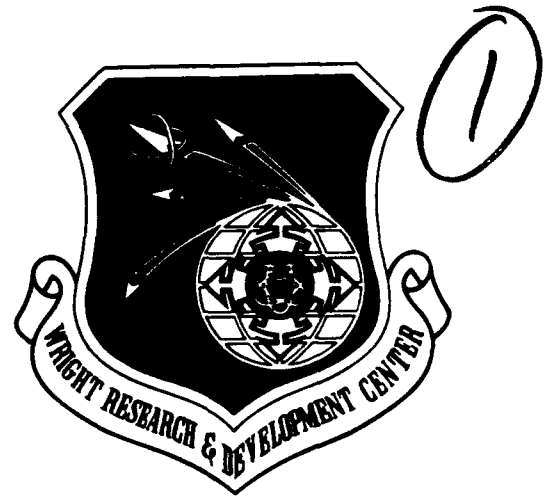


WRDC-TR-90-8007  
Volume V  
Part 5

**AD-A252 450**



INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)  
Volume V - Common Data Model Subsystem  
Part 5 - Neutral Data Definition Language (NDDL) Development  
Specification

J. Althoff, M. Apicella, S. Singh

Control Data Corporation  
Integration Technology Services  
2970 Presidential Drive  
Fairborn, OH 45324-6209



September 1990

Final Report for Period 1 April 1987 - 31 December 1990

Approved for Public Release; Distribution is Unlimited

**92-14514**



MANUFACTURING TECHNOLOGY DIRECTORATE  
WRIGHT RESEARCH AND DEVELOPMENT CENTER  
AIR FORCE SYSTEMS COMMAND  
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6533

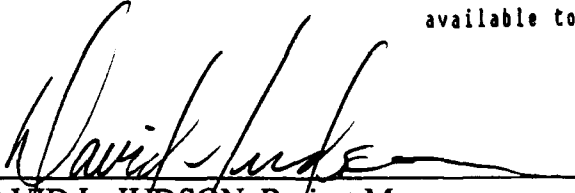
**92 6 02 015**

## NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, regardless whether or not the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data. It should not, therefore, be construed or implied by any person, persons, or organization that the Government is licensing or conveying any rights or permission to manufacture, use, or market any patented invention that may in any way be related thereto.

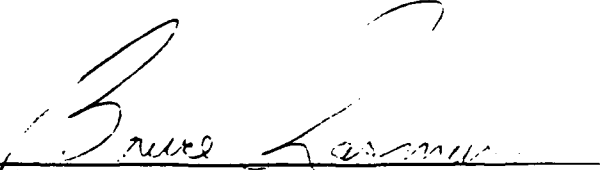
This technical report has been reviewed and is approved for publication.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations

  
DAVID L. JUDSON, Project Manager  
WRDC/MTI  
Wright-Patterson AFB, OH 45433-6533

25 July 91  
DATE

FOR THE COMMANDER:

  
BRUCE A. RASMUSSEN, Chief  
WRDC/MTI  
Wright-Patterson AFB, OH 45433-6533

25 July 91  
DATE

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WRDC/MTI, Wright-Patterson Air Force Base, OH 45433-6533 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				
1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS None		
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution is Unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE				
4. PERFORMING ORGANIZATION REPORT NUMBER(S) DS 620341100		5. MONITORING ORGANIZATION REPORT NUMBER(S) WRDC-TR-90-8007 Vol. V, Part 5		
6a. NAME OF PERFORMING ORGANIZATION Control Data Corporation; Integration Technology Services		6b. OFFICE SYMBOL (if applicable) WRDC/MTI		7a. NAME OF MONITORING ORGANIZATION WRDC/MTI
6c. ADDRESS (City, State, and ZIP Code) 2970 Presidential Drive Fairborn, OH 45324-6209		7b. ADDRESS (City, State, and ZIP Code) WPAFB, OH 45433-6533		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Wright Research and Development Center, Air Force Systems Command, USAF		8b. OFFICE SYMBOL (if applicable) WRDC/MTI		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUM. F33600-87-C-0464
8c. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB, Ohio 45433-6533		10. SOURCE OF FUNDING NOS.		
11. TITLE (Include Security Classification) See Block 19		PROGRAM ELEMENT NO. 78011F	PROJECT NO. 595600	TASK NO. F95600 WORK UNIT NO. 20950607
12. PERSONAL AUTHOR(S) Control Data Corporation: Althoff, J. L., Apicella, M. L., Singh, S.				
13a. TYPE OF REPORT Final Report	13b. TIME COVERED 4/1/87-12/31/90	14. DATE OF REPORT (Yr., Mo., Day) 1990 September 30		15. PAGE COUNT 237
16. SUPPLEMENTARY NOTATION WRDC/MTI Project Priority 6203				
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify block no.)		
FIELD	GROUP	SUB GR.		
1308	0905			
19. ABSTRACT (Continue on reverse if necessary and identify block number)				
<p>This development specification establishes the development, test and qualification requirements of the Neutral Data Definition Language (NDDL) Processor computer program.</p> <p>Block 11 - INTEGRATED INFORMATION SUPPORT SYSTEM (IISS) Vol V - Common Data Model Subsystem Part 5 - Neutral Data Definition Language (NDDL) Development Specification</p>				
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED x SAME AS RPT. DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a. NAME OF RESPONSIBLE INDIVIDUAL David L. Judson		22b. TELEPHONE NO. (Include Area Code) (513) 255-7371		22c. OFFICE SYMBOL WRDC/MTI

DD FORM 1473, 83 APR

EDITION OF 1 JAN 73 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

FOREWORD

This technical report covers work performed under Air Force Contract F33600-87-C-0464, DAPro Project. This contract is sponsored by the Manufacturing Technology Directorate, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Bruce A. Rasmussen, Branch Chief, Integration Technology Division, Manufacturing Technology Directorate, through Mr. David L. Judson, Project Manager. The Prime Contractor was Integration Technology Services, Software Programs Division, of the Control Data Corporation, Dayton, Ohio, under the direction of Mr. W. A. Osborne. The DAPro Project Manager for Control Data Corporation was Mr. Jimmy P. Maxwell.

The DAPro project was created to continue the development, test, and demonstration of the Integrated Information Support System (IISS). The IISS technology work comprises enhancements to IISS software and the establishment and operation of IISS test bed hardware and communications for developers and users.

The following list names the Control Data Corporation subcontractors and their contributing activities:

<u>SUBCONTRACTOR</u>	<u>ROLE</u>
Control Data Corporation	Responsible for the overall Common Data Model design development and implementation, IISS integration and test, and technology transfer of IISS.
D. Appleton Company	Responsible for providing software information services for the Common Data Model and IDEF1X integration methodology.
ONTEK	Responsible for defining and testing a representative integrated system base in Artificial Intelligence techniques to establish fitness for use.
Simpact Corporation	Responsible for Communication development.



<b>Accession For</b>	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
<b>Availability Codes</b>	
Dist	Avail and/or Special
A-1	



Structural Dynamics  
Research Corporation

Responsible for User Interfaces,  
Virtual Terminal Interface, and Network  
Transaction Manager design,  
development, implementation, and  
support.

Arizona State University

Responsible for test bed operations  
and support.

Table of Contents

		<u>Page</u>
SECTION 1.	SCOPE .....	1-1
1.1	Identification .....	1-1
1.2	Functional Summary .....	1-1
SECTION 2.	DOCUMENTS .....	2-1
2.1	Reference Documents .....	2-1
2.2	Terms and Abbreviations .....	2-2
SECTION 3.	REQUIREMENTS .....	3-1
3.1	Computer Program Definition .....	3-1
3.1.1	System Capacities .....	3-1
3.1.2	Interface Requirements .....	3-1
3.2	Detailed Functional Requirements .....	3-3
3.2.1	Initialization .....	3-3
3.2.2	Input/Output .....	3-4
3.2.3	Error Handling .....	3-4
3.2.4	Parse Commands .....	3-5
3.2.5	Database Access .....	3-7
3.2.6	General Command Processing .....	3-8
3.2.7	Termination .....	3-10
3.2.8	Individual Command Processing .....	3-10
3.3	Performance Requirements.....	3-161
3.3.1	Programming Methods .....	3-161
3.3.2	Program Organization .....	3-161
3.3.3	Modification Consideration.....	3-161
3.3.4	Special Features .....	3-162
3.3.5	Expandability .....	3-162
3.4	Human Performance .....	3-162
3.5	Database Requirements .....	3-162
3.5.1	Database Overview .....	3-162
3.5.2	Relations Between Tables and Views.....	3-162
3.5.3	Detailed Description of Tables and Views.....	3-162
SECTION 4.	QUALITY ASSURANCE PROVISIONS .....	4-1
4.1	Introduction and Definitions .....	4-1
4.2	Computer Programming Test and Evaluation .....	4-1
SECTION 5.	PREPARATION FOR DELIVERY .....	5-1
SECTION 6.	NOTES .....	6-1

Table of Contents

	<u>Page</u>
APPENDIX A. Oracle Data Dictionary .....	A-1
APPENDIX B. CDM Tables Accessed .....	B-1
APPENDIX C. Internal Tables used by NDDL Commands .....	C-1

List of Illustrations

<u>Figure</u>	<u>Title</u>	<u>Page</u>
1-1	NDDL Functionality Matrix .....	1-3
3-1	NDDL Processor Interface .....	3-2
3-2	Organization of NDDL Functional Requirements .....	3-3

## SECTION 1

### SCOPE

#### 1.1 Identification

This specification establishes the development, test and qualification requirements of a computer program identified as the Neutral Data Definition Language Processor, known in this document as the NDDL Processor. The NDDL Processor is one configuration item of the Integrated Information Support System (IISS) Common Data Model (CDM) Subsystem.

#### 1.2 Functional Summary

The NDDL processor is a language used to manipulate and populate information in the Common Data Model (CDM) of the IISS System database. It provides the user with three modes of operation: (1) Batch Mode allows NDDL command files to be executed; (2) Interactive Mode allows the user to enter NDDL commands at a terminal; and (3) Forms Mode allows the user use of the IISS forms processor to display input and output screens of NDDL commands. The NDDL processor allows the user to populate and maintain the three schemas of the CDM: external, conceptual and internal and the mappings between each. The NDDL also provides capabilities for manipulation of many IDEF-1 models and submodels needed during the process of developing the single integrated model of the conceptual schema. Only the integrated model may be mapped to the external and internal schemas. The NDDL was designed by a joint working group of IISS coalition members described in the Integration Task Report, Reference 8. The language is modeled after SQL and the command features a combination of a few simple verbs (operators) along with the necessary parts of the CDM (objects). The functionality of NDDL is summarized in the matrix of Figure 1-1. The following notes refer to the foot notes of the matrix.

1. Internal Schema Objects are defined rather than created since IISS assumes internal schema describes actual, previously existing databases.
2. DESCRIBE serves the purpose of creating, altering and dropping descriptions for the applicable objects.
3. Aliases are maintained for entities and attributes only.
4. Keywords are maintained only for entities, attributes and relations. They can only be created when associated with entities, attributes or relations.
5. The noted objects can only be altered through use of DROP and CREATE/DEFINE operators.
6. ALTER commands generally have ADD, DROP and ALTER suboperators for subobjects.

7. Data items are created and dropped as subobjects of views.
8. Data types are created and dropped as subobjects of domains.
9. Data fields are created as subobjects of records.
10. Maps do not have names of their own and cannot be renamed or described.

# OBJECT

OPERATOR	ALGORITHM	ALIAS	ATTRIBUTE	CATEGORY	DBMS	DATATYPE	DATATYPE	DATATYPE	DOMAIN	ENTITY	FIELD	HOST	KEYWORD	MAP	MODEL	MODULE	PARTITION	PRECEDENCE	RELATION	SET	UNION	VIEW
ALTER	5	X	X	X	X	5	X	5	5	X	X	X	X	X	X	X	X	X	X	5	X	5
CHECK										X					X	X						
COMBINE															X							
COPY			X	X	X	1	7	1	8	X	X	9	1	4	X	X	1	1	1	X	1	X
CREATE		X	X	X	X	1	7	1	8	X	X	9	1	4	X	X	1	1	1	X	1	X
DEFINE	X		X	X	X	X	X	X	X	X	X	X	X	10	X	X	X	X	X	X	X	X
DESCRIBE	X	X	X	X	X	7	X	8	X	X	X	X	X	X	X	X	X	X	X	X	X	X
DROP	X	X	X	X	X	7	X	8	X	X	X	X	X	X	X	X	X	X	X	X	X	X
MERGE															X							
RENAME			X	X		X	X	X	X	X	X	X	X	10	X			X	X	X		X
HALT																						

Figure 1-1. NDDL Functionality Matrix

## SECTION 2

### DOCUMENTS

#### 2.1 Reference Documents

1. Control Data Corporation, Test Bed System Requirement Document (Draft); SRD620340000, 31 May 1988.
2. Control Data Corporation, Test Bed System Design Specification; SDS620340000, 31 May 1988.
3. ICAM Documentation Standards; IDS15012000A, 28 December 1981.
4. Control Data Corporation, IISS Software Development Guidelines/Conventions (Draft); 31 May 1988.
5. Structural Dynamics Research Corporation, IISS User Interface Management System Services User Manual; UM620344100, 31 May 1988.
6. Structural Dynamics Research Corporation, IISS Form Processor Users Manual; UM620344200, 31 May 1988.
7. Control Data Corporation, IISS Neutral Data Definition Language (NDDL) User's Guide (Preliminary Draft); 31 May 1988.
8. Hughes IISS Integration Task; 16 April 1984.
9. Softech, ICAM Architecture, Part II, Vol. V, Information Modeling (IDEF1); FTR110210000.
10. D. Appleton CO., CDM Administrator's Manual; UM620341000, 31 May 1988.
11. D. Appleton Co., CDM1-IDEF, Model of the Common Data Model; CCS620341000, 31 May 1988.
12. General Electric Co., Quality Assurance Plan; QAP620344000, 31 May 1988.
13. D. Appleton Co., Embedded NDML Programmer's Reference Manual; IPRM620341200, 31 May 1988.
14. Control Data Corporation, NTM Programmer's Guide; UM620340001, May 1988.

#### 2.2 Terms and Abbreviations

Attribute Use Class: (AUC).

Application Interface: (AI) A collection of routines with the same calling sequences as the Forms Processor and Virtual

Terminal callable routines that enables applications to be hosted on computers other than the host of the User interface.

Assertion: Predicate that applies to one or more attributes; checked after completion of an action to determine if the results should be committed.

Conceptual Schema: (CS).

Common Data Model Processor: (CDMP).

Common Data Model: (CDM) Describes common data application process formats, form definitions, etc, of the IISS and includes conceptual schema, external, internal schemas, and schema transformation operators.

Data Field: (DF) An element of data in the internal schema. Generally, a DBMS will reference data by this name.

Data Item: (DI) An element of data in the external schema. An NDML programmer references data by this name.

Data Type: A specific computer representation of a domain.

Distributed Request Supervisor: (DRS) This IISS CDM Subsystem Configuration Item controls the execution of distributed NDML queries and non-distributed updates.

Domain: A logical definition of legal attribute class values.

Domain Constraint: Predicate that applies to a single domain.

External Schema: (ES).

Forms: Structured views which may be imposed on windows or other forms. A form is composed of fields where each field is a form, item, or window.

Forms Processor: (FP) A set of callable execution time routines available to an application program for form processing.

Internal Schema: (IS).

Integrated Information Support System: (IISS), a computing environment used to investigate, demonstrate, test the concepts and produce application for information management and information integration in the context of Aerospace Manufacturing. The IISS addresses the problems of integration of data resident on heterogeneous databases supported by heterogeneous computers interconnected via a local Area Network.

Mapping: The correspondence of independent objects in two schemas: ES to CS or CS to IS.

NDDL User: The CDM administrator or his designated representative.



Network Transaction Manager: (NTM) Performs the coordination, communication and housekeeping functions required to integrate the Application Processes and System Services resident on the various hosts into a cohesive system.

Neutral Data Definition Languages: (NDDL) A language used to manipulate and populate information in the Common Data Model (CDM) or IISS System Database.

Neutral Data Manipulation Language: (NDML) A language developed by the IISS project to provide uniform access to common data, regardless of database manager or distribution criteria. It provides distributed retrieved and single node update.

ORACLE: Relational DBMS based on the SQL (Structured Query Language, a product of ORACLE Corp, Menlo Park, CA). The CDM is an ORACLE database.

Object: Named Common Data Model item; for example, entity class, relation class, attribute class.

Trigger: Action that is invoked at the commit completion of another action.

User Interface: (UI) Controls the user's terminal and interfaces with the rest of the system.

Virtual Terminal Interface: (VTI) Performs the interfacing between different terminals and the UI. This is done by defining a specific set of terminal features and protocols which must be supported by UI software which constitutes the virtual terminal definition. Specific terminals are then mapped against the virtual terminal software by specific software modules written for each type of real terminal supported.

## SECTION 3

### REQUIREMENTS

#### 3.1 Computer Program Definition

The NDDL processor is the computer program that translates the command statements of this language and performs the operations requested, updating the CDM database. The NDDL language is non-procedural. The NDDL processor is essentially an interpreter, executing one command at a time, in the order presented by the user.

Each command is parsed for syntactic correctness. Control is transferred to the individual command processor for the semantic validation of the command. If all semantic checks are found to be correct, the database is updated or information retrieved.

##### 3.1.1 System Capacities

The NDDL is designed to allow multiple users at the same time. Data limits are imposed only by the capacity of the DBMS. Processing speed limits are imposed by the speed of the computer and the number of other users and the speed and efficiency of the NTM subsystem and the IISS Forms Processor. A number of COBOL and C fixed size tables will be used to temporarily hold information. These limits can be changed very easily in a virtual memory environment.

##### 3.1.2 Interface Requirements

The NDDL processor shall make use of the IISS Forms Processor for command input and shall allow batch input as well. The NDDL processor shall make use of IISS NDML wherever possible to retrieve data from the CDM native ORACLE wherever NDML is not sufficient.

### 3.1.2.1 Interface Block Diagram

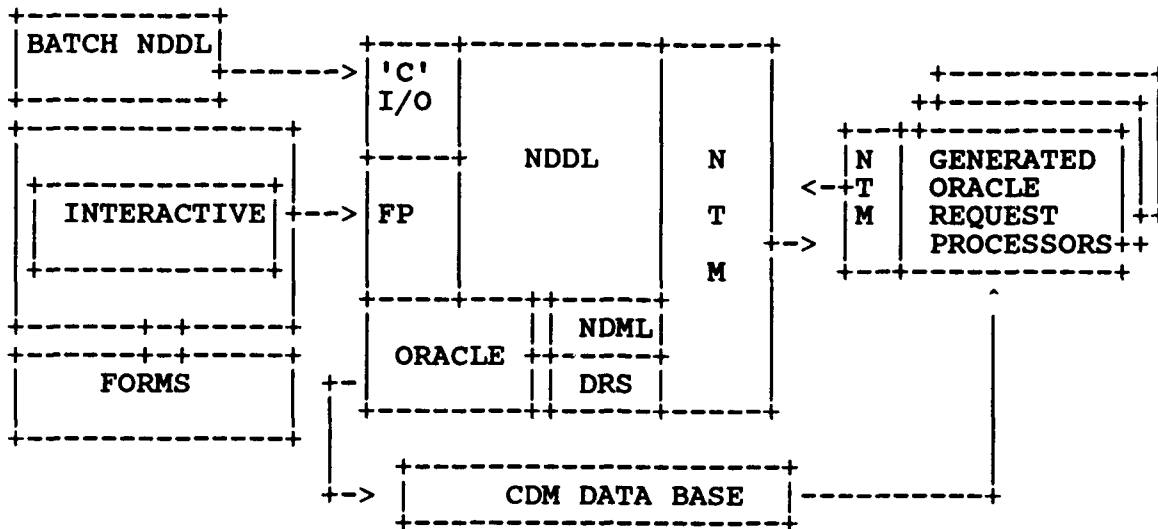


Figure 3-1. NDDL Processor Interfaces

### 3.1.2.2 Interface Requirements

The NDDL processor makes use of the IISS Forms Processor directly for forms interactive input. NDDL also makes use of the standard C input/output library to allow user non-forms interactive input or batch input via file redirection. Database access is through a combination of:

- (1) Oracle for Recursive Searches and Updates not supported by the NDML precompiler and
- (2) NDML for all other retrievals and Updates.

The NDML routines are precompiled by the IISS NDML precompiler and ORACLE request processors are generated. The Request Processors communicate with NDDL through the DRS which uses IISS NTM services.

It is a design goal to replace the use of ORACLE with NDML to achieve DBMS independence, and to allow the CDM database itself be distributed.

It is a design goal to make NDDL an application controlled by the IISS User Interface subsystem to make use of its capabilities.

### 3.2 Detailed Functional Requirements

This description of functional requirements is broken down into nine subfunctional areas. These areas are identified in the block diagram, Figure 3-2, which includes the following paragraph numbers. The forty-two commands currently making up NDDL are each described in Subsection 3.2.8.

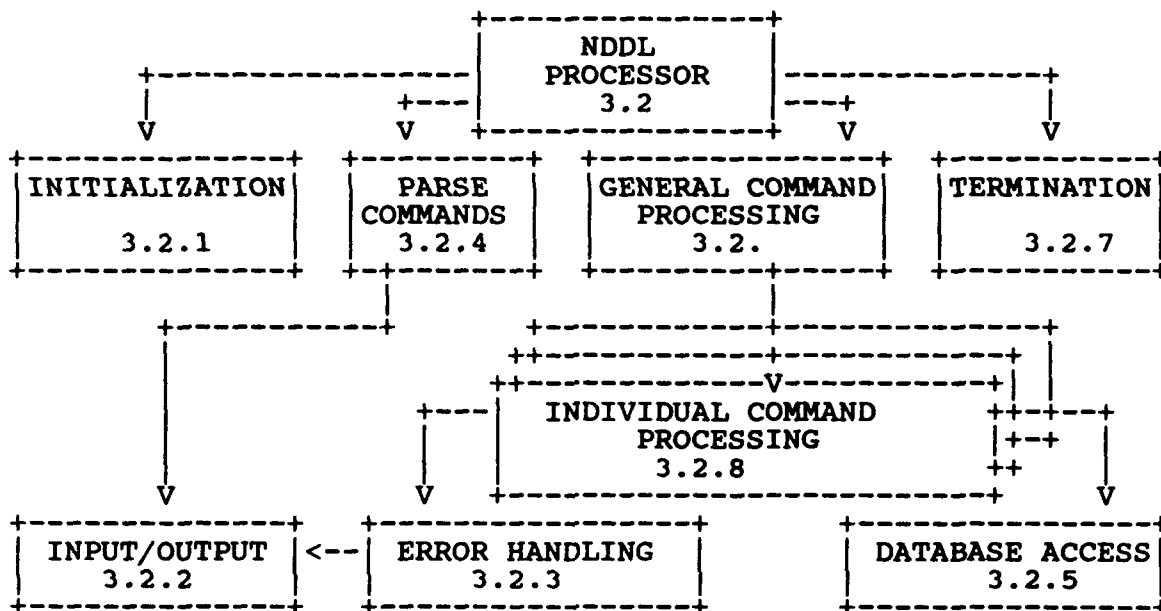


Figure 3-2. Organization of NDDL Functional Requirements

#### 3.2.1 Initialization

##### A. Function:

Initialization will allow the NDDL processor to perform all initialization requirements with other subsystems and software environments.

##### B. CDM Requirements:

None

##### C. Processing:

1. Determine the processing mode, either interactive, through forms, or batched.
2. Initialize with NTM through use of INITEX service. In the future, this should be changed to INITAL when UI services are required.

3. If in forms mode, initialize to the forms processor by using the INITFP service, OPNFRM and ADDFRM to create the user's initial form.
4. Log on to the ORACLE DBMS. The log on data area will be global data structures.
5. Initialize any other data structures necessary for any commands to their null, or initial state.
6. Initialize any other global variables such as current model, current database, current commit mode, current output mode, etc.

### 3.2.2 Input/Output

#### A. Function:

Provide user input to the other components of the NDDL processor in an invisible manner, without respect to the means in which the input was obtained. Provide output for NDDL processor to the user through a standard interface to allow the same invisibility.

#### B. CDM Requirements:

None

#### C. Processing:

The standard C character input/output routines will be used. They will be modified, however, to recognize the current input/output mode. Batch mode will use the existing capability of the C library. Interactive mode will make use of forms processor calls. Because the forms processor can return many screens of information at a time, the modified input/output routines shall extract a single character at a time from the data structures. Since output consists of simple information messages to the user, PMSGLS forms processor calls can be used for all messages or PRINTF, if batch mode, to standard output. Output of generated NDDL uses the standard C file I/O primitives.

### 3.2.3 Error Handling

#### A. Function:

Provide a single standard means of communicating errors to the NDDL user. The interface shall be simple, readily usable and invisible to the particular input/output mode. The error handling shall also make database transaction rollback conditions simple to

recognize. User requirements for command skipping after a semantic error shall be implemented.

B. CDM Requirements:

None

C. Processing:

Three major entry points to the error handling functionality of NDDL shall be established corresponding to the three types of errors. These are:

1. An entry point for "fatal" errors. These are errors from other subsystems or software other than NDDL. These errors are to be handled in accordance with the IISS Error Handling Philosophy, Reference Number 12. A standard routine is called to log the message in a central place. These error conditions shall also be communicated to the user as type 2 below.
2. An entry point for user errors. These are errors that are caused by the user and can be recovered by user action. An example may be creating an entity that already exists. This error handler must set a flag so at the end of the command, any database changes are backed out through a rollback procedure supplied by the DBMS or NDML.
3. An entry point for warning messages. These are indications of problems of user understanding, such as dropping a set that does not exist, or simple informative messages about actions that have occurred, such as "model altered".

It is the responsibility of the general command processor, Section 3.2.6, not to process commands in batch after a user error or fatal error has occurred. This prevents a later command from causing unpredictable harm.

3.2.4 Parse Commands

A. Function:

The Parse Command subfunction of IISS will provide a mechanism for accepting user command input, validating correct syntax, reporting syntax errors and saving pertinent command information in data structures independent of the syntax.

B. CDM Requirements:

None

C. Processing:

This function will be provided through code generated by the UNIX tools YACC and LEX and interface routines provided as part of this function (UNIX is a trademark of Bell Labs).

1. LEX is a tool that generates lexical analyzers. Given a specification of the reserved words or tokens, LEX will generate a routine that will accept user input and return control to the caller on each token recognized.
2. YACC is a tool that generates a parser that validates user input as matching the grammar or syntax of the language. The parser generated has the capability of calling user specified routines or code called "actions". YACC is commonly used in compiler construction. YACC uses a syntax specification of the NDDL commands and generates the NDDL parser. This specification is not treated as an IISS deliverable because to do so would require the user or target site to have the UNIX tools.
3. Token primitive routines will be developed that store user entered command data, or tokens, in a special data structure. This data structure is simply a matrix of pointers into a string of concatenated tokens. The columns of the matrix are called lists. Lists generally have like tokens; for example, all the keywords entered on a CREATE ENTITY command. The rows of the matrix are generally meaningless, unless the syntax defines a special correspondence between lists. An example is CREATE VIEW, where data items and attributes must match up.

The token primitives are the only kind of action statements used in the YACC input. It is conceivable that each entire command processor could be called as YACC action statements, but this is not the case. The design goal was to build command processors independent of the input mechanism, in this case, command syntax.

### 3.2.5 Database Access

#### A. Function:

This subfunction outlines the functional requirements of the database access used in the NDDL processor. All database access shall be for the ORACLE based CDM. All database access shall use the facilities of the IISS NDML wherever possible. The ORACLE SQL facilities may be substituted only if the NDML does not provide the needed functionality. Any use of ORACLE's SQL shall be written in C. This is because the necessary log on database area must be kept in a global area to avoid the database access routines requiring any DBMS specific interface parameters. This is to allow eventual conversion of all ORACLE database access routines to NDML and achieve DBMS independence for the NDDL processor.

#### B. CDM Requirements:

The ORACLE DBMS must be used due to previous decisions on the initial DBMS to host the CDM.

#### C. Processing:

1. ORACLE SQL will be used for the following requirements:
  - 1.1 SELECT where a Bill of Materials type recursive search is needed.
  - 1.2 UPDATE (or MODIFY) operations, when a key is to be modified.
2. COBOL embedded NDML will be used for all other database retrieval, verification, insert, delete and modify modules. A distinction is made between verification or look up modules and modules expected to retrieve more than one row.
  - 2.1 The verification type module shall be called with the search parameters as inputs and the database value(s) found on the single row as output. Very often a zero valued object number will be used as a "no-find" status.
  - 2.2 For routines expected to find many rows, the routine will receive the simple search parameters as input as before. Within the NDML ( and ), logic will be coded for processing each row. Very often calls to other routines which may process a single row will be made. If row processing is simple enough, calls are not necessary.



These requirements promote DBMS independence and simplicity of data structures common to more than one module.

3. For purposes of database concurrence and integrity, the logical unit of work shall be defined to be a single NDDL command execution. That is, the command is wholly executed with the results as expected by the user or none of the command is executed. Therefore, an NDDL command can be considered a transaction.

### 3.2.6 General Command Processing

#### A. Function

General Command Processing will handle such functions as pre-command initialization, control of parsing, control of forms input/output, and post command control of database commit or rollback. It must also control parsing of commands that cannot be executed due to previous errors. This subfunction will also provide facilities for CDM object numbering and number reuse. The subfunction must provide for generalized access to the parser data structures.

#### B. CDM Requirements

One table necessary for object numbering will be used. This is NEXT\_NUMBER which contains the next number to be used for each object type.

#### C. Processing

1. The user input form must be displayed the first time in forms mode.
2. Each command entered by the user on the input screen must be processed, skipping commands after an error is encountered.
3. For a completed command, a count of errors must be displayed.
4. If there were errors, the entire screen must be redisplayed. Also, the previous set of error messages need to be blanked out and a "no errors" message displayed. If the user asked to refresh and keep his command on the screen, this too must be done.
5. The current database, model, current output mode (file or screen) and current commit mode (whether automatic or natural) must also be kept on the screen.

6. If the user entered the quit key, a halt command must be generated and processed.
7. When a user has entered a command, the parser must be called. The return status of parsing must be examined.
8. If the command is to be processed, then a routine to effect the transfer of control to the proper command processor is executed.
9. After the individual command is executed, the current model and database must be established. If the command was successful, the database transactions are committed or, if unsuccessful, rolled back.
10. The CDM objects that shall be numbered follow. Each object type below has an object type number.

<u>OBJECT TYPE NUMBER</u>	<u>OBJECT TYPE</u>
1	MODEL
2	ENTITY
3	ATTRIBUTE
4	KEY CLASS
5	RELATION
6	TAG
7	DOMAIN
8	KEYWORD
9	VIEW
10	DATABASE
11	SET
12	DATA TYPE
13	DATA ITEM
14	DATA FIELD
15	RECORD

Objects are numbered to ease renaming and to allow a central place for storing object descriptions.

One subfunction exists to promote consistent handling of these numbers.

- 10.1 Get next number will obtain a new, unused number for an object being created. It must first search the list of available numbers for this object type. If one is found, it must be deleted from the list of reusable numbers. If one is not found, the next available number is retrieved from the CDM's NEXT\_NUMBER table. This number is incremented and updated in the NEXT\_NUMBER table.

11. Finally, this subfunction must supply routines that allow the command processor to access the lists of user command tokens built by the parser. These functions shall allow access to the first token on the list, the next token from the list and accessing a token from one list corresponding to another list (same row). The functions should return a count of tokens on the list and an end of list indicator.

### 3.2.7 Termination

#### A. Function

Termination will allow the NDDL processor to perform all termination requirements with other subsystems and software environments.

#### B. CDM Requirements

None

#### C. Processing:

1. Log off from ORACLE.
2. If the forms mode of input was used, use the FP service TERMFP.
3. Issue a call to send a finish up message to the DRS and to terminate NTM activities.

### 3.2.8 Individual Command Processing

The following subparagraphs outline the functional requirements of each command making up the NDDL. Consult the Table of Contents for a quick reference to a specific command.

#### 3.2.8.1 ALTER ALIAS - Switch the primary and alias names of a conceptual attribute or entity.

##### A. Function:

Alter Alias performs the following functions:

1. Changes the primary name of an attribute or entity to alias
2. Changes the alias name of an attribute or entity to primary

##### B. CDM Requirements:

1. The primary name of the attribute or entity must exist in the current model.

2. The alias name of the attribute or entity must exist in the current model.

C. Processing:

1. Alter Alias verifies that the primary and alias names to be switched exist in the current model.
2. If attribute names are being switched, the primary and alias entries in the ATTRIBUTE\_NAME table are updated.
3. If entity names are being switched, the primary and alias entries in the ENTITY\_NAME table are updated.

3.2.8.2 ALTER ATTRIBUTE - Alter a Conceptual Attribute

A. Function:

Alter Attribute performs the following functions:

1. Change a domain name for an attribute
2. Add keywords to an attribute
3. Drop keywords from an attribute
4. Alter ownership of an attribute from the old owner to new owner entity
5. Insert attribute as member of a specified key in new owner entity

B. CDM Requirements:

1. The attribute to be altered must exist in the current model.
2. The entity class who already owns the attribute must exist.
  - 2.1. If the attribute is a discriminator in category relation, the domain change will only occur if the new domain does not invalidate the discriminating values already being used in the category relation.
3. If the domain is to be modified, the new domain must exist.
4. If a keyword is to be dropped, it must exist.
5. If the ownership of the attribute is to be altered, the entity class to be the new owner must exist.

C. Processing:

1. Alter Attribute verifies that the attribute to be altered exists.
2. If the domain is to be changed, the existence of the new domain is verified and the ATTRIBUTE\_CLASS table is modified to contain the new domain number.
3. If a keyword is to be dropped, a check is performed to verify that the keyword is assigned to the attribute. If so, the keyword is deleted from the AC\_KEYWORD table.
4. If a keyword is to be added to an attribute, the keyword table is searched to determine whether the keyword exists. If not, the new keyword is inserted into the keyword table. The new keyword is then inserted into the AC\_KEYWORD table, if it did not already exist there.
5. Determine if the ownership of an attribute is to be altered from the old owner entity to a new owner entity.
  - 5.1 If the alter ownership clause is not present, exit command processing.
  - 5.2 Else, initialize local variables and tables.
6. Retrieve from the parser list the name of the new owner entity to which attribute ownership is to be altered.
  - 6.1 Verify that the new owner entity exists.
  - 6.2 If the entity does not exist issue an error message and exit command processing.
7. Retrieve information about the old owner entity.
  - 7.1 Retrieve the tag name, tag number, old entity number.
  - 7.2 Determine if the tag was a member of any key in the old owner. If the tag is a key member, move "yes" to KEY-FOUND-IN-OLD-OWNER-FLAG.
8. Determine if relations exist in which the new owner is the independent entity and the old owner is the dependent entity.

- 8.1 For each relation found, increment the MIGRATES-BACK-COUNTER.
9. Retrieve key migration information, if any, for the attribute use class in the old owner. Process as follows:
  - 9.1 Select the dependent entity, independent entity, relation, key and inherited attribute from the CDM tables INHERITED ATT USE LINK\_RELATION, and CATEGORY\_RELATION. For each row retrieved:
    - 9.1.1 Populate the internal table OLD\_TAG\_MIGRATION\_TBL.
    - 9.1.2 If the dependent entity found in the search is the same as the new owner entity
      - 9.1.2.1 Increment OLD-MIGRATES-TO-NEW-COUNTER
      - 9.1.2.2 Select all the key numbers if the inherited attribute is a member of the new owner entity. If the attribute is a key member in the new owner entity:
        - 9.1.2.2.1 Complete populating of OT-KC-NO-NEW of OLD\_TAG\_MIGRATION\_TBL with the key class numbers retrieved.
        - 9.1.2.2.2 Move "YES" to KEY-FOUND-IN-NEW-OWNER-FLAG
        - 9.1.2.2.3 Increment OT-KC-NO-USED for each key class number retrieved.
10. Determine all cases in which attribute ownership cannot be altered.
  - 10.1 If a dependency loop exists between the old owner entity and new owner entity, i.e. if MIGRATES-COUNTER and MIGRATES-BACK-COUNTER both are greater than zero, issue an error message and exist command processing.

- 10.2 If the attribute has migrated more than once from the old owner to the new owner, i.e. if it already exists as two or more inherited attributes in the new owner, only one of those inherited attributes may belong to any of the new owner's keys. If multiple keys are found, this is an error condition, and is determined by searching in the table populated earlier.
  - 10.2.1 Search OLD-TAG-MIGRATION\_TBL for entries where the dependent entity populated is the same as the new owner entity. If multiple entries are found, only one of these entries should have any keys associated with it (i.e., OT-KC-NO-USED > 0). If more than one entry has keys associated, issue an error message and exit command processing.
- 10.3 Compare the new entity number with the old entity number. If they are the same, issue an error message that the attribute is being altered to an entity where it is already owned.
- 10.4 If attribute is a key number that is used in a categorization, no ownership change is allowed. Keys that are used in category relation cannot be altered.
- 10.5 If attribute is a discriminator for a category relation, no ownership change is allowed. Ownership of discriminators cannot be altered.
- 10.6 Migrates clause cannot be specified if new entity is part of a category relation. Keys used in category relations cannot be altered.
11. Retrieve the key name, KC\_NAME\_LST, from the parser lists in which the attribute is to be placed as a key member. Also, Retrieve the link relation name, RC\_NAME\_LST, from the parser lists if any, through which the key can migrate back to the old owner.

Set FIRST\_TIME\_FLAG = 1
12. If KC\_NAME\_LST is empty
  - 12.1 If KEY\_FOUND\_IN\_NEW\_OWNER\_FLAG = "NO", initialize KC\_NAME\_LST = "NONKEY"

- 12.2 Continue processing at Step 14.
- 13. If KC\_NAME\_LST is not empty, i.e. "as member of" clause is specified
  - 13.1 Verify if this KC\_NAME\_LST has already been processed by searching KEY\_CLASS\_PROCESSED\_TBL. If the key has been processed
    - 13.1.1 Issue an error message and exit command processing
    - 13.1.2 Else, add this key to the KEY\_CLASS\_PROCESSED\_TBL table.
  - 13.2 Determine if this key exists in the new owner entity
    - 13.2.1 If the key does not exist
      - 13.2.1.1 Insert the key KC\_NAME\_LST for the new owner entity into the CDM table KEY\_CLASS. This is added as a primary key if a primary does not already exist. Else, it will be an alternate key.
      - 13.2.1.2 Continue processing at Step 14.
    - 13.2.2 If the key does exist in the new owner entity, i.e. KC\_NO\_NEW is not zero.
      - 13.2.2.1 Retrieve from the CDM tables COMPLETE\_RELATION and LINK\_RELATION, all key migrations of KC\_NO\_NEW.
      - 13.2.2.2 Populate the table NEW\_TAG\_MIGRATION\_TBL with the dependent entity number and the link relation name and number that KC\_NO\_NEW was migrated through.
      - 13.2.2.3 Each entry has an associated DEPENDENT-ENTITY-COUNT representing the total number of times the entity appears as dependent in all



new owner migrations. This count is used later to facilitate generation of unique tag names for inherited attributes.

14. If the attribute in the old owner is key and does not migrate from the old owner to the new owner or cannot migrate from the new owner back to the old owner, all key migrations of the attribute will be deleted. Process as follows:
  - 14.1 If the attribute is not key in the old owner continue processing at Step 17.
  - 14.2 If old key migrations have previously been preserved or deleted, i.e. FIRST-TIME-FLAG not = 1, continue processing at Step 17.
  - 14.3 If the attribute is key in the old owner and the MIGRATES-BACK-COUNTER is greater than zero continue at Step 17.
  - 14.4 If the MIGRATES-COUNTER is greater than zero
    - 14.4.1 Continue at Step 15
    - 14.4.2 Else continue processing at Step 16, which deletes all key migrations of the attribute in the old owner.
15. Since the attribute does migrate from the old owner entity to the new owner entity, key migrations, if any, of the new entity's key containing this inherited attribute may need to be preserved.
  - 15.1 If the user specified that the attribute is to be left as nonkey in the new owner (KC-NAME-LST equals "nonkey") the old owner's migrations need not be preserved. Continue processing at Step 16.
  - 15.2 Check whether the inherited attribute is a key class member in the new owner. If the attribute is key and the user did not specify a key class in the "as member" clause, the migrations of this attribute need to be preserved. Or, if the user specified the same key name where the attribute is already a key member, the migrations will again need to be preserved. Process as follows:
    - 15.2.1 If the attribute is not member of key, i.e., KEY-FOUND-IN-NEW-OWNER-FLAG = "No", no migrations need to be preserved; continue processing at Step 16.

