MDBS DATA BASE DESIGN REFERENCE MANUAL

The MDBS DDL MANUAL

Version 3.08

Micro Data Base Systems, Inc.

P. O. Box 248

Lafayette, Indiana 47902

USA

Telex: 209147 ISE UR

(312) 303-6300 (in Illinois)

December 1985

Rev. 85A

3250321

NEW RELEASES, VERSIONS, AND A WARNING

Any programming endeavor of the magnitude of the MDBS software will necessarily continue to evolve over time. Realizing this, Micro Data Base Systems, Inc., vows to provide its users with updates to **this version** for a nominal handling fee.

New versions of MDBS software will be considered as separate products. However, bona fide owners of previous versions are generally entitled to a preferential rate structure.

Finally, each copy of our software is personalized to identify the licensee. There are several levels of this personalization, some of which involve encryption methods guaranteed to be combinatorially difficult to decipher. Our products have been produced with a very substantial investment of capital and labor, to say nothing of the years of prior involvement in the data base management area by our principals. Accordingly, we are seriously concerned about any unauthorized copying of our products and will take any and all available legal action against illegal copying or distribution of our products.

PREFACE

By the mid-1960s, application developers were well aware of the data handling limitations of programming languages and file management systems. To overcome these limitations, data base management systems began to appear. By 1971, four major approaches to data base management had taken shape: the hierarchical, shallow-network, relational, and CODASYL-network approaches to logical data structuring and manipulation. Each represented an advance over the old file-oriented data handling methods and the latter two approaches offered advances over the former two. By the mid-1970s, data base management software was well established as the cornerstone for application development on mainframes and some mini computers.

Near the end of the decade, microcomputers -- with their phenomenal computing power on a per dollar basis -- began to proliferate. The acceptance of mainframe data base management systems coupled with the rise of microcomputers led to the formation of Micro Data Base Systems Incorporated by a group with expertise in both areas. The initial objective was to make genuine data base management tools available in the micro realm. This objective was fulfilled in 1979 with the release of MDBS I -- the first authentic and viable data base management system (dbms) for microcomputers. Over the years, this has evolved into the present MDBS III which operates not only on many microcomputers but on larger machines as well. The evolution of MDBS III is highlighted with many firsts: first micro dbms with built-in logging and recovery, first full implementation of a postrelational dbms, first multiuser micro dbms, first dbms to run under PCDOS, MSDOS and UNIX.

Today, MDBS III offers professional application developers a degree of power and flexibility unavailable with any other data base management software -- be it on micros, minis or mainframes. This is partially due to the highly efficient, proprietary implementation techniques. MDBS III is not a mainframe retread shoehorned into a microcomputer. It is also due to the innovative data modeling features that MDBS III provides. Because these features go far beyond those of the older data base management approaches, MDBS III is variously referred to as postrelational or multiarchical or extended-network. The emphasis in this approach to data base management is on direct, natural representation of the application world. The result is a tremendous increase in developer productivity. As stated in the authoritative Database - the 2nd Generation: State of the Art Report (ed. D.J.L. Gradwell, Pergamon Press, Oxford, England, 1982):

> **"The data modelling capability of MDBS III is superior to any other commercially available DBMS." MDBS III is "... a product that is, in many ways, ahead of mainframe DBMSs."** (D.J.L. Gradwell)

All of this translates into convenience and productivity for application system developers and administrators.

The MDBS R&D Lab's expertise in the areas of decision support systems and artificial intelligence has resulted in two unique environments for processing MDBS III data bases. One is a decision support environment called KnowledgeMan. It functions as a universal knowledge management system, allowing users to represent and process knowledge in many different ways -- including spreadsheets, text, graphics, forms, procedural models, relational data bases, and postrelational MDBS III data bases. The second is a revolutionary artificial intelligence environment called Guru. It makes the benefits of both expert system technology and natural language processing easily accessible to business users, without sacrificing familiar business computing capabilities.

This manual provides an introductory overview of application development with MDBS III. It then concentrates in detail on the features and utilization of the MDBS III Data Description Language. Companion manuals discuss the MDBS III Data Manipulation Language and various optional modules. This is a reference manual. It is therefore not intended to be a tutorial. For a tutorial treatment of data base management (explaining its value, characteristics, and advantages), the reader is advised to consult such suitable references as:

1. Micro Database Management - Practical Techniques for Application Development by R.H. Bonczek, et. al., 536 pages, Academic Press, New York, 1984.
2. "A Perspective on Data Models," PC Tech Journal, Vol. 2, No. 1, 1984.
3. "Micros Get Mainframe Data Scheme," Systems and Software, Vol. 3, No. 5, 1984.
4. "Uniting Relational and Postrelational Database Management Tools", Systems and Software, Vol. 3, No. 11, 1984.

The first reference provides the most definitive coverage to date of the postrelational approach, comparing and contrasting it with the four older data base management approaches. That book also includes extensive examples of MDBS III usage. It is available through your local bookstore, from the publisher, or from MDBS, Inc. For many years, the Company has offered practical MDBS III training seminars in major cities and at customer sites.

The MDBS, Inc. commitment to customer success does not stop with software innovation, quality and support. The Company offers a full range of consulting and application development services to clients through regional offices in the Dallas, Chicago and New York offices. Services include:

1. Customized application systems design

2. Customized application systems development using Guru, KnowledgeMan and/or the MDBS III tools

3. Communications interfaces including mainframe-micro links

4. Special training on a wide variety of topics including specific application systems, C programming, MDBS III usage, and Guru usage

5. General consulting including feasibility studies and hardware/software recommendations

The experienced professional staff handles consulting and application development needs of some of the world's largest corporations and governments. It has developed very extensive micro application systems in such diverse areas as cash management, strategic planning, human resources administration, waste disposal management, and distributed service support. In many cases, these application systems have involved multiuser processing or mainframe-micro links.

MDBS and KnowledgeMan are registered trademarks of Micro Data Base Systems, Incorporated. Guru, Screen Maker and Menu Builder are trademarks of Micro Data Base Systems, Inc.

Table of Contents

Page

FIGURES

TABLES

APPENDICES

I. OVERVIEW

A. Manual Organization

This **MDBS DDL Manual** is the first component in the set of MDBS III Reference Manuals. It describes the data structuring facilities available for designing MDBS data bases and the Data Description Language (DDL) for formally specifying data base designs. The second component in the manual set is the **MDBS DMS Manual**, which describes data manipulation commands available to application programmers. Beyond these two basics, other manuals in the set describe various MDBS software options.

The MDBS III Reference Manuals are not oriented toward any specific hardware, operating system, or host programming language. They discuss those aspects of MDBS usage that are unaffected by the host hardware, operating system, and programming language. The variations in MDBS usage caused by differences in hardware, operating systems and host languages are detailed in the various **MDBS System Specific Manuals.** These variations are relatively minor.

MDBS III is available in two forms: Version 3a and Version 3c. Version 3a is implemented with assembly language. Version 3c is implemented with the C language. To an application developer, Versions 3a and 3c are practically indistinguishable. With a few noted exceptions, documentation in the MDBS Reference Manuals is equally applicable to both implementations.

The MDBS III data structuring features presented in this manual can be partitioned into two levels: fundamental and advanced. Advanced features are denoted by a heavy vertical bar in the outside margin. The reader who is new to the data base field is strongly advised to initially concentrate on the fundamental features, ignoring those portions of the documentation that are marked with a vertical bar. An understanding of the fundamentals is sufficient for designing many application systems using MDBS.

When the fundamentals have been mastered, the advanced features should be examined (on a second pass through the manual). The advanced features include facilities for governing the mapping of records into storage, for segmenting a data base into physically distinct areas, for providing data security, for feasibility range checking, for performance tuning, etc. For the application developer who does not use the advanced features, there is a standard default provided for each such feature when the data base is initialized. In using the advanced features, an application developer has greater flexibility and control over the characteristics of the resultant application system.

Chapter I of this manual provides an introduction to the family of MDBS III application development tools. It concludes with a summary of the DDL features and associated software. The MDBS constructs available for building an MDBS schema are rigorously defined in Chapter II. That chapter also contains an example schema.

Chapter III documents the Data Description Language syntax. DDL source specifications for the Chapter II schema are included in Chapter III. Data security specifications are treated in Chapter IV and automatic data integrity facilities are presented in Chapter V. Thus Chapters II-V deal with utilizing the MDBS Data Description Language. Chapter VI describes the DDL Analyzer software. Called MDBS.DDL, this software can be used for text entry to create a DDL source specification, text line editing, and DDL analysis to initialize a data base. Chapter VII discusses the diagnostics that can arise during DDL analysis.

B. Introduction to MDBS III

MDBS III is an extremely powerful data base management system (dbms) for serious application development. Specially designed to maximize the productivity of professional application developers. MDBS III addresses all the major issues facing OEMs, VARs, and information services groups: providing quality application system software, minimizing the development time and cost, facilitating the effective administration of application systems, promoting data sharing among customized application systems and generic decision support software, and porting application software into multiple operating environments.

Because of MDBS III, truly extensive application systems that were once possible only on large computers are now in everyday use on microcomputers. Application systems built with MDBS III span virtually all industries and functional areas. They include the most ambitious application systems ever developed for microcomputers -- systems which would have been technically or economically infeasible without MDBS III. The development times and costs for these sophisticated micro-based application systems are substantially less than for larger machines. MDBS run-time tokens allow these application systems to be inexpensively distributed.

One reason for these savings is that MDBS III allows development to proceed with inexpensive, dedicated microcomputers. But even more important is MDBS's unique postrelational approach to data modeling, which has yet to be equaled -- even by the most advanced mainframe dbms. The innate naturalness, rich flexibility, and expressive power furnished by this innovative data model combine to yield significant increases in developer productivity by eliminating many of the limitations inherent in conventional hierarchical, network and relational dbms approaches.

The MDBS III facilities for enforcing data security, ensuring data integrity, automatic program generation, concurrency control and performance tuning are all on a par with the best that is available in the mainframe world. These are important not only for productive application development, but also for on-going effective administration. Operating costs of large-scale application systems built with MDBS III are normally substantially less than mainframe or time sharing alternatives.

MDBS III data bases are implemented with a proprietary technique that results in very rapid response times. It is much more efficient than the old chaining and redundancy techniques that are so commonplace among leading mainframe data base management systems. Coupled with performance advantages implicit in the postrelational model and with the built-in performance tuning mechanisms, this proprietary technique yields very pleasing performance -- even for quite sizable data bases being processed on microcomputers.

Subject to its highly refined integrity and security controls, MDBS III promotes data sharing in both local area network and multiuser operating system settings. The same data base can simultaneously be accessed by a variety of application programs whose host languages may vary from COBOL to C. It can be directly accessed by expert systems using the Guru inference engine. It can also be directly accessed through end user facilities such as the nonprocedural query interface and the popular KnowledgeMan system -- a comprehensive and integrated decision support environment for graphics, text processing, spreadsheet analysis, modeling, customized report generation, etc.

MDBS III operates on a wide variety of hardware including the IBM PC series and compatibles, the Intel 286/310, 68000-based machines, the AT&T 3B2-3B5, the DEC PDP-11 series and VAX 11/780 under operating systems such as PCDOS, MSDOS, UNIX, XENIX and ULTRIX. In many of these environments, Screen Maker is available as a screen management complement to MDBS III data base management. This versatile tool for designing and processing polished end user interfaces supports a high degree of terminal independence. The resultant portability of application systems built with MDBS III and Screen Maker is a significant consideration.

C. MDBS Software Modules

There are two essential MDBS III software modules: the DDL Analyzer (MDBS.DDL) and the Data Base Control System (MDBS.DMS). All other modules are optional.* A data base design is specified with the Data Description Language (DDL). The DDL Analyzer uses this DDL specification to produce a corresponding data dictionary and initialize the data base. After this initialization, Data Manipulation Language (DML) commands can be used in application programs to create, retrieve and modify data in the data base. These commands are processed by the data base control system software.

---

* All of the optional software modules may not be available in all environments at all times.

In addition to the DML, other means are optionally available for accessing the data base contents. Each of these involves an additional software module that in turn uses the Data Base Control System in carrying out access requests. Optional access modules exist for interactive DML processing, English-like queries, data base browsing, batch loading, customized report generation, access within a decision support environment, and inference engine processing of data base contents. Optional modules are also available to help data base administrators and developers modify or restructure existing data bases to conform with new designs.

Although screen management and data base management are separate activities, both are important to application developers. The MDBS data base management software is complemented by two Screen Maker software modules. One allows a developer to interactively design screens and preserve them in a screen dictionary. Screen Manipulation Language (SML) commands can be used in application programs to display and process these screens. The SML commands are executed by the second module: the Screen Control System software.

There are two basic forms of the MDBS III software: standard MDBS and the RTL form of MDBS. The RTL (recovery and transaction logging) form has all the capabilities of standard MDBS. However, it has a very different Data Base Control System with built-in recovery and transaction logging facilities that do not exist in the standard MDBS. The RTL form is also furnished with a recovery utility for processing transaction logs. The **MDBS RTL Manual** describes the MDBS recovery and transaction logging facilities.

Figure I-1 shows how the various software modules can fit together in constructing and using an application system. This diagram refers to the following terms:

DDL Analyzer - the Data Description Language Analyzer that transforms a DDL schema specification into an internal data dictionary and initializes a corresponding postrelational data base.

DMU - the Design Modification Utility that a data base administrator uses to monitor space utilization, expand data base size, and modify user access privileges.

DBRS - the Data Base Restructuring System that a data base administrator uses to revise the schema of an existing data base and to re-optimize physical storage utilization.

Data Base Control System - the dictionary-driven Data Base Control System that controls all access to the encrypted postrelational data base, controls all aspects of data sharing among concurrent users, and (for the RTL form) automatically logs all transactions.

BLF - the Batch Load Facility that automatically loads the contents of an external file into a postrelational data base.

RCV - the Recovery Utility that a data base administrator uses to monitor the transaction log, roll the data base back to a former state, and selectively recover from data base corruption.

DML - the complete postrelational Data Manipulation Language whose commands can be embedded in a wide variety of host languages.

RDL Analyzer - the Report Definition Language Analyzer that a developer uses to automatically generate interactive C application programs containing all necessary I/O statements, computations, and DML logic.

ISD - the Interactive Screen Designer that allows screens to be interactively created and revised for an application system's screen dictionary.

SCS - the dictionary-driven Screen Control System that carries out all SML requests made by an application program.

SML - the high-level Screen Manipulation Language whose commands can be embedded in an application program with full screen independence and terminal independence.

IDML - the Interactive Data Manipulation Language for interactive DML training, testing DML logic, and ad hoc data base surgery.

IBS - the Interactive Browsing System for interactively browsing through data base records for purposes of viewing, editing, or creating their contents.

QRS - the non-procedural, English-like Query Retrieval System for satisfying ad hoc and "what-if" information needs.

KnowledgeMan - the decision support environment with a complete set of synergistically integrated knowledge management facilities including spreadsheet, graphics, text processing, procedural modeling, local data management, SQL inquiry, etc.; also a valuable prototyping tool for developers.

Guru - the artificial intelligence environment for constructing and consulting business expert systems.

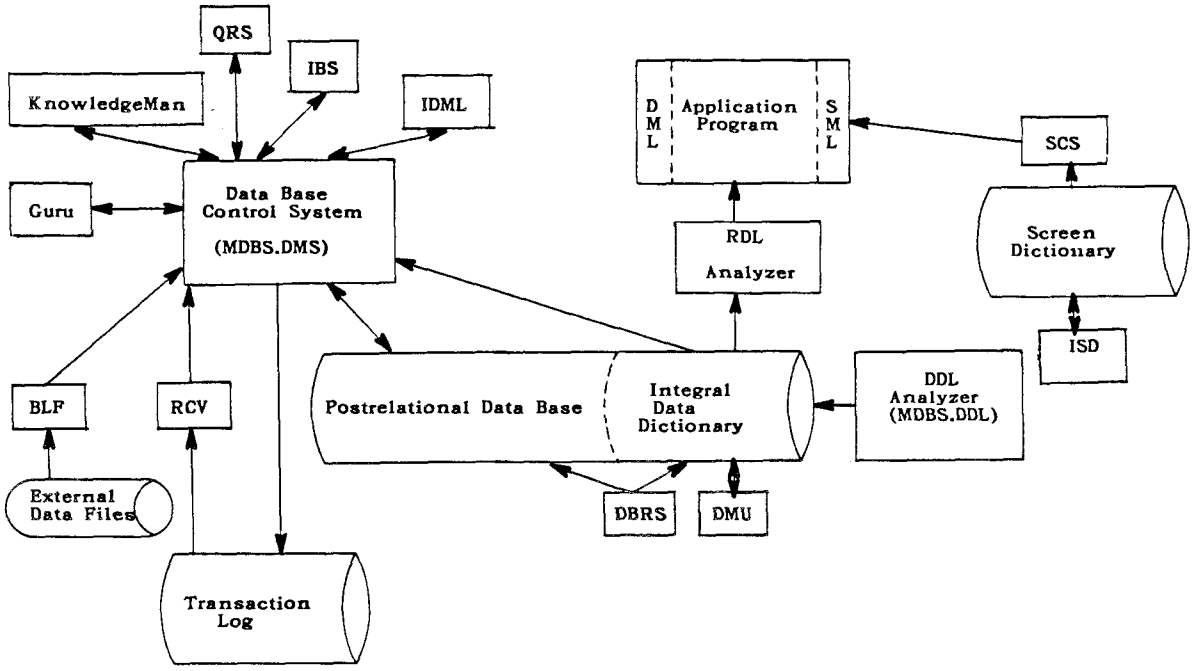


Figure I-1. MDBS Software Modules

D. Data Base Design

An initial step in developing an application system is designing the logical structure of its data base. This logical structure is called a schema. It shows the kinds of data and relationships that will exist in the data base. The importance of dbms facilities for schematically representing an application world in a clear, concise, self-documenting fashion cannot be overstated. MDBS III provides a wealth of logical data structuring facilities that are non-existent in data base management systems that follow the old inverted, hierarchical, relational and network data models.* By eliminating the old structuring restrictions, MDBS III makes it easy for the application developer to design a schema that accurately mirrors the real world.

* For a detailed comparative analysis of the postrelational and conventional data models, the most definitive presentation to date is: Micro Database Management - Practical Techniques for Application Development by R.H. Bonczek, et. al., Academic Press, 1984, 536 pp.

The expressive power and flexibility that MDBS III provides for schema design result from its capacity to directly represent every kind of naturally occurring relationship that is commonly encountered in real world applications. With conventional data models, schema design involves an attempt to fit the application into a schema that does not violate the structural restrictions imposed by a relational, hierarchical, or network dbms. With MDBS III, a developer is free to simply allow the schema to conform to the application world. Every relationship can be given a semantically meaningful name and each relationship's integrity is automatically guaranteed by the data base control system.

The real world relationships that can be directly represented in a postrelational schema include:

- one-to-one, one-to-many, and many-to-many relationships between two record types (Figure I-2)
- recursive one-to-one, one-to-many, and many-to-many relationships involving a single record type (Figure I-3)
- forked one-to-one, one-to-many, and many-to-many relationships among many record types (Figure I-4)

Multiple semantically-distinct relationships can exist between the same pair of record types and there is absolutely no restriction on the overall configuration of a schema's relationships.

Each MDBS schema looks like the world it represents, with every relationship depicted in a clear, self-documenting way. MDBS schemas are invariably more lucid and concise than conventional schemas where a developer must resort to field redundancy, artificial record types, extraneous sets, or various modeling tricks.

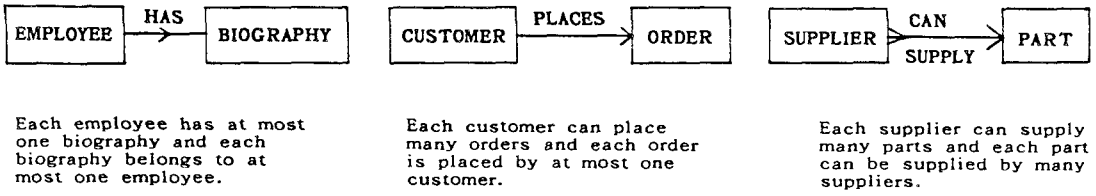


Figure I-2. Examples of the Three Kinds of Relationships between a Pair of Record Types

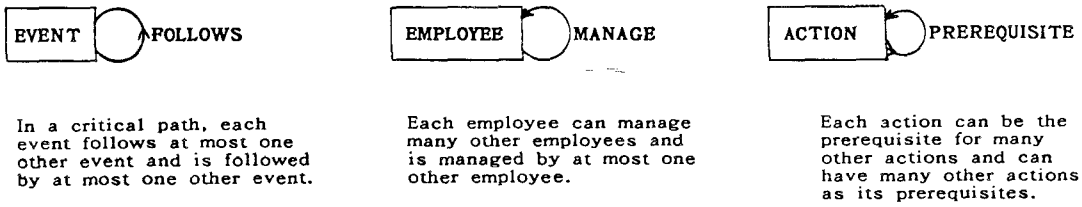


Figure I-3. Examples of the Three Kinds of Recursive Relationships Involving a Single Record Type

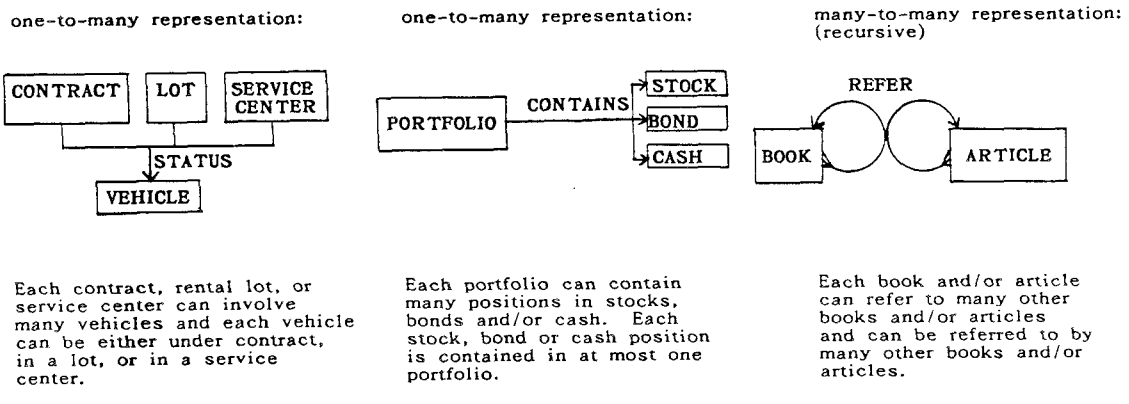


Figure I-4. Examples of Some Forked Relationships

DDL Analyzer. The MDBS III Data Description Language is a streamlined, free-form language for formally specifying the schematic structure of a postrelational data base. It supports nine data types for fields and numerous relationship orderings such as lifo, fifo, next, prior, and multi-field sorts. Built-in sorting sequences are provided for several languages including English, Danish, French, German, Swedish, Norwegian and Finnish. In addition to specifying a data base's logical structure, the DDL can also be used to control physical placement of records, to prescribe data security conditions, and to define data integrity constraints.

The DDL Analyzer (MDBS.DDL) is a program that can be used interactively to create, edit, save, and retrieve the DDL specification of a given application's schema. This DDL source specification can also be analyzed by MDBS.DDL for consistency. If an inconsistency is detected, a message explaining the error appears on the screen. It can then be corrected with the MDBS.DDL editing facilities. If no inconsistency is detected, the user can request MDBS.DDL to generate a data dictionary from the DDL source and initialize a corresponding data base. The foregoing steps are summarized in Figure I-5. MDBS.DDL can also be used in a batch mode to initialize a data base.

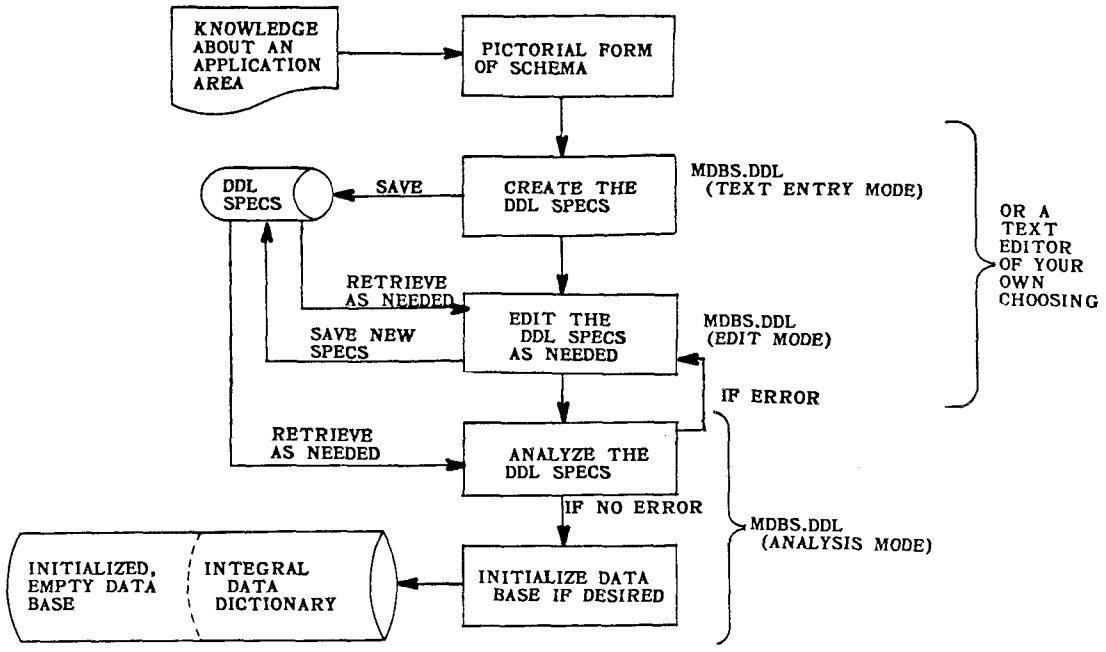


Figure I-5. Using the MDBS.DDL Module

Chapters II - VII of this manual describe the MDBS Data Description Language and DDL Analyzer operations in detail.

DBRS. The MDBS III Data Base Restructuring System (DBRS) supports all types of schematic restructuring for existing postrelational data bases. This includes the addition and removal of fields, record types, or relationships for MDBS schemas. Data Base contents are automatically revised to conform with the new schematic structure. This optional software module is documented in the **MDBS DBRS Manual.**

E. Access Languages

MDBS III supports a variety of access languages for processing postrelational data bases. Each language is oriented toward a particular kind of processing need or purpose. Each can be used without any knowledge of how data is physically organized in the data base. The most that is needed is the data base's schematic diagram or a subschema diagram that portrays only those aspects of the overall schema that are of interest to a particular processing task. Processing can begin anywhere in a schema and any schematic relationship can be processed in any direction.

