Title
BDMS: BERKELEY DATA-BASE MANAGEMENT SYSTEM USER'S MANUAL (VERSION 1.2)

Author
Richards, David R.

Publication Date
1976-04-01

BDMS

Berkeley Database Management System

USER'S MANUAL

(Version 2.2)

DISCLAIMER

LBL-4683
(Revised

BDMS

Berkeley Database Management System

USER'S MANUAL

(Version 2.2)

David R. Richards

Computer Science and Applied Mathematics
Department

Lawrence Berkeley Laboratory

University of California

Berkeley, California 94720

WARNING

This manual is preliminary and in a few cases describes features of BDMS that are not yet fully implemented. In particular,

1. double precision and bit string type data elements may be used procecurally, but not in the editor and query user languages,

2. DISPLAY and SUPPRESS commands are not yet implemented,

3. the DUMP, CLEAN, and BALANCE utilities are not yet available.

ACKNOWLEDGEMENTS

BDMS had its origin in the joint development of a Particle Physics Data System by the Berkeley Particle Data Group and the Caltech Data Compilation Group. The diversity of databases that make up that system necessitated the use of a common database management system, while the complexity of the data demanded capabilities not available in any existing DBMS. The new system which was developed to meet this need has proven to be of general usefulness, and responsibility for its continued support and further development has been assumed by the Computer Science and Applied Mathematics Department at LBL.

As the system evolved, many people contributed ideas and assisted with programming. They include Tricia Coffeen, Paul Chan, Geoffrey Fox, Marge Hutchinson, Tom Lasinski, Deane Merrill, Gill Ringland, Alan Rittenberg, Silvia Sorell, Paul Stevens, Tom Trippe, Vicky White, and George Yost.

The users of BDMS also deserve thanks for contending with the seemingly endless change to which a developing system is subject. Their comments, criticism, and unfailing ability to flush out even the most recluse bugs have been and continue to be invaluable.

This work was supported by the U. S. Department of Energy (formerly U. S. Energy Research and Development Administration and U. S. Atomic Energy Commission), the National Science Foundation, and the National Bureau of Standards.

TABLE OF CONTENTS 08 FEB 78

1.1 INTRODUCTION

BDMS is a general-purpose database management and information retrieval system with a broad range of capabilities for creating, maintaining, and accessing computer databases. It frees programmer and end user alike from concern with the physical storage of data, instead making it possible to deal with data at a logical level. In other words, it allows one to deal with information rather than data.

Many applications, which previously would have required extensive special program development, can be handled by the system with little, if any, extra software. Its capabilities include -

1. A natural and easy-to-use database definition language.

2. A powerful editor that operates directly on the database. Data are entered in a simple, natural way, and many short-cuts are available to expedite the entry of large amounts of data. Modifications to existing data are immediately effective.

3. Extensive retrieval facilities including controlled database inversion, Boolean and relational operators, nested parentheses in search expressions, truncated and range search, and reference to previously retrieved sets.

4. A standard listing format that makes the structure of the data readily apparent. When working interactively, retrieved records may be listed at the terminal or printed off-line. They also may be dumped in a format suitable for data exchange and database reloading.

5. A common command and data language for both batch and interactive use.

6. Exits to user-supplied routines at several places in the system to allow input data validation, data transformation on input, output, indexing, searching, and automatic data element generation.

7. Utilities for loading and dumping databases and database space maintenance.

A designer wishing to base a special-purpose system on BDMS will find it relatively easy to use just those modules needed. All system capabilities are available via FORTRAN subroutine calls. For example-

1. Special software might be written to build a database from data initially stored in some special format, e.g. fixed field card images. This could be done by writing a program which reads data in the special format and calls appropriate subroutines to load it directly into the database, bypassing the standard data input language. In fact, such a routine could be coded as a replacement for the standard input routine with the system behaving normally in all other

respects.

2. The system could be modified through the provision of a replacement for the standard data output routine so that retrieved data would be listed in a format better suited to its end use.

3. A very ambitious system designer might choose to replace the entire user interface with one providing extensive prompting and data validation tailored to his specific application.

In all of these cases, the designer/programmer is freed from concern with physical storage mechanisms for his data. Instead, he can use existing software to access and modify the data, referring to it by name.

BDMS is highly modular and coded for the most part in machine-independent FORTRAN IV. The operating system interface and machine-dependent code are isolated in a few modules so that the system is easily transportable.

1.2 DATABASE STRUCTURE

A BDMS database is structured into records, the units in which data pass between the system and disc storage. Normally, a record will have some significance to the user, e.g. a record in a bibliographic database would be a description of a single document, but this is not always necessary or desirable.

The individual data items within a record are called data elements; they are the smallest units of data with any meaning to the system, although an individual data element might have some internal structure known to an application program. A data element has a unique name and is normally referenced by name (or a synonym). There is essentially no limit on the number of data elements that may be defined for a database.

A hierarchical structure may be imposed on the records when a database is defined. This means that some data elements are declared to be subordinate to other, parent, data elements. Those data elements for which no parent is declared are called record-level data elements; it is often useful to consider the record itself to be their parent. There is essentially no limit on the number of levels which may be defined in the record structure.

Within a given record, each record-level data element may occur once, several times, or not at all. Likewise, each occurrence of a data element at any lower level in the hierarchy may have linked to it one, several, or no occurrences of each of its subordinate data elements. There is essentially no limit on the number of times any data element may occur in a single record. Furthermore, there is no storage overhead associated with data elements that do not occur at all in either the record or a particular occurrence of their parent.

Data elements are classified into six types according to the values they can assume - character or bit string, integer or real (single or double precision) vector, and pure node. Character or bit strings may be of any length with no limit beyond that imposed by run-time memory restrictions. Integer or real (floating point) data elements may be scalars (single numbers) or arbitrary length vectors (i.e. an ordered set of numbers, which are the components of the vector). Real data elements may be single or double precision. Pure node data elements carry no value; they may be used to link together subordinate data elements in the record hierarchy or as flags.

Any data element, regardless of type, may serve as a node in the hierarchical record structure. In general, if one of a group of related data elements may occur only once in each occurrence of the group, it should be made the parent of the rest of the data elements. However, if all of the data elements in such a group may occur multiply, they should all be linked to a pure node parent.

Any data element may be declared to be a record key. The system will then maintain an index for that data element to allow efficient retrieval. In an index, key values have a fixed length that is declared in the database definition. Data element values are

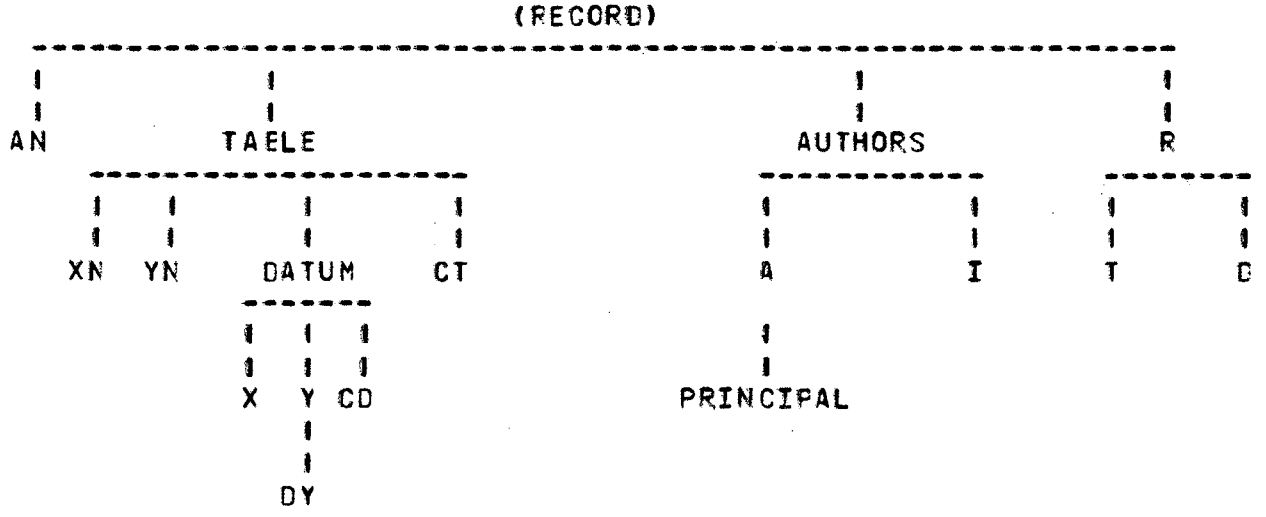
truncated or padded as necessary when they are put into an index.