- 15.2.2 If the attribute is a member of a key in the new owner, search for a row in the OLD-TAG-MIGRATION-TBL populated earlier where the OT-DEP-EC-NO is equal to the new entity number. Flag the row with an asterisk to be preserved if:
  - 15.2.1 No "as member" clause was specified (i.e., KC-NAME-LIST equal spaces)
  - 15.2.2 Or one of the KC numbers that the inherited attribute is a key member of (OT-KC-NO-NEW) is equal to the key number of the key specified in the "as member" clause (KC-NO-NEW)
  - 15.2.3 If none of the rows of the OLD-TAG-MIGRATION table were flagged with an asterisk, continue processing at Step 16; otherwise, there are migrations that need to be preserved as follows:
    - 15.2.3.1 Remove the OLD-TAG-NO from all keys in the old owner entity, by deleting from the CDM table KEY\_CLASS\_MEMBER and ATTRIBUTE\_USE\_CL.
    - 15.2.3.2 Search the OLD-TAG-MIGRATION-TBL. If the row has not been flagged with an asterisk, drop all migrations of the old owner to all dependent entities except where the dependent entity is the new owner. This is done by using a recursive search of the INHERITED\_ATT\_USE\_CDM table and matching the OLD-TAG-NO with the KCM TAG NO and the relation number from the OLD-TAG-MIGRATION-TBL (OT-RC-NO) with the RC NO. Delete the migrations from the CDM Tables

INHERITED ATT USE,  
KEY\_CLASS\_MEMBER, and  
ATTRIBUTE\_USE CL where  
there is a match.

- 15.2.3.3 If the row has been flagged with an asterisk, convert the inherited attribute of the new owner found in the OLD-TAG-MIGRATION-TBL (OT-TAG-NO-NEW) to owned by deleting its entry in CDM Table INHERITED ATT USE. If an "as member of key" clause was specified (i.e., KC-NAME-LIST not equal to spaces) drop the migrations of all the keys in the new owner except the one specified in the clause. This is done by using a recursive search of the INHERITED ATT USE CDM Table and matching the new tag number (OT-TAG-NO-NEW) found in the OLD-TAG-MIGRATION-TBL with KCM-TAG-NO and the key number from the OLD-TAG-MIGRATION-TBL with KC\_NO. Delete the migrations from the CDM Tables INHERITED ATT USE, KEY\_CLASS\_MEMBER and ATTRIBUTE\_USE CL where there is a match.
- 15.2.3.4 Issue a warning message that some keys may now be "SUSPECT".
- 15.2.3.5 Continue processing at Step 22.

- 16. Process deleting the old owner's migrations as follows:
  - 16.1 Delete the attribute as member of all keys in the old owner entity by deleting from the CDM table KEY\_CLASS\_MEMBER.
  - 16.2 Delete all migrations down the entire chain using a recursive search accessing CDM tables INHERITED ATT USE, KEY\_CLASS\_MEMBER and ATTRIBUTE\_USE\_CL.

- 16.3 Issue a warning message that some keys may now be "SUSPECT".
17. Using the table created in Step 13.2.2, NEW-TAG-MIGRATION-TBL, which contains migration information of the key specified by the user for the new owner entity, create attribute use and inherited attribute use occurrences for the attribute whose ownership is being altered. Process as follows:
  - 17.1 If table is empty, continue processing at Step 19.
  - 17.2 If this is not the first time through, continue processing at Step 18.
  - 17.3 If a complete relation has been specified in case of multiple migrations from the new owner back to the old owner entity (i.e., RC-NAME-LST is not empty):
    - 17.3.1 Verify the link relation specified does exist in the NEW-TAG-MIGRATION-TBL and that the dependent entity is the old owner entity. If this check fails:
      - 17.3.1.1 Issue an error message and exit command processing.
      - 17.3.1.2 Else, continue processing at Step 18.
  - 17.4 If a complete relation is not specified (i.e., RC-NAME-LST is empty) search through the NEW-TAG-MIGRATION-TBL to locate an entry where the dependent entity is the old owner entity. Move this entry's link relation to RC-NAME-LST. This processing is done to establish the same tag name for the first inherited attribute created and suffixed tag names for all the remaining inherited attributes in the same dependent entity.
18. For each entry found in NEW-TAG-MIGRATION-TBL, process as follows:
  - 18.1 If the attribute has already been established as an attribute use in the new owner entity and inserted as a key member in the key specified by the user, continue processing at Step 18.5.

- 18.2 If the attribute use has previously been established in the new owner (i.e., FIRST-TIME-FLAG NOT = 1 but the key occurrence has not been created, insert an occurrence in the CDM table KEY\_CLASS\_MEMBER. Continue processing at Step 18.4.
- 18.3 Since this is the first time through, establish the attribute use in the new owner, and insert a key class occurrence in the key specified.
  - 18.3.1 If the MIGRATES-BACK-COUNTER > zero (i.e., the attribute migrates back from the new owner to the old owner) insert an attribute use occurrence in the new owner entity. Generate a new tag number and insert into the CDM table ATTRIBUTE\_USE\_CL. Move the new tag number to the variable KCM-TAG-NO to reflect that the AUC has been established in the new owner.
  - 18.3.2 If the MIGRATES-BACK-COUNTER = 0 (i.e., the attribute does not migrate back to the old owner) modify the attribute use occurrence. Update the CDM table ATTRIBUTE\_USE\_CL to reflect the tag is an attribute use of the new owner entity, instead of the old owner. Move this TAG-NO-OLD to KCM-TAG-NO to establish the AUC has been established in the new owner.
- 18.4 Set a flag to denote the key member has been created.
- 18.5 If the attribute migrates back from the new owner to the old owner entity, the attribute is converted from owned to inherited. This conversion is done only once, and uses the information populated in RC-NAME-LST. For every other migration of the new owner to all other dependent entities, an attribute use and inherited attribute use occurrence pair is created. Process as follows:
  - 18.5.1 If the dependent entity from the table is the same as EC-NO-OLD and FIRST-TIME-FLAG = 1, and the link relation from the table is the same as RC-NAME-LST

18.5.1.1 Insert an inherited attribute occurrence in the CDM table INHERITED\_ATT\_USE

18.5.1.2 Else, create an inherited attribute use and attribute use pair by performing Step 23.

18.5.2 Continue processing at Step 22.

19. Process the situation, if any, where the new owner entity's new key can migrate back to the old owner entity through a user specified incomplete relation. Process as follows:

19.1 If KC-NAME-LST is not empty (i.e., a key has been specified) and RC-NAME-LST is not empty (i.e., a link relation has been specified).

19.1.1 Verify the link relation specified exists with new owner as the independent entity and the old owner as the dependent entity. If the verification fails, issue an error message and exit command processing.

19.1.2 Verify the link relation specified is incomplete. If an incomplete relation has not been specified, issue an error message and exit command processing.

19.1.3 Create a complete relation occurrence by inserting into the CDM table COMPLETE\_RELATION, specifying the new key migrates through the specified link relation.

19.1.4 If the attribute use occurrence has not previously been established (i.e., FIRST-TIME-FLAG = 1):

19.1.4.1 Generate a unique tag number. Move this tag number to KCM-TAG-NO, to indicate the attribute use for the new owner entity has been established.

19.1.4.2 Create an attribute use occurrence for the new owner by inserting into the CDM table ATTRIBUTE\_USE\_CL.

- 19.1.4.3 Create an inherited attribute use occurrence to convert the attribute in the old owner from owned to inherited. Insert into the CDM table INHERITED\_ATT\_USE.
- 19.1.4.4 Place the attribute as member of the user specified key in the new owner entity. Insert into CDM table KEY\_CLASS\_MEMBER.
- 19.1.5 Else (attribute use has been established for new owner)
  - 19.1.5.1 Generate an attribute use and inherited attribute use occurrence for the old owner entity. Perform Step 23.
  - 19.1.5.2 Place the attribute as member of the user specified key, if not already placed, by inserting into the CDM table KEY\_CLASS\_MEMBER.
- 19.1.6 Continue processing at Step 22.
- 20. Determine if the attribute is to be placed as key in the new owner entity.
  - 20.1 If KC-NAME-LST is not empty (i.e., user has specified a key):
    - 20.1.1 Insert the attribute as member of key specified in the new owner entity, by inserting into the CDM table KEY\_CLASS\_MEMBER.
    - 20.1.2 Establish the attribute use occurrence for the new owner, if not previously established, by updating the CDM table ATTRIBUTE\_USE\_CL to reflect that the attribute now is an attribute use in the new owner.
    - 20.1.3 Continue processing at Step 22.
- 21. If the attribute is to be placed as nonkey in the new owner entity:

- 21.1 Modify the attribute use occurrence by updating the CDM table ATTRIBUTE\_USE\_CL to reflect that the attribute is now an attribute use in the new owner entity, instead of the old owner entity.
- 22. Prepare for the next iteration of the "as member" clause. Initialize the parser variables. Retrieve from the parser lists the next "as member" clause.
  - 22.1 If as "NONKEY" specified, issue an error message.
  - 22.2 If KC-NAME-LST is spaces
    - 22.2.1 Update the CDM table OWNED\_ATTRIBUTE to reflect the attribute is now owned by the new owner entity.
    - 22.2.2 Delete any empty keys from the CDM table KEY\_CLASS and incomplete link relations from the CDM table COMPLETE\_RELATION that resulted from altering the attributes ownership.
    - 22.2.3 EXIT program.
  - 22.3 If a key name has been specified
    - 22.3.1 Retrieve from the parser list, if any, the corresponding link relation name, RC-NAME-LST.
    - 22.3.2 Initialize the NEW-TAG-MIGRATION-TBL. Set a flag to indicate the key member occurrence has not been created. Increment FIRST-TIME-FLAG.
    - 22.3.3 Continue processing with this key class name specified, at Step 13.
- 23. Create an Attribute Use and Inherited Attribute Use pair for the dependent entity that the attribute from the new owner will migrate to. Special processing is required when the new owner migrates back to the old owner multiple times. For these cases suffixes will be appended to the old owner's tag name. Process as follows:
  - 23.1 If FIRST-TIME-FLAG = 1 (i.e., this is the first time through) and the dependent entity being processed is the same as the old owner entity:



- 23.1.1 If the link relation is specified by the user (i.e. RC-NAME-LST is the same as the link relation being processed) no suffixing is performed.
- 23.1.2 Else suffix the tag name, by appending a circumflex character and a number to the attribute (role) name.
- 23.1.3 Continue processing at Step 23.3.
- 23.2 If this is not the first time through (i.e., FIRST-TIME-FLAG > 1) verify if the tag name exists in the dependent entity being processed.
  - 23.2.1 If the tag name exists, generate a unique inherited attribute (role) name by suffixing.
- 23.3 Generate a unique tag number for the inherited attribute in the dependent entity.
- 23.4 Create an attribute use occurrence by inserting into the CDM table ATTRIBUTE\_USE\_CL.
- 23.5 Create an inherited attribute use occurrence in the dependent entity and pair it with the attribute use occurrence in the new owner entity.

3.2.8.3 ALTER CATEGORY - Alter a conceptual category relation.

A. Function:

Alter Category performs any or all of the following functions:

- 1. Add/drop category entities for the category relation being altered.
- 2. Add/drop associated keywords for the category relation being altered.
- 3. Change relation from complete to incomplete or incomplete to complete.

B. CDM Requirements:

- 1. The category relation must exist.
- 2. The category entity must exist.

3. Added category entities can not have any keys and their liter value must be valid in the domain of the discriminating attribute.

C. Processing:

1. Incomplete/complete

- 1.1 An incomplete category can be changed to complete or vice versa. If the category relation is complete, a check will be made to make sure that there are at least two category entities.

2. Add

- 2.1 The category relation can be adjusted by adding entities.
- 2.2 The primary key of the generic entity will be migrated as the primary key of the category entities. A new key number will be assigned to the category entity. The key name of the generic entity's primary key will be inserted into KEY\_CLASS with the new key\_number.
- 2.3 The value for the discriminator in the category must be unique for each of the category entities in the category relation and be a valid specific value in the domain of the discriminator.
- 2.4 If a keyword is to be added to the category relation, the keyword table is searched to determine whether the keyword exists. If it does not exist, the new keyword is inserted into the keyword table. The new keyword is then associated with the category relation.

3. Drop

- 3.1 The category relation can be adjusted by deleting entities.
- 3.2 The attribute use and inherited attribute use originally created for each key member migrated to the category entity are deleted. Also, the attribute use and inherited attribute use created for each key member migrated to lower level dependent entities are also deleted. Then all keys and complete relations that become empty due to the deletion of migrated key members are deleted.

- 3.3 The category entity is deleted from CATEGORY MEMBER. If the category entity being dropped is the only category member in the category relation, the category relation is dropped with complete ripple down.
- 3.4 If keywords are specified, the relation concurrence is also deleted.

3.2.8.4 ALTER DATABASE - Alter the definition of a database in CDM internal schema.

A. Function:

The command allows the user to alter a database definition in the CDM by changing password information, schema information, database location information, PSB information, null information, ntm directory information, or by adding and/or deleting files or areas.

B. CDM Requirements:

The database to be altered must exist in the CDM. If the host is to be altered, it must be defined for the DBMS for which the database is defined.

C. Processing:

- 1. Verify the existence of the database to be altered. If it does not exist, flag a user error. If no other clauses are specified, the database specified is made the current database.
- 2. If the host is to be changed:
  - 2.1 Verify that the host exists. If it does not exist, flag a user error.
  - 2.2 Otherwise, modify the database occurrence.
- 3. If the DBMS is IMS:
  - 3.1 Check if the PSB clause is present in the Command.
  - 3.2 If it is, modify the PSB occurrence.
- 4. If the DBMS is ORACLE:
  - 4.1 Check if the password clause is present in the command.
  - 4.2 If it is, modify the database password occurrence.

5. If the DBMS is CODASYL (IDMS, IDS-II, or VAX-11):
  - 5.1 Check if the schema clause is present in command.
    - 5.1.1 If it is:
      - 5.1.1.1 Modify the database schema occurrence.
      - 5.1.1.2 Check if location clause is present. If it is not, flag a user error. If it is, modify the database location occurrence.
  - 5.2 If areas are to be added:
    - 5.2.1 Verify that the area does not exist. If it does exist, flag a user error.
    - 5.2.2 Otherwise, insert the database/area occurrence.
  - 5.3 If areas are to be dropped:
    - 5.3.1 Verify that the area exists. If it does not exist, flag a user error.
    - 5.3.2 Check if the area specified is the last remaining area associated with the database. If it is, flag a user error.
    - 5.3.3 Otherwise, delete the database/area occurrence.
6. If nulls (character or integer) are to be changed, the occurrence in the CDML table DATA\_BASE will be modified to the new null value specified.
7. If ntm directory is to be changed, the occurrence in the CDM table DATA\_BASE will be modified to the new ntm directory value specified.

3.2.8.5 ALTER DBMS - Change a Data Base/Host association in the CDM.

A. Function:

Alter DBMS performs the following function:

1. Alter DBMS allows the user to add and delete host associations to a DBMS definition and change DBMS type code.

B. CDM Requirements:

1. The DBMS description must exist in the CDM.
2. The host being added must not already be associated with the DBMS.
3. The host being dropped must be associated with the DBMS.
4. In both cases, the host must be in the CDM.
5. The DBMS must not exist on host for host to be dropped.

C. Processing:

1. The CDM is checked to see if DBMS definition exists.
2. If DBMS definition exists:
  - 2.1 If host is to be added:
    - 2.1.1 The CDM is checked to see if the host exists.
    - 2.1.2 If host exists:
      - 2.1.2.1 The CDM is checked to see if host is already associated with the DBMS.
      - 2.1.2.2 If host already associated with database:
        - 2.1.2.2.1 An error message is issued.
        - 2.1.2.2.2 Continue processing the next host.
      - 2.1.2.3 If host not associated with DBMS:
        - 2.1.2.3.1 The host/database associated is added to the CDM.
  - 2.2 If a host is to be dropped:
    - 2.2.1 The CDM is checked to see if the host exists.
    - 2.2.2 If host exists:
      - 2.2.2.1 The CDM is checked to see if host is associated with the DBMS.

2.2.2.2 If host is associated with database:

2.2.2.2.1 The host/database association is dropped from the CDM.

2.2.2.3 If host is not associated with DBMS:

2.2.2.3.1 An error message is issued.

2.2.2.3.2 Continue processing the next host.

2.3 If DBMS type code is to be changed:

2.3.1 The DBMS type code is updated with the new code.

3.2.8.6 ALTER DOMAIN - Alter the definition of a domain in the CDM.

A. Function:

Alter Domain allows the NDDL user to perform the following modifications to the definitions of existing domains:

1. Addition of non-standard data types
2. Deletion of non-standard data types
3. Changing the meta data description of existing data types of the domain (i.e. changing the type, size and number of decimal digits)
4. Promoting a non-standard data type to standard, converting the former standard to non-standard
5. Add valid values
6. Add valid ranges
7. Delete existing values
8. Delete existing ranges
9. Delete all existing values
10. Delete all existing ranges

B. CDM Requirements:

The domain name referenced must be found in the CDM. Any data types to be dropped or altered must already be defined for the domain. Any data types to be added must not already be defined anywhere else in the CDM. There must always be a standard data type for the domain, i.e. the current standard data type cannot be dropped.

C. Processing:

1. The user entered domain name to be altered is verified to be in the CDM. With this number, each of the data type changes can be processed.
2. For each user data type request, determine if it's an ADD, DROP or ALTER.
  - 2.1 For an ADD or ALTER, the legal data types are checked. They must be SIGNED, UNSIGNED, INTEGER, FLOAT, PACKED, or CHARACTER. The user does not enter this for a DROP.
  - 2.2 For an ADD and optionally for an ALTER, the size and number of decimal digits are checked. They must both be numeric and the decimal digits must not exceed the size. They are not specified for a DROP.
  - 2.3 If the user has requested to alter a data type:
    - 2.3.1 The data type is verified to exist for the domain and be either standard or non-standard. The user is warned if it cannot be found. If it is a standard, only type, size and number of decimals may be changed and Step 3.3.1 must be performed.
    - 2.3.2 For standard data type, or a change to only the type, size and number of decimals, this change is recorded in the USER\_DEF\_DATA\_TYPE table. Step 3.3.1 must be performed.
    - 2.3.3 Otherwise, the user has requested a switch from non-standard to standard. In this case, the old standard data type name is fetched and is changed to non-standard and the name specified by the user is changed to become the standard data type. Step 3.3.1 must be performed.

- 2.4 If the user has requested to drop a data type:
  - 2.4.1 The data type name is verified to be in the domain.
  - 2.4.2 If the data type is found to be standard, the user is informed that it cannot be dropped.
  - 2.4.3 If the data type is non-standard, then any usage of this data type is checked.
    - 2.4.3.1 The database is searched to find any references by a data field.
    - 2.4.3.2 The database is searched to find any references by a data item.
    - 2.4.3.3 The database is searched to find any references by an attribute class. This is probably unnecessary since it has been checked to be non-standard in 2.4.2 and only standards can map to attributes.
  - 2.4.4 If it was not referenced elsewhere, the data type and its description text is deleted.
- 2.5 If the user has requested to add a data type:
  - 2.5.1 For a standard data type, the database is searched for an existing standard data type. If one is not found:
    - 2.5.1.1 The data type name is checked to see that it was not used for some other domain. If not,
    - 2.5.1.2 The data type is checked to be valid by using a database look up in the table DATA\_TYPE. If ok,
    - 2.5.1.3 The data type information is stored in the CDM table USER\_DEF\_DATA\_TYPE, with a unique object number, as standard for this domain.
  - 2.5.2 For a non-standard data type, the data type name is checked as in step 2.5.1.1.



- 2.5.2.1 The data type is checked as in 2.5.1.2.
- 2.5.2.2 The data type information is stored in the CDM table USER\_DEF\_DATA\_TYPE with a unique object number as non-standard (or "USER") for this domain.
- 3. For each value and/or range request, determine if it is an ADD, DROP, or DROP ALL.
  - 3.1 For a DROP ALL, all values and/or ranges are deleted from the DOMAIN\_VALUE and/or DOMAIN\_RANGE tables.
  - 3.2 For a DROP, the specified values and/or ranges on the request are deleted from the database.
  - 3.3 For an ADD, the specified values and/or ranges are inserted into the database.
    - 3.3.1 All the values and/or ranges on the database must be retrieved and validated against the standard data type information since new values and/or ranges have been added or the standard data type information has been changed.
- 4. For any value and/or range request, a new verification module is created.

3.2.8.7 ALTER ENTITY - Alter a conceptual entity

A. Function:

Alter Entity performs the following functions:

- 1. Add/drop keys for the entity being altered
- 2. Add/drop owned attributes for the entity being altered
- 3. Add/drop associated keywords for the entity being altered
- 4. Rename the role (tag) name of the specified attribute use in the entity being altered
- 5. Rename a key name in the entity being altered
- 6. Alter a key type from alternate to primary in the entity being altered.

7. Add additional members to a specific key of the entity being altered
8. Drop existing members of a specific key of the entity being altered
9. Substitute members of a specific key with new key members, and continue to preserve the migrations of the old members for the substituted attributes

B. CDM Requirements:

1. The entity to be altered must exist in the current model.
2. If owned attributes are to be added, the attribute must exist in the current model. If owned attributes are to be dropped, they must be owned by the entity being altered.
3. If a key is to be dropped, it must be a key for the entity being altered.
4. If a keyword is to be dropped, it must be associated with the entity being altered.
5. The role name being renamed must exist in the entity being altered.
6. The key being altered must exist in the specified entity.
7. The tag names being added and dropped must exist in the entity being altered.
8. Identify candidate keys as being the primary key or alternate keys.

C. Processing:

1. The Alter Entity process verifies that the entity to be altered exists in the current model. If it does not exist, an error is issued and processing is terminated.
2. If keys are being added, a new occurrence of key is added to the entity. The key will be a primary or alternate key as specified. If it is not specified, the key will be the primary key if a primary key does not exist. Else it will be an alternate key. A new occurrence of attribute use is created for each attribute named as part of the key, if one does not exist for the entity. A new occurrence of key class members is created for the

entity for each attribute named in the key clause. If entity being altered is part of a category relation, only alternate keys may be added.

3. If owned attributes are being added for the entity, the existence of the attribute class is determined within the current model. If the attribute does not exist, an error is issued and processing is terminated. If they do exist, each attribute is created as an owned attribute and attribute use for the entity.
4. If a keyword is to be added to the entity, the keyword table is searched to determine whether the keyword exists. If it does not exist, the new keyword is inserted into the keyword table. The new keyword is then associated with the entity.
5. If keys are being dropped, the existence of the key for the entity being altered is determined. If it does exist, the key and attributes inherited via the migrated keys and key members are dropped. If the key being dropped is from a complete relation, the complete relation is also deleted. Identity being altered is part of a category relation, only alternate keys can be dropped.
6. If owned attributes are being dropped, they are verified to determine if they belong to the entity being altered. If they belong to the entity, the owned attribute occurrence and the attribute use and any tag constraints for each attribute named is deleted from the entity being altered. If mappings exist for the attribute, an error is issued, processing is terminated and no attribute is dropped. If owned attribute to be dropped is a discriminating attribute in a category relation, the category relation is dropped also with complete ripple down.
7. If keywords are to be dropped, a check is performed to verify that the keyword is associated with the entity being altered. If so, the keyword association is deleted from the entity. If any associations exist for the keyword, an error is issued, processing is terminated and no keywords are dropped.
8. If a tag name is to be renamed:
  - 8.1 Verify that the old tag name is an attribute use in the specified entity.
  - 8.2 Verify the new tag name does not previously exist as an attribute use in the specified entity.

- 8.3 If either of the above conditions are not satisfied, issue an error message and continue with the next iteration.
- 8.4 If both the above conditions are true, update the CDM table ATTRIBUTE\_USE\_CL, with the new tag name.
- 8.5 Continue processing with the next tag name to be renamed.
9. If the key name is to be renamed:
  - 9.1 Verify that the old key name does exist in the specified entity.
  - 9.2 Verify that the new key name does not previously exist in the specified entity.
  - 9.3 If either of the above conditions are not satisfied, issue an error message and continue with the next iteration.
  - 9.4 If both the above conditions are true, update the CDM table KEY\_CLASS with the new key name.
10. Determine if the key members of a specified key of the entity being altered are to be added, dropped or substituted and branch to the appropriate case. If entity is part of a category relation, no altering of primary keys is allowed.
11. If an attribute use is to be added as a member of the key:
  - 11.1 Verify that the tag exists as an attribute use of the specified entity.
  - 11.2 Verify that the tag does not already exist as member of the specified key.
  - 11.3 If either of the above conditions are not satisfied, issue an error message and continue with the next iteration.
  - 11.4 If both the above conditions are true, create an occurrence of this tag as member of the specified key, by inserting into the CDM table KEY\_CLASS\_MEMBER.
  - 11.5 Select all the dependent entities that the specified key migrated to through a link relation. For each dependent entity found, add this tag as a non-key attribute. Process as follows:

- 11.5.1 Verify that this tag name is currently not an attribute use in this dependent entity.
- 11.5.2 If the tag already exists, issue an error message and exit command processing.
- 11.5.3 If the tag does not exist in this entity, create an attribute use occurrence for this tag by inserting into the CDM table ATTRIBUTE\_USE\_CLASS.
- 11.5.4 Also, create an inherited attribute class occurrence for this tag by inserting into the CDM table INHERITED\_ATT\_USE.
- 11.6 Continue processing with the next key member to be added.
- 12. If the attribute use is to be removed as a key member of a specified key:
  - 12.1 Verify that the tag does exist as an attribute use of the entity being altered.
  - 12.2 Verify that the tag currently is a member of the key being altered.
  - 12.3 Verify that this tag is not the only key member of the key being altered.
  - 12.4 If either of the above conditions are not satisfied, issue an error message and exit command processing.
  - 12.5 If all three conditions are satisfied, remove this tag as a key member by deleting an occurrence from CDM table KEY\_CLASS\_MEMBER.
  - 12.6 Also, delete all inherited attributes of this tag. Delete the inherited attribute occurrences from the CDM tables INHERITED\_ATT\_USE, ATTRIBUTE\_USE\_CL, KEY\_CLASS\_MEMBER.
  - 12.7 Continue processing with the next key member to be dropped.
- 13. After determining that key members are to be substituted in the entity being altered:
  - 13.1 Retrieve from the parser lists the tag names to be added as key members of the specified key.

13.2 For each tag name retrieved:

- 13.2.1 Verify the tag exists in the entity being altered.
- 13.2.2 Verify this tag is not a member of the key being altered.
- 13.2.3 If either of the above conditions are not satisfied, issue an error message and exit command processing.
- 13.2.4 If both the above conditions are true, populate the internal table ADD-TAG-LIST.

13.3 Retrieve from the parser lists the tag names to be removed as key members of the specified key.

13.4 For each tag name retrieved:

- 13.4.1 Verify that this tag exists in the entity being altered.
- 13.4.2 Verify that this tag is a member in the key being altered.
- 13.4.3 If either of the above conditions are not satisfied, issue an error message and exit command processing.
- 13.4.4 If both the above conditions are true, populate the internal table DROP-TAG-LIST.

13.5 Check that the number of entries in ADD-TAG-LIST is equal in number to those entries in DROP-TAG-LIST. If the number of tags being added does not equal the number of tags being dropped from the key, issue an error message, and exit command processing.

13.6 For each tag in the DROP-TAG-LIST select all its inherited attributes. For each inherited attribute number and dependent entity selected, process as follows:

- 13.6.1 Verify that the corresponding tag name, add-tag-name in the ADD-TAG-LIST does not exist as an attribute use in the dependent entity that the drop-tag-name migrated to.

- 13.6.2 If a duplicate tag name is found, issue an error message and exit command processing.
  - 13.6.3 If the tag-name being added will not result in a duplicate name being created, populate another internal table UPD-TAG-LIST with the inherited attribute number selected and in Step 13.6 the tag name to be substituted for it from the corresponding entry in ADD-TAG-LIST.
  - 13.6.4 Continue processing with the next entry in the DROP-TAG-LIST.
14. Using the three internal tables that were populated in the previous steps, (ADD-TAG-LIST, DROP-TAG-LIST and UPD-TAG-LIST), and the entity and key being altered, the CDM tables will be modified. Process as follows:
- 14.1 For each entry in the UPD-TAG-LIST, rename the inherited attribute with its new role name. Update the CDM table ATTRIBUTE\_USE CL, setting the drop-tag-name to become the add-tag-name.
  - 14.2 For each corresponding entry in the ADD-TAG-LIST and DROP-TAG-LIST, update the CDM table KEY\_CLASS\_MEMBER, to reflect that the key being altered has a new key member. Substitute the drop-tag-number with the add-tag-number.
  - 14.3 For each corresponding entry in the ADD-TAG-LIST and DROP-TAG-LIST, update the CDM table INHERITED\_ATT\_USE to reflect that the inherited attributes of drop-tag-name are not inherited from add-tag-name. Substitute the drop-tag-number with add-tag-number.
15. If the type of key is being modified, it may be altered from alternate to primary only. The previous primary key is then made an alternate key. If the entity being altered is part of a category relation, no key type change is allowed for the alternate keys.
- 3.2.8.8 ALTER FIELD - Alters an existing data field in a previously defined record.

A. Function:

Alter field performs the following functions:

1. Verifies that the field, record and database exist
2. Alters a data type name or converts to null in case of a group data field
3. Alters a field to be repeating by adding an occurs clause or vice-versa
4. Adds, alters or drops the depending on field
5. Adds, alters or drops the redefine field
6. Adds or drops the indexed by field
7. Converts a non-key to key, or convert a key to be unique or duplicates allowed key
8. Specifies whether a field is known or unknown to the DBMS

B. CDM Requirements:

1. The database/PCB and record-type must be previously defined.
2. The field name must exist.
3. The depending on field name must exist.
4. The field being redefined name must exist.

C. Processing:

1. The CDM is checked to see if the database exists. If it has not been defined, issue an error message and exit command processing.
2. The CDM is checked to see if the record type exists. If it has not been defined, issue an error message and exit command processing.
3. The CDM is checked to see if the field name exists. If it has not been defined, issue an error message and exit command processing.
4. All the existing data fields belonging to the record are retrieved into a temporary structure. Syntax and semantic checks are performed on the data field being altered to ensure the integrity of the record layout is maintained.
5. Additional processing is required in case the indexed-by field is altered. If the user specifies that the field is "not indexed", its existing index



will be deleted after verifying that the index field is not mapped. If the user specifies the field is to be indexed, there are two possibilities. If the indexed by field name exists, this field now has to be updated by setting a flag indicating it is an index. However, if the indexed by field was not previously defined, a new data field has to be created and inserted with a numeric index data type name and an index indicator of "G".

6. Now the following checks are performed on the field being altered:
  - 6.1 An ORACLE DBMS does not support repeating fields, COBOL type indexes, re-definition or keys.
  - 6.2 Re-definition of TOTAL fields must be specified as a single field.
  - 6.3 A repeating field or repeating group cannot be key.
  - 6.4 The indexed by field cannot re-define, or be a component or group or repeating field or key.
  - 6.5 The data type name for an indexed field and a depending-on field must be numeric. Additionally, all elementary items must have a data type and the data type name entered by the user must have been previously defined in USER\_DEF\_DATA\_TYPE.
  - 6.6 If the indexed-by field exists in the record, it must appear before the field it is indexing.
  - 6.7 If it is a repeating data field, the number of times it occurs must be greater than one. Additionally, the depending on field must be previously defined.
  - 6.8 If the field is re-defined, the re-defined field must be previously defined. Additionally, the re-defines field must be key if the re-defined field is key.
  - 6.9 The depending-on field cannot be group or repeating or an index.
  - 6.10 If a component data field is key, one of its subcomponents cannot also be a unique key. It may be a duplicate or secondary key.
7. If the record layout conforms with the checks performed and no errors occurred, an insert or

delete or modify (as the case may be) is performed to the indexed field. The data field itself is modified.

8. If errors occurred, the user is informed and no updates to the CDM are performed.

3.2.8.9 ALTER HOST - Change Host/Database Management System associated in the CDM.

A. Function:

Alter Host performs the following function:

1. Alter Host allows the user to add and delete DBMS associations to the specified host.

B. CDM Requirements:

1. The HOST and DBMS must be previously defined.

C. Processing:

1. The CDM is checked to see if host is defined.
2. If the host is not defined:
  - 2.1 Issue an error message.
  - 2.2 Exit command processing.
3. If the Host is defined:
  - 3.1 If DBMS association is to be added:
    - 3.1.1 The CDM is checked to see if the DBMS is defined.
    - 3.1.2 If the DBMS is not defined, issue an error message and continue processing the next DBMS definition.
    - 3.1.3 If the DBMS is defined, the DBMS association with the host is added to the CDM.
  - 3.2 If a DBMS association is to be dropped:
    - 3.2.1 The CDM is checked to see if the DBMS is defined.
    - 3.2.2 If the DBMS is not defined, issue an error message and continue processing the next DBMS definition.

3.2.3 If the DBMS is defined, the DBMS/Host association is deleted from the CDM.

3.2.8.10 ALTER MAP - Modify a CS-IS Mapping

A. Function:

Alter Map allows the user to perform the following functions:

1. Switch a preference mapping along with its map and category.
2. Alter the map and category of a tag for a preference.
3. Allow the tag to be mapped for the same preference to a data field.
4. Allow the tag to be mapped for the same preference to an additional set.
5. Allow an additional record to be mapped to an entity.
6. Alter the value in tag to set mapping for a stated preference.
7. Delete a tag to data field mapping for a stated preference.
8. Delete a tag to set mapping for a stated preference.
9. Delete a relation to set mapping.
10. Delete a record mapped to an entity.
11. Alter the distributed rules for an entity.

B. CDM Requirements:

The map to be altered must exist in the CDM. In addition, all database names, record names, data field names and set names referenced must exist in the CDM.

C. Processing:

1. Determine if the user wants to alter an entity, AUC, link relation, or category relation mapping.
  - 1.1 If entity:
    - 1.1.1 Verify that the entity name exists and that the entity to record mapping exists in the CDM.

1.1.2 Retrieve update and retrieval rules from the parser. If they were input, update these rules in the DISTRIBUTED\_RULES table of the CDM.

1.2 If AUC:

1.2.1 Verify that the entity name and tag name exist in the CDM.

1.2.2 Check if this AUC is involved in a horizontal partition and set flag to indicate this fact.

1.2.3 Retrieve the preference number(s) from the parser. Verify if the tag is mapped for the stated preference.

1.2.4 If the mapping preference is to be altered, retrieve the new preference number from the parser.

1.2.4.1 If the new preference number specified has a mapping associated with it, the mapping preferences are switched along with the map and map category.

1.2.4.2 If the new preference number specified has no associated mapping:

1.2.4.2.1 If the mapping preference to be altered is 1, issue an error message and exit command processing.

1.2.4.2.2 Else change the old preference number to the new one, then validate that the new preference number corresponds to the old map category (i.e. "ACTIVE" has preference numbers of 1-50 and "PASSIVE" has preference numbers of 51-99).

- 1.2.4.3 If the mapping preference is altered, no other function may be performed.
    - 1.2.5 If map and/or map category are to be altered, validate the map category and preference number combination. Alter AUC\_IS\_MAPPING table with the new map and/or map category.
  - 1.3 If Relation:
    - 1.3.1 Retrieve relation name and entity name from parser and verify that the relation exists, either a category or link relation.
- 2. Retrieve the ALTER MAP option from the parser. If ALTER ADD or ALTER DROP, the command can refer to a field, record or set; if ALTER ALTER the command refers to a set only.
  - 2.1 For ALTER ADD AUC to data field mapping, the following rules apply:
    - 2.1.1 The AUC must not have been previously mapped to a set for the stated PREF\_NUMB.
    - 2.1.2 The AUC must not have been previously mapped to a data field unless the entity is horizontally partitioned.
    - 2.1.3 If the entity is horizontally partitioned, verify the tag is being mapped to a data field that exists in a fragment of the partitioned entity.
    - 2.1.4 If all the above conditions are satisfied, create an occurrence in the CDM tables AUC\_IS\_MAPPING and PROJECT\_DATA\_FIELD.
  - 2.2 For ALTER DROP AUC to data field mapping, the following rules apply:
    - 2.2.1 Dropping a tag to data field deletes a mapping of one fragment of the partitioned entity.
    - 2.2.2 After the drop is performed, at least one data field must remain mapped to the AUC.

- 2.2.3 If the above condition is satisfied, the stated data field mapping is deleted from the CDM tables AUC\_IS\_MAPPING and PROJECT\_DATA\_FIELD.
- 2.3 For ALTER ADD AUC to set mapping, the following rules apply:
  - 2.3.1 A data field mapping must not exist for the AUC for stated PREF\_NUMB unless the entity is horizontally partitioned.
  - 2.3.2 The sets to be mapped to must be single member sets.
  - 2.3.3 The set to be mapped to must not have been previously mapped from a link relation, category relation, or another AUC.
  - 2.3.4 All AUC to set maps must map to the same database for a particular AUC unless the entity is horizontally partitioned.
  - 2.3.5 All sets mapped to from an AUC must have the same record type as its owner; the owner may be different in case of horizontal partitioning.
  - 2.3.6 If the entity is horizontally partitioned, verify that the sets being mapped to have owner records that belong to a fragment of the partitioned entity.
  - 2.3.7 All AUC to set maps must contain a value which must be unique for a particular AUC.
  - 2.3.8 The AUC must have been previously mapped for specified preference number.
  - 2.3.9 An AUC must be mapped to at least two sets of same database/record.
  - 2.3.10 If the above conditions are met, create an occurrence the CDM table AUC\_IS\_MAPPING and AUC\_ST\_MAPPING.
- 2.4 For ALTER DROP AUC to set mapping, the following rules apply:
  - 2.4.1 When dropping, at least two sets of the same database must remain mapped to the AUC.

- 2.4.2 If the above condition is met, the specified set is deleted from the CDM table AUC\_ST\_MAPPING.
- 2.5 For ALTER ALTER set mapping, the following rules apply:
  - 2.5.1 The stated AUC\_ST\_MAPPING must exist for the stated preference.
  - 2.5.2 The AUC value must be unique for all mappings from a particular AUC.
  - 2.5.3 If the above conditions are met, the AUC to SET MAP specified will be modified in the CDM table AUC\_ST\_MAPPING.
- 2.6 For ALTER ADD relation to set mapping, the following rules apply:
  - 2.6.1 The set must not have been previously mapped.
  - 2.6.2 The member record name must be specified if the set being mapped to is a multi-member set.
  - 2.6.3 If the above rules are obeyed, an RC\_BASED\_REC\_SET entity occurrence is created.
- 2.7 For ALTER DROP relation to set mapping, no special validations are performed. An RC\_BASED\_REC\_SET entity occurrence is deleted.
- 2.8 For ALTER ADD entity to record mapping, additional records can be mapped to the entity.
- 2.9 For ALTER DROP entity to record mapping, the following rules apply:
  - 2.9.1 The mapping for the entity to the specified records are dropped.
  - 2.9.2 After the drop is performed, at least one record must remain mapped to the entity.
  - 2.9.3 A record cannot be dropped if a field in that record is still mapped.
- 2.10 For an entity to record mapping, the distributed rules for the entity can be altered. An entity to record mapping must exist for the entity.

3.2.8.11 ALTER MODEL - Creates a current model for a NDDL session using modeling commands.

A. Function:

Alter Model performs the following functions:

1. Updates the date of the model showing the model change date
2. Creates a current model for an NDDL session
3. Changes the status of the model to "UNCHECKED"

B. CDM Requirements:

Alter Model requires that the model to be altered exists in the CDM.

C. Processing:

1. The Alter Model process verifies that the model was created previously. If the model does not exist, an error occurs and an error message is issued.
2. The model being altered becomes the current model for an NDDL modeling session with all subsequent model processing identified with the model. The CDM MODEL\_CLASS table is updated to show the system date as the date the model was updated and changes the model status to "UNCHECKED".

3.2.8.12 ALTER MODULE - Alter the software module definition

A. Function:

ALTER MODULE allows the NDDL User to change the definition of the module parameter list or the module language.

B. CDM Requirements:

The module named must currently exist in the CDM. Any module parameters to be dropped must currently exist. Module parameters to be added to the parameter list must currently not exist in the CDM. Data type names used when adding module parameters must currently exist. Definitions may not be altered if the module has mapping algorithms established for it.