The additional ability of MDBS III access languages to support refined field-oriented processing results in a high degree of data independence, insulating application programs and queries from schematic changes. All processing of an MDBS III data base is entirely dictionary driven. A processing request from any of the access languages is always executed by the Data Base Control System (MDBS.DMS) on the basis of information held in the data dictionary. It is fully independent of all host programming languages. This software module is the guardian of data base security and integrity, the monitor of concurrent processing situations, the virtual memory manager, and the controller of all internal physical aspects of data access. By mainframe standards, it is very compact in spite of its support of advanced postrelational processing.

DML. The MDBS III Data Manipulation Language (DML) provides a complete set of commands for processing all aspects of a postrelational data base. After a data base has been initialized, application programs containing DML commands can be written. The language in which the application program is written is called the host language (e.g., COBOL). An application program serves as an interface between the data base and an end user of the application system. The end user typically is not a programmer and is not familiar with data structures. Nevertheless, the end user desires to put data into the data base or modify data that exists in the data base or extract data from the data base. Application programs are usually menu-driven; through the menu, the program determines what it is that an end user wants to do. Control is then passed to a portion of the program having DML commands that will perform the desired data manipulation: loading, modification, or extraction of data. However, this data manipulation is not explicitly performed in the application program. Instead, the application program's DML requests are actually executed by the data base control system.

A major software component of MDBS is its data base control system. This is a collection of data base management routines, collectively referred to as MDBS.DMS. An application program calls these routines as needed. They carry out all physical details required to perform the data manipulation indicated by a DML command. The routines consult the data dictionary in the course of performing the data manipulation. By embedding a DML command in an application program, the programmer is giving a command to the data base control system. In order to use DML commands, the writer of an application program needs only to know the data base's schema. The application programmer is not concerned with the physical details involved in executing a DML command. There is no concern with pointers, with searching indices, with merging files, with file formats, with disk characteristics, with disk I/O, with free space management, etc. The working relationship between an application program and MDBS.DMS in an application system is shown in Figure I-6.

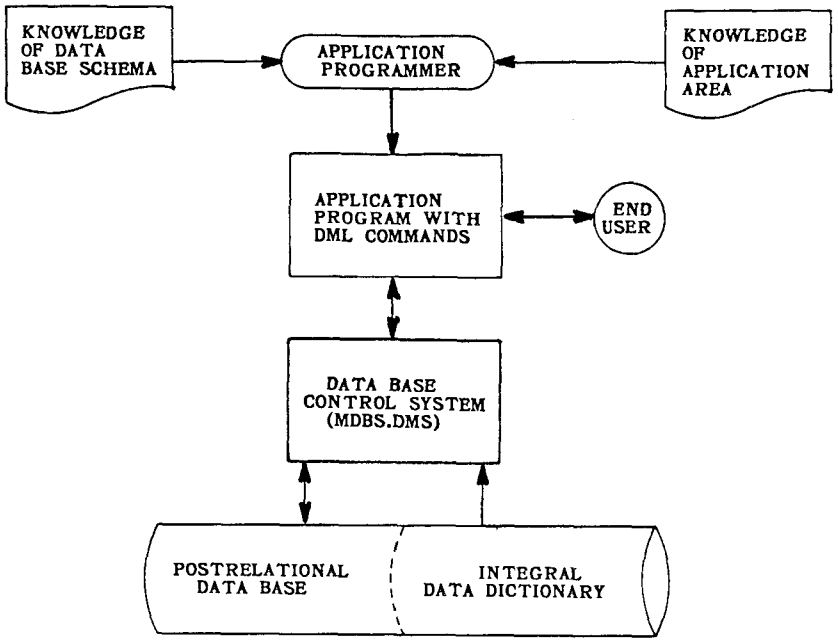


Figure I-6. Using the MDBS.DMS Module

It must be stressed that the MDBS DML provides a standard, uniform language for handling all data manipulation regardless of the application system being developed and regardless of:

the central processor being used,
the mixture of disk drives available,
the operating system being employed,
the host languages desired.

There are many DML commands for record-at-a-time processing. There is also a group of extremely powerful Boolean DML commands -- each of which generates an entire collection of records satisfying desired criteria. The MDBS III DML eliminates the troublesome aspects of CODASYL-network DML. Because the MDBS III DML allows many records to be simultaneously available for immediate processing, CODASYL's current-of-run-unit bottleneck is absent. Furthermore, the postrelational DML logic obsoletes the restrictive current-of-set navigation method that lies at the heart of CODASYL DML.

MDBS III DML commands can be invoked from application programs written in COBOL, Pascal, BASIC. FORTRAN, C and/or assembler. They can also be directly invoked as innate operations at any point within an interactive KnowledgeMan decision support session. Developers can also embed DML commands in programs written in the KnowledgeMan programming language or in expert system rule sets built in the Guru artificial intelligence environment. The DML commands are documented in the **MDBS DMS Manual.**

IDML. The MDBS III Interactive Data Manipulation Language (IDML) supports the interactive execution of nearly all DML commands. It is independent of all programming languages. The optional IDML software module has a built-in help facility, supports data dictionary inquiries, and parameterized macros for frequently used command fragments or command sequences. This access language is especially useful for DML training, testing DML logic, and performing ad hoc data base surgery. It is described in the **MDBS IDML Manual.**

QRS. The MDBS III Query Retrieval System (QRS) accepts nonprocedural structured English requests from non-programmers. After analyzing a query request, QRS internally employs the necessary DML logic for accessing desired data from a data base. This data is then presented as a report in tabular form or saved as a flat file on disk. QRS can be used directly from the operating system or it can be used at any juncture within an interactive KnowledgeMan or Guru session. In either mode of operation, QRS is particularly valuable for satisfying the ad hoc and "what if " information needs of end users, developers and administrators.

The QRS language itself is considerably more powerful and flexible than those of hierarchical or network systems, and it has been specially designed to be much easier to use than the relational SQL language. Rather than specifying the tables to be used and equating redundant fields existing in those tables, a QRS user merely states the names of the desired relationships.

Included among the many QRS features are user-definable jargon and commands, tabular end user views, virtual fields, data dictionary viewing, environment control options, fully selective retrieval, arithmetic expressions, dynamic output sorting, control breaks (with full statistics), value labels, batched query execution, and a DIF output capability. All aspects of the query language and the optional QRS module are presented in the **MDBS QRS Manual.**

RDL. The MDBS III Report Definition Language (RDL) is a nonprocedural language that can be used to formally specify the characteristics of a desired report and the prompting behavior of a program that will generate that report. An RDL specification is input to the RDL Analyzer. This optional MDBS module is an automatic program generator which produces an efficient C language program containing all DML logic, all computations, all input prompting, and all output formatting needed to generate a report having the desired content and appearance.

Included among the many RDL features are extensive output controls, built-in statistical functions, user-definable functions, free-form layouts for report details, detail sorting, headers/footers for pages and for detail groupings, multi-level grouping, fully selective retrieval criteria, wildcard string and character matching, character class matching, and computed variables. The **MDBS RDL Manual** contains a full description of the RDL and the RDL Analyzer.

IBS. The MDBS III Interactive Browsing System (IBS) is a menu driven interface to postrelational data bases. It can be used to browse through a data base's records in order to view their data values or for on-screen editing of record contents. IBS can also be used to create and delete records. As such, it provides an access facility that can be used by non-programmers, as well as the application system's developer and administrator. IBS has a built-in help facility, supports data dictionary inquiries, and allows multiple methods for reaching a desired record. The optional IBS module is fully described in the **MDBS IBS Manual.**

BLF. The MDBS III Batch Load Facility (BLF) is an optional module that automatically loads the contents of external flat files into a postrelational data base. From a single file it is able to create data base records of various types and correctly establish their interrelationships within a data base. Thus, BLF allows data from external sources to be incorporated into a data base without any programming effort. This software module is fully described in the **MDBS BLF Manual.**

F. Data Integrity Facilities

Every professional developer realizes that preserving an application system's data integrity is essential. MDBS III provides a variety of automated facilities for reducing the possibility of integrity breaches and for recovering from any data base destruction or corruption that may occur. These integrity facilities exist above and beyond MDBS III's extensive data security mechanisms for preventing unauthorized modifications to the data base.

The MDBS III data base control system automatically prevents a data value from being stored for a field if it is not consistent with the field's type. MDBS.DMS also checks for valid data values. For instance, invalid dates and times are automatically prevented for date and time fields. For any field, the developer can designate a range of feasible values. Held in the data dictionary, these feasibility ranges are automatically enforced by the data base control system. The referential integrity of all one-to-one and one-to-many relationships is automatically guaranteed. A developer can also specify that fixed relationship retention is to be enforced for any schematic relationships.

To reduce the possibility of data base corruption due to external factors, standard MDBS supports developer controlled buffer flushing. In addition, the RTL form supports page image posting, transaction logging, and selective recovery. Page image posting allows warm restarts in single user processing situations. Transaction logging is pertinent to both single user and multiuser applications. RTL's data base control system can automatically log all transactions to a log file. The scope of logical work units is developer controllable. Each transaction is accompanied by the identity of the user who performed it. In the event of data base corruption, the Recovery (RCV) Utility can regenerate a valid data base up to the point of the last completed logical work unit. Selective regeneration is also permitted to allow data base contents to be rolled back to any desired earlier state. The RTL form of MDBS is fully described in the **MDBS RTL Manual.**

G. Data Security Facilities

The security mechanisms built into MDBS III are extensive, even by mainframe standards. Security control information for a data base is maintained in the integral data dictionary. This includes each user's password and access privileges, plus the access restrictions defined for each field, record type, area and relationship existing in the schema. The security control information itself can be viewed and modified only by authorized users of DBRS or the MDBS III Design Modification Utility (DMU). Design modification capabilities of the optional DMU module are described in the **MDBS DMU Manual.**

To be able to use any of the MDBS III access languages or utilities for data base access, a bona fide user name and associated password must be provided. The retrieval and/or modification access privileges of that user are automatically observed by the data base control system. If a user's privileges do not supersede the access restrictions on a data resource, then that user's attempts to access it are denied. One of the more innovative aspects of this security mechanism is the ability to define access restrictions for data relationships, a valuable feature rarely supported by other dbms software.

Of course, no security mechanism is complete without built-in data encryption. MDBS III supports fully selective encryption, allowing the developer to control which fields are to have their values automatically encrypted. Naturally, this encryption occurs for data held in both the data base and any of its transaction logs.

H. Performance Tuning Facilities

Because of its direct, concise postrelational representation of real world relationships, MDBS III enjoys very significant processing and storage advantages over conventional data base management systems. The avoidance of traditional chaining and redundancy methods for implementing relationships gives MDBS III a further processing speed advantage. Beyond these inherent performance benefits, MDBS III provides a variety of performance tuning controls that allow the developer to adjust certain aspects of data base control system's behavior in the interests of maximizing its performance with respect to a particular application system.

The MDBS III tuning facilities include automatic data compression for designated fields, assignment of record types to desired areas (e.g., devices), control over area page sizes, physical clustering of logically related records, variable field replication, record storage/retrieval via hashing, and multiple secondary key indexes for any record type. In addition, a developer can request the MDBS III Data Base Restructuring System (DBRS) to rebuild an existing data base so that the physical placements of contents are optimized with respect to data base design criteria.

I. Multiuser Processing

Ensuring the integrity of a data base that is being used concurrently by multiple users is a crucial concern for application system developers. Versions of MDBS III implemented for multiuser environments provide the developer with a complete range of locking and contention controls. These enable the developer to effectively avoid disastrous consequences that can result when one end user attempts to either view or alter data that another user is updating.

For maximum flexibility and performance optimization, locking can be applied to individual records. relationships, or groups of records. MDBS III supports both passive and active locking techniques. A contention count protocol is provided to release the system from potential deadlock situations. For each application program, the developer can control the number of automatic retries that will occur whenever an attempt is made to access a locked resource. The time interval between retries is also developer controllable.

The MDBS III locking and contention controls are the same across all supported multiuser operating systems and local area networks. These include the UNIX, XENIX and ULTRIX multiuser operating systems and the IBM, Novell, and 3COM EtherShare local area networks. The underlying data base control system implementation is dependent on the nature of the multiuser environment. Characteristics of MDBS III multiuser processing that are applicable to particular environments are documented in pertinent **MDBS System Specific Manuals** and/or **Multiuser Supplements.**

J. MDBS III and Screen Maker

Screen Maker is a very extensive software tool that enables professional application developers to create highly sophisticated, extremely portable end user interfaces. Its flexible, powerful screen management facilities are an ideal complement to MDBS III's data base management abilities in the construction and administration of truly polished application systems. As with MDBS III, maximizing developer productivity is the keynote of the Screen Maker. The various Screen Maker capabilities are fully documented in the set of **Screen Maker Reference Manuals.**

Using Screen Maker's Interactive Screen Designer (ISD), a developer can interactively create and revise the screens that are to be used by an application system. The screen designs are maintained in a screen dictionary. To access and process these stored screens, a high-level screen manipulation language (SML) is provided. SML commands can be invoked from within the host programming language used to implement an application system. Because all SML processing is dictionary-driven, Screen Maker furnishes very high degrees of both screen and terminal independence, as well as very significant savings in application system program size and development time.

ISD is a menu-driven screen design facility used to both build and maintain an application system's screen dictionary. Each screen in the dictionary is created only once, regardless of how many different application programs will be using it. As each screen is designed, the developer can associate multiple windows with it. Similarly, many frames can be associated with each window of a screen as that window is designed. A frame is simply a screen position through which an application program can output a data value for end user viewing and/or input a data value from an end user into a program variable.

With ISD, the developer has complete control over visual locations of a screen's windows and their respective frames. As each screen, window or frame is interactively designed, the developer can give it various special effects (including foreground/background colors, full/half intensity, etc.), specify any literal characters it is to contain, create help text for it, and assign a message line to it. ISD gives the developer full control over the positioning, content and special effects for such literals, message lines and help text.

As each aspect of a screen, window or frame is designed, a visual image of it appears on the console screen. Any characteristics of a screen, window or frame can be redesigned at any time by the application developer or administrator. Such changes automatically alter the screen dictionary and are entirely independent of existing application programs. Thus ISD allows the layout and other characteristics of any screen to be quickly and easily changed without modifying or even recompiling any application programs that use the screen.

In addition to this high degree of screen independence, Screen Maker supports a high degree of terminal independence. Both the screen dictionary and the application programs that use it are fully insulated from changes in the types of terminals being used. A single screen dictionary can simultaneously drive the terminals being used. A single screen dictionary can simultaneously drive the screen I/O processing of the same or different application programs concurrently executing on machines with different kinds of terminals. Thus Screen Maker greatly amplifies the results that a developer can achieve, by making an application system's software and screen dictionary perfectly portable across a wide range of terminal environments.

Screen Maker's high-level SML commands can be invoked from a variety of host programming languages to make use of any existing screen dictionary. Each of the high-level SML commands has a very simple syntax and it typically replaces dozens or even hundreds of program statements that would be required to accomplish a comparable screen I/O task. In this way, Screen Maker drastically extends a host language's screen handling abilities, by allowing an application developer to govern all the dynamics of screen usage via a few high level commands. Any application program containing SML commands for screen processing can also contain MDBS III DML commands for data base processing.

SML commands are furnished for the following kinds of screen manipulation tasks:

- selecting a desired screen, window or frame existing in the screen dictionary
- displaying a frame, a window with any or all of its frames, or a screen with any or all of its windows
- accepting end user data input or data editing for any frame
- saving the present appearance of any frame, window or screen
- clearing any displayed frame, window or screen from the terminal screen
- restoring any saved frame, window or screen appearance to the terminal screen
- moving windows or frames to new positions on the terminal screen
- displaying, moving and clearing the help text or message line associated with any screen, window or frame

Screen Maker's Screen Control System (SCS) carries out each SML request embedded in an application program. The dictionary driven SCS processing causes all screen, window and frame characteristics defined in the screen dictionary to be automatically observed while an application program executes.

For advanced developers with very unusual screen management needs, Screen Maker also provides a comprehensive library of over one hundred specialized screen support routines. Like the higher-level SML commands, these routines can be invoked within a host programming language.

K. MDBS III and KnowledgeMan: Decision Support

KnowledgeMan is the highly acclaimed decision support environment that provides end users with a means for satisfying many of their own knowledge management needs. By helping end users and casual developers to be more productive, KnowledgeMan frees the professional application developer to concentrate on truly challenging applications involving large data bases. KnowledgeMan can directly access these massive MDBS III data bases subject to all of the usual security and integrity controls enforced by the MDBS III data base control system. The tight integration of MDBS III with KnowledgeMan makes the data held in massive MDBS III data bases susceptible to fast local decision support processing, without endangering the security or integrity of these central data bases.

KnowledgeMan is a single program that encompasses a broad spectrum of knowledge management facilities for supporting decision makers. These* include:

relational data base management
spreadsheet analysis
business graphics
full-screen text processing
customized report generation
ad hoc, SQL inquiry
mathematical functions and calculations
forms management
customized procedures
remote communications

They are fully described in the **KnowledgeMan Reference Manual** and its various supplements.

---

* Some KnowledgeMan facilities are optional and some may not be available in all environments at all times.

Unlike ordinary integrated software, KnowledgeMan's facilities are not merely separate tools operating within a window manager. KnowledgeMan obsoletes the cumbersome process of "cutting and pasting" and the inconvenience of being forced to switch back and forth among different tools and windows. Nor does KnowledgeMan follow the restrictive nested approach to "integration" used by 1-2-3, Symphony, Framework, and their clones.* With KnowledgeMan, a user is not forced to carry on all processing within the constraints of a single dominant component such as a spreadsheet or outline processor.

KnowledgeMan is based on a synergistic philosophy of integration that allows any facility to be used independently of the others. It also allows multiple facilities to be used together within a single operation, resulting in a total effect that is much greater than the sum of individual effects. Individually, each KnowledgeMan facility is quite extensive when compared to standalone packages. Fused together, they support many kinds of processing that are simply impossible with collections of leading standalone packages or with conventional "integrated" software.

KnowledgeMan has been carefully designed to give users plenty of room to grow. It is simple enough that novice users can become productive in a very short time. As experience and needs increase, a user is free to grow in any desired direction with KnowledgeMan's universe of knowledge management possibilities. One direction is toward the use of massive postrelational MDBS III data bases. For instance, in the midst of a KnowledgeMan session QRS can be invoked to query a postrelational data base. Upon exiting QRS, KnowledgeMan processing resumes and a single request will incorporate query results into KnowledgeMan's local relational data base. Furthermore, MDBS III application programs for processing postrelational data bases can be executed at any desired juncture within a KnowledgeMan session. These programs can use data exported by KnowledgeMan for updating a postrelational data base or they can extract information from the postrelational data base for subsequent import into KnowledgeMan's local processing environment.

There is an even closer connection between MDBS III and KnowledgeMan. As explained in the **KnowledgeMan Reference Manual**, the entire MDBS III data manipulation language is available as a fully integral component of KnowledgeMan. Any DML command can be invoked at any point in an interactive KnowledgeMan session to transfer data between a central postrelational data base and local KnowledgeMan variables, arrays, spreadsheets, or relational tables. DML commands can also be embedded in the cell definitions of spreadsheets and in customized KnowledgeMan procedures. Application developers can use these latter two techniques to provide KnowledgeMan users with direct access to the latest information held in a central MDBS III data base. This requires no DML comprehension on the part of KnowledgeMan's end users.

---

* 1-2-3 and Symphony are trademarks of Lotus and Framework is a trademark of Ashton-Tate.

L. MDBS III and Guru: Expert Systems

Guru is an artificial intelligence environment designed especially for business problem solving. In addition to its extensive facilities for business computing (spreadsheet, procedural modeling, graphics, etc.), Guru provides full-scale facilities for expert system construction and expert system consultation. All of these are blended together into a single environment. Guru is well-suited for developing artificially intelligent application systems with their own built-in expert reasoning capabilities. Expert systems built with Guru can directly process spreadsheets, relational data tables, models, forms, etc. In addition, they can directly access MDBS III data bases subject to all the usual security and integrity controls enforced by the MDBS data base control system.

Ordinary expert system development tools do not support even the most basic kinds of knowledge representation and processing: real data base management, spreadsheets, structured programming, business graphics, remote communications, forms management, text processing and so on. Not only does Guru handle all of these, it also handles reasoning knowledge in the convenient guise of rules. An expert system is constructed by building a set of rules that capture an expert's knowledge about solving problems in an application area. Unlike ordinary tools, Guru allows rules to be specified in terms of spreadsheet calculations, relational data base retrieval, statistics generation, graphics display, model execution, MDBS DML commands, and so forth.

When an end user consults an expert system, Guru's innate inference engine reasons with the rule set in order to reach a solution. End users can directly ask for a consultation or the developer can embed consultation requests in application programs that end users execute. In either case, the inference engine is able to carry out all business computing actions specified in the rule set's rules. Guru's inference engine is able to use both forward and reverse reasoning, accommodate fuzzy variables, reason about uncertain situations, and explain the line of reasoning it used in reaching a solution. Guru's capabilities are fully described in the two-volume **Guru Reference Manual.**

M. Summary of the MDBS Data Description Language and Software

The remainder of this manual concentrates on the Data Description Language that can be used to formally specify schemas designed for a postrelational data base. It also focuses on the DDL Analyzer which is the software module provided for processing a DDL specification. This chapter concludes with summaries of the DDL and DDL Analyzer features.

1. Summary of MDBS DDL Data Structuring Features:

   a. Up to 255 record types are allowed per logical data structure.

   b. Variable and fixed length records are supported.

   c. Up to 65535 data items (fields) are allowed per record type.

   d. The following kinds of data items are allowed: character, integer, real (floating point), string (variable length character data item), binary (variable length data item), unsigned, internal decimal, time, date.

   e. Encryption of a data item's values is supported.

   f. A feasibility range can be specified for a data item's values.

   g. The following kinds of sets are allowed for defining relationships among record types: one-to-many (i.e., CODASYL set), one-to-one, many-to-many.

   h. A record type can be the owner of many sets (of any kind) and the member of many sets (of any kind); many sets (of any kind) can exist between two record types.

   i. Any set can be forked into multiple member record types and/or multiple owner record types.

   j. Recursive sets are permitted.

   k. Sorted, LIFO, FIFO, next, prior, and immaterial orderings supported for member records of a one-to-many set. These orderings are supported for both owner records and member records of a many-to-many set.

   l. A sorted ordering can be ascending or descending and can be based on multiple data item types.

m. Sorted orderings can be based on English sequences or on any of a variety of non-English (e.g., French) sequences.

n. Automatic or manual record insertion for sets.

o. Fixed or optional record retention for sets.

p. Aside from the main data base area, up to 15 extra (physically distinct) areas can be defined for holding data base records. In some environments, there is a maximum of seven extra areas.

q. Occurrences of a record type can be assigned to one or more areas.

r. Related occurrences of more than one record type can be physically clustered.

s. Records can be "hashed" into the data base for fast, direct access.

t. An extensive access code mechanism is provided for automatic enforcement of read and write access constraints placed on each data base user.

u. Record interrelationships are not implemented by redundancy of data values (as in flat file systems), nor are they implemented by pointer chaining (as in most CODASYL network and hierarchical systems). For superior performance, MDBS record interrelationships are implemented with a proprietary technique involving enhanced, multi-level, indexed, dynamic pointer arrays.

2. Summary of the DDL Analyzer (MDBS.DDL) Features:

   a. Supports free format DDL source text.

   b. Can be used in batch manner to analyze a DDL source text file and initialize a data base for it.

   c. Can be used interactively in text entry mode which supports:

      i) Creating DDL source text
      ii) Listing DDL source text
      iii) Deleting DDL source text
      iv) Saving DDL source text on a disk file
      v) Retrieving DDL source text from a disk file
      vi) Renumbering DDL source text

   d. Can be used interactively in an edit mode which allows line editing of DDL source text.

   e. Can be used interactively in an analysis mode which:

      i) Analyzes DDL source text for consistency
      ii) Displays error message if error detected
      iii) Can initialize the data base if no error is detected

II. DEFINITIONS AND EXAMPLES

A. Data Base Areas and Pages

A data base consists of one or more areas. An area exists on a disk file. Each area consists of two or more pages. A page is a fixed-size block of bytes or words that is transferred as a logical whole between central memory and disk. This transfer is automatically performed as needed by MDBS.DMS in order to carry out DML commands. Each area can hold up to 4095 pages (8191 in some environments). A few pages in each area are reserved for use by MDBS.DMS. These are called System pages. System pages contain data dictionary (and other) information. A DML command to store data will cause the MDBS.DMS to place that data on the non-System pages in an area.

The name of a data base is stated with the DDL. When the data base is initialized, one area having the same name as the data base is automatically prepared for use. This area is referred to as the main data base area. The System pages of this area contain the data dictionary. Additional areas (up to 7 or 15 more) can be optionally defined with the DDL. The DDL is used to give each of these extra areas a name and each is automatically prepared for use when the data base is initialized. The designer can control:

the name of the file on which an area resides.
the disk on which an area resides, and
the disk drive to which an area is assigned.

This means that all data base areas do not need to be on-line when the data base is initialized by MDBS.DDL or when the data base is used by MDBS.DMS in response to a DML command (or query). However, the area having the data base name (i.e., the main area) should always be kept on-line, since it contains the data dictionary. MDBS.DMS needs to consult the data dictionary in order to carry out DML commands. Notice that some part of the data base can exist on fast drives. while another part of the data base is on relatively slower (less expensive) drives.

Figure II-1 illustrates the foregoing relationships. In this example, the data base DBEX consists of four areas. These areas are named DBEX, AR4, BAREA and AR3. They have been defined as residing on FILEA, FILEB, FILEC, and FILED, respectively. DBEX has three System pages that contain the data dictionary; the other areas have two System pages apiece. FILEA is on Disk1, FILED is on Disk3, and the other two files are on Disk2. The DBEX area has been assigned to Drive1. The other three areas have been assigned to Drive2. Suppose an application program needs to access data from the DBEX and BAREA areas, then Disk1 is mounted in Drive1 and Disk2 is mounted in Drive2. Suppose that another application program needs to access data from the DBEX, AR3, and BAREA areas. Since both AR3 and BAREA are assigned to the same drive but are on different disks, they cannot be used simultaneously. However, the application program can prompt the end user to assure that the appropriate disk is in Drive2 at the proper time. Simultaneous usage of the two areas is accomplished by allowing

one of them to reside on Disk1 or by allowing them to reside on the same disk or by assigning one of them to a third drive. The file name and drive for an area can be altered with the DML when that area is "opened" in an application program.