The system assigns a record ID to each record as it is created. This guarantees that each record has a unique identifier by which it can be selectively retrieved even if none of its data elements is defined to be a key so that no indices are maintained.

Physically, a database is divided into a data file, which contains the database definition and data records, a directory file used to access records in the data file, and an inversion file comprising the indices (if any exist).

1.3 EXAMPLE DATABASE

Throughout this manual, we shall use for illustration a hypothetical database whose records are summaries of some kind of experiment reports and whose structure may be diagrammed as follows -



The cata elements have the following meanings -

AN is a document accession number.

TABLE is the name of a collection of data. It links together definitions of the X and Y variables, XN and YN, possible comments on the table, CT, and the actual data points in the table, DATUM. DATUM is a pure node data element that links X and Y values and possible comments on the datum, CD. DY is the error on Y. Ir a given record, TAELE might occur multiply; in a given TAELE, DATUM and CT might occur multiply; in a given DATUM, CD might occur multiply. The rest of the data elements, XN, YN, X, Y, and DY would normally occur only once within a given occurrence of their parent data element.

AUTHORS is a pure node that links one or more authors, A, with their institutioral affiliation(s), I. In general, all of these data elements might occur multiply. Note how PRINCIPAL, a pure node type data element, is used to flag a particular author as the principal author. Normally, only one author would be so flagged.

R is the cocument reference (e.g. report number) and T and D are its type and cate.

A diagram of this type only shows the hierarchical relationships among cata elements. In general, any of the data elements may occur several times or not at all in any particular recorc. Thus, the pictorial representation of a real record would require the use of a third dimension to cisplay the multiple occurrences.

1.4 SYSTEM ORGANIZATION

From the user's point of view, BDMS is divided into several functional modules.

The database definition compiler is a separate program used to create a new database before any data are loaded into it. The same editor used for data input and modification is used to construct a database definiton that is converted by the compiler into tables that will drive the rest of the system when the new database is used. The database definition language and use of the compiler are the subject of Chapter 2.

The executive is the overall control program. Some executive commands are executed immediately while others cause control to be passed to other system modules, which will then interpret additional user input. The executive functions are discussed in Chapter 3.

The editor provides facilities for creating and modifying data records. It is controlled by a set of editor commands and understands the system's external data language. The use of the editor is described in detail in Chapter 4.

The query subsystem, described in Chapter 5, interprets user queries and selectively retrieves data records for subsequent display or modification.

The utilities, described in Chapter 6, are stand-alone programs for database maintenance operations such as efficient loading and dumping and removal of the dead space resulting from update activity.

1.5 USER EXITS

As mentioned in the introduction, exits to user routines are provided at several places in the system, allowing it to be tailored to specific databases and applications.

The subroutines that may be supplied by the user are described briefly below. Details of their use and calling parameters may be found in the Programmer's Manual.

IPROC is passed a data element value before it is stored in the record. IPROC may modify the value or check it for validity. If a value is found to be in error, IPROC can instruct the calling routine not to store it, and/or output an error message.

OPROC is passed a data element value before it is output. It may modify the value, check it for validity, or suppress its output altogether. Normally, IPROC and OPROC would be used as a pair to transform between the internal and external forms of a data element.

KPROC is passed the value of a key data element before it is stored in an index (or deleted from an index when restoring a modified record or adding a record). It may modify the value or check for validity. It can suppress the storage of a key in an index.

KMAP is passed the value of a CHAR type key data element before it is stored in or deleted from an index. It may perform an arbitrary character code mapping in order to enforce a desired collating sequence for values in the index.

QPROC is passed the value of a key data element before an index search. Normally it would transform the value in the same way as KPROC, (it might consist of no more than a call to KPROC) but it is provided to allow the use of different forms of a key value during data input and retrieval.

SPROC is called before a record is stored in the database. It may perform an intra-record data integrity check, and generate or modify data element values. It can prevent the storage of the record if errors are detected.

FPROC is called just after a record is fetched from the database. Its primary use is to rematerialize any virtual data element occurrences that the database designer wishes to make visable to the user.

The IPROC, OPROC, KPROC, and QPROC routines are called only if the database definition instructs the system to do so. KMAP, SPROC and FPROC are always called but, of course, may be do-nothing routines.

2.1 INTRODUCTION

To set up a BDMS database, one must define the nature of the data, e.g. names and types of data elements, their hierarchical relationships, what indices are to be maintained, etc. The system must be informed of any user-supplied routines for input processing and validation, output, key or query processing.

All of this information is coded in a database definition language which is processed by the database definition compiler. The output of the compiler is a binary file definition table (FDT) which becomes the zero'th record of the newly-defined database and controls the operation of the system while that database is being accessed.

Plans exist for extending the compiler to provide facilities for modifying the definition of an existing database, e.g. adding or deleting data elements and synonyms.

2.2 DATABASE DEFINITION LANGUAGE

The database definition language of BDMS has the same syntax as the data language used by the editor (c.f. Chapter 4). That is, it consists of a series of statements of the form

<attribute label> = <value>;

By way of illustration, a file definition for the example database described in Chapter 1 would be coded as follows, assuming that it is desired to index the records by the document accession number, AN, the individual authors, A, and the type of data reported, YN -

```
FILE=EXPT-REPORTS;
DE.=AN; SYN=ACCESSION-NO; TY=INTEGER; KEY; LENGTH=1;
DE.=TABLE; TY=CHAR;
DE.=DATUM; PAR=TABLE; TY=NODE;
DE.=X; SYN=X-VALUE; PAR=DATUM; TY=REAL;
DE.=Y; SYN=Y-VALUE; PAR=DATUM; TY=REAL;
DE.=DY; SYN=Y-ERROR; PAR=Y; TY=REAL;
DE.=CD; SYN=COMMENT-DATUM; PAR=DATUM; TY=CHAR;
DE.=XN; SYN=X-NAME; PAR=TABLE; TY=CHAR;
DE.=YN; SYN=Y-NAME; PAR=TABLE; TY=CHAR; KEY; LENGTH=1;
DE.=CT; SYN=COMMENT-TABLE; PAR=TABLE; TY=CHAR;
DE.=AUTHORS; TY=NODE;
DE.=A; SYN=AUTHOR-NAME; PAR=AUTHORS; TY=CHAR; KEY; LENGTH=3;
DE.=I; SYN=INSTITUTION; PAR=AUTHORS; TY=CHAR;
DE.=PRINCIPAL; PAR=A; TY=NODE;
DE.=R; SYN=REFERENCE; TY=CHAR;
DE.=T; SYN=TITLE; PAR=R; TY=CHAR;
DE.=D; SYN=DATE; PAR=R; TY=CHAR;
```

The meaning and allowed values of the database definition attributes are -

FILE=<file name>;

This is not yet used by the system.

DE=<preferred data element name>;

This is the name that will be used when the data element is output. Names must be unique within a database. They may not contain embedded blanks, equal signs, or semicolons. The size of the FDT will be minimized and the editor's processing of input to the defined database will be slightly more efficient if the data element names all have similar lengths (i.e. all fit into the same number of computer words).