C. Processing:

1. Verify that the module named (MOD\_ID) currently exists on the SOFTWARE\_MODULE Table.



2. The module being altered must be defined for a complex mapping algorithm (STATUS\_IND = "C")
3. Retrieve the language from the Lang-Lst.
  - 3.1 If LIST-COUNT equals zero, no change is made to the language.
  - 3.2 If a language is retrieved, modify LAND\_NAME of the SOFTWARE\_MODULE Table.
4. Update the LATEST\_REV DATE of the SOFTWARE\_MODULE Table with the current system date.
5. Check the Drp-Parm-Lst for parameters to be dropped. If parameters exist on the list:
  - 5.1 Verify that the parameter (PARM\_ID) exists for the software module named (MOD\_ID).
  - 5.2 Delete the module parameter from the MODULE\_PARAMETER Table.
  - 5.3 Check the return status for a Referential Integrity Test Failed.
    - 5.3.1 If return status not equal "zero" issue an error message.
  - 5.4 Retrieve next module parameter to be dropped.
6. check the After-Before-Lst for where new module parameters should be positioned in the parameter list.
  - 6.1 If LIST COUNT equals zero, no module parameters are to be added to the parameter list and the program is exited.
  - 6.2 If LIST\_COUNT does not equal zero, retrieve the word "Before" or "After".
7. Select all module parameters associated with the software module named into an internal program table.
8. Retrieve the name of the parameter (PARM\_ID) the new parameters should be positioned before or after from the Parm-Lst.
  - 8.1 Verify that this module parameter (PARM\_ID) currently exists for the software module named (MOD\_ID).

- 8.1.1 If module parameter doesn't exist, issue an error message and exit program.
  - 8.2 Check to see how many new parameters should be added before or after this PARM\_ID (Call CPFVAL on Parm-Lst) and process these parameters as follows:
    - 8.2.1 Retrieve the parameter name to be added (PARM\_ID) from the Add-Parm-Lst.
    - 8.2.2 Verify that this parameter to be added (PARM\_ID) doesn't currently exist for the software module name (MOD\_ID).
      - 8.2.2.1 If it currently exists, issue an error message and exit the program.
    - 8.2.3 Retrieve the Data Type Name of the new parameter from the Data Lst.
    - 8.2.4 Verify that this name (DATA\_TYPE\_NAME) exists on the USER\_DEF\_DATA\_TYPE Table.
      - 8.2.4.1 If it doesn't exist, issue an error message and exit from the program.
  - 8.3 Once all module parameters to be added to the parameter list have been checked and verified resequence the internal program structure.
  - 8.4 Delete all parameters associated with the software module named (MOD\_ID) from the MODULE\_PARAMETER Table.
    - 8.4.1 If return status equals a Referential Integrity Test Failed, issue an error message and exit the program.
  - 8.5 Insert the resequenced parameter list from the internal program structure into the MODULE\_PARAMETER Table.
- 3.2.8.13 ALTER PARTITION - Alter the definition of a horizontal partition for an entity.

A. Function:

Alter Partition allows the NDDL user to perform the following modifications to the definitions of existing horizontal partitions:

1. Deletion of records included in the horizontal partition.
2. Addition of records for a horizontal partition.

B. CDM Requirements:

The record names references must be found in the CDM. Any record names dropped must already be defined for the partition. Any record names added must not already be defined in the partition. There must always be two partitions specified for an entity.

C. Processing:

1. The user entered entity name is verified to be in the CDM and the entity number is retrieved.
2. A "1" is used as the horizontal partition number if none is specified by the user.
3. Verify that the partition number currently exists for the entity.
4. For each DROP RECORDS request the following checks are made:
  - 4.1 The DB\_NAME and RT\_ID currently are defined in the CDM.
  - 4.2 The RT\_NO is retrieved from this selection.
  - 4.3 Verify that the Entity and Record combination exists. Check that the HP\_NO retrieved is the same as the one entered.
  - 4.4 Delete that record from the horizontal partition table.
5. For each ADD RECORDS request the following checks are made:
  - 5.1 The DB\_NAME and RT\_ID currently are defined in the CDM.
  - 5.2 Retrieve the RT\_NO from selection in 5.1.
  - 5.3 Verify that the entity and record combination doesn't currently exist for the partition number.
  - 5.4 Insert the partition record.

6. After all adds and drops are processed, all the partition records are selected for the entity and horizontal partition number.
  - 6.1 If no horizontal partition is found for an entity (NDML-COUNT = 0), the horizontal partition has been inadvertently deleted by the ALTER Command. Process an error with an appropriate message.
  - 6.2 If an entity is horizontally partitioned over less than 2 records, (NDML-COUNT < 2) process an error with an appropriate message.

3.2.8.14 ALTER PSB - Change a PSB name/host association in the CDM.

A. Function:

1. Alter PSB name allows user to change a PSB/host association in the CDM.

B. CDM Requirements:

1. PSB name must be previously defined in the CDM.
2. Host must be previously defined in the CDM.

C. Processing:

1. The CDM is checked to see if the PSB name exists.
2. If PSB name exists:
  - 2.1 For the host identification to be changed.
    - 2.1.1 The CDM is checked to see if the host exists.
    - 2.1.2 If the host exists, the PSB name/host association will be updated in the CDM.
    - 2.1.3 If the host does not exist:
      - 2.1.3.1 Issue an error message.
      - 2.1.3.2 Exit command processing.
3. If PSB name does not exist:
  - 3.1 Issue an error message.
  - 3.2 Exit command processing.

3.2.8.15 ALTER RECORD - Alters an existing record type for a previously defined database/PCB. Areas

and field names may be added and dropped.

A. Function:

Alter record performs the following functions:

1. Verifies that the database exists
2. Verifies that the record type exists
3. If the DBMS is CODASYL, verify that the areas association to be added do not exist or those to be dropped do exist. Additionally, the area association with the record may be inserted or deleted
4. Verifies that the fields to be added do not previously exist and the fields to be dropped exist
5. The fields being added may be defined as repeating, group, elementary, indexed, component or re-defined. Additionally, they may be specified as keys, and indicate whether known or unknown to the DBMS
6. The fields will be dropped if specified by the user. All of its subcomponent fields will also be dropped. If the field being dropped is used as an indexed-by, or occurs-depending-on of another data field, the latters indexed-by or occurs-depending is set to null. If the field being dropped is redefined by another field, the latter is dropped and all its subcomponent fields are dropped

B. CDM Requirements:

1. The database must be previously defined.
2. CODASYL database areas must be previously defined.
3. The record being altered must exist.
4. The fields being added must not previously exist.
5. The fields being dropped must previously exist.
6. The occurs depending on and redefines field must be defined earlier.

C. Processing:

1. The CDM is checked to see if the database exists. If it has not been defined, issue an error message and exit command processing.

2. The CDM is checked to see if the record type exists. If it has not been defined, issue an error message and exit command processing.
3. For each area association to be added:
  - 3.1 The CDM is checked to ensure the area has been previously defined.
  - 3.2 The CDM is checked to see the database area assignment is not associated with the record.
  - 3.3 If both conditions are true, the area association is inserted.
  - 3.4 If any one of the conditions is false, issue an error message and exit command processing.
4. All the existing data fields belonging to the record are retrieved into a temporary structure. Syntax and semantic checks on the data fields being added and/or dropped are performed within this temporary table before the data fields are added or deleted in the CDM.
5. If the data field is being added:
  - 5.1 If the keyword "after" or "before" is specified, the user specified data fields are inserted at the appropriate position, and later resequenced.
  - 5.2 If the keywords are not specified, the new fields are appended at the end of the existing record layout.
6. For each field being added, perform the following checks:
  - 6.1 Each subcomponent field must specify a level number to ensure subcomponent structures are properly implemented.
  - 6.2 If the field name is the reserved word "filler", it must have a corresponding numeric filler-size and be unknown to the DBMS.
  - 6.3 The data type name for an indexed field and depending-on field must be numeric.
  - 6.4 Each elementary item must have a data type and the user-enter data type name must have been previously defined in USER\_DEF\_DATA\_TYPE.
  - 6.5 If it is a repeating data field, the number of times it occurs must be greater than one. If

the indexed by field is not defined, it will be created with a default numeric index data type name and an index-indicator of "G".

- 6.6 The depending on field must be previously defined.
- 6.7 If the field is redefined, the redefines field name must be previously defined.
- 6.8 The indexed-by field cannot be a key, group, component or a redefine.
- 6.9 The depending-on field cannot be a group, repeating, or an index.
- 6.10 A repeating field or repeating group cannot be key.
- 6.11 If a component data field is key, one of its subcomponents cannot also be a unique key. It may be a secondary or duplicates allowed key.
- 6.12 An ORACLE DBMS does not support repeating fields, indexes, redefinitions, keys or fillers.
- 6.13 Redefinitions of TOTAL fields must be specified as a single field.
- 6.14 For a relational database (ORACLE or DB2), all fields must be 01 level and specified as known to the DBMS.
- 7. If the record layout conforms with the checks performed, the existing data fields of the record are deleted. The existing and new data fields residing in the temporary structure are inserted into the CDM.
- 8. For each area to be dropped:
  - 8.1 The CDM is checked to verify the area is previously associated with the record.
  - 8.2 If the association exists, delete the database area assignment for the record.
  - 8.3 If no association exists, issue an error message and exit command processing.
- 9. The CDM is checked to verify that the data field is not mapped-to. If the data field is present in a record union mapping, complex mapping algorithm or mapped to a tag (AUC), issue an error message. Continue processing with the next data field.

10. For each data field to be dropped:
  - 10.1 Delete the field specified by the user.
  - 10.2 Delete any subcomponent data fields.
  - 10.3 If the data field appears as the indexed-by data field of another field in the record, update the latter's indexed-by to null.
  - 10.4 If the data field appears as the occurs-depending-on data field of another field in the record, update the latter's depending-on to null.
  - 10.5 If the data field appears as the redefines data field of another field in the record, drop the redefining field and all its subcomponents.
  - 10.6 If the data field is indexed, drop the indexing field if it is a "G" index indicator; otherwise, set the index indicator to N and do not drop the indexed-by field.
11. If no errors occurred in item 10, the user specified data fields are deleted along with its subcomponents. Delete any textual description for the data fields being dropped.

3.2.8.16 ALTER RELATION - Alter a conceptual link relation.

A. Function:

Alter Relation performs any or all of the following functions:

1. Change to cardinality of an existing link relation
2. Migrate the key of the independent entity to the dependent entity, creating a complete relation and inherited attribute uses
3. Assign new tag names to the key class members migrated to the dependent entity
4. Associate one or more keywords with the link relation
5. Drop the key from the relation and from the dependent entity and all subsequent entities that inherited the key
6. Delete any empty keys that result from dropping a key migration



B. CDM Requirements:

1. Key for independent entity must exist.
2. Key members for independent entity must exist
3. An attribute use for each key member must exist.
4. Link relation must exist.
5. Independent entity must exist.
6. Dependent entity must exist.
7. If a key is to be migrated to the dependent entity, the key must not have been previously migrated to the dependent entity.
8. If a key is to be dropped from the dependent entity and all subsequent entities, the key must have been previously migrated.

C. Processing:

Processing varies depending on the options chosen by the user. If an error is detected, processing continues with the next option on the command.

1. **CARDINALITY**

Any cardinalities specified replace the original values established when the link relation was created, unless an error is detected, a warning message is generated and the cardinality defaults to its original value.

2. **ADD MIGRATES**

An attribute use and an inherited attribute use for the dependent entity is created for each key member migrated to the dependent entity. If the set phrase is specified, TAG\_NAME1 (the independent entity is tag name) is migrated to the dependent entity with the new name TAG\_NAMED. A complete relation occurrence is created. If a keyword is specified, the keyword is created in the CDM if it doesn't already exist and a class keyword occurrence is created.

3. **DROP MIGRATES**

The complete relation occurrence for the relation is deleted. The attribute use and inherited attribute use originally created for each key member

migrated to the dependent entity class are deleted. Also, any tag constraints are deleted. In addition, the attribute uses and inherited attribute uses created for each key member migrated to lower level dependent entities are also deleted. Then all keys and complete relations which become empty due to the deletion of migrated keys members are deleted. Finally, any text descriptions for empty keys are deleted. If keywords are specified, the relation keyword occurrence are also deleted.

3.2.8.17 ALTER UNION - Alter the union of conceptual entities representing a record type in the CDM.

A. Function:

ALTER UNION allows the NDDL user to alter a record union definition by dropping and/or adding entities.

B. CDM Requirements:

The data base and record named must currently exist in the CDM. Any entities dropped must currently exist in the CDM. Any entities to be added to the union must not currently exist in the CDM.

C. Processing:

1. Verify that the data base name and record name currently exist. Retrieve the record type number (RT\_NO) for this selection.
2. For each DROP ENTITY request, the following occurs:
  - 2.1 Verify that the entity named exists in the CDM.
  - 2.2 Verify that the union currently exists on the ECRTUD Table. Use RT\_NO and EC\_NO for the select.
  - 2.3 Delete all records from the ECRTUD Table where RT\_NO and EC\_NO match those entered.
  - 2.4 For the entity we are dropping, check AUC\_IS\_MAPPING for tags that belong to this entity to see if they map to any data fields that are part of the record for this union. If not, delete the entity to record mapping (EC\_RT\_MAPPING) for EC\_NO and RT\_NO.
  - 2.5 Check to see if the entity is part of any other entity to record mappings. If not, delete the distributed rules (DISTRIBUTED\_RULES) for the entity (EC\_NO).

3. For each ADD ENTITIES request, the following occurs:
  - 3.1 Verify that the entity named exists in the CDM.
  - 3.2 Verify that the data fields entered are associated with the database and record named and retrieve the datafield number (DF\_NO).
  - 3.3 Verify that the entity does not currently exist in the union.
  - 3.4 Verify that the union discriminator value is compatible with the data type of the data field.
  - 3.5 Insert the union into the CDM.
  - 3.6 Check for an integrity test failure which indicates that no entity to record mapping exists for the record union.
4. After all drops and adds have been processed the ECRTUD Table is checked. The number of entities for the record type named is counted.
  - 4.1 If the count is zero, an error message is issued. The union has been deleted by the ALTER UNION command. The DROP UNION command should be used to delete a record union.
  - 4.2 If the count is less than two, an error message is issued. The union has been altered leaving only one entity defined for the union which is illegal.

3.2.8.18 CHECK MODEL - Determines if the model conforms to specified IDEF1 rules

A. Function:

The check model performs the following functions:

1. Verifies that the model exists in the CDM
2. Verifies that the model has one or more entities
3. Verifies that the entities have at least one attribute
4. Verifies that the model follows the specified IDEF1 rules (see Rules under Processing)

5. Updates the model in the CDM to show that it is a "checked" model and the date it was checked

B. Requirements:

The check model process requires that the model exist in the CDM. (See Rules under Processing).

C. Processing:

The check model process determines if the model follows the following IDEF1 rules.

Rules:

1. No non-specific relations are allowed (independent cardinality greater than one).
2. No incomplete link relations (key has not been migrated).
3. Each entity must have at least one attribute use.
4. Each owned attribute must have a domain and that domain must have a standard data type.
5. A primary key must be defined for each entity.
6. Multiple keys of an entity must not be subsets of one another.
7. No one to one relations are allowed.
8. No dependency loops are allowed e.g. A->B->C->D->B.
9. At least one entity must exist in the model.
10. Category Relations have more than one category member.
11. Link Relations migrated primary key instead of alternate keys.
12. Inherited keys are not split among key and nonkey attributes in the dependent entity.
13. If any of the above rules are not satisfied, a message is printed out to a file.

The following rules cannot be checked for the model:

1. One to none or one relationships imply identical keys.

2. Key uniqueness throughout the model is not checked, i.e., no two entities may have the same key unless they are related to each other with a one to none or one relation.

The processing verifies the existence of the model. The process then selects each entity belonging to the model and checks the relations, keys, and attributes.

Next the process checks the hierarchical dependencies both up and down to determine if there are any dependency loops within the model.

The output is generated to a user defined file or screen. If all rules have been followed the CDM MODEL\_CLASS table is updated to reflect the date and the model status of CHECKED.

#### 3.2.8.19 COMBINE ENTITY - Combine two conceptual entities.

##### A. Function:

Combine Entity performs the following functions:

1. Combine two entities that exist either within the same model (intra-model) or between two models (inter-model)
2. Generate NDDL commands on a file or screen to populate the to-model entity with the attributes, relations, aliases, keywords, keys, and key members associated with the from-model entity

##### B. CDM Requirements:

1. If the to-model is not specified an inter-model combine is assumed, and the to-model must exist in the CDM.
2. If the to-model and from-model are specified, both models must exist in the CDM.
3. The two entities to be combined must exist in the model(s).

##### C. Processing:

1. If it is an intra-model combine, to-model defaults to the from-model.
2. First, verify that the two entities to be combined exist in the from-model and to-model. Processing

halts if any of the verification checks fail. The NDDL commands to combine the from-entity and to-entity are generated in a user defined file or screen. Retrieve from global variables the name of the file.

3. Verify that the from entity is not a generic entity of the to\_entity. If it is, generate an error message and reject the command. If from\_entity is a category entity, the to\_entity must be the immediate generic entity for the from\_entity only. If it is not, generate an error message and reject the command.
4. Now determine if a relation exists between the to-entity and from-entity. If one does, generate a command to drop this relation. Drop the relation only if it is a link relation. If it is a category relation, delete the category member occurrence. If this was the only category member, then drop the category relation with complete ripple down. Also, if it is an intra-model combine generate a command to delete the from-entity.
5. The from-entities' keys and key members are saved in a temporary key list, in order to migrate the keys via new relations that will be created after the from-entity has been combined into the to-entity.
6. Generate an "Alter entity and owned attributes..." for all the attributes that belonged to the from-entity. If the user specified that he wanted keywords, aliases and/or descriptions, NDDL commands are generated for the same. The from-entity name is generated as an alias for the to\_entity.
7. Next, select all relations in the from-model where the from-entity is the dependent entity in the relation. If this from-independent entity(s) exists in the to-model, generate Create relation ... migrates...commands for the same. The key class that was inherited via this relation has to be generated for the to-entity. Generate an Alter Entity add key...for the inherited key class.
8. Next, select all the relations in the from-model where the from-entity is the independent entity in the relation. If this from-dependent entity exists in the to-model, generate a Create relation...migrates...command using the information stored earlier in the temporary key list. If this from\_category member entity(s) exists in the to\_model, generate Create relation.

9. Commands are also generated to associate keywords and descriptions with the relation if any exist. Finally, close the user defined file at the end of processing.

3.2.8.20 COMMIT - Commit the changes made to the CDM since the last commit point.

A. Function:

Perform ORACLE and NDML commit.

B. Processing:

1. Call a routine to perform the NDML commit.
2. Issue an ORACLE commit.

3.2.8.21 COMPARE MODEL - Compare two IDEF1 models.

A. Function:

Compare models to see if their entity names, attribute names, entity keywords, attribute keywords and relation keywords match each other.

B. CDM Requirements:

The two models to be compared must exist.

C. Processing:

1. First, verify the existence of both models. If either of these two models does not exist, flag a user error. Next, retrieve from global variables the name of the file where the output is to be directed.
2. Parse the "Alias Exception List". If the Parser doesn't return the word "ALIAS":
  - 2.1 Compare entities based on identical names either primary or alias names.
  - 2.2 Compare attributes based on identical names either primary or alias names.
  - 2.3 Compare entity keywords to determine if entities from both models use the same keyword.
  - 2.4 Compare attribute keywords in the same manner as entities.
  - 2.5 Compare relation keywords.

- 2.6 Compare link relations based on identical names either primary or alias names.
  - 2.7 Compare category relations based on identical names either primary or alias names, incomplete/complete status, discriminating attribute names, and number of category members.
3. If the Parser returned the word "ALIAS":
    - 3.1 Compare entities based on identical primary names.
    - 3.2 Compare attributes based on identical primary names.
    - 3.3 Compare entity keywords to determine if entities from both models use the same keyword.
    - 3.4 Compare attribute keywords in the same manner as entities.
    - 3.5 Compare relation keywords.
    - 3.6 Compare relation names based on identical primary names.
    - 3.7 Compare category relations based on identical primary names, incomplete/complete status, discriminating attribute name, and number of category members.
  4. For each successful comparison above, a message will be written to a file to indicate a match is found in two models.
- 3.2.8.22 COPY ATTRIBUTE - Copies an attribute and all associated information from one model to another model (inter-model) or within a model (intra-model).
- A. Function:
- Copy Attribute performs the following functions:
1. Verifies that the model specified exists
  2. Copies an attribute within a model or to another model
  3. Verifies that the new attribute does not exist in the current model
  4. When indicated, copies all description text, aliases, and keywords related to the attribute



5. Optionally, places the created NDDL commands in a file or directs the output to the screen
6. Optionally, interactively performs the intra-model or inter-model copy

B. CDM Requirements:

The Copy Model process requires that the from-model and the attribute exist in the CDM database.

C. Processing:

The following rules apply to the copy attribute process:

1. The except clause, when used, indicates what items associated with the attribute are not to be copied. If the except clause is omitted all keywords, aliases, and textual descriptions of the attribute are copied.
2. The process verifies that the current model exists in the CDM database. If the FROM clause is not specified, the FROM\_MODEL defaults to the current model. Additionally, the attribute to be copied ('FROM' ATTRIBUTE) is verified to exist in the from-model. If either attribute or model is not found, an error message is issued and the processing terminates.
3. The processing determines whether the copy is to be interactive or a copy to a file screen.
4. If the 'FROM' option is present, an inter-model copy is assumed.
  - 4.1 The process verifies the FROM model exists in the CDM. If the model does not exist, issue an error message and exit command processing.
  - 4.2 Verify if the 'FROM' attribute and 'TO' attribute have been specified. If not, the 'TO' attribute name defaults to the 'FROM' attribute name.
  - 4.3 Verify the 'TO' attribute does not exist in the model. If the attribute does exist, issue an error message and exit command processing.
  - 4.4 If the copy is 'DIRECTLY'

- 4.4.1 Insert the 'TO' attribute in the CDM tables ATTRIBUTE\_NAME and ATTRIBUTE\_CLASS for the current model.
- 4.4.2 If keywords have not been excepted, verify the keyword exists in the current model and insert the keyword occurrence in the CDM table AC\_KEYWORD.
- 4.4.3 If the descriptions have not been excepted, copy the 'FROM' attributes description text for the 'TO' attribute in the CDM table DESC\_TEXT.
- 4.4.4 If the aliases have not been excepted, verify the 'FROM' attributes alias name does not exist as an attribute in the current model. If the alias name does not previously exist, create an alias occurrence for the 'TO' attribute by inserting in the CDM table ATTRIBUTE\_NAME.
- 4.5 If the copy is not 'DIRECTLY'
  - 4.5.1 Generate NDDL commands to create the 'TO' attribute in the current model.
  - 4.5.2 If keywords have not been excepted, generate the keyword clause.
  - 4.5.3 If aliases have not been excepted, generate the CREATE ATTRIBUTE alias command.
  - 4.5.4 If descriptions have not been excepted, generate the Describe commands for the new attribute.
- 5. If the 'FROM' option is not present, an intra-model copy is assumed.
  - 5.1 Verify that the 'FROM' and 'TO' attribute names have been specified and are not the same. If not, issue an error message and exit command processing.
  - 5.2 Verify the 'TO' attribute name does not exist in the current model. If it does exist, issue an error message and exit command processing.
  - 5.3 If the copy is 'DIRECTLY'

- 5.3.1 Insert the TO\_ATTRIBUTE in the CDM tables ATTRIBUTE\_NAME and ATTRIBUTE\_CLASS for the current model.
- 5.3.2 If keywords are not excepted, insert the FROM\_ATTRIBUTE keyword for the TO\_ATTRIBUTE in the CDM table AC\_KEYWORD.
- 5.3.3 If descriptions are not excepted, insert the 'FROM' attribute's textual description for the 'TO' attribute in the CDM table AC\_KEYWORD.
- 5.3.4 If aliases have not been excepted, issue a warning message. The attribute aliases are not created intra-model.

5.4 If the copy is not 'DIRECTLY'

- 5.4.1 Generate NDDL commands to create the 'TO' attribute in the current model.
- 5.4.2 If keywords, aliases and/or descriptions have not been excepted, generate NDDL commands to create the corresponding keywords, aliases and descriptions for the TO\_ATTRIBUTE in the current model.

3.2.8.23 COPY DATABASE - Generate NDDL commands to make a copy of databases.

A. Function:

COPY DATABASE allows the NDDL user to copy specified databases or all databases within the CDM. Optionally, the textual descriptions for the database, records, datafields and sets are copied. At the user's discretion, the generated NDDL commands are either outputted to a screen or file.

B. CDM Requirements:

The database/databases to be copied must exist in the CDM.

C. Processing:

- 1. Global input parameters are checked to determine if the generated NDDL commands will be appended to an existing output file or written to a new file or outputted to a screen. Input parameters are set with a previous "SET OUTPUT" statement.

2. A database name or the word "ALL" is obtained from a Database Name Parser List.
  - 2.1 If the Parser returns the word "ALL", retrieve all the databases in the DATA\_BASE Table and store the database name, host name, DBMS name, null value and NTM directory code in an internal program table.
  - 2.2 If the Parser returns a database name, verify that the database name exists in the CDM. Store each database named on the Parser List in an internal program table, along with its host name, DBMS name, null value and NTM directory code.
3. Databases are retrieved into the internal program table sequenced by database name.
4. Process each database name on the internal program table as follows.
  - 4.1 Start generating the "DEFINE ... DATABASE" NDDL command using the DBMS, database and host name.
  - 4.2 If the DBMS name is "IMS":
    - 4.2.1 Use the keyword "PCB" instead of "DATABASE" after the DBMS name when generating the command in step 4.1.
    - 4.2.2 Select the PSB Name, sequence number and feedback length from the PSB PCB Table and generate the "WITH POSITION" clause of this command.
  - 4.3 If the DBMS name is "ORACLE" or "DB2":
    - 4.3.1 Use the keyword "DATABASE" instead of "PCB" after the DBMS name when generating the command in step 4.1.
    - 4.3.2 Select the database password from the DB\_PASSWORD Table and generate the "WITH PASSWORD" clause of this command.
  - 4.4 If the DBMS name is "TOTAL":
    - 4.4.1 Use the keyword "DATABASE" instead of "PCB" after the DBMS name when generating the command in step 4.1.
    - 4.4.2 Select the file names from the DATA\_BASE\_AREA Table and generate the "WITH FILES" clause of this command.

- 4.5 If the DBMS name is one of the CODASYL DBMS VAX-11, IDMS or IDS-II):
  - 4.5.1 Use the keyword "DATABASE" instead of "PCB" after the DBMS name when generating the command in step 4.1.
  - 4.5.2 Select the schema and subschema name from the SCHEMA\_NAMES Table and generate the "WITH SCHEMA ... AND SUBSCHEMA" clause of this command.
  - 4.5.3 Select the area name/names for the DATA\_BASE\_AREA Table and append the "AREAS" clause to the "WITH SCHEMA" clause generated in 4.5.2.
  - 4.5.4 Select the database location from the SCHEMA\_NAMES Table and generate the "LOCATED AT" clause.
- 4.6 Generate the "STORES CHARACTER/INTEGER NULL AS" clause using the null value stored in the internal program table.
- 4.7 Generate the "NTM DIRECTORY" clause using the two character field representing the NTM Directory where the generated request processors can be found, which is stored in the internal program table.
- 4.8 Generate an "ALTER DATABASE" NDDL command using the database name.
- 4.9 Retrieve all the record names belonging to the database being processed, from the RECORD\_TYPE Table.
- 4.10 Records are retrieved into the table created in step 4.9 sequenced by record name.
- 4.11 Process each record name in the internal program table as follows:
  - 4.11.1 Start generating the "DEFINE RECORD" NDDL command using the record name.
  - 4.11.2 Select all areas associated with the record name and database number for a CODASYL DBMS.
    - 4.11.2.1 If areas are retrieved generate the "IN AREAS" clause using the area names.

- 4.11.2.2 If no areas are retrieved, omit this clause.
- 4.11.3 Retrieve all the information from the DATA FIELD Table for the record named. Retrieve the data fields ordered by record sequence number.
- 4.11.4 Generate the "WITH FIELD" clause for each data field associated with the record. Use the information obtained in step 4.11.3.
- 4.11.5 For each field always generate:
  - 4.11.5.1 The level of each field (i.e. if not a subcomponent, the level number defaults to 1).
  - 4.11.5.2 Repeating field information (i.e. if the field doesn't repeat, occurs defaults to 1).
  - 4.11.5.3 Whether the field is known or unknown to the DBMS.
  - 4.11.5.4 Whether the field is a unique or duplicate key.
- 4.11.6 For each field generate, whenever applicable:
  - 4.11.6.1 The level of the subcomponent fields (i.e. if the field repeats, the level number will be greater than 1.)
  - 4.11.6.2 Any filler information.
  - 4.11.6.3 Repeating field information (i.e. the "OCCURS" clause, "DEPENDING ON" clause and INDEXED BY" clause).
  - 4.11.6.4 The "REDEFINES" clause.
  - 4.11.6.5 The data type name.
- 4.11.7 Parse the Description Exception List once.
  - 4.11.7.1 If the word "DESCRIPTION" was not retrieved, generate a "DESCRIBE" NDDL command for the record if a textual description exists. Generate separate "DESCRIBE" NDDL commands if textual information exists for data fields.

- 4.11.7.2 If the word "DESCRIPTION" was retrieved, omit generating "DESCRIBE" commands.
- 4.12 Select every set belonging to the database being processed from the RECORD\_SET Table. As each set is retrieved process as follows:
  - 4.12.1 Start generating the "DEFINE SET...FROM" NDDL command using the set's name and the set's owner record name.
  - 4.12.2 Select all the members and required/optional indicators of the set named from the SET\_TYPE\_MEMBER Table. Generate the "TO" clauses using the member's record names and required/optional indicator obtained from this select.
  - 4.12.3 If the DBMS of the database is TOTAL, retrieve the data field name of the owner record for the set and generate the "LINKED BY" clause.
  - 4.12.4 Check what was retrieved from the Description Exception List.
    - 4.12.4.1 If the word "DESCRIPTION" was not retrieved, generate a "DESCRIBE" NDDL command for the set if textual descriptions exist.
    - 4.12.4.2 If the word "DESCRIPTION" was retrieved, omit generating a "DESCRIBE" command.
- 4.13 Check what was retrieved from the Description Exception List.
  - 4.13.1 If the word "DESCRIPTION" was not retrieved generate a "DESCRIBE" NDDL command for the database if textual descriptions exist.
  - 4.13.2 If the word "DESCRIPTION" was retrieved, omit generating a "DESCRIBE" command.
- 3.2.8.24 COPY DBMS - Generate NDDL commands to copy a DBMS or all DBMS

A. Function:

COPY DBMS allows the NDDL user to copy a specified DBMS or all DBMS within the CDM. Optionally, all databases

associated with the DBMS' are copied. A "DESCRIBE" command may also be generated if the user desires. At the user's discretion, the generated NDDL commands are either outputted to a screen or file.

B. CDM Requirements:

The DBMS/DBMS' to be copied must exist in the CDM.

C. Processing:

1. Global input parameters are checked to determine if the generated NDDL commands will be appended to an existing output file or written to a new file or outputted to a screen. Input parameters are set with a previous "SET OUTPUT" statement.
2. A DBMS name or the the word "ALL" is obtained from a DBMS Name Parser List.
  - 2.1 If the Parser returns the word "ALL" process every DBMS in the DBMS Table.
  - 2.2 If the Parser returns a DBMS name, verify that the DBMS name exists in the CDM.
3. Process each DBMS name obtained from the DBMS Name Parser List or every DBMS in the CDM as follows:
  - 3.1 Start generating the "DEFINE DBMS" NDDL command using the DBMS name and its model type.
  - 3.2 Retrieve any hosts associated with the DBMS.
    - 3.2.1 If no hosts are retrieved for the DBMS being processed, terminate the "DEFINE DBMS" NDDL command.
    - 3.2.2 If hosts are retrieved for the DBMS being processed, generate the "ON HOST" clause using the host name/names, then terminate the "DEFINE DBMS" command.
4. Parse the Database Inclusion List once.
  - 4.1 If nothing was retrieved, omit copying all databases associated with the DBMS named.
  - 4.2 If the word "DATABASE" was retrieved process step 5 through step 8.
5. Select all the databases from the DATA\_BASE Table associated with the DBMS being processed.
6. Store the database name, host name, DBMS name, null value and NTM directory code on an internal program



table.

7. Databases are retrieved into the internal program table sequenced by database name.
8. Process each database name on the internal program table as follows.
  - 8.1 Start generating the "DEFINE ... DATABASE" NDDL command using the DBMS, database and host name.
  - 8.2 If the DBMS name is "IMS":
    - 8.2.1 Use the keyword "PCB" instead of "DATABASE" after the DBMS name when generating the command in step 8.1.
    - 8.2.2 Select the PSB Name, sequence number and feedback length from the PSB PCB Table and generate the "WITH POSITION" clause of this command.
  - 8.3 If the DBMS name is "ORACLE" or "DB2":
    - 8.3.1 Use the keyword "DATABASE" instead of "PCB" after the DBMS name when generating the command in step 8.1.
    - 8.3.2 Select the database password from the DB\_PASSWORD Table and generate the "WITH PASSWORD" clause of this command.
  - 8.4 If the DBMS name is "TOTAL":
    - 8.4.1 Use the keyword "DATABASE" instead of "PCB" after the DBMS name when generating the command in step 8.1.
    - 8.4.2 Select the file names from the DATA\_BASE AREA Table and generate the "WITH FILES" clause of this command.
  - 8.5 If the DBMS name is one of the CODASYL DBMS' (VAX-11, IDMS or IDS-II):
    - 8.5.1 Use the keyword "DATABASE" instead of "PCB" after the DBMS name when generating the command in step 8.1.
    - 8.5.2 Select the schema and subschema name from the SCHEMA NAMES Table and generate the "WITH SCHEMA ... AND SUBSCHEMA" clause of this command.
    - 8.5.3 Select the area name/names for the DATA\_BASE\_ AREA Table and append the

"AREAS" clause to the "WITH SCHEMA" clause generated in step 4.5.2.

- 8.5.4 Select the database location from the SCHEMA NAMES Table and generate the "LOCATED AT" clause.
- 8.6 Generate the "STORES CHARACTER/INTEGER NULL AS" clause using the null value stored in the internal program table.
- 8.7 Generate the "NTM DIRECTORY" clause using the two character field representing the NTM Directory where the generated request processors can be found, which is stored in the internal program table.
- 8.8 Generate an "ALTER DATABASE" NDDL command using the data base name.
- 8.9 Retrieve all the record names belonging to the database being processed, from the RECORD\_TYPE Table. Store the record names along with their record numbers in an internal program table.
- 8.10 Records are retrieved into the table created in step 4.9 sequenced by record name.
- 8.11 Process each record name in the internal program table as follows:
  - 8.11.1 Start generating the "DEFINE RECORD" NDDL command using the record name.
  - 8.11.2 Select all areas associated with the record name and database number for a CODASYL DBMS.
    - 8.11.2.1 If areas are retrieved generate the "IN AREAS" clause using the area names.
    - 8.11.2.2 If no areas are retrieved, omit this clause.
  - 8.11.3 Retrieve all the information from the DATA FIELD Table for the record named. Retrieve the data fields ordered by record sequence number.
  - 8.11.4 Generate the "WITH FIELD" clause for each data field associated with the record. Use the information obtained in step 4.11.3.

8.11.5 For each field always generate:

- 8.11.5.1 The level of each field (i.e. if not a subcomponent, the level number defaults to 1).
- 8.11.5.2 Repeating field information (i.e., if the field doesn't repeat, occurs defaults to 1).
- 8.11.5.3 Whether the field is known or unknown to the DBMS.
- 8.11.5.4 Whether the field is a unique or duplicate key.

8.11.6 For each field generate, whenever applicable:

- 8.11.6.1 The level of the subcomponent fields (i.e. if the field repeats, the level number will be greater than 1).
- 8.11.6.2 Any filler information.
- 8.11.6.3 Repeating field information (i.e., the "OCCURS" clause, the "DEPENDING ON" clause and "INDEXED BY" clause).
- 8.11.6.4 The "REDEFINES" clause.
- 8.11.6.5 The datatype name.

8.11.7 Parse the Description Exception List once.

- 8.11.7.1 If the word "DESCRIPTION" was not retrieved, generate a "DESCRIBE" NDDL command for the record if a textual description exists. Generate separate "DESCRIBE" NDDL commands if textual information exists for data fields.
- 8.11.7.2 If the word "DESCRIPTION" was retrieved, omit generating "DESCRIBE" commands.

8.12 Select every set belonging to the database being processed from the RECORD\_SET Table. As

each set is retrieved process as follows:

- 8.12.1 Start generating the "DEFINE SET...FROM" NDDL command using the set's name and the set's owner record name.
- 8.12.2 Select all the members and required/optional indicator of the set named from the SET\_TYPE\_MEMBER Table. Generate the "TO" clauses using the member's record names and required/optional indicator obtained from this select.
- 8.12.3 If the DBMS of the database is TOTAL, retrieve the data field name of the owner record for the set and generate the "LINKED BY" clause.
- 8.12.4 Check what was retrieved from the Description Exception List.
  - 8.12.4.1 If the word "DESCRIPTION" was not retrieved, generate a "DESCRIBE" NDDL command for the set if textual descriptions exist.
  - 8.12.4.2 If the word "DESCRIPTION" was retrieved, omit generating a "DESCRIBE" command.
- 8.13 Check what was retrieved from the Description Exception List.
  - 8.13.1 If the word "DESCRIPTION" was not retrieved generate a "DESCRIBE" NDDL command for the database if textual descriptions exist.
  - 8.13.2 If the word "DESCRIPTION" was retrieved, omit generating a "DESCRIBE" command.
- 3.2.8.25 COPY DESCRIPTION - Copy CDM objects and/or descriptions.

A. Function:

COPY DESCRIPTION allows the NDDL user to copy descriptions from the CDM. These descriptions can be copied directly or the NDDL DESCRIBE commands can be generated on a file. The user may specify one, many or all description types.

B. CDM Requirements:

The object(s) identified must exist in the CDM.

The description type and description text of the object to be copied must exist.

If the copy involves the modeling objects (entities, attributes and relations), a current model must be established.

If "directly" is specified, a "to" object must exist, and a description text must not exist for that object.

C. Processing:

1. Retrieve the model name from the parser list. If no name was found, set the "from" model name to the current model name. Otherwise, verify the model name.
2. Retrieve the "directly" clause off the parser list. If "directly" was found, set a flag.
3. Retrieve the description type from the parser list. If ALL descriptions are to be copied, set a flag.
4. If the user wishes to copy the specified description types directly:
  - 4.1 Verify the object type and retrieve the object number for the object to be copied. If the object does not exist, issue a user error and stop processing.
  - 4.2 Verify the object type and retrieve the object number of the object to be copied to. If the to object does not exist, issue a user error and stop processing.
  - 4.3 For each description type specified:
    - 4.3.1 Select the description text of the first object and copy it to the second object and exit processing. If the description text does not exist, issue a user error.
5. If the user wishes to copy all descriptions directly:
  - 5.1 Verify the object type and retrieve the object number for the object to be copied. If the object does not exist, issue a user error and stop processing.

- 5.2 Verify the object type and retrieve the object number of the object to be copied to. If the "to" object does not exist, issue a user error and stop processing.
- 5.3 For every description of the "from" object in the CDM:
  - 5.3.1 Retrieve a description from the CDM for the object.
  - 5.3.2 Copy the description text of the first object to the second object.
- 5.4 Exit command processing.
6. If the user wishes to copy specified descriptions on file:
  - 6.1 Open the output file.
  - 6.2 Verify the object type and retrieve the object number for the object to be copied. If the object does not exist, issue a user error and stop processing.
  - 6.3 If the "to" object was specified, verify the object type and retrieve the object number of the object to be copied to. If the "to" object does not exist, issue a user error and stop processing.
  - 6.4 If the object is an attribute, entity or relation, generate an ALTER MODEL command.
  - 6.5 For each description type specified, generate a DESCRIBE command to create the description on file.
  - 6.6 Close the file and exit processing.
7. If the user wishes to copy all descriptions on file:
  - 7.1 Open the output file.
  - 7.2 Verify the object type and retrieve the object number for the object to be copied. If the object does not exist, issue a user error and stop processing.
  - 7.3 If the "to" object was specified, verify the object type and retrieve the object number of the object to be copied to. If the "to" object

does not exist, issue a user error and stop processing.

- 7.4 If the object is an attribute, entity or relation, generate an ALTER MODEL command.
- 7.5 Generate a DESCRIBE command to create the description on file for each description in the CDM.
- 7.6 Close the file and exit processing.

3.2.8.26 COPY DESCRIPTION TYPE - Generate NDDL commands to copy existing description types.

A. Function:

- 1. COPY DESCRIPTION TYPE generates NDDL commands to create one or more description types.
- 2. The generated NDDL is outputted to a user defined field or to the screen.
- 3. Allows the user to determine specific description types to be copied.

B. CDM Requirements:

A specified description type must exist.

C. Processing:

- 1. Open the user specified output device.
- 2. If "ALL" is specified, all description types stored in the CDM will be copied.
- 3. If certain description types are specified:
  - 3.1 The CDM table DESCRIPTION\_TYPE is checked to verify that the description type exists:
    - 3.1.1 If the description type does not exist:
      - 3.1.1.1 The user is notified and a warning message is issued.
      - 3.1.1.2 Continue processing with the next specified description type.
    - 3.1.2 If the description type does exist:
      - 3.1.2.1 Generate the create description type command.