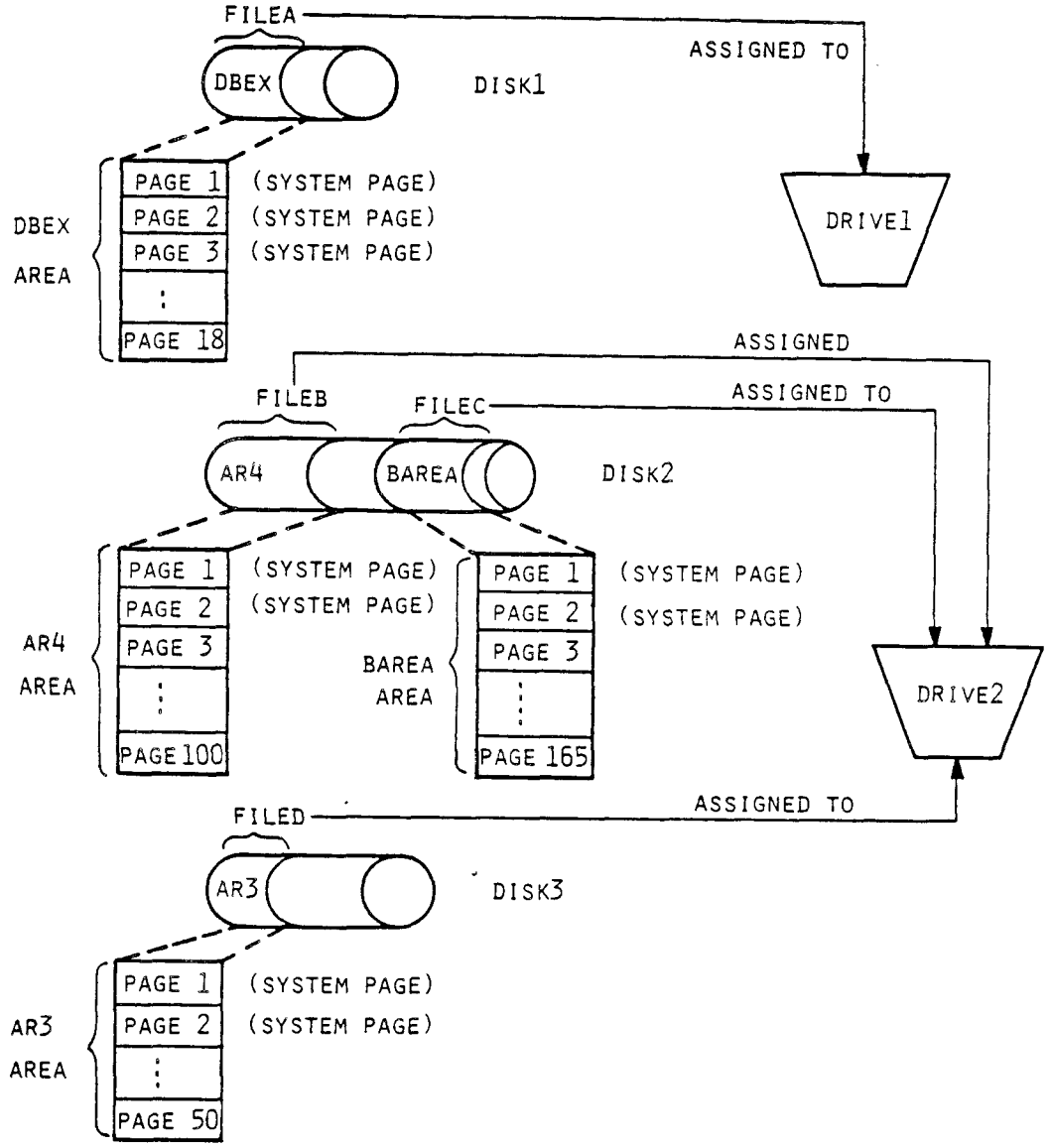


Figure II-1. Areas and Pages

The selection of an appropriate page size is important, since it influences the number of pages in memory during system execution and thus impacts the performance of the system. If the memory available for page buffers is relatively limited, it is advisable to select a small page size to maximize the number of pages in memory. For instance, performance with four small pages in memory will typically surpass performances based on two larger pages in memory. On the

other hand, if the buffer region is large, then a comparatively large page size can be advantageous, providing a minimum of about 5 to 8 of these pages will fit in the buffer region.

There are certain lower limits on the page size. For Version 3c, it must be at least 256. For Version 3a, the page size must be a multiple of 256 and. furthermore, it must be large enough to hold an occurrence of the largest record type in the area. Another consideration involved is how the record occurrences are distributed on a page. It is possible (depending on the size of the record) that records are put on a page in such a way that a great deal of room is "wasted" on each page. As an example, consider the extreme case where only one record type of 260 bytes has been assigned to an area with a page size of 512 bytes. Since records are not split over two or more pages, this would mean that only one record is being stored on a page. This can result in a great deal of "wasted" space. except that MDBS will attempt to use such space to hold internally used indices. Consideration of the factors such as these, will aid the application developer in selecting a good page size for an area.

B. The Logical Structure of an MDBS Data Base

Data Items and Record Types

The logical structure of a data base is specified in terms of data items. record types. and sets. A record type is a named group of zero, one, or more data items. Data items (fields) are the most elemental components of a logical structure. Every data item is given a name and is described in terms of the type and size of its values. The permissible types and sizes are discussed later in Section II-C. Two data items in the same record type cannot be given the same name (the same item name may be used in different record types).

As an example. we may want a data base to contain information about all departments in an organization. For each department we may want to keep track of the department's number, name, and location. The logical structure for such a data base would have a record type (call it DEPT) which consists of three data items: call them DNUMBER, NAME, LOCATION. Pictorially, a record type is represented by a labeled rectangle. with the names of its data items inside. The DEPT record type is represented as shown in Figure II-2. This Figure also shows four other record types: EMPLOYEE, BIOGRAPH, JOB, SKILL. No two record types can have the same name. The meaning of the arrows in this logical structure is discussed later.

For each record type declared in a data base's logical structure, there can exist zero, one, or more occurrences of that record type in the data base itself. An occurrence of a record type consists of a value for each of the record type's data items. For instance, an occurrence of the DEPT record type might be [101,SALES,CHICAGO]. Another occurrence of the same record type might be [108,FOUNDRY,DETROIT]. A sample occurrence of the record type JOB is [A15,JANITOR]. MDBS places no upper limit on the number of occurrences for a record type that can exist in a data base.

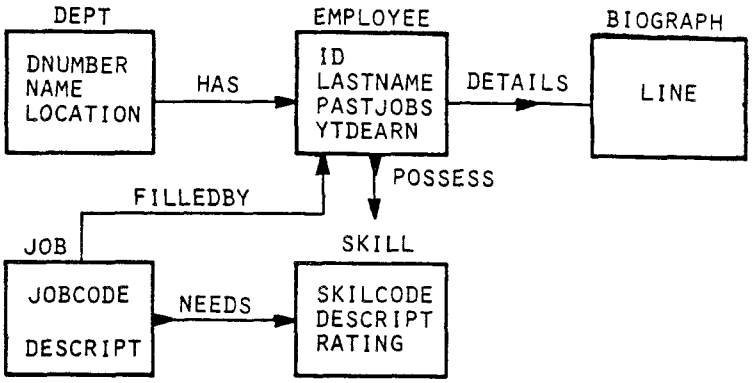


Figure II-2. A Sample Logical Structure

A data item can optionally be specified to be an array. It is then called a repeating item. For instance, PASTJOBS in EMPLOYEE could be a repeating item of length 3, if we desire to keep track of the three most recent jobs of each employee. A data value of PASTJOBS would consist of an array of length 3 (i.e., an array having three entries).

An occurrence of a record type is called a record occurrence (or simply: a record). A value for a data item is called an item occurrence. Thus a record occurrence exists in the data base and contains an item occurrence for each data item in the record type. It must be emphasized that item occurrences and record occurrences exist in the data base itself, while data items and record types are used to define the logical structure of the data base. The DDL is used to specify a data base's logical structure in terms of data items and record types. The DML is used to create, modify, and extract item occurrences and record occurrences of a data base. This data manipulation is always made in accordance with the data base's logical structure. To create a new occurrence of the DEPT record type, a DML create command is qualified by DEPT. If another application program needs to generate a list of all employee names, it would use the DML to request all record occurrences of EMPLOYEE and for each of these it would request the values of the LASTNAME and FNAME data items. Needless to say, MDBS.DMS automatically performs the tasks of physically locating the record occurrences, bringing them into central memory, and depositing the desired data values into application program variables.

As each record type is defined in the DDL, the designer has the option of assigning it to one or more areas. This means that when MDBS.DMS creates occurrences of that record type, they are stored in the assigned area(s). More than one record type can be assigned to the same area. One result is that frequently accessed record occurrences can be clustered for use on a fast disk drive. Another result is that record occurrences with particularly sensitive information can be physically protected, by assigning them to an area on a separate disk. This disk is manually placed on-line only for users with sufficient security clearance.

Relationships Between Record Types

In designing the logical structure of a data base, it is vital for an application developer to have some means for defining relationships among record types. In systems that support only flat file structures, this is accomplished by redundancy. These flat file structures are also called "relational" or tabular structures. One or more data items in one record type are repeated in another record type. Occurrences of one record type that are related to a given occurrence of another record type can then be found by looking for matching values between the repeated data items. For instance, to represent the fact that a department has employees, the DNUMBER data item of DEPT could be repeated as a data item in EMPLOYEE. For a given occurrence of EMPLOYEE, a DEPT occurrence with a matching value for DNUMBER could be found. Conversely, for a given occurrence of DEPT, all EMPLOYEE occurrences having a matching DNUMBER value could be found. This approach to representing a relationship between two record types (cross-referencing by redundancy) is entirely permissible within MDBS data structures.

However, it must be pointed out that the flat file approach to data structuring has a number of inherent drawbacks. It does not convey the semantics of a relationship. This is important when multiple, semantically-distinct relationships exist between two record types. From a data integrity standpoint, there is no structural mechanism for restricting a relationship to be a one-to-many or one-to-one relationship. In looking at a flat file structure, the relationships between record types are not always easy to pick out, especially if two record types are indirectly related through several other record types. This problem is compounded for realistic logical structures, which typically involve many record types (not just two or three). The number of record types in a flat file structure proliferates rapidly if the application developer decides to use a third normal form (from relational theory). For realistic application development, flat file structures tend to cause greater storage and processing overhead than do the alternative approaches to representing inter-record type relationships (discussed below).

Although the application developer using MDBS is free to define flat file structures, he should be aware of the inherent limitations and inconveniences of the flat file approach. The application developer using MDBS could also restrict himself to hierarchical structures, shallow network structures, or CODASYL network structures. Here again, limitations, inefficiencies, and inconveniences are frequently incurred.

With MDBS, a named relationship between two record types can be defined as a set. One record type is called the owner of the set and the other record type is called the member of the set.[1] The term "set" as used in data base management has no relationship to the

---

[1] Sets with multiple owner record types and multiple member record types, as well as recursive sets, are also allowed in MDBS; these are explained in a later section entitled "Special Ways of Using Sets."

mathematical notion of a set. In the pictorial representation of a logical structure (e.g., Figure II-2), a set is represented by a line between the owner record type and the member record type. The name of the set is placed alongside the line. An arrowhead appearing on a line always points from the owner record type to the member record type. The positioning of arrowheads on lines is used to distinguish among the various kinds of sets. If an arrowhead is adjacent to the member record type, then there can be many member record occurrences related to each record occurrence, otherwise there is no more than one related member record for any owner record. If an arrowhead is adjacent to the owner record type, then there can be many owner record occurrences related to each member record occurrence, otherwise there is no more than one related owner record for any member record.

Four distinct kinds of sets are allowed in MDBS data structures:

1:N Set for representing a one-to-many relationship (Owner ⟶ member)
N:M Set for representing a many-to-many relationship (Owner ➤⟶ member)

1:1 Set for representing a one-to-one relationship (Owner ⟶➤ member)
N:1 Set for representing a many-to-one relationship (Owner ➤⟶ member)

In all MDBS manuals the term "set", used by itself, refers to all four kinds of MDBS sets.

A 1:N set (⟶) is a traditional CODASYL set. (It is also analogous to what is called a parent-child relationship in hierarchical systems.) In specifying a 1:N set, the application developer is stating that one occurrence of the owner record type can be associated with N ($N \geq 0$) occurrences of the member record type. However, an occurrence of the member record type is associated with no more than one occurrence of the owner record type. In Figure II-2, HAS is an example of a 1:N set. One department can have many employees, but an employee is associated with no more than one department. FILLEDBY is another 1:N set; a job type can be filled by many employees, but an employee fills only one job at a time. Unlike shallow network systems, in MDBS structures a record type can be the owner of many sets and it can be the member of many sets. Unlike hierarchical systems, a record type can be the member of many sets.

An N:M set (➤⟶) relationship between two record types means that an occurrence of the owner record type may be associated with M ($M \geq 0$) occurrences of the member record type. Furthermore, an occurrence of the member record type may be associated with N ($N \geq 0$) occurrences of the owner record type. Thus an N:M set is used to directly represent a many-to-many relationship between record types. In Figure II-2, NEEDS is an N:M set. A job may need many skills and a skill may be needed by many jobs. POSSESS is another N:M set; an employee may possess many skills and a skill may be possessed by many employees. Such relationships cannot be directly represented in hierarchical, shallow network, or CODASYL network structures. For

instance. the CODASYL approach to representing the relationship between JOB and SKILL necessitates the definition of an additional, artificial record type that is the member of a set owned by JOB and the member of another set owned by SKILL. This indirect method for representing many-to-many relationships is supported in MDBS. However, it is recommended that the N:M set be used instead, since it gives a simpler logical structure, makes application programming much easier, requires less storage, and yields faster processing.

A 1:1 set (⟵▶⟶) relationship between two record types indicates that an occurrence of the owner record type is associated with, at most, one occurrence of the member record type and an occurrence of the member record type is associated with, at most, one occurrence of the owner record type. In Figure II - 2, DETAILS is an example of a 1:1 set. An employee has no more than one biography and a biography is associated with no more than one employee. LINE is a string (i.e., variable length) data item, which could have a very long occurrence for some employees and a short value for other employees. LINE could have been included in the EMPLOYEE record type. However, if it is less frequently used than other employee data, it is reasonable to place LINE in a separate record type. This means that lengthy biographical data will not be brought into main memory each time an EMPLOYEE record is accessed. It also allows biographical information to be assigned to an area that is usually not on-line and that is utilized by a slow access disk drive.

The N:1 set (▶⟶) is an operationalization of the functional dependency notion in relational theory. It is the exact converse of a 1:N set. For a N:1 set, a member occurrence can have N ($N \geq 0$) owner occurrences. However, an owner occurrence is associated with, at most, one member occurrence.

A set occurrence consists of an occurrence of the set's owner record type and all associated member record occurrences (for example. a department and all employees in that department; another example, a job and all skills that job needs). Set occurrences for a 1:N set do not overlap; they have no record occurrences in common. The same is true for set occurrences of a 1:1 set. Set occurrences for a N:M set can overlap. For instance, a SKILL occurrence can belong to more than one JOB occurrence.

Set Ordering

For a given set, the member record occurrences associated with an owner record occurrence can be declared (in the DDL) to be ordered. This is referred to as the set's member order. Similarly, the owner record occurrences associated with a member record can be declared to be ordered. This is referred to as the set's owner order. Specifying a member order is meaningful only for 1:N sets and N:M sets (for 1:1 sets and N:1 sets there is no more than one member record occurrence per owner record occurrence). Specifying an owner order is meaningful only for N:M sets and N:1 sets (for 1:1 sets and 1:N sets there is at most, one owner record occurrence per member record occurrence). All member records for a given owner record can be accessed in the logical

sequence defined by the set's member order. All owner records for a given member record can be accessed in the logical sequence defined by the set's owner order. In either case, the MDBS DML also allows reverse sequence accessing.

There are six allowable set orders: **FIFO, LIFO, NEXT, PRIOR, IMMATERIAL, SORTED.** These orderings are logical; they do not control the physical placement of record occurrences. Regardless of the ordering chosen, MDBS automatically maintains that logical order during the creation, deletion, and modification of record occurrences. The six kinds of set orders are described with respect to specifying member order. The descriptions would be identical for specifying owner order except that the term "owner" replaces the term "member" and vice versa.

FIFO (first in. first out) - The first member record occurrence that becomes associated with a given owner record is logically the first to be accessed. The second member record to become associated with that owner record is logically the second to be accessed, and so forth.

Example: If the member order for HAS in Figure II-2 is FIFO, then the employees associated with a given department can be accessed on a first in - first out basis.

LIFO (last in. first out) - The last member record occurrence to become associated with a given owner record is logically the first to be accessed. The second-to-last is the second to be accessed, and so forth.

Example: If the member order for HAS is LIFO. then the employees associated with a given department can be accessed on a last in-first out basis.

NEXT - A new member record that becomes associated with a given owner record is inserted into the logical sequence of member records immediately after the "current" member record of the set. ("Current" is defined in the **MDBS DMS Manual** discussion; a DML command allows a user to make any member record "current.")

PRIOR - A new member record that becomes associated with a given owner record is inserted into the logical sequence of member records immediately before the current member record of the set.

Example: Suppose the member order of HAS is PRIOR. Suppose department 101 has four employee record occurrences and the third one is presently the current member of HAS. The next employee record occurrence to become associated with department 101 will enter the logical sequence of employee records immediately before the third one.

IMMATERIAL - This order option should be used if the application designer does not care about the set order. This ordering allows MDBS to realize certain efficiencies and should be used if possible.

SORTED - The member record occurrences for a given owner record occurrence are logically sorted on the basis of one or more data items in the member record type. These data items are collectively called the member sort key. The sorting for each data item in a sort key may be on an ascending or descending basis. In the DDL, the application designer can also specify how MDBS.DMS should handle member records with duplicate sort key values. The options are to prohibit duplicates or to incorporate them on a FIFO, LIFO or IMMATERIAL basis. Finally, the designer can optionally specify how many bytes of a sort key are to be used in maintaining the set's member order index.

Example: Suppose the member order for the HAS set is sorted on last name, first name, and ID (all on an ascending basis). Then all employees in a given department can be accessed according to this sorted order. It also permits MDBS.DMS to very quickly find the EMPLOYEE record occurrence that has a particular sort key value.

These six ordering options have been described for a set's member order. The same options are available for a set's owner order. For instance, the NEEDS set in Figure II-2 might have as its owner order: sorted on JOBCODE. The member order could be: sorted on SKILCODE. The result is that all jobs for a particular skill can be accessed in sorted order on the basis of JOBCODE values. All skills for a given job can be accessed in sorted order on the basis of SKILCODE values.

To have MDBS.DMS maintain two (or more) different orderings, two (or more) sets between the two record types can be defined. For example, if it is desired to have the employees in a department maintained in sorted order on the basis of ID and to be maintained in sorted order on the basis of LASTNAME, then two 1:N sets would be defined in place of HAS. One would have as its member order: sorted on ID. The other would have as its member order: sorted on LASTNAME.

Special Ways of Using Sets

To this point a set has been used to represent a relationship between two different record types defined by the application designer. There are four other ways for using sets: SYSTEM-owned sets, recursive sets, sets with multiple member record types, and sets with multiple owner record types. In the last three situations, all four kinds of sets are allowed. In the case of SYSTEM-owned sets, only 1:1 and 1:N sets are permitted. There is also a special 1:N set called $SYSSET that is automatically defined by MDBS.DDL when the data base is initialized.

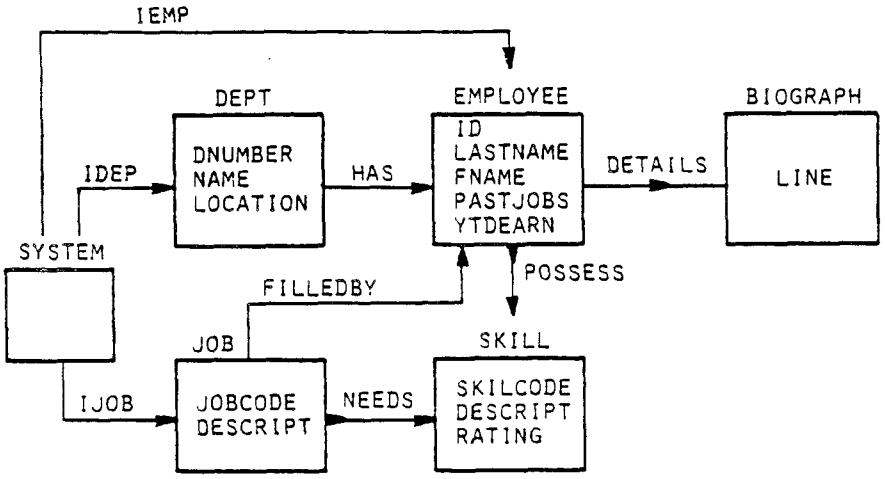


Figure II-3. System-owned Sets

System-owned set. When a data base is initialized, MDBS.DDL automatically creates a special record type called SYSTEM. This special record type has one record occurrence in the data base; the SYSTEM record occurrence is also created when the data base is initialized. It contains no data. A system-owned set is a set that has SYSTEM as its owner and some other record type as its member. Figure II-3 shows three system-owned sets: IEMP, IDEP, IJOB. A record type can be the member of zero, one, or more system-owned sets.

Declaring a sorted system-owned set causes MDBS to automatically form and maintain an index into the occurrences of the member record type. For efficiency, this is actually a multi-level, balanced index. The index is maintained in accordance with the member order declared for the system-owned set (e.g., FIFO, LIFO, SORTED, etc.). Suppose that IEMP is sorted on LASTNAME and FNAME. Then a sorted list (on last and first names) of employees can be quickly produced through IEMP. This also allows MDBS.DMS to quickly find a particular employee occurrence on the basis of its sort key value (i.e., a last and first name). Many system-owned sets can be declared for the same member record type. A system-owned set incurs very little overhead relative to most other kinds of sets.

Recursive set. A set is recursive when it has the same record type as both owner and member. It is a means for relating occurrences of a record type to other occurrences of the same record type. For example, an employee can manage many other employees who, in turn, manage still other employees, and so forth. Thus there is a one-to-many relationship among employees. This recursive situation is easily handled by the recursive set, MANAGES, that is shown in Figure II-4. Just like any other 1:N set, MANAGES will have a member order. The need for recursive 1:N and N:M sets is encountered in many application areas (e.g., the chart of accounts in accounting applications). Recursive sets are not allowed in flat file, hierarchical, shallow network, or CODASYL network systems.

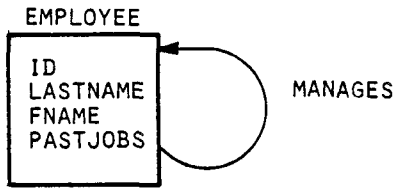


Figure II-4. A Recursive Set

Multiple Member Forked Sets. Just as with CODASYL networks, MDBS extended networks permit 1:N sets that have multiple member record types. With MDBS, multiple member record types are also allowed for 1:1, N:1, and N:M sets. Figure II-5 gives an example of a forked 1:N set with multiple member record types. This set is called CONTAINS.

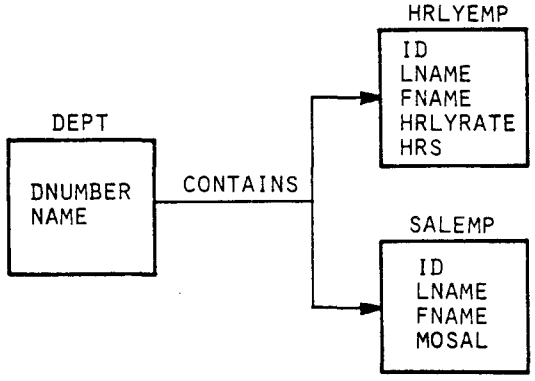


Figure II-5. Multiple Member Forked Set

Because this is a 1:N set, a record occurrence of DEPT can have many member record occurrences, some of the type HRLYEMP and some of the type SALEMP. Note that the two member record types have some data items in common and others that are different.

All of the usual set orders are allowed with a multiple member set. For instance, CONTAIN could be sorted (ascending) on LNAME and FNAME. The employees in a given department can then be accessed alphabetically, regardless of whether they are salaried or hourly workers. In addition, the record type name could be included as part of the sort key. If CONTAINS is sorted first on (ascending) record type name and then on LNAME and FNAME, then for a given department, an alphabetical list of hourly employees followed by an alphabetical list of salaried employees could be accessed. A maximum of 127 member record types are allowed for any set.

Multiple Owner Forked Sets. With MDBS, more than one record type is allowed as the owner of any set. An example of a forked N:M set with multiple owner record types is shown in Figure II-6. It indicates that many skills can be possessed by an hourly or salaried employee. Conversely, a skill may be possessed by many employees. If the member order for POSSESS is sorted (ascending) on SKILCODE. then the skills for each employee (salaried or hourly) can be accessed in

sorted order. If the owner order for POSSESS is sorted (ascending) on ID, then all employees having a given skill (regardless of whether they are salaried or hourly) can be accessed in order according to ID. A maximum of 127 owner record types are allowed for any set.

$SYSSET Set. This is a special FIFO 1:N set that MDBS.DDL automatically defines whenever a data base is initialized. Its owner record type is SYSTEM. Its member record type(s) is (are) automatically and dynamically established (and re-established) by MDBS.DMS during the execution of an application program. The member record occurrences of $SYSSET at any given moment during the execution of an application program are those record occurrences that are the result of a Boolean DML command. The result of a Boolean DML command consists of zero, one, or more record occurrences (see the **MDBS.DMS Manual** for a full description of Boolean DML commands).

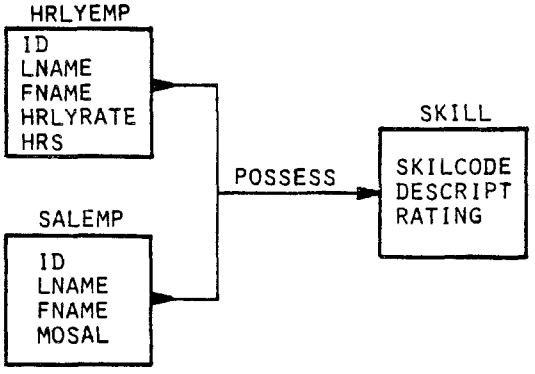


Figure II-6. Multiple Owner Forked Set

Set Retention

Any set in the DDL source can be declared to have a FIXED or OPTIONAL set retention. If the set retention is FIXED, then any connection that is made between an owner record occurrence and a member record occurrence is permanent. Although either record could be deleted, they cannot be disconnected from each other without deletion. OPTIONAL set retention means that there is no restriction on disconnecting an owner record from a member record, (i.e., the two records can be disconnected without deleting either record).

Set Insertion (Connection)

Inserting a member (or owner) record into a set means that it is being connected to an owner (or member) record. The member record type(s) of a set can be declared to have either automatic (AUTO) or MANUAL set insertion. AUTO means that whenever an occurrence of the member record type is created in the data base, it is automatically connected to an occurrence of the set's owner record type(s). MANUAL

means that whenever a member record is created in the data base, a DML command can be used to connect it to an occurrence of the set's owner record type(s). The owner record type(s) of a set can also be declared to be either AUTO or MANUAL. AUTO means that whenever an occurrence of the owner record type is created in the data base, it is automatically connected to an occurrence of the set's member record type(s). MANUAL means that this connecting is not automatic, but can be accomplished as needed with a DML command.

C. Types and Sizes of Data Items

Each data item declared with the DDL must be given a type and size. MDBS supports nine different types of data items. Data item size refers to the length of an occurrence of the data item. Permissible data item size depends on the type of the data item. The nine types of data items are:

integer	unsigned	string
real	internal decimal (idec)	time
binary	character	date

No size is explicitly stated in the DDL source for date or time data items. The size for an internal decimal data item is stated in terms of the total number of digits to be used (the number of digits to the right of the decimal point is also stated). Sizes for the other six types of data items are stated in terms of bytes. Automatic data compression occurs for string, binary, date, and time data items. Suppose FNAME is a string data item of 12 bytes. When the item occurrence 'Bob' is stored, MDBS allocates fewer than 12 (not 12) bytes. Suppose LINE in Figure II-2 is a string data item of size 250 that repeats five times. Thus a maximum of 1250 bytes are allowed in a LINE occurrence. If only the first 375 bytes are nonblank in a given LINE occurrence, MDBS allocates only about 375 bytes for that occurrence.

Table II-1 shows the minimum and maximum size (n) that can be specified in the DDL for each type of data item. The corresponding storage requirement (in bytes) for a specified n is shown in the table. For example, an occurrence of an IDEC data item of size n=8 will occupy 5 bytes of storage. The maximum range of values for each type of data item is also given. These are maximums with respect to the data base. A host language may impose additional constraints (see MDBS System Specific Manuals). For instance, a data item that is integer of size n=2 can take on any value in the range

-32768 to 32767 (-2^{8n-1} to $2^{8n-1}-1$ where $n=2$).

Further data item range examples are shown in Appendix E.

The following are guidelines for determining what type to declare for a data item. If the data item's values can take on positive and negative integer values, then an appropriate type declaration is integer. If the values are integers, but never negative, then unsigned is a proper choice. The selection of real versus internal

decimal representation for a data item is largely dependent on the host language(s) being used (see the appropriate system specific manuals). For a host language that supports both real and internal decimal variables, real is used for data items with floating point values; if great precision is required for a data item's values, in order to avoid small round-off errors during host language computations, then the internal decimal (IDEC) type should be used. The binary type is used when a data item's values need to be manipulated in a binary form within a host language program (e.g., representation of graphical images, voice images, etc.). If a data item's values are times or dates, then the time or date type (respectively) should be used. This takes advantage of MDBS data compression. The exceptions are where the times cover more than 256 hours, where dates range across more than 126 years, or where dates prior to 01/01/0000 are needed. In such cases, the character data type can be used.

Table II-1. Types of Data Items

Type	Min Size(n)	Max Size(n)	Storage requirement in bytes (n=specified size)	Maximum Range (n=specified size)
integer n	1	16	n	-2^{8n-1} to $2^{8n-1}-1$
real n	2	16	n	$-2^{127}(1-256^{1-n})$ to $2^{127}(1-256^{1-n})$
binary n	1	65535	typically < n	00000000 to 11111111 (binary) for each byte
unsigned n	1	16	n	0 to $256^{n}-1$
idec n(digits)	1	30	((n+1)/2)+1 rounded down	$-10^{63}(1-10^{-n})$ to $10^{63}(1-10^{-n})$ if n is even $-10^{63}(1-10^{-n-1})$ to $10^{63}(1-10^{-n-1})$ if n is odd
character n	1	65535	n	any legitimate character (usually 0 to 255) for each byte
string n	1	250	typically < n	any character (except control characters) for each byte
time	--	--	3	00:00:00 to 255:59:59
date	--	--	2	any 126 year period

For character data there is a choice of using either the character type or the string type. If the string type is selected, data values are compressed while in the data base, through the elimination of trailing blanks and the compression of consecutive repeating characters. Of course, these values are restored to their original form when extracted from the data base. For the string type, any control characters (including tabs) are permanently transformed into blanks. For alphanumeric data values, (especially for text data) the string type is recommended if there are numerous instances of

consecutive repeating characters. If there are very few consecutive repeating characters or if n is small (less than 7), then the character type is recommended. If a string type is declared, but no compression is possible for a data value, then the storage space used for that data value is n + 2 + n/127 (rounded down). This is also the worst case storage space requirement for a binary data type (occurring when a binary value cannot be compressed). The best case storage requirement for a value of a string or binary data item is 1 byte.

With the exception of binary, any of these types of data items may appear in a sort key. Except for binary data items, any data item can have a feasibility range specified with the DDL. When a feasibility range for a data item exists, the data base control system will automatically perform appropriate data integrity checks whenever attempts are made to store or modify an occurrence of the data item.

When a data value is stored into a data base, it is automatically converted into an internal MDBS representation. This internal representation is the same, regardless of the host language of the application program that loaded the data. When data is retrieved, it is automatically converted from its internal MDBS representation into the representation used by the host language of the retrieval program (or by the query system). Thus application programs in many different host languages can access the same data base. The conversion between internal MDBS representation and a host language representation is described in the system specific manual for that host language. Since most host languages do not support all of the MDBS types. two different MDBS types may be mapped into a single variable type in the host language during data storage and retrieval. These mappings are explained in the system specific manuals. An application developer who intends to use multiple host languages. should be aware of the different language mappings and should select a data item's type and size on the basis of the most restrictive host language mapping.

D. Controlling Record Placement

As mentioned earlier, a record type (and therefore its occurrences) can be assigned to one or more areas. This gives the application developer some control over the physical placement of records. An even greater degree of control is allowed. With MDBS, the application developer has three options for governing the placement of a record occurrence within its allotted area(s): CALC, clustering, and system default.

CALC. Any data item or group of data items within a record type can be declared to be the calc key for that record type. The calc key value for an occurrence of that record type is input to a hash routine that controls the physical placement of the record occurrence in the allotted area(s). The application designer has the option of prohibiting duplicate calc key values, if desired. If a record type has a calc key, any of its record occurrences can be directly accessed on the basis of a calc key value. Note that a record type can have a calc key and can also be the member of one or more system-owned sets.

MDBS has several calc algorithms. If the first algorithm yields a page with insufficient space for the record occurrence being created, the second algorithm is used, and so forth until a page with sufficient space is found. If none of the algorithms calculates a page with sufficient space. then a spillover approach to overflow is used commencing with the first page following the page computed by the last algorithm. The spillover is restricted to the area in which the last computed page is located and is a wrap-around spillover within that area. If no pages in the area have sufficient space. then MDBS.DMS issues an error indicating that there is insufficient room.

Clustering. In the DDL source specification. an application designer can request MDBS to cluster member (or owner) records close to the owner (or member) record(s) with which they are associated (i.e., connected) via an indicated set. The purpose is to have all member (or owner) records for a particular owner (or member) record clustered in such a way that they can all be resident in central memory at once. This is useful (from the standpoint of minimizing disk I/O) if it is often necessary to access all member (or owner) records whenever an owner (or member) record is accessed. Note that clustering is comparable to the VIA SET record location mode proposed for CODASYL data base management systems, except that it is also available for N:M sets in MDBS.

System default. If neither CALC nor clustering is selected to control record placement within an area(s), then MDBS will control this placement.

E. Data Security

In addition to data encryption and user passwords. highly selective data security is provided by an access code technique. This is fully described in Chapter IV.

F. Miscellaneous

Comments are allowed in the DDL source for documentation purposes. Each data base, area, record type, set, and data item name can be given synonyms and/or a title. A synonym can be substituted for the original name when issuing a QRS query. Titles and synonyms are incorporated into the data dictionary. Titles can be used to provide comments about a data item, set, etc., for users of the data dictionary. They also appear on the console screen during IBS browsing. The data dictionary can be interactively queried via the QRS, IDML, and IBS modules.

III. MDBS.DDL SPECIFICATIONS

A. Notational Conventions

After the logical structure for a data base has been drawn out, as in Figure II-2, it can be formally specified with the MDBS Data Description Language. A sample data base description on an introductory level is shown in Figure III-1. An elaboration of that schema description, incorporating many advanced features, is shown in Figure III-2. This chapter gives a detailed explanation of the DDL.

A DDL source specification consists of several sections. Each section consists of various kinds of clauses; some of which are optional and some of which are required. These clauses are presented on a section-by-section basis. Discussion of those clauses that can be regarded as advanced features are denoted in the left margin by a vertical bar. The complete BNF syntactic description of all clauses appears in Appendix B. Figure III-3 shows the meanings of the notations used.

A clause is composed of one or more of the following terms: keywords, identifiers, file names, strings, integers, user names, and passwords.

Keywords are denoted by upper case letters. All other terms in a clause are in lower case. This upper case vs. lower case distinction is for documentation purposes only. Upper and lower case can be mixed in an actual DDL source specification (see Figure III-1, for example). When the data base is initialized, all lower case characters are mapped into upper case characters in the data dictionary, with the exception of characters enclosed in double quotes (""). A complete list of DDL keywords appears in Appendix A.

Identifiers used in describing a DDL section are denoted by id-1, id-2, id-3, etc. Identifiers are selected by the application designer. An identifier consists of an alphabetic character followed by from zero to seven alphanumeric characters. A DDL keyword can also be used as an identifier, by using one of the following DDL keywords prior to the identifier: IS, ARE, WITH, OF, TO, BY. A list of MDBS-QRS keywords is in Appendix G. These QRS keywords should not be used as identifiers.

Identifiers that have already been defined as the names of data items are denoted by di-1, di-2, etc. Identifiers that have previously been defined as names of record types are denoted by rt-1, rt-2, etc. Identifiers that have already been defined as set names are denoted by st-1, st-2, etc. Identifiers previously defined as area names are represented by ar-1, ar-2, etc.

File names used in describing a DDL section are denoted by file-1, file-2, file-3, etc. File names are chosen by the application developer; they must be fully qualified file names within the host operating system.


```
/******** sample data base description (introductory level) *********/
/********************* data base identification ********************/
database name is JOBS
     file "JOBS.DB", size is 300 pages, page size is 512 bytes
/****************** user and password definitions ******************/
user is "BOB SMITH" with GTC
user is ANALYST     with 7778XK4
/********************** record type defintions **********************/
record name is DEPT
     item name is DNUMBER      integer 1
     item name is NAME         character 12
     item name is LOCATION     string 35
record name is EMPLOYEE
     item ID                   character 9
     item LASTNAME             string 20
     item FNAME                string 12
     item PASTJOBS             string 25 occurs 3 times
     item YTDEARN              idec 7,2
record BIOGRAPH
     item LINE                 string 50 occurs 5 times
record SKILL
     item SKILCODE             integer 2
     item DESCRIPT             string 55
     item RATING               real 2
record JOB
     item JOBCODE              integer 2
     item DESCRIPT             string 30
/************************** set definitions **************************/
set name is IEMP
          type is 1:n
          owner is SYSTEM
          member is EMPLOYEE order is sorted
                             by ascending (LASTNAME, FNAME)
set name POSSESS    type is n:m
          owner is EMPLOYEE  order is sorted by az ID
          member is SKILL    order is immaterial
set DETAILS         type is 1:1
          owner is EMPLOYEE, member is BIOGRAPH
set FILLEDBY        type is 1:n
          owner is JOB, member is EMPLOYEE order is fifo
set NEEDS           n:m
          owner JOB sorted descending JOBCODE
          member SKILL sorted ascending (RATING,SKILCODE)
set HAS             1:n
          owner DEPT, member EMPLOYEE order lifo
set IDEP            1:n
          owner SYSTEM      member DEPT
set IJOB            1:n
          owner SYSTEM      member JOB fifo
end
```

Figure III-1.

```
/*****
 *
 *  sample data base description (advanced level)
 *
 *****/

db JOBS         file "JOBS.DB"
                size 300 pages, page size 1024
                logfile "JOBS.LOG"

/****************** define defaults for item types ******************/

   default for unsigned 2
   default for str 50

user "BOB SMITH" with GTC      read access (a,b) write access a
user ANALYST with 7778XK4      read (b-e)  write (b-f)
user "K FERGUSON" with "tashi"  access (b,p)
user "D LEHR" with "smiles" access (a-p)

/** define additional areas to supplement the main data base area ***/

area name is jobl
     file name is "JOBl.DB"
     size is 100 pages, pointers not allowed
     page size is 512  /* Note:  this page size would not be
                               allowed in Version 3a because
                               the main area's page size is 1024 */
     read access is (a-d)  write access is (a,p)
area JOB2       file "JOB2.DB"
                size 700 pages
     read (b,d)  write (a,b,f,p)

/********************** record type definitions **********************/

record DEPT
     in JOBS calc key is NAME nodup
     read access b  write access (a,b)
          item DNUMBER int 1
                       range is 1 to 42
                       syn is DNO
          item is NAME char 12   encrypted
                       write access b
          item LOCATION  str 35   syn LOC
                         range "A" to "ZZZZZZ"
record employee in JOBl, key is ID nodup
     read access (a,d)  write access (a,p)
          item ID char 9  encrypted range is "0" to "999999999"
          item LASTNAME str 20      range "Aa" "Zz"
          item FNAME    str 12      range "Aa" "Zz"
          item PASTJOBS str 25  occurs 3 times
          item YTDEARN  idec 7,2  encrypted range 0.00  94000.00
               read access a  write access a
```

Figure III-2. Sample DDL


```
record BIOGRAPH in JOB2
     read access b  write access p
          item LINE  str occurs 5 times
record SKILL in (JOBS,JOB2) calc key is SKILCODE nodup
     read access (b,d)  write access f
          item SKILCODE unsigned syn SC  range 0 to 3000
          item DESCRIPT str 55
          item RATING real 2               range 0.0 to 4.0
record JOB in any area
     read access b  write access (a,p)
          item JOBCODE unsigned 1          range 1 to 250
          item DESCRIPT  str

/************************ set definitions ***************************/

set IEMP, type 1:n, retention fixed
     read access d  write access p
          owner SYSTEM
          member EMPLOYEE, order sorted by ascending (lastname, Fname)
               insertion auto
set POSSESS, type n:m
     read access (a,d)  write access (f,p)
          owner EMPLOYEE, order sorted by az ID
          member SKILL
set DETAILS, type 1:1, fixed
     read access (b,d)  write access p
          owner EMPLOYEE
          member BIOGRAPH auto
set FILLEDBY, type 1:n
     read access (a-d)  write access p
          owner JOB
          member EMPLOYEE fifo auto
set NEEDS n:m
     read access b  write access (f,p)
          owner JOB sorted za JOBCODE
          member SKILL sorted az (RATING,SKILCODE)
set HAS 1:n
     read access (a,b)  write access a
          owner DEPT
          member EMPLOYEE lifo auto
set IDEP 1:n,  read access b
     owner SYSTEM   member DEPT auto
set IJOB 1:n,  read access b
     owner SYSTEM   member JOB auto fifo
END
```

Figure III-2. Sample DDL (continued)

(See Appendix H for an alternative layout of this DDL specification.)

Integers used in describing a DDL section are denoted by int-1, int-2, int-3, etc.

Strings of characters are denoted by string-1, string-2, etc. Each string within a DDL section has a unique number as its suffix. The actual string values are chosen by the application designer. A string value must fit within a pair of matching double quotes on one line of the DDL source specification.

User names, denoted by usr, are one to sixteen character (alphanumeric) names.

Passwords, denoted by pass, are one to twelve character (alphanumeric) strings.

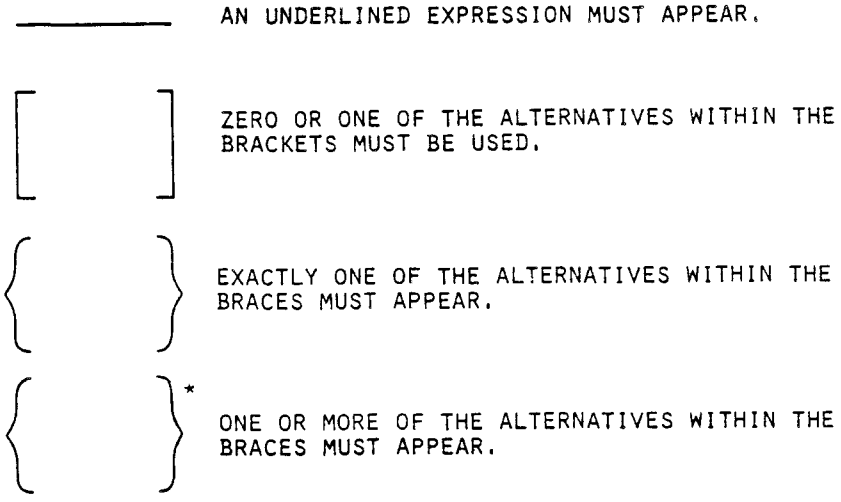


Figure III-3. Notation

B. DDL Sections

A DDL source specification consists of up to nine kinds of sections. These sections must appear in the order given below. Clauses within a section are stated in a free format (i.e., column positioning is unimportant for the DDL analyzer). Upper case and lower case can be used interchangeably. More than one clause can appear on a line; a clause (but not a term) can be split over more than one line; unless otherwise indicated, optional clauses within a section can appear in any order. Clauses can be separated by a comma or blanks. A section is terminated with a blank or period. Comments can appear between any two terms or clauses within a DDL source specification.

The kinds of DDL sections and their ordering are as follows:

```
Identification Section
User Section
Area Section (optional)
Record Sections
     Data Item Sections for each record section (optional)
Set Sections (optional)
     Owner Section for each set section
     Member Section for each set section
End Section
```

The identification section identifies the data base. The user section indicates the possible data base users (or user classes) and gives a password to each. The optional area section is used to define extra areas if they are desired. There follows a series of record sections. There is one record section for each record type in the logical structure. Associated with each record section is a data item section for each of that record type's data items. If a record type has no data items, then the corresponding record section has no data item sections. After all record types (and associated data item types) have been defined in the record (and data item) sections, all sets in the logical structure are defined in the set sections. There is one set section per set. Associated with each set section is an owner section indicating the set's owner(s) and a member section indicating the set's member(s). A DDL source specification always terminates with the end section.

Comments in a DDL source specification

/* located anywhere in the DDL source specification indicates that all text that follows is a comment, until the comment is terminated by */

Comments can extend over many lines.

Examples: /* THIS IS A COMMENT */

```
/*****
 *
 *  DDL SOURCE SPECIFICATION
 *
 *****/

/*
    RECORD TYPE DEFINITIONS
*/
```


1. Identification Section:

The identification section must be present. This section is used to give a name to the data base. In addition, the data base designer can optionally specify:

- o the physical file that will hold the data base
- o the number of pages in the data base
- o the size of a data base page
- o the physical file that will hold the logged transactions
- o a title for the data base
- o synonyms for the data base name
- o data item size defaults for the DDL specification
- o the language (English, French, German, etc.) to be used as a basis for sorting

Required Clause

{ DATABASE / DB } NAME IS id-1

The data base name is indicated by identifier id-1. A data base area named id-1 is automatically created when the data base is initialized. This is referred to as the main area of the data base.

Examples: DATABASE NAME IS MED
DB NAME IS SCHOOL
DATABASE PILOT
DB PILOT

Optional Clauses

a) File clause FILE NAME IS "file-1"

The main data base area will physically reside on the disk file indicated by file-1. MDBS.DDL will automatically create file-1 during data base initialization. File-1 is a fully qualified file name for the host operating system.
Default: based on the data base name id-1. but operating system dependent

Examples: FILE NAME IS "SCHDB"
FILE "INVEN/DB:1"
FILE "B:DEB"
FILE "/usr/db/customer.db"

b) Page clause SIZE IS int-1 PAGES

The number of pages in the area id-1 will be int-1. If the environment allows 16 areas, the maximum size is 4095 pages. If only 8 areas are allowed, the maximum size is 8191.

Default: operating system dependent (typically 50)

Examples: SIZE IS 35 PAGES
SIZE 215
SIZE 1000 PAGES

c) Page size clause PAGE SIZE IS int-2 [BYTES / WORDS]

The size of each page in the area id-1 is int-2 bytes or words.

Defaults: i) operating system dependent (typically 512 bytes)
ii) if WORDS is not specified, the page size is in terms of bytes

Examples: PAGE SIZE IS 256 BYTES
PAGE 1024 WORDS
PAGE 1024

Note: i) with MDBS version 3a, int-2 must be a multiple of 256
ii) with MDBS version 3c, int-2 must be at least 256 (but need not be a multiple of 256)
iii) with MDBS version 3a, each area must have page sizes of int-2
iv) with MDBS version 3c, no area can have page sizes that exceed int-2

d) Log file clause {LOG FILE / LOGFILE} NAME IS "file-2"

Transactions will be logged to the file indicated by file-2. MDBS.DDL will automatically create file-2 during data base initialization. This file is reinitialized by the MDBS-RTL recovery utility (called RCV). File-2 is a fully qualified file name in the host operating system.

Default: no log file for transaction logging

Examples: LOGFILE NAME IS "MED.LOG"
LOG FILE NAME IS "PILOTL"
LOG "B:DEBLOG"

e) Title clause TITLE IS "string-1"

The data base is given the title indicated by string-1.

Default: no title

Examples: TITLE IS "MEDICAL SCHEDULING DATA BASE"
TITLE "PILOT PROJECT: VERSION 5.2"

f) Synonym clause { SYN / SYNONYM } IS id-2

The name indicated by id-2 is a synonym for the data base name id-1. Many synonyms can be defined by repeated use of this clause. MDBS imposes no maximum on the number of synonyms.

Default: none

Examples: SYNONYM IS MEDSCH
SYN S

g) Default clause

DEFAULT SIZE FOR { { UNSIGNED / REAL / INTEGER / INT / BINARY / BIN / CHARACTER / CHAR / STRING / STR } { int-3 } / { DECIMAL / IDEC } { int-4[,int-5] } }

This clause is used to specify the default size for an indicated type of data item (real, integer, unsigned, etc.). A specified default size for a given type applies to all data items defined within the data base description for id-1. If no size is specified for a data item in a data item section, then the default size stated with this clause is assumed. One default clause can be used for each of the nine types of data items, except for time and date. **Default clauses must be the last clauses of an identification section.** The default size (in bytes) is indicated by int-3 with the exception of IDEC. For IDEC, int-4 is the total default size (in digits) and the optional int-5 indicates the number of digits to the right of the decimal point. If int-5 is not specified, it is assumed to be zero.

Default: none

Examples: DEFAULT SIZE FOR REAL 4
DEFAULT CHAR 200
DEFAULT IDEC 10,1
DEFAULT DECIMAL 8

h) Language clause

```
                  { DANISH   }
                  { ENGLISH  }
                  { FINNISH  }
LANGUAGE IS       { FRENCH   }
                  { GERMAN   }
                  { NORWEGIAN}
                  { SWEDISH  }
```

Any sorting carried out by any MDBS software for the data base's contents will use the normal character collating sequence of the specified language. This includes the maintenance of sorted sets, dynamic sorting of QRS output, and the sorting of report contents produced by programs that RDL generates.

Default: English (i.e., standard ASCII sequence)

Examples: LANGUAGE IS FINNISH
LANGUAGE FRENCH

2. User Section:

The user section must be present. It is used to give the names and passwords of all data base users. The following clause is used to define each user. For a data base to have five different users (or user groups), five successive clauses of this kind must be specified. No two users can have the same user name. The value of a password for providing data security is discussed in Chapter IV.

Required Clause

```
                 { usr   }      [      ]      { pass   }
USER NAME IS     {"usr"  }      [WITH  ]      {"pass"  }
                 {       }      [ ,    ]      {        }
```

The user whose name is usr has the password pass. If a user name or password is not enclosed in double quotes, then its alphabetic characters are converted to upper case. When double quotes are used, no case conversion occurs.

Examples: USER IS "Alvin Wade" WITH SECRET
USER GC15 , "DJL"
USER PRESIDENT MONROE

Optional Access Clause

An optional access clause can accompany any instance of a USER clause. This access clause is discussed in Chapter IV.

3. Area Section (optional)

The area section is optional. It is used to define additional areas for the data base. Depending on the environment, a maximum of either 7 or 15 extra areas can be defined in the area section. See the pertinent System Specific Manual.

Required Clause

AREA NAME IS id-1

An area named id-1 is automatically created when the data base is initialized. This kind of clause (together with its optional clauses shown below) is required for each additional area. Two areas should not have the same name.

Examples: AREA NAME IS REALM4
AREA IS PAYROLL
AREA FASTAC1

Optional Clauses

a) File clause FILE NAME IS "file-1"

The area named id-1 will physically reside on the disk file indicated by file-1. MDBS.DDL will automatically create file-1 during data base initialization. Two areas cannot be assigned to the same file. File-1 is a fully qualified file name for the host operating system.

Default: File-1 default is based on the id-1 area name, but is operating system dependent, (e.g., if id-1 is TREE, then the default for file-1 could be TREE.DBA).

Examples: FILE NAME IS "B:RAE"
FILE IS "USV/DB/SALES.DBA"
FILE "SCHREC/DBA:2"

b) Page clause SIZE IS int-1 PAGES

The number of pages in the area id-1 is indicated by int-1. For environments allowing 16 areas there is a maximum of 4095 pages per area. The maximum is 8191 for environments allowing no more than 8 areas.

Default: operating system dependent (typically 50)

Examples: SIZE IS 89 PAGES
SIZE 200 PAGES
SIZE 77

c) Page size clause PAGE SIZE IS int-2 [BYTES / WORDS]

The size of each page in area id-1 is int-2 bytes or words.

Defaults: i) defaults to the size of the main area
ii) if WORDS is not specified, the page size is in terms of bytes

Examples: PAGE SIZE IS 512 BYTES
PAGE 512 WORDS
PAGE 256

Note: i) with MDBS version 3a, each area's page size must equal the page size of the main area of the data base
ii) with MDBS version 3c, one area's page size can differ from another area's page size; the page size for an area must be at least 256 bytes, but cannot exceed the page size of the main area

d) Pointer control clause POINTERS ARE [NOT] ALLOWED

This clause allows the application designer to control whether or not dynamic pointer indices are allowed within the area. Disallowing dynamic pointer indices in an area, with many CALCed records. can yield higher CALC access performance. **If pointer indices are allowed in an area, then every user must have write access to that area and that area must be on-line for all DML commands that alter the data base.**

Default: dynamic pointer indices are allowed in the area

Examples: POINTERS NOT ALLOWED
POINTERS ALLOWED

e) Title clause TITLE IS "string-1"

The area is given the title indicated by string-1.

Default: no title for area

Examples: TITLE IS "AREA FOR SALARY INFORMATION"
TITLE "BIOGRAPHICAL DATA AREA"

f) Synonym clause { SYNONYM / SYN } IS id-2

The name indicated by id-2 is a synonym for the area name id-1. More than one (no maximum) synonym can be defined.

Default: no synonym

Examples: SYNONYM IS BIO
SYN WAGEAREA

g) An optional access clause can be specified for any area. This access clause is discussed in Chapter IV.

4. Record Section

One or more record sections must appear in a DDL source specification. Each record section is used to define a record type. If the record type has data items, then data item sections for each data item must immediately follow the record section. Record types can be defined in any sequence within the group of record sections. More than 255 record sections (i.e., record types) per data base are not allowed.

Required Clause

RECORD NAME IS id-1

This clause defines a record type having the name id-1. The name SYSTEM is not allowed. The SYSTEM record type is automatically defined during data base initialization.

```
Examples: RECORD NAME IS SKILL
          RECORD IS EMPLOYEE
          RECORD DEPT
```

Optional Clauses

a) Record location clause

```
            ┌ ┌ ANY AREA       ┐                         ┐
            │ └ ( {ar-1,}* )   ┘   [<CALC-key-clause>]   │
 { WITHIN } │                                            │
 {   IN   } │           ┌ OWNER  ┐                       │
            │ AREA OF   │        │     OF  id-2          │
            └           └ MEMBER ┘                       ┘
```

This optional clause allows the application designer to control where the occurrences of a record type will be located in the data base. The record type can be assigned to any area or to one or more specified areas (ar-1, ar-2, etc.) with the option of specifying a CALC key for the record type. Alternatively, the record type can be assigned to the area(s) of the owner or member of the set id-2, in which case MDBS will physically cluster those record occurrences that are related to each other through the set id-2. Later in the DDL source specification, id-2 must appear as the name of a set.

The optional CALC-key-clause, within the record location clause, has the following form:

<CALC-key-clause>:

```
{CALC KEY }[IS ]                                [NODUP ALLOWED                    ]
{CALC KEYS}[ARE]({id-3,}*) [DUPLICATES ARE [NOT] ALLOWED]
```

A calc key consists of the data item id-3 or the sequence of data items (id-3, id-4, etc.). The sequence of data items determines the calc key values. Parentheses are not required if there is only one data item in the calc key. All data items that make up a record type's calc key must be defined to be data items for that record type. The application designer can indicate whether or not two record occurrences with the same calc key values are allowed.

Default: duplicates are allowed

Examples:
```
KEY IS ID NODUP ALLOWED
KEY IS (LNAME,FNAME)
KEY IS LNAME FNAME
KEY (ID,LNAME,FNAME) NODUP
```

Defaults (for record location clause):

i) omission of this clause means that occurrences of the record type can be placed in any defined area

ii) omission of the optional CALC-key-clause means that MDBS determines the placement of record occurrences within the assigned area(s)

Examples:
```
WITHIN ANY AREA
        CALC KEY ID NODUP
WITHIN PAYROLL
        KEY (LNAME,FNAME)
IN ANY
IN DBAREA2 DBAREA3
IN (AREA9, AREA7)
        KEY IS JOBCODE NODUP
IN AREA OF MEMBER OF POSSESS
IN AREA OF OWNER FILLEDBY
IN MEMBER POSSESS
WITHIN OWNER FILLEDBY
```

b) Title clause TITLE IS "string-1"

The record type is given the title indicated by string-1.

Default: no title for the record type

Examples:
```
TITLE IS "EMPLOYEE DATA"
TITLE "SKILL DESCRIPTION AND RATING"
```

c) Synonym clause { SYNONYM / SYN } IS id-5

The name indicated by id-5 is a synonym for the record type name id-1. Multiple (no maximum) synonym clauses are allowed.

Default: no synonym

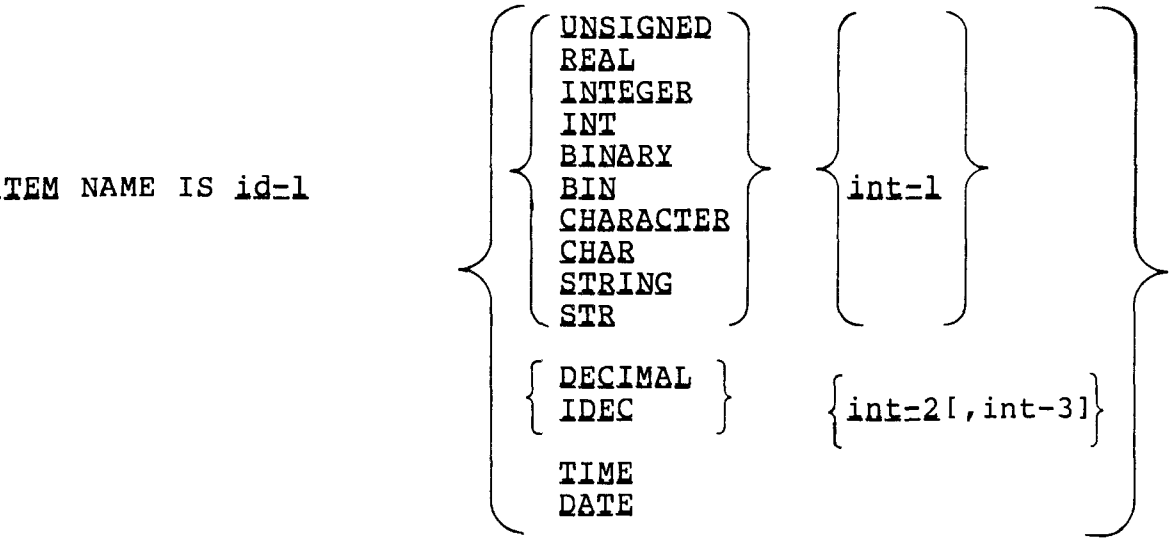
Examples: SYNONYM EMP
SYN DEP

d) An optional access clause can be specified for any record type. This access clause is discussed in Chapter IV.

5. Data Item Section

Following each record section is a group of (possibly zero) data item sections. There is one data item section for each data item in a record type. The sequence of data item sections for a given record section is irrelevant to the DDL analyzer. No more than 65535 data items per record type are allowed.

Required Clause



This clause defines a data item having the name id-1. It also indicates the type and size of that data item. Permissible types are UNSIGNED, REAL, INTEGER (or INT), BINARY (or BIN), CHARACTER (or CHAR), STRING (or STR), internal decimal denoted by IDEC or DECIMAL, TIME, and DATE. No size is specified for TIME or DATE. For internal decimal, int-2 indicates the number of digits stored and int-3 indicates the number of digits to the right of the decimal point; int-3 cannot exceed 7. The default for the optional int-3 is 0. For all other types, int-1 indicates the number of bytes for the data item. Maximum and minimum sizes were discussed in the preceding chapter.

DATE data values can be automatically converted to alternative formats by the DBCNV DML command and by the DATFORM QRS command. The DBCNV command can also be used to permit null (i.e., blank) DATE and TIME values.

```
Examples: ITEM NAME IS NAME CHAR 25
          ITEM YTDWAGE IDEC 8,2
          ITEM ZIP STRING 9
          ITEM BIRTHDT DATE
          ITEM NAME ELAPSETM TIME
          ITEM AGE INT 2
```


Optional Clause

a) Occurs clause OCCURS int-4 TIMES

This clause is used to indicate that the data item is a repeating item. Each occurrence of the repeating item is an array of length int-4. The maximum value for int-4 is 255.

Default: the data item is not a repeating item (in other words, int-4 = 1)

Examples: OCCURS 8 TIMES
OCCURS 35

b) Encryption clause IS ENCRYPTED

This clause indicates that values of the data item are stored in the data base in encrypted form. Those values are automatically decrypted when they are retrieved (subject to data security restrictions, discussed in Chapter IV).

Default: no encryption

Examples: IS ENCRYPTED
ENCRYPTED

c) Title clause TITLE IS "string-1"

The data item is given the title indicated by string-1.

Default: no title for the data item

Examples: TITLE IS "GROSS PAY"
TITLE "EMPLOYEE ADDRESS"

d) Synonym clause { SYNONYM / SYN } IS id-2

The name indicated by id-2 is a synonym for the data item name id-1. Multiple (no maximum) synonyms can be declared.

Default: no synonym

Examples: SYNONYM GP
SYN IS EMPADD

e. An optional access clause can be specified for any data item. This access clause is discussed in Chapter IV.

f. The optional range clause, for automatic feasibility checking, is discussed in Chapter V.

6. Set Section

Zero, one, or more set sections must appear in a DDL source specification. Each set section is used to define a set. A set section must be immediately followed by an owner section, which is immediately followed by a member section. Sets can be defined in any sequence within the group of set sections. MDBS imposes no maximum on the number of set sections in a DDL source specification.

Required Clause

SET NAME IS id-1

This clause defines a set having the name id-1.

Examples: SET NAME IS NAME
SET IS CONTAINS
SET POSSESS

Optional Clauses

a) Set type clause

TYPE IS { 1:N | N:M | M:N | 1:1 | N:1 }

This clause indicates whether the set is 1:N, N:M (or equivalently M:N), 1:1, or N:1. Any two distinct alphabetic characters can be used in place of N and M.

Default: 1:N

Examples: TYPE IS N:M
TYPE 1:A
IS 1:1
TYPE M:N
X:Z
A:A

b) Retention clause RETENTION IS { FIXED | OPTIONAL }

This clause declares the set retention to be either FIXED (a participating record occurrence cannot be disconnected; it can be deleted) or optional (a record occurrence can be connected and disconnected as desired).

Default: optional

Examples: RETENTION IS FIXED
RETENTION FIXED
IS OPTIONAL
FIXED

c) Title clause TITLE IS "string-1"

The set is given the title indicated by string-1.

Default: no title for the set

Examples: TITLE IS "MANY-TO-MANY RELATIONSHIP"
TITLE "EMPLOYEE POSSESSES SKILL"

d) Synonym clause { SYNONYM / SYN } IS id-2

The name indicated by id-2 is a synonym for the set name id-1. Multiple (no maximum) synonym clauses are allowed.

Default: none

Examples: SYNONYM IS POSS
SYN FBY

e) An optional access clause can be specified for any set. This access clause is discussed in Chapter IV.

7. Owner Section

An owner section must immediately follow each set section. An owner section describes the owner(s) of a set.

Required Clause

```
{OWNER IS rt-1                    }
{OWNERS ARE ({rt-2,}*)            }
```

If the set has a single owner record type, its name is indicated by rt-1. If the set has multiple owner record types, those record types are indicated by (rt-2, rt-3, etc.); the ordering of these multiple record type names is irrelevant. No more than 127 record types can own a set.

Examples:
```
OWNER IS SYSTEM
OWNER DEPT
OWNERS ARE (HRLYEMP,SALEMP)
OWNERS (SALEMP,HRLYEMP)
OWNERS REC1 REC2
```

Optional Clauses

a) Set insertion clause

```
INSERTION IS {AUTO  }
             {MANUAL}
```

This clause indicates whether or not the creation of an owner record occurrence causes it to be automatically (AUTO) connected to a member record.

Default: MANUAL

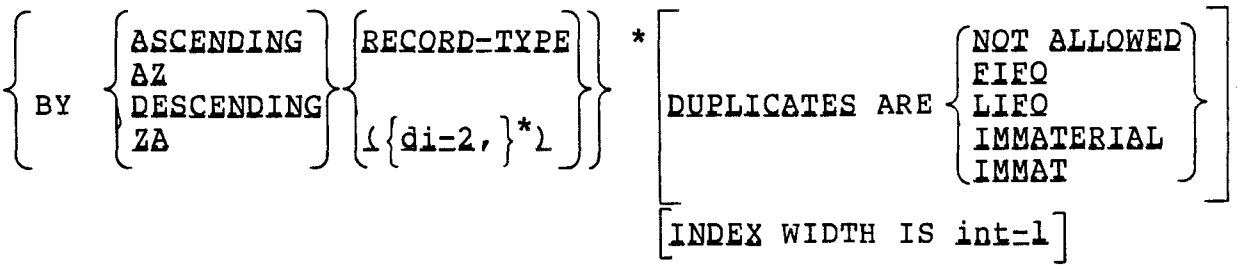
Examples:
```
INSERTION IS AUTO
MANUAL
INSERTION AUTO
```

b) Owner order clause

```
                 {FIFO                    }
                 {LIFO                    }
                 {NEXT                    }
ORDER IS         {PRIOR                   }
                 {IMMATERIAL              }
                 {IMMAT                   }
                 {SORTED≤sort-clause≥     }
```

This clause indicates the ordering of owner record occurrences that are associated with any given member record occurrence. The permissible orderings are FIFO, LIFO, NEXT, PRIOR, IMMATERIAL (or IMMAT), and SORTED. The sort-clause has the following form:

<sort-clause>:



If the owner order is SORTED, then the sort-clause is required. It indicates the basis for the sorting of owner record occurrences that are connected to a member record occurrence. The sort is made on the basis of one or more data items (di-1, di-2, etc.) that have been defined for the owner record type. A binary data item may not be used as a basis for sorting. If the set has multiple owner record types, then the owner record type names can also be used as a criterion for sorting. Thus a sort key consists of a sequence of data items, plus the possible inclusion of the literal: RECORD-TYPE. The ordering of these components within the sort key definition is from the most important to the least important. The entire sort key can be declared to be ASCENDING (AZ) or DESCENDING (ZA). Alternatively, each component (or groups of sort key components) is declared to be either ASCENDING or DESCENDING. The character code of the host operating system determines ascending and descending sequences.

Once the sort key has been specified, the designer can optionally indicate how MDBS should handle record occurrences with duplicate sort key values. The options are: NOT ALLOWED, FIFO, LIFO, IMMATERIAL (or IMMAT). The designer can also optionally indicate how many bytes (int-1) of the sort key are to be used in the multilevel owner index. For instance, if int-1 is 7, then the first seven bytes of the sort key values will be used in maintaining the index. The owner record occurrences are still sorted on the basis of the entire sort key. The index width (of up to 28 bytes) is normally chosen to maximize the uniqueness of the entries in the index, while minimizing storage requirements. For Version 3c an even width (rounded up, if necessary) is always used internally.

Defaults: i) the default for duplicates is IMMAT
ii) the default for int-1 is never longer than the sort key size


```
Examples: BY ASCENDING (LASTNAME,FNAME)
             DUPLICATES ARE IMMAT
          BY AZ LASTNAME DESCENDING ID
             DUPLICATES NOT ALLOWED
          ZA JOBCODE DUPLICATES NOT ALLOWED
             INDEX LENGTH IS 2
          ASCENDING RECORD-TYPE AZ (LNAME FNAME)
             INDEX 1
```

Default (for owner order clause): IMMAT

```
Examples: ORDER IS LIFO
          ORDER FIFO
          LIFO
          SORTED BY AZ (LASTNAME,FNAME)
          SORTED BY AZ LASTNAME ZA FNAME
             DUPLICATES FIFO
          ORDER IS SORTED ZA JOBCODE
             DUPLICATES NOT ALLOWED
             INDEX 2
```

8. Member Section

A member section must immediately follow each owner section. The member(s) of a set are described in a member section.

Required Clause

```
{MEMBER IS rt-1                    }
{MEMBERS ARE ({rt-2,}*)            }
```

If the set has a single member record type, its name is indicated by rt-1. If the set has multiple member record types, those record types are indicated by (rt-2, rt-3, etc.); the ordering of these multiple record type names is irrelevant. No more than 127 record types can be declared as members of a set.

Examples: MEMBER IS DEPT
MEMBERS ARE (HRLYEMP, SALEMP)
MEMBERS SALEMP HRLYEMP

Optional Clauses

a) Set insertion clause INSERTION IS {AUTO / MANUAL}

This clause indicates whether or not the creation of a member record occurrence causes it to be automatically (AUTO) connected to an owner record.

Default: MANUAL

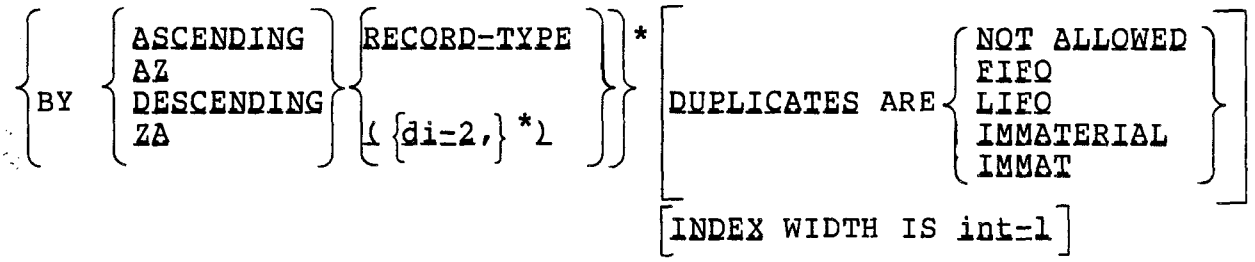
Examples: INSERTION IS AUTO
AUTO
INSERTION MANUAL

b) Member order clause

ORDER IS { FIFO / LIFO / NEXT / PRIOR / IMMATERIAL / IMMAT / SORTED ≤sort clause≥ }

This optional clause indicates the ordering of member record occurrences within a set occurrence. The permissible orderings are FIFO, LIFO, NEXT, PRIOR, IMMATERIAL (or IMMAT), and SORTED. The sort-clause has the same form as shown for the optional owner order clause in an owner section.

<sort-clause>:



If the member order is SORTED, then the sort-clause is required. It indicates the basis for the sorting of member record occurrences that are connected to an owner record occurrence. The sort is made on the basis of one or more data items (di-1, di-2, etc.) that have been defined for the member record type. A binary data item may not be used as a basis for sorting. If the set has multiple member record types, then the member record type names can also be used as a criterion for sorting. Thus a sort key consists of a sequence of data items, plus the possible inclusion of the literal: RECORD-TYPE. The ordering of these components within the sort key definition is from the most important to the least important. The entire sort key can be declared to be ASCENDING (or AZ) or DESCENDING (ZA). Alternatively, each component (or groups of sort key components) is declared to be either ASCENDING or DESCENDING.

Once the sort key has been specified, the designer can optionally indicate how MDBS should handle record occurrences with duplicate sort key values. The options are: NOT ALLOWED, FIFO, LIFO, IMMATERIAL (or IMMAT). The designer can also optionally indicate how many bytes (int-1) of the sort key are to be used in the multilevel member index. For instance, if int-1 is 7, then the first seven bytes of the sort key values will be used in maintaining the index. The member record occurrences are still sorted on the basis of the entire sort key. The maximum width is 28 bytes and the width is always even (rounded up, if needed) for Version 3c.

Defaults: i) the default for duplicates is IMMAT
ii) the default for int-1 is never longer than the sort key size

Examples:
```
BY ASCENDING SKILCODE DUPLICATES NOT ALLOWED
   DESCENDING RATING AZ SKILCODE
      DUPLICATES NOT ALLOWED INDEX 4
   ZA (DNUMBER,NAME) DUPLICATES NOT ALLOWED
   AZ LASTNAME DUPLICATES LIFO
   BY AZ (LNAME) ZA (RECORD-TYPE) INDEX 3
```

Default (for member order clause): IMMAT

Examples:
```
ORDER IS FIFO
FIFO
IS LIFO
ORDER NEXT
SORTED BY ASCENDING SKILCODE
   DUPLICATES NOT ALLOWED
SORTED AZ LASTNAME DUPLICATES LIFO INDEX 1
```

9. End Section

The end section terminates a DDL source specification.

Required Clause:

END[.]

Example: end

This page intentionally left blank.

IV. DATA SECURITY

Three approaches to data security are provided by MDBS: passwords, encryption, and access codes. The password approach is required for every MDBS data base. An application designer can optionally make use of encryption and access codes to provide very extensive data security. As discussed in Chapter II, physical security is provided by allowing some portions of the data base to be off-line while the remainder of the data base is being used.

A. Passwords

Many data base users can be defined in the User Section of a DDL source specification. MDBS will not allow any part of the data base to be accessed through the DML in an application program or through the query system, unless a user can furnish his or her assigned password. With the optional DMU module, an application developer can add, delete or change any user's password at any time, without re-initializing the data base (subject to security restrictions).

B. Encryption

Any data item defined in a DDL source specification can optionally be declared to be encrypted. Each occurrence of an encrypted data item is stored in an encrypted form. When an occurrence of an encrypted data item is accessed with either the DML or query language, MDBS automatically decodes the encrypted value. Thus the fact that a data item is encrypted is invisible to the users of DML application programs and the query language.

Encryption protects the data from unauthorized viewing that can occur through a sequential scan of auxiliary memory. If, for instance, employee name and salary information are stored in the same record, then a memory scan via the operating system can reveal the salary that accompanies each name. This kind of security breach is averted by declaring either (or both) the name or the salary data item to be encrypted.

C. Access Codes

Overview

An application designer can optionally use access codes to restrict a user's access to only certain parts of a data base. MDBS differentiates between two kinds of access: read access and write access. Read access allows data to be found and viewed. Write access allows data (and data interrelationships) to be stored, modified or deleted. Table IV-1 shows what is permitted when a user has read or write access to a data item, record type, set, or area. This assumes that the user has furnished the correct password.

Table IV-1. Read and Write Access

If a user has access to a	Read Access	Write Access
data item	occurrences of the data item can be retrieved	occurrences (values) of the data item can be modified
record type	occurrences of the record type can be retrieved	occurrences of the record type can be created, deleted, or modified
set	the owner record occurrences associated with (connected to), a member record occurrence can be found; the member records connected to an owner record occurrence can be found	owner record occurrences can be connected to and disconnected from occurrences of the set's member record type
area	record occurrences and data item occurrences in the area can be retrieved for those record types and data items to which the user has read access	record occurrences and data item occurrences in the area can be modified (created, deleted), for these record types and data items to which the user has write access

The MDBS technique for allowing the application designer to selectively restrict a user's access to various data items, record types, sets and areas. is based on an access code principle. In MDBS there are 16 access codes: a, b, c, ..., p. A combination of one or more access codes can be assigned to any data item to specify its read access. Another, (or the same) combination can be used to specify the data item's write access. For example. SKILCODE might have a read access of a, c, j and a write access of c-f (i.e., c,d,e and f). Notice that with 16 different access codes. there are 65535 different access assignments that could be made. A combination of one or more access codes can also be assigned to any record type, set, or area to specify its read access. Similarly, a write access combination can be specified for a record type. set, or area.

The following eight rules **must** be observed when assigning access codes:

1. A data item's read access codes must be a subset of the read access codes of its record type.

2. A data item's write access codes must be a subset of the write access codes of its record type.

3. A record type's read access codes must overlap (i.e., intersect with) the read access codes of each set in which that record type participates.

4. A record type's write access codes must overlap with the write access codes of each set in which that record type participates.

5. For record types having a calc key: All areas to which the record type is assigned must have one or more read access codes in common. Furthermore, the record type's read access codes must be a subset of these common access codes. (In other words, the record type's read access codes are a subset of the intersection of the areas' read access codes.)

6. Same as 5, except for write access.

7. For record types with no CALC key: The record type's read access codes must overlap with the read access codes of each area to which the record type is assigned.

8. Same as 7, except for write access.

9. If pointer indices are allowed in an area, then that area's write access codes must overlap with every user's write access codes.

The application developer cannot assign an access code to the main area of a data base. The main area can be accessed (both read and write) by any bona fide data base user.

An application designer can assign a combination of access codes to any user, to establish that user's read access authorization. Another (or the same) combination is used to establish that user's write authorization. If a user's read access codes overlap (i.e., intersect with) an area's read access codes, then the user has read access to that area (see Table IV-1). Suppose an area has b, g, m read access. Suppose user-1 has an a-c read authorization and user-2 has an h and p read authorization; then user-1 **does** have read access to the area (b overlaps, $\{b,g,m\} \cap \{a,b,c\} = \{b\}$). User-2 **does not** have read access to the area (no overlap, $\{b,g,m\} \cap \{h,p\} = \emptyset$). Similarly, if the access codes in a user's write authorization overlap the write access codes for an area, then the user has write access to that area.

If a user's read access codes overlap a set's read access codes, then the user has read access to the set. If a user's write access codes overlap a set's write access codes, then the user has write access to that set. For a user to have read access to occurrences of a record type, the user's read access codes must overlap both the record type's read access codes, and the read access codes of the area(s) containing the record occurrences. A user has write access to a record type's occurrences if the user's write access codes overlap with both the write access codes of the record type and the write access codes of the area(s) containing the record occurrences.

For a user to have read access to occurrences of a data item, the user's read access codes must overlap both the data item's read access codes and the read access codes of the area(s) containing the data item occurrences. A user has write access to a data item's occurrences if the user's write access codes overlap with both the write access codes of the data item and the write access codes of the area(s) containing the record occurrences.

No access code can be assigned to a data item, record type, set, or area that has not also been assigned to at least one user.

DDL Specification of Access Codes

An application designer can optionally specify a combination of access codes for any user, area, set, record type or data item appearing in a DDL source specification. This is accomplished by means of the optional access clause which can be

- appended to any user specification in the user section
- appended to any area specification in the area section
- included in any record type section
- included in any set section
- included in any data item section.

An access clause has the same form, regardless of which section it is used in. This form is given below.

An access code list, denoted by aclist, can be either a single access code or a parenthesized list of access codes and/or access code ranges. A single access code is simply one of the letters from a through p. An access code range is indicated by two single access codes (in alphabetic order) with a hyphen between them. For instance, the access code range e-i is a shorthand way of referring to the single access codes: e,f,g,h,i.

Access clause forms:

```
[READ ACCESS IS aclist-1] [WRITE ACCESS IS aclist-2]

ACCESS IS aclist-3
```

The read access codes (for a user, area, set, record type, or data item) are specified in aclist-1. The write access codes are specified in aclist-2. If the read and write access codes are identical, the second form of this clause (ACCESS IS aclist-3) can be used. The codes specified in aclist-3 will be used as both the read and the write access codes.

Defaults: i) for a user, the read and write access codes default to a
ii) for an area, the read and write access codes default to a

iii) for a record type, the read and write access codes default to a
iv) for a data item, the read and write access codes default to the read and write access codes (respectively) of that data item's record type
v) for a set, the read and write access codes default to a

If an application designer allows all access codes to default. then the eight rules for access code assignment are satisfied. If the designer allows only some (or none) of the access codes to default. he or she must be careful that the eight assignment rules are satisfied. Failure to observe these rules results in an error diagnostic when the DDL Analyzer is executed.

Examples:

```
READ ACCESS IS d WRITE ACCESS IS j
READ d WRITE J
READ (a,d,m-p) WRITE (a,f-g,e,i-k)
ACCESS IS (a-e)
ACCESS (a,c,d,m-n)
READ (a,K,P)
WRITE (A-P)
```


This page intentionally left blank.

V. DATA INTEGRITY

With MDBS, the application developer can use either (or both) of two broad approaches to providing data integrity. One involves the inclusion of integrity checking in those DML application programs that can modify data or data relationships. The other approach is to declare various integrity constraints in the DDL source specification; these are automatically enforced by MDBS during all DML and query processing.

Incorporating integrity checking into DML application programs gives an application developer very extensive and flexible control over data integrity. For instance, compatibility checking and combination checks are straightforward within a DML application program. The degree and extent of integrity control that can be achieved within DML application programs far exceeds what can be provided with a stand-alone, nonprocedural language. Nonprocedural languages are not at all well-suited for stating the detailed, complex integrity checks that are very often needed in devising a rigorous, professional application system.

MDBS automatically enforces certain kinds of data integrity on the basis of the DDL source specification. These include the prevention of unauthorized changes, fixed set retention, exclusion of duplicate CALC key values and duplicate sort key values, and limiting the values of a data item to a desired feasible range. Of course, the integrity of 1-to-1, 1-to-many, and many-to-many relationships is automatically enforced. The DDL source specification can also be used to define a transaction logging file. All alterations to a data base, since the last data base backup, are recorded on this file if the MDBS-RTL package has been acquired. In the event of a breach in data base integrity, all (or, if desired, some) of the logged transactions can automatically be reapplied to the data base backup copy to reestablish the data base's integrity.

A. Preventing Unauthorized Changes

The MDBS data security features presented in Chapter IV give an application developer extensive and highly selective automatic control over who can make which kinds of alterations to the data and data interrelationships. This guards against both accidental and intentional assaults on data integrity.

B. Fixed Set Retention

As discussed in prior chapters, the fixed set retention option allows an application developer to declare a fixed data interrelationship. Once owner and member record occurrences are connected for a fixed set, their interrelationships can never be altered. The integrity of data relationships that should never change is, therefore, automatically maintained.

C. Duplicate Key Exclusion

If it is important to prohibit the existence of two record occurrences with the same sort key value, this can be accomplished by indicating (in a sort-clause) that duplicates are not allowed. Duplicate calc key values can also be prohibited from the data base by indicating (in a CALC-key-clause) that duplicates are not allowed.

D. Invalid Dates and Times Prohibited

If a data item is declared to be of the type date, then invalid occurrences of that data item are automatically prohibited from the data base. For instance, 09/31/1957 is not a valid occurrence of a date data item. Any attempt to create such an occurrence is automatically prohibited. The maximum day allowed for a February date is 28, except in leap years. In leap years, the maximum day for a February date is 29. Not all years that are divisible by 4 are leap years. For instance, 1900 is not a leap year and such facts are known by MDBS. All leap years from the year 0000 to the year 9999 are handled by MDBS. The maximum day allowed for April, June, September or November is 30. The maximum day allowed for any other month is 31.

If a time data item is declared in a DDL specification, then invalid occurrences of that data item are automatically prohibited from the data base. For instance, 221:65:03 is not a valid occurrence of a time data item. Any attempt to create such an occurrence is automatically prohibited. The maximum number of seconds that can be used in a time occurrence is 59. The maximum number of minutes that can be used in a time occurrence is 59.

E. Feasibility Range Specification

An optional clause may be used in any data item section to specify the range of permissible values for the data item being defined. The one exception is that a range of feasible values cannot be specified for a data item whose type is declared to be binary. Whenever an attempt is made to create or modify an occurrence of a data item, MDBS checks the new data value to determine whether it is in the range of feasible values for that data item. If it is out of the feasible range, the creation or modification does not take place and an error message to that effect is issued.

The permissible lower and upper bounds for a data item's feasibility range depend on that data item's type and size. The minimum lower bound and the maximum upper bound for each type of data item (of size n) is shown in Table V-1. Notice that the maximum upper bound for date data items is 126 years above whatever lower bound is specified. Upper and lower bounds for character and string data items must be in quotes.

If a DDL source specification gives an upper bound that exceeds the maximum upper bound or is less than the minimum lower bound. then the DDL Analyzer will issue an error message. A DDL source specification that gives a lower bound. which is less than the minimum lower bound or greater than the maximum upper bound, will also result in an error when the DDL Analyzer is executed. The DDL Analyzer also issues an error if the stated lower bound exceeds the specified upper bound. Numeric ordering is used for numeric data items. The character code convention of the host operating system (e.g., ASCII) is used for string and character data item orderings. For string data, control characters are treated as blanks. For string and character data, some parity bits may be stripped away by MDBS. Chronological ordering is used for date and time data items.

Table V-1. Minimum Lower Bounds and Maximum Upper Bounds for Feasibility Ranges

Type	Size	Minimum Lower Bound	Maximum Upper Bound
integer	n	-2^{8n-1}	2^{8n-1}
real	n	$-2^{127}(1-256^{1-n})$	$2^{127}(1-256^{1-n})$
unsigned	n	0	256^{n-1}
idec	n even	$-10^{63}(1-10^{-n})$	$10^{63}(1-10^{-n})$
	n odd	$-10^{63}(1-10^{-n-1})$	$10^{63}(1-10^{-n-1})$
character	n	the lowest character in the host operating system's character code	a sequence of n characters, each of which is the highest character in the host operating systems's character code
string	n	the lowest character in the host operating system's character code	a sequence of n characters, each of which is the highest character in the host operating systems's character code
time	--	000:00:00	255:59:59
date	--	01/01/0000	126 years beyond the specified lower bound

Letting lbound indicate lower bound and ubound indicate upper bound. the optional clause for stating a data item's feasibility range is:

RANGE IS { lbound / LOWEST } TO { ubound / HIGHEST }

If LOWEST is declared rather than lbound. then the lower bound takes on a default value. If HIGHEST is specified rather than ubound, then the upper bound assumes a default value. Stating that RANGE IS LOWEST TO HIGHEST has the same logical effect as altogether omitting the range clause; defaults are used for both the upper and lower bounds.

Defaults: With the exception of date data items, the lower bound defaults to the minimum lower bound and the upper bound defaults to the maximum upper bound. The lower bound default for a date data item is 01/01/1900. The upper bound default for a date data item is 126 years beyond the lower bound.

Examples:
```
RANGE IS -273 TO 10000
RANGE 0.00 TO 4900.00
RANGE LOWEST TO 65:00:00
RANGE "BROWN" TO "BROWN"
RANGE IS "AARDVARK" "ZEBRA"
RANGE 01/01/1869 TO HIGHEST
RANGE 240 HIGHEST
RANGE "A" "ZZZZZZZZ"
RANGE "3" TO "am9"          (ASCII assumed)
RANGE "mmm" "AXY"           (ASCII assumed)
```

F. Data Base Recovery and Transaction Logging

The RTL form of MDBS supports additional data integrity mechanisms including page image posting. transaction logging and transaction recovery. These mechanisms are fully described in the MDBS RTL Manual.

VI. USING THE MDBS.DDL SOFTWARE

After designing a data base schema and specifying that schema in terms of the MDBS Data Description Language. an application developer can make use of the MDBS.DDL software. The MDBS.DDL program can be executed either interactively or in a batch basis. When executing MDBS.DDL interactively, a variety of commands are available. As explained in Chapter I (Figure I-5), these commands allow the MDBS.DDL user to enter, edit, and analyze a DDL source specification.

This chapter explains the interactive usage of MDBS.DDL, including a discussion of each interactive command. These commands are briefly summarized in Figure VI-1. Appendix J presents an overview of the control flow in using MDBS.DDL. Batch usage of the MDBS.DDL program is also described in this chapter. The procedure for invoking MDBS.DDL (either interactively or in batch) is operating system dependent and is therefore documented in the **MDBS System Specific Manuals.** When invoking MDBS.DDL, a user can optionally indicate. on the operating system command line, how many bytes are to be allocated for use by MDBS.DDL. The allocation is for the number of bytes beyond the MDBS.DDL program size. The allocation is made by using the -bnn argument on the operating system command line, where -b is a literal and nn is the number of bytes to be allocated. If -bnn is not used, then all available bytes are allocated to MDBS.DDL (regardless of whether or not MDBS.DDL needs all of these bytes).

When using Version 3a, a -v argument must be included on the command line if there is an intention of eventually processing the data base with the "obsolete" DML commands (see Appendix A of the **MDBS DMS Manual** for a description of these commands).

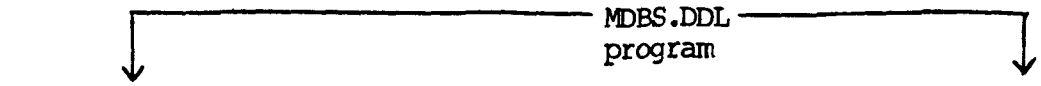
If MDBS.DDL is overlaid, all overlay files must be on the proper drive(s) as they are needed during the execution of MDBS.DDL. If they are not, a message to the effect that the DDL overlay file(s) is (are) not found is displayed. In this event, overlay files should be put onto the proper drives before the command is resubmitted. Again, these operating system dependent factors are discussed in the appropriate **MDBS System Specific Manuals.**

A. Interactive Usage of the MDBS.DDL Program

The **MDBS System Specific Manual** for your host operating system shows how to invoke the MDBS.DDL program so that it will interact with you. MDBS.DDL first of all responds with the MDBS copyright notice, and possibly your name and serial number. The user is then presented with the prompt symbol (::). This symbol always indicates that MDBS.DDL is ready to accept a line of input from you. The line that you enter will contain either an interactive command (see Figure VI-1) or DDL source text.

If the input line begins with from one to four digits (0-9999), then MDBS.DDL interprets the rest of the line as being a line in the

DDL source text. The one to four digit number indicates the relative position of that line in the DDL source text. The other kind of input line simply consists of a command for help, text handling, line editing, DDL analysis, or returning control to the operating system. Text lines and command lines can be inter-mixed as desired. Regardless of which type of line is being input, the standard conventions of the host operating system apply for such tasks as backspacing, restarting a line, etc. While system specific manuals cover more on this, the following general comments are usually applicable.



Interactive Commands

for help	h	description of all of the available interactive commands
for text entry (and deletion)	d	delete all text
	d nn	delete line number nn
	d nn mm	delete all line of text from nn through mm
	i	insert text on line 1
	i nn	insert text after line number nn
	l	list text
	l nn	list line number nn of the text
	l nn mm	list the text from line nn through mm
	n	renumber the text
	n nn	renumber the text so the first line is numbered nn
	n nn mm	renumber the text to begin with nn using increments of mm
	r	read DDL source text from a disk file
	r filename	read DDL source text from the named disk file
	w	write DDL source text to a disk file
	w filename	write DDL source text to the named disk file
for line editing	e	edit all lines of text
	e nn	edit line number nn
	e nn mm	edit lines of text from nn through mm
for global editing	c/ps/ns	change all occurrences of the present string ps to the new string ns
	c nn/ps/ns	change occurrences of ps in line nn to the new string ns
	c nn mm/ps/ns	change occurrences of ps to ns in lines nn through mm
	g/ps	get and display all lines containing an occurrence of the string ps
	g nn mm/ps	get and display all lines from nn through mm that contain ps
for DDL analysis	init	analyze DDL source text and initialize data base
	init filename	analyze DDL source text on the named disk file and initialize data base
for stopping	stop	stop the execution of MDBS.DDL and return control to operating system

Figure VI-1. Using MDBS.DDL Interactively

Special Keys. When entering a text line or command line, there are several keys that perform special functions. The key that must be pressed to perform a special function differs from one operating system to another. The special keys that exist within a given operating system are described in the pertinent system specific manual. The special functions performed by these keys are:

- a key to terminate an input line (e.g., RETURN)
- a key to interrupt the line entry and restart the input line (e.g., CONTROL-X)
- a key to cause a physical backspace and character deletion in the line being input (e.g., CONTROL-H)
- a key to cause a tab character to be placed in the line (e.g., CONTROL-I)
- a key to return control to the operating system (e.g., CONTROL-C)
- a key to cause the :: prompt to appear (e.g., ESCAPE)
- a key to toggle the DDL Analyzer output between the console and printer (e.g., CONTROL-P)

Input Line Length and Composition. The input buffer for MDBS.DDL is limited to 100 positions. Thus an input line that exceeds 100 characters (including blanks) cannot be entered. If the user attempts to enter more than 100 characters on a line, MDBS.DDL will not use the excess.

There are no strict rules for the placement of a command or DDL source text within an input line. MDBS.DDL ignores leading blanks and tabs. For instance, any of the following input lines will result in the "help" command being executed.

```
::h

::        h

::                  h
```

Any of the following input lines will produce the same DDL source text on line 5834.

```
::5834   ITEM   ADDRESS   STRING   40

::       5834   ITEM   ADDRESS   STRING   40

::                 5834   ITEM   ADDRESS   STRING   40
```

Commands can be entered in either upper or lower case. DDL source text can also be entered in either upper or lower case. When the DDL Analyzer is used to initialize a data base, information from the DDL source (for example, record type names, set names, etc.) is stored in the data dictionary as upper case. The exception is that characters enclosed in double quotes (a password, user name, filename, or title) are not translated into upper case within the data dictionary. Of course, actual data values stored in the data base via MDBS.DMS are maintained exactly as specified by the person who creates

or modifies the data values. The only exception is for character and string data items where parity bits may be stripped away and control characters are transformed into blanks.

B. The Interactive Modes of Operation

This section explains each mode of operation that is available when using MDBS.DDL interactively. The presentation is organized in accordance with Figure VI-1. Help, text handling (entry/deletion), line editing, DDL analysis, and stopping are discussed. The syntax and meaning of the commands pertinent to each of these categories is explained. If a command is not recognizable (due to misspelling, for instance), MDBS.DDL responds with the message:

Invalid DDL command

and the :: prompt is displayed.

1. Help. The help command provides an alphabetic listing of all other commands that can be used when the :: prompt is displayed. A short explanation of each command is furnished.

   Help Command Syntax: h or help

2. Text Entry (and deletion). A DDL source specification that is built through the MDBS.DDL text entry mode will consist of many source lines, each of which is preceded by a number in the range of 0 through 9999. MDBS.DDL maintains these DDL source lines in numeric order. A DDL source specification can also be built and maintained by a separate, stand-alone text editor that is available in the host operating system. In this case the source lines may not be numbered.

   During the execution of MDBS.DDL, only one DDL source specification can be operated on at any given moment. With the exception of the "read" command, all commands pertaining to text entry operate on this local DDL source specification. The "read" command takes the DDL source text existing on a disk file and makes it the new local DDL source specification. Line numbers are automatically added to this local copy. To save the prior local source text on a disk file, the "write" command would be used prior to the read command. Whenever MDBS.DDL is used, there is no local DDL source text until the user either gives a "read" command or begins entering (i.e., creating) text via the MDBS.DDL text entry mode.

   As noted in Figure VI, there are several commands that can be used in connection with text entry and deletion. As explained in Section VI-B, lines of DDL source text can also be generated without using any command at all. Both approaches are discussed here.

a. Text Insertion without a Command: A line of text to be inserted in the local source text is preceded by a number from 0 to 9999 and terminated by pressing the RETURN (or ENTER) key. Entered source lines are maintained in the order indicated by their line numbers. If a line of the same number already exists in the local text, the newly entered line will replace the existing line. If there is no local text at the time of a line insertion, the inserted line becomes the local source text. Valid examples of this approach to line insertion are:

```
10 LINE OF TEXT
20 MORE
0299 STILL MORE
0000 this is OK
8393so is this
9999 HIGHEST LINE NUMBER
```

Of course, these particular lines are not descriptive of a schema.

If too much text is entered (i.e., if there is insufficient memory to hold all of the text) the following message is displayed:

```
Insufficient room in memory
```

and the current line being entered is lost. However, all text entered up to this point is still preserved in the local source text. In the event of this problem, the user may choose to use the standard text editor available on his computer to create and maintain the DDL source specification; this is stored on a disk file. Such files can be processed by the "read" and "write" commands, and by the MDBS.DDL analyzer mode.

If the user desires to enter many lines in numeric sequence, the "insert" command can be used. When an "insert" command is given, MDBS.DDL automatically provides line number prompts, so that line numbers need not be keyed in. The "insert" command is described next.

b. Insert Command Syntax:

```
i       insert source lines beginning at the start of
        the local source text
i nn    insert source lines after line nn of the
        local source text
```

Explanation: This command results in line number prompts starting with line number nn (or, if that line already exists, starting with line number nn+1). Upon receiving a line number prompt, the user keys in the

desired line of DDL source text and presses the RETURN (or ENTER) key. MDBS.DDL will then issue a prompt for the next line nn+1 (or nn+2), and so forth. Lines of existing source text following nn are automatically "pushed down" and renumbered when conflicts arise. In order to stop the line number prompts, press the RETURN key when the next line number prompt appears. MDBS.DDL will respond with the :: prompt.

Just as described for line insertion without a command, the message:

Insufficient room in memory

can occur. This can be remedied as described earlier, through the use of a standard text editor.

Examples of Valid Usage:

I	Begins insertion at line 0; where line 0 already exists, the existing lines are pushed down with renumbering as needed.
I 2001	Begins insertions at line 2001 if that line does not already exist; insertions begin at 2002 (with pushdown renumbering) if that line already exists.

c. Delete Command Syntax:

d	deletes the entire local source text
d nn	deletes line nn from the local source text
d nn mm	deletes lines nn through mm from the local source text

Explanation: This command deletes the indicated line. When no line is specified, MDBS.DDL responds with the prompt of:

Delete all (y/n)

The entire text is deleted if a response of y is given. No deletion is made if the user response is n. Commas or blanks serve as delimiters between nn and mm. When the deletion has taken place or if there are no lines in the range to be deleted, MDBS.DDL gives the :: prompt.

Examples of Valid Usage:

```
d 10            Deletes line 10
d 977,1024      Deletes lines 977,978,...,1023,1024
d 30 40         Deletes lines 30,31,...,39,40
d 18 25         Deletes lines, 18,19,...,24,25
d
   Delete all?(y/n)   MDBS.DDL prompt
y          User responds with y to delete all lines.
```

d. List Command Syntax:

```
l          lists all lines in the local DDL source
           text
l nn       lists line nn of the local source text
l nn mm    lists lines nn through mm of the local
           source text
```

Explanation: This command lists the indicated lines of the local DDL source text. Commas or blanks can serve as the delimiter between nn and mm. During a listing, pressing any key (except CONTROL-C or ESCAPE) causes a pause until any key (except CONTROL-C or ESCAPE) is again pressed. The ESCAPE key terminates the listing process and MDBS.DDL responds with the :: prompt. When a listing has gone to completion or if there are no lines in the range to be listed, then MDBS.DDL gives the :: prompt.

Examples of Valid Usage:

```
l              Lists all lines of text
15834,6000     Lists lines 5834 through 6000
l 0            Lists line 0
l 10 20        Lists lines 10 through 20
122, 30        Lists lines 22 through 30
15             Lists line 5
```

e. Renumber Command Syntax:

```
n          renumbers all text giving the first line
           the number 0, with increments of 1
n nn       renumber all text giving the first line
           the number nn and incrementing by 1
n nn mm    renumber all text giving the first line
           the number nn and incrementing by mm
```


Explanation: This command renumbers all text using the starting point and increment indicated. The mnemonic ren can be used in place of n. Commas or blanks can serve as the delimiter between nn and mm. If an increment of zero is specified or if the parameters nn and mm are such that a line number greater than 9999 would result, then the message:

requested resequencing not possible!

is printed and the :: prompt is given. At the conclusion of the renumbering process, MDBS.DDL gives the :: prompt.

Examples of Valid Usage:

n	First line will be 0000, second 0001, etc.
ren	First line will be 0000, second 0001, etc.
n 30	First line will be 0030, second 0031, etc.
renl0, 10	First line will be 0010, second 0020, etc.
n 10 10	First line wil be 0010, second 0020, etc.
n0,3	First line will be 0000, second 0003, etc.

f. Read Command Syntax:

r	read DDL source text from disk file, making it the local text
r filename	read text from the file "filename", making it the local text

Explanation: This command allows a user to read a previously saved text file. This becomes the local source text, replacing all other source text that happened to be local before the command was made. The new local text is automatically numbered and can be operated on by any of the MDBS.DDL commands.

If the first form of the "read" command is used, then MDBS.DDL will prompt the user for a file name. Pressing the RETURN (ENTER) key, without having given a file name, will result in the :: prompt. The file name that a user specifies with the "read" command must be a fully qualified file name within the host operating system. When MDBS.DDL has successfully completed a read, the message

xxxx bytes

is displayed, where xxxx indicates the total number of source text bytes that were transferred into central memory. The :: prompt then appears.

If not enough memory is available for the text to be read-in, then the message

Insufficient room in memory

is displayed and the :: prompt is given. The text in memory before the "read" command will probably be lost. A data base can, nevertheless, still be initialized for the oversized DDL source specification by using the "init filename" command described in Section B-4 of this Chapter.

There are other causes of read errors such as bad disk sectors, hardware problems, failure to give a fully qualified file name, absence of the indicated file on the indicated disk drive, etc. In cases such as these, the message

Error in reading file

is displayed (in addition to any messages the operating system may print) and the :: prompt is given. If a file is not successfully read, the local text present in memory may still be intact, depending on the nature of the read error.

Examples of Valid Usage:

r	Give the "read" command without a file name
filename::	MDBS.DDL asks for a file name
DEPT.DDL	User responds with fully qualified file name such as DEPT.DDL
463 bytes	MDBS.DDL responds with the bytes of text read
r DEPT.DDL	Read the text from the file DEPT.DDL
463 bytes	MDBS.DDL responds with the bytes of text read

g. Write Command Syntax:

w	write the local DDL source text onto a disk file
w filename	write the local DDL source text onto the "filename" disk file

Explanation: This command allows a user to save an image of the local source text on a disk file. Line numbers of local text are stripped away during the

write operation. After the write has occurred, the text is still local as well. If the first form of the "write" command is used, then MDBS.DDL will prompt the user for a file name. Pressing the RETURN (ENTER) key, without having given a file name, will result in the :: prompt. The file name that a user specifies with the "write" command must be a fully qualified file name within the host operating system. When MDBS has successfully completed a write, the message

xxxx bytes

is displayed, where xxxx indicates the total number of source text bytes that were transferred to the disk file. The :: prompt then appears.

Typical causes of write errors include hardware malfunctions, failure to use a fully qualified filename, absence of the indicated file on the indicated disk drive. In cases such as these, the message

error

is displayed (in addition to any messages the operating system may print) and the :: prompt is given.

Examples of Valid Usage:

w	Give the "write" command without a file name
filename::	MDBS.DDL asks for a file name
DEPT.DDL	User responds with fully qualified file name such as DEPT.DDL
463 bytes	MDBS.DDL responds with the bytes of text written
w DEPT.DDL	Write an image of the local text onto the file DEPT.DDL
463 bytes	MDBS.DDL responds with the bytes of text written

3. Line Editing. In the line editing mode, alterations within any line of the local DDL source text can be made without retyping the entire line. The line editor is entered through the "edit" command (recall Figure VI-1).

a. Edit Command Syntax:

e	edit the entire local DDL source text
e nn	edit the local text in line nn
e nn mm	edit the local text from line nn through mm

Explanation: This command is used to enter the line editing mode. Commas or blanks can be used as the delimiter between nn and mm. If there is no text within the range of lines to be edited, MDBS.DDL returns the :: prompt. If there is text within the range of lines to be edited, the first such line is displayed and the user is given the editor prompt of]: . At this point, a user has the options of advancing to the next line of text, changing the current line of text, or exiting from the line editing mode. A given line can be repeatedly altered before advancing to the next line. Within the line edit mode, the "change" command is used to change text and the "stop" command is used to exit from the line edit mode.

If an improper command syntax is entered while in the line editor, a prompt of ? is given. When this occurs, the user is still in the line editor mode and can give a correct response.

Examples of Valid Usage:

e	Edit all text starting with the first line
e24	Edit line 24
e 1000,1026	Edit all text from lines 1000 through 1026
e 10 15	Edit all text from lines 10 through 15

b. Moving Through Text: The user can progress from the current line of text to the next line of text by pressing the RETURN (ENTER) key in response to the editor prompt of]: or ?. This results in the next line of text being displayed, followed by the]: prompt. If there is no more text or if the next line is beyond the range specified when entering the line editor (beyond mm), then MDBS.DDL exits from the line edit mode and displays the :: prompt.

c. Change Command Syntax:

cdPRESENTdNEWd	where d is any delimiter, change the PRESENT string within the current line to the NEW replacement string.

Explanation: This command replaces a specified string of characters that presently exists in a line, with a new specified string of characters. The first character following the c is used as the delimiter. The final delimiter following the new string can be omitted. If the specified present string is not found,

the editor responds with a ? prompt. When a change has been made, the line editor will display the revised line and will display the prompt]:.

Examples of Valid Usage:

Assume the following is being edited:

```
00010 HTIS IS TWO DEMANSTRATE THE LINE LINE EDTOR
```

c/HT/TH results in:

```
0010 THIS IS TWO DEMANSTRATE THE LINE LINE EDTOR
```

c.TWO.TO results in:

```
0010  THIS IS TO DEMANSTRATE THE LINE LINE EDTOR
```

cQMAQMO results in:

```
0010  THIS IS TO DEMONSTRATE THE LINE LINE EDTOR
```

c. LINE. results in:

```
0010  THIS IS TO DEMONSTRATE THE LINE EDTOR
```

c/ED/EDI results in:

```
0010  THIS IS TO DEMONSTRATE THE LINE EDITOR
```

d. Repeated Changes: A user can make the same change repeatedly within a line and across lines. When a change is entered, it is remembered. The next change replaces the remembered change and becomes the new remembered change. Upon entering the line editing mode, the remembered change is

```
C///
```

A remembered change will not be forgotten until another properly entered change is submitted or until the user exits from the line editing mode.

Command Syntax:

```
C     makes the remembered change
```

Explanation: Entering a C followed by a RETURN (or ENTER) invokes the last remembered change. This can be done repeatedly within a line or, after advancing lines, within another line.

Examples of Valid Usage:

Suppose the following local text is to be edited:

```
0010 PRODUCT 64KGB IS NOT NEW
0020 64KGB AND 64KGB DERIVATIVES
0030 HAVE BEEN AROUND FOREVER
0040 ...64KGB REPORT
```

and the user enters e in response to the :: prompt. This results in:

```
0010 PRODUCT 64KGB IS NOT NEW
```

being displayed. At this point the remembered change is:

C///

Suppose 64KGB is to be replaced by T47S. The user may enter:

C.64KGB.T47S

which will result in:

0010 PRODUCT T47S IS NOT NEW

and the current remembered change is:

C.64KGB.T47S

Pressing the RETURN key will result in:

0020 64KGB AND 64KGB DERIVATIVES

Pressing C and then RETURN results in:

0020 T47S AND 64KGB DERIVATIVES

since C.64KGB.T47S was remembered. Repeating with C and RETURN gives:

0020 T47S AND T47S DERIVATIVES

Pressing RETURN gives:

0030 HAVE BEEN AROUND FOREVER.

Pressing RETURN gives:

0040 ... 64KGB REPORT.

Entering C and RETURN gives:

0040 ... T47S REPORT.

e. Stop Editing Command Syntax:

s stop line editing

Explanation: This command is used to exit from the line editor mode. MDBS.DDL returns the :: prompt.

4. Global Editing. In addition to line-oriented editing, it is possible to make more global changes to a DDL source specification. Two related commands are provided for this purpose. One will get and display all lines containing a specified string of characters. The other actually changes all instances of a specified character string to a new character string.

a. Get and Display Command Syntax:

g/ps	get and display all lines containing an occurrence of the indicated string ps
g nn mm/ps	get and display all lines from nn to mm that contain ps

Explanation: This command is typically used to preview all lines containing a particular character string prior to making a global change for that string. If no lines are specified in the command, then all DDL source lines containing the string are displayed. If a range of lines is specified, then the get and display operation is restricted to the lines within that range. Delimiters other than the / symbol are permissible.

Examples of Valid Usage:

g/retention	displays all lines containing "retention"
g 25 100/RETENTION	displays all lines in the range 25 through 100 that contain "RETENTION"
g.nodup	displays all lines containing "nodup"
g 16,83.FIFO	displays all lines in the range 16 through 83 that contain "FIFO"

b. Change Command Syntax:

c/ps/ns	change all occurrences of the present string ps to the new string ns
c nn/ps/ns	change occurrences of ps in line nn to the new string ns
c nn mm/ps/ns	change occurrences of ps in lines nn through mm to ns

Explanation: This command can be used to change many instances of a particular character string in a single operation. The scope of this change can be a single line, a range of lines, or the entire DDL source specification. Delimiters other than the / symbol are permissible.

Examples of Valid Usage:

c/EMPLOYEE/emp	changes all instances of "EMPLOYEE" to "emp"
c/nodup/	changes each instance of "nodup" to a blank
c 75 90/FIFO/LIFO	changes all instances of "FIFO" in lines 75 through 90 to "LIFO"
c 1,23/STR 10/STR 12	changes all "STR 10" string in lines 1 through 23 to "STR 12"

5. DDL Analysis. This command is used to enter the DDL analysis mode. A successful analysis gives the user the option of initializing a data base. MDBS.DDL accepts **ddl** in place of **init.** The first form of this command uses the local DDL source specification. The other form uses the DDL source specification residing on the named file (fully qualified).

Initialization Command Syntax:

init	Analyze and initialize the local DDL source specification.
init filename	Analyze and initialize the DDL source specification on the indicated disk file.

Explanation: The DDL analysis mode is used to check the syntactic and logical consistency of a DDL source text specification. If the DDL Analyzer detects an error in the DDL source text, the analysis halts and a message is displayed indicating the nature of that error. A full explanation of each possible error message is given in Chapter VII. The explanations include suggestions for correcting the error.

When DDL processing halts, due to an error, the :: prompt will appear. The error can then be corrected and the user can re-enter the DDL analysis mode. If no error is detected, MDBS.DDL responds with the messages:

***** Schema description successfully analyzed *****

and

Do you wish to initialize the data base? (y/n/s/o)

If the user replies with n, then the data base is not initialized and the :: prompt is given. If the user replies with y, then the data base is initialized. This means that MDBS.DDL stores the data dictionary, initializes the system pages in each area, and initializes all non-system pages in

the data base area(s). Prior to processing each area, Version 3a of MDBS.DDL prompts the user to ready the disk which will hold that area (the disk must be on-line on the proper drive). When the disk has been readied, the user presses the RETURN (or ENTER) key, which permits MDBS.DDL to perform the initialization for that area. After initialization, area statistics are displayed to the user. If a transaction log file has been defined in the DDL source specification, the Version 3a user is also prompted to ready the disk that is to hold the transaction log file. When initialization of all areas and the log file (if any) is complete, the :: prompt is given.

If the user desires to initialize only a short form of the data base, s is pressed rather than y. This has the same effect as y, except none of the non-system pages are initialized. This allows the physical file of each area to be very short. The optional DMU module can later be used to initialize the non-system pages, thereby extending the size of the physical file for each area (up to the maximum number of pages in the area). This is a very important facility for OEM application developers. It allows an OEMer to send an end user a brief version of the data base (e.g., on a floppy disk). The end user can then invoke a DMU command in order to expand the data base to its full potential size (e.g., on hard disk), by initializing all data base pages. A page must be initialized before data can be loaded into it.

The **o** option also initializes a short form of the data base. This option has the added effect of disabling the DISPLAY command. As described in the **MDBS QRS Manual**, QRS users can invoke the DISPLAY command to query the data dictionary. The o option allows an OEMer to prevent the display of a schema developed for an application system.

Area statistics are displayed for both the regular form and short forms of initialization. For each area, these statistics include the page size, the number of pages allocated, the number of system pages, and the remaining bytes. The minimum single user DMS buffer region size is also displayed. This size (in bytes) is exclusive of any file control or data blocks that may be needed by a host language program. The buffer region size for a host language program is allocated with the SETPBF command as described in the **MDBS DMS Manual.**

Examples of Valid Usage:

```
init
ddl  DEPT.DDL
init DEPT.DDL
```


6. Stop. The stop command terminates the interactive use of MDBS.DDL. Control is returned to the operating system. The local copy of text is destroyed.

   Stop Command Syntax: stop (or bye or end or quit)

C. Batch Usage of MDBS.DDL

The MDBS.DDL program can be used on a batch basis by invoking it from the operating system, with a file name specified as a parameter on the operating system command line. The indicated file should contain a valid DDL source specification. The contents of the file are analyzed and the data base is fully initialized without any user interaction. Upon completion of the analysis and initialization, control passes back to the operating system. In the event of an unsuccessful analysis or initialization, the operating system is informed of an abnormal termination.

The optional byte allocation (-b) and obsolete DML vintage (-v) arguments can be used in this batch mode, just as they are used in interactive mode. In addition, several other optional arguments are available. These arguments and the -b or -v arguments can appear in any order on the command line. The additional arguments are:

-q suppress the page initialization message during data base initialization

-h (for version 3a only) suppress the "disk-ready" prompts for area initializations

-s initialize a short form of the data base

-o initialize an "OEMer" form of the data base

-n no initialization is performed

Of the last three options only one can be used on a command line.

This page intentionally left blank.

VII. DDL ANALYZER MESSAGES

This chapter describes the error messages that can result from the DDL Analyzer. These errors arise out of improper syntax or errors in the logical structure as specified in the DDL description. In each case, an explanation of what the error message means is given. The possible situations under which the message would be obtained, together with possible solutions to the problem, are also described.

The error messages described here do not include errors that result from improper usage of the MDBS.DDL program. These have been discussed in the preceding chapter. Most of the error messages described here can occur in both Version 3a and Version 3c. Where errors occur exclusively with one of the versions, these have been appropriately indicated.

***** A CALC key item has not been defined.**

Explanation:
All items forming part of the CALC key for a record type must be defined as data items for that record type.
Possible Causes and/or Solutions:
1. One or more data items forming part of the CALC key for a record type have not been defined.
2. Check data item specifications in RECORD and ITEM sections.

***** Access level not in the range a to p.**

Explanation:
The access level defined for the construct (item, record, set or area) is not in the range 'a' to 'p'.
Possible Causes and/or Solutions:
1. The access level for any item, record, set or area must be in the range 'a' to 'p'.
2. Check item, record, set and area access level descriptions.

***** An item in the sort clause is not in a record.**

Explanation:
An item declared in the sort clause has not been defined in the record type.
Possible Causes and/or Solutions:
1. A set has been declared sorted by an item which has not been defined earlier in the owner/member record type.
2. Check item definitions in owner/member record types.

***** Area and data base page sizes must be equal.**

Explanation:
The page size defined for the data base and area(s) must be equal. This error occurs only in Version 3a.
Possible Causes and/or Solutions:
1. Check page size definitions in the IDENTIFICATION and AREA sections.

***** Area has pointer indices, so all users must have write access.**

Explanation:
All users must have write access to an area that allows pointer indices.
Possible Causes and/or Solutions:
1. Check write access codes of area and users.
2. User the POINTERS NOT ALLOWED clause in the area definition.

***** Area not previously defined.**

Explanation:
The area has not been defined earlier.
Possible Cause and/or Solutions:
1. Check the main area definition in IDENTIFICATION section, and subsequent area specifications in the AREA section.

***** Area page exceeds data base page size.**

Explanation:
The page size for one or more areas is greater than the page size defined for the main data base area. This error occurs only in Version 3c.
Possible Causes and/or Solutions:
1. The page size for any of the areas must be less than or equal to the page size defined for the main data base area.
2. Check page size definitions for the various areas.

***** AREA or RECORD section missing or misplaced.**

Explanation:
The system was expecting an AREA or RECORD section based on an end-of-section descriptor.
Possible Causes and/or Solutions:
1. Check end-of-section descriptors (e.g., a period) in the AREA and RECORD sections.
2. Check AREA and RECORD section specifications.

***** A sort-key item is not encrypted across participating records.**

Explanation:
For sets with multiple owner/member record types, the sort-key item is not encrypted across participating owner/member record types; i.e., the item has been defined to be encrypted in one or more owner/member record types, and not encrypted in one or more owner/member record types.

Possible Causes and/or Solutions:
1. For sets sorted by their multiple owner/member record types, the sort key item, if defined to be encrypted in any one owner/member record type, must be defined as encrypted in all participating owner/member record types.
2. Check sort-key item description in participating owner/member record types.

***** A sort-key item length not constant across participating records.**

Explanation:
For sets with multiple owner/member record types, the length of the sort-key item is not constant across participating owner/member record types. This error occurs only for sort-key items which have been declared as string items.

Possible Causes and/or Solutions:
1. For sets sorted by their multiple owner/member record types, the length of the sort key item must be the same in the participating owner/member record types.
2. Check sort-key item description in participating owner/member record types.

***** A sort-key item replication count not equal across participating records.**

Explanation:
For sets with multiple owner/member record types, the sort-key item replication count is not equal across participating owner/member record types.

Possible Causes and/or Solutions:
1. For sets sorted by their multiple owner/member record types, the replication count for the repeating sort-key item must be the same in the participating owner/member record types.
2. Check sort-key item description in participating owner/member record types.

***** A sort-key item type not equal across participating records.**

Explanation:
For sets with multiple owner/member record types, the sort key item type is not comparable across participating owner/member record types.
Possible Causes and/or Solutions:
1. For sets sorted by their multiple owner/member record types, the sort key item type must be the same in the participating owner/member record types (e.g., CHAR, REAL).
2. Check sort-key item description in participating owner/member record types.

***** At least one owner is required in a set.**

Explanation:
A set description requires at least one owner record type.
Possible Causes and/or Solutions:
1. Check 'owner' clause description of SET section.

***** At least one member is required in a set.**

Explanation:
A set description requires at least one member record type.
Possible Causes and/or Solutions:
1. Check 'member' clause description of SET section.

***** Bad character in number field.**

Explanation:
A number field in a RANGE specification has a bad character.
Possible Causes and/or Solutions:
1. Check RANGE specification for this item.

***** Binary items cannot be ranged.**

Explanation:
A range cannot be specified for binary items.
Possible Causes and/or Solutions:
1. Check binary item descriptions in ITEM section.

***** Binary items cannot be part of a sort-key.**

Explanation:
A sort-key item cannot include a binary item.
Possible Causes and/or Solutions:
1. Check item descriptions of sort-key items.

***** Cannot open file.**

Explanation:
An error has occurred while trying to open a file for initializing the data base.
Possible Causes and/or Solutions:
1. Disk on which file is to be opened may be full.
2. Bad sector on disk.

***** Data base IDENTIFICATION section missing or out of order.**

Explanation:
Self-explanatory.
Possible Causes and/or Solutions:
1. Check sequence of different sections in DDL description.

***** Data base must have at least one record type.**

Explanation:
Self-explanatory.
Possible Causes and/or Solutions:
1. The DDL specification contains no record type declaration.
2. The record type declarations are not in the proper position (e.g., they follow the set declarations).

***** Duplicate area name.**

Explanation:
The same area name has been defined twice.
Possible Causes and/or Solutions:
1. Check area descriptions.

***** Duplicate area specified in area list.**

Explanation:
An area name has been specified twice in the area list of a record description.
Possible Causes and/or Solutions:
1. Check area specifications in the RECORD section.

***** Duplicate clause has an unrecognizable descriptor.**

Explanation:
The duplicate clause in the RECORD section does not have a valid descriptor.
Possible Causes and/or Solutions:
1. The duplicate clause in the RECORD section does not have any of the following descriptors: NOT ALLOWED, LIFO, FIFO, IMMAT.
2. Check duplicate clause section in RECORD section.

***** Duplicate item within this record.**

Explanation:
A record type has a duplicate item specification.
Possible Causes and/or Solutions:
1. Check item specifications for different record types.

***** Duplicate member record types for this set.**

Explanation:
A set has the same record type specified twice as a member.
Possible Causes and/or Solutions:
1. Check member clause specification in SET section.

***** Duplicate owner record types for this set.**

Explanation:
A set has the same record type specified twice as an owner.
Possible Causes and/or Solutions:
1. Check owner clause specification in SET section.

***** Duplicate user.**

Explanation:
A user has been defined twice with different (or same) passwords.
Possible Causes and/or Solutions:
1. Check user-password specifications in USER section.

***** Duplicate record type name.**

Explanation:
The same record type has been specified twice.
Possible Causes and/or Solutions:
1. Check record type names in RECORD section.

***** Duplicate set name.**

Explanation:
The same set name has been specified twice.
Possible Causes and/or Solutions:
1. Check set name specifications in SET section.

***** Excessive date or time field.**

Explanation:
The range specified for the date or time field is excessive.
Possible Causes and/or Solutions:
1. Check possible ranges for date and time fields.
2. Check range specifications for date and time fields.

***** Excessive width for sort-key.**

Explanation:
The width specified for a sort-key exceeds the size of the sort-key's fields.
Possible Causes and/or Solutions:
1. Add the sizes of the sort-key's fields to determine the maximum sort-key width.

***** Expecting a number.**

Explanation:
A character other than a number was specified where the system was expecting a number.
Possible Causes and/or Solutions:
1. Check required specification of construct.

***** Expecting a quoted file name.**

Explanation:
A file name was not specified in quotes.
Possible Causes and/or Solutions:
1. Any file name in the DDL description should be enclosed in quotes.
2. Check file name specifications in IDENTIFICATION and AREA sections.

***** Expecting SET or END statement.**

Explanation:
The system was expecting a SET or END statement based on an end-of-section descriptor.
Possible Causes and/or Solutions:
1. Check end-of-section descriptors (e.g., a period) in the SET section.
2. Check SET and END sections.

***** Expecting 'RECORD-TYPE' statement in sort-clause.**

Explanation:
The system was expecting a 'RECORD-TYPE' statement in the sort-clause of the SET section.
Possible Causes and/or Solutions:
1. Check sort clause specification of SET section.

***** File not found.**

Explanation:
Only for systems having the DDL source specification file overlayed. The overlays are not on the currently logged-on drive.
Possible Causes and/or Solutions:
1. For systems with the DDL file overlayed, the overlays must be on the currently logged-on drive.
2. Check files on currently logged-on drive to make sure DDL overlays exist on this drive.

***** High date must not precede low date.**

Explanation:
In the range specification for a date item type, the high date must be specified after the low date.

Possible Causes and/or Solutions:
1. The high date has been specified before the low date in the range specification for a date item type.
2. Check range specifications in date item type.

***** Inconsistent insertion for "set-name"**

Explanation:
A set's insertion mode(s) must be consistent with set type and record location.

Possible Causes and/or Solutions:
1. Both owner and member insertion are automatic.
2. Member (owner) insertion is automatic with member (owner) records being placed in the area of the owner (member).
3. Members are declared to be clustered in an owner area and vice versa; or, clustering is requested for a recursive set.

***** Inconsistent item and record read access codes.**

Explanation:
The read access codes defined for any item must be a subset of the read access codes for the record type containing that item.

Possible Causes and/or Solutions:
1. The read access codes defined for the item and the record type containing that item have no elements in common.
2. Check item and record type read access code specification.

***** Inconsistent item and record write access codes.**

Explanation:
The write access codes defined for any item must be a subset of the write access codes for the record type containing that item.

Possible Causes and/or Solutions:
1. The write access codes for the item and the record type containing that item have no elements in common.
2. Check item and record type write access code specifications.

***** Inconsistent set and record read access codes.**

Explanation:
The read access codes defined for the record type and the set having the given record type as owner/member must have at least one element in common.

Possible Causes and/or Solutions:
1. The read access codes for the record type and the set having that record type as owner/member have no elements in common.
2. Check record type and set read access code specifications.

***** Inconsistent set and record write access codes.**

Explanation:
The write access codes defined for the record type and the set having the given record type as owner/member must have at least one element in common.
Possible Causes and/or Solutions:
1. The write access codes for the record type and the set having that record type as owner/member have no elements in common.
2. Check record type and set write access code specifications.

***** Insufficient room for DDL processing.**

Explanation:
There is insufficient room in memory for building tables during DDL processing.
Possible Causes and/or Solutions:
1. There is not enough room in memory for the DDL analyzer to build tables.
2. If the file has been read into memory using the 'R' command, the 'INIT' command may instead be used to initialize the data base without reading the file into memory.

***** Insufficient room in memory.**

Explanation:
There is not enough room in memory for any further processing, (i.e., to read a text file, to input a line of DDL source specification, etc.).
Possible Causes and/or Solutions:
1. Not enough room is available to read a file into memory.
2. Not enough room is available to input a line of DDL text.
3. The 'INIT' command may be used instead of a 'R' command to initialize a data base without reading the file into memory.
4. A text editor may be used instead of the DDL to input DDL source specification.

***** Integers may not have fractional parts.**

Explanation:
The range specifications for an item defined as an integer have been declared incorrectly.
Possible Causes and/or Solutions:
1. An integer item has a fractional part in its range specifications.
2. Check range specifications of item.

***** Invalid date or time delimiter.**

Explanation:
An invalid delimiter has been specified in the range clause for a date or time field.
Possible Causes and/or Solutions:
1. A character other than a '/' (for a date field), or other than a ':' (for a time field) has been specified as a delimiter on the range clause for these items.
2. Check range clause specifications for date and time data items.

***** Item length must be at least 1.**

Explanation:
The minimum size (number of bytes) for a data item, except a date or time item, is 1.
Possible Causes and/or Solutions:
1. The size specified for any item (except date and time data items) must be at least 1.
2. Check size specifications in data item section.

***** Item or set section missing or misplaced.**

Explanation:
The system was expecting an ITEM or SET statement based on an end-of-section descriptor.
Possible Causes and/or Solutions:
1. Check end-of-section descriptors (e.g., a period) in the ITEM and SET sections.
2. Check ITEM and SET section specifications.

***** Item access codes must be a subset of the record.**

Explanation:
The access codes defined for any item must be a subset of the access codes for the record type containing that item.
Possible Causes and/or Solutions:
1. An access code has been defined for the item which is not defined as an access code for the record type containing that item.
2. Check item and record type access code specifications.

***** Item length too long.**

Explanation:
The length of the data item is larger than the maximum allowable for that data item type.
Possible Causes and/or Solutions:
1. The item size (number of bytes) is longer than the maximum permissible for that data item type.
2. Check maximum allowable lengths (sizes) for different data item types.
3. Check item size specifications.

***** Item type was not specified.**

Explanation:
The type of the data item (e.g., CHAR,BIN) was not specified in the DDL source specification.
Possible Causes and/or Solutions:
1. The data item specification did not include the type of the data item (e.g., CHAR,BIN,IDEC).
2. Check data item specifications.

***** Maximum of 7 decimal positions for IDEC fields.**

Explanation:
For IDEC fields, the number of digits to the right of the decimal point cannot be greater than 7.
Possible Causes and/or Solutions:
1. A number greater than 7 has been specified for the number of digits to the right of the decimal position in the size specification for an IDEC item.
2. Check item size specification of IDEC items.

***** Missing set with record clustering for "record-type-name".**

Explanation:
The set specified in the record clustering clause for the indicated record type is missing from the SET section of the DDL specification.
Possible Causes and/or Solutions:
1. The set specified in the clustering clause has not been defined in the SET section.
2. Typographical error in specifying set name.
3. Check SET section and clustering specifications in RECORD TYPE section.

***** Misspecified access code range.**

Explanation:
The range for the access code has not been specified correctly. (e.g., f-a, etc.)
Possible Causes and/or Solutions:
1. An error has been made while specifying the range of access codes for the construct.
2. Check access code specifications.

***** Multiple owner/member declaration incorrect.**

Explanation:
The specification of multiple owners/members for a set is incorrect.
Possible Causes and/or Solutions:
1. While specifying multiple owners/members for a set, the format 'OWNER/MEMBER IS' rather than 'OWNERS/MEMBERS ARE' has been used.
2. Check owner/member specifications.

***** Need at least one ascending/descending clause.**

Explanation:
For a set declared to be sorted, at least one ascending/descending clause has to be specified in the sort clause.
Possible Causes and/or Solutions:
1. A set has been declared to be sorted, and no ascending/descending clause has been specified as part of the sort clause in the SET section.
2. Check sort clause specifications in SET section.

***** No item list in key-clause.**

Explanation:
The calc key clause (record location clause) in the RECORD section has no item list specified.
Possible Causes and/or Solutions:
1. No items have been specified in the calc key clause (or record location clause) in the RECORD section of the DDL specifications.
2. Check record location clause in RECORD section.

***** Not enough pages allocated.**

Explanation:
Not enough pages have been allocated for the main area and/or other area(s) to initialize the data base.
Possible Causes and/or Solutions:
1. The number of pages allocated for the main area and/or other area(s) are not sufficient to hold the system tables.
2. Increase the number of pages allocated for the main area and/or other area(s).

***** No user can access construct.**

Explanation:
An access code specification has been set up such that the construct (item, record, set or area) cannot be accessed by any user.
Possible Causes and/or Solutions:
1. The access code specifications for this construct do not overlap (have no code in common) with the access codes of any of the users. No user can, therefore, access this construct.
2. Check access code specifications for this construct and user access code specifications.

***** Numerical overflow.**

Explanation:
This error arises from improper range specifications for a data item type. Self-explanatory.
Possible Causes and/or Solutions:
1. The range specification for a data item type has caused a numerical overflow.
2. Check range specifications in ITEM section.

***** Numerical underflow.**

Explanation:
This error arises from improper range specifications for a data item type. Self-explanatory.
Possible Causes and/or Solutions:
1. The range specification for a data item type has caused a numerical underflow.
2. Check range specifications in ITEM section.

***** Only one title allowed per construct.**

Explanation:
Any area, item, record type, or set can have at most, one title specification.
Possible Causes and/or Solutions:
1. More than one title has been specified for a construct (area, item, record type or set).
2. Check title clause specifications for the construct.

***** Page size must be a multiple of 256 bytes.**

Explanation:
This error arises only in Version 3a. The page size (for the main area and area(s)) must be a multiple of 256 bytes.
Possible Causes and/or Solutions:
1. The page size defined for the main area and/or other area(s) is not a multiple of 256 bytes.
2. Check page size specification in IDENTIFICATION and AREA sections.

***** Page size must be at least 256 bytes.**

Explanation:
A page size must be at least 256 bytes.
Possible Causes and/or Solutions:
1. The page size defined for an area is less than 256 bytes.
2. Check page sizes specification in IDENTIFICATION and AREA sections.

***** Reals must be at least 2 bytes long.**

Explanation:
The minimum item size specification for a REAL data item is 2 bytes.
Possible Causes and/or Solutions:
1. The item size specification for a data item defined as REAL is less than 2 bytes.
2. Check item size specifications.

***** Record and area access codes inconsistent.**

Explanation:
This error may arise in either of the following two cases:
1. For record types having a CALC key, the record type's access codes must be a subset of the intersection of the access codes for all areas to which the record type is assigned.
2. For record types not having a CALC key, the record type's access codes must overlap with the access codes for each area to which the record type is assigned.

Possible Causes and/or Solutions:
1. For record types having a CALC key, an access code has been defined for the record type which is not common to all areas to which the record type is assigned.
2. For record types not having a CALC key, an access code has been defined for the record type which does not overlap with the access codes for one or more of the areas to which the record type is assigned.
3. Check access code specifications for the record type and all areas to which the record type has been assigned.

***** Record name not present with clustering for "set-name"**

Explanation:
The record type specified does not properly participate in the SET name indicated in the clustering clause.

Possible Causes and/or Solutions:
1. A record type having a clustering clause has not been defined as the owner/member of the set indicated in the clustering clause.
2. Typographical error in record type specification.
3. Check owner/member specifications and record type specification in clustering declaration.

***** Record too big -- increase page size.**

Explanation:
The page size specified must be at least as large as the length of the largest record type defined in the data base.

Possible Causes and/or Solutions:
1. The page size is less than the length of the largest record type defined in the data base.
2. For the same length of the above record type, the page size must be increased.

***** Record type not previously defined**

Explanation:
A record type which has not been defined earlier has been used in the specification of a section of DDL text.
Possible Causes and/or Solutions:
1. A record type used in a particular section of DDL text has not been defined earlier in the RECORD section.
2. Typographical error in record type specification.
3. Check record type specification and subsequent usage.

***** Replication already specified.**

Explanation:
The replication factor in the 'occurs' clause of the ITEM section has already been specified.
Possible Causes and/or Solutions:
1. The 'occurs' clause of the ITEM section has been specified twice.
2. Check 'occurs' clause in ITEM section.

***** Requested resequencing not possible using REN.**

Explanation:
An improper numbering sequence has been specified, or the numbering sequence specified extends beyond line 9999.
Possible Causes and/or Solutions:
1. The numbering sequence specified causes the file to extend beyond line 9999.
2. A negative increment (decrement) has been specified using the REN command.
3. Respecify REN command.

***** SYSTEM cannot be a member of a set.**

Explanation:
The SYSTEM record type can only be declared as an owner, and not as a member of any set.
Possible Causes and/or Solutions:
1. SYSTEM has been declared as a member of some set.
2. Check owner/member specifications.

***** SYSTEM cannot own a multiple owner set.**

Explanation:
A set with multiple owner record types cannot have SYSTEM as an owner.
Possible Causes and/or Solutions:
1. Check the set's owner clause.

***** Text had to be renumbered.**

Explanation:
The text of the DDL specification had to be extended beyond line 9999.
Possible Causes and/or Solutions:
1. The numbering sequence specified in the DDL command would have caused the file to extend beyond line 9999.
2. Respecify DDL command.

***** The decimal point position exceeds the size.**

Explanation:
The position specified for the decimal point in internal decimal (IDEC) fields cannot be greater than the size of the field.
Possible Causes and/or Solutions:
1. The decimal point position on an IDEC field is larger than the size (number of bytes) specified for the data item.
2. Decrease decimal point position so that it is less than the item size.
3. Increase the item size (number of bytes).

***** Too many areas.**

Explanation:
The number of areas specified in the AREA section is more than the maximum number of permissible areas.
Possible Causes and/or Solutions:
1. More than 15 areas have been specified in the AREA section for an environment that allows up to 16 areas per data base.
2. More than 7 areas have been specified in the AREA section for an environment that allows up to 8 areas per data base.
3. Check area specifications in AREA section.

***** Too many owners/member in set.**

Explanation:
A set can have at most 127 record types defined as owner/members.
Possible Causes and/or Solutions:
1. More than 127 record types have been defined as owners/members for the set.
2. Check owner/member specifications for set.

***** Too many pages per area.**

Explanation:
Too many pages are specified for an area.
Possible Causes and/or Solutions:
1. The pages specified for an area exceeds 4095 (for environments that allow 16 areas per data base) or 8191 (for environments allowing no more than 8 areas per data base).

***** Type already specified.**

Explanation:
A type specification for a data item has already been made.
Possible Causes and/or Solutions:
1. The type specification for a data item is made twice (e.g., CHAR 10 BIN).
2. Check item specifications in ITEM section.

***** Unable to create or open file.**

Explanation:
The system has encountered an error when trying to open or create a main area or other area file.
Possible Causes and/or Solutions:
1. Not enough room on disk.
2. Bad sector on disk.

***** Unrecognizable identifier.**

Explanation:
The system was expecting a valid identifier name.
Possible Causes and/or Solutions:
1. The name for a data base, area, set, record type or data item exceeds eight characters.
2. The name for a data base, area, set, record type or data item contains non-alphanumeric characters.

***** Unrecognizable or misplaced input.**

Explanation:
The indicated DDL text input is out of sequence, has a missing word, or contains a misspelled term.
Possible Causes and/or Solutions:
1. Check sequence of DDL text input.
2. Check syntax of DDL text input.
3. Check keywords specification in DDL text input.

***** Upper value in range is less than lower value.**

Explanation:
Self-explanatory.
Possible Causes and/or Solutions:
1. Check range specifications in ITEM section.

***** User section missing or misplaced.**

Explanation:
The system was expecting a USER statement based on an end-of-section descriptor.
Possible Causes and/or Solutions:
1. Check end-of-section descriptor (e.g., a period) in USER section.
2. Check user-password specifications in USER section.

***** Write error.**

Explanation:
The system has encountered an error while trying to write to disk during the data base initialization procedure.
Possible Causes and/or Solutions:
1. The number of pages defined for the main data base is less than the number of system pages required to initialize the data base.
2. Increase the number of pages specified in IDENTIFICATION section.

MDBS DDL MANUAL

Appendix A

Data Description Language Keywords

Appendix A

MDBS.DDL recognizes the following as keywords:

access	idec	owners
allowed	immat	page
any	immaterial	pages
are	in	prior
area	index	range
ascending	insertion	read
auto	int	real
az	integer	record
bin	is	retention
binary	item	set
by	key	size
bytes	keys	sorted
calc	language	str
char	lifo	string
character	log	syn
database	logfile	synonym
date	lowest	time
db	manual	times
decimal	member	title
default	members	to
descending	name	type
duplicates	next	unsigned
encrypted	nodup	user
end	not	width
fifo	occurs	with
file	of	within
fixed	optional	words
for	order	write
highest	owner	za

No keyword can be used as an identifier, unquoted password, or unquoted user name in DDL source specification unless it is preceded by one of the following words: IS, ARE, WITH, OF, TO, or BY.

MDBS DDL MANUAL

This page intentionally left blank.

MDBS DDL MANUAL

Appendix B

MDBS Data Description Language Syntax

Appendix B

See Chapters III-V for a detailed presentation of the DDL.

NOTATION:

_____	AN UNDERLINED EXPRESSION MUST APPEAR.
[ ]	ZERO OR ONE OF THE ALTERNATIVES WITHIN THE BRACKETS MUST BE USED.
{ }	EXACTLY ONE OF THE ALTERNATIVES WITHIN THE BRACES MUST APPEAR.
{ }*	ONE OR MORE OF THE ALTERNATIVES WITHIN THE BRACES MUST APPEAR.

DEFINITIONS:

<access-clause>:

{[READ ACCESS IS aclist-1] [WRITE ACCESS IS aclist-2]
 ACCESS IS aclist-3}

<CALC-key-clause>:

{CALC KEY / CALC KEYS} [IS / ARE] (id-3,)* [NODUP ALLOWED / DUPLICATES ARE [NOT] ALLOWED]

<sort-clause>:

{BY {ASCENDING / AZ / DESCENDING / ZA} {RECORD-TYPE / (di-2,)*}}* [DUPLICATES ARE {NOT ALLOWED / FIFO / IMMATERIAL / IMMAT}]

[INDEX WIDTH IS int-1]

<comment>:

/* comments */

ORDERING OF DDL SECTIONS:

Identification Section
User Section
Area Section (optional)
Record Sections
    Data Item Sections for each record section
Set Sections
    Owner Section for each set section
    Member Section for each set section
End Section

DDL SECTIONS:

1. Identification Section

{DATABASE / DB} NAME IS id-1

[FILE NAME IS "file-1"]

[SIZE IS int-1 PAGES]

[PAGE SIZE IS int-2 [BYTES / WORDS]]

[{LOG FILE / LOGFILE} NAME IS "file-2"]

[TITLE IS "string-1"]

[[{SYN / SYNONYM} IS id-2]*]

[DEFAULT SIZE FOR {{UNSIGNED / REAL / INTEGER / INT / BINARY / BIN / CHARACTER / CHAR / STRING / STR} [int-3] / {DECIMAL / IDEC} {int-4[,int-5]}}]*

[LANGUAGE IS {DANISH / ENGLISH / FINNISH / FRENCH / GERMAN / NORWEGIAN / SWEDISH}]

2. User Section

```
USER IS { usr   } [ WITH ] { pass   }
        { "usr" }  [ ,   ] { "pass" }
[<access-clause>]
```

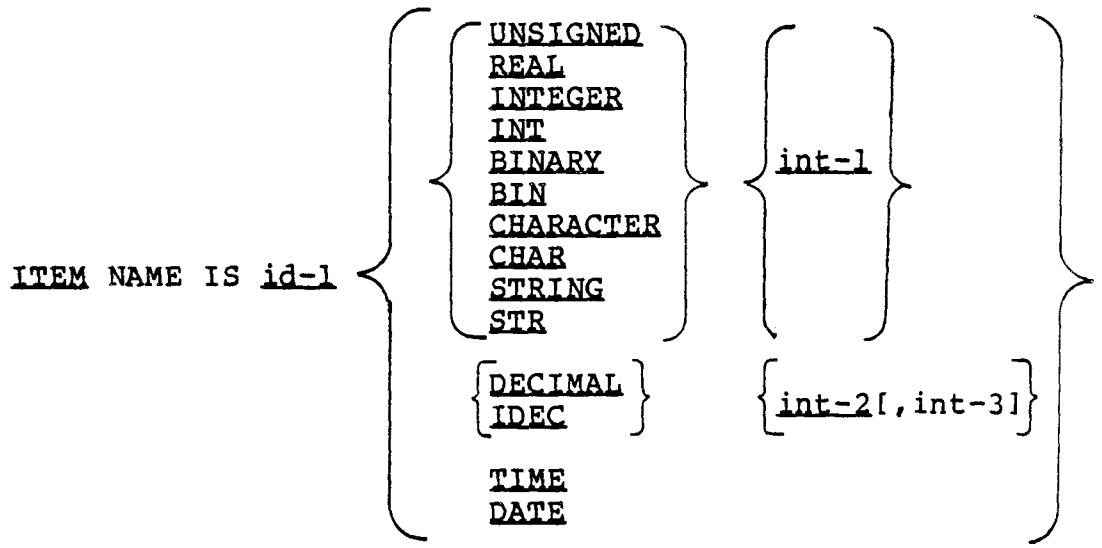
3. Area Section (zero, one, or more per schema)

```
AREA NAME IS id-1
[FILE NAME IS "file-1"]
[SIZE IS int-1 PAGES]
[PAGE SIZE IS int-2 [BYTES]]
                   [WORDS]
[POINTERS ARE [NOT] ALLOWED]
[<access-clause>]
[TITLE IS "string-1"]
[{ {SYN    } IS id-2 }* ]
  { {SYNONYM}         }
```

4. Record Section (one or more per schema)

```
RECORD NAME IS id-1
[ {WITHIN} { {ANY AREA  } [<CALC-key-clause>] }  ]
  {IN    } { {(ar-1,)*  }                       }
           { AREA OF     {OWNER } OF id-2      }
                         {MEMBER}              }
[<access-clause>]
[TITLE IS "string-1"]
[{ {SYNONYM} IS id-5 }* ]
 { {SYN    }         }
```


5. Item Section (zero, one, or more per record type)

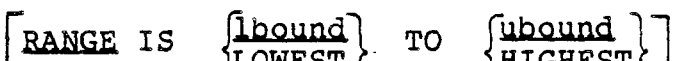


[OCCURS int-4 TIMES]

[IS ENCRYPTED]

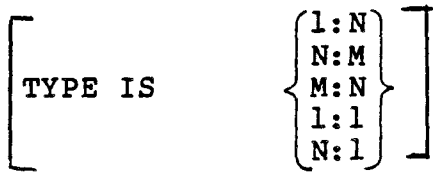
[<access-clause>]

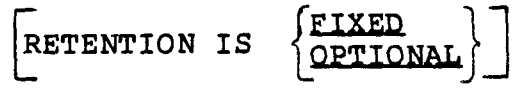
[TITLE IS "string-1"]



6. Set Section (zero, one, or more per schema)

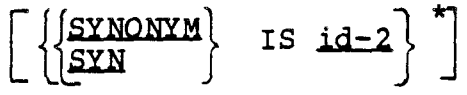
SET NAME IS id-1





[<access-clause>]

[TITLE IS "string-1"]



7. Owner Section (one per set)

```
{OWNER IS rt-1                          }
{OWNERS ARE ( {rt-2,}* )                }
[INSERTION IS {AUTO  }]
              {MANUAL}
[                    {FIFO                        }]
[                    {LIFO                        }]
[                    {NEXT                        }]
[ORDER IS            {PRIOR                       }]
[                    {IMMATERIAL                  }]
[                    {IMMAT                       }]
[                    {SORTED <sort-clause>        }]
```

8. Member Section (one per set)

```
{MEMBER IS rt-1                         }
{MEMBERS ARE ( {rt-2,}* )               }
[INSERTION IS {AUTO  }]
              {MANUAL}
[                    {FIFO                        }]
[                    {LIFO                        }]
[                    {NEXT                        }]
[ORDER IS            {PRIOR                       }]
[                    {IMMATERIAL                  }]
[                    {IMMAT                       }]
[                    {SORTED <sort-clause>        }]
```

9. End Section (one per schema)

```
END [.]
```


MDBS DDL MANUAL

Appendix C

Maximums and Minimums in a DDL Source Specification

Appendix C

	Minimum	Maximum imposed by MDBS software
Identifier name	1 alphanumeric character	8 alphanumeric characters
Password	1 alphanumeric character	16 alphanumeric characters
User Name	1 alphanumeric character	12 alphanumeric characters
Title string	1 character	determined by line length
Areas per data base	1	8 or 16
Pages per area*	1	8191 or 4095
Page size* (Version 3a)	256 bytes	multiple of 256 bytes=area page size
(Version 3c)	256 bytes	main area page size
Users per data base	1	no limit
Passwords per user	1	1
Record types per data base	1	255
Record types per area	0	255
Areas per record type	1	16
Record Occurrences per area	1	no limit*
Record Occurrences per data base	1	no limit*
Data items per record type	0	65535
Data items per CALC-key	1	65535
Data items per sort-key	1	65535
Repetitions of a repeating item	1	255
Data item size (integer)	1	16
(real)	2	16
(binary)	1	65535
(unsigned)	1	16
(fixed decimal)	1,0	30,7
(character)	1	65535
(string)	1	250
Data item feasibility range	see Table V-1	
Sets per data base	0	no limit
Owner record types per set	1	127

* Subject to an overall data base size of 232 bytes (i.e., over 4 billion bytes) for Version 3a, and in some cases larger for Version 3c.

	Minimum	Maximum imposed by MDBS software
Member record types per set	1	127
Owner record occurrences	0	no limit*
Member record occurrences	0	no limit*
Access code combinations per data base	1	65535

The foregoing are limitations imposed by MDBS III software. The environment (hardware, operating system, host language) within which MDBS is used may impose more restrictive practical limitations. For instance, if there is insufficient main memory to accommodate pages exceeding 4096 bytes apiece, then a fixed length record occurrence cannot exceed about 4000 bytes (nor can a data item value exceed this size).

* Subject to an overall data base size of 232 bytes (i.e., over 4 billion bytes) for Version 3a, and in some cases larger for Version 3c.

MDBS DDL MANUAL

Appendix D

Defaults

Appendix D

Optional Omission	Default
Data base (main area) file name	Generated by MDBS; based on data base name
Area file name	Generated by MDBS; based on area name
Main area's number of pages	Operating system dependent (typically 50)
Area's number of pages	Operating system dependent (typically 50)
Main area page size	Operating system dependent (typically 512 bytes)
Area page size	Page size for main area
Record location	Any area
CALC-key duplicates	Allowed
Data item replication factor	1
Data item encryption	No encryption
Data item feasibility range (with the exception of date)	
(lower bound)	LOWEST
(upper bound)	HIGHEST
Set type	1:N
Set retention	Optional
Set insertion for owner	Manual
Set insertion for member	Manual
Set owner order	Immaterial
Set member order	Immaterial
Sort-key duplicates	Immaterial
Sort-key index width	Determined by MDBS (does not exceed sort-key length)
Sorting sequence	English
Title	No title
Synonym	No synonym
Read access	
(area, record type, set)	a
(data item)	Read access of the data item's record type
Write access	
(area, record type, set)	a
(data item)	Write access of the data item's record type

MDBS DDL MANUAL

This page intentionally left blank.

MDBS DDL MANUAL

Appendix E

Data Item Range Examples

Appendix E

These are **examples** of the maximum range for various types and sizes of data items. The ranges have been calculated in accordance with the maximum range formulas given in Table II-1.

Item Type	Size	Maximum Range (inclusive)
integer	1	-128 to 127
integer	2	-32768 to 32767
unsigned	1	0 to 255
unsigned	2	0 to 65535
real (rounded to 7 digits)	4	-1.701411 (10^{38}) to +1.701411 (10^{38})
real (rounded to 17 digits)	8	-1.7014118346046921 (10^{38}) to +1.7014118346046921 (10^{38})
idec	7	-.99999999 (10^{63}) to +.99999999 (10^{63})
idec	8	-.99999999 (10^{63}) to +.99999999 (10^{63})
binary	1	00000000 to 11111111 (binary)
date	--	01/01/1799 to 12/31/1924
time	--	00:00:00 to 255:59:59

MDBS DDL MANUAL

Appendix F

Estimating Data Base Sizes

APPENDIX F

1) Let NRO be total number of record occurrences for a record type

R in Version 3a, R=3; in Version 3c, R=4

DL be the length of the data in a record occurrence after data compression (in bytes)

S_{SYS} number of sorted system-owned sets this record type participates in

S'_{SYS} number of non-sorted system-owned sets this record type participates in

$S_{1:1}$ number of 1:1 sets this record type participates in

$S_{1:N}$ number of non-system-owned sorted 1:N sets this record type is a member of

$S'_{1:N}$ number of non-system-owned non-sorted 1:N sets this record type is a member of

$T_{1:N}$ number of 1:N sets this record type is an owner of

$S_{N:M}$ number of N:M sets this record type participates in, for which occurrences of this record type are sorted

$S'_{N:M}$ number of N:M sets this record type participates in, for which occurrences of this type are not sorted

A average number of records related to an occurrence of this record type via an N:M set

W average width (in bytes) of sort indices involving this record type

Then the approximate total number of bytes used by the occurrences of this record type is obtained by multiplying NRO times the quantity:

$$R+DL+4(S'_{SYS}+S_{1:1}+S_{1:N}+T_{1:N}+S_{N:M}+S'_{N:M}(1+A))+8(S'_{1:N}+S'_{N:M})+(4+W)(S_{1:N}+AS_{N:M}+S_{SYS})$$

2) Note that the number of bytes "used" by a record occurrence are not contiguous in memory. The actual size of a record occurrence is approximately:

$$DL+R+4(S_{1:1}+S_{1:N}+S'_{1:N}+T_{1:N}+S_{N:M}+S'_{N:M})$$

3) On each page there are 8 bytes that cannot be used to store record occurrences.

4) In each area there are a few pages reserved as system pages, these cannot be used to hold record occurrences. The number of System pages in each area is reported to the console when MDBS.DDL initializes a data base.

5) The smallest "hole" of free space that is permitted on a page is 4 bytes.

MDBS DDL MANUAL

Appendix G

List of QRS Keywords

MDBS DDL MANUAL

Appendix G

AND
BY
BYE
CLOSE
COMPUTE
DATFORM
DBSTAT
DEFINE
DISP
DISPLAY
ECHO
EDIT
EQ
FOR
FROM
GE
GT
HELP
IN
LE
LIST
LT
NE
NOT
OPEN
OR
QUIT
READ
SELECT
SET
SPEW
STATS
THRU
WRITE
XOR

MDBS DDL MANUAL

This page intentionally left blank.

MDBS DDL MANUAL

Appendix H

Alternative Layout for DDL Specification of Figure III-2

Appendix H

```
/*****
 *
 *  sample data base description (advanced level)
 *
 *****/

db JOBS
               file "JOBS.DB"
               size 300 pages, page size 1024
               logfile "JOBS.LOG"


/****************** define defaults for item types ******************/

   default for unsigned 2
   default for str 50

user "BOB SMITH"    with GTC       read access (a,b) write access a
user ANALYST        with 7778XK4   read (b-e)  write (b-f)
user "K FERGUSON"   with "tashi"   access (b,p)
user "D LEHR"       with "smiles"  access (a-p)


/** define additional areas to supplement the main data base area ***/

area name is job1
     file name is "JOB1.DB"
     size is 100 pages, pointers not allowed
     page size is 512  /* Note:  this page size would not be
                               allowed in Version 3a because
                               the main area's page size is 1024 */
     read access is (a-d)  write access is (a,p)
area JOB2
     file "JOB2.DB"
     size 700 pages
     read (b,d)  write (a,b,p,f)

/********************* record type definitions **********************/

record DEPT
     in JOBS calc key is NAME nodup
     read access b  write access (a,b)
          item DNUMBER
                         int 1
                         range is 1 to 42
                         syn is DNO
          item is NAME
                         char 12   encrypted
                         write access b
          item LOCATION
                         str 35   syn LOC
                         range "A" to "ZZZZZZ"
```



```
record employee in JOB1, key is ID nodup
     read access (a,d)  write access (a,p)
          item ID
                         char 9
                         encrypted
                         range is "0" to "999999999"
          item LASTNAME
                         str 20
                         range "Aa" "Zz"
          item FNAME
                         str 12
                         range "Aa" "Zz"
          item PASTJOBS
                         str 25
                         occurs 3 times
          item YTDEARN
                         idec 7,2
                         encrypted
                         range 0.00  94000.00
                         read access a
                         write access a
record BIOGRAPH in JOB2
     read access b  write access p
          item LINE
                         str occurs
                         5 times
record SKILL in (JOBS,JOB2) calc key is SKILCODE nodup
     read access (b,d)  write access f
          item SKILCODE
                         unsigned
                         syn SC
                         range 0 to 3000
          item DESCRIPT
                         str 55
          item RATING
                         real 2
                         range 0.0 to 4.0
record JOB in any area
     read access b  write access (a,p)
          item JOBCODE
                         unsigned 1
                         range 1 to 250
          item DESCRIPT
                         str

/************************ set definitions ***************************/

set IEMP, type 1:n, retention fixed
     read access d  write access p
          owner SYSTEM
          member EMPLOYEE
          order sorted by ascending (lastname, Fname)
          insertion auto
```



```
set POSSESS, type n:m
     read access (a,d)  write access (f,p)
          owner EMPLOYEE
          order sorted by az ID
          member SKILL
set DETAILS, type 1:1, fixed
     read access (b,d)  write access p
          owner EMPLOYEE
          member BIOGRAPH
          auto
set FILLEDBY, type 1:n
     read access (a-d)  write access p
          owner JOB
          member EMPLOYEE
          fifo
          auto
set NEEDS n:m
     read access b  write access (f,p)
          owner JOB
          sorted za JOBCODE
          member SKILL
          sorted az (RATING,SKILCODE)
set HAS 1:n
     read access (a,b)  write access a
          owner DEPT
          member EMPLOYEE
          lifo
          auto
set IDEP 1:n,  read access b
          owner SYSTEM
          member DEPT
          auto
set IJOB 1:n,  read access b
          owner SYSTEM
          member JOB
          auto
          fifo
END
```


MDBS DDL MANUAL

Appendix I

MDBS.CNV

Appendix I

MDBS.CNV is a utility program, available in many environments, to aid a Version 1 MDBS user in converting to Version 3 of MDBS. This utility converts an existing Version 1 schema description into its corresponding Version 3 form. Because of the many differences in the two formats, the Version 3 output may not provide a perfect equivalent of the Version 1 description. Most discrepancies will arise if the Version 1 schema contains depending-on items or more than 16 separate access level specifications. If these situations are encountered, a warning message is issued, however processing will continue. Depending-on items are merely ignored in the Version 3 schema. Access levels will be mapped correctly if there are less than 16 separate access level specifications in the Version 1 schema. If there are more than 16 separate access levels, the first sixteen will be mapped consistently and the remainder will take on a default value. It is suggested that MDBS.CNV be used only as a learning aid to gain understanding of the format of a Version 3 schema description.

To use MDBS.CNV, the user must have on-line the MDBS.CNV program as well as a copy of the DDL specification of the schema to be converted. It is imperative that the Version 1 specification be free from errors (i.e., it must be able to survive a pass through the Version 1 DDL analyzer).

The MDBS.CNV program is menu driven and very easy to use. When this program is executed in the host operating system, the user is prompted for the name of the old DDL file (i.e., the name of the file containing the Version 1 schema description). The user is then prompted for the name of the file name in which the Version 3 schema description is to be placed. If this file already exists, its contents will be overwritten. Otherwise, a file having the specified name will be created.

Because of differences in various language interfaces, the user is prompted for the host language being used.

During the course of MDBS.CNV processing, the Version 1 schema description is first displayed, followed by the Version 3 description. The Version 3 description may now be input to the Version 3 DDL analyzer for analysis.

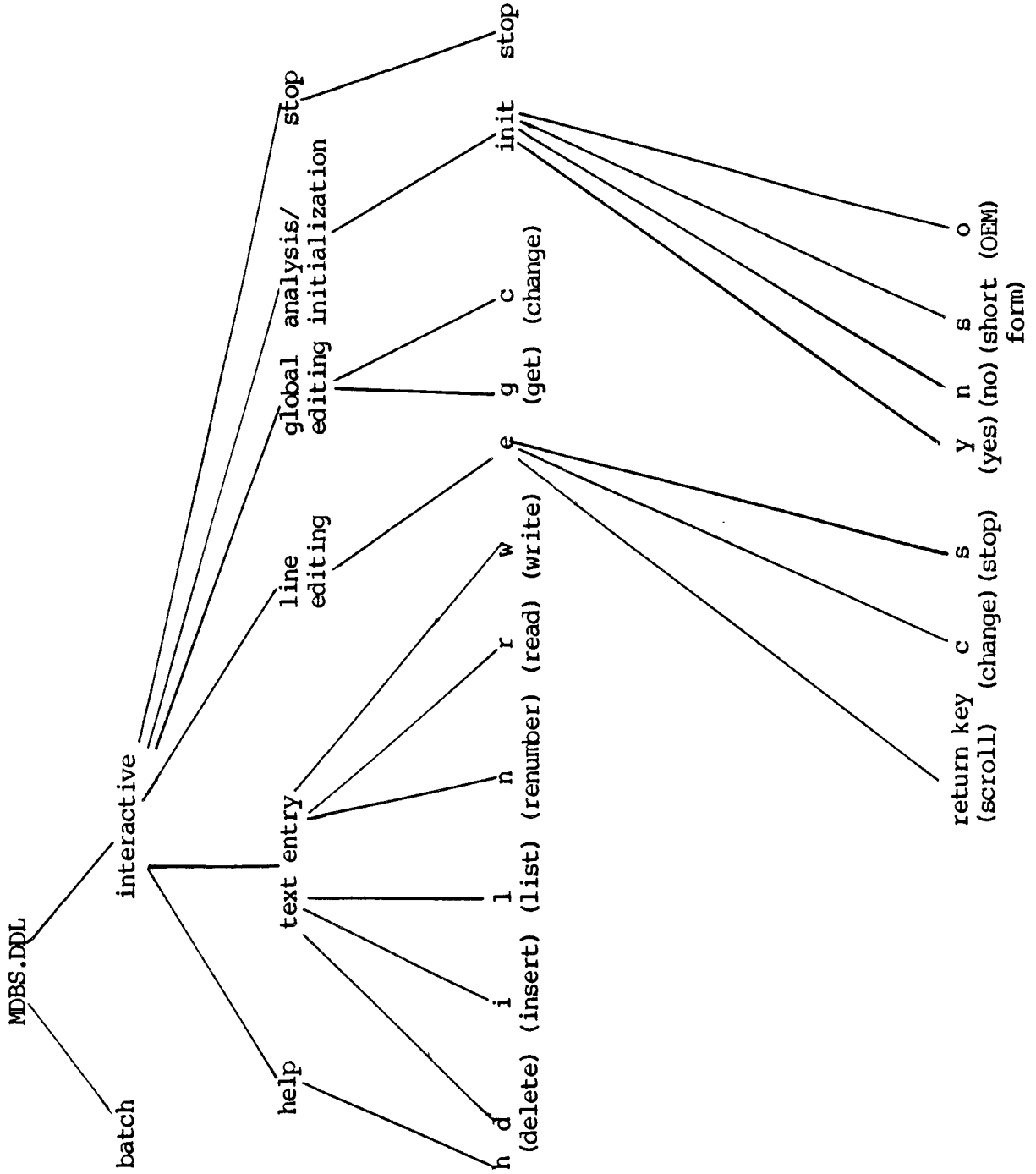
MDBS DDL MANUAL

Appendix J

Flow of Control When Using MDBS.DDL

Appendix J

Flow of Control When Using MDBS.DDL



MDBS DDL MANUAL

This page intentionally left blank.

DOCUMENTATION COMMENT FORM

MDBS Document Title:_____

We welcome and appreciate all comments and suggestions that can help us to improve our manuals and products. Use this form to express your views concerning this manual.

Please do not use this form to report system problems or to request materials, etc. System problems should be reported to **MDBS** by phone or telex, or in a separate letter addressed to the attention of the technical support division. Requests for published materials should be addressed to the attention of the marketing division.

Sender:

_____ _____
(name) (position)

_____ _____
(company) (telephone)

_____
(address)

_____
(city, state, zip)

COMMENTS:

Areas of comment are general presentation, format, organization, completeness, clarity, accuracy, etc. If a comment applies to a specific page or pages, please cite the page number(s).

_____

_____

_____

_____

_____

_____

_____

_____

Continue on additional pages, as needed. Thank you for your response.