SYN=<synonym for data element name>;

Synonyms may be used interchangeably with the preferred data element name for input and in queries. They must satisfy the same rules of construction. Again, it is preferable for all synonyms to have similar lengths.

PAR=<parent data element preferred name>;

This specifies the parent to which a data element will be linked in the hierarchical record structure. The parent data element must have been defined previously. PAR is omitted for record-level data elements.

TY=<data element type>;

Allowed values are -

INTEGER	integer (vector)
REAL	real (vector)
DOUBLE	double precision real (vector)
CHAR	character string
BIT	bit string
NODE	pure node

VIRTUAL;

If this follows a DE specification, the data element will be discarded when a record is stored in the database. It may be input, however, and may be used as a KEY. It will still exist in the record when the SPROC routine is called. If it is desired that a VIRTUAL data element be rematerialized when a record is fetched from the database, this may be done by the FPROC routine.

KEY;

If this follows a DE specification, an index will be maintained for the data element.

LENGTH=<key length in words>;

This may follow a KEY attribute. If absent, the key length defaults to 1 word.

IPROC=<format of DE value>;

If this follows a DE specification, the IPROC routine will be called by the editor when the data element is input. Allowed format specifications are -

EXTERNAL	The DE value is passed to and is returned by IPROC in external format, i.e. unpacked character string.
INTERNAL	The DE value is passed to and returned by IPROC in internal format, i.e. internal binary numeric representation, packed character string, or bit string.
CONVERT	The DE value is passed to IPROC in external format and returned by IPROC in internal format.

OPROC=<format of DE value>;

If this follows a DE specification, the OPROC routine will be called by the list routine before the data element is output. Allowed format specifications are the same as for IPROC except that the CONVERT has the opposite meaning, i.e.

CONVERT The DE value is passed to OPROC in internal format and returned by OPROC in external format.

KPROC=<format of DE value>;

If this follows a DE specification, the KPROC routine will be called prior to storing a DE value in an index. Allowed format specifications are the same as for IPROC.

QPROC=<format of DE value>;

If this follows a DE specification, the QPROC routine will be called by the query interpreter when it encounters the DE in a query. Allowed format specifications are the same as for IPROC.

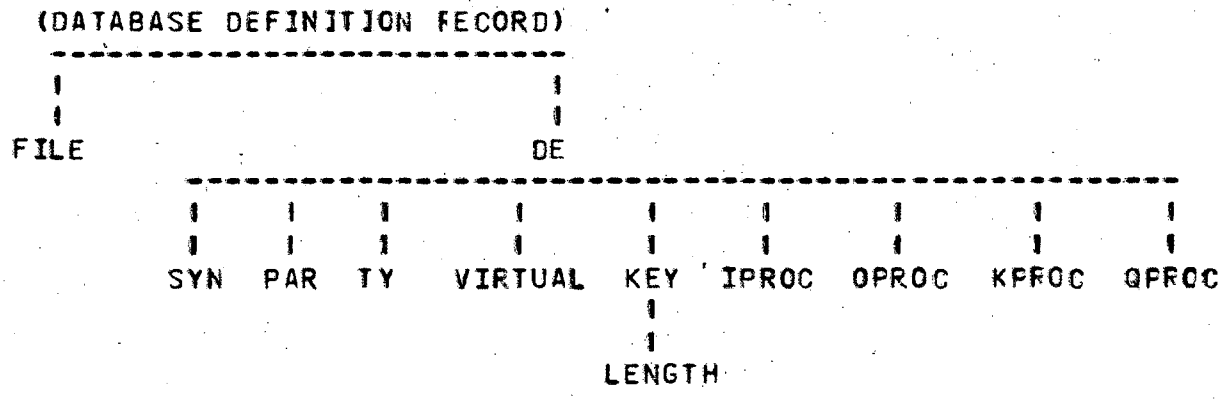
IPROC, OPROC, KPROC, and QPROC are not valid attributes for a pure node type DE.

2.3 COMPILER COMMANDS

The commands necognized by the compiler are described ir this section. Only the first four letters of a command are necessary for recognition.

DEFINE

This command invokes the editor, which initially will be in append mode, so that a new database definitior may be input. This creates a database definition record in core which has the following hierarchical structure -



VIRTUAL and KEY are pure node type and LENGTH is integer type. All other data elements are CHAR type.

As this record is being input, errors or omissions may be remedied using the edit mode commands *I, *R, *S, and *D just as if it were a data record. It may be listed at any time with the editor command *L. When the definition is complete, the editor must be terminated by a ** command.

COMPILE

This command initiates compilation of the database definition created by a preceding DEFINE command. The compiler responds with

COMPILING DEFINITION

followed by either

COMPILATION SUCCESSFUL

or a list of errors detected and

COMPILATION TERMINATED

MODIFY

This command invokes the editor to allow correctior of the errors detected in a preceding compilation. When the changes have been made,

the editor must be terminated by a ** command. It differs from a DEFINE command in that DEFINE will completely erase an existing definition record while MODIFY allows changes to an existing definition record.

LINE,<n>

This sets the input/output line length to <n> characters. The default is 80 characters.

STOP

This terminates execution of the compiler. If the definition has been compiled successfully, an empty database will exist ready for the addition of data records.

The audit facility is always turned on when the database definition compiler is run; all input to the system from the INPUT file (the terminal when running interactively) is echoed on the AUDIT file. This provides a record of the definition that may be edited and used as batch-mode input to the compiler if the definition needs to be changed later. This copy of the definition might also be saved so that it could be used when reloading a dumped copy of the database, e.g. following transmittal to another site.

2.4 COMPILER ERROR MESSAGES

The error messages that may be generated by the database definition compiler are summarized in this section. The prefix ***ERROR***, which is common to all messages, has been omitted.

ILLEGAL RECURRENCE OF RECORD LEVEL ATTRIBUTE

One of the attributes FILE, or DE has recurred without a following period. It is ignored.

INVALID COMMAND

A command has been input whose first four letters do not match any of those in Section 2.3.

INVALID PARENT SPECIFIED FOR DE.n

The parent specified for DE.n has not been defined prior to defining DE.n or is misspelled.

INVALID TYPE SPECIFIED FOR DE.n

Valid types are INTEGER, REAL, DOUBLE, CHAR, BIT, or NODE.

INVALID IPROC TYPE SPECIFIED FOR DE.n

INVALID OPROC TYPE SPECIFIED FOR DE.n

INVALID KPROC TYPE SPECIFIED FOR DE.n

INVALID QPROC TYPE SPECIFIED FOR DE.n

Valid types are EXTERNAL, INTERNAL, or CONVERT.

NO DATA ELEMENTS DEFINED

The database definition contains no DE attribute specification statements at all.

NO TYPE SPECIFIED FOR DE.n

Every data element must have a type specification.

PROC SPECIFIED FOR PURE NODE DE.n

Since a pure node type data element carries no value, it is meaningless for a processor to operate on it.

WORK SPACE EXCEEDED -- COMMAND SKIPPED

The database definition is too large to compile in the available work space. The compiler must be recompiled with a larger work space before proceeding.

WORK SPACE EXCEEDED -- RUN ABORTED

There was not even enough work space to initialize the system. This is an unlikely occurrence indicative of system malfunction.

3.1 INTRODUCTION

The BDMS executive is a self-contained facility that allows a user to access and maintain an existing database. Commands are provided to add new records to the database, to search for records satisfying stated criteria, and to subsequently display, print, dump, modify, and delete such records. In addition, several commands allow user control over certain aspects of the system's operation such as input/output, line length, and whether to maintain an audit trail.

An overview of the executive will be presented in this chapter, and the use of most commands will be described in detail. Two areas are sufficiently involved to warrant separate treatment in the following two chapters - the editor and query language.

3.2 EXECUTIVE COMMANDS

All executive commands take the form of an English-language verb followed optionally by a comma and an integer identifying the record to be acted upon. The commands are terminated by a blank and hence may not contain embedded blanks.

Whenever the executive is waiting for a user command, it issues the message

ENTER COMMAND

The command verbs recognized by the executive are -

ADD	Add new records to the database.
FIND	Retrieve and make current a set of records.
SET	Make a previously created set current.
PURGE	Purge all previously created sets or the last set created.
LIST	List at the terminal the current set or a selected record.
PRINT	Print off-line the current set or a selected record.
DUMP	Dump the current set or a selected record in external format.
DISPLAY	Set default list of data elements to be listed, printed, or dumped.
SUPPRESS	Set default list of those data elements not to be listed, printed, or dumped.
MODIFY	Modify a record selected from the current set.
DELETE	Delete from the database a record selected from the current set.
LINE	Set input/output line length.
AUDIT	Turn on audit facility (default).
NOAUDIT	Turn off audit facility.
STOP	Terminate program execution.

In somewhat more detail, these commands function as follows.

ADD

This command informs the executive that one or more new records are to be added to the database. An empty record will be created and assigned the next available record ID. Control is then transferred to the editor so that the user may enter data in the record(s). (See Chapter 4 for instructions on the use of the editor.) When the user exits from the editor (with a ** or *C editor command), the number of records added to the database is reported in the form

<n> RECORD(S) ADDED

FIND

This command invokes the query subsystem and is followed by a condition terminated by a ** delimiter. The database will be searched for records satisfying the condition and if any are found, they form a new current set on which subsequent LIST, PRINT, DUMP, MODIFY, or DELETE commands will act. The set identifier and number of records in

the set are reportec to tte user in the form

<n> RECORD(S) IN SET <s>

If <n>= 0, no set is createc and <s> is omitted. The format of queries will be covered ir detail in Chapter 5.

SET,<set number>

This command makes a previously-created <set number> the current set on which subsequent LIST, PRINT, DUMP, MODIFY, and DELETE commands will act.

PURGE,<set number>

This command purges a previously-created <set number>, freeing the disc and work space it uses. If <set number> is omitted, all existing sets are purged. (At present, only the last-created set, or all sets, may be purged with this command.)

LIST,<n>,<data element name 1>,...<data element name i>

This command lists the n'th record in the current set on the system file LOG (which is the terminal when running interactively). If the number <n> is omitted, all records in the set will be listed.

Each data element occurrence will begin on a new line and will be numbered if several occurrences of that data element are linked to the same parent occurrence, or if it is a record level data element which occurs more than once. Subordinate data elements will be listed following the parent occurrence to which they are linked and indented according to their level in the record structure.

If the list of data element names is omitted, the last DISPLAY or SUPPRESS command issued controls which data elements are displayed. If no DISPLAY or SUPPRESS command has been issued, all data elements are displayed.

If a list of data element names separated by commas follows the command verb (and record number <n>, if present), only those data elements in the list will be displayed. The entire command, including the list of cata element names, must not contain embedded blanks.

PRINT,<n>,<data element name 1>,...<data element name i>

This is identical to LIST, except that the record(s) is listed on the system file PRINT, which can be disposed for printing off-line when the job is concluded.

DUMP,<n>,<data element name 1>,...<data element name i>

This command is similar to PRINT, except that the record(s) is dumped on the system file DUMP in a format readable by the editor. This file may be used subsequently to load another database.

DISPLAY,<data element name 1>,...<data element name i>

This command sets the data elements to be listed, printed, or dumped by LIST, PRINT, or DUMP commands that are not followed by a list of data element names. This default is effective until another DISPLAY or SUPPRESS command is issued. The data element names in the list are separated by commas. The entire command must not contain embedded blanks.

SUPPRESS,<data element name 1>,...<data element name i>

This command has a function similar to the DISPLAY command. The difference is that a SUPPRESS command specifies which data elements are not to be listed, printed, or dumped by LIST, PRINT, or DUMP commands that are not followed by a list of data element names. This default is effective until another DISPLAY or SUPPRESS command is issued.

If no list of data element names follows a SUPPRESS command, the default is reset to all data elements.

MODIFY,<n>

This command causes the <n>'th record of the current set to be fetched from the database into core and then passes control to the editor. The record can be modified as desired using editor commands. When all editing is complete, control is returned to the executive, which restores the record in the database and updates all indices to reflect the changes in the record. Successful completion of this operation generates the message

RECORD MODIFIED

DELETE,<n>

This causes the <n>'th record of the current set to be removed from the database. All indices will be updated to reflect this action. Successful completion generates the message

RECORD DELETED

LINE,<n>

This sets the input/output line length to <n> characters. The default is 80 characters.

PRECISION,<n>

This sets the format used for the display of real values to fixed precision with <n> places after the decimal point.

SIGNIFICANCE,<n>

This sets the format used for the display of real values to <n> significant figures. The default is 5 sigrificant figures.

AUDIT

This command turns on the audit facility (if it has been turned off previously with a NOAUDIT command). While it is on, all input to the system from the INPUT file (the terminal when running interactively) is echoed on the AUDIT file. Besides providing a record of activity, this file may be used to update a backup copy of the database if a run is terminated abnormally (e.g. a file-preserving system crash).

NOAUDIT

This command turns off the audit facility. Normally, it should be the first command in a batch run, since in batch mode the INPUT file may be saved and used in lieu of an AUDIT file.

STOP

This signals the end of a run. The executive will perform necessary housekeeping functions and terminate.

3.3 EXECUTIVE ERROR MESSAGES

The error messages that may be generated by the executive are summarized in this section. The prefix ***ERROR***, which is common to all messages, has been omitted.

INVALID COMMAND

Either the command verb was unrecognized, or the command syntax was incorrect.

INVALID RECORD NUMBER

The record number specified was larger than the number of records in the current set.

INVALID SET NUMBER

The set number specified was larger than the number of existing sets.

RECORD NOT DELETED

It was not possible to delete the specified record.

RECORD NOT FULLY DE-INDEXED

If a record was being deleted, it was not possible to remove some or all of its index entries. If a record was being modified, it was not possible to remove some or all of the index entries for those key data elements whose values were changed. As a result, the record may satisfy some queries that it should not.

RECORD NOT FULLY INDEXED

If a new record was being added, it was not possible to make some or all of its index entries. If a record was being modified, it was not possible to make some or all of the new index entries for those key data elements whose values were changed. As a result, the record will not satisfy some queries that it should.

RECORD NOT RETRIEVED

It was not possible to retrieve a record specified in a LIST, PRINT, DUMP, MODIFY, or DELETE command.

RECORD NOT STORED

It was not possible to store a new or modified record.

WORK SPACE EXCEEDED

There was not sufficient work space to carry out some requested operation. Before it will be possible, the system must be recompiled with a larger work space.

4.1 INTRODUCTION

The editor is the subsystem of BDMS which is invoked by the executive to create a new record in core prior to its addition to a database (ADD executive command) or to modify an existing record which has been fetched into core from the database (MODIFY executive command). It is also invoked by the database definition compiler to create or modify the definition of a new database.

It has two operating modes- append mode, which is used to build or extend a record, and edit mode, which allows selective replacement or modification of data element values and insertion or deletion of data element occurrences. Normally, a new record will be created in append mode, but one might shift into edit mode to correct an error or omission before the record is stored. Likewise, most editing of an existing record probably will be done in edit mode, but append mode might be used to extend the record by simply adding data element occurrences.