3.1.2.2 Continue processing with  
the next description type.

3.2.8.27 COPY DOMAIN - Generate NDDL commands to make a copy of  
domains.

A. Function:

COPY DOMAIN allows the NDDL user to copy specified domains or all domains within the CDM. Optionally, all user defined data types are copied along with all textual descriptions for the domain and data types. All domain values and ranges are copied. At the user's discretion, the generated NDDL commands are either outputted to a screen or file.

B. CDM Requirements:

The domain/domains to be copied must exist in the CDM.

C. Processing:

1. Global input parameters are checked to determine if the generated NDDL commands will be appended to an existing output file or written to a new file or outputted to a screen. Input parameters are set with a previous "SET OUTPUT" statement.
2. A domain name or the word "ALL" is obtained from a Domain Name Parser List.
  - 2.1 If the Parser returns the word "ALL", process every domain retrieved from the DOMAIN\_CLASS table.
  - 2.2 If the Parser returns a domain name, verify that the domain name exists in the CDM.
3. Process each domain name obtained from the Domain Name Parser List or every domain in the CDM as follows:
  - 3.1 Select the standard data type's name, type-id, size and number of decimals from the CDM.
  - 3.2 Start generating the "CREATE DOMAIN" NDDL command specifying the standard data type with information obtained in step 3.1.
  - 3.3 Parse the User Data Type Exception List.
    - 3.3.1 If the word "USER" is retrieved, select all user defined data types and generate the "TYPE" clauses of the NDDL "CREATE DOMAIN" statement.



- 3.3.2 Save the datatypes retrieved in a temporary table.
- 3.3.3 If nothing is retrieved, omit the "TYPE" clauses.
- 3.4 Select all domain values and generate the "VALUE" clause.
- 3.5 Select all the domain ranges and generate the "RANGE" clause.
- 3.6 Check if a verification module exists for a domain.
  - 3.6.1 If one exists, generate a comment line informing the user of the module's name.
  - 3.6.2 If one doesn't exist, omit this comment line.
- 3.7 Parse the Description Exception List.
  - 3.7.1 If the word "DESCRIPTION" is not retrieved, generate a "DESCRIBE" NDDL command for the domain if textual information exists.
  - 3.7.2 Also generate "DESCRIBE" NDDL commands for the datatypes saved in step 3.3.2.
  - 3.7.3 If the word "DESCRIPTION" is retrieved, omit generating a "DESCRIBE" command.

3.2.8.28 COPY ENTITY - Copy a conceptual entity.

A. Function:

COPY ENTITY allows the user to perform the following:

1. Copy an entity within the same model, directly, giving it a different name.
2. Copy an entity within the same model, generating the NDDL commands that the user would otherwise have to type in.
3. Copy an entity from one model to another, directly, giving it a different name.
4. Copy an entity from one model to another, generating the necessary NDDL to create the entity in the other model. This kind of copy can include all subordinate entities in the tree structure of the target entity, or all the relations associated with the target entity.

5. Copy, at the user's discretion, the descriptions, aliases, keywords, attributes, keys for the entity and descriptions, aliases, and keywords for the entity's attributes.
6. Exclude, at the user's discretion, certain dependent entities from being copied when a copy with relation or structure is done.
7. Copy, at the user's discretion, just key information for the entity (when except nonkeys is specified).

B. CDM Requirements:

The entity to be copied must previously exist in the CDM. The entity copied to must not exist in the target model.

C. Processing:

1. Begin processing by determining if the copy is inter-model or intra-model. If a model name is specified, assume inter-model and check if the current model was established and the from-model exists.

Verify the from-entity name. If intra-model, verify that the to-entity name was entered, and if it was, verify that it doesn't already exist and that it is not the same name as the from-entity name.

If inter-model, and if no to-entity name was entered, assume it to be the same as the from-entity name, and verify that it doesn't already exist.

Retrieve the With clause off the parser list. Determine if the copy is directly or generated (On file). If directly was not specified, assume generated (see SET OUTPUT).

If intra-model and directly, perform step 2  
if intra-model and generated, perform step 3  
if inter-model and directly, perform step 4  
if inter-model and generated and no with clause (simple), perform step 5  
if inter-model and with structure, perform step 6  
if inter-model and with relation, perform step 7

2. For an intra-model copy entity directly, the database is updated directly.

- 2.1 A new unique number is obtained and the entity and its primary name are stored. The "TO" entity name will be made primary.
  - 2.2 If the user desires, all keywords for the old entity are also associated with the new entity.
  - 2.3 If the user desires, all descriptions from the old entity are copied to the new entity.
  - 2.4 No aliases will be copied in an intra-directly, because an alias can uniquely identify only one entity in the model.
  - 2.5 No attributes are copied, because they already exist in the model and are already owned by the "FROM" entity.
3. For an intra-model copy entity generated, NDDL is generated on a file or to the screen (see SET OUTPUT) to perform the copy.
    - 3.1 Generate a comment stating that the only NDDL that will execute successfully in this case, will be the create entity and its associated keywords and descriptions.
    - 3.2 Generate an 'ALTER MODEL' command and a 'CREATE ENTITY' command for the target entity. Also, if the user desires, generate the entity's 'DESCRIBE', 'ALTER ENTITY ADD KEYWORD', and the 'CREATE ALIAS' for entity commands.
    - 3.3 Generate, if the user desires, 'CREATE ATTRIBUTE' commands for each attribute owned by the entity, along with each attribute's descriptions, aliases, and keywords if the user desires. Also, generate 'ALTER ENTITY ADD OWNED ATTRIBUTE' commands for each attribute owned by the entity.
  4. For an inter-model copy entity directly, the current model is updated by creating the 'TO' entity.
    - 4.1 A new unique number is obtained and the entity and its primary name are stored. The "TO" entity name will be made primary.
    - 4.2 If the user desires, all keywords for the old entity are also associated with the new entity.
    - 4.3 If the user desires, all descriptions from the old entity are copied to the new entity.

- 4.4 If the user desires, the old entity's aliases will be created for the new entity in the current model, provided the alias does not exist in the current model.
- 4.5 If the user desires, for all attributes owned by the from-entity:
  - 4.5.1 A unique number is obtained and the attribute and its primary name are stored unless the attribute already exists in the to-model.
  - 4.5.2 If the user desires, the attribute's descriptions and keywords are associated with the new attribute in the to-model.
  - 4.5.3 If the user desires, the attribute's aliases will be created for the new attribute in the to-model unless the alias already exists in the to-model.
  - 4.5.4 The newly created attribute is stored as an owned attribute and an attribute use class for the new entity.
  - 4.5.5 If the user desires, all keys for the "FROM" entity can be copied to the new entity. The keys of the entity being copied, found in step 4.5, are stored in a table. If not already in the table, a new key is established with the new attribute as its key member. If the key class was on the list, a new key class member is created for the new attribute and the new key previously created.
  - 4.5.6 If nonkeys are excepted, only attributes that are part of a key are generated, and no 'ALTER ENTITY ADD OWNED ATTRIBUTE' command is generated for nonkey attributes.
- 5. For an inter-model copy simple (no with clause):
  - 5.1 Generate an 'ALTER MODEL' command and a 'CREATE ENTITY' command for the target entity. Also, if the user desires, generate the entity's 'DESCRIBE', 'ALTER ENTITY ADD KEYWORD', and the 'CREATE ALIAS' for entity commands.

- 5.2 Generate, if the user desires, 'CREATE ATTRIBUTE' commands for each attribute owned by the entity, along with each attribute's descriptions, aliases, and keywords if the user desires. If the attribute already exists in the to-model, a comment is generated. Also, generate 'ALTER ENTITY ADD OWNED ATTRIBUTE' commands for each attribute owned by the entity.
- 5.3 If attributes are excepted, keys and nonkeys are automatically excepted.
- 5.4 Generate, if the user desires, 'ALTER ENTITY ADD KEY' commands for the entity's keys. A comment is generated stating that some key members may not be owned attributes, and therefore, the NDDL will not execute successfully.
- 5.5 If nonkeys are excepted, only attributes that are part of a key are generated, and no 'ALTER ENTITY ADD OWNED ATTRIBUTE' command is generated for nonkey attributes.
- 6. For an inter-model copy with structure:
  - 6.1 Generate an 'ALTER MODEL' command.
  - 6.2 If the user specified a list of dependent entities to be excepted:
    - 6.2.1 Verify that the excepted entity exists in the from-model.
    - 6.2.2 Verify that the excepted entity is not the same as the entity being copied.
    - 6.2.3 Load each of the excepted entities into a COBOL table, along with all the entities in each of the excepted entities' dependent tree structure.
  - 6.3 Generate a 'CREATE ENTITY' command for the target entity. Also, if the user desires, generate the entity's 'DESCRIBE', 'ALTER ENTITY ADD KEYWORD', and the 'CREATE ALIAS' commands.
  - 6.4 Generate, if the user desires, 'CREATE ATTRIBUTE' commands for each attribute owned by the entity, along with each attribute's descriptions, aliases, and keywords if the user desires. If the attribute already exists

in the to-model, a comment is generated. Also, generate 'ALTER ENTITY ADD OWNED ATTRIBUTE' commands for each attribute owned by the entity.

- 6.5 If attributes are excepted, keys and nonkeys are automatically excepted.
- 6.6 The keys of the top entity are built. A search of keys and members is made of the original entity and stored in a structure. The original EC\_NO, KC\_NO, KC\_NAME, KCM\_TAG\_NO, and KCM\_TAG\_NAME are saved.
- 6.7 The entities, keys, and key members of the dependent tree structure of the original entity are stored in the same data structure of step 6.5. Do not store entities that have been excepted. A recursive search of the CDM's LINK\_RELATION and CATEGORY\_RELATION tables is done to accomplish this.
- 6.8 Next, for each level of relations in the structure find the keys for each dependent entity which contain attributes inherited via the relation. Populate a COBOL table with the level, dependent entity number and relation number. Do not populate the table for entities that have been excepted. This information will facilitate generation of the alter entity...add key clause at the appropriate level; i.e., after all the CREATE\_RELATION and CREATE\_CATEGORY commands have been generated, establishing all the inherited attributes making up the key.
- 6.9 An 'ALTER ENTITY' command for the new top entity can now be generated which declares each of the keys and its members found in the data structure stored in step 6.5.
- 6.10 Generate all the attributes, keys, and relations for the tree structure by traversing the constructed data structure and table to make sure that what is being generated is not part of an excepted entity.
7. For an inter-model copy with relation, the target entity is generated as in steps 6.1 through 6.5:
  - 7.1 If the user specified a list of dependent entities to be excepted:
    - 7.1.1 Verify that the excepted entity exists in the from-model.

- 7.1.2 Verify that no excepted entity is not the same as the entity being copied.
  - 7.1.3 Verify that a relation does exist from the copied entity to each excepted entity.
  - 7.1.4 Verify that no excepted entity is an independent entity in a relation with the copied entity, because only dependent entities can be excepted.
  - 7.1.5 Load each of the excepted entities into a COBOL table.
  - 7.2 Build a data structure for entities, keys, and key members for the top entity and all entities that are involved as independent entities in relations with the top entity.
  - 7.3 Generate 'CREATE RELATION' commands for all relations in which the copied entity is dependent.
  - 7.4 Generate 'ALTER ENTITY ADD KEY' commands for the copied entity.
  - 7.5 Generate 'CREATE RELATION' commands for all relations in which the copied entity is independent, unless one of the dependent entities was excepted. This is done by searching the table built in step 7.1.5.
  - 7.6 Generate "CREATE CATEGORY" commands for all relations that the copied entity is a category member.
  - 7.7 Generate "CREATE CATEGORY" commands for all relations where the copied entity is the generic entity, omitting the category members that were on the excepted entity list.
- 3.2.8.29 COPY HOST - Generate NDDL commands to make a copy of hosts.

A. Function:

COPY HOST allows the NDDL user to copy specified hosts or all hosts within the CDM. Data Base Managements Systems (DBMS) associated with the host are copied. Optionally, textual description for the host or PSBs associated with the host may be copied. At the user's discretion, the generated NDDL commands are either outputted to a screen or file.

B. CDM Requirements:

The host/hosts to be copied must exist in the CDM.

C. Processing:

1. Global input parameters are checked to determine if the generated NDDL commands will be appended to an existing output file or written to a new file or outputted to a screen. Input parameters are set with a previous "SET OUTPUT" statement.
  2. A host name or the word "ALL" is obtained from a Host Name Parser List.
    - 2.1 If a Parser returns the word "ALL", process all hosts retrieved from the HOST table in the CDM.
    - 2.2 If the Parser returns a host name, verify that the host exists in the CDM.
  3. Continue processing each host name obtained from the Host Name Parser List or every host in the CDM as follows:
    - 3.1 Start generating a "DEFINE HOST" NDDL command using the host's name.
    - 3.2 Parse the PSB Exception List.
      - 3.2.1 If the word "PSB" is not retrieved, select PSBs associated with the host and generate "DEFINE PSB" NDDL commands.
      - 3.2.2 If the word "PSB" is retrieved, omit these commands.
    - 3.3 Parse the Description Exception List.
      - 3.3.1 If the word "DESCRIPTION" is not retrieved, generate a "DESCRIBE" NDDL command for the host if textual description exists.
      - 3.3.2 If the word "DESCRIPTION" is retrieved, omit generating a "DESCRIBE" command.
- 3.2.8.30 COPY MAP - Generate NDDL commands to make a copy of mapping definitions.

A. Function:

COPY MAP allows the NDDL user to copy mapping definitions between conceptual and internal schemas.



Specified mapping definitions for an entity, relation, record or set can be copied, or all mapping definitions for an entity, relation, record or set can be copied within the CDM. Optionally, horizontal partitions and/or records unions are copied for entity and record mapping definitions. At the user's discretion, the generated NDDL commands are either outputted to a screen or file.

B. CDM Requirements:

The entity, attribute use, relation, record or set for which the map is defined must exist in the CDM.

C. Processing:

1. Global input parameters are checked to determine if the generated NDDL commands will be appended to an existing output file or written to a new file or outputted to a screen. Input parameters are set with a previous "SET OUTPUT" statement.
2. Parse the Mapping Exception List and retrieve the words "PARTITION" and/or "UNION", setting the except flags as needed.
3. Parse the Conceptual/Internal Object List and retrieve either the word "ENTITY", "RELATION", "RECORD" or "SET".
4. If the word "ENTITY" was returned in Step 3:
  - 4.1 Check what the Parser returned from the Object Name List.
    - 4.1.1 If the word "ALL" was returned process all primary entity names in the ENTITY\_NAME Table.
    - 4.1.2 If the word "ALL" wasn't returned, the first object name returned was that of an entity. Verify that the entity named exists in the CDM and return the primary entity name.
  - 4.2 Process the entity named or all entities in the CDM as follows:
    - 4.2.1 If partitions are to be generated, select all the partition numbers from the HORIZONTAL PART Table for the entity named. Generate a "CREATE PARTITION" NDDL command for each entity partition.

- 4.2.2 For each entity returned, generate a CREATE MAP command for the entity and record mapping. Select all the entity to record mappings from the EC\_RT MAPPING Table for the entity specified. Retrieve the distributed rules from DISTRIBUTED\_RULES table and generate retrieval and update commands.
- 4.2.3 Select all the tag names and numbers for the entity from the ATTRIBUTE\_USE\_CL Table, the database number and record name from the RECORD\_TYPE Table, preference number, map type, map and map category from the AUC\_IS\_MAPPING Table and database name from the DATA\_BASE Table.
- 4.2.4 For each tag name returned, map that tag for a stated preference:
  - 4.2.4.1 If Map-Type = "FIELD", generate a "CREATE MAP" NDDL command using the entity name and tag name. Use the database, record and field names in the "TO FIELD" clause. Continue NDDL command with MAP\_CATEGORY, MAP\_CLASS and preference number. Select the field names from PROJECT\_DATA\_FIELD Table for the database number, record name and tag number.
  - 4.2.4.2 If Map-Type = "SET", generate a "CREATE MAP" NDDL command using the entity name and tag name. Use the data base name, set name and AUC values in the "TO SET" clause. Continue NDDL command with MAP\_CATEGORY, MAP\_CLASS and preference number. Select all the set names and AUC values from the AUC\_ST\_MAPPING Table.
  - 4.2.4.3 If Map-Type = "COMPLEX", retrieve the module name, module instance, algorithm use-code and algorithm parameters from the COMPLEX\_MAPPING\_PARM Table for the tag number. Check the module name, use code and

instance against DI\_PARM to see if it is mapped to a data item. If so, omit generating the "DEFINE ALGORITHM" command. Also, check the module name and instance against an internal program table (MDTBL) where the module name and instance of prior algorithms that have been copied is stored. If it is not found, store the module name and instance on the table and generate a "DEFINE ALGORITHM" NDDL command. If it is found, omit generating a "DEFINE ALGORITHM" command since it would be a duplication of one already generated.

4.2.5 If unions are to be generated, select all the database number, record name combinations from the ECRTUD Table where the entity is part of a record union. Retrieve the data field name, union value and comparison operator from this select. The entity names are retrieved from the ENTITY\_NAME Table using the entity number from this select.

4.2.6 When the first entity is returned from the ECRTUD Table, check the database number, record name combination against the internal program table (UNTBL) where all prior record unions that have been copied are stored. If the record union is not found, update the program table (TABLE1) with the database number, record name combination and generate a "CREATE UNION" NDDL command for the record union. If the record union is found, exit from this select and omit generating a "CREATE UNION" NDDL command thereby eliminating duplicate commands.

5. If the word "RELATION" was returned in Step 3:

5.1 Check what the Parser returned from the Object Name List.

5.1.1 If the word "ALL" was returned, all relation to set maps in the CDM must be copied, ordered by independent entity, relation name and dependent entity.

5.1.2 If the word "ALL" was not returned, the first object named is the independent

entity, the second object named is the relation name and the optional third object named is the dependent entity. Verify that this link or category relation named exists in the CDM.

- 5.2 Process the relation named or all relations in the CDM as follows:
  - 5.2.1 Select all the maps between the relation named and record sets from the RC\_BASED\_REC\_SET Table.
  - 5.2.2 If maps are retrieved, start generating the "CREATE MAP" NDDL command using the independent entity name, relation class name and optional dependent entity name. Generate the "TO SET" clause for every set returned.
  - 5.2.3 If no maps are retrieved, omit generating a "CREATE MAP" NDDL command.
6. If the word "RECORD" was returned in Step 3:
  - 6.1 Check what the Parser returned from the Object Name List.
    - 6.1.1 If the word "ALL" was returned, the maps for all records in the CDM, ordered by record name, will be copied.
    - 6.1.2 If the word "ALL" was not returned, the first object named is the database name and the second object named is the record name. Verify that this database, record combination exists in the CDM.
  - 6.2 Process the record named or all records in the CDM as follows:
    - 6.2.1 Generate a CREATE MAP command for the entity to record mappings for all the entities in which the record named participates in. Select the entity name and number from the ENTITY\_NAME Table, the record name from the RECORD\_TYPE Table and the database name from the DATA\_BASE Table. Retrieve the distributed rules from DISTRIBUTED\_RULES table and generate retrieval and update commands.
    - 6.2.2 If partitions are to be generated, select all the entities and partition numbers from the HORIZONTAL\_PART Table

where the record named participates in the entity's partition.

- 6.2.3 When the first entity and partition numbers are returned, check against an internal program table (HPTBL) where all entity and horizontal partition numbers of partitions already copied are stored. If not found, store the entity and partition number into HPTBL and generate a "CREATE PARTITION" NDDL command.

If found, omit generating a "CREATE PARTITION" NDDL command for this entity, thereby avoiding duplicate commands.

- 6.2.4 Select the database number and record name from the RECORD\_TYPE Table, preference number, map type, map-class and map-category for the record named from the AUC\_IS\_MAPPING Table, associated tag and primary entity names from the ATTRIBUTE\_USE\_CL Table and the database name from the DATA\_BASE Table.

- 6.2.5 For each mapping of that tag for a stated preference returned in step 6.2.1:

6.2.5.1 If Map-Type = "FIELD", generate a "CREATE MAP" NDDL command using the entity name and tag name. Use the database, record and field names in the "TO FIELD" clause. Continue NDDL command with MAP\_CATEGORY, MAP\_CLASS and preference number. Select the field names from PROJECT\_DATA\_FIELD Table for the data\_base number, record name and tag number.

6.2.5.2 If Map-Type = "SET", generate a "CREATE MAP" NDDL command using entity name and tag name. Use the set names and AUC values in the "TO SET" clause. Continue NDDL command with MAP\_CATEGORY, MAP\_CLASS and preference number. Select all the set names and AUC values from the AUC\_ST\_MAPPING table.

6.2.5.3 If Map-Type = "COMPLEX", retrieve the module name,

module instance, algorithm  
use-code and algorithm  
parameters from the  
COMPLEX\_MAPPING\_PARM Table for  
the tag number. Check the  
module name, use code and  
instance against DI\_PARM to see  
if it is mapped to a data item.  
If so, do not generate the  
"DEFINE ALGORITHM" command.  
Check the module name, use code  
and instance against an  
internal program table (TABLE2)  
where the module name and  
instance of prior algorithms  
that have been copied is  
stored. If it is not found,  
store the module name and  
instance on the table and  
generate a "DEFINE ALGORITHM"  
NDDL command. If it is found,  
omit generating a "DEFINE  
ALGORITHM" command since it  
would be a duplication of one  
already generated.

- 6.2.6 If unions are to be generated, select  
all the data base number, record name  
combinations from the ECRTUD Table for  
the record named. GENERATE a "CREATE  
UNION" NDDL command.

7. If the word "SET" was returned in Step 3:

- 7.1 Check what the Parser returned from the Object  
Name List.
  - 7.1.1 If the word "ALL" was returned, all  
relation to set maps in the CDM, ordered  
by database name and set name will be  
copied.
  - 7.1.2 If the word "ALL" was not returned,  
the first object named is the database  
name, the second object named is the  
set name. Verify that the set named  
exists in the CDM.
- 7.2 Process the set named or all sets in the CDM as  
follows:
  - 7.2.1 Select the relation names from the  
RC\_BASED\_REC\_SET Table using the set  
named.

- 7.2.2 If a relation to set mapping is retrieved generate a "CREATE MAP...TO SET" NDDL command using the relation name and set name.

3.2.8.31 COPY MODEL - Generate NDDL commands to copy an existing model

A. Function:

Copy Model performs the following functions:

1. Generates NDDL commands to create a model along with its attributes and entities with their associated keywords, aliases and descriptions and the keys, key members, owned attributes, relations and migrated keys for the entities and dependent entities of the model being copied.
2. NDDL is outputted to a user defined file or to the screen.
3. Allows the user to determine whether attributes, entities, keys, nonkeys, relations, descriptions, aliases, keywords or specific entities will be excepted from the new model.

B. CDM Requirements:

1. The model to be copied must exist in the CDM.
2. The model to be copied into should not exist in the CDM.

C. Processing:

1. If the model to be copied (the from-model) is not specified, the model name defaults to the current model. The copy model command verifies that the from-model exists, and that the new model being created (the to-model) does not exist. Processing halts if any of the verification checks fail.
2. If there is a list of dependent entities excepted, a table is built containing the specified entities and their dependent entities.
3. If attributes are excepted:
  - 3.1 Set a flag indicating that exception.
  - 3.2 Set a flag indicating that keys are also to be treated as excepted.

- 3.3 Set a flag indicating that no 'migrates...set...' clause NDDL will be generated for link relations.
- 3.4 Category relations will be generated, but need editing for discriminating attributes.
- 4. If attributes are not excepted, the NDDL is generated to create the attribute, its keywords, description text and aliases for all attributes of the model with the following exceptions:
  - 4.1 Determine if the attribute is owned by an entity which exists on the previously created excepted entity table. If it is, do not generate the NDDL to create that attribute.
  - 4.2 If nonkeys are excepted, determine if the attribute is a key member. If it is not, do not generate the NDDL to create that attribute.
  - 4.3 If descriptions are excepted, do not generate the NDDL to create the description text.
  - 4.4 If attribute is not primary and aliases are excepted, do not generate the NDDL to create the alias for that attribute.
  - 4.5 If keywords are excepted, do not generate the NDDL to create the 'keyword' clause for that attribute.
- 5. If entities are excepted, the output file is closed and processing halts.
- 6. If entities are not excepted, the NDDL is generated to create the entity, its owned attributes, keywords and aliases for all entities of the model with the following exceptions:
  - 6.1 Determine if the entity exists on the previously created entity exception table. If so, do not generate the NDDL to create that entity.
  - 6.2 If attributes are excepted, do not generate the NDDL to create the 'owned attribute' clause for that entity.
  - 6.3 If non-keys are excepted, determine if the owned attribute is a key class member, if it is not, do not generate the 'owned attribute' clause.



- 6.4 If keywords are excepted, do not generate the 'keyword' clause for that entity.
- 6.5 If the entity is not primary and aliases are excepted, do not generate the 'create alias' clause for that entity.
- 7. If keys and/or attributes are not excepted:
  - 7.1 Build a temporary key list to store all the keys and key members for each entity with the exception of those entities on the table of excepted entities. This list will be used later to generate the keys for the entity after all the migrations have been determined.
  - 7.2 Another temporary list is built to store all the key information at each level in the model structure for every dependent entity, with the exception of those on the excepted entity table. This table contains attributes inherited via the relation for each level of relations in the model being copied.
  - 7.3 An ORACLE tree search is used to determine inheritance at all lower levels for an entity, after the top node has been identified, with the exception of those entities on the table of excepted entities. The model being copied must not contain any dependency loops. An 'alter entity add key' clause is generated for the top node in this model's tree structure.
- 8. If keys are excepted:
  - 8.1 Set a flag indicating that exception.
  - 8.2 Set a flag indicating that no key migration clauses will be generated for link relations.
  - 8.3 Do not build the temporary tables to store key and key member information.
  - 8.4 Category relations will be generated, but need to be edited because of keys automatically being migrated.
  - 8.5 Do not perform the ORACLE tree search.
- 9. If relations are excepted, no further processing is necessary; therefore, the user defined file is closed and processing halts. No category or link relations are generated.

10. If relations are not excepted, both of the temporary lists are used to generate NDDL statements necessary to create the relations for the entity. For each level of relations, migrate the keys and add keys for each dependent entity in the from model. NDDL commands are also generated for keywords and descriptions associated with the relations. The following exceptions apply:
  - 10.1 If the entity exists on the previously created exception list, do not generate the NDDL to create the relations for that entity.
  - 10.2 If keys and/or attributes and/or inherited attributes are excepted, do not generate the NDDL to create the 'migrates...set...' clause or link relations.
  - 10.3 If keys are excepted, do not generate the 'add key' clause for the dependent entities.
  - 10.4 If keywords are excepted, do not generate the 'keyword' clause for the relation.
11. Finally, the user defined file is closed and processing halts.

3.2.8.32 COPY MODULE - Generate NDDL commands to make a copy of user defined software modules.

A. Function:

COPY MODULE allows the NDDL user to copy specified modules or all modules within the CDM that were previously user defined (not generated). All the software module's parameters are copied. At the user's discretion, the generated NDDL commands are either out-putted to a screen or file.

B. CDM Requirements:

The software module/modules to be copied must exist in the CDM.

C. Processing:

1. Global input parameters are checked to determine if the generated NDDL commands will be appended to an existing output file or written to a new file or outputted to a screen. Input parameters are set with a previous "SET OUTPUT" statement.
2. A module name or the word "ALL" is obtained from a module Name Parser List.

- 2.1 If the Parser returns the word "ALL", process all modules retrieved from the SOFTWARE MODULE table with a status indicator of "C". The "C" indicates software modules created for the purpose of defining Complex Mapping Algorithms.
  - 2.2 If the Parser returns a module name, verify that the module exists in the SOFTWARE MODULE table and has a status indicator of "C".
  3. Continue processing each valid module name obtained from the Module Name Parser List or every module in the CDM with a status indicator of "C" as follows:
    - 3.1 Start generating a "DEFINE MODULE" NDDL command using the module's name and language.
    - 3.2 Select all parameters and their data type names for the module, sequenced by the PARM\_ID.
    - 3.3 Generate the "PARAMETER" clause for these parameters using the parameter's name and its data type name.
- 3.2.8.33 COPY RECORD - Generate NDDL Commands to make a copy of records.

A. Function:

COPY RECORD allows the NDDL user to copy specified records or all records within the CDM. The record layout is copied along with all the information pertaining to each data field. At the user's discretion, the generated NDDL commands are either outputted to a screen or file. Optionally, the "DESCRIBE" command with any textual descriptions may be copied.

B. CDM Requirements:

The record/records to be copied must exist in the CDM.

C. Processing:

1. Global input parameters are checked to determine if the generated NDDL commands will be appended to an existing output file or written to a new file or outputted to a screen. Input parameters are set with a previous "SET OUTPUT" statement.
2. Parse the Database List.

- 2.1 If a database name is retrieved, verify that the database named exists in the CDM and retrieve its DBMS.
  - 2.2 If nothing is retrieved, use the global input parameter containing the current database name that has been established for the session and retrieve its DBMS.
3. Generate an "ALTER DATABASE" NDDL command using the database name from step 2.
4. A record or the word "ALL" is obtained from a Record Name Parser List.
  - 4.1 If the Parser returns the word "ALL", retrieve every record from the RECORD TYPE Table for the database name obtained in step 2. Retrieve the record names sequenced by record name.
  - 4.2 If the Parser returns a record name, verify that the record named exists in the CDM and is associated with the database determined in step 2.
5. Process each record name obtained from the Record Name Parser List or every record in the CDM as follows:
  - 5.1 Start generating the "DEFINE RECORD" NDDL command using the record name.
  - 5.2 Select all areas associated with the record name and database number for a CODASYL DBMS.
    - 5.2.1 If areas are retrieved generate the "IN AREAS" clause using the area names.
    - 5.2.2 If no areas are retrieved omit this clause.
  - 5.3 Retrieve all the information from the DATA FIELD Table for the record named. Retrieve the data fields ordered by the record sequence number into an internal table.
  - 5.4 Generate the "WITH FIELD" clause for each data field associated with the record. Use the information obtained in step 5.3.
  - 5.5 For each field always generate:
    - 5.5.1 The level of each field (i.e. if not a subcomponent, the level number defaults to 1).
    - 5.5.2 Repeating field information (i.e. if the field doesn't repeat, occurs defaults to 1).

- 5.5.3 Whether the field is known or unknown to the DBMS.
  - 5.5.4 Whether the field is a unique or duplicate key.
  - 5.6 For each field generate, whenever applicable:
    - 5.6.1 The level of the subcomponent fields (i.e. if the field repeats, the level number will be greater than 1).
    - 5.6.2 Any filler information.
    - 5.6.3 Repeating field information (i.e. the "OCCURS" clause, the "DEPENDING ON" clause and "INDEXED BY" clause).
    - 5.6.4 The "REDEFINES" clause.
    - 5.6.5 The datatype name.
  - 5.7 Parse the Description Exception List.
    - 5.7.1 If the word "DESCRIPTION" is not retrieved, generate a "DESCRIBE" NDDL command for the record and a separate "DESCRIBE" NDDL commands for each data field.
    - 5.7.2 If the word "DESCRIPTION" is retrieved, omit generating "DESCRIBE" commands.
- 3.2.8.34 COPY SET - Generate NDDL commands to make a copy of sets.
- A. Function:

COPY SET allows the NDDL user to copy specified sets or all sets within the CDM. At the user's discretion, the generated NDDL commands are either outputted to a screen or file.
  - B. CDM Requirements:

The set/sets to be copied must exist in the CDM.
  - C. Processing:
    - 1. Global input parameters are checked to determine if the generated NDDL commands will be appended to an

existing output file or written to a new file or outputted to a screen. Input parameters are set with a previous "SET OUTPUT" statement.

2. Parse Database List.
  - 2.1 If a database name is retrieved, verify that the database named exists in the CDM.
  - 2.2 If nothing is retrieved, use the global input parameter containing the current database name that has been established for the session.
3. Generate an "ALTER DATABASE" NDDL command using the database name from step 2.
4. A set name or the word "ALL" is obtained from the Set Name Parser List.
  - 4.1 If the Parser returns the word "ALL", process every set retrieved from the RECORD\_SET Table. The sets retrieved are ordered alphabetically.
  - 4.2 If the Parser returns a set name, verify that the set named exists in the CDM.
5. Process each set name obtained from the Set Name Parser List or every set in the CDM as follows:
  - 5.1 Start generating the "DEFINE SET...FROM" NDDL command using the set's name and the set's owner record name.
  - 5.2 Select all the members and required/optional indicators of the set named from the SET\_TYPE\_MEMBER Table. Generate the "TO" clauses using the member's record names and required/optional indicator obtained from this select.
  - 5.3 If the DBMS of the database is TOTAL, retrieve the data field name of the owner record for the set and generate the "LINKED BY" clause.
  - 5.4 Parse the Description Exception List.
    - 5.4.1 If the word "DESCRIPTION" is not retrieved, generate a "DESCRIBE" NDDL command for the set if a textual description exists.
    - 5.4.2 If the word "DESCRIBE" is retrieved, omit generating a "DESCRIBE" command.

3.2.8.35 COPY VIEW - Generate NDDL commands to make a copy of views

A. Function:

COPY VIEW allows the NDDL user to copy specified views or all views within the CDM. All data items are copied. If the data item is an input or output parameter of a Complex Mapping Algorithm, the algorithm will be copied if the user chooses that option. Another user option is to have all textual description for the view copied. At the user's discretion, the generated NDDL commands are either outputted to a screen or file.

B. CDM Requirements:

The view(s) to be copied must exist in the CDM.

C. Processing:

1. Global input parameters are checked to determine if the generated NDDL commands will be appended to an existing output file, written to a new file, or outputted to a screen. Input parameters are set with a previous "SET OUTPUT" command.
2. Check the parser list to see if descriptions have been excepted. If they have, set a flag.
3. Check the parser list to see if algorithms have been excepted. If they have, set a flag.
4. A view name or the word "ALL" is obtained from the View Name Parser List.
  - 4.1 If the Parser returns the word "ALL", process every view retrieved from the USER\_VIEW table.
  - 4.2 If the Parser returns a view name, verify that the view named exists in the CDM.
5. Process each view name obtained from the View Name Parser List or every view in the CDM as follows:
  - 5.1 Select all the data item names for the view from the CDM Table DATA\_ITEM ordered by DI\_NO, and generate the data item names and associated data type names.
  - 5.2 Start generating the "CREATE VIEW" NDDL command using the view name retrieved in Step 4.
  - 5.3 Join the two CDM tables (VIEW\_EC\_XREF and ENTITY\_NAME) to select the primary entity names and entity numbers for the view. Store the

entity name(s) and entity number(s) retrieved during this select in an internal program table (EC-NAME-TABLE). Assign unique two character abbreviations for the entity names and store them together in EC-NAME-TABLE.

- 5.4 Select the tag name(s) and entity number(s) by joining two CDM tables (PROJECT DATA ITEM and ATTRIBUTE\_USE CL) and ordering by DI\_NO. Search the EC-NAME-TABLE generated in Step 5.3 to get the abbreviation. Generate the "AS SELECT" clause and the "DISTINCT" clause if DISTNCT-IND = "Y", using the abbreviations and tag names retrieved.
- 5.5 Generate the "FROM" clause of the "CREATE VIEW" NDDL command using the unique entity name(s) and abbreviation(s) stored in EC-NAME-TABLE.
- 5.6 Select all Qualification Criteria items for the view from CDM table VIEW\_QUALIFY\_CRITERIA, ordering by QC\_TEXT\_NO.
- 5.7 Generate the "WHERE" clause using the qualification text items retrieved in Step 5.6 in text line number order. If no qualification text items were selected, omit this clause.
  - 5.7.1 If a text identification type "T" is retrieved:
    - 5.7.1.1 Select a tag name and associated entity number for a given tag number from ATTRIBUTE\_USE\_CLASS.
    - 5.7.1.2 Using the unique entity names and abbreviations stored in EC-NAME-TABLE during Step 5.3, build a (ABBR.TAG-NAME) combination as a Qualification Criteria text line item.
  - 5.7.2 If a text identification type "C" is retrieved, list the Qualification Criteria text line item with single quotes.
  - 5.7.3 If the text identification type is not "T" or "C", list the Qualification Criteria text line items retrieved during Step 5.6 in the same format as they are stored in the CDM.
- 5.8 If algorithms have not been excepted:



- 5.8.1 Verify if a complex mapping algorithm exists using each of the data items of the view.
- 5.8.2 If a complex mapping algorithm doesn't exist, omit generating the "DEFINE ALGORITHM" NDDL command.
- 5.8.3 If a complex mapping algorithm exists:
  - 5.8.3.1 Retrieve the algorithm's software module name, module instance and use-code from the DI-PARM Table.
  - 5.8.3.2 Start generating the "DEFINE ALGORITHM" NDDL command using the information obtained in Step 5.8.3.1.
  - 5.8.3.3 Depending on the module name, module instance and algorithm use-code, select the algorithm data items, constant and/or attribute parameters.
  - 5.8.3.4 Retrieve all the parameters of the software module, in correct sequence, using the module name information obtained in Step 5.8.3.1.
  - 5.8.3.5 Generate the "USING PARAMETER" clause of the "DEFINE ALGORITHM" NDDL command with information obtained in Steps 5.8.3.3 and 5.8.3.4.
- 5.8.4 If the word "ALGORITHM" is retrieved, omit generating the "DEFINE ALGORITHM" NDDL command.

5.9 If descriptions were not excepted:

- 5.9.1 Generate a DESCRIBE NDDL command for any descriptions which exist with the view.

3.2.8.36 CREATE ALIAS - Create an alias for an entity or attribute.

A. Function:

Create Alias performs the following function:

1. Allows the user to add a secondary name, or alias, for any attribute or entity of the current model.

B. CDM Requirements:

The entity or attribute named in the command must exist in the user's current model.

C. Processing:

1. CREATE ALIAS shall access the object type from the user command. It must be either ATTRIBUTE or ENTITY. The command processor must then access the user entered identifier for the entity attribute and verify its presence with a database lookup. The potential new alias name is also verified to make sure it has not already been used. Finally, having the entity attribute number from the first verification, the new alias name can be inserted.

3.2.8.37 CREATE ATTRIBUTE - Create a conceptual attribute

A. Function:

Create Attribute performs the following functions:

1. Create an attribute for a model
2. Assign a domain for the attribute
3. Add keywords to an attribute

B. CDM Requirements:

A current model must be established.

C. Processing:

1. If a domain is specified, the Create Attribute command verifies the existence of the domain to which the attribute is being assigned. If the domain is not specified, the domain will default to zero or undefined.
2. Next, a check is performed to verify that the attribute does not previously exist in the model. Then the attribute is inserted into the ATTRIBUTE\_CLASS and ATTRIBUTE\_NAME tables. This attribute is created as the Primary attribute.
3. If keywords are to be added to the attribute, verify that the keyword has not previously been assigned to the attribute. Keyword references are then created

for the attribute by inserting into the AC\_KEYWORD table. Also, the keyword will be inserted into the KEYWORD table if it did not previously exist.

4. Processing halts if any of the verification checks fail.

3.2.8.38 CREATE CATEGORY - Create a conceptual category relation.

A. Function:

Create category performs the following functions:

1. Create a category relation for generic entity with the specified sub-entities (category entities).
2. Create associated keywords for the relation.
3. Migrate the primary key from the generic entity to the category entities.
4. Associate each category entity with a discriminator value of the discriminator in the generic entity.
5. Validate discriminator values with the domain values of the discriminating attribute.

B. CDM Requirements:

The generic and category entities must exist in the current model. The generic entity must have a primary key. The generic entity must have the attribute that is the discriminator. The discriminator attributes must have a domain with a specific set of values.

C. Processing:

1. The Create Category Relation process verifies that the generic and category entities exist in the current model. If these do not exist, an error is issued and processing is terminated. A check is made to determine if the relation to be created already exists for the generic entity. If one exists, an error is issued and processing is terminated. An attribute use for the discriminator in the generic entity is verified. If one does not exist, an error is issued and processing is terminated. The existence of a primary key in the generic entity is verified. If one does not exist, an error is issued and processing is terminated. The discriminator values for each of the category entities are verified to be unique. If they are not unique, an error is issued and processing is terminated.

2. An attribute use and an inherited attribute use for each category entity is created for each key member of the generic entity migrated to the category entities. The definition is inserted into the entities Relation Class, Category\_Relation, Category\_Number, Complete\_Relation, Key\_Class, and Key\_Class\_Number.
3. A domain value is associated with each category entity's discriminator value.
4. If a keyword is to be added to the relation, the keyword table is searched to determine whether the keyword exists. If it does not exist, the new keyword is inserted into the keyword table. The keyword is then associated with the relation.
5. If the set phrase is specified, TAG NAMED (the generic entity's tag name) is migrated with the new name of TAG\_NAME1 to the associated category member entity.

3.2.8.39 CREATE DESCRIPTION TYPE - Add new description type(s) in the CDM.

A. Function:

1. Create Description Type allows the user to add new, legal description types in the CDM.

B. CDM Requirements:

1. DESC\_TYPE must not be previously defined in the CDM.

C. Processing:

1. Convert DESC\_TYPE to upper case if entered in lower case.
2. CDM is checked to see if the DESC\_TYPE name exists.
  - 2.1 If the DESC\_TYPE name exists:
    - 2.1.1 The user is notified; issue error message.
    - 2.1.2 Continue processing with next DESC\_TYPE name.
  - 2.2 If the DESC\_TYPE name does not exist:
    - 2.2.1 Insert the new DESC\_TYPE name in the CDM.

3.2.8.40 CREATE DOMAIN - Create a domain

A. Function:

Create Domain performs the following functions:

1. Adds a new domain to the system.
2. Associates a standard data type with the new domain.
3. Associates optional user-defined data types with the new domain.
4. Associates valid values with the new domain according to the standard data types definition.
5. Associates valid ranges with the new domain according to the standard data types definition.
6. Generates a verification module of these new values and/or ranges for the new domain.

B. CDM Requirements:

1. The domain must not previously exist in the system.
2. The standard data type for the new domain must be specified in the first data type clause.
3. Additional data types may be specified in subsequent data type clauses. These data-types are considered user-defined data types for the new domain.
4. Values may be specified in value clause. Ranges may be specified in range clause.

C. Processing:

1. Create Domain verifies that the domain to be added does not exist in the system. Then the new domain is inserted into the DOMAIN CLASS table using a unique domain number from the reusable pool.
2. The data type specified for the standard or user-defined DATA\_TYPE\_NAME is verified as a legal data type, and the number of decimals, if specified, must not exceed the maximum field length of DATA\_TYPE\_NAME. The DATA\_TYPE\_NAME is then inserted into USER\_DEF\_DATA\_TYPE tables as a user-defined data type for the new domain.
3. Before the DATA\_TYPE\_NAME is inserted into the USER\_DEF\_DATA\_TYPE table for the standard data type,

any values and/or ranges specified are validated against the standard data type definition.

4. The DATA\_TYPE\_NAME is then inserted into the USER\_DEF\_DATA\_TYPE table as a standard data type for the new domain.
5. A verification module is generated for the new domain. The name of the verification module is inserted into CDM tables SOFTWARE\_MODULE, VERIF\_MODULE and MODULE\_PARAMETER. All valid values are inserted into the DOMAIN\_VALUE table. All valid ranges are inserted into the DOMAIN\_RANGE table.

#### 3.2.8.41 CREATE ENTITY - Create a conceptual entity

##### A. Function:

CREATE ENTITY will allow the NDDL user to perform the following functions:

1. Create a new entity for the current model
2. Associate attributes as owned attributes for this entity
3. Define keys of the owned attributes only, (unowned attributes must be migrated to the entity by use of the CREATE or ALTER RELATION commands)
4. Associate keywords with the entity

##### B. CDM Requirements:

The entity being created must not previously be found in the current model. Any attribute referenced must already be defined for the current model and must not be previously owned by any entity.

##### C. Processing:

1. First the entity itself is created. The name is accessed from the command and database lookup verifies that it is not already present. If not found, the entity is assigned a unique number and stored and the user entered name is stored as the entity's primary or preferred name, not an alias.
2. Any user specified attributes are then processed. Each attribute name must be found in the current model and also verified not to be owned by any other entity in the model. If these conditions are met, the attribute can be associated with the entity by

storing an occurrence of OWNED\_ATTRIBUTE. Finally, a unique tag number is obtained, and the attribute is also stored as an ATTRIBUTE\_USE\_CLASS of this entity.

3. When the user has specified any keys to be defined for this entity, the user may omit the attribute names for the key. Therefore, the key name may be taken as the attribute name if none follows the = sign. For each key specified the program must:
  - 3.1 Verify the key name was not used for this entity.
  - 3.2 Obtain a unique number for the key.
  - 3.3 Store a key occurrence for the entity first created. The key type is identified as primary or alternate. KEY\_TYPE is set to "PRIMARY" or "ALTERNATE", respectively. If primary or alternate is not specified, the first key will be the primary key and all following keys will be alternate keys.
  - 3.4 Process each attribute name specified for the current key.
    - 3.4.1 Determine if the attribute has already been associated with the entity as an ATTRIBUTE\_USE\_CLASS.
    - 3.4.2 If it had not, an attempt is made to make the attribute owned by the entity as specified in step 2 above.
    - 3.4.3 Finally, an occurrence of KEY\_CLASS\_MEMBER can be stored.
4. Finally, any keywords specified by the user can be related to the entity first created. For each keyword entered:
  - 4.1 The keyword is verified in the table of keywords (independent of model).
  - 4.2 If the keyword is new, then a unique number for the keyword is obtained and a keyword occurrence is stored.
  - 4.3 Finally, the keyword to entity cross reference (EC\_KEYWORD) is stored, relating the keyword to the entity just created.

### 3.2.8.42 CREATE MAP - Create CS-IS mapping

#### A. Function:

The CREATE MAP command allows the user to map an entity to a record, an attribute use class (AUC) to a data field or set for a single preference, or a relation to a set.

#### B. CDM Requirements:

1. The mapping definition must not exist in the CDM for an AUC to set map, a relation to set map, or an AUC to data field map unless the entity is horizontally partitioned.
2. All database names, record names, data field names, entity names, module names, set names, relations and AUC names referenced must exist in the CDM.
3. An entity to record mapping must exist before an attribute of that entity can be mapped to a data field of the record or a set using that record.

#### C. Processing:

1. Determine whether an entity, tag or relation mapping is specified.
2. If mapping is AUC:
  - 2.1 Verify the existence of the entity name and tag name specified.
  - 2.2 If MAP\_CATEGORY was not specified, default to "ACTIVE".
  - 2.3 If PREF\_NUMB was not specified, it is set to 1 by default for active mappings and 51 for passive mappings.
  - 2.4 Check the MAP\_CATEGORY and PREF\_NUMB combination. If PREF\_NUMB is between 1 and 50, MAP\_CATEGORY must be "ACTIVE". If PREF\_NUMB is between 51 and 99, MAP\_CATEGORY must be "PASSIVE". If not, flag as an error and exit the program.
  - 2.5 If MAP\_CLASS not specified, default to "ORIGINAL SOURCE" if PREF\_NUMB is equal to 1, "REPLICATION" if PREF\_NUMB is between 2 and 50, or "REDUNDANT" if PREF\_NUMB is between 51 and 99.



- 2.6 Check to see if this entity is involved in a horizontal partition. If so, set a flag to indicate this fact.
- 2.7 Determine whether the mapping is to a field or to a set.
- 2.8 For an AUC-to-datafield mapping:
  - 2.8.1 Check to see if a data field mapping already exists in the CDM for preference 1.
    - 2.8.1.1 If it does not exist and user input preference number is not equal to 1, issue an error message.
    - 2.8.1.2 If it does exist and the user input preference number is also 1, the entity must be horizontally partitioned, otherwise an error message is issued.
    - 2.8.1.3 If it does exist and the user input preference number is greater than 1, check to see if a data field mapping already exists in the CDM for user preference number. If a match was found and the entity was not horizontally partitioned, issue an error message.
  - 2.8.2 Retrieve database name, record name and data field name from the parser list; verify the data field name exists in the CDM.
  - 2.8.3 If horizontally partitioned, verify that this data field mapping is valid.
  - 2.8.4 Check if FIELD\_NAME can be mapped to. If the data field is both a group field and repeats, the field cannot be mapped to.
  - 2.8.5 Verify that an entity to record mapping exists for the entity of the attribute and the record of the data field being mapped.

- 2.8.6 Check if data types of the from and to data fields are compatible. If not, issue an error message.
- 2.8.7 Verify that the tag has not already been mapped to this record. If it has not, insert the AUC\_IS\_MAPPING entity occurrence and PROJECT\_DATA\_FIELD entity occurrence in the CDM.
- 2.9 For an AUC-to-set mapping:
  - 2.9.1 Check to see if a data field mapping already exists in the CDM for preference 1.
    - 2.9.1.1 If it does not exist and user input preference number is not equal to 1, issue an error message.
    - 2.9.1.2 if it does exist and the user input preference number is also 1, the entity must be horizontally partitioned, otherwise an error message is issued.
    - 2.9.1.3 if it does exist and the user input preference number is greater than 1, check to see if a data field mapping already exists in the cdm for user preference number. if a match was found and the entity was not horizontally partitioned, issue an error message.
  - 2.9.2 Retrieve and verify database name, set name and value string.
  - 2.9.3 Determine whether the set is a single member set. If not, flag a user error.
  - 2.9.4 Determine whether a relation class mapping exists. If it exists, flag a user error.
  - 2.9.5 Verify that the owner record and the entity to record mapping exists in the CDM.
  - 2.9.6 Verify the AUC values specified are unique for each set having the same owner record.

- 2.9.7 Determine whether all sets specified have the same owner record. If different owner records are found for the specified sets, check if the entity is horizontally partitioned.
  - 2.9.7.1 If the entity is not horizontally partitioned, issue an error message.
  - 2.9.7.2 If the entity is horizontally partitioned, verify the owner records of the sets specified are record fragments of the partitioned entity.
- 2.9.8 Verify at least two sets are specified having the same owner record.
- 2.9.9 Insert the AUC\_IS\_MAPPING entity occurrence and the AUC\_ST\_MAPPING entity occurrence in the CDM.
- 3. If mapping is Relation:
  - 3.1 Verify that the relation exists. If it does not, issue an error message. Check is made to either category or link relation, based on whether or not a dependent entity name is specified.
  - 3.2 Check to see if entity is involved in a horizontal partition. If so, set a flag to indicate this fact.
  - 3.3 Retrieve database name and verify that it exists in the CDM. Retrieve the set name and record name as well.
  - 3.4 Determine whether any previous mapping to the set exists. If it does, flag a user error.
  - 3.5 Determine whether set mapping contains member records of more than one type. If it is not a single member record type set, the member record name is required.
  - 3.6 Insert the RC\_BASED\_REC\_SET entity occurrence in the CDM.
- 4. If mapping is for an Entity:
  - 4.1 Verify that the entity already exists in the CDM. If it does not, issue an error message.
  - 4.2 Retrieve the distributed rules (update and retrieval) from the parser. If they were not specified, default to the value "DISALLOW".

- 4.3 Verify that the distributed rules don't already exist for the entity. If they do exist, issue an error message; otherwise insert the DISTRIBUTED\_RULES entity occurrence in the CDM.
- 4.4 Check to see if entity is involved in a horizontal partition. If so, set a flag to indicate this fact.
- 4.5 Retrieve database name and record name from the parser. Verify that they exist in the CDM. If they do not, issue an error message.
- 4.6 Insert the EC\_RT\_MAPPING entity occurrence in the CDM.

3.2.8.43 CREATE MODEL - Create a new IDEF1 model.

A. Function:

A new model is created as unchecked in the system.

B. CDM Requirements:

The model to be created must exist in the CDM.

C. Processing:

1. First, verify whether the model to be created exists in the system. If it does, flag an error; otherwise, obtain a model number for the model name.
2. Store the model number, model name into the CDM.

3.2.8.42 CREATE PARTITION - Represents an entity as being horizontally partitioned against two or more record types.

A. Function:

CREATE PARTITION allows the NDDL User to create a horizontal partition between record types for an entity.

B. CDM Requirements:

The entity to be partitioned must already exist in the database. The records named in the partition must currently exist in the database.

C. Processing:

1. If INTEGER1 is omitted on the CREATE PARTITION command, a "1" is assumed for HP\_NO.

2. The entity name (EC\_NAME) is used to verify that the entity exists on the ENTITY\_NAME table and the entity number is retrieved (EC\_NO).
3. The EC\_NO is used along with the HP\_NO to assure that this partition number doesn't currently exist for the entity class.
4. For each database name (DB\_NAME) Record Name (RT\_ID) combination, the following is done:
  - 4.1 At least two partitions must be specified for an entity (LIST-COUNT must be  $\geq$  "2" for DB\_LST).
  - 4.2 Verify that the database name and record name combination currently exist. Retrieve the record type number (RT\_NO) from this select.
  - 4.3 The EC\_NO and RT\_NO must be unique on the horizontal partition table. This will assure that the same record type won't exist under another partition number of the same entity class.
  - 4.4 If unique, insert EC\_NO, RT\_NO and HP\_NO into HORIZONTAL\_PART.

3.2.8.45 CREATE RELATION - Create a link relation

A. Function:

Create Relation performs the following functions:

1. Creates a link relation for user entered independent and dependent entity
2. Creates associated keywords for the relation
3. Migrates a key from the independent entity to the dependent entity

B. CDM Requirements:

The independent and dependent entity must exist in the current model. If the migrates clause is present, the key for the independent entity must exist in the current model.

C. Processing:

1. The Create Relation process verifies that both the independent and dependent entity exist in the

current model. If both do not exist, an error is issued and processing is terminated. A check is made to determine if the relation to be created already exists between the user entered entities. If one does exist, as above, an error is issued and processing is terminated.

2. Next, the process validates the cardinality of the relation to be created. If a cardinality is omitted by the user, the relation is assigned a default value. The default for the independent cardinality is a value of one. The default for the dependent cardinality is a value of zero for the left dependent cardinality and a value of 99 for the right dependent cardinality.
3. If the migrates clause is entered, the existence of the key for the independent entity is determined. If the key does not exist, an error is issued and processing is terminated. An attribute use and an inherited attribute use for the dependent entity is created for each key member of the independent entity migrated to the dependent entity. If the set phrase is specified, TAG\_NAME2 (the independent entity's tag name) is migrated with the new name of TAG\_NAME1.
4. If a keyword is to be added to the relation class, the keyword table is searched to determine whether the keyword exists. If it does not exist, the new keyword is inserted into the keyword table. The new keyword is then associated with the relation class.

3.2.8.46 CREATE UNION - Create a union of conceptual entities to represent a record type in the CDM.

A. Function:

CREATE UNION allows the NDDL User to create a union of record types for an entity.

B. CDM Requirements:

The entities' names must currently exist in the Integrated Model. Data fields stated must be associated to the database and record type named. The union must not currently exist in the CDM.

C. Processing:

1. Verify that the database name and record name currently exist. Retrieve the record type number (RT\_NO) for this selection.

2. The following is done for entities entered:
    - 2.1 At least two entities must be named to create a union.
    - 2.2 The Entity Name must currently exist on the ENTITY\_NAME Table and the entity number is retrieved (EC\_NO).
    - 2.3 Find the number of data fields, operators and strings that are associated with the entity. (CPFVAL)
  3. For each data field, verify that it is defined for the database and record type named and retrieve the data field number (DF\_NO).
  4. Verify that the union created doesn't currently exist on the ECRTUD table.
  5. Verify that the union discriminator value is compatible with the data type of the data field.
  6. Insert the record union into the CDM.
  7. Check for an integrity test failure, which indicates that no entity to record mapping exists for the record union.
- 3.2.8.47 CREATE VIEW - Create an external schema view and mappings to the conceptual schema.

A. Function:

Create a view of the entity and relation existing in the conceptual schema of the Common Data Model.

B. CDM Requirements:

The following elements must exist in the CDM:

1. Independent entities specified.
2. Dependent entities specified.
3. Entities of the view.
4. Data items defined and attribute use of the entities must be from the same domain.

C. Processing:

1. Verify if the view to be created already exists. If it does, flag a user error.
2. Obtain a unique number for the view to be created.
3. Retrieve the distinct clause from the parser and set the DISTINCT-IND flag.
4. Insert the view name and view number into the CDM table USER\_VIEW.
5. Construct in-code table to store parser information from the FROM CLAUSE, the SELECT CLAUSE, and the WHERE CLAUSE as follows:
  - 5.1 Retrieve an entity from the parser, verify it exists in the CDM, and store the information in the VIEW\_FROM\_LIST.
  - 5.2 Retrieve the abbreviation from the parser list. If no abbreviation exists, verify there is only one entity. If more than one entity exists, issue a user error. If an abbreviation is found, store the abbreviation in the VIEW\_FROM\_LIST and repeat Steps 5.1 and 5.2 for each entity. Unique abbreviations are verified for each subsequent entity.
  - 5.3 Retrieve a data item from the parser. Retrieve the data type, if specified, and store in the DATA\_ITEM\_LIST. Repeat this step for each data item.
  - 5.4 Retrieve a tag. If the first tag name is an asterisk:
    - 5.4.1 Verify that no data items were specified. If specified, issue an error message and exit processing.
    - 5.4.2 Verify only one entity was specified. If not, issue an error message and exit processing.
    - 5.4.3 Store asterisk and entity name in VIEW\_RETRIEVE\_LIST.
  - 5.5 If the first tag name was not an asterisk:



- 5.5.1 Retrieve the abbreviation.
- 5.5.2 If no abbreviation is found, verify there is only one entity. If not, issue an error message and exit processing.
- 5.5.3 Store tag name and abbreviation in VIEW\_RETRIEVE\_LIST and VIEW\_QUALIFY\_CRITERIA.
- 5.5.4 Repeat Steps 5.4 and 5.5 for each tag name. In addition, for each subsequent tag:
  - 5.5.4.1 Verify that the tag exists and retrieve the tag number.
  - 5.5.4.2 Verify that the tag exists in entity specified.
- 5.6 Retrieve the first operator from the WHERE clause. If there is no operator (i.e. no WHERE clause) verify that only one entity exists. If not, issue an error message and exit processing. If one entity exists, go to Step 6.
- 5.7 If an operator was found in Step 5.6, retrieve the first operand and build the Boolean List and CS\_EQUAL\_CRITERIA\_LIST as follows:
  - 5.7.1 Place any logic operator (AND, OR, XOR) in the Boolean List and set the BOOLEAN\_CSQ pointer to point to the next CS\_QUAL\_CRITERIA\_LIST entry.
  - 5.7.2 Place "(", ")" or "NOT" in the Boolean List and set the BOOLEAN\_CSQ pointer to the CS\_QUAL\_CRITERIA\_LIST entry.
  - 5.7.3 If operator = "BETW", expand the entry to store tag 1 > = operand 1 and tag < = operand 2.
  - 5.7.4 If operator = "NBTW" (Not Between), expand the entry to store tag < operand 1 or tag > operand 2.
  - 5.7.5 If operator = "NN" (Not Null) or "NL" (Null) place the operator in the Boolean List and set the BOOLEAN\_CSQ pointer to

point to the CS\_QUAL\_CRITERIA\_LIST entry where the tag is then stored.

- 5.7.6 If operator is a comparison operator (=, !=, U=, >, <, >=, <=) place the operator in the Boolean List and set the BOOLEAN\_CSQ pointer to point to the CS\_QUAL\_CRITERIA\_LIST entry where both + operands will be stored. Examine the type of operand 2; if a tag, store tag-no. If numeric constant, store as number. If character constant, store as character constant. If operator="LK"(LIKE) place the operator in the Boolean list and set the BOOLEAN\_CSQ pointer to point to the CS\_QUAL\_CRITERIA\_LIST entry where both operands will be stored. Insure that operand 2 is a character constant and store it as such

Upon completion of any one of the six options, retrieve the next operator and operand if necessary.

- 5.8 Take the Boolean List and CS\_QUAL\_CRITERIA\_LIST and expand them by first eliminating XORs by changing "operand 1 XOR operand 2" to "operand 1 or operand 2 and not (operand 1 and operand 2)." Then eliminate NOTs by changing comparison operators to their opposites.
6. Insert into VIEW\_EC\_XREF every entity for the view number.
7. If specified, verify that the number of data items equals the number of tags selected.
8. If the view is a one entity view and all the tags were selected ("\*"):
- 8.1 Retrieve each DOMAIN\_NO, TAG\_NO, and TAG\_NAME combination from the CDM. For each set selected:
- 8.1.1 Retrieve the standard data type name.
- 8.1.2 Retrieve a unique data item number.
- 8.1.3 Insert an instance into the CDM table DATA\_ITEM.
- 8.1.4 Insert an instance into the CDM table

PROJECT\_DATA\_ITEM.

9. If the view is a one entity view and specific tags were selected:
  - 9.1 Retrieve the domain number and tag number for the tag name.
  - 9.2 Retrieve the standard data type name.
  - 9.3 Retrieve a unique data item number.
  - 9.4 Insert an instance into the CDM table DATA\_ITEM.
  - 9.5 Insert an instance into the CDM table PROJECT\_DATA\_ITEM.
  - 9.6 Repeat Steps 9.1 through 9.5 for each tag.
10. If the view is a multiple entity view:
  - 10.1 Validate that the minimum number of joins (1 less than number of entities) were specified. If not, issue an error message and exit processing.
  - 10.2 Validate for each join condition:
    - 10.2.1 The join is based on two related entities, either through a link or category relation.
    - 10.2.2 The join is on at least one key class from the independent entity.
  - 10.3 Verify that the structure is a confluent hierarchy.
  - 10.4 Verify that the number of items selected are equal to the number of items in the data item list.
  - 10.5 For each data item:
    - 10.5.1 Retrieve the corresponding tag's EC\_NUMBER.
    - 10.5.2 Retrieve the domain number for the tag.
    - 10.5.3 Retrieve a unique data item number.

10.5.4 If the user specified data types:

- 10.5.4.1 Verify that the FLAG\_NAME and DATA\_ITEM names have same domains.
- 10.5.4.2 Retrieve the standard data type name.
- 10.5.4.3 Validate that the ES-CS and CS-ES conversion is possible.
- 10.5.4.4 Insert an instance into the CDM table DATA\_ITEM.
- 10.5.4.5 Insert an instance into the CDM table PROJECT\_DATA\_ITEM.

10.5.5 If the user did not specify data types:

- 10.5.5.1 Retrieve the standard data type name.
- 10.5.5.2 Insert an instance into the CDM table DATA\_ITEM.
- 10.5.5.3 Insert an instance into the CDM table PROJECT\_DATA\_ITEM.

- 11. If there was a WHERE clause, retrieve the information from the Boolean List and CS\_QUAL\_CRITERIA\_LIST and insert it into the VIEW\_QUALIFY\_CRITERIA table of the CDM. (An extra set of parenthesis is inserted around the join conditions.) Every combination of view number, tag number and entity number is inserted into VIEW\_QUAL\_XREF.

3.2.8.48 DEFINE ALGORITHM - Define the use of a software module as a complex mapping algorithm.

A. Function:

Define Algorithm performs the following functions:

- 1. Verifies that the software module exists in the CDM.

2. Verifies that the algorithm being defined uses the same number of parameters and in the same order, as previously defined for the module in the 'Define Module' command.
3. The algorithm specifies the conversion direction, i.e. from ES to CS to IS (update) or IS to CS to ES (retrieval). This command verifies whether the parameter coincides with legal NDML retrieval and update requests.
4. Verify that the input and output parameters specified for the algorithm have the same data type names as those specified for the module.
5. Input and output parameters may specify data items, data fields, attributes or a record. A constant value may be specified as an input parameter, and module status must be defined as the last output parameter. Validate these input and output parameters.
6. For a CS-IS algorithm, a mapping is inserted in AUC\_IS\_MAPPING for each tag in the algorithm.
7. A CS-IS or CS-ES algorithm is inserted into COMPLEX\_MAPPING\_PARM.

B. CDM Requirements:

1. The module must exist in the CDM.
2. The mapping types being specified for the algorithm, namely data items, attributes, data field and/or record, must exist in the CDM.

C. Processing:

1. A temporary structure is used to store the input and output parameters specified for the algorithm. Check that parameters coincide exactly with the list of parameters defined for the software module.
2. Extract from the parser lists the module name and the module instance. The module instance defaults to 1 if not specified. Verify that the software module being used as a complex mapping algorithm has been previously defined.
  - 2.1 If the module has not been defined, issue an error message and exit command processing.

- 2.2 Retrieve the module's parameters into the temporary table, in sequence.
3. Extract from the parser list the preference number. Preference number if not specified, defaults to 1.
  - 3.1 Extract from the parser's lists and populate the temporary table with inputs and outputs for the algorithm. External schema to conceptual schema or vice versa mappings involve data items and attributes. For these cases, verify that the data items and attributes have been previously defined. If mappings are between conceptual and internal, they involve attributes and data fields/records. For these cases, verify that the attributes and data fields or records have been previously defined. If any of the mapping types specified do not exist in the CDM, issue an error message and exit command processing.
4. The following checks are performed to ensure the legality of the algorithm.
  - 4.1 Verify that the number of parameters defined for the module agree with the number of parameters specified for the algorithm.
  - 4.2 If the algorithm is specified for use on an NDML retrieval:
    - 4.2.1 The input parameter may be a single attribute followed by any number of constants and a single output data item or
    - 4.2.2 The input parameter may be a single record type or any number of data fields (from the same record type) followed by any number of constants and any number of output attributes.
  - 4.3 If the algorithm is specified for use on NDML updates:
    - 4.3.1 The input parameter may be a single data item followed by any number of constants and a single output attribute or
    - 4.3.2 The input parameter may be any number of attributes (from the same entity class) followed by any number of constants, followed by any number of data fields

(from the same record type) or a single record.

4.4 The last parameter must be the module status, which conforms to the IISS error handling philosophy. If the algorithm conformed with legal mapping types and retrieval and update rules, the algorithm is inserted into the CDM. Check that the data types between the software modules parameters and the algorithm parameters are compatible.

4.5 If the mapping is between conceptual schema and internal schema, for each tag specified in the algorithm:

4.5.1 Check whether a mapping exists in AUC\_IS\_MAPPING for the stated preference, through a complex mapping, to the target record.

4.5.2 If a mapping does not exist, insert AUC\_IS\_MAPPING. Check for an integrity test failure. If one occurs, issue an error message that no entity to record mapping exists for the entity of the tag specified and the target record.

4.6 If no errors occurred, insert the algorithm into the appropriate CDM tables: AUC\_PARM, DF\_PARM, DI\_PARM, RI\_PARM, CONST\_PARM.

3.2.8.49 DEFINE DATABASE - Describe the definition of a database to the CDM internal schema.

A. Function:

The command defines a database to the CDM.

B. CDM Requirements:

1. The database to be defined must not exist in the CDM.
2. The host and DBMS must be previously defined.
3. In addition, the host must exist for the DBMS for which the database is to be defined.

C. Processing:

1. Verify the existence of the DBMS. If it does not exist, flag a user error.
2. Verify the existence of the database to be defined. If it exists, flag a user error.

3. Obtain from the parser lists a character and integer null value and NTM directory string. If not present, character and integer null value default to "NULL" for a relational DBMS and zeros for a non-relational DBMS. NTM directory defaults to "GR".
4. Verify the existence of the host. If it does not exist, flag a user error and exit.
5. Obtain a unique number for the database.
6. Insert the database entity occurrence.
7. If the DBMS is ORACLE:
  - 7.1 Check if the password is provided in the command. If not, flag a user error.
  - 7.2 Otherwise, insert the password entity occurrence of the database.
8. If the DBMS is IMS:
  - 8.1 Check if the start position and feedback length are provided in the command.
  - 8.2 If they are not in the command, flag a user error.
  - 8.3 Verify the existence of the PSB of the IMS database. If it does not already exist, flag a user error. Otherwise, insert the PCB entity occurrence.
9. If the DBMS is CODASYL (IDMS, IDS-II, or VAX-11):
  - 9.1 Check if the schema and database location clauses are present in the command. If they are not, flag a user error.
  - 9.2 Insert the schema and database location occurrence of the CODASYL database.
  - 9.3 Verify the existence of the area of the CODASYL database. If it already exists, flag a user error. Otherwise, insert the area occurrence.

3.2.8.50 DEFINE DBMS - Add a Database Management System (DBMS) definition to the CDM.

A. Function:

DEFINE DBMS performs the following function:



1. DEFINE DBMS allows the user to add a DBMS definition to the CDM and associated host identifications.

B. CDM Requirements:

1. The DBMS must not be previously defined.
2. DBMS model type must be one of the following:

H - Hierarchic  
R - Relational  
N - Network

C. Processing:

1. The CDM is checked to see if the DBMS has been previously defined.
2. If the DBMS has previously been defined, issue an error message and exit command processing
3. If the DBMS has not been previously defined:
  - 3.1 For each host:
    - 3.1.1 Check if host has been previously defined.
    - 3.1.2 If host has not been defined:
      - 3.1.2.1 Issue an error message and exit command processing
      - 3.1.2.2 Exit command processing
    - 3.1.3 Add Host/DBMS association
  - 3.2 Add the DBMS definition to the CDM.

3.2.8.51 DEFINE HOST - Add a host definition to the CDM.

A. Function:

DEFINE HOST performs the following function:

1. DEFINE HOST allows the user to add a new host to the CDM.

B. CDM Requirements:

1. The host must not be previously defined.

C. Processing:

1. The CDM is checked to see if the host is already defined.
2. If the host has been previously defined, issue an error message and exit command processing.
3. If the host has not been previously defined:
  - 3.1 The host definition is added to the CDM.
  - 3.2 If the Database Management System (DBMS) definition is to be associated with host:
    - 3.3.1 The CDM is checked to see if the DBMS is defined.
    - 3.3.2 If the DBMS definition exists, the DBMS/HOST association is added to the CDM.
    - 3.3.3 If the DBMS definition does not exist, issue an error message and exit command processing.

3.2.8.52 DEFINE MODULE - Define software modules in the CDM.

A. Function:

DEFINE MODULE allows the NDDL User to define software modules used for complex mapping algorithms.

B. CDM Requirements:

The module named must not currently exist in the CDM. The data type names of the parameters of the module must currently exist in the CDM.

C. Processing:

1. Verify that the module named (MOD\_ID) doesn't already exist in the SOFTWARE\_MODULE Table.
2. Retrieve the language (LANG\_NAME) as is from the Lang-List. No editing is done on this value.
3. The value for the module title and abstract is "USER DEFINED SOFTWARE MODULE".
4. The status indicator is a "C" indicating a module used for a complex mapping algorithm.

5. Insert into the SOFTWARE\_MODULE Table with this information.
6. Retrieve the parameter name (PARM\_ID) from the Parm-Lst. If no parameters are named, exit the program since parameters can be omitted.
7. Verify that the data type name (DATA\_TYPE\_NAME) retrieved from Data-List currently exists on the USER\_DEF\_DATA\_TYPE Table.
8. Insert each parameter in the order that they were entered into the MODULE\_PARAMETER Table.

3.2.8.53 DEFINE PSB - Add an IMS Program Specification Block (PSB) name to the CDM.

A. Function:

1. DEFINE PSB allows user to add a PSB name and associated host identification to the CDM.

B. CDM Requirements:

1. PSB name must not be previously defined in the CDM.
2. Host identification being added must be previously defined in the CDM.

C. Processing:

1. The CDM is checked to see if the PSB name has been previously defined.
2. If the PSB name has been previously defined, issue an error message and exit command processing.
3. If the PSB name has not been previously defined:

3.1 For the host ID:

- 3.1.1 Check if host has been previously defined.
- 3.1.2 If host has not been defined, issue an error message and exit command processing.
- 3.1.3 If host has been defined, add host/PSB name association to the CDM.

3.2.8.54 DEFINE RECORD - Creates a record type/table/segment for a previously defined database/PCB

A. Function:

DEFINE RECORD performs the following functions:

1. Verifies that the database exists.
2. Inserts the record into the CDM, provided it has not been previously defined.
3. If the DBMS is CODASYL, it associates the records with previously defined database areas.
4. Allows the fields in a record to be defined as repeating, group, elementary, indexed, component, redefined or key.
5. Indicates whether the field is known or unknown to the DBMS, i.e. can the DBMS address the field by name.

B. CDM Requirements:

1. The database must be previously defined.
2. CODASYL database areas must be defined.
3. The record and its fields must not be previously defined.
4. The occurs-depending-on field and field being redefined must be defined in the record prior to the depending field and the redefining field.

C. Processing:

1. The CDM is checked to see if the database exists. If it has not been defined, issue an error message and exit command processing.
2. If the database exists and the record does not previously exist, insert the record type in the CDM.
  - 2.1 For each CODASYL area specified, after verifying that the area exists, insert the record association into the database area assignment.

- 2.2 For a TOTAL database, the record name is truncated to four positions and inserted into the database area and the database area assignment.
- 2.3 For each field/column/element/item specified, store this information in a temporary structure and perform the following checks:
  - 2.3.1 Each subcomponent field must specify a level number to ensure subcomponent structures are accurately implemented.
  - 2.3.2 If the field name is "Filler", it must have a corresponding numeric filler-size. It must also be unknown to the DBMS.
  - 2.3.3 The data type name for an indexed field and depending on field must be numeric (i.e., the data type must not be of character type or contain decimal specification).
  - 2.3.4 The data-type-name entered by the user must have been previously defined in USER\_DEF\_DATA\_TYPE.
  - 2.3.5 If it is a repeating data field, the number of times it occurs must be greater than one. The indexed-by field, if not previously defined, is created internally and defined with a default numeric data type name and an index indicator of "G".
  - 2.3.6 If there is a depending on clause, the depending on field must be previously defined. The indexed-by field, if defined in the record, must appear before the field it is indexing.
  - 2.3.7 If the field is redefined, the redefines field name must be previously defined.
  - 2.3.8 The indexed-by field cannot be a key, group, component or a redefine.
  - 2.3.9 The depending on field cannot be a group, repeating, or an index itself.
  - 2.3.10 A repeating field or repeating group cannot be key.
  - 2.3.11 If a component data field is key, one of

its subcomponents cannot also be a unique key. It may be a duplicate key.

2.3.12 An ORACLE DBMS does not support repeating fields, indexes, redefinitions, keys or fillers.

2.3.13 For an ORACLE DBMS, all fields must be 01 level and known to the DBMS.

2.3.14 Redefinitions of a TOTAL database must be specified as a single field.

2.4 If the record layout conforms with the checks performed, the data fields are inserted into the CDM.

3.2.8.55 DEFINE SET - Define an internal set/path for CODASYL, TOTAL and IMS DBMSs

A. Function:

DEFINE SET performs the following functions:

1. Create a set/path for a CODASYL (VAX-11, IDMS, IDS), IMS or TOTAL DBMS
2. Allow a set between owner and multiple members for CODASYL, but only single member for other DBMS

B. CDM Requirements:

The database must be established during the session. The owner and member record types must exist. If creating a set for a TOTAL DBMS, the data field from the variable record must exist.

C. Processing:

1. DEFINE SET verifies the existence of the database/PCB in which the set is to be created. If the database is not specified, it defaults to the database established during the current session.
2. Next, a check is performed to verify that the set to be created does not exist. For an IMS database, the path name is derived by combining the owner record and member record names.
3. For a TOTAL or IMS database, verify that the owner and member records have previously been defined. In addition, verify that the data field of the variable (member) record to which the set is to be linked,

has also been defined. The set information is then inserted into the DF\_SET\_LINKAGE, SET\_TYPE\_MEMBER and RECORD\_SET tables.

4. For a CODASYL DBMS multiple members are allowed. Verify that the owner record and member record(s) have been previously defined. A required/optional entry must be specified for the member record types. The set information is then inserted into the SET\_TYPE\_MEMBER and RECORD\_SET tables.
5. Processing halts if any of the verification checks fail.

### 3.2.8.56 DESCRIBE - Describe Objects

#### A. Function:

The DESCRIBE command allows description text of the following object types to be entered, modified or deleted. The object types are:

1. Database
2. Set
3. Record
4. Data field
5. Domain
6. User data type
7. View
8. Data item
9. Keyword
10. Entity
11. Attribute
12. Relation, Link or Category

#### B. CDM Requirements:

The object to be described must exist in the CDM.

#### C. Processing:

1. The user entered description type is validated against the description type table maintained by the CDM administrator.

2. The object's existence is validated.
3. The description text originates from three sources: a text file, from the command line or from the UI/UTI Screen Editor. If the text originated from a text file and if the file contains data, only pre-existing description text of the proper type is deleted prior to the insertion of the new text. If the file contains no data, the description text is not deleted and an error message is generated. If the text originates from the command line, the old description text, if any, is deleted prior to the insertion of new text, if any. Therefore, to delete old description text, the user must describe the object with a null description on the command line.
4. If the description text is to come from the UI/UTI Screen Editor, pre-existing description text if any, is extracted from the database and written to a file.
5. The UI/VTI Screen Editor is called to edit the file. If changes are made, the old description text is replaced by the text output from the editor. If no editing changes were made, the database is not modified.
6. If object type is relation, the following must be specified: Category Relations- EC\_NAME1 and RC\_NAME, Link Relations- EC\_NAME!, RC\_NAME, and EC\_NAME2

3.2.8.57 DROP ALGORITHM - Delete all references to a software module defined as a complex mapping algorithm.

A. Function:

Drop Algorithm performs the following functions:

1. Verifies that the software module is currently defined for use as a complex mapping algorithm
2. Disassociates the software module as a complex mapping algorithm from the CDM
3. Note that the software module is not dropped. A "Drop Module" will delete the software module.

B. CDM Requirements:

The software module being used as a complex mapping algorithm must be defined in the CDM.



C. Processing:

1. Extract from the parser lists the module to be dropped, the module instance, whether the module was specified for use on NDML retrieval or update requests, and the preference number. Module instance, if not specified, defaults to 1. Preference number, if not specified, defaults to 1.
2. The CDM is checked to see if the software module is used as a complex mapping algorithm. If it has not been defined, issue an error message and exit command processing.
3. Using a view of COMPLEX\_MAPPING\_PARM and AUC\_IS\_MAPPING, retrieve all the tags that are defined for that algorithm and have been mapped for the specified preference through a complex mapping algorithm. These tags are saved in a table (TAGTBL).
4. Drop the association of the software module as a complex mapping algorithm from the CDM Table COMPLEX\_MAPPING\_PARM.
5. Check whether the algorithm transformed between conceptual and internal. If it is a CS-IS algorithm, the re-mapping has to be deleted.
  - 5.1 Delete from AUC\_IS\_MAPPING for the case where the tag is mapped through a complex mapping algorithm but the tag is no longer referenced in any algorithm in COMPLEX\_MAPPING\_PARM.
  - 5.2 Check if the EC\_NO and RT\_NO in the deleted mapping is used in any other AUC\_IS\_MAPPING. If not, delete the entity to record mapping (EC\_RT\_MAPPING) and check if the entity of the deleted mapping is used in any other entity to record mapping. If not, delete the DISTRIBUTED\_RULES for the entity.
  - 5.3 For each tag saved in Step 3, delete from AUC\_IS\_MAPPING for the case where a tag is mapped through a complex mapping algorithm but the tag is used in conjunction with a data item (a CS-ES Algorithm) in COMPLEX\_MAPPING\_PARM. In this case, the CS-IS mapping in AUC\_IS\_MAPPING is no longer necessary.
    - 5.3.1 Delete from AUC\_IS\_MAPPING for the case where a tag is mapped through a complex mapping algorithm but the tag is used in conjunction with a data item (a CS-ES

Algorithm) in COMPLEX\_MAPPING\_PARM. In this case, the CS-IS mapping in AUC\_IS\_MAPPING is no longer necessary.

3.2.8.58 DROP ALIAS - Deletes an alias established for an attribute or entity name.

A. Function:

DROP ALIAS performs the following functions:

1. Verifies whether the alias is for an attribute or entity
2. Verifies that the alias exists for the attribute or entity for a specified model
3. Deletes the Alias for the attribute or entity

B. CDM Requirements:

DROP ALIAS requires the presence of an alias for the attribute or entity.

C. Processing:

1. The DROP ALIAS process will determine whether the alias is of an attribute or entity and verifies if the attribute or entity exists for the specified model.
2. The process will then verify the alias name and delete the alias for the attribute/entity from the CDM Entity or Attribute name table.

3.2.8.59 DROP ATTRIBUTE - Drop a Conceptual Attribute

A. Function:

DROP ATTRIBUTE deletes one or more user specified attributes from the CDM.

B. CDM Requirements:

The attribute(s) to be dropped must exist in the current model.

C. Processing:

1. DRPATT, after verifying that the attribute exists, determines whether the attribute to be dropped is owned. If so, the attribute is deleted from the OWNED\_ATTRIBUTE and ATTRIBUTE\_USE\_CL tables.

2. If the attribute to be dropped has been inherited, all instances of inheritance are deleted. The tables effected are INHERITED\_ATT\_USE, ATTRIBUTE\_USE\_CL, BY\_CLASS\_MEMBER and DESC\_TEXT. The ORACLE tree search feature is used to identify inheritance at all lower levels.
3. After all owned or inherited instances of the attribute are deleted, the attribute is deleted from ATTRIBUTE\_CLASS, ATTRIBUTE\_NAME, AC\_KEYWORD DESC\_TEXT, AUC\_CONSTRAINT and CONSTRAINT\_INPUT.
4. If the attribute deleted was a key member, and if it was the only member of a particular key the corresponding entries in the KEY\_CLASS, COMPLETE\_RELATION and KEY\_CLASS\_MEMBER tables are deleted.
5. If mappings exist anywhere, an error is issued, processing is terminated, and no deletions are performed.
6. If attribute is a discriminator in a category relation, the category relation is dropped with complete ripple down.

3.2.8.60 DROP CATEGORY - Deletes the category relation and all references to the relation from the CDM database.

A. Function:

The DROP CATEGORY RELATION process performs the functions for one or more category relations:

1. Verifies the category relation exists.
2. Verifies the generic and category entities exist.
3. Deletes the category relation from the CDM.
4. Deletes the relation from the CDM.
5. Deletes the complete relation.
6. Deletes all keys that have migrated from the he relation.
7. Deletes all keywords associated with the relation.
8. Deletes all textual descriptions associated with the relation.

B. CDM Requirements:

The DROP CATEGORY RELATION process requires the presence of the relation, generic entity, and category entities within the current model.

C. Processing:

1. The DROP CATEGORIZATION process verifies that the generic entity and the categorization relation exists in the CDM. If these do not exist, an error is issued and processing terminates.
2. The process returns the key that allows the process to determine of the associated items.
3. Using the key, the process deletes all migrating key members based on the relation. In turn, each key member is deleted from KEY CLASS\_MEMBER, ATTRIBUTE\_USE\_CL, and INHERITED\_ATTRIBUTE\_USE based on its association to the relation.
4. After all the key members have been deleted, the process deletes the association in the complete relation, the relation class, the category relation, tag constraints, any keywords associated with the relation, and all textual descriptions of the relation.
5. If any mappings exist anywhere, an error is issued, processing is terminated, and no deletions are performed.

3.2.8.61 DROP DATABASE - Delete a database from the Common Data Model.

A. Function:

DROP DATABASE deletes all references to the database from the Common Data Model.

B. CDM Requirements:

1. The database or IMS PCB must exist in the Common Data Model.
2. The Common Data Model database cannot be dropped.

C. Processing:

1. Retrieve from the parser lists the database name and DBMS name. Verify that the database or IMS PCB to be dropped does exist.
  - 1.1 If the database does not exist, issue an error message and exit command processing.
2. Verify whether any objects of the database (records, fields, etc.) have been mapped to. Verify the mapping from the following tables:
  - 2.1 AUC\_IS\_MAPPING (CS-IS MAPPING)
  - 2.2 RC\_BASED\_REC\_SET (Relation - Set Mapping)
  - 2.3 ECRTUD (Record Union)
  - 2.4 HORIZONTAL\_PART (Horizontal Partition)

If a mapping is found, issue an error message and exit command processing.
3. If no mapping is found, delete all references of the database or PCB from the Common Data Model. Process the deletes by records, fields and sets of the database.
4. Delete the textual descriptions for the database and all records, fields and sets of the database.

3.2.8.62 DROP DBMS - Drops a Database Management System definition from the CDM.

A. Function:

DROP DBMS performs the following function:

1. DROP DBMS allows the user to drop a Database Management System definition from the CDM.

B. CDM Requirements:

1. The Database Management System must be defined. There must be no database/host association in the CDM.

C. Processing:

1. The CDM is checked to see if the Database Management System definition exists.

2. If the DBMS definition exists:
  - 2.1 The CDM is checked for any hosts associated with the DBMS.
  - 2.2 If any hosts are associated with the DBMS, the user is notified and the program is exited.
  - 2.3 If no hosts are associated with the DBMS, the DBMS definition is deleted from the CDM.
3. If the DBMS definition does not exist, issue an error message and exit command processing.

3.2.8.63 DROP DESCRIPTION TYPE - Drop description type(s) from the CDM.

A. Function:

1. DROP DESCRIPTION TYPE allows user to drop a DESC\_TYPE from the CDM.

B. CDM Requirements:

1. The DESC\_TYPE must be previously defined in the CDM.

C. Processing:

1. Convert DESC\_TYPE to upper case if entered in lower case.
2. CDM is checked to see if the DESC\_TYPE name exists.
3. If the DESC\_TYPE name exists:
  - 3.1 The CDM is checked for any object(s) associated with the DESC\_TYPE name.
  - 3.2 If any object(s) are associated with the DESC\_TYPE name (referential integrity check fails):
    - 3.2.1 The user is notified; issue error message.
    - 3.2.2 Continue command processing with the next DESC\_TYPE name.
  - 3.3 If no object(s) are associated with the DESC\_TYPE name, the DESC\_TYPE name is deleted from the CDM, if no referential integrity check fails.
4. If the DESC\_TYPE name does not exist, issue an error

message and continue command processing.

3.2.8.64 DROP DOMAIN - Drop a domain definition from the CDM.

A. Function:

DROP DOMAIN allows the NDDL User to drop the definitions of one or more domains from the CDM.

B. CDM Requirements:

The domains to be dropped must currently exist, independent of model, and no attributes, data items or data fields must be associated with the data types defined for the domain.

C. Processing:

1. For each domain name specified by the user, the following is done:
  - 1.1 The domain name is verified, retrieving its domain number.
  - 1.2 Any attribute class associated with the domain is searched. This search is possible because the standard data type of the domain is the only data type associated with attributes.
  - 1.3 For each data type associated with the domain:
    - 1.3.1 Usage of the data type by any internal schema data fields is determined and displaced to the user.
    - 1.3.2 Usage of the data type by any external schema data items is determined and displaced to the user.
  - 1.4 If the usage count of any data types of this domain is not zero, the user is given a message and the domain will not be deleted.
  - 1.5 If the domain can be deleted, the following steps are executed:
    - 1.5.1 All values associated with the standard data type of the domain to be deleted are dropped.
    - 1.5.2 All ranges associated with the standard data type of the domain to be deleted are dropped.
    - 1.5.3 All data types can be deleted. The data type descriptions are also deleted.

1.5.4 Drop the verification module from the CDM tables SOFTWARE\_MODULE, MODULE\_PARAMETER and VERIF\_MODULE.

1.5.5 The DOMAIN CLASS entry itself can be deleted, along with any associated text descriptions. Any textual information associated with the domain class itself is deleted.

3.2.8.65 DROP ENTITY - Drop a conceptual entity

A. Function:

DROP ENTITY performs the following functions:

1. Deletes one or more user specified entities from the CDM
2. Deletes the primary name of the entity and all associated aliases, keywords and description text
3. Deletes all associated owned attributes, attribute use classes and inherited attributes
4. Deletes all associated keys and key members
5. Deletes all relations associated with the entity and its keywords

B. CDM Requirements:

1. Each entity to be dropped must exist in the current model.
2. No mappings must exist for each entity to be dropped.

C. Processing:

1. The DROP ENTITY process verifies that the entity to be dropped exists in the current model. If it does not exist an error is issued and processing is terminated. Processing then checks if any union mappings exist for the entity or if the entity is horizontally partitioned. If so, an error is issued and processing is terminated.
2. The process next determines all owned attributes and attribute uses for the entity and these are dropped. The tag constraints are also dropped. Further, its keys and attributes inherited via the migrated keys and key members are also dropped. If the deletion of the entity resulted in any empty keys for the model, these are then deleted.



3. All link relations where the entity is independent and dependent are deleted as is its associated keywords. All category relations where the entity is generic entity are deleted. If the entity is a category member, the category member occurrence is deleted and if it is the only category member, the category relation is then deleted. For each relation the entity participates in, if any CS/ES or Relation\_Set mappings exist, an error is issued and processing is terminated.
4. The primary name of the entity and all of its aliases, keywords, description text and entity constraints are deleted from the model. If any complex or CS/IS mappings exist for the entity, an error is issued and processing halts.

3.2.8.66 DROP FIELD - Drops an existing field of a previously defined record and database.

A. Function:

DROP FIELD performs the following functions:

1. Verifies that the database exists
2. Verifies that the record type exists
3. Verifies that the fields to be dropped exist
4. Drops the datafield and all of its subcomponents
5. If the field being dropped is used as index or occurs-depending-on by another data field, the latter's indexed-by or occurs-depending is set to null and updated in the record layout. If the field being dropped is indexed, drop the indexing field if it is an index-indicator of "G"; otherwise, set the index indicator of the index field to "N"
6. If the field being dropped is redefined by another data field, the latter is dropped along with its subcomponent fields
7. All description text is deleted for each data field to be dropped

B. CDM Requirements:

The database, record type and fields to be dropped must be previously defined.

C. Processing:

1. The CDM is checked to see if the database exists. If it has not been defined, issue an error message and exit command processing.
2. The CDM is checked to see if the record exists. If it has not been defined, issue an error message and exit command processing.
3. All existing data fields belonging to the record are retrieved into a temporary structure. Syntax and semantic checks are performed within this temporary table before the data fields are deleted or modified, as necessary.
4. For each data field to be dropped:
  - 4.1 The CDM is checked to verify the data field is not mapped to. If the data field participates in a union mapping, complex mapping algorithm or is mapped to a tag, issue an error message. Continue processing with the next field.
  - 4.2 Retrieve into a second temporary table the data field and its component data fields. Perform the following steps for each data field in this table.
    - 4.2.1 If the data field being dropped is indexed, drop the indexing field if it has an index indicator of "G"; otherwise, set the index field indicator to "N" and retain the field.
    - 4.2.2 If the data field being dropped is an index of another data field in the record, update the latter's indexed-by to null.
    - 4.2.3 If the field is the occurs-depending-on of another data field in the record, update the latter's depending-on field to null.
    - 4.2.4 If the data field appears as the redefines of another field in the record, drop the redefining field and all of its subcomponents.
5. The second temporary table contains a list of all the fields to be deleted. For each field to be dropped:

- 5.1 If it is a TOTAL DBMS, verify if the data field is used as a link to a record set.
- 5.2 If a data field to set linkage exists, verify that the set is not mapped to a tag or a relation. If it is mapped, issue an error message and continue processing. If no set mapping exists:
  - 5.2.1 Drop the data field to set linkage.
  - 5.2.2 Drop the set and any of its members.
  - 5.2.3 Delete the textual description for the set.
- 5.3 Delete the textual description for the data field.
- 5.4 Drop the data field from the CDM table DATA\_FIELD.

3.2.8.67 DROP HOST - Delete a host definition from the CDM.

A. Function:

DROP HOST performs the following functions:

1. DROP HOST allows the user to delete a host definition from the CDM

B. CDM Requirements:

1. The host must be defined on the CDM.

C. Processing:

1. The CDM is checked to see if the host is defined.
2. If the host is not defined, issue an error message and exit command processing.
3. If the host is defined:
  - 3.1 If no DBMS is associated with the host, the host is deleted from the CDM.
  - 3.2 If a DBMS is associated with host, the user is notified. The program is exited.

3.2.8.68 DROP KEYWORD - Delete an object keyword.

A. Function:

DROP KEYWORD performs the following function:

1. Delete the named keyword and associations with any attribute, entity and/or relation class.

B. CDM Requirements:

The keyword to be dropped must exist in the CDM.

C. Processing:

1. DROP KEYWORD verifies the existence of the keyword. The keyword is deleted from the AC\_KEYWORD, EC\_KEYWORD, RC\_KEYWORD and from the IISS\_KEYWORD tables.
2. Processing halts if any of the verification checks fail.

3.2.8.69 DROP MAP - Delete a CS-IS Mapping

A. Function:

DROP MAP performs the following functions:

1. Deletes AUC to field mappings for a stated preference.
2. Deletes AUC to set mappings for a stated preference.
3. Deletes all mappings for all preferences for a specified entity.
4. Deletes relation to set mappings.

B. CDM Requirements:

The map to be dropped must exist in the CDM.

C. Processing:

1. For an AUC mapping:
  - 1.1 Check whether a mapping exists through a software module. If so, flag user error, since a complex mapping is deleted by the Drop Algorithm command.
  - 1.2 If PREF\_NUMB equals 1, delete all AUC\_IS\_MAPPING, PROJECT\_DATA\_FIELD and

AUC\_ST\_MAPPING entity occurrences in the CDM for the specified tag.

- 1.3 If PREF\_NUMB is greater than 1, delete all AUC\_IS\_MAPPING, PROJECT\_DATA\_FIELD and AUC\_ST\_MAPPING entity occurrences in the CDM for the specified tag and preference number.
- 1.4 No matter what the specified preference number, after deleting the AUC\_IS\_MAPPING entity occurrence, check to see if the entity and record number of the deleted map is used for any other map. If not, delete the entity to record mapping in the EC\_RT\_MAPPING Table of the CDM.
- 1.5 After the maps are deleted, count the number of entity to record maps for the entity specified. If none exist, delete the DISTRIBUTED\_RULES occurrence for the entity specified.
- 1.6 If the user specified all mappings to be deleted for an entity, process as follows: For each tag selected, delete all mappings from the CDM tables AUC\_IS\_MAPPING, AUC\_ST\_MAPPING and PROJECT\_DATA\_FIELD. Delete all entity to record mappings from the EC\_RT\_MAPPING CDM Table for the entity the attribute is owned by. Delete the DISTRIBUTED\_RULES for the entity.

2. For a RELATION mapping:

- 2.1 Verify existence of link or category RELATION in the CDM. If it does not exist, flag user error.
- 2.2 Delete RC\_BASED\_REC\_SET entity occurrences in the CDM.

3.2.8.70 DROP MODEL - Delete a model from the CDM.

A. Function:

DROP MODEL performs the following functions:

1. Drops all entities associated with the model
2. Drops all attributes, attribute uses, and inherited attributes associated with the model
3. Drops all keys and key members associated with the model
4. Drops all relations, category and link, associated with the model

5. Drops all descriptions, aliases and keywords for the entities, attributes and relations associated with the model

B. CDM Requirements:

The model to be dropped must exist in the CDM.

C. Processing:

1. DROP MODEL verifies that the model to be dropped exists. The INTEGRATED\_MODEL cannot be dropped.
2. For each entity found in the model, its owned attributes, keywords, descriptions and the entity itself is dropped. Further, its keys and attributes inherited via the migrated keys and key members are also dropped. Relations where the entity is both dependent and independent and its associated keywords and descriptions are deleted.
3. For each attribute in the model, the attribute keywords, descriptions and the attribute itself is dropped.
4. Processing halts if any of the verification checks fail.

3.2.8.71 DROP MODULE - Drop software modules from CDM

A. Function:

DROP MODULE allows the NDDL user to drop software modules along with their parameters from the CDM.

B. CDM Requirements:

The modules named (MOD\_ID) must currently exist in the CDM. Any module associated as a complex mapping algorithm can't be deleted.

C. Processing:

1. Verify that the modules named (MOD\_ID) exist on the SOFTWARE\_MODULE Table.
2. If the status indicator is a "C" (complex mapping algorithm):
  - 2.1 Delete module parameters associated with the software module.
  - 2.2 Check the return status.

- 2.2.1 If return status = KES-TYPE-2-FAIL, an error occurred during a referential integrity test; i.e., the module is associated as a complex mapping algorithm.
  - 2.2.2 Issue an error message and retrieve next module name.
  - 2.3 Delete the module from the SOFTWARE\_MODULE Table.
  - 3. If the status indicator is not a "C" (implying it is a module generated or modified by the NDML precompiler), search the CDM table CDMP\_GENERATED\_MOD to retrieve all the modules generated for MOD ID. For each generated module retrieved, delete the Internal Schema software cross reference and the generated module itself.
    - 3.1 Delete from CDM tables RECORD SET\_USAGE, DATA\_FIELD\_USAGE, SOFTWARE\_MODULE, and NDML\_MODULE.
    - 3.2 If MOD ID is not a User's Application program (if FDL-USED = ZERO), then this module does not have any associated External Schema cross references, nor would it be associated with any NDML generated programs.
    - 3.3 Otherwise, delete from CDM tables ES\_USAGE and CDMP\_GENERATED\_MOD.
    - 3.4 Delete the software module MOD ID from MODULE\_PARAMETER and SOFTWARE\_MODULE as in step 2.
- 3.2.8.72 DROP PARTITION - Drop a horizontal partition from the CDM.
- A. Function:

DROP PARTITION allows the NDDL User to drop a horizontal partition for a specific entity from the CDM.
  - B. CDM Requirements:

The horizontal partition for an entity must currently exist.
  - C. Processing:
    - 1. If a partition number is not specified by the user, a "1" is assumed.

2. The user entered entity name is verified to exist and the entity number is retrieved.
3. Verify that the horizontal partition number currently exists for the entity.
4. All partition records are deleted that have the entered entity and horizontal partition number.

3.2.8.73 DROP PSB - Drop an IMS Program Specification Block (PSB) name(s) from the CDM.

A. Function:

1. DROP PSB allows user to drop a PSB name from the CDM.

B. CDM Requirements:

1. The PSB name must be previously defined in the CDM.

C. Processing:

1. The CDM is checked to see if the PSB name to be dropped exists.
2. If the PSB name exists:
  - 2.1 The CDM is checked for any PCBs associated with the PSB name.
  - 2.2 If any PCBs are associated with the PSB name:
    - 2.2.1 The user is notified; issue an error message.
    - 2.2.2 Continue processing with the next PSB name.
  - 2.3 If no PCBs are associated with the PSB name, the PSB name is deleted from the CDM.
3. If the PSB name does not exist, issue an error message and continue command processing.

3.2.8.74 DROP RECORD - Delete the record type/table/segment from the Internal Schema database.

A. Function:

DROP RECORD performs the following functions:



1. Deletes all references to the record type/table/segment from the Internal Schema portion of the CDM.
2. Deletes all associated data fields, record to area assignments, data field linkage, record sets and record set members. Additionally, all textual descriptions for the record type/segment/data field, and record sets are deleted.

B. CDM Requirements:

The record/table/segment to be dropped, and the database must exist in the CDM database.

C. Processing:

1. DROP RECORD verifies the existence of the database/PCB specified and the record type/table/segment specified. If the database or record type does not exist, processing stops and an error message is issued.
2. The record is not deleted if any mappings to the Conceptual Schema are found. The following tables are checked to see if the record is mapped:
  - a) AUC\_IS\_MAPPING - Mapping exists to an Attribute Use Class.
  - b) ECRTUD - If the record is unioned against two or more entities.
  - c) HORIZONTAL PART - If the record is one of the specified fragments in a partition.
3. If the record is an owner record or member record in a non-relational DBMS, the record is not deleted if a set is mapped to. The following tables are checked to see if the record set is mapped:
  - a) AUC\_ST\_MAPPING - If a mapping exists for an Attribute Use Class.
  - b) RC\_BASED\_REC\_SET - If a relation class to set mapping exists.
4. The process then queries for and deletes all database area assignments associated with the record/table/segment. All data fields that belong to the record are deleted along with their associated textual description.

5. For a TOTAL DBMS, DF\_SET\_LINKAGE table removes all references to the record.
6. The process deletes all RECORD\_SET or RECORD\_SET members that contain the record/table/segment to be dropped. The RECORD\_SET processing will delete all references to the record/table/segment where it is an owner record or member record and any set that becomes memberless when the dropped record was the set member.
7. The record/table/segment is deleted from the CDM; all associated textual description about the record are deleted.

3.2.8.75 DROP RELATION - Deletes the link relation and all references to the relation class from the CDM database.

A. Function:

The DROP RELATION process performs the following functions for one or more relations:

1. Verifies that the link relation exists
2. Verifies that the independent and dependent entities exist
3. Deletes the link relation from the CDM
4. Deletes the relation from the CDM
5. Deletes the complete relation
6. Deletes all keys that have migrated from the relation
7. Deletes all keywords associated with the relation
8. Deletes all textual descriptions associated with the relation

B. CDM Requirements:

The DROP RELATION process requires the presence of the Relation Class, Independent Entity, and Dependent Entity within the current model.

C. Processing:

1. The DROP RELATION process verifies that the independent entity, dependent entity, and the

relation exist in the CDM. If any of these do not exist, an error is issued and processing terminates.

2. The process verifies whether the relation is complete and if so, returns the key which allows the process to determine the migration of the associated items.
3. Utilizing the key the process deletes all migrating key member based on the relation. In turn, each key member is deleted from KEY\_CLASS\_MEMBER, ATTRIBUTE\_USE\_CL, and INHERITED\_ATTRIBUTE\_USE based on its association to the relation.
4. After all the key members have been deleted, the process deletes the association in the complete relation, the relation itself, tag constraints, any keywords associated with the relation, and all textual descriptions of the relation.
5. If any mappings exist anywhere, an error is issued, processing is terminated, and no deletions are performed.

#### 3.2.8.76 DROP SET - Drop a record set from the Internal Schema

##### A. Function:

DROP SET allows the NDDL user to drop the set specified from the Internal Schema portion of the CDM.

##### B. CDM Requirements:

The set to be dropped must have been already defined to the CDM.

##### C. Processing:

1. The user entered database name is used to determine if the database exists in the CDM at this point. The set name is used along with the database number to determine if the set actually exists.
  - 1.1 If the set exists, verify that the set is not mapped to the Conceptual Schema. Verify the mapping from the following tables.
    - 1.1.1 RC\_BASED\_REC\_SET, CS to IS relation class to set mappings.
    - 1.1.2 AUC\_ST\_MAPPING, CS to IS attribute to set mappings.
  - 1.2 If no mapping exists, delete the set from the following tables.

- 1.2.1 SET\_TYPE\_MEMBER, all member record types for the set.
- 1.2.2 RECORD\_SET, the record set occurrence itself.
- 1.2.3 For TOTAL DBMS only, the DF\_SET\_LINKAGE occurrence used to indicate the presence of foreign control keys needed for TOTAL link path traversal.

3.2.8.77 DROP UNION - Drop the record union definition from the CDM.

A. Function:

DROP UNION allows the NDDL user to drop the definition of a record union from the CDM.

B. CDM Requirements:

The record union must currently exist in the Integrated Model.

C. Processing:

- 1. Verify that the database and record name currently exist. Retrieve the record type number (RT\_NO) from this selection.
- 2. Verify that this record type currently exists in the record union definitions.
- 3. Delete all record unions which match the record type entered.

3.2.8.78 DROP VIEW - Delete NDDL User specified view(s) from the CDM.

A. Function:

DROP VIEW allows the NDDL user to drop specified views from the CDM. All data items and related textual descriptions are deleted unless the data item is an input or output parameter of a Complex Mapping Algorithm. Also, any textual descriptions associated with the View Name are dropped.

B. CDM Requirements:

The view(s) to be dropped must exist in the external schema portion of the CDM.

C. Processing:

1. A View Name is obtained from a View Name Parser list.
2. Process each View Name obtained from the View name List as follows:
  - 2.1 Verify that the View Name exists in the CDM.
    - 2.1.1 If the View Name does not exist, issue a user error message and continue processing with the next View Name on the Parser List.
  - 2.2 Delete all project data items associated with the View Name from the CDM table PROJECT\_DATA\_ITEM.
  - 2.3 Select all the data items for the View Name from the CDM table DATA\_ITEM.
    - 2.3.1 Verify that each data item does not participate in a conversion algorithm. If a Complex Mapping Algorithm does exist, issue a user error message and continue processing with the next View Name on the Parser List.
    - 2.3.2 Delete any textual descriptions associated with the data item from CDM table DESC\_TEXT.
  - 2.4 Delete all data items associated with the View from the CDM table DATA\_ITEM.
    - 2.4.1 If any data item cannot be deleted, issue an NDML error message and continue processing with the next View Name on the Parser List.
  - 2.5 Delete all occurrences of VIEW\_QUAL\_XREF (if any exist) which are associated with the view.
    - 2.5.1 If any occurrences cannot be deleted, issue an NDML error message and continue processing with the next View Name on the Parser List.
  - 2.6 Delete the occurrence or occurrences of VIEW\_EC\_XREF which are associated with the view.

2.6.1 If any of the occurrences cannot be deleted, issue an NDML error message and continue processing with the next View Name on the Parser List.

2.7 Delete any qualification criteria text items associated with the View from CDM table VIEW\_QUALIFY\_CRITERIA.

2.7.1 If any occurrences cannot be deleted, issue an NDML error message and continue processing with the next View Name on the Parser List.

2.8 Delete the View Name from the CDM table USER\_VIEW.

2.8.1 If the View Name cannot be deleted, issue an NDML error message and continue processing with the next View Name on the Parser List.

2.9 Delete any textual descriptions associated with the View Name from the CDM.

2.9.1 If the textual description cannot be deleted, issue an NDML error message and continue processing with the next View Name on the parser list.

3.2.8.79 HALT - Terminate the current NDDL session

A. Function:

HALT terminates the current NDDL session.

B. Processing:

1. If the commit mode is Automatic:

1.1 If any errors were detected during the NDDL session, an ORACLE rollback is performed. If no errors were detected, an ORACLE commit is performed.

2. If the commit mode is Manual:

2.1 Retrieve from the parser lists the optional "ROLLBACK" clause.

If rollback is specified:

2.1.1 Perform an Oracle Rollback

Else

2.1.2 Perform an Oracle Commit

3.2.8.80 MERGE MODEL - Merge two conceptual models.

A. Function:

MERGE MODEL performs the following functions:

1. Merge two models into the first named model or into a newly created third model.
2. Generate NDDL commands on a file or screen to populate either the first model or the third model with the attributes, entities, relations, key, key members, aliases, keywords and descriptions from model one and model two.

B. CDM Requirements:

1. Model one must exist in the CDM.
2. Model two must exist in the CDM.
3. If model three is specified, it must not exist in the CDM.

C. Processing:

1. Verify the existence of model one and model two. Note that processing halts if any verification checks fail.
  - 1.1 If model three was not specified, default model three to model one; otherwise, verify that model three does not exist.
  - 1.2 If model three does not exist, copy everything from model one to model three using the COPY MODEL routines.
2. Build the key list for all the entities in model two.
3. Add all the model two top node entities to model three. If the model two entity does not exist in model one, then COPY ENTITY routines are used to add the entity to model three. Otherwise, COMBINE ENTITY routines are used to combine the model two entity

with the model one entity with the result added to model three.

4. Add all the model two dependent entities, attributes, link relations, keys, key members, aliases keywords, category relations, and descriptions to model three, level by level. If the model two entity, attribute and/or relation does not exist in model one, then COPY ENTITY routines are used to add the information to model three. Otherwise, COMBINE ENTITY routines are used to combine the model two information with model one with the result added to model three.

3.2.8.81 RENAME - Rename object names for a particular object type.

A. Function:

RENAME performs the following function:

1. Change an old object name to a new object name for object types - entity, attribute, keyword, model, domain, view, relation, host, data item, data field, database, record type, set, and data type name.

B. CDM Requirements:

1. Object names to be renamed must exist in the CDM.
2. Except for keywords, new object names must not previously exist in the CDM.

C. Processing:

1. RENAME verifies the existence of the old object name. If it does not exist, flag a user error.
2. To rename a link relation, the independent entity, relation name, and dependent entities existence is verified. If it does not exist, flag a user error.
3. To rename a category relation, the generic entity and relation name existence is verified. If it does not exist, flag a user error.
4. Verify that the new object name(s) does not previously exist for the particular object type.
  - 4.1 If the object type is KEYWORD and the new name already exists, replace all uses of the old keyword number to the new keyword number in the three keyword tables - AC\_KEYWORD, EC\_KEYWORD, and RC\_KEYWORD and delete the old keyword from the keyword table.



- 4.2 If the object type is not KEYWORD and the new name does not already exist, flag a user error.
5. If all verifications have passed, the old object name is updated in the CDM with the new object name.

3.2.8.82 ROLLBACK - Rollback or undo all changes made to the CDM since the last commit point.

A. Function:

Perform ORACLE and NDML rollback.

B. Processing:

1. Call a routine to perform the NDML rollback.
2. Issue an ORACLE rollback.

3.2.8.83 SET COMMIT - Establish the commit mode.

A. Function:

SET COMMIT establishes the current commit mode for the session - Automatic or Manual.

B. Processing:

1. Retrieve from the parser lists whether the commit mode is 'Automatic' or 'Manual'.
2. Update global variables to reflect the current commit mode for the NDDL session.

3.2.8.84 SET OUTPUT - Establish the output mode.

A. Function:

SET OUTPUT establishes the current output for the NDDL session - whether the output is to be directed to a user defined file or to the screen.

B. Processing:

1. Retrieve from the parser lists whether the output is to be generated to a file or the screen.
2. If output is to be generated to a file, retrieve from the parser lists whether the optional word

"NEW" has been specified. If "NEW" has been specified, set a flag to denote the file named has to be opened and written to for each NDDL command in the session. If "NEW" has not been specified, set a flag to denote that the file named can be opened and appended to.

3. If the output is to be directed to the screen, set a flag to denote that the current mode is screen.
4. Establish the file name and the flag as global variables.

### 3.3 Performance Requirements

#### 3.3.1 Programming Methods

Structured Design, structured code walkthroughs and structured programming will be used wherever possible. Debugging through use of a symbolic debugger will also be used.

#### 3.3.2 Program Organization

NDDL processor will be organized as a single executable image. It will use the forms processor directly. The DBMS access will be done through ORACLE services communicating with the actual DBMS processes. The NDML will be used for database access and it is designed to communicate via the IISS NTM to the actual request processors. The NDDL processor will consist of a main routine, an initialization routine to establish all the environments, a command initialization routine to communicate with the parser command processor interface data structure, a command processor entry point for each command and a termination routine.

#### 3.3.3 Modification Consideration

It would be useful to investigate the creation of a separate process for each command processor. Each would have its own processor. UI function screen or menu interface would allow command selection. A forms driven, rather than syntax driven, approach could also be considered. Continued evolution of NDML facilities should be monitored, such as generation of "in-line" code, replacement of ORACLE with NDML update facilities and removal of many calls to routines that provide integrity tests, currently coded as separate NDML verification routines, since NDML would generate these. Security considerations for a group of different user types must also be considered. Facilities for displaying the CDM contents must also be considered, specifically generating the NDDL that originally populated the object. The NDDL processor must continually be updated as new features and data tables are added to the CDM.

#### 3.3.4 Special Features

#### 3.3.5 Expandability

The NDDL can be expanded very simply in the area of new commands. Parsing directives must be written and new command processors designed and implemented without effecting the command processing shell or existing commands.

#### 3.4 Human Performance

The NDDL processor should allow the CDMA to reliably maintain the most important parts of the CDM and to effectively perform the function of data administration.

#### 3.5 Database Requirements

##### 3.5.1 Database Overview

The CDM database is relational, meaning that it consists of tables that resemble traditional sequential files. The rows of the tables are similar to records in a file and the columns are similar to fields on the records. Columns from different tables are sometimes combined to form "views" to ease manipulation of the data. The CDM database was derived from the definitions found in the CDM-1 model, Reference Number 11.

##### 3.5.2 Relations Between Tables and Views

A single complete view has been created for each table of the CDM accessed by NDDL except in the case of DATA\_FIELD and COMPLEX\_MAPPING\_PARM.

##### 3.5.3 Detailed Description of Tables and Views

There follows an ORACLE listing of the tables columns of the CDM. By using the table names, definitions of each may be found in the CDM-1 model.

## SECTION 4

### QUALITY ASSURANCE PROVISIONS

#### 4.1 Introduction and Definitions

"Testing" is a systematic process that may be preplanned and explicitly stated. Test techniques and procedures may be defined in advance and a sequence of test steps may be specified. "Debugging" is the process of isolation and correction of the cause of an error.

#### 4.2 Computer Programming Test and Evaluation

The quality assurance provisions for tests will consist of the normal testing techniques that are accomplished during the construction process. They consist of design and code walk-throughs, unit testing, and integration testing. These tests will be performed by the design team.

The integration test developed for the NDDL will consist of a list of commands (and their expected outputs) which will be used by the tester. This session will test each command to ensure its correct operation. Results of the session may be compared with those of the unit testing.

Because rather flat hierarchy of modules is designed for the NDDL, unit testing will primarily involve testing each of the NDDL interface routines and internal functions for correct processing and output. Below the level of modules implementing each command will be a small set of procedures for database commit and rollback and error handling.

SECTION 5

PREPARATION FOR DELIVERY

The implementation site for the constructed software will be the ICAM Integrated Support System (IISS) Test Bed Site located at Arizona State University, Tempe, Arizona. The software associated with the NDDL will be clearly identified and will include instructions on procedures to be followed for installation of the release.

DS 620341100  
30 September 1990

SECTION 6

NOTES

Please refer to the Software Availability Bulletin, Volume III, Part 16, CI# SAB620326000, for current IISS software and documentation availability.

APPENDIX A

ORACLE DATA DICTIONARY

TABLE NAME	COLUMN NAME	TYPE	WIDTH	NULLS
-----	-----	-----	-----	-----
AC_KEYWORD	AC_NO	NUMBER	6	NOT NULL
	KW_NO	NUMBER	6	NOT NULL
ATTRIBUTE_CLASS	AC_NO	NUMBER	6	NOT NULL
DOMAIN_NO	NUMBER 6	NOT NULL		
MODEL_NO	NUMBER 6	NOT NULL		
ATTRIBUTE_NAME	AC_NAME	CHAR	30	NOT NULL
	AC_NAME_TYPE	CHAR	8	NOT NULL
	AC_NO	NUMBER	6	NOT NULL
ATTRIBUTE_USE_CL	AC_NO	NUMBER	6	NOT NULL
	EC_NO	NUMBER	6	NOT NULL
	TAG_NAME	CHAR	30	NOT NULL
	TAG_NO	NUMBER	6	NOT NULL
AUC_CONSTRAINT	CONSTRAINT_NO	NUMBER	6	NOT NULL
	EC_NO	NUMBER	6	NULL
	TAG_NO	CHAR	30	NOT NULL
	TAG_NO	NUMBER	6	NOT NULL
AUC_IS_MAPPING	EC_NO	NUMBER	6	NOT NULL
	MAP_CATEGORY	CHAR	8	NULL
	MAP_CLASS	CHAR	30	NULL
	MAP_TYPE	CHAR	10	NULL
	PREF_NO	NUMBER	2	NULL
	RT_NO	NUMBER	6	NOT NULL
	TAG_NO	NUMBER	6	NOT NULL
AUC_ST_MAPPING	AUC_VALUE	CHAR	30	NULL
	DB_ID	NUMBER	6	NOT NULL
	EC_NO	NUMBER	6	NULL
	RT_NO	NUMBER	6	NULL
	SET_ID	CHAR	30	NOT NULL
	TAG_NO	NUMBER	6	NOT NULL
CDMP_GENERATED_MOD	CASE_NO	NUMBER	6	NULL
	DB_ID	NUMBER	6	NULL
	FILE_NAME	CHAR	30	NULL
	GENERATED_BY	CHAR	30	NULL
	GENERATED_MOD_ID	CHAR	10	NOT NULL

TABLE NAME	COLUMN NAME	TYPE	WIDTH	NULLS
CDMP_GENERATED_MOD	GEN_DATE	DATE	7	NULL
	HOST_ID	CHAR	30	NULL
	IS_ACTION	CHAR	1	NULL
	LOCAL_REMOTE	CHAR	1	NULL
	LUW_NAME	CHAR	30	NULL
	MODULE_TYPE	CHAR	10	NOT NULL
	SUBTRANS_ID	NUMBER	6	NULL
	USER_MOD_ID	CHAR	1	NULL
COMPLETE_RELATION	KC_NO	NUMBER	6	NOT NULL
	RC_NO	NUMBER	6	NOT NULL
COMPLEX_MAPPING_PARM	ALG_USE_CODE	CHAR	1	NULL
	CONSTANT_VALUE	CHAR	30	NULL
	DF_NO	NUMBER	6	NULL
	DI_NO	NUMBER	6	NULL
	MOD_ID	CHAR	10	NOT NULL
	MOD_INST	NUMBER	6	NULL
	PARM_ID	NUMBER	6	NULL
	RT_NO	NUMBER	6	NULL
	TAG_NO	NUMBER	6	NULL
	UNION_DISC	NUMBER	6	NULL
CONSTRAINT_INPUT	CONSTRAINT_NO	NUMBER	6	NOT NULL
	MOD_ID	CHAR	10	NOT NULL
	PARM_ID	NUMBER	2	NOT NULL
	STMT_ACTION	CHAR	30	NOT NULL
	TAG_NO	NUMBER	6	NOT NULL
DATA_BASE	CHARACTER_NULL	CHAR	30	NULL
	DBMS_NAME	CHAR	30	NULL
	DB_ID	NUMBER	6	NULL
	DB_NAME	CHAR	30	NULL
	HOST_ID	CHAR	30	NULL
	INTEGER_NULL	CHAR	30	NULL
	NTM_DIRECTORY	CHAR	2	NULL
DATA_BASE_AREA	AREA_ID	CHAR	30	NOT NULL
	DB_ID	NUMBER	6	NOT NULL
DATA_FIELD	COMPONENT_OF_DF	NUMBER	6	NULL
DATA_TYPE_NAME	CHAR 30	NULL		



TABLE NAME	COLUMN NAME	TYPE	WIDTH	NULLS
DATA_FIELD	DBMS_ACCESS	CHAR	1	NULL
	DB_ID	NUMBER	6	NOT NULL
	DF_ID	CHAR	30	NOT NULL
	DF_NO	NUMBER	6	NOT NULL
	FILLER_SIZE	NUMBER	6	NULL
	INDEX_BY_DF_NO	NUMBER	6	NULL
	INDEX_INDICATOR	CHAR	1	NULL
	NO_OF_OCCURS	NUMBER	6	NULL
	OCC_DEPEND_DF_NO	NUMBER	6	NULL
	REC_KEY_CODE	CHAR	1	NULL
	REC_SEQ_NO	NUMBER	6	NOT NULL
	REDEF_DF_NO	NUMBER	6	NULL
	RT_ID	CHAR	30	NOT NULL
DATA_FIELD_USAGE	DF_NO	NUMBER	6	NOT NULL
	DF_USAGE_CODE	CHAR	1	NULL
	MOD_ID	CHAR	10	NOT NULL
DATA_ITEM	DATA_TYPE_NAME	CHAR	30	NOT NULL
	DI_ID	CHAR	30	NOT NULL
	DI_NO	NUMBER	6	NULL
	VIEW_NO	NUMBER	6	NOT NULL
DATA_TYPE	TYPE_DESC	CHAR	60	NOT NULL
	TYPE_ID	CHAR	1	NOT NULL
DBMS_ON_HOST	DBMS_NAME	CHAR	30	NOT NULL
	HOST_ID	CHAR	30	NOT NULL
DB_AREA_ASSIGNMENT	AREA_ID	CHAR	30	NOT NULL
	DB_ID	NUMBER	6	NOT NULL
	RT_ID	CHAR	30	NOT NULL
DB_PASSWORD	DB_ID	NUMBER	6	NOT NULL
	DB_PASSWORD	CHAR	30	NOT NULL
DESCRIPTION_TYPE	DESC_TYPE	CHAR	30	NOT NULL
DESC_TEXT	DESC_TEXT	CHAR	79	NULL
	DESC_TYPE	CHAR	30	NOT NULL
DESC_TEXT	LINE_NO	NUMBER	6	NOT NULL
	OBJECT_NO	NUMBER	6	NOT NULL
	OBJECT_TYPE	CHAR	30	NOT NULL

DS 620341100  
30 September 1990

<u>TABLE NAME</u>	<u>COLUMN NAME</u>	<u>TYPE</u>	<u>WIDTH</u>	<u>NULLS</u>
DF_SET_LINKAGE	DB_ID	NUMBER	6	NOT NULL
	DF_ID	CHAR	30	NOT NULL
	LINKAGE_TYPE	CHAR	1	NOT NULL
	RT_ID	CHAR	30	NOT NULL
	SET_ID	CHAR	30	NOT NULL
DISTRIBUTED_RULES	DISTR_RETR_RULE	CHAR	8	NULL
	DISTR_UPDT_RULE	CHAR	8	NULL
	EC_NO	NUMBER	6	NOT NULL
DOMAIN_CLASS	DOMAIN_NAME	CHAR	30	NULL
	DOMAIN_NO	NUMBER	6	NULL
DOMAIN_RANGE	BEGIN_VALUE	CHAR	30	NULL
	DOMAIN_NO	NUMBER	6	NOT NULL
	END_VALUE	CHAR	30	NULL
DOMAIN_VALUE	DOMAIN_NO	NUMBER	6	NOT NULL
	SPECIFIC_VALUE	CHAR	30	NULL
ECRTUD	COMPARISON_OP	CHAR	2	NULL
	DF_NO	NUMBER	6	NOT NULL
	EC_NO	NUMBER	6	NOT NULL
	RT_NO	NUMBER	6	NOT NULL
	UNION_VALUE	CHAR	30	NOT NULL
EC_CONSTRAINT	CONSTRAINT_NO	NUMBER	6	NULL
	EC_NO	NUMBER	6	NULL
	STMT_ACTION	CHAR	30	NULL
EC_KEYWORD	EC_NO	NUMBER	6	NOT NULL
	KW_NO	NUMBER	6	NOT NULL
EC_RT_MAPPING	EC_NO	NUMBER	6	NOT NULL
	RT_NO	NUMBER	6	NOT NULL
ENTITY_CLASS	EC_NO	NUMBER	6	NOT NULL
	MODEL_NO	NUMBER	6	NOT NULL
ENTITY_NAME	EC_NAME	CHAR	30	NOT NULL
	EC_NAME_TYPE	CHAR	8	NOT NULL
	EC_NO	NUMBER	6	NOT NULL
ES_USAGE	DI_NO	NUMBER	6	NULL
	DI_USAGE_CODE	CHAR	1	NULL
	MOD_ID	CHAR	10	NULL
	VIEW_NO	NUMBER	6	NULL

<u>TABLE NAME</u>	<u>COLUMN NAME</u>	<u>TYPE</u>	<u>WIDTH</u>	<u>NULLS</u>
FILE_NAME_HOST	HOST_ID	CHAR	30	NOT NULL
	LAST_FILE_USED	CHAR	30	NOT NULL
HORIZONTAL_PART	EC_NO	NUMBER	6	NULL
	HP_NO	NUMBER	6	NULL
	RT_NO	NUMBER	6	NULL
IISS_DBMS	DBMS_NAME	CHAR	30	NOT NULL
	DB_MODEL	CHAR	1	NOT NULL
IISS_HOST	HOST_ID	CHAR	30	NOT NULL
	HOST_NO	NUMBER	6	NULL
IISS_KEYWORD	CDM_KEYWORD	CHAR	30	NOT NULL
	KW_NO	NUMBER	6	NOT NULL
IISS_PSB	HOST_ID	CHAR	30	NOT NULL
	PSB_NAME	CHAR	8	NOT NULL
INHERITED_ATT_USE	KCM_TAG_NO	NUMBER	6	NOT NULL
	KC_NO	NUMBER	6	NOT NULL
	RC_NO	NUMBER	6	NOT NULL
	TAG_NO	NUMBER	6	NOT NULL
KEY_CLASS	EC_NO	NUMBER	6	NOT NULL
	KC_NAME	CHAR	30	NOT NULL
	KC_NO	NUMBER	6	NOT NULL
KEY_CLASS_MEMBER	KC_NO	NUMBER	6	NOT NULL
	TAG_NO	NUMBER	6	NOT NULL
LOG_UNIT_WORK	LAST_CASE_NO	NUMBER	6	NULL
	LUW_NAME	CHAR	30	NULL
MACRO_CODE	LIBRARY_NAME	CHAR	30	NULL
	MACRO_CODE	CHAR	72	NULL
	MACRO_LINE_NO	NUMBER	6	NULL
	MACRO_NAME	CHAR	8	NULL
MODEL_CLASS	DATE_CREATED	DATE	7	NOT NULL
	DATE_MODIFIED	DATE	7	NOT NULL
	MODEL_NAME	CHAR	30	NOT NULL
	MODEL_NO	NUMBER	6	NOT NULL
	MODEL_STATUS	CHAR	10	NOT NULL

DS 620341100  
30 September 1990

<u>TABLE NAME</u>	<u>COLUMN NAME</u>	<u>TYPE</u>	<u>WIDTH</u>	<u>NULLS</u>
MODULE_PARAMETER	DATA_TYPE_NAME	CHAR	30	NULL
	MOD_ID	CHAR	10	NOT NULL
	PARM_ID	NUMBER	6	NULL
	PARM_NAME	CHAR	30	NULL
	PARM_PURPOSE	CHAR	60	NULL
NDML_MODULE	LAST_COMP_STAT	CHAR	1	NULL
	LUW_NAME	CHAR	30	NULL
	MOD_ID	CHAR	10	NULL
	PRECOMP_DATE	CHAR	9	NULL
NEXT_MOD_NAME	LAST_NAME_USED	CHAR	10	NOT NULL
NEXT_NUMBER	AC_NO	NUMBER	6	NOT NULL
	NEXT_NO	NUMBER	6	NOT NULL
	OBJECT_NAME	CHAR	30	NULL
OWNED_ATTRIBUTE	AC_NO	NUMBER	6	NOT NULL
OWNED_ATTRIBUTE	EC_NO	NUMBER	6	NOT NULL
PROJECT_DATA_FIELD	DB_ID	NUMBER	6	NOT NULL
	DF_ID	CHAR	30	NOT NULL
	EC_NO	NUMBER	6	NULL
	RT_ID	CHAR	30	NOT NULL
	RT_NO	NUMBER	6	NULL
	TAG_NO	NUMBER	6	NOT NULL
PROJECT_DATA_ITEM	DI_NO	NUMBER	6	NOT NULL
	EC_NO	NUMBER	6	NULL
	PRIM_SECONDARY	CHAR	1	NULL
	TAG_NO	NUMBER	6	NULL
	VIEW_NO	NUMBER	6	NULL
PSB_PCB	DB_ID	NUMBER	6	NOT NULL
	KEY_FEEDBACK_LEN	NUMBER	6	NOT NULL
	PCB_SEQ_NO	NUMBER	6	NOT NULL
	PSB_NAME	CHAR	8	NOT NULL
PSB_USAGE	MOD_ID	CHAR	10	NOT NULL
	PSB_NAME	CHAR	8	NULL
RC_BASED_REC_SET	DB_ID	NUMBER	6	NOT NULL
	RC_NO	NUMBER	6	NOT NULL
	RT_ID	CHAR	30	NOT NULL
	SET_ID	CHAR	30	NOT NULL

<u>TABLE NAME</u>	<u>COLUMN NAME</u>	<u>TYPE</u>	<u>WIDTH</u>	<u>NULLS</u>
RC_KEYWORD	KW_NO	NUMBER	6	NOT NULL
	RC_NO	NUMBER	6	NOT NULL
RECORD_SET	DB_ID	NUMBER	6	NOT NULL
	RT_ID OF_OWNER	CHAR	30	NOT NULL
	SET_ID	CHAR	30	NOT NULL
	SET_NO	NUMBER	6	NULL
	TOTAL_NUM_MEM	NUMBER	6	NOT NULL
RECORD_SET_USAGE	MOD_ID	CHAR	10	NULL
	SET_NO	NUMBER	6	NULL
RECORD_TYPE	DB_ID	NUMBER	6	NOT NULL
	RT_ID	CHAR	30	NOT NULL
	RT_NO	NUMBER	6	NULL
RELATION_CLASS	DEP_EC_NO	NUMBER	6	NOT NULL
	IND_EC_NO	NUMBER	6	NOT NULL
	MAX_NO_DEP_ENT	NUMBER	6	NOT NULL
	MIN_NO_DEP_ENT	NUMBER	6	NOT NULL
	NO_IND_ENT	NUMBER	6	NOT NULL
	RC_NAME	CHAR	30	NOT NULL
	RC_NO	NUMBER	6	NOT NULL
SCHEMA_NAMES	DB_ID	NUMBER	6	NOT NULL
	DB_LOCATION	CHAR	30	NULL
	SCHEMA_NAME	CHAR	30	NOT NULL
	SUBSCHEMA_NAME	CHAR	30	NOT NULL
SET_TYPE_MEMBER	DB_ID	NUMBER	6	NOT NULL
	REQ MEM IND	CHAR	1	NOT NULL
	RT ID OF_			
	MEMBER	CHAR	30	NOT NULL
	SET_ID	CHAR	30	NOT NULL
SOFTWARE_MODULE	LANG NAME	CHAR	10	NULL
	LATEST_REV_			
	DATE	CHAR	9	NULL
	MOD ABSTRACT	CHAR	60	NULL
	MOD_ID	CHAR	10	NOT NULL
	MOD TITLE	CHAR	30	NULL
USER_DEF_DATA_TYPE	STATUS_IND	CHAR	1	NULL
	DATA_TYPE_IND	CHAR	4	NULL
	DATA_TYPE_NAME	CHAR	30	NULL
	DOMAIN_NO	NUMBER	6	NULL
	MAX SIZE	NUMBER	6	NULL
	NO OF DECIMALS	NUMBER	6	NULL
	TYPE_ID	CHAR	1	NULL
	USDF_DT_NO	NUMBER	6	NULL

DS 620341100  
30 September 1990

<u>TABLE NAME</u>	<u>COLUMN NAME</u>	<u>TYPE</u>	<u>WIDTH</u>	<u>NULLS</u>
USER_VIEW	DISTINCT_IND	CHAR	1	NOT NULL
	VIEW_ID	CHAR	30	NOT NULL
	VIEW_NO	NUMBER	6	NOT NULL
VERIF_MODULE	DOMAIN_NO	NUMBER	6	NULL
	MOD_ID	CHAR	10	NULL
VIEW_EC_XREF	EC_NO	NUMBER	6	NOT NULL
	VIEW_NO	NUMBER	6	NOT NULL
VIEW_QUALIFY_CRITERIA	QC_CONDITION_NO	NUMBER	6	NULL
	QC_COND_TYPE	NUMBER	1	NULL
	QC_TEXT	CHAR	30	NULL
	QC_TEXT_NO	NUMBER	6	NOT NULL
	QC_TEXT_TYPE	CHAR	1	NULL
	VIEW_NO	NUMBER	6	NOT NULL
VIEW_QUAL_XREF	EC_NO	NUMBER	6	NOT NULL
	TAG_NO	NUMBER	6	NULL
	VIEW_NO	NUMBER	6	NOT NULL

266 records selected.

SQL> SPOOL OFF

APPENDIX B

CDM TABLES ACCESSED

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
ALTER ALIAS	ATTRIBUTE_CLASS	SELECT	VERNMA	N
	ATTRIBUTE_NAME	SELECT	VERNMA	N
	ATTRIBUTE_NAME	MODIFY	UPDACAL	N
	ENTITY_CLASS	SELECT	VERNME	N
	ENTITY_NAME	SELECT	VERNME	N
	ENTITY_NAME	MODIFY	UPDECAL	N
ALTER ATTRIBUTE	AC_KEYWORD	SELECT	ADDKWA	N
	AC_KEYWORD	DELETE	DELKWAC	S
	AC_KEYWORD	INSERT	INSKWAC	S
	ATTRIBUTE_CLASS	SELECT	VERATT	N
	ATTRIBUTE_CLASS	MODIFY	UPDAC	S
	ATTRIBUTE_NAME	SELECT	RETATTR	N
	ATTRIBUTE_NAME	SELECT	VERATT	N
	ATTRIBUTE_USE_CL	INSERT	INSAUC	S
	ATTRIBUTE_USE_CL	MODIFY	MODAUCE	S
	ATTRIBUTE_USE_CL	SELECT	RETATTR	N
	ATTRIBUTE_USE_			
	CLASS	SELECT	VERATAG	S
	COMPLETE_RELATION	INSERT	INSCRC	S
	COMPLETE_RELATION	SELECT	NEWTAGM	S
	COMPLETE_RELATION	SELECT	VERIREL	S
	DOMAIN_CLASS	SELECT	VERDOM	N
	ENTITY_CLASS	SELECT	DELMTK	S
	ENTITY_CLASS	SELECT	VERENT	N
	ENTITY_NAME	SELECT	VERENT	N
	IISS_KEYWORD	SELECT	VERKW	N
	IISS_KEYWORD	INSERT	INSKW	S
	INHERITED_ATT_USE	SELECT	DELMIGK	S
	INHERITED_ATT_USE	SELECT	OLDTAGM	N
	INHERITED_ATT_USE	INSERT	INSIAUC	S
	INHERITED_ATT_USE	SELECT	DMIGKRC	S
	INHERITED_ATT_USE	DELETE	DELIAUC	S
	KEY_CLASS	SELECT	DELMTKC	S
	KEY_CLASS	INSERT	INSKEYC	N
	KEY_CLASS	SELECT	VERKC	N
	KEY_CLASS_MEMBER	SELECT	DELMTKC	S
	KEY_CLASS_MEMBER	INSERT	INSKCM	S
	KEY_CLASS_MEMBER	SELECT	VERKCM	N
	KEY_CLASS_MEMBER	DELETE	DELKCMT	S

<u>COMMAND</u>	<u>TABLE NAME</u>	<u>FUNCTI</u>	<u>MODULE</u>	(S=SQL N=NDML)
	OWNED_ATTRIBUTE	MODIFY	MODACEC	N
	OWNED_ATTRIBUTE	SELECT	RETATTR	N
	RELATION_CLASS	SELECT	VERRC	N
	RELATION_CLASS	SELECT	SELRELC	N
	RELATION_CLASS	SELECT	NEWTAGM	S
	RELATION_CLASS	SELECT	OLDTAGM	N
ALTER DATABASE	DATA_BASE	SELECT	VERDBAL	N
	DATA_BASE	MODIFY	MODDBAL	N
	DATA_BASE_AREA	SELECT	VERARL	N
	DATA_BASE_AREA	SELECT	VERARLA	N
	DATA_BASE_AREA	INSERT	INSAREA	N
	DATA_BASE_AREA	DELETE	DELAREA	N
	DB_PASSWORD	MODIFY	MODPWRD	N
	IISS_HOST	SELECT	VERHST	N
	PSB_PCB	MODIFY	MODPCB	N
	SCHEMA_NAMES	MODIFY	MODSCH	N
	SCHEMA_NAMES	MODIFY	MODLOC	N
ALTER DBMS	DBMS_ON_HOST	INSERT	INSDBH	N
	DBMS_ON_HOST	DELETE	DELDBH	N
	DBMS_ON_HOST	SELECT	VERDBH	N
	IISS_DBMS	SELECT	VERDBMS	N
	IISS_DBMS	MODIFY	ALTDDBT	N
	IISS_HOST	SELECT	VERHST	N
ALTER DOMAIN	ATTRIBUTE_CLASS	SELECT	VERACDT	N
	DATA_ITEM	SELECT	VERDIDT	N
	DATA_TYPE	SELECT	VERTYP	N
	DOMAIN_CLASS	SELECT	VERDOM	N
	DOMAIN_RANGE	INSERT	INSRNG	N
	DOMAIN_RANGE	SELECT	RETRNGA	N
	DOMAIN_RANGE	SELECT	RETRNG	N
	DOMAIN_RANGE	DELETE	DRPRNGA	N
	DOMAIN_RANGE	DELETE	DRPRNG	N
	DOMAIN_RANGE	SELECT	CHKRNG	N
	DOMAIN_RANGE	SELECT	VERRNG	N
	DOMAIN_VALUE	SELECT	RETVALA	N
	DOMAIN_VALUE	DELETE	DRPVAL	N
	DOMAIN_VALUE	INSERT	INSVAL	N
	DOMAIN_VALUE	SELECT	CHKVAL	N
	DOMAIN_VALUE	SELECT	VERVAL	N
	ELEMENTARY_DATA_FIELD	SELECT	VERDFDT	N
	MODULE_PARAMETER	INSERT	INSPARM	N
	MODULE_PARAMETER	SELECT	VERMPDT	N



<u>COMMAND</u>	<u>TABLE NAME</u>	<u>FUNCTI</u>	<u>MODULE</u>	(S=SQL N=NDML)
	MODULE_PARAMETER	DELETE	DELPARM	N
	SOFTWARE_MODULE	INSERT	INSSMOD	N
	SOFTWARE_MODULE	DELETE	DELSMOD	N
	USER_DEF_DATA_			
	TYPE	SELECT	RETSTD	N
	USER_DEF_DATA_			
	TYPE	DELETE	DELDLT	N
	USER_DEF_DATA_			
	TYPE	MODIFY	UPDIND	N
	USER_DEF_DATA_			
	TYPE	MODIFY	VERSDT	N
	USER_DEF_DATA_			
	TYPE	MODIFY	VERDTD	N
	USER_DEF_DATA_			
	TYPE	SELECT	RETDT	N
	USER_DEF_DATA_			
	TYPE	SELECT	VERDT	N
	USER_DEF_DATA_			
	TYPE	INSERT	INSDT	N
	USER_DEF_DATA_			
	TYPE	MODIFY	UPDTDT	N
	VERIF_MODULE	SELECT	RETVMOD	N
	VERIF_MODULE	INSERT	INSVMOD	N
	VERIF_MODULE	DELETE	DRPVMOD	N
ALTER ENTITY	ATTRIBUTE_CLASS	SELECT	VERATT	N
	ATTRIBUTE_NAME	SELECT	VERATT	N
	ATTRIBUTE_USE_CL	SELECT	VTAUCL	S
	ATTRIBUTE_USE_CL	SELECT	DELOAC	N
	ATTRIBUTE_USE_CL	SELECT	VERAUC	N
	ATTRIBUTE_USE_CL	DELETE	DELAUCL	S
	ATTRIBUTE_USE_CL	INSERT	INSAUC	S
	ATTRIBUTE_USE_CL	UPDATE	MODAAUC	S
	ATTRIBUTE_USE_CL	SELECT	VERATAG	S
	AUC_CONSTRAINT	DELETE	DELAON	N
	COMPLETE_RELATION	SELECT	KCINH	S
	COMPLETE_RELATION	DELETE	DELCMPR	S
	COMPLETE_RELATION	SELECT	VERCRC	N
	CONSTRAINT_INPUT	DELETE	DELCONI	N
	EC_KEYWORD	SELECT	ADDKWE	N
	EC_KEYWORD	DELETE	DELKWE	S
	ENTITY_CLASS	SELECT	VERENT	N
	ENTITY_NAME	SELECT	VERENT	N
	IISS_KEYWORD	SELECT	VERKW	N
	IISS_KEYWORD	INSERT	INSKW	S
	INHERITED_ATT_USE	SELECT	DMIGKKC	S
	INHERITED_ATT_USE	DELETE	DELIAUC	S

<u>COMMAND</u>	<u>TABLE NAME</u>	<u>FUNCTI</u>	<u>MODULE</u>	(S=SQL N=NDML)
	INHERITED_ATT_USE	SELECT	FETITAG	S
	INHERITED_ATT_USE	INSERT	INSIAUC	S
	INHERITED_ATT_USE	UPDATE	MODIAUC	S
	INHERITED_ATT_USE	UPDATE	MODAKCM	S
	INHERITED_ATT_USE	SELECT	DELMIGK	S
	KEY_CLASS	UPDATE	MODKCNM	S
	KEY_CLASS	SELECT	VERKC	N
	KEY_CLASS	DELETE	DELKC	S
	KEY_CLASS	INSERT	INSKC	S
	KEY_CLASS_MEMBER	SELECT	VERTKCM	S
	KEY_CLASS_MEMBER	DELETE	DELAKCM	S
	KEY_CLASS_MEMBER	SELECT	DRPMGKM	N
	KEY_CLASS_MEMBER	DELETE	DELKCM	S
	KEY_CLASS_MEMBER	INSERT	INSKCM	S
	OWNED_ATTRIBUTE	SELECT	DRPAC	N
	OWNED_ATTRIBUTE	SELECT	VEROAC	N
	OWNED_ATTRIBUTE	INSERT	INSOAC	S
	RELATION_CLASS	SELECT	GETRCID	N
	RELATION_CLASS	SELECT	KCINH	S
	VIEW_QUAL_XREF	SELECT	VERVWQU	N
ALTER FIELD	AUC_ST_MAPPING	SELECT	FNDASA	N
	DATA_BASE	SELECT	VERDBAS	N
	DATA_FIELD	SELECT	RETRFLD	S
	DATA_FIELD	DELETE	DELDFL3	S
	DATA_FIELD	INSERT	INSTFLD	S
	DATA_FIELD	UPDATE	MODYFLD	S
	DATA_FIELD_USAGE	SELECT	DELDBDF	N
	DESC_TEXT	DELETE	DELTEXT	S
	DF_PARM	SELECT	VERDFPA	N
	DF_SET_LINKAGE	SELECT	VERDSL3	N
	DF_SET_LINKAGE	DELETE	DELDSL3	S
	ECRTUD	SELECT	VERMUNI	N
	PROJECT_DATA_			
	FIELD	SELECT	VERMPDF	N
	RC_BASED_REC_SET	SELECT	VERRCBS	N
	RECORD_SET	DELETE	DELIRST3	S
	RECORD_TYPE	SELECT	VERRT	N
	SET_TYPE_MEMBER	DELETE	DELSTM3	S
	USER_DEF_DATA_			
	TYPE	SELECT	VERDTYP	N
ALTER HOST	DBMS_ON_HOST	INSERT	INSDBH	N
	DBMS_ON_HOST	SELECT	VERDBH	N
	DBMS_ON_HOST	DELETE	DELDBH	N
	IISS_DBMS	SELECT	VERDBMS	N
	IISS_HOST	SELECT	VERHST	N

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
ALTER MAP	ATTRIBUTE_CLASS	SELECT	VERTAG	N
	ATTRIBUTE_USE_CL	SELECT	VERTAG	N
	ATTRIBUTE_USE_CL	SELECT	VERAUC	N
	AUC_IS_MAPPING	DELETE	DEL1AIM	N
	AUC_IS_MAPPING	MODIFY	ALTAISM	N
	AUC_IS_MAPPING	SELECT	VERASET	N
	AUC_IS_MAPPING	SELECT	VERAISM	N
	AUC_IS_MAPPING	INSERT	INSAISM	N
	AUC_IS_MAPPING	SELECT	CHKAUCV	N
	AUC_IS_MAPPING	SELECT	VERAIMR	N
	AUC_IS_MAPPING	SELECT	VERAPDF	N
	AUC_IS_MAPPING	SELECT	SELAISM	N
	AUC_IS_MAPPING	SELECT	CHKHPRT	N
	AUC_IS_MAPPING	MODIFY	MODAISM	N
	AUC_IS_MAPPING	SELECT	VOMAPS	N
	AUC_ST_MAPPING	INSERT	INSAUCS	N
	AUC_ST_MAPPING	SELECT	FNDASA	N
	AUC_ST_MAPPING	SELECT	CHKAUCV	N
	AUC_ST_MAPPING	SELECT	FNDASM	N
	AUC_ST_MAPPING	SELECT	VOMAPS	N
	AUC_ST_MAPPING	SELECT	CHKSTMP	N
	AUC_ST_MAPPING	SELECT	VERASET	N
	AUC_ST_MAPPING	DELETE	DEL1ASM	N
	AUC_ST_MAPPING	MODIFY	UPDAUCS	N
	DATA_BASE	SELECT	VERDFID	N
	DATA_BASE	SELECT	VERDTFL	N
	DATA_BASE	SELECT	VERDB	N
	DATA_FIELD	SELECT	VERDFID	N
	DATA_FIELD	SELECT	VERMAPD	N
	DATA_FIELD	SELECT	VERDTFL	N
	DISTRIBUTED_RULES	SELECT	VERRULE	N
	DISTRIBUTED_RULES	MODIFY	UPDRULE	N
	EC_RT_MAPPING	SELECT	ECRTALL	N
	EC_RT_MAPPING	DELETE	ALT1ERT	N
	ELEMENTARY_DATA_			
	FIELD	SELECT	GETDTN	N
	ENTITY_CLASS	SELECT	VERENT	N
	ENTITY_NAME	SELECT	VERTAG	N
	ENTITY_NAME	SELECT	VERENT	N
	HORIZONTAL_PART	SELECT	VERHZP	N
	HORIZONTAL_PART	SELECT	VERHPDF	N
	HORIZONTAL_PART	SELECT	VERHPST	N
	HORIZONTAL_PART	SELECT	CHKHPRT	N
	PROJECT_DATA_			
	FIELD	SELECT	FNDPDF	N
	PROJECT_DATA_			
	FIELD	INSERT	INSPDF	N
	PROJECT_DATA_			
	FIELD	DELETE	DEL1PDF	N
	PROJECT_DATA_			
	FIELD	SELECT	VERAPDF	N
	RC_BASED_REC_SET	SELECT	VERRCBS	N

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
ALTER MAP	RC_BASED_REC_SET	SELECT	VERRCMP	N
	RC_BASED_REC_SET	INSERT	INSRCRS	S
	RC_BASED_REC_SET	DELETE	DELIRCS	S
	RC_BASED_REC_SET	SELECT	FNDRC M	N
	RECORD_SET	SELECT	VERHPST	N
	RECORD_SET	SELECT	VOMAPS	N
	RECORD_SET	SELECT	VERSMS	N
	RECORD_TYPE	SELECT	VERHPDF	N
	RECORD_TYPE	SELECT	VERDTFL	N
	RECORD_TYPE	SELECT	VERHPST	N
	RELATION_CLASS	SELECT	VERRC	N
	SET_TYPE_MEMBER	SELECT	FND1MEM	N
	USER_DEF_DATA_			
	TYPE	SELECT	VERTAG	N
	USER_DEF_DATA_			
	TYPE	SELECT	VERDTYP	N
ALTER MODEL	MODEL_CLASS	SELECT	VERMOD	N
	MODEL_CLASS	MODIFY	UPDMOD	S
ALTER MODULE	MODULE_PARAMETER	SELECT	VERAPRM	N
	MODULE_PARAMETER	DELETE	DELAPRM	N
	MODULE_PARAMETER	SELECT	SELPARM	N
	MODULE_PARAMETER	INSERT	INSPARM	N
	MODULE_PARAMETER	DELETE	DELPARM	N
	SOFTWARE_MODULE	MODIFY	UPDSMOD	N
	SOFTWARE_MODULE	SELECT	VERSMOD	N
	USER_DEF_DATA_			
	TYPE	SELECT	VERDTYP	N
ALTER PARTITION	DATA_BASE	SELECT	VERDBRT	N
	ENTITY_CLASS	SELECT	VERECN	N
	ENTITY_NAME	SELECT	VERECN	N
	HORIZONTAL_PART	SELECT	VERHPN	N
	HORIZONTAL_PART	INSERT	INSPART	N
	HORIZONTAL_PART	DELETE	DELHPN	N
	HORIZONTAL_PART	SELECT	VERPART	N
	MODEL_CLASS	SELECT	VERECN	N

COMMAND	TABLE NAME	FUNCTION	MODULE	(S=SQL N=NDML)
ALTER PSB	DBMS_ON_HOST	SELECT	VERDBH	N
	IISS_HOST	SELECT	VERHST	N
	IISS_PSB	SELECT	VERPSB	N
	IISS_PSB	MODIFY	UPD1PSB	N
ALTER RECORD	AUC_ST_MAPPING	SELECT	FNDASA	N
	DATA_BASE	SELECT	VERDBAS	N
	DATA_BASE_AREA	INSERT	INSAREA	N
	DATA_BASE_AREA	SELECT	VERAREA	N
	DATA_BASE_AREA	INSERT	INSDAA	N
	DATA_FIELD	SELECT	RETRFLD	S
	DATA_FIELD	DELETE	DELDFL3	S
	DATA_FIELD	DELETE	DELFLDA	S
	DATA_FIELD	UPDATE	MODFLD	S
	DATA_FIELD	INSERT	INSFLD	S
	DATA_FIELD_USAGE	SELECT	VERDFUS	N
	DB AREA			
	ASSIGNMENT	DELETE	DELAAA	N
	DB AREA			
	ASSIGNMENT	SELECT	VERDBAA	N
	DESC TEXT	DELETE	DELTEXT	S
	DF_PARM	SELECT	VERDFPA	N
	DF_SET_LINKAGE	SELECT	VERDSL3	N
	DF_SET_LINKAGE	DELETE	DELDSL3	S
	ECRTUD	SELECT	VERMUNI	N
	PROJECT_DATA_			
	FIELD	SELECT	VERMPDF	N
	RC_BASED_REC_SET	SELECT	VERRCBS	N
	RECORD_SET	DELETE	DELIRST3	S
	RECORD_TYPE	SELECT	VERRT	N
	SET_TYPE_MEMBER	DELETE	DELSTM3	S
	USER_DEF_DATA_			
	TYPE	SELECT	VERDTYP	N
ALTER RELATION	ATTRIBUTE_USE_CL	SELECT	ADDMIG	N
	ATTRIBUTE_USE_CL	INSERT	INSAUC	S
	ATTRIBUTE_USE_CL	DELETE	DELAUCL	S
	ATTRIBUTE_USE_CL	SELECT	VERAUC	N
	AUC_CONSTRAINT	DELETE	DELAON	N
	COMPLETE_RELATION	SELECT	VERRCC	N
	COMPLETE_RELATION	DELETE	DELCMPR	S
	COMPLETE_RELATION	INSERT	INSCRC	S
	CONSTRAINT_INPUT	DELETE	DELCONI	N
	DESC_TEXT	DELETE	DELTEXT	S

<u>COMMAND</u>	<u>TABLE NAME</u>	<u>FUNCTI</u>	<u>MODULE</u>	(S=SQL (N=NDML)
	ENTITY_CLASS	SELECT	DELMTKC	N
	ENTITY_CLASS	SELECT	VERENT	N
	IISS_KEYWORD	SELECT	VERKW	N
	IISS_KEYWORD	INSERT	INSKW	S
	INHERITED_ATT_USE	SELECT	DLMIGRC	S
	INHERITED_ATT_USE	DELETE	DELIAUC	S
	KEY_CLASS	SELECT	VERKC	N
	KEY_CLASS	SELECT	DELMTKC	S
	KEY_CLASS	DELETE	DELKC	S
	KEY_CLASS_MEMBER	SELECT	ADDMIG	N
	KEY_CLASS_MEMBER	DELETE	DELKCMT	S
	KEY_CLASS_MEMBER	SELECT	DRPMGRC	N
	KEY_CLASS_MEMBER	SELECT	DELMTKC	S
	RC_KEYWORD	SELECT	ADDKWR	N
	RC_KEYWORD	INSERT	INSKWRC	S
	RC_KEYWORD	DELETE	DELEKWR	S
	RELATION_CLASS	SELECT	GETCARD	N
	RELATION_CLASS	SELECT	VERRC	N
	RELATION_CLASS	INSERT	UPDTRC	S
	VIEW_QUAL_XREF	SELECT	VERVWQU	N
ALTER UNION	DATA_BASE	SELECT	VERDB	N
	DATA_BASE	SELECT	VERDF	N
	DATA_FIELD	SELECT	VERDF	N
	DATA_FIELD	SELECT	VERUDT	N
	DATA_FILED	SELECT	VERDFID	N
	DISTRIBUTED_RULES	DELETE	DRPRULE	N
	ECRTUD	SELECT	SELURT	N
	ECRTUD	INSERT	INSUNIN	N
	ECRTUD	SELECT	SELUNIN	N
	ECRTUD	SELECT	VERUNIN	N
	ECRTUD	DELETE	DELUNIN	N
	EC_RT_MAPPING	DELETE	DRPECRT	N
	EC_RT_MAPPING	DELETE	DEL1ERT	N
	EC_RT_MAPPING	SELECT	ECRTALL	N
	ELEMENTARY_DATA_			
	FIELD	SELECT	VERUDT	N
	ENTITY_CLASS	SELECT	VERECN	N
	ENTITY_NAME	SELECT	VERECN	N
	MODEL_CLASS	SELECT	VERECN	N
	RECORD_TYPE	SELECT	VERDF	N
	RECORD_TYPE	SELECT	VERRT	N
	USER_DEF_DATA_			
	TYPE	SELECT	VERUDT	N

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
CHECK MODEL	ATTRIBUTE_CLASS	SELECT	CHKATT	N
	ATTRIBUTE_USE_CL	SELECT	CHKATT	N
	COMPLETE_RELATION	SELECT	VERRCC	N
	ENTITY_CLASS	SELECT	CHKLOOP	S
	ENTITY_CLASS	SELECT	GETECS	N
	ENTITY_NAME	SELECT	RETRECP	N
	ENTITY_NAME	SELECT	GETECS	N
	KEY_CLASS	SELECT	CHKKEYS	S
	KEY_CLASS MEMBER	SELECT	CHKKEYS	S
	MODEL_CLASS	SELECT	CHKLOOP	S
	MODEL_CLASS	SELEC	VERMOD	N
	OWNED_ATTRIBUTE	SELECT	CHKATT	N
	RELATION_CLASS	SELECT	CHKREL	N
	RELATION_CLASS	SELECT	TLOOPCK	S
	RELATION_CLASS	SELECT	BLOOPCK	S
	RELATION_CLASS	SELECT	CHKLOOP	S
COMBINE ENTITY	AC_KEYWORD	SELECT	GENAKW	N
	ATTRIBUTE_CLASS	SELECT	CMBOA	N
	ATTRIBUTE_CLASS	SELECT	VERATT	N
	ATTRIBUTE_NAME	SELECT	CMBOA	N
	ATTRIBUTE_NAME	SELECT	CMBACAL	N
	ATTRIBUTE_NAME	SELECT	VERATT	N
	ATTRIBUTE_USE_CL	SELECT	BLKCL1	N
	ATTRIBUTE_USE_CL	SELECT	SELIAUC	S
	COMPLETE_RELATION	SELECT	VERRCC	N
	DESC TEXT	SELECT	GENDESC	N
	DOMAIN_CLASS	SELECT	CMBOA	N
	EC_KEYWORD	SELECT	CMBEKW	N
	EC_KEYWORD	SELECT	VERKWE	N
	ENTITY_CLASS	SELECT	CMBALI	N
	ENTITY_CLASS	SELECT	VERENT	N
	ENTITY_CLASS	SELECT	SELIAUC	S
	ENTITY_CLASS	SELECT	BLKCL1	N
	ENTITY_NAME	SELECT	VERENT	N
	ENTITY_NAME	SELECT	SELECNM	N
	ENTITY_NAME	SELECT	VERALI	N
	ENTITY_NAME	SELECT	CMBALI	N
	IISS_KEYWORD	SELECT	CMBRKW	N
	IISS_KEYWORD	SELECT	VERKWE	N
	IISS_KEYWORD	SELECT	CMBEKW	N
	IISS_KEYWORD	SELECT	GENAKW	N
	IISS_KEYWORD	SELECT	VERKWR	N
	INHERITED_ATT_USE	SELECT	SELIAUC	S
	KEY_CLASS	SELECT	BLKCL1	N

<u>COMMAND</u>	<u>TABLE NAME</u>	<u>FUNCTI</u>	<u>MODULE</u>	(S=SQL N=NDML)
	KEY CLASS MEMBER	SELECT	BLKCL1	N
	MODEL CLASS	SELECT	VERMOD	N
	OWNED ATTRIBUTE	SELECT	CMBOA	N
	RC_KEYWORD	SELECT	CMBRKW	N
	RC_KEYWORD	SELECT	VERKWR	N
	RELATION CLASS	SELECT	DEPFROM	N
	RELATION CLASS	SELECT	VERRC	N
	RELATION CLASS	SELECT	SELRCNM	N
	RELATION CLASS	SELECT	INDFROM	N
COMPARE MODEL	AC_KEYWORD	SELECT	RETACKW	N
	ATTRIBUTE CLASS	SELECT	RETRAC1	N
	ATTRIBUTE NAME	SELECT	RETRAC1	N
	ENTITY CLASS	SELECT	RETREC1	N
	ENTITY CLASS	SELECT	VERENT	N
	ENTITY CLASS	SELECT	RETECKW	N
	ENTITY CLASS	SELECT	RECKW2	N
	ENTITY CLASS	SELECT	RELKW	N
	ENTITY NAME	SELECT	RETREC1	N
	ENTITY NAME	SELECT	RETRECP	N
	ENTITY NAME	SELECT	VERENT	N
	IISS_KEYWORD	SELECT	RETRCKW	N
	IISS_KEYWORD	SELECT	RETACKW	N
	IISS_KEYWORD	SELECT	RETECKW	N
	MODEL CLASS	SELECT	VERMOD	N
	RC_KEYWORD	SELECT	RETRCKW	N
	RC_KEYWORD	SELECT	RRCKW2	N
	RELATION CLASS	SELECT	RETRCKW	N
	RELATION CLASS	SELECT	RRCKW2	N
	RELATION CLASS	SELECT	GETRCID	N
COPY ATTRIBUTE	AC_KEYWORD	SELECT	WRTACKW	N
	AC_KEYWORD	INSERT	INSKWAC	S
	AC_KEYWORD	SELECT	GENAKW	N
	ATTRIBUTE CLASS	SELECT	VERACNM	N
	ATTRIBUTE CLASS	SELECT	VERATT	N
	ATTRIBUTE NAME	SELECT	WRTANAM	N
	ATTRIBUTE NAME	SELECT	WRTALI	N
	ATTRIBUTE NAME	SELECT	VERATT	N
	ATTRIBUTE NAME	SELECT	VERACNM	N
	ATTRIBUTE NAME	SELECT	FCOPATT	N
	DESC TEXT	SELECT	GENDESC	N
	DOMAIN CLASS	SELECT	VERACNM	N
	IISS_KEYWORD	SELECT	GENAKW	N
	MODEL CLASS	SELECT	VERMOD	N



COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
COPY DATABASE	DATA_BASE	SELECT	ALLDB	N
	DATA_BASE	SELECT	VERDBAL	N
	DATA_BASE_AREA	SELECT	VERDBA	N
	DATA_FIELD	SELECT	RETRFLD	S
	DB AREA			
	ASSIGNMENT	SELECT	SELDBAA	N
	DB PASSWORD	SELECT	VERPASS	N
	DESC TEXT	SELECT	GENDESC	N
	DF SET LINKAGE	SELECT	GETDFSL	N
	PSB PCB	SELECT	VERPCB	N
	RECORD SET	SELECT	ALLSET	N
	RECORD_TYPE	SELECT	ALLREC	N
	SCHEMA NAMES	SELECT	VERSCH	N
	SET_TYPE_MEMBER	SELECT	GETMEMB	N
COPY DBMS	DATA_BASE	SELECT	SELDB	N
	DATA_BASE_AREA	SELECT	VERDBA	N
	DATA_FIELD	SELECT	RETRFLD	S
	DBMS_ON_HOST	SELECT	SELHOST	N
	DB AREA			
	ASSIGNMENT	SELECT	SELDBAA	N
	DB PASSWORD	SELECT	VERPASS	N
	DESC TEXT	SELECT	GENDESC	N
	DF SET LINKAGE	SELECT	GETDFSL	N
	IISS DBMS	SELECT	ALLDBMS	N
	IISS DBMS	SELECT	VERDBMS	N
	PSB PCB	SELECT	VERPCB	N
	RECORD SET	SELECT	ALLSET	N
	RECORD_TYPE	SELECT	ALLREC	N
	SCHEMA NAMES	SELECT	VERSCH	N
	SET_TYPE_MEMBER	SELECT	GETMEMB	N
COPY DESC TYPE	DESCRIPTION_TYPE	SELECT	SELDSTP	N
	DESCRIPTION_TYPE	SELECT	VERDSTP	N
COPY DESCRIPTION	ATTRIBUTE_CLASS	SELECT	VERATT	N
	DATA_BASE	SELECT	VERDB	N
	DATA_FIELD	SELECT	VERDFLD	N
	DATA_ITEM	SELECT	VERDI	N
	DESC TEXT	SELECT	GENDESC	N
	DESC TEXT	SELECT	SELDESC	N
	DESC TEXT	SELECT	GEN1DSC	N
	DESC TEXT	SELECT	SEL1DSC	N
	DESC TEXT	INSERT	INSDESC	N
	DOMAIN_CLASS	SELECT	VERDOM	N
	ENTITY_CLASS	SELECT	VERENT	N

<u>COMMAND</u>	<u>TABLE NAME</u>	<u>FUNCTI MODULE</u>	(S=SQL N=NDML)
	IISS_HOST	SELECT VERHST	N
	IISS_KEYWORD	SELECT VERKW	N
	MODEL_CLASS	SELECT VERMOD	N
	RECORD_SET	SELECT VERRSET	N
	RECORD_TYPE	SELECT VERRT	N
	USER_DEF_DATA_		
	TYPE	SELECT VERUDTN	N
	USER_VIEW	SELECT VERVIEW	N
COPY DOMAIN	DESC_TEXT	SELECT GENDESC	N
	DOMAIN_CLASS	SELECT ALLDOM	N
	DOMAIN_RANGE	SELECT ALLRNG	N
	DOMAIN_VALUE	SELECT ALLVALU	N
	USER_DEF_DATA_		
	TYPE	SELECT ALLD	N
COPY ENTITY	AC_KEYWORD	SELECT COPAKW	N
	AC_KEYWORD	SELECT WRTACKW	N
	ATTRIBUTE_CLASS	SELECT COPOA	N
	ATTRIBUTE_CLASS	SELECT VERATT	N
	ATTRIBUTE_CLASS	INSERT INSAC	S
	ATTRIBUTE_CLASS	SELECT COPYAC	N
	ATTRIBUTE_NAME	SELECT COPOA	N
	ATTRIBUTE_NAME	SELECT WRTALI	N
	ATTRIBUTE_NAME	SELECT COPAALI	N
	ATTRIBUTE_NAME	INSERT INSACNM	S
	ATTRIBUTE_NAME	SELECT VERATT	N
	ATTRIBUTE_USE_CL	SELECT BLKCL1	N
	ATTRIBUTE_USE_CL	SELECT SELIKEY	N
	ATTRIBUTE_USE_CL	SELECT SELKCM	N
	ATTRIBUTE_USE_CL	INSERT INSAUC	S
	ATTRIBUTE_USE_CL	SELECT COPYAC	N
	ATTRIBUTE_USE_CL	SELECT DPKCLST	S
	ATTRIBUTE_USE_CL	SELECT SELIAUC	S
	COMPLETE_RELATION	SELECT VERRCC	N
	DESC_TEXT	SELECT WRTDESC	S
	DESC_TEXT	INSERT WRTDESC	S
	DESC_TEXT	SELECT GENDESC	N
	DOMAIN_CLASS	SELECT COPOA	N
	EC_KEYWORD	SELECT COPEKW	N
	EC_KEYWORD	SELECT WRTECKW	N
	ENTITY_CLASS	INSERT INSEC	S
	ENTITY_CLASS	SELECT VERENT	N
	ENTITY_CLASS	SELECT COPVALI	N
	ENTITY_CLASS	SELECT SELIAUC	S
	ENTITY_CLASS	SELECT BLKCL1	N
	ENTITY_CLASS	SELECT COPYAC	N

<u>COMMAND</u>	<u>TABLE NAME</u>	<u>FUNCTI</u>	<u>MODULE</u>	(S=SQL N=NDML)
	ENTITY_NAME	SELECT	COPVALI	N
	ENTITY_NAME	SELECT	DEPENT	S
	ENTITY_NAME	SELECT	VERENT	N
	ENTITY_NAME	SELECT	ECSTRUC	S
	ENTITY_NAME	SELECT	DEPEC	S
	ENTITY_NAME	SELECT	WRTENAM	N
	ENTITY_NAME	SELECT	SELECNM	N
	IISS_KEYWORD	SELECT	COPEKW	N
	IISS_KEYWORD	SELECT	COPAKW	N
	IISS_KEYWORD	SELECT	COPRKW	N
	INHERITED_ATT_USE	SELECT	SELIAUC	S
	INHERITED_ATT_USE	SELECT	SELIKEY	N
	INHERITED_ATT_USE	SELECT	VERITAG	N
	KEY_CLASS	SELECT	BLKCL1	N
	KEY_CLASS	INSERT	INSKC	S
	KEY_CLASS	SELECT	KEYLOOK	N
COPY ENTITY	KEY_CLASS	SELECT	GENKEY	N
	KEY_CLASS	SELECT	DPKCLST	S
	KEY_CLASS_MEMBER	SELECT	BLKCL1	N
	KEY_CLASS_MEMBER	SELECT	SELIKEY	N
	KEY_CLASS_MEMBER	SELECT	DPKCLST	S
	KEY_CLASS_MEMBER	SELECT	KEYLOOK	N
	KEY_CLASS_MEMBER	SELECT	VERKCM	N
	KEY_CLASS_MEMBER	INSERT	INSKCM	S
	KEY_CLASS_MEMBER	SELECT	SELKCM	N
	MODEL_CLASS	SELECT	VERMOD	N
	OWNED_ATTRIBUTE	SELECT	VEROAC	N
	OWNED_ATTRIBUTE	SELECT	COPOA	N
	OWNED_ATTRIBUTE	INSERT	INSOAC	S
	OWNED_ATTRIBUTE	SELECT	COPYAC	N
	RC_KEYWORD	SELECT	COPRKW	N
	RELATION_CLASS	SELECT	DEPENT	S
	RELATION_CLASS	SELECT	DPKCLST	S
	RELATION_CLASS	SELECT	BLRCKC1	S
	RELATION_CLASS	SELECT	LVLCHK1	S
	RELATION_CLASS	SELECT	DEPREL	S
	RELATION_CLASS	SELECT	ECSTRUC	S
	RELATION_CLASS	SELECT	DEPEC	S
	RELATION_CLASS	SELECT	DEPKCM	N
	RELATION_CLASS	SELECT	GENDEP	N
	RELATION_CLASS	SELECT	GENIND	N
	RELATION_CLASS	SELECT	SELRELC	N

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
COPY HOST	DESC_TEXT	SELECT	GENDESC	N
	IISS_HOST	SELECT	VERHST	N
	IISS_HOST	SELECT	SELHSTS	N
	IISS_PSB	SELECT	GENPSB	N
COPY MAP	ATTRIBUTE_USE_CL	SELECT	ALLMREC	N
	ATTRIBUTE_USE_CL	SELECT	SELAPRM	N
	ATTRIBUTE_USE_CL	SELECT	SELMTAG	N
	ATTRIBUTE_USE_CL	SELECT	SELMREC	N
	AUC_IS_MAPPING	SELECT	ALLMREC	N
	AUC_IS_MAPPING	SELECT	SELMREC	N
	AUC_IS_MAPPING	SELECT	GENCMPX	N
	AUC_IS_MAPPING	SELECT	SELMTAG	N
	AUC_PARM	SELECT	GENCMPX	N
	AUC_PARM	SELECT	SELAPRM	N
	AUC_ST_MAPPING	SELECT	GENTOST	N
	CONST_PARM	SELECT	GTCNTPR	N
	DATA_BASE	SELECT	ALLMREC	N
	DATA_BASE	SELECT	ALLRSET	S
	DATA_BASE	SELECT	SELDFPM	N
	DATA_BASE	SELECT	VERREC	N
	DATA_BASE	SELECT	GENECRT	N
	DATA_BASE	SELECT	ALLMREL	S
	DATA_BASE	SELECT	SELRMAP	N
	DATA_BASE	SELECT	SELSET	S
	DATA_BASE	SELECT	VERDB	N
	DATA_BASE	SELECT	SELRTPM	N
	DATA_BASE	SELECT	SELMTAG	N
	DATA_BASE	SELECT	SELDBNM	N
	DATA_FIELD	SELECT	RETRFLD	S
	DATA_FIELD	SELECT	GENCRUN	N
	DATA_FIELD	SELECT	SELDFPM	N
	DESC_TEXT	SELECT	GENDESC	N
	DF_PARM	SELECT	SELDFPM	N
	DISTRIBUTED_RULES	SELECT	VERRULE	N
	DISTRIBUTED_RULES	SELECT	VERRULE	N
	DI_PARM	SELECT	SELDI	N
	ECRTUD	SELECT	GENCRUN	N
	ECRTUD	SELECT	GENEUN	N
	EC_RT_MAPPING	SELECT	GENCREC	N
	EC_RT_MAPPING	SELECT	GENECRT	N
	ENTITY_CLASS	SELECT	VERENT	N
	ENTITY_NAME	SELECT	ALLMREL	S
	ENTITY_NAME	SELECT	SELPEC	N
	ENTITY_NAME	SELECT	ALLMENT	N
	ENTITY_NAME	SELECT	ALLMREC	N

<u>COMMAND</u>	<u>TABLE NAME</u>	<u>FUNCTI</u>	<u>MODULE</u>	(S-SQL N=NDML)
	ENTITY_NAME	SELECT	SELSET	S
	ENTITY_NAME	SELECT	SELMREC	N
	ENTITY_NAME	SELECT	ALLRSET	S
	ENTITY_NAME	SELECT	SELAPRM	N
	ENTITY_NAME	SELECT	ALLECRT	N
COPY MAP	ENTITY_NAME	SELECT	VERENT	N
	ENTITY_NAME	SELECT	SELECNM	N
	ENTITY_NAME	SELECT	GENCREC	N
	HORIZONTAL_PART	SELECT	GENCRPT	N
	HORIZONTAL_PART	SELECT	GENEHP	N
	MODEL_CLASS	SELECT	ALLMENT	N
	MODEL_CLASS	SELECT	ALLECRT	N
	MODULE_PARAMETER	SELECT	GTCNTPR	N
	MODULE_PARAMETER	SELECT	SELAPRM	N
	MODULE_PARAMETER	SELECT	SELDFPM	N
	MODULE_PARAMETER	SELECT	SELRTPM	N
	PROJECT_DATA_			
	FIELD	SELECT	GENTODF	N
	RC_BASED_REC_SET	SELECT	ALLMREL	S
	RC_BASED_REC_SET	SELECT	SELRMAP	N
	RC_BASED_REC_SET	SELECT	SELSET	S
	RC_BASED_REC_SET	SELECT	ALLRSET	S
	RECORD_SET	SELECT	VERRSET	N
	RECORD_TYPE	SELECT	ALLMREC	N
	RECORD_TYPE	SELECT	VERRTNO	N
	RECORD_TYPE	SELECT	SELMTAG	N
	RECORD_TYPE	SELECT	GENECRT	N
	RECORD_TYPE	SELECT	VERREC	N
	RECORD_TYPE	SELECT	VERRT	N
	RECORD_TYPE	SELECT	SELRTPM	N
	RECORD_TYPE	SELECT	GENEUN	N
	RECORD_TYPE	SELECT	GENTOST	N
	RECORD_TYPE	SELECT	SELMREC	N
	RELATION_CLASS	SELECT	ALLMREL	S
	RELATION_CLASS	SELECT	ALLRSET	S
	RELATION_CLASS	SELECT	VERRC	N
	RELATION_CLASS	SELECT	SELRMAP	N
	RELATION_CLASS	SELECT	SELSET	S
	RT_PARM	SELECT	SELRTPM	N

COMMAND	TABLE NAME	FUNCTI MODULE	(S=SQL N=NDML)
COPY MODEL	AC KEYWORD	SELECT GENAKW	N
	ATTRIBUTE_CLASS	SELECT MGENOA	N
	ATTRIBUTE_CLASS	SELECT MODATT	N
	ATTRIBUTE_CLASS	SELECT MODOATT	N
	ATTRIBUTE_NAME	SELECT MGENOA	N
	ATTRIBUTE_NAME	SELECT MODOATT	N
	ATTRIBUTE_NAME	SELECT MODATT	N
	ATTRIBUTE_USE_CL	SELECT BLKCLST	N
	ATTRIBUTE_USE_CL	SELECT SELIKEY	N
	ATTRIBUTE_USE_CL	SELECT SELIAUC	S
	ATTRIBUTE_USE_		
	CLASS	SELECT MODOATT	N
	COMPLETE RELATION	SELECT VERRCC	N
	DESC TEXT	SELECT GENDESC	N
	DOMAIN_CLASS	SELECT MODOATT	N
	DOMAIN_CLASS	SELECT MODATT	N
	EC KEYWORD	SELECT GENEKW	N
	EC KEYWORD	SELECT COPEKW	N
	ENTITY_CLASS	SELECT BLKCLST	N
	ENTITY_CLASS	SELECT MODENT	N
	ENTITY_CLASS	SELECT VERENT	N
	ENTITY_CLASS	SELECT TOPNODE	S
	ENTITY_CLASS	SELECT SELIAUC	S
	ENTITY_CLASS	SELECT BLRCKC	S
	ENTITY_CLASS	SELECT MODREL	S
	ENTITY_NAME	SELECT MODENT	N
	ENTITY_NAME	SELECT VERENT	N
	IISS KEYWORD	SELECT GENAKW	N
	IISS KEYWORD	SELECT GENEKW	N
	IISS KEYWORD	SELECT COPEKW	N
	IISS KEYWORD	SELECT GENRKW	N
	INHERITED ATT_USE	SELECT SELIKEY	N
	INHERITED ATT_USE	SELECT SELIAUC	S
	KEY_CLASS	SELECT MODKC	N
	KEY_CLASS_MEMBER	SELECT MODKC	N
	KEY_CLASS_MEMBER	SELECT SELIKEY	N
	MODEL_CLASS	SELECT COPTOPE	S
	MODEL_CLASS	SELECT VERMOD	N
	MODEL_CLASS	SELECT BLRCKC	S
	MODEL_CLASS	SELECT MODREL	S
	OWNED_ATTRIBUTE	SELECT MGENOA	N
	OWNED_ATTRIBUTE	SELECT MODATT	N
	OWNED_ATTRIBUTE	SELECT MODOATT	N
	RC KEYWORD	SELECT GENRKW	N
	RELATION_CLASS	SELECT TOPNODE	S
	RELATION_CLASS	SELECT MODREL	S
	RELATION_CLASS	SELECT BLRCKC	S

COMMAND	TABLE NAME	FUNCTION	MODULE	(S=SQL N=NDML)
COPY MODULE	MODULE_PARAMETER	SELECT	CPY1SMD	N
	SOFTWARE_MODULE	SELECT	VERSMOD	N
	SOFTWARE_MODULE	SELECT	SELMODS	N
COPY RECORD	DATA_FIELD	SELECT	RETRFLD	S
	DB_AREA			
	ASSIGNMENT	SELECT	SELDBAA	N
	RECORD_TYPE	SELECT	ALLREC	N
COPY SET	RECORD_TYPE	SELECT	VERRT	N
	DF_SET_LINKAGE	SELECT	GETDFSL	N
	RECORD_SET	SELECT	ALLSET	N
	RECORD_SET	SELECT	VERDBST	N
COPY VIEW	SET_TYPE_MEMBER	SELECT	GETMEMB	N
	ATTRIBUTE_USE_CL	SELECT	GTAUCPR	N
	ATTRIBUTE_USE_CL	SELECT	WRTSLCT	N
	ATTRIBUTE_USE_CL	SELECT	SELTGEC	N
	AUC_PARM	SELECT	GTAUCPR	N
	CONST_PARM	SELECT	GTCNTPR	N
	DATA_ITEM	SELECT	GENALG	N
	DATA_ITEM	SELECT	WRTDITM	N
	DATA_ITEM	SELECT	GTDIPR	N
	DESC_TEXT	SELECT	GENDESC	N
	DI_PARM	SELECT	GENALG	N
	DI_PARM	SELECT	GTDIPR	N
	ENTITY_NAME	SELECT	GTAUCPR	N
	ENTITY_NAME	SELECT	SELECX	N
	MODULE_PARAMETER	SELECT	GTAUCPR	N
	MODULE_PARAMETER	SELECT	GTDIPR	N
	MODULE_PARAMETER	SELECT	GTCNTPR	N
	PROJECT_DATA_ITEM	SELECT	WRTSLCT	N
	USER_VIEW	SELECT	GTDIPR	N
	USER_VIEW	SELECT	VERVIEW	N
	USER_VIEW	SELECT	GTVIEWS	N
	VIEW_EC_XREF	SELECT	SELECX	N
	VIEW_QUALIFY			
	CRITERIA	SELECT	WRTWHCL	N

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
CREATE ALIAS	ATTRIBUTE_CLASS	SELECT	VERATT	N
	ATTRIBUTE_NAME	SELECT	VERATT	N
	ATTRIBUTE_NAME	INSERT	INSACNM	S
	ENTITY_CLASS	SELECT	VERENT	N
	ENTITY_NAME	SELECT	VERENT	N
	ENTITY_NAME	INSERT	INSECNM	S
CREATE ATTRIBUTE	AC_KEYWORD	SELECT	ADDKWA	N
	AC_KEYWORD	INSERT	INSKWAC	S
	ATTRIBUTE_CLASS	SELECT	VERATT	N
	ATTRIBUTE_CLASS	INSERT	INSAC	S
	ATTRIBUTE_NAME	SELECT	VERATT	N
	ATTRIBUTE_NAME	INSERT	INSACNM	S
	DOMAIN_CLASS	SELECT	VERDOM	N
	EC_KEYWORD	SELECT	ADDKWE	N
	EC_KEYWORD	INSERT	INSKWEC	S
	IISS_KEYWORD	SELECT	VERKW	N
	IISS_KEYWORD	INSERT	INSKW	S
	RC_KEYWORD	SELECT	ADDKWR	N
	RC_KEYWORD	INSERT	INSKWRC	S
CREATE DESCRIPTION	DESCRIPTION_TYPE	SELECT	VERDSCT	N
	DESCRIPTION_TYPE	INSERT	INSDSCT	N
CREATE DOMAIN	DATA_TYPE	SELECT	VERTYP	N
	DOMAIN_CLASS	SELECT	VERDOM	N
	DOMAIN_CLASS	INSERT	INSDOM	N
	DOMAIN_VALUE	INSERT	INSVAL	N
	DOMAIN_VALUE	INSERT	INSRNG	N
	MODULE_PARAMETER	INSERT	INSPARM	N
	MODULE_PARAMETER	SELECT	VERMPDT	N
	SOFTWARE_MODULE	INSERT	INSSMOD	N
	USER_DEF_DATA_	SELECT	VERSDT	N
	TYPE	SELECT	VERDTD	N
	USER_DEF_DATA_	SELECT	VERDT	N
	TYPE	INSERT	INSDT	N
	USER_DEF_DATA_	INSERT	INSVMD	N
	TYPE	INSERT	INSVMD	N



COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
CREATE ENTITY	ATTRIBUTE_CLASS	SELECT	VERATT	N
	ATTRIBUTE_NAME	SELECT	VERATT	N
	ATTRIBUTE_USE_CL	SELECT	VERAUC	N
	ATTRIBUTE_USE_CL	INSERT	INSAUC	S
	EC_KEYWORD	SELECT	ADDKWE	N
	EC_KEYWORD	INSERT	INSKWEC	S
	ENTITY_CLASS	SELECT	VERENT	N
	ENTITY_CLASS	INSERT	INSEC	S
	ENTITY_NAME	SELECT	VERENT	N
	ENTITY_NAME	INSERT	INSECNM	S
	IISS_KEYWORD	SELECT	VERKW	N
	IISS_KEYWORD	INSERT	INSKW	S
	KEY_CLASS	SELECT	VERKC	N
	KEY_CLASS	INSERT	INSKC	S
	KEY_CLASS_MEMBER	INSERT	INSKCM	S
	OWNED_ATTRIBUTE	INSERT	INSOAC	S
	OWNED_ATTRIBUTE	SELECT	VEROAC	N
CREATE MAP	ATTRIBUTE_CLASS	SELECT	VERTAG	N
	ATTRIBUTE_USE_CL	SELECT	VERTAG	N
	ATTRIBUTE_USE_CL	SELECT	VERAUC	N
	AUC_IS_MAPPING	SELECT	CHKAUCV	N
	AUC_IS_MAPPING	SELECT	VOMAPS	N
	AUC_IS_MAPPING	SELECT	VERAISM	N
	AUC_IS_MAPPING	SELECT	VERAIMR	N
	AUC_IS_MAPPING	INSERT	INSAISM	N
	AUC_IS_MAPPING	SELECT	VERAIM	N
	AUC_ST_MAPPING	SELECT	CHKSTMP	N
	AUC_ST_MAPPING	SELECT	CHKAUCV	N
	AUC_ST_MAPPING	INSERT	INSAUCS	N
	AUC_ST_MAPPING	SELECT	FNDASA	N
	AUC_ST_MAPPING	SELECT	VOMAPS	N
	COMPONENT_DATA_			
	FIELD	SELECT	VERMAPD	N
	DATA_BASE	SELECT	VERDTFL	N
	DATA_BASE	SELECT	VERDFID	N
	DATA_BASE	SELECT	VERDB	N
	DATA_FIELD	SELECT	VERMAPD	N
	DATA_FIELD	SELECT	VERDTFL	N
	DATA_FIELD	SELECT	VERDFID	N
	DISTRIBUTED_RULES	SELECT	VERRULE	N
	DISTRIBUTED_RULES	INSERT	INSRULE	N
	DISTRIBUTED_RULES	INSERT	INSRULE	N
	EC_RT_MAPPING	SELECT	VERECRT	N
	EC_RT_MAPPING	INSERT	INSECRT	N
	EC_RT_MAPPING	SELECT	VERECRT	N

<u>COMMAND</u>	<u>TABLE NAME</u>	<u>FUNCTI</u>	<u>MODULE</u>	(S-SQL N=NDML)
	EC_RT_MAPPING	INSERT	INSECR	N
	ELEMENTARY_DATA_			
	FIELD	SELECT	GETDTN	N
	ENTITY_CLASS	SELECT	VERENT	N
	ENTITY_NAME	SELECT	VERTAG	N
	ENTITY_NAME	SELECT	VERENT	N
	HORIZONTAL_PART	SELECT	VERHPDF	N
	HORIZONTAL_PART	SELECT	VERHPST	N
	HORIZONTAL_PART	SELECT	VERHZP	N
	PROJECT_DATA_			
	FIELD	SELECT	FNDPDF	N
	PROJECT_DATA_			
	FIELD	INSERT	INSPDF	N
	RC_BASED_REC_SET	SELECT	VERRCBS	N
	RC_BASED_REC_SET	SELECT	VERRCMP	N
	RC_BASED_REC_SET	INSERT	INSRCRS	S
	RECORD_SET	SELECT	VERHPST	N
	RECORD_SET	SELECT	VOMAPS	N
	RECORD_SET	SELECT	VERSMS	N
	RECORD_TYPE	SELECT	VERHPDF	N
	RECORD_TYPE	SELECT	VERDTFL	N
	RECORD_TYPE	SELECT	VERHPST	N
	RELATION_CLASS	SELECT	VERRC	N
CREATE MAP	SET_TYPE_MEMBER	SELECT	FND1MEM	N
	USER_DEF_DATA_			
	TYPE	SELECT	VERTAG	N
	USER_DEF_DATA_			
	TYPE	SELECT	VERDTYP	N
CREATE MODEL	MODEL_CLASS	SELECT	VERMOD	N
	MODEL_CLASS	INSERT	INSMOD	S
CREATE PARTITION	DATA_BASE	SELECT	VERDBRT	N
	ENTITY_CLASS	SELECT	VERECN	N
	ENTITY_NAME	SELECT	VERECN	N
	HORIZONTAL_PART	INSERT	INSPART	N
	HORIZONTAL_PART	SELECT	VERPART	N
	HORIZONTAL_PART	SELECT	VERHPN	N
	MODEL_CLASS	SELECT	VERECN	N

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
CREATE RELATION	ATTRIBUTE_USE_CL	SELECT	VERAUC	N
	ATTRIBUTE_USE_CL	SELECT	VERKCMG	N
	ATTRIBUTE_USE_CL	SELECT	ADDMIG	N
	ATTRIBUTE_USE_CL	INSERT	INSAUC	S
	COMPLETE			
	RELATION	SELECT	VERRCC	N
	COMPLETE			
	RELATION	INSERT	INSCRC	S
	IISS_KEYWORD	INSERT	INSKW	S
	INHERITED_ATT			
	USE	INSERT	INSIAUC	S
	KEY_CLASS	SELECT	VERKC	N
	KEY_CLASS_MEMBER	SELECT	ADDMIG	N
	KEY_CLASS_MEMBER	SELECT	VERKCMG	N
	RC_KEYWORD	SELECT	ADDKWR	N
	RC_KEYWORD	INSERT	INSKWRC	S
	RELATION_CLASS	INSERT	INSRC	S
	RELATION_CLASS	SELECT	VERRC	N
CREATE UNION	DATA_BASE	SELECT	VERDB	N
	DATA_BASE	SELECT	VERDF	N
	DATA_FIELD	SELECT	VERDF	N
	DATA_FIELD	SELECT	VERUDT	N
	DATA_FIELD	SELECT	VERDFID	N
	ECRTUD	SELECT	VERUNIN	N
	ECRTUD	INSERT	INSUNIN	N
	ELEMENTARY_DATA			
	FIELD	SELECT	VERUDT	N
	ENTITY_CLASS	SELECT	VERECN	N
	ENTITY_NAME	SELECT	VERECN	N
	MODEL_CLASS	SELECT	VERECN	N
	RECORD_TYPE	SELECT	VERDF	N
	RECORD_TYPE	SELECT	VERRT	N
	USER_DEF_DATA			
	TYPE	SELECT	VERUDT	N

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
CREATE VIEW	ATTRIBUTE_CLASS	SELECT	ALLVIEW	N
	ATTRIBUTE_CLASS	SELECT	VERTAG	N
	ATTRIBUTE_CLASS	SELECT	GETDOM	N
	ATTRIBUTE_CLASS	SELECT	GTDTTYP	N
	ATTRIBUTE_USE_CL	SELECT	ALLVIEW	N
	ATTRIBUTE_USE_CL	SELECT	GETDOM	N
	ATTRIBUTE_USE_CL	SELECT	VERTAG	N
	ATTRIBUTE_USE_CL	SELECT	GTDTTYP	N
	COMPLETE_RELATION	SELECT	VALVWRC	N
	DATA_ITEM	INSERT	INSDI	N
	ENTITY_CLASS	SELECT	VERENT	N
	ENTITY_NAME	SELECT	VERTAG	N
	ENTITY_NAME	SELECT	SELECNM	N
	KEY_CLASS_MEMBER	SELECT	VALVWRC	N
	PROJECT_DATA_ITEM	INSERT	INSPDI	N
	RELATION_CLASS	SELECT	VERRC	N
	RELATION_CLASS	SELECT	VALVWRC	N
	USER_DEF_DATA_			
	TYPE	SELECT	VERSDT	N
	USER_DEF_DATA_			
	TYPE	SELECT	GTDTTYP	N
	USER_DEF_DATA_			
	TYPE	SELECT	VERDTYP	N
	USER_DEF_DATA_			
	TYPE	SELECT	VERTAG	N
	USER_DEF_DATA_			
	TYPE	SELECT	VERDTD	N
	USER_VIEW	SELECT	VERVIEW	N
	USER_VIEW	INSERT	INSVIEW	N
	VIEW_EC_XREF	INSERT	INSECRF	N
	VIEW_QUALIFY_			
	CRITERIA	INSERT	INSQCRF	N
	VIEW_QUAL_XREF	INSERT	INSQCTG	N

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
DEFINE ALGORITHM	ATTRIBUTE_CLASS	SELECT	VERTAG	N
	ATTRIBUTE_USE_CL	SELECT	VERTAG	N
	AUC_IS_MAPPING	SELECT	VERALMP	N
	AUC_IS_MAPPING	INSERT	INSAISM	N
	AUC_IS_MAPPING	SELECT	VERAISM	N
	AUC_PARM	INSERT	INSTGPA	N
	AUC_PARM	SELECT	VERALGI	N
	CONST_PARM	INSERT	INSCOPA	N
	DATA_BASE	SELECT	VERDBRT	N
	DATA_BASE	SELECT	VERDTFL	N
	DATA_FIELD	SELECT	VERDTFL	N
	DATA_ITEM	SELECT	VERDIDN	N
	DF_PARM	INSERT	INSDFPA	N
	DI_PARM	INSERT	INSDIPA	N
	ELEMENTARY_DATA_			
	FIELD	SELECT	GETDTN	N
	ENTITY_CLASS	SELECT	VERENT	N
	ENTITY_NAME	SELECT	VERTAG	N
	ENTITY_NAME	SELECT	VERENT	N
	MODULE_PARAMETER	SELECT	VERPARM	N
	RECORD_TYPE	SELECT	VERDBRT	N
	RECORD_TYPE	SELECT	VERRTNO	N
	RECORD_TYPE	SELECT	VERDTFL	N
	RT_PARM	INSERT	INSRTPA	N
	SOFTWARE_MODULE	SELECT	VERPARM	N
	USER_DEF_DATA_			
	TYPE	SELECT	VERTAG	N
	USER_DEF_DATA_			
	TYPE	SELECT	VERDTYP	N
	USER_VIEW	SELECT	VERDIDN	N
DEFINE DATABASE	DATA_BASE	SELECT	VERDB	N
	DATA_BASE	INSERT	INSDB	N
	DATA_BASE_AREA	SELECT	VERAREA	N
	DATA_BASE_AREA	INSERT	INSAREA	N
	DB_PASSWORD	INSERT	INSPWRD	N
	IISS_DBMS	SELECT	VERDBMS	N
	IISS_HOST	SELECT	VERHST	N
	IISS_PSB	SELECT	VERDPSB	N
	PSB_PCB	INSERT	INSPCB	N
	REUSABLE_NUMBER	SELECT	NRGET	S
	SCHEMA_NAMES	INSERT	INSSCH	N

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
DEFINE DBMS	DBMS_ON_HOST	INSERT	INSDBH	N
	IISS_DBMS	INSERT	INSDBMS	N
DEFINE HOST	DBMS_ON_HOST	INSERT	INSDBH	N
	IISS_HOST	SELECT	VERHST	N
	IISS_HOST	INSERT	INSHST	N
DEFINE MODULE	MODULE_PARAMETER	SELECT	VERAPRM	N
	MODULE_PARAMETER	INSERT	INSAPRM	N
	SOFTWARE_MODULE	SELECT	VERSMOD	N
	SOFTWARE_MODULE	INSERT	INSSMOD	N
	USER_DEF_DATA_ TYPE	SELECT	VERDT	N
DEFINE PSB	DBMS_ON_HOST	SELECT	VERDBH	N
	IISS_HOST	SELECT	VERHST	N
	IISS_PSB	INSERT	INS1PSB	N
	IISS_PSB	SELECT	VERPSB	N
DEFINE RECORD	DATA_BASE	SELECT	VERDBAS	N
	DATA_BASE_AREA	INSERT	INSAREA	N
	DATA_BASE_AREA	SELECT	VERAREA	N
	DATA_FIELD	INSERT	INSFLD	S
	DB_AREA			
	ASSIGNMENT	INSERT	INSDAA	N
	RECORD_TYPE	SELECT	VERRT	N
	RECORD_TYPE	INSERT	INSRTYP	S
	USER_DEF_DATA_ TYPE	SELECT	VERDTYP	N

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
DEFINE SET	DATA_BASE	SELECT	VERDTFL	N
	DATA_BASE	SELECT	VERDBAS	N
	DATA_FIELD	SELECT	VERDTFL	N
	DATA_FIELD	SELECT	VERDFLD	N
	DF SET LINKAGE	INSERT	INSDSL	S
	ELEMENTARY_DATA_			
	FIELD	SELECT	GETDTN	N
	RECORD_SET	INSERT	INSRSET	S
	RECORD_SET	SELECT	VERRSET	N
	RECORD_TYPE	SELECT	VERDTFL	N
	RECORD_TYPE	SELECT	VERRT	N
	SET_TYPE_MEMBER	INSERT	INSSTM	S
DESCRIBE	ATTRIBUTE_CLASS	SELECT	VERATT	N
	ATTRIBUTE_NAME	SELECT	VERATT	N
	DATA_BASE	SELECT	VERDB	N
	DATA_FIELD	SELECT	VERDFLD	N
	DATA_ITEM	SELECT	VERDI	N
	DESCRIPTION_TYPE	SELECT	VERDSTP	N
	DESC_TEXT	SELECT	OUTDESC	N
	DESC_TEXT	INSERT	RDDESC	S
	DESC_TEXT	INSERT	FILEINS	S
	DESC_TEXT	INSERT	RDDESC	S
	DESC_TEXT	INSERT	STRINS	S
	DESC_TEXT	DELETE	DELTXT	S
	DOMAIN_CLASS	SELECT	VERDOM	N
	ENTITY_CLASS	SELECT	VERENT	N
	ENTITY_NAME	SELECT	VERENT	N
	IISS KEYWORD	SELECT	VERKW	N
	RECORD_SET	SELECT	VERRSET	N
	RECORD_TYPE	SELECT	VERRT	N
	RELATION_CLASS	SELECT	VERRC	N
	USER_DEF_DATA_			
	TYPE	SELECT	VERUDTN	N
	USER_VIEW	SELECT	VERVIEW	N

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)	
DROP ALGORITHM	AUC_IS_MAPPING	DELETE	DLTPRFA	S	
	AUC_IS_MAPPING	DELETE	DLMTMAP	S	
	AUC_IS_MAPPING	DELETE	DLDTAG	S	
	AUC_IS_MAPPING	SELECT	CMPTAGS	S	
	AUC_IS_MAPPING	SELECT	VERAISM	N	
	AUC_PARM	SELECT	VERALG	N	
	COMPLEX_MAPPING_PARM	DELETE	DELCMP	S	
	COMPLEX_MAPPING_PARM	SELECT	CMPTAGS	S	
	COMPLEX_MAPPING_PARM	DELETE	DLMTMAP	S	
	DISTRIBUTED_RULES	DELETE	DRPRULE	N	
	EC_RT_MAPPING	DELETE	DRPECRT	N	
	EC_RT_MAPPING	DELETE	DEL1ERT	N	
	EC_RT_MAPPING	SELECT	ECRTALL	N	
	DROP ALIAS	ATTRIBUTE_CLASS	SELECT	VERATT	N
		ATTRIBUTE_NAME	SELECT	VERATT	N
		ATTRIBUTE_NAME	DELETE	DELECAL	S
ATTRIBUTE_NAME		DELETE	DELACAL	S	
ATTRIBUTE_NAME		SELECT	GETACAL	N	
ENTITY_CLASS		SELECT	VERENT	N	
ENTITY_NAME		SELECT	VERENT	N	
DROP ATTRIBUTE	ENTITY_NAME	SELECT	GETECAL	N	
	AC_KEYWORD	DELETE	DELACKW	S	
	ATTRIBUTE_CLASS	SELECT	VERATT	N	
	ATTRIBUTE_CLASS	DELETE	DELAC	S	
	ATTRIBUTE_NAME	SELECT	VERATT	N	
	ATTRIBUTE_NAME	DELETE	DELACNM	S	
	ATTRIBUTE_USE_CL	SELECT	DELOAC	N	
	ATTRIBUTE_USE_CL	DELETE	DELAUCL	S	
	AUC CONSTRAINT	DELETE	DELAON	N	
	COMPLETE RELATION	DELETE	DELCMPR	S	
	CONSTRAINT_INPUT	DELETE	DELCONI	N	
	DESC TEXT	DELETE	DELTEXT	S	
	ENTITY CLASS	SELECT	DELMTKC	S	
	INHERITED_ATT_USE	SELECT	DELMIGK	S	
	INHERITED_ATT_USE	DELETE	DELIAUC	S	
	KEY_CLASS	SELECT	DELMTKC	S	
	KEY_CLASS	DELETE	DELKC	S	
	KEY_CLASS_MEMBER	SELECT	DELMTKC	S	
	KEY_CLASS_MEMBER	DELETE	DELKCMT	S	
OWNED ATTRIBUTE	DELETE	DELOWAC	S		
VIEW_QUAL_XREF	SELECT	VERVWQU	N		



COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
DROP DATABASE	AUC IS MAPPING	SELECT	VERAIMD	N
	DATA BASE	SELECT	VERDBAS	N
	DATA BASE	DELETE	DELDBS1	S
	DATA BASE AREA	DELETE	DELDBA1	S
	DATA FIELD	DELETE	DELDFL3	S
	DATA FIELD USAGE	SELECT	DELDBDF	N
	DB AREA			
	ASSIGNMENT	DELETE	DELDA1	S
	DB PASSWORD	DELETE	DELPSWD	S
	DF SET LINKAGE	DELETE	DELDSL1	S
	ECRTUD	SELECT	VERDUNI	N
	EC RT MAPPING	SELECT	VERERMD	N
	HORIZONTAL PART	SELECT	VERHORZ	N
	IMS SEGMENT SIZE	DELETE	DELISS1	S
	PSB PCB	DELETE	DELPCB	S
	RC BASED REC SET	SELECT	VERDBRL	N
	RECORD SET	SELECT	VERSTNO	N
	RECORD SET	SELECT	DELDBST	N
	RECORD SET	DELETE	DELRS2	S
	RECORD SET USAGE	SELECT	VERRSUS	N
	RECORD TYPE	DELETE	DELRTY1	S
	RECORD TYPE	SELECT	VERDUNI	N
	RECORD TYPE	SELECT	VERAIMD	N
	RECORD TYPE	SELECT	VERERMD	N
	RECORD TYPE	SELECT	VERAIMD	N
	RECORD TYPE	SELECT	DELDBRT	N
	RP MAIN	SELECT	VERRPM	N
	RP SUBROUTINE	SELECT	VERRPS	N
	SCHEMA NAMES	DELETE	DELSN1	S
	SET TYPE MEMBER	DELETE	DELSTM1	S
DROP DBMS	IISS DBMS	SELECT	VERDBMS	N
	IISS DBMS	DELETE	DELDBMS	N
DROP DESCRIPTION	DESCRIPTION TYPE	SELECT	VERDSCT	N
	DESCRIPTION TYPE	DELETE	DELDSCT	N

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
DROP DOMAIN	ATTRIBUTE_CLASS	SELECT	VERACDT	N
	DATA_ITEM	SELECT	VERDIDT	N
	DESC_TEXT	DELETE	DELTEXT	S
	DOMAIN_CLASS	SELECT	VERDOM	N
	DOMAIN_CLASS	DELETE	DELDOM	N
	DOMAIN_RANGE	DELETE	DRPRNGA	N
	DOMAIN_VALUE	DELETE	DRPVALA	N
	ELEMENTARY_DATA_			
	FIELD	SELECT	VERDFDT	N
	MODULE_PARAMETER	DELETE	DELPARM	N
	SOFTWARE_MODULE	DELETE	DELSMOD	N
	USER_DEF_DATA_			
	TYPE	DELETE	DELDTD	N
	USER_DEF_DATA_			
	TYPE	SELECT	DELDTNO	N
	USER_DEF_DATA_			
	TYPE	SELECT	DOMUSAG	N
	VERIF_MODULE	DELETE	DRPVMOD	N
	VERIF_MODULE	SELECT	RETVMOD	N
	DROP ENTITY	ATTRIBUTE_USE_CL	SELECT	DELOAC
ATTRIBUTE_USE_CL		DELETE	DELAUCL	S
ATTRIBUTE_USE_CL		SELECT	FNDAUC	N
AUC_CONSTRAINT		DELETE	DELAON	N
AUC_IS MAPPING		SELECT	VERTAUC	N
COMPLETE_RELATION		DELETE	DELCMPR	S
COMPLEX_MAPPING_				
PARM		SELECT	VERTAP	N
CONSTRAINT_INPUT		DELETE	DELCONI	N
DESC_TEXT		DELETE	DELTEXT	S
ECRTUD		SELECT	VERECR	N
EC_CONSTRAINT		DELETE	DELECON	N
EC_KEYWORD		DELETE	DELECKW	S
ENTITY_CLASS		SELECT	VERENT	N
ENTITY_CLASS		DELETE	DELEC	S
ENTITY_CLASS		SELECT	DELMTKC	S
ENTITY_NAME		SELECT	VERENT	N
ENTITY_NAME		DELETE	DELECNM	S
HORIZONTAL_PART		SELECT	SELHP	N
INHERITED_ATT_USE		DELETE	DELIAUC	S
INHERITED_ATT_USE		SELECT	DELMIGK	S
KEY_CLASS		SELECT	DELMTKC	S
KEY_CLASS		DELETE	DELKC	S
KEY_CLASS_MEMBER		SELECT	DELMTKC	S
KEY_CLASS_MEMBER		DELETE	DELKCMT	S

<u>COMMAND</u>	<u>TABLE NAME</u>	<u>FUNCTI</u>	<u>MODULE</u>	(S=SQL N=NDML)
	OWNED_ATTRIBUTE	SELECT	FNDOAC	N
	OWNED_ATTRIBUTE	DELETE	DELOWAC	S
	PROJECT_DATA_ITEM	SELECT	VERTPDI	N
	RC_BASED_REC_SET	SELECT	VERRCB	N
	RC_KEYWORD	DELETE	DELRCCKW	S
	RELATION_CLASS	SELECT	DRPRCE	N
	RELATION_CLASS	DELETE	DELRC	S
	VIEW_QUAL_XREF	SELECT	VERVWQU	N
	VIEW_RC_COMPONENT	SELECT	VERSRC	N
DROP FIELD	AUC_ST_MAPPING	SELECT	FNDASA	N
	DATA_BASE	SELECT	VERDBAS	N
	DATA_FIELD	SELECT	RETRFLD	S
	DATA_FIELD	UPDATE	MODFLD	S
	DATA_FIELD	DELETE	DELDFL3	S
	DESC_TEXT	DELETE	DELTEXT	S
	DF_PARM	SELECT	VERDFPA	N
	DF_SET_LINKAGE	SELECT	VERDSL3	N
	DF_SET_LINKAGE	DELETE	DELDSL3	S
	ECRTUD	SELECT	VERMUNI	N
	PROJECT_DATA_			
	FIELD	SELECT	VERMPDF	N
	RC_BASED_REC_SET	SELECT	VERRCBS	N
	RECORD_SET	DELETE	DELIRST3	S
	SET_TYPE_MEMBER	DELETE	DELSTM3	S
DROP HOST	IISS_HOST	DELETE	DELHST	N
	IISS_HOST	SELECT	VERHST	N
DROP KEYWORD	AC_KEYWORD	DELETE	DELKWAC	S
	DESC_TEXT	DELETE	DELTEXT	S
	EC_KEYWORD	DELETE	DELKWEC	S
	IISS_KEYWORD	SELECT	VERKW	N
	IISS_KEYWORD	DELETE	DELKW	S
	RC_KEYWORD	DELETE	DELKWRC	S

DS 620341100  
30 September 1990

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
DROP MAP	ATTRIBUTE_USE_CL	SELECT	DRPALTG	N
	ATTRIBUTE_USE_CL	SELECT	VERAUC	N
	AUC_IS_MAPPING	SELECT	VERAISM	N
	AUC_IS_MAPPING	SELECT	VERRPDB	N
	AUC_IS_MAPPING	DELETE	DELAISM	N
	AUC_IS_MAPPING	SELECT	DