE clause of the named record; at least one of the areas specified in the WITHIN clause of the named record; all sets in which the named record is defined as an automatic member; all data items, records, and sets specified or referenced in the SET OCCURRENCE SELECTION clauses and ASCENDING/DESCENDING KEY clauses of those sets in which the named record is defined as an automatic member.

Figure 2.3-12: THE STORE COMMAND (cont)

Function

- To relinquish control over the specified areas and make them available to requesting run-units which are delayed.
- To relinquish the effect of any KEEP statement on all record occurrences in the specified areas.

General Format

Format 1

```
CLOSE ALL [FOR SET set-name-1 [,set-name-2]... ]
```

Format 2

```
CLOSE AREA area-name-1 [,area-name-2]...
```

Syntax Rules

1. All the set-names and area-names specified must be included in the sub-schema named in the Data Division of the program.

Figure 2.3-13: THE CLOSE COMMAND

Function

To specify procedures to be executed when specified Error Status Conditions occur.

General Format

```
USE IF ERROR-STATUS [ IS integer-1 [,integer-2]... ]
```

Syntax Rules

1. Integer-1, integer-2,...must be a valid, four-digit Error Status Condition code.

2. If integers are specified, multiple USE statements can exist within a program. However, the conditions in each USE statement must be unique; that is, different USE statements cannot specify the same integers.

3. See the previously referenced Journal of Development for additional rules.

General Rules

1. If the execution of a DML imperative-statement results in a specified Error Status Condition, the procedure following the USE statement is executed.

2. If no Error Status Conditions are specified, the procedure following the USE statement is executed whenever any Error Status Condition other than zero occurs.

3. If an EXIT or an END DECLARATIVE statement is encountered in the execution of a USE procedure, control returns to the statement immediately following the DML statement which resulted in the Error Status Condition.

4. Whenever the execution of a DML statement results in an Error Status Condition, the following information is available to the run-unit.

   - The Error Status Condition code is available in the special register ERROR-STATUS.
   - The name of the area in which the error occurred is available in the special register ERROR-AREA.
   - The name of the set, if appropriate, in which the error occurred is available in the special register ERROR-SET.

Figure 2.3-14: THE USE COMMAND

2.4 PROTECTION OF THE DATA IN THE DBTG PROPOSAL

The schema and sub-schema DDLs and the DML include provision for the protection of data in the "social environment" of a shared data base. A shared data base is one which contains data relevant to many aspects of an organization's operations or one in which the data is shared by multiple programs or applications. In this type of environment, two kinds of protection are required:

A. protection against unauthorized access of data-privacy.

B. safeguarding the data from untoward interaction of programs - integrity.

PRIVACY OF DATA

Protection of the data in the data base is furnished through a mechanism of PRIVACY LOCKS which are specified in the schema and sub-schema, and PRIVACY KEYS which must be provided by a run-unit seeking to access or alter data which is protected by means of privacy locks. The schema and sub-schema DDLs provide for declaring privacy locks at:

A. the schema level

B. the sub-schema level

C. the area level

D. the record level

E. the data-item and data-aggregate level

F. the set level

See figures 2.4-1 & 2.4-2 for the general format for the syntax of privacy lock statement which apply to schema and sub-schema.

Where run-units are permitted to interact with the same data concurrently, they require protection against each other. The DML includes such protection as illustrated in the following.

IF run-unit-1 retrieves record occurrence A (executes a Find command and then a Get command) and decides to modify it.

AND in the interval of time between run-unit-1 executing the Find command and his attempted execution of a Modify command, a concurrent run-unit, namely run-unit-2, has executed a successful Modify command on record occurrance A.

THEN the system will not execute the Modify command for run-unit-1 of the situation by means of an Error Status Condition Code.

At this point run unit-1:

A. can decide to modify record occurrence A in any case (by executing a free command and then a Modify command), or

B. can decide to make a new decision based on the potentially changed values for data-items AA and AB (by executing a free command, followed by a Get command, followed by a Get command).

The number of record occurrences and the duration of time for which notification of interference by concurrent run-units will

be given to a run-unit is under the control of that run-unit in that it may execute a KEEP command against all the record occurrences for which it requires such notification. A KEEP command may be cancelled only by a countermanding FREE command, by closing the appropriate area, or by the termination of the run-unit.

Function

To specify the privacy lock(s) for certain operations (LOCKS, DISPLAY, COMPILE, ALTER) which apply to the use of a sub-schema.

General Format

```
PRIVACY LOCK [ FOR [ LOCKS | DISPLAY | COMPILE | ALTER ] ] IS { literal-1 | lock-name-1 | PROCEDURE data-base-procedure-1 }

[ OR { literal-3 | lock-name-2 | PROCEDURE data-base-procedure-2 } ] . . .
```

Syntax Rules

1. A separate PRIVACY LOCK clause may be stated for each restricted operation (LOCKS, DISPLAY, COMPILE, ALTER). However, the same operation must not be specified in more than one PRIVACY LOCK clause.

2. By their appearance in a PRIVACY LOCK clause, lock-name-1 and lock-name-2 are implicitly declared as data items with implementor-defined data characteristics.

3. All literals must conform to the implementor-defined data characteristics of privacy locks.

Figure 2.4-1: PRIVACY LOCK COMMAND FOR THE SUB-SCHEMA

Function

To specify the privacy lock(s) for certain operations (LOCKS, DISPLAY, COPY, ALTER) which apply to the schema.

General Format

```
PRIVACY LOCK [ FOR [ LOCKS | DISPLAY | COPY | ALTER ] ] IS { literal-1 | lock-name-1 | PROCEDURE data-base-procedure-1 }

[ OR { literal-2 | lock-name-2 | PROCEDURE data-base procedure-2 } ] . . .
```

Syntax Rules

1. A separate PRIVACY clause may be stated for each restricted operation (LOCKS, DISPLAY, COPY, ALTER). However, the same operation must not be specified in more than one PRIVACY clause.

2. By their appearance in a PRIVACY clause, lock-name-1 and lock-name-2 are implicitly declared as data items with implementor-defined data characteristics.

3. All literals must conform to the implementor-defined data characteristics of privacy locks.

Figure 2.4-2: PRIVACY LOCK COMMAND FOR THE SCHEMA

2.5 SUMMARY

Chapter 2 attempts to explain the:

- Historical motivation
- objectives
- commands
- utility features

of proposal of the CODASYL (CODA71) DBTG. It does not attempt to debate the merits and/or demerits of the specific approaches taken to implement them. Such debates have taken place and probably will continue to be held. It should be remembered that the role of the DBTG was to recommend systems specifications for a Data Base Management System.

CHAPTER 3: COMPLIANCE OF DMS 1100 WITH CODASYL STANDARDS

3.0 INTRODUCTION

This chapter discusses the features of DMS 1100 Data Base Management System. Given this basis a comparison is made with the CODASYL DBTG Proposal for a DBMS.

A. Structure

   This will be a general comparison of the general pattern of organization of the DMS 1100 with the organization of the CODASYL (CODA71) DBTG Proposal.

B. Commands

   This area will compare the DML commands proposed by the CODASYL (CODA71) proposal and those commands required by the DMS 1100.

C. Support Features

   A comparison is made of the support and utility features of the two areas.

The major objectives in designing and releasing the DMS 1100 were to provide Sperry Univac 1100 series users with a data base management system that would:

A. would remove from the individual users of data base functions the burden of constructing specific read and write commands to the data base.

B. provide for data base protection and integrity

C. provide for data to be independent of a specific user language

D. separate the data definition and data manipulation functions.

3.1 STRUCTURE OF DMS 1100

DMS 1100 has three principle components:

A. data description language (DDL)

1. schema definition

a. The primary function of DDL is to describe or define the structure of the entire data base.

b. The structure of the data base is described using three types of descriptors: the area section, the record section and the set section.

The area section describes a named sub-division of the addressable storage space in the data base. The description includes occurrences of records and sets or parts of sets of various types.

The record section which is a named collection of one or more data items. The record section describes the characteristics of each record in the data base including the way it is stored and a definition.

The set section which defines the named collection of record types. It describes the relationship that exists between records as owners and members.

A schema is produced using the schema data definition language

(DDL). Schemas describe the data base as it exists on mass storage. The DDL processor must be executed before any other processor within the data management system. Its input is the user source schema and its output is in the schema tables used by the data management routine (DMR). An example of the data description language that defines a schema is shown in figure 3.1-1.

```
IDENTIFICATION DIVISION.

     SCHEMA NAME IS schema-name IN FILE schema-file

          LOCK FOR COPY IS copy-lock

          ACCESS CONTROL LOCK FOR schema-list IS schema-lock

          ACCESS CONTROL KEY IS schema-key

DATA DIVISION.

AREA SECTION.

     AREA NAME IS area-name

     AREA LOOKS INCLUDE  QUICK-BEFORE-LOOKS
                         AFTER-LOOKS
                         BEFORE-LOOKS
                         NO-LOOKS

     MODE IS DATA     AREA
             INDEX
             POINTER

     ACCESS CONTROL LOCK FOR area-list IS area-lock

     ACCESS CONTROL KEY IS area-key

     ALLOCATE ### PAGES ### OVERFLOW PAGES AT END
                        ### OVERFLOW PAGES EVERY ### DATA PAGES
                        DYNAMICALLY EXPANDABLE TO ### PAGES

     PAGES ARE ### WORDS

RECORD SECTION.

     RECORD NAME IS record-name

     LOCATION MODE IS DIRECT
                      CALC
                      VIA set-name SET
                      INDEX SEQUENTIAL

     WITHIN area name

     FOR ENCODING CALL encode-procedure
         DECODING CALL decode-procedure

     ACCESS CONTROL LOCK FOR record-list IS record-lock
```

Figure 3.1-1: SCHEMA SYNTAX

```
ACCESS CONTROL KEY IS record-key

    05  item-name PICTURE IS pic-list USAGE IS usage-list

        CHECK IS data-check
```

S C H E M A S Y N T A X

```
SET SECTION.

    SET NAME IS set-name

    MODE IS CHAIN
            POINTER ARRAY

    ORDER IS FIRST  DUPLICATES ARE FIRST
             LAST                  LAST
             NEXT                  NOT ALLOWED
             PRIOR
             SORTED

    ACCESS CONTROL LOCK FOR set-list IS set-lock

    ACCESS CONTROL KEY IS set-key

    OWNER IS record-name

    MEMBER IS record-name AUTOMATIC
                          MANUAL
```

Figure 3.1-1: SCHEMA SYNTAX (cont)

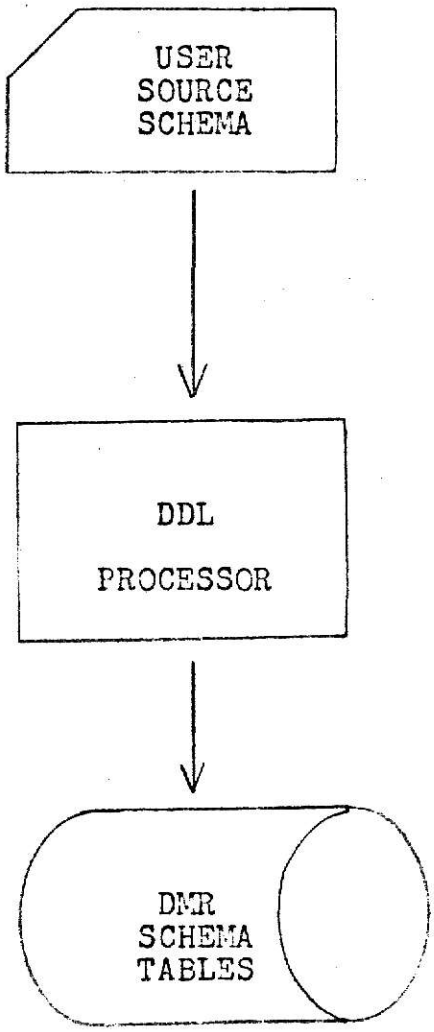


Figure 3.1-2: SCHEMA FLOW

2. Subschema definition

A subschema is produced using the subschema data definition language (SDDL). The SDDL processor must be executed after the DDL processor and before the data manipulation language processor - see figure 3.1-3 -. Its inputs are the user source subschema and the schema tables from the DDL. Its outputs are the subschema tables used by the DMR, the record-name and data-item-name definitions used by the DML, and the subschema working relocatable used by the collector. Subschemas describe the data base, or portion of the data base which is visable to the user program. It allows any area, record, set or item names to be renamed. The area section describes the areas available to the user program. The record section describes the records and items available to the user program.

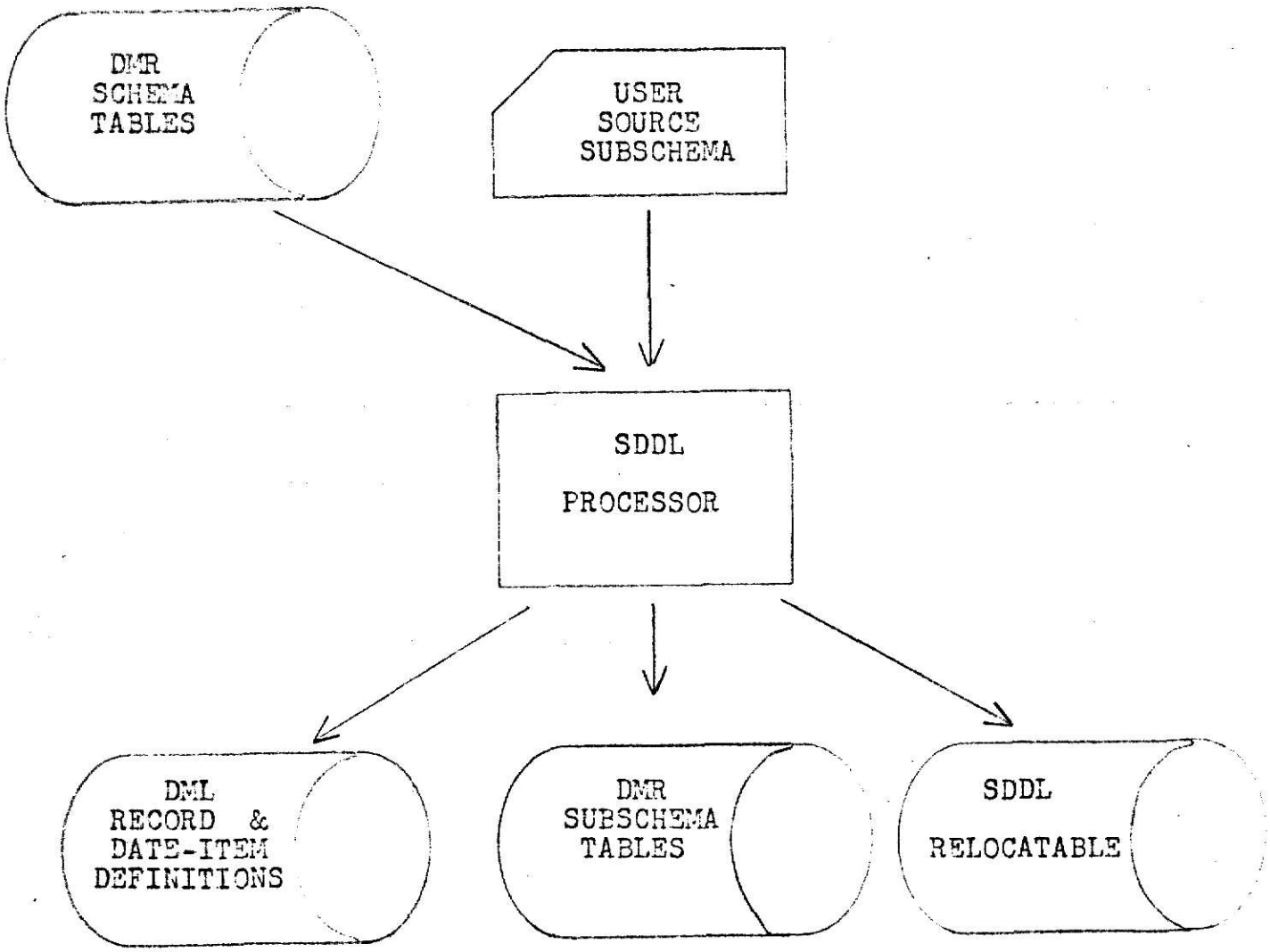


Figure 3.1-3: SUBSCHEMA FLOW

The set section describes the sets available to the user program. The QLP section describes the path and root records available to the query language processor (QLP). An example of the data description language that defines a subschema is shown in figure 3.1-3.

```
IDENTIFICATION DIVISION.

     SUBSCHEMA NAME IS subschema-name IN FILE subschema-file
     OF SCHEMA schema-name

     HOST LANGUAGE IS COBOL
                      FORTRAN
                      PL1
                      QLP
                      DMU

     KEY FOR COPY IS copy-key

     LOCK FOR INVOKE IS invoke-lock

DATA DIVISION.

AREA SECTION.

     AREAS ARE ALL
               ALL EXCEPT area-name(s)
               area-name(s)

     AREA NAME IS new-area-name FROM old-area-name

RECORD SECTION.

     RECORDS ARE ALL
                 ALL EXCEPT record-name(s)
                 record-name(s)

     RECORD NAME IS new-record-name FROM old-record-name


          ITEMS ARE ALL
                    ALL EXCEPT item-name(s)
                    item-name(s)

          ITEM NAME IS new-item-name FROM old-item-name

SET SECTION.

     SETS ARE ALL
              ALL EXCEPT set-name(s)
              set-name(s)

     SET NAME IS new-set-name FROM old-set-name
```

Figure 3.1-4: SUBSCHEMA SYNTAX

```
QLP SECTION.

    SEGMENT NAME IS segment-name THRU SET set-name TO RECORD
    record-name

    PATH NAME IS path-name

    ROOT IS RECORD record-name TO segment-name THRU set-name
    TO RECORD record-name
```

Figure 3.1-4: SUBSCHEMA SYNTAX (cont)

B. Data Manipulation Language (DML)

1. Again similar to the CODASYL proposal the primary function of the data manipulation language is to provide an interface between the user run-unit and the data base itself. The DML makes use of a data manipulation routine and preprocessors and other various processors to produce an executable program. The DML preprocessor must be executed after the SDDL processor and before the high level language processor (see figure 3.1-4). Its inputs are the user source program, the subschema tables from the SDDL, and the record-name and data-name definitions from the SDDL. Its outputs are the high level language used by that processor and the DML working relocatable used by the collector. The 'invoke' section defines the subschema schema to be used. The working-storage section defines the record delivery and Data Management Routines (DMR) communication areas. The procedure division has the statements that effect the data base.

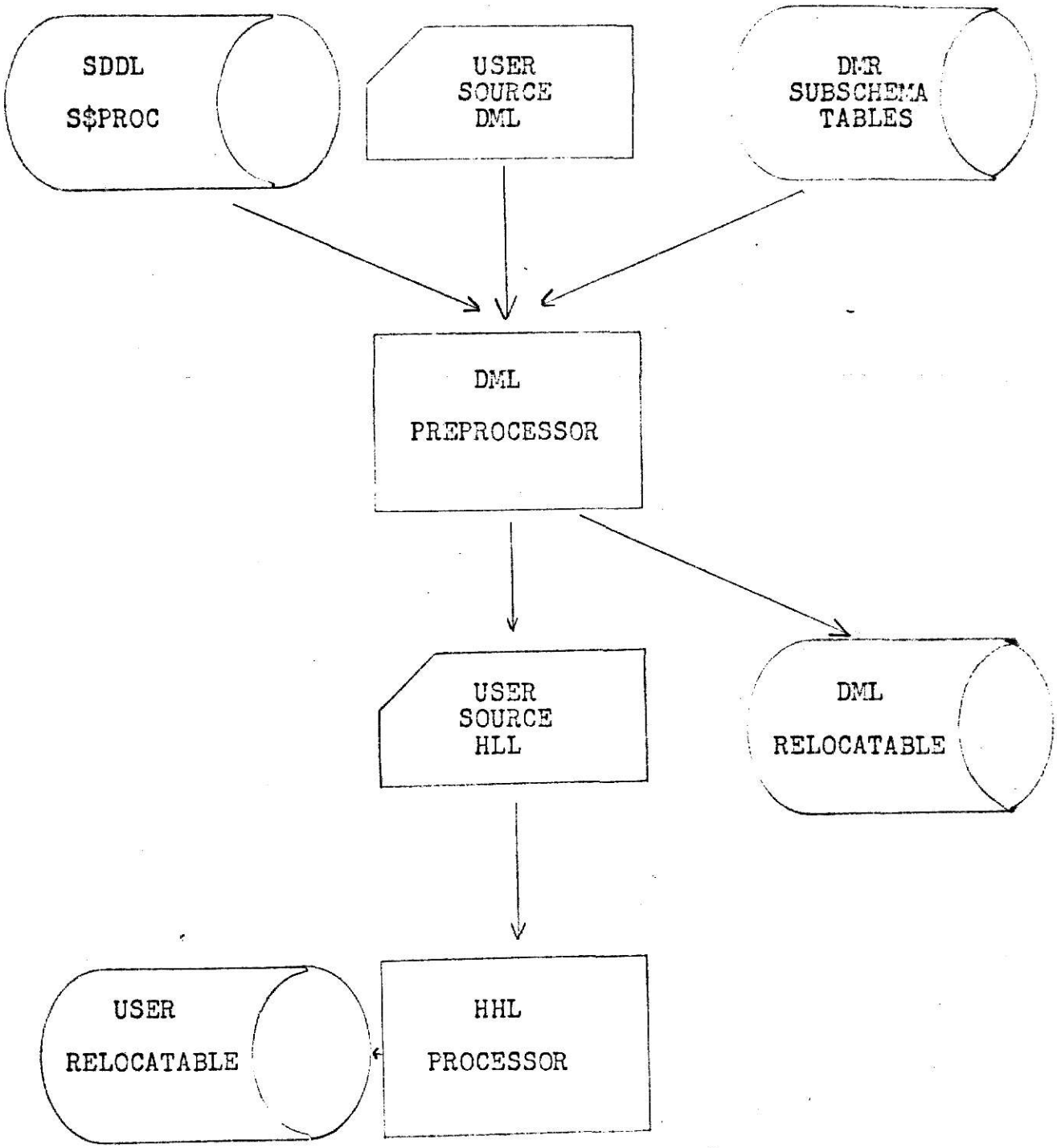


Figure 3.1-5: PROGRAM FLOW

```
IDENTIFICATION DIVISION.

ENVIRONMENT SECTION.

     SOURCE-COMPUTER.  UNIVAC-1100.

DATA DIVISION.

SUBSCHEMA SECTION.

     INVOKE SUBSCHEMA subschema-name IN FILE subschema-file
         OF    SCHEMA    schema-name

     KEY FOR INVOKE IS invoke-key

     COPYING RECORDS INTO WORKING
                          COMMON
                          LINKAGE

     COPYING ITEM-NAMES INTO WORKING
                             COMMON

     RUN-UNIT-ID IS run-unit-name

     RECORD DELIVERY-AREA IS rda-name WORKING
                                      COMMON
                                      LINKAGE

     DATA MANAGEMENT COMMUNICATION-AREA IS dmca-name WORKING
                                                     COMMON
                                                     LINKAGE

     ERROR RECOVERY IS error-paragraph

     ROLLBACK IS rollback-paragraph

WORKING-STORAGE SECTION

     01   rda-name

     01   dmca-name

COMMON-STORAGE SECTION.

     01   record-name(s)

     01   item-name(s)
```

Figure 3.1-6: COBOL SYNTAX

```
PROCEDURE DIVISION.

     "run unit commands"

         IMPART.

         DEPART.

     "area commands"

         OPEN ALL        USAGE-MODE IS INITIAL LOAD.
              area-name                 RETRIEVAL.
                                        EXCLUSIVE RETRIEVAL.
                                        PROTECTED RETRIEVAL.
                                        UPDATE
                                        EXCLUSIVE UPDATE
                                        PROTECTED UPDATE

         CLOSE area-name(s).

     "record commands"

         STORE record-name.

         GET record-name RECORD.

         MODIFY record-name RECORD.

         DELETE record-name RECORD ONLY.
                                   ALL.

         KEEP record-name record.

         FIND record-name RECORD.
         FETCH

         FIND  CURRENT RECORD WITHIN record-name RECORD.
         FETCH

         FIND  OWNER   RECORD WITHIN set-name SET.
         FETCH CURRENT                area-name AREA.
               NEXT
               PRIOR
               FIRST
               LAST

         FIND  NEXT  record-name WITHIN set-name SET.
         FETCH PRIOR                      area-name AREA.
               FIRST
               LAST
```

Figure 3.1-6: COBOL SYNTAX (cont)

```
    FIND  NEXT  WITHIN record-name RECORD.
    FETCH PRIOR
          FIRST
          LAST

    FIND  record-name VIA set-name USING

"set commands"

    ACQUIRE

    INSERT record-name RECORD INTO set-name(s).

    INSERT record-name RECORD INTO ALL SETS.

    REMOVE record-name RECORD FROM set-name(s)

    REMOVE record-name RECORD FROM ALL SETS.

"conditional commands"

    IF set-name SET EMPTY.     stmt ELSE stmt.
                    NOT EMPTY.

    IF RECORD OWNER       OF ANY      SET stmt ELSE stmt.
              NOT OWNER       set-name
              MEMBER
              NOT MEMBER
```

Figure 3.1-6: COBOL SYNTAX (cont)

C. Data Management Routine (DMR)

The Data Management Routine is a data management tool used within the context of the DML to provide the following functions:

1. core allocation - deallocation
2. input - output
3. table handling
4. rollback - recovery

Its inputs are DML commands and its outputs are the records from the data base. Figure 3.1-6 depicts a DMR map.

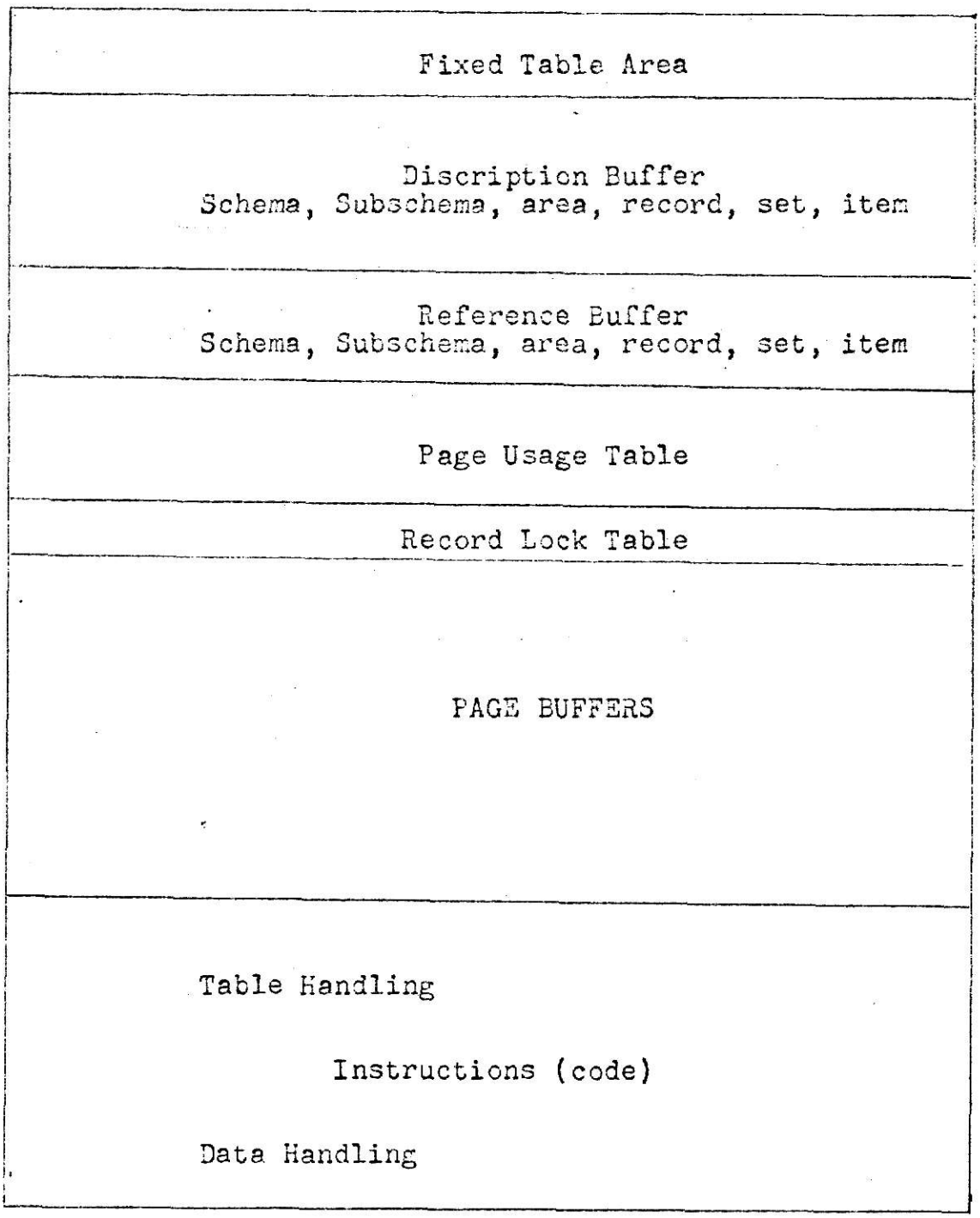


Figure 3.1-7: DMR MAP

CODASYL PROPOSAL	DMS 1100
DATA DESCRIPTION LANGUAGE	SAME
SCHEMA DEFINITION	SAME
DEFINED BY DDL	SAME
DDL COMPOSED OF 3 MAIN AREAS:	SAME
AREA SECTION	SAME
RECORD SECTION	SAME
SET SECTION	SAME
SUB-SCHEMA DEFINITION	SAME
DATA MANULATION LANGUAGE (DML)	SAME
IMPLIED IN THE PROPOSAL	DATA MANAGEMENT ROUTINES (DMR)

Figure 3.1-8: STRUCTURE COMPARISON

3.2 DMS 1100 LANGUAGE COMMANDS

This section lists the DML Commands and gives a short description of their specific functions. In addition, Figures 3.2-1 through 3.2-3 show the syntax of the various DMS 1100 Commands.

A. UPDATE COMMANDS

1. Modify

   Function: to replace the values of the data-items of the object records occurrence in the data base with the values from the record delivery area.

2. Store

   Function: to cause the values of the data-items in the record delivery area to be included in the occurrence of the object record in the data base.

3. Delete

   Function: to remove the object record from all set occurrences in which it is a member.

4. Insert

   Function: to make the object record a member of occurrences of the set-names, provided it is a member of those sets.

5. Remove

   Function: to cancel the membership of the object record in the occurrences of the set-names in which it is a member.

B. Retrieval Commands

1. Find/Fetch

   Function: to establish a record occurrence specified by a record selection expression. to optionally transfer the contents of the object record occurrence into the record delivery area.

2. Get

   Function: to transfer the contents of the object record into the record delivery area.

C. Misc. Commands

1. Free

   Function: to release all locks to enable other runs to access these specific locations.

2. Keep

   Function: to apply a lock to that location even though no alteration of that location may have occurred.

3. If

   Function: causes a condition to be evaluated, then the subsequent action depends on whether the value of the condition is true or false.

4. Move

   Function: to pass the contents of the specified currency status indicator to the run-unit as data base key or as area-name or area-key.

5. Log

   Function: to provide the user with a mechanism to save its data on the audit trail tape.

D. Control Commands

1. Impart

   Function: to register the run-unit with the DMR.

2. Depart

   Function: to inform the DMR of the termination of the run-unit.

3. Open

   Function: to indicate a run-unit's intent to access an area and the conditions associated with the access.

4. Close

   Function: to indicate the completion of access into a specified area.

The skeleton syntax of each of these commands appears in figure 3.2-1 through 3.2-3. A comparison of the CODASYL (CODA71) Proposal Commands and the DMS 1100 Commands appears in figure 3.2-4.

Syntax Skeleton:

"Run-Unit Oriented Commands"

```
IMPART [ON ERROR GO TO error-paragraph] .
DEPART [WITH ROLLBACK] [ON ERROR GO TO error-paragraph] .
FREE [ON ERROR GO TO error-paragraph] .
```

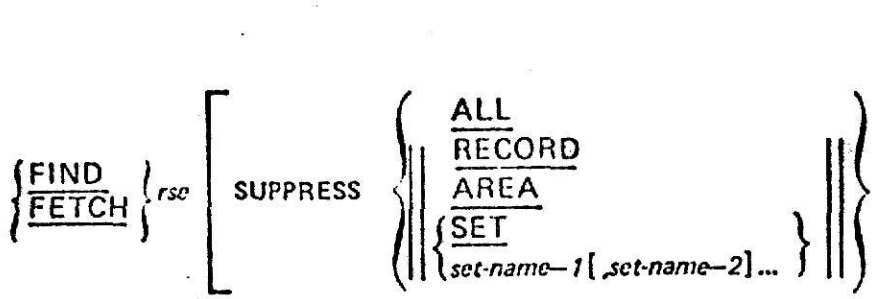
"Area Oriented Commands"

```
OPEN ALL [ USAGE-MODE IS { [ EXCLUSIVE ]  { RETRIEVAL } } ]
         [ PROTECTED ]   { UPDATE    }
         [ INITIAL LOAD ]
[ON ERROR GO TO error-paragraph] .

OPEN area-name-1 [ USAGE-MODE IS { [ EXCLUSIVE ]  { RETRIEVAL } } ]
                                 [ PROTECTED ]   { UPDATE    }
                                 [ INITIAL LOAD ]
[ , area-name-2 [ USAGE-MODE IS { [ EXCLUSIVE ]  { RETRIEVAL } } ] ]
                                [ PROTECTED ]   { UPDATE    }
                                [ INITIAL LOAD ]
[ON ERROR GO TO error-paragraph ].

CLOSE { ALL                                  } [ON ERROR GO TO error-paragraph] .
      { area-name-1 [ ,area-name-2 ] ... }
```

"Record Oriented Commands"

```
STORE record-name [SUPPRESS { ALL                                  } ]
                            { RECORD                               }
                            { AREA                                 }
                            { SETS                                 }
                            { set-name-1 [ ,set-name-2 ] ...       }
    [ON ERROR GO TO error-paragraph] .
```

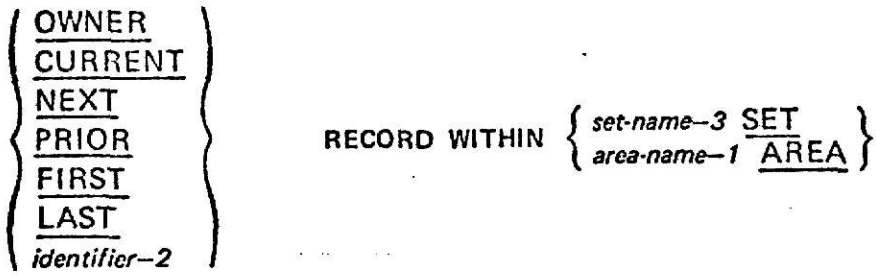
Figure 3.2-1: DMS 1100 COMMANDS SYNTAX



"Where the rse is one of the following"

Identifier–1 [,*identifier–2*]

CURRENT RECORD WITHIN *record-name–1* RECORD



[NEXT DUPLICATE WITHIN] *record-name–3* RECORD
record-name–4 VIA *set-name–5* [USING *data-base-identifier–1*
[*data-base-identifier–2*]...]
NEXT DUPLICATE WITHIN *set-name–6* USING *data-base-identifier–3*
[*data-base-identifier–4*]...

GET [ON ERROR GO TO *error-paragraph*].
MODIFY [*data-base-identifier–1* [,*data-base-identifier–2*]...]
[ON ERROR GO TO *error-paragraph*].
DELETE [ONLY / ALL] [ON ERROR GO TO *error-paragraph*].
KEEP [ON ERROR GO TO *error-paragraph*].

"Set Oriented Commands"

INSERT INTO *set-name–1* [,INTO *set-name–2*]... [ON ERROR GO TO *error-paragraph*].
INSERT INTO ALL SETS [ON ERROR GO TO *error-paragraph*].
REMOVE FROM *set-name–1* [,FROM *set-name–2*]... [ON ERROR GO TO *error-paragraph*].
REMOVE FROM ALL SETS [ON ERROR GO TO *error-paragraph*].

Figure 3.2-2: DMS 1100 COMMANDS SYNTAX

"Conditional Commands"

```
IF set-name-1 SET [NOT] EMPTY;
    {statement-1 | NEXT SENTENCE}  [ELSE {statement-2 | NEXT SENTENCE}].
IF RECORD [NOT] {OWNER | MEMBER} OF {set-name-2 | ANY} SET;
    {statement-3 | NEXT SENTENCE}  [ELSE {statement-4 | NEXT SENTENCE}].
```

"Supporting Commands"

```
LOG data-name-1 WORDS [FOR RECOVERY POINT]

MOVE CURRENCY STATUS FOR { RUN-UNIT
                         { record-name RECORD
                         { area-name AREA
                         { set-name SET
    TO identifier-1 [ON ERROR GO TO error-paragraph].

MOVE || AREA-KEY | AREA-NAME || FOR { RUN-UNIT
                                    { record-name RECORD
                                    { area-names AREA
                                    { set-name SET
                                    { identifier-2
    TO || identifier-3 | identifier-4 ||  [ON ERROR GO TO error-paragraph].
```

Figure 3.2-3: DMS 1100 COMMANDS SYNTAX

CODASYL IMPERATIVE STATEMENT	DMS 1100 STATEMENT
DELETE	Same
CLOSE	Same
FIND	Used in conjunction with fetch inst.
FREE	Same
GET	Same
IF	Same
INSERT	Same
KEEP	Same (except DMS 1100 usage emphasizes the locking condition).
MODIFY	Same
MOVE	Same
OPEN	Same
ORDER	Not present in DMS 1100 repretoire.
REMOVE	Same
STORE	Same
USE	Not present in DMS 1100
Not present in DBTG Proposal.	FETCH
Not present in DBTG Proposal.	LOG
Not present in DBTG Proposal.	IMPART
Not present in DBTG Proposal.	DEPART

Figure 3.2-4: COMPARISON OF COMMANDS

3.3 DMS 1100 SUPPORT FEATURES

This section briefly lists and describes some of the support features and utilities available with the DMS 1100. Figure 3.6-1 displays a comparison of the general features of DMS 1100 with the CODASYL (CODA71) proposal recommendations.

A. Query Language Processor (QLP 1100)

QLP 1100 is an English-Language inquiry system that allows inquires to be made to data bases generated under DMS 1100. It uses a command language designed around a simplified english language Syntax and requires a minimum knowledge of the DMS 1100 data base structure. Its two major component modules, the Scan Parser, which analyzes incoming commands, and the Task Translator, which accesses the data bases are both re-entrant.

B. An auxiliary function of DMS 1100 provides the data dictionary contents. The contents of the data dictionary are described and given at various levels such as:

1. schema
2. areas
3. sets
4. data items alphabetically

Future additions to the data dictionary will be:

1. All data items that appear as input to a program transaction.
2. All data base records accessed by a program transaction.
3. All program transactions that contain a specific

data item.

4. All program transactions that contain a specific data base record.

C. DMS 1100 Utility Processor

This is a processor available with the DMS 1100 that will handle the normal needed file handling and management functions such as:

1. Page compaction/expansion
2. Area initialization
3. Data printing
4. Loading/unloading
5. Other common verification and error handling procedures

The CODASYL Proposal states that the specifications for a "Complete DBMS" should include utility and service routines that are required to support a data base in day to day operation. The support features described above fall into this category.

3.4 SECURITY AND INTEGRITY

Sperry Univac's stated objective for data base security for the DMS 1100 system is to preserve the accuracy of the data from possible destruction. The data (individual items, records, sets and area) are subject to destruction or incorrect modification by hardware, software, or operational errors. In addition, DMS 1100 provides facilities for restoration of the data base after such errors have occurred.

SECURITY

The CODASYL proposal suggests that privacy locks should be available at the schema level, subschema level, and various "lower" data levels.

The DMS 1100 DBMS made no provision to incorporate privacy locks at any level in their implementation. In later releases of DMS 1100 they did, however, make available some privacy features.

INTEGRITY

The vehicle for providing data base integrity (recovery and rollback) is the page "LOOK". A LOOK is a copy of a data page which is written to a file for use at a later point in time if it becomes necessary to re-establish the contents of the data base.

The types of LOOK available are:

1. Before LOOK

   The basic information in a before LOOK is a copy of

the data base prior to alteration. The before LOOKS are used to move backward over time to remove the effects of an error on the data base. The before LOOK is written to an audit trail which is normally a sequential file (e.g., tape). A data management routine (DMR) writes control information along with the copy of the page for the before LOOK.

2. After LOOK

   The basic information in an after LOOK is a copy of the data page after the alteration has been made. After LOOKS are used to move forward over time to re-establish the data base to a stable state. Similarly to the before LOOKS, the DMR appends control type information and writes the after LOOK to an audit trail tape. This write is performed in conjunction with the write of the page to the data base. An after LOOK is taken whenever the modified page is returned to the data base - not every time the page is modified. If a page is not altered, it is overlaid without being written back to the data base. A page is only written back to the data base when it has been modified and one of the following conditions exist:

   a. it must be swapped to make room for another page
   b. the run unit terminates
   c. a free command is issued.

3. Quick-before LOOK

The quick-before LOOK contain the same information as the before LOOK. However, the quick-before LOOK are written to a file placed on a fast device (something other than a tape). This file is segmented by run unit and is specifically available to provide a capability for command and run unit rollbacks.

The type of LOOK desired can be specified on the area basis in the schema. The emphasis on data integrity in the DMS 1100 system is at the data base level as opposed to the run unit level. Consequently, the responsibility for determining what types of data base protection is to be employed and what special actions are required are determined at the data administrator level (system generation and schema description) as opposed to having the responsibility for data integrity at the run unit level. Only limited control is left to the user program, e.g., specifications of command quick-before LOOKS vs. run unit quick-before LOOKS.

Following the DBTG guidelines, the subschema limits the run-unit's view of the data to only that defined in the subschema. In addition, DMS 1100 provides for the use of access lock-keys to be set and checked on areas defined as such.

3.5 EXECUTION

Univac maintains that their DBMS implementation (DMS 1100) represents a combination of features that allow users standardized data management techniques and yet operates in an optimized manner.

The DMS 1100 contains a Data Reorganization Utility (DRU) that provides for the optimization of the physical placement of records within the data base without the need for tailored unload and reload programs. The DRU consists of two modules: a reorganization syntae analysis (RSA) module, which accepts reorganization specifications and the data base schema as input; and a reorganization module (ReORG), which accomplishes the reorganization directly against the data base in an optimized manner.

DMS 1100 also has the capability for invoking modules that allow run-unit statistics that give a data base administrator tools for studying and making decisions from the run-unit statistics that will keep the data base management system operating at maximum efficiency.

Of course, the invoking of these modules that create the many run-unit statistics will slow normal execution time somewhat but not significantly.

The DBTG Proposal makes no statements concerning efficiency or

timing constraints that a particular data base management system should maintain.

3.6 OTHER COMPARISON CONSIDERATIONS

Other considerations such as ease of use and portability were investigated as far as the DMS 1100 was concerned. These areas, although interesting to consider, do not lend themselves for easy comparison to the CODASYL (CODA71) proposal.

The CODASYL proposal makes no recommendation concerning portability. The DMS 1100, although compatible with the CODASYL proposal, is not directly compatible with any other architecture except the Univac 1100 series processors.

The DMS 1100, by being compatible with CODASYL, is not at all difficult for the user to master assuming some prior knowledge of DBTG language is present. In addition, the learning process is helped by the Query Language Processor, (QLP), which is oriented to the non-data processing person. Also, Univac has a Remote Processing System (RPS 1100) which provides access to the system resources by non-programming oriented users. RPS 1100 is an interactive data management and file processing system. The RPS 1100 data base files are created and maintained under DMS 1100.

CODASYL PROPOSAL	DMS 1100
SECURITY	
PRIVACY LOCKS	ONLY AVAILABLE IN LATER RELEASES
INTEGRITY	SAME
IMPLIED IN THE PROPOSAL	EXECUTION DMS 1100 CONTAINS A DATA REORGANIZATION UTILITY (DRU) WHICH PROVIDES OPTIMIZATION FUNCTION
NO SPECIFIC RECOMMENDATION	PORTABILITY DMS 1100 IS COMPATIBLE ONLY WITH SPERRY UNIVAC 1100 SERIES PROCESSORS
NO SPECIFIC RECOMMENDATION	EASE OF USE QUERY LANGUAGE PROCESSOR (QLP) REMOTE PROCESSING SYSTEM (RPS 1100)
NOT PROPOSED	QUERY LANGUAGE PROCESSOR (QLP 1100)
NOT PROPOSED AS SUCH	DATA DICTIONARY
IMPLIED	DMS 1100 UTILITY PROCESSOR

Figure 3.6-1: COMPARISON OF SUPPORT FEATURES

CODASYL standards or not to follow CODASYL standards? Sperry Univac apparently chose to design a DBMS to be as compatible as possible with the DBTG proposal. This is actually not surprising in that:

A. DMS 1100 was announced and released in late 1972 just a year after the landmark CODASYL publication.

B. Univac was a member of the DBTG itself and participated in the development of the data management specifications.

C. It is stated by the corporation that "Univac's implementation of the 1100 series Data Management System (DMS 1100) is based on the DBTG specifications."

What were the potential advantage for Univac to design a CODASYL compatible DBMS? These advantage include:

A. potential DBMS portability or independence from specific hardware.

B. learning transference occurs for an individual familiar with one CODASYL DBMS when learning another CODASYL DBMS.

C. the protection and marketing pressure offered by a wide installed DBMS - an estimated over 4,000 users of CODASYL DMBS.

On the basis of DMS 1100, it appears that a committee of computer vendors, business and government can make a very notable contribution to the data processing community. The expected future influence of the CODASYL is somewhat uncertain, depending on growth and strength of the newer DBMS, the relational model,

and data base machines.

However, there is a large base of current CODASYL DBMS users and not to be overlooked is that having a system wrapped around a "standard" means diminished dependency on one vendor. The industry seems to be moving along these lines and probably that is what is likely to survive for the next three to five years.

REFERENCES

(FORM80) Forman, E.H., "What Management Should Know About DBMS", Computer World, November 1980

(CODA71) CODASYL, "Data Base Task Group Report". Association for Computing Machinery, New York, N.Y., April 1971

(DATA78) Datapro Research Corp., "Data Pro 70", Delran, New Jersey, December 1978

(JARD73) Jardine, Donald A., "Data Base Management Systems", North-Holland Publishing Co., 1973

(SPER79) Sperry Univac, "DMS 1100 DML Programming", Princeton, New Jersey, 1979

(SPER72) Sperry Univac, "DMS 1100 Schema Definition", Princeton, New Jersey, 1972

COMPLIANCE WITH CODASYL STANDARDS: A CASE STUDY

by

C. LARRY KEELER

B. A., Kansas University, 1963

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

1981

A comprehensive comparison report was made detailing the similarities and differences of the CODASYL DATA BASE TASG GROUP 1971 report and that of Sperry Univac's Data Base Management System, the DMS 1100.

This report describes in some detail the features of the CODASYL DBTG Proposal as well as the general features of the Univac DMS 1100, then makes a comparison of the features of each. The report shows in detail the specific areas Univac's DMS 1100 follows the proposal guidelines and those areas in which they do not comply with the proposal. After the comparisons, conclusions are made showing to what extent this major vendor was influenced by the publishing of the 1971 proposal. The report predicts the future influence the CODASYL's committee, the Data Base Task Group, is likely to have on the data base community based upon this commercial vendor compliance thus far.