The editor reads free format input comprising commands and data element names and assignment statements. The commands tell it what operations to perform on the record, thus setting the operating mode. Data element names may occur alone or as part of an assignment statement, depending on the operation specified by the last command. In edit mode they usually are qualified by one or more occurrence numbers which identify the particular data element occurrence to be altered. Assignment statements assign values to data element occurrences.

Section 4.2 of this chapter will describe in detail the form of assignment statements to prepare the reader for the discussion of editor commands and operating modes in Sections 4.3-4.5. Section 4.6 is a summary of the error messages that might be encountered while using the editor.

4.2 ASSIGNMENT STATEMENTS

When a value is to be assigned to a data element occurrence, it is done with an assignment statement whose general form is-

<data element name> = <value>;

DATA ELEMENT NAMES

A data element name is either the preferred name or one of the synonyms specified in the database definition. In edit mode, it may be qualified by one or more occurrence numbers -- integers separated from the name and each other by periods. Occurrence numbers will be discussed in Section 4.5.

NUMERIC VALUES

The value of an INTEGER type data element must be a series of digits preceded optionally by a + or - sign. The value may be expressed as a binary, octal, decimal, or hexadecimal number; the base is specified by a single letter following the number as follows-

B	Binary
O or Q	Octal
D or absent	Decimal
H or Z	Hexadecimal

The set of allowable digits is that subset of (0-9, A-F) appropriate for the base chosen.

The value of a REAL or DOUBLE type data element must be a series of decimal digits preceded optionally by a + or - sign. It may, in addition, have a decimal point and/or a power-of-ten multiplier expressed as an E followed by the integer exponent with an optional sign.

Blanks anywhere within a numeric value are ignored and the value is terminated by a semicolon. Numeric values must not exceed the maximum size which can be stored as a single precision number (or double precision number for DOUBLE type) by the computer being used.

A numeric vector value is represented as a series of component values constructed according to the above rules, separated by commas and terminated by a semicolon. For example,

X = 1.5,3.2,7.6;

CHARACTER STRING VALUES

The value of CHAR type data element is the character string between the relational operator and a terminating semicolon. Leading and trailing blanks are ignored but embedded blanks are considered to be part of the value. If leading and/or trailing blanks are desired, they may be forced through use of the symbol for logical negation, (a backward slash in ASCII character set). Each leading or trailing negation symbol is converted into a blank when the value is stored. If the character string contains semicolons, they must be doubled to avoid confusion with the terminating semicolon; any semicolon immediately followed by another semicolon will be stored as a single semicolon and will not terminate the string value.

NULL VALUES

A data element occurrence of any type may be assigned a null (i.e. undefined) value by an assignment statement of the form

<data element name> = ;

This is occasionally useful if it is necessary to enter a value for a subordinate data element but no value is known for its parent.

4.3 EDITOR COMMANDS

The editor is controlled by commands which either set its operating mode and determine how subsequent data element names and assignment statements will be interpreted, or cause some immediate action to be taken. All commands are single letters preceded by an asterisk and followed by a blank. They will be summarized here and discussed at length in the following two sections.

COMMANDS THAT SET OPERATING MODE

Append mode

*A Append data element occurrence(s). This is the default mode.

Edit mode

*R Replace data element value(s).

*S Substitute string(s) in data element value(s).

*D Delete data element occurrence(s).

*I Insert data element occurrence(s).

COMMANDS THAT CAUSE IMMEDIATE ACTION

** Exit from editor - record will be stored or replaced by edited version.

*C Cancel edit - do not store new or altered record.

This command simply terminates the editor and returns control to the executive. If a new record was being input, it will not be stored in the database. If an existing record was being edited, no changes made will be reflected in the database. It is primarily useful in an interactive session to terminate a hopelessly confounded edit or one which was mistakenly begun.

*L List the present state of the record.

This command allows a record to be visually checked anytime while it is being created or edited. It does not affect the operating mode or alter the edit command in effect when it is given.

*E End of record.

This marks the end of a record when multiple records are being

ADDed. It is usually not needed (c.f. section 4.4, subsection on automatic record generation). When MODIFYing a record, it has the same effect as **.

4.4 APPEND MODE

When first entered, the editor is in append mode. It may be returned to append mode at any time with a *A command.

ORDER OF ASSIGNMENT STATEMENTS

In append mode the order in which assignment statements and pure node data element names are entered determines the order of the data element occurrences created and their linkage to parent occurrences. The occurrences of a data element linked to each occurrence of its parent data element (or the record level data element) form an ordered list. In general, each assignment statement or pure node name creates a new occurrence of a data element which will be added to the end of the list of occurrences that is linked to the last-created occurrence of its parent data element.

For example, if data were being added to the database described in Chapter 1,

```
AUTHORS.;
A.=JONES;
A.=SMITH;
I.=LBL;
AUTHORS.;
A.=BAKER;
I.=UCB;
```

would create two AUTHORS groups, the first with two authors at LBL and the second whose single author is affiliated with UCB.

The periods following the data element names mean "next". A data element name without a period means "first" or occurrence number 1. The next two subsections will elucidate the significance of these conventions.

Other than having data element occurrences in the desired order and following the correct parent data element occurrence, the order of assignment statements in append mode is immaterial. An assignment statement for the reference, R, could have been inserted anywhere in the preceding example without affecting the creation of the AUTHORS groups.

AUTOMATIC PARENT NODE GENERATION

In append mode, NODE type parent data element names can usually be omitted from the input stream. The occurrences of such data elements necessary to link together subordinate data element occurrences will be generated automatically if

1) one of the subordinate data elements is encountered in the input stream and no parent occurrence yet exists, or

2) occurrence number 1 of one of the subordinate data elements is encountered, i.e. the data element name appears without a period following it, and the last-created parent occurrence already has at least one occurrence of that data element linked to it.

Thus, the preceding example could have been entered in the simpler form

```
A=JONES;
A.=SMITH;
I=LBL;
A=BAKER;
I=UCB;
```

or equivalently,

```
I=LBL;
A=JONES;
A.=SMITH;
I=UCB;
A=BAKER;
```

Each occurrence of A in the first case or I in the second causes the creation of an AUTHORS node.

Automatic parent node generation will also work if the parent data element is not NODE type. In this case, an automatically generated parent occurrence will have a null value.

AUTOMATIC RECORD GENERATION

When ADDing records, if a record-level data element name appears more than once without a following period, the second occurrence will trigger the storage of the previous record and begin a new one just as if a *E command had been encountered. The second occurrence of the record-level data element will become a part of the new record.

The record itself may be considered to be the pure node parent of all record-level data elements, so automatic record generation is completely analogous to the automatic parent node generation described in the preceding subsection.

If the editor has been invoked by a MODIFY executive command, automatic record generation is meaningless, so that in append mode, a recurrence of a record-level data element name without a period will be flagged as an input error.

MULTIPLE ASSIGNMENT STATEMENTS

Several consecutive occurrences of a data element may be created without the necessity of repeating the data element name. This is done with a multiple assignment statement, whose most general form is

<DE name> = <value 1>;<value 2>;...

Using a multiple assignment statement, the preceding example could have been input in the still simpler form

```
A=JONES; SMITH; I=LBL;
A=BAKER; I=UCB;
```

or equivalently,

```
I=LBL; A=JONES; SMITH;
I=UCB; A=BAKER;
```

The difference between the two assignment statements

```
X=1,2,3;
```

and

```
X=1;2;3;
```

should be clearly understood. The first creates a single occurrence of X whose value is a three-component vector (assuming X is defined to be a numeric type data element). The second creates three occurrences of X whose values are 1, 2, and 3, respectively.

In order to implement multiple assignment statements, the input processor will attempt to interpret anything which is not a recognizable data element name or assignment statement as another value in the preceding assignment statement. If a data element name is misspelled, the erroneous assignment statement cannot be distinguished from another value for a preceding CHAR type data element. It will only be recognized as an error if the preceding data element is not CHAR type.

4.5 EDIT MODE

Any of the commands *R, *S, *D, or *I places the editor in edit mode.

NAMING DATA ELEMENT OCCURRENCES

In edit mode, the data element occurrence to be altered is identified by specifying its position within the record structure. To completely identify a particular data element occurrence, one must specify its name and occurrence number, the number of the parent occurrence to which it is linked, the number of the grandparent occurrence to which its parent is linked, etc. up to the record-level ancestor. The series of occurrence numbers is appended to the data element name in this order, i.e., in order of increasing remoteness of the ancestor, as a series of integers separated from the name and each other by periods.

Referring to the example record structure of Chapter 1, A.4.2 would identify the fourth author A.4 of the second authors group, AUTHORS.2. The (first and only) X value in the second DATUM in the first TABLE would be identified as X.1.2.1, etc.

Under certain circumstances it is not necessary to specify all these occurrence numbers since the editor will supply default values. This will be discussed below in the subsection on path memory.

REPLACE COMMAND (*R)

This is followed by assignment statements which assign new values to existing data element occurrences. For example,

```
*R A.2.1=J.DOE;
```

would cause the value of A.2.1 to be replaced by J.DOE.

It is not meaningful to replace a pure NODE type data element since it carries no value.

SUBSTITUTE COMMAND (*S)

This allows the replacement of a selected substring within a CHAR type data element value. It works like the *R command except that the data element values within assignment statements which follow a *S command have the form

<delimiter><old substring><delimiter><new substring><delimiter>.

The <delimiter> may be any character that does not occur in either the <old substring> or the <new substring>. All blanks within the substrings are significant.

The <new substring> will replace the first substring in the data element value that matches the <old substring>. If the <old substring> is null, the <new substring> will be inserted at the beginning of data element value. If the <new substring> is null, the first substring in the data element value that matches the <old substring> will be deleted.

For example, if A.2.1 has the value J.DEC, the command

*S A.2.1=/EC/OE/;

would change it to J.IOE. The command

*S A.2.1=//B/;

would change that value to BJ.DOE, while the command

*S A.2.1=/J//;

would leave as a final value B.DOE.

The modified data element value is echoed following execution of a *S command so that the correctness of the substitution may be ascertained.

DELETE COMMAND (*D)

This is followed by the names of the data element occurrences to be deleted, each terminated by a semicolon. All occurrences of subordinate data elements linked to a deleted data element occurrence are also deleted.

For example,

*D DATUM.2.1;

would delete DATUM.2 in TABLE.1 and any occurrences of X, Y, DY, and CD linked to it.

After a data element occurrence has been deleted, any occurrences of that data element linked to the same parent occurrence and following the deleted occurrence will be identified by occurrence numbers one smaller than originally.

INSERT COMMAND (*I)

This is followed by assignment statements (or occurrence names for pure node type data elements). Each of them will cause an occurrence of a data element to be inserted before the named occurrence.

For example, if the author name which should have been A.2 in AUTHORS.1 had been mistakenly omitted from a record, it could be inserted with the command

*I A.2.1=J.DOE;

After a data element occurrence has been inserted, any occurrences of that data element linked to the same parent occurrence and following the inserted occurrence will be identified by occurrence numbers one larger than originally. Hence the inserted occurrence becomes the named occurrence.

In order to add a data element occurrence after all existing occurrences linked to a given parent occurrence, one either specifies an occurrence number greater than the number of occurrences already existing, or specifies occurrence number 0, or simply omits the last qualifier, but not its preceding period. Hence, the command in the preceding example would have the desired effect even if AUTHORS.1 had only one subordinate A occurrence prior to the insertion.

A multiple assignment statement following an insert command causes several new occurrences to be inserted before the specified occurrence of the data element. They will occur in the modified record in the order of their appearance in the assignment statement. After insertion, any occurrences of that data element linked to the same parent occurrence and following the inserted occurrences will have their occurrence numbers incremented by the number of inserted occurrences.

For example,

*I A.4.2 = J.SMITH; B.JONES;

would leave AUTHORS.2 with A.4=J.SMITH; and A.5=B.JONES;. If an A.4 existed prior to insertion, it would now be A.6, etc.

ORDER OF EDITS

One point made in the preceding discussion warrants further emphasis - data element occurrence numbers are relative list positions and can change during editing. After data element occurrences are inserted or deleted, any occurrences following the insertion or deletion point (linked to the same parent occurrence) must be identified by occurrence numbers larger or smaller than originally.

In order to avoid confusion, especially when working from a printed listing or doing batch updates, replacements and substitutions should be done first, and insertions and deletions should be done from the bottom up, i.e. those data element occurrences with the highest occurrence numbers should be edited first. Then successively modified occurrences will retain the occurrence numbers of the original record.

If one does become confused while working interactively, the current values of the occurrence numbers may be ascertained by listing the record with a *L command.

PATH MEMORY

In edit mode, the editor will attempt to supply a default parent hierarchy for a data element being modified if the user has not given complete position information.

This is implemented with a path memory, which works as follows. The editor remembers the path taken through the record structure to arrive at the position of a modification. Then, if the position of the next modification is not completely specified, an attempt will be made to link whatever partial path is specified to the path remembered from the last modification. If this linkage can be achieved, the resulting path will be used as the position of the new modification and the path memory updated.

The path memory is cleared when the editor enters append mode.

The best way to understand the effect of this general mechanism is through the consideration of some specific examples.

1. Suppose one wishes to replace the values of several A's within AUTHORS.2. This can be accomplished with the command

```
*R A.1.2=E.JONES;
   A.2=E.SMITH;
   A.4=C.BAKER;
```

   The occurrence of the parent data element AUTHORS is not specified for A.2 and A.4 and so it is assumed to be AUTHORS.2, as specified for A.1.

2. Suppose that the entire second AUTHORS group was inadvertently skipped when a record was initially input. It could be inserted with the command

```
*I AUTHORS.2;
       A=E.JONES; E.SMITH; J.DOE; C.BAKER;
       I=LBL;
```

   Since no parent occurrence is specified for A.1, A.2, A.3, A.4, and I.1, it is picked up from the path memory as

AUTHORS.2.

3. Suppose that a new third datum is to be added to TABLE.2. It can be inserted with the command

```
*I DATUM.3.2;
      X=1.537;
      Y=3.206; DY=0.001;
      CD=NEW DATA POINTS;
```

Since no parent occurrence is specified for X.1, Y.1, and CD.1, it is picked up from the path memory as DATUM.3 in TABLE.2. For DY.1, it is assumed to be Y.1 in DATUM.3 in TABLE.2.

4. Suppose that after the edit in 3., one desires to replace the value of X.1 in DATUM.2 in TABLE.2. The command

```
*R X.1.2=2.372;
```

will accomplish this. It is not necessary to give a complete position specification for X, namely X.1.2.2, because the default occurrence of TABLE resulting from the previous edit is TABLE.2.

4.6 EDITOR ERROR MESSAGES

In this section, the editor error messages will be summarized and their meanings elaborated. It must be clearly recognized, however, that the editor is designed to make every effort to find a legal interpretation for all input it processes. The implications of this were discussed in section 4.4. (subsection on multiple assignment statements).

In gereral, when an error is detected, that part of the input stream which caused the error will be skipped and processing will continue with the following input. In append mode, this means that some data element occurrence will not be created. In edit mode, some edit operation will not be performed. If one is working interactively, the error usually can be corrected immediately. In batch mode, one error will sometimes cause several others, e.g. a parent data element occurrence is not created due to a misspelling, causing several subordinate data element occurrences to be linked incorrectly to a previously created parent occurrence.

All error messages issued by the editor are preceded by ***ERROR***. This prefix has been omitted in the following list of messages.

DATA ELEMENT VALUE IN DELETE COMMAND

It is not mearingful to specify a new value for a data element occurrence being deleted. Only data element names terminated by semicolons may follow a delete command.

ILLEGAL RECURRENCE OF RECORD-LEVEL DATA ELEMENT

A record-level data element name without a period occurred in append mode while MODIFYing a record which already has an occurrence of that data element. While ADDing records, this is not an error but will cause the automatic generation of a new record.

INCOMPLETE ASSIGNMENT STATEMENT

In batch mode, an end-of-file condition has been sensed while processing a data element name or value. Probably the terminating semicolon was omitted.

INCORRECT QUALIFIER

The editor could not determine the path to a data element occurrence based on the occurrence numbers given and the current state of the path memory. This message is also given if too many occurrence numbers are specified for the depth of a data element within the hierarchical record structure.

INVALID LENGTH RETURNED BY IPROC

The user-supplied IPROC routine has returned a negative data element length.

INVALID NUMERIC DATA ELEMENT VALUE

The value specified for a numeric data element does not have a form valid for its type.

MISSING DATA ELEMENT VALUE

A non-NODE type data element name is followed immediately by a semicolon. This is only allowed in a delete comand.

MULTIPLE ASSIGNMENT STATEMENT IN REPLACE OR SUBSTITUTE COMMAND

Only one data element occurrence at a time may be replaced. Multiple assignment statements are not meaningful following a replace or substitute command.

NO MATCHING SUBSTRING

The old substring specified in a substitute command does not occur in the string being scanred.

NON-CHAR DATA ELEMENT IN SUBSTITUTE COMMAND

A substitute command may only operate on CHAR type data elements.

PARENT DATA ELEMENT MISSING

An edit operation cannot be performed because an indicated parent data element occurrence does not exist.

PURE NODE DATA ELEMENT IN REPLACE OR SUBSTITUTE COMMAND

Since a NODE type data element carries no value, it is mearirgless to replace it. Only assignment statements may follow a replace or substitute command.

QUALIFIED DATA ELEMENT NAME IN APPEND MODE

Data element names may not be followed by occurrence numbers in append mode.

SPECIFIED OCCURRENCE DOES NOT EXIST

The data element occurrence specified in an edit command does not exist. An odcurrence number probably was input incorrectly.

SUBSTITUTE COMMAND SYNTAX

The syntax of a substitute command is incorrect, e.g. the delimiter character does not occur exactly three times or there are non-blank characters between the third delimiter and the terminating semicolon.

UNDEFINED DATA ELEMENT NAME

The input stream contains a data element name which could not be

recognized and could not be interpreted as another value in a preceding assignment statement.

WORK SPACE EXCEEDED

A data element name or value is longer than the available work space or there is no more space available for expansion of the path memory. In general, the system must be recompiled with a longer work space to guarantee that the error will not recur when accessing this database. Sometimes, storing the record and then retrieving it again will free enough work space to alleviate the problem and allow further editing. This is likely if extensive edits have resulted in a large amount of dead space in the record.

4.7 LIST ERROR MESSAGES

Certain error corditions that generate messages can arise while listing a record. They are summarized in this section.

All error messages issued by the list routine are preceded by ***ERROR***. This prefix has been omitted in the following list of messages.

INVALID LENGTH RETURNED BY OPROC -- DATA ELEMENT SKIPPED

The user-supplied OPROC routine has returned a negative data element length.

INVALID MODE PARAMETER -- LIST ABORTED

This will only occur when the user calls LIST directly with an invalid mode parameter.

RECORD BUFFER EMPTY -- LIST ABORTED

There is no record in core to be listed, not even an empty one. This will only occur if a user calls LIST directly without properly initializing the record buffer.

WORK SPACE EXCEEDED -- DATA ELEMENT SKIPPED

WORK SPACE EXCEEDED -- LIST ABORTED

The record is so large that there is insufficient work space for LIST to operate. The system should probably be recompiled with a larger work space for use with this database.

5.1 INTRODUCTION

The BOMS query language permits a user to search a database for those records satisfying an arbitrarily complex condition on key (indexed) data element values. The condition is constructed as a Boolean combination of key value specifications, including inequalities and ranges. Furthermore, it is possible to search for records having an occurrence of a specified data element regardless of value, or for those having an occurrence of the data element with a null value. Truncated value specification for character string keys may be used to search for those records having an occurrence of the data element beginning in a particular way.

The retrieval facility is invoked by the FIND executive command. This must be followed by a condition that is terminated by a **, i.e.

FIND <condition> **

The set of records that satisfy the condition is assigned a set identification number, and the number of records in the set is reported in the form.

<n> RECORD(S) IN SET <s>

If no records satisfy the condition, the response is

0 RECORD(S) IN SET

The number <s> assigned to a non-empty set may be used in subsequent queries in combination with further conditions.

The next two sections describe the format of conditions.

5.2 SIMPLE CONDITIONS

A simple condition has the form -

```
<SIMPLE CONDITION>::=    {{= }            [  {{=]}           ]}
                         {{> } <VALUE1>; [TO{ < } <VALUE2>;]}
                         {{>=}           [  {<= }           ]}
                         {                                  }
<KEY DATA ELEMENT NAME> {{< }                               }
                         {{<=} <VALUE>;                     }
                         {{<>}                              }
                         {                                  }
                         { ;                                }
```

where curly brackets {} surround a set of options, one of which must be chosen, and square brackets [] surround a completely optional element. The relational operators <=, >=, <> stand for "less than or equal", "greater than or equal", and "not equal", respectively. Thus, an exact value, an inclusive or exclusive upper or lower bound, or a range of values may be specified.

Examples of valid simple conditions are -

A = JONES;

A <> SMITH;

X <= 7;

X = 5; to 8;

X > 5; TO < 8;

The later two examples differ in that the second excludes both endpoints of the range.

The condition

<key data element name> <> <VALUE>;

will be satisfied by all records which

a) have at least one occurrence of the named data element, and

b) have no occurrence of that data element whose value matches that given.

If no value appears after the relational operators = or <>, a search is made for those records which respectively do or do not have an occurrence of that data element with a null value. Note that this corresponds to the syntax used to enter a null value using the editor (cf. Chapter 4). For example,

A =;

would result in a search for all records having a null value for A and

A <>;

defines the complement set.

The last form allowed for a simple condition, e.g.

<key data element name>;

defines the set of all records having any occurrence of the specified data element, regardless of value.

The format of a value depends on the data element type and is the same as that accepted by the editor (cf. Chapter 4). Key values in an index are fixed length, derived from the corresponding data element values by truncation or padding; data element values in a query will be truncated or padded in the same way to ensure valid comparison with index values.

5.3 COMPLEX CONDITIONS

The most general condition that may appear in a query is constructed out of simple conditions and previously defined sets according to the following recursive definition -

```
<CONDITION>=[NOT] {<SIMPLE CONDITION>} [ {AND} <CONDITION>]
                  {<SET NUMBER>      } [ {OR }             ]
```

That is, simple conditions and previously-defined sets, identified by number, may be combined using the Boolean operators NOT, AND, and OR. NOT has the highest precedence and OR the lowest; this ordering may be overridden through use of parentheses.

Note that simple conditions and set numbers play equivalent roles in a complex query. This is because each simple condition may be viewed as defining an (intermediate) set. These sets, along with any existing sets appearing in the query, are then combined by union (OR), intersection (AND), and complement (NOT).

It should be noted that the two conditions

<key data element name> <> <VALUE>;

and

NOT <key data element name> = <VALUE>;

are not equivalent. The meaning of the first was elucidated in the preceding section. The second differs in that records having no occurrence of the named data element will also satisfy it. This happens because NOT complements the set defined by a following condition and the universal set in terms of which complementation is defined is the entire database. Thus, the effect of a NOT operator cannot simply be absorbed into the relational operator and

NOT	=		<>
NOT	<		>=
NOT	>	are not equivalent to	<=
NOT	<=		>
NOT	>=		<
NOT	<>		=

5.4 RECORD ID

As mentioned in Chapter 1, every record in a database is assigned a unique and permanent integer record ID when it is first created. The pseudo data element name REC-ID may be used in a simple condition to retrieve records by their ID's. Any form of the simple condition including ranges and

```
REC-ID;
```

is allowed. The set of records satisfying the latter "condition" is the entire database.

Simple conditions on REC-ID may be used as components of a complex condition.

5.5 TRUNCATED SEARCH

Truncation provides a way of searching for records that contain occurrences of some data element whose value begins in a specified way, regardless of the rest of the value. For example, one might like to find all papers authored by Smith, regardless of how the first name had been entered in the database. If authors' names were stored last name first, one could just do a truncated search for Smith. Truncation is indicated in the BDMS query language by a single / (slash) following the partial value. Thus, the preceding example would be expressed as

```
FIND A=SMITH/; **
```

Any record with a value for A whose first five characters are SMITH would satisfy the condition.

If one desires to search for a value which happens to end with a /, the slash may be doubled to prevent its being interpreted as a truncation delimiter. For example,

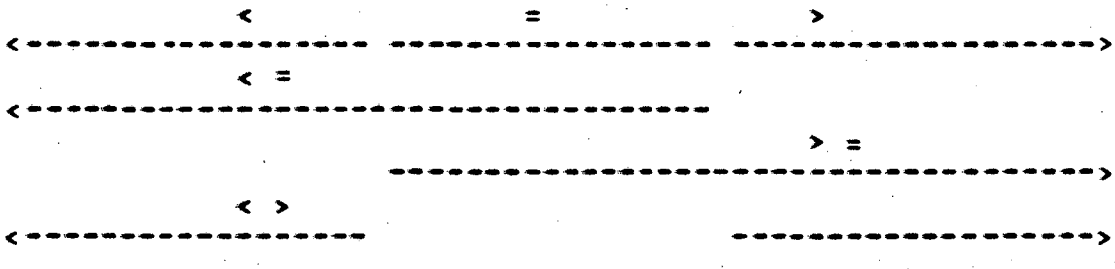
```
FIND CT=ABC//; **
```

would find all records having a value ABC/ for CT. If one desires to truncate immediately following a /, then three slashes are required. For example,

```
FIND CT=ABC///; **
```

would find all records containing values of CT which begin with the characters ABC/.

Truncated values may be used in conjunction with any relational operator. Either or both of the endpoints of a range search may be expressed as truncated values in the same way. Since equality to a truncated value actually defines a range of values that will satisfy the condition, the meanings of the remaining relational operators will be affected in a corresponding way, elucidated by the following diagram -



The range of record values satisfying an equality condition is represented by the middle segment of the top line while the ranges satisfying other corditions involving the same value but different relational operators are represented by the other line segments.

Simple conditions involving truncated values may be used as components of a complex condition.

5.6 RETRIEVAL ERROR MESSAGES

The error messages that can be generated while processing a query are listed below. The entire query is scanned for errors and only if none are detected is the database actually searched.

All error messages from the query processor are preceded by ***ERROR***. This prefix has been omitted in the following list of messages.

INCOMPLETE QUERY

In batch mode, an end of file condition has been sensed while processing a query. Probably the query terminator (**) was omitted.

INVALID LENGTH RETURNED BY QPROC

The user-supplied QPROC routine has returned a negative data element length.

INVALID NUMERIC DATA ELEMENT VALUE

The value specified for a numeric data element does not have a form valid for its type.

INVALID RANGE SPECIFICATION

A meaningless set of relational operators has been used in a range condition, e.g. X < 7; TO > 9;.

INVALID SET NUMBER

A set number has been used that is less than or equal to 0, or larger than the last-created set number.

NON-KEY DATA ELEMENT IN QUERY

A data element used in the query is not defined to be a KEY.

QUERY SYNTAX

The query has not been properly constructed out of simple conditions and Boolean operators, e.g. two or more simple conditions are not joined by a Boolean operator.

TOO MANY LEFT PARENTHESES

TOO MANY RIGHT PARENTHESES

The parentheses appearing in the query are not properly paired.

UNDEFINED DATA ELEMENT NAME

A data element name appearing in the query cannot be recognized. It has probably been misspelled.

VALUE SPECIFIED FOR PURE NODE DATA ELEMENT

The query includes a condition on the value of a NODE data element. This is meaningless since a pure NODE carries no value. Perhaps the data element name is misspelled.

WORK SPACE EXCEEDED

The system has run out of work space while processing the query. It may be necessary to recompile the system with a larger work space for use with this database.

6.1 INTRODUCTION

Utility programs are provided with BDMS to perform the maintenance functions of initial database loading, dumping an entire database in external format for transmittal, compressing dead space from the data file, and rebalancing the index trees. Use of these programs is explained in the following sections.

6.2 LOAD

The LOAD utility is used for initial batch loading of a database. At present, it cannot be used to add records to an existing database - that must be done with the executive ADD command. LOADing is a much more efficient operation than ADDing because the index entries are saved until all records have been stored in the data file, and then the index trees are built from the bottom up in a single pass. Thus, the index trees are perfectly balanced, leading to maximum efficiency in query processing.

Input to the LOAD utility consists of one or more commands followed by the data to be loaded in external editor format, with records separated by *E and the last record terminated by **. This is the format produced by the DUMP utility or the executive DUMP command. The commands that may precede the input data are -

LINE,<n>

This sets the input line length to <n> characters. If no LINE command is present, the line length defaults to 80 characters.

FREE,<n>

This sets the amount of free space to be left on each index page to allow future database expansion without the possible consequence of unbalancing the index trees. If <n> <> 0, <n> percent free space will be left on each page. If <n> = 0, the pages will be completely filled to achieve minimum index size for a static database.

LOAD

This terminates the command stream and initiates the data stream. A LOAD command must precede the data even if no other commands are present in the input stream.

These commands are terminated by a blank and hence must not contain embedded blanks.

6.3 DUMP

The DUMP utility is used to dump an entire database in a format suitable for subsequent reloading. This is primarily useful if it is necessary to transmit the database to another site. To achieve maximum efficiency, the records are output in the (random) order in which they occur in the data file, rather than in record ID or key sequence. If it is desired to dump only selected parts of a database, this may be done with the executive FIND and DUMP commands.

The DUMP utility reads from the input file commands specifying the format of the dump file. They are -

LINE,<n>

This sets the output line length to <n> characters. If no LINE command is present, the line length defaults to 80 characters.

EXPAND

This selects an expanded dump format in which each data element occurrence begins a new line. This may be useful if a text editor is to be used to modify the dump file prior to reloading the database. If no expand command is present, a more compact and efficiently readable format is used in which the output lines are completely filled.

These commands are terminated by a blank and hence must not contain embedded blanks.

6.4 CLEAN

The CLEAN utility is used to compress out of the data file the dead space resulting from update activity. The frequency with which this operation should be carried out depends on the nature and frequency of update activity for the database.

6.5 BALANCE

The BALANCE utility is used to rebalance the index trees. This operation may be necessary after extensive update activity in order to optimize query processing.

The input file optionally may contain the following command.

FREE,<n>

This sets the amount of free space to be left on each index page to allow future database expansion without the possible consequence of unbalancing the index trees. If <n> <> 0, <n> percent free space will be left on each page. If <n> = 0, the pages will be completely filled to achieve minimum index size for a static database.

?????????.?????????.?????????.?????????.?????????.?????????.?????????.?????????
?????????.?????????.?????????.?????????.?????????.?????????.?????????.?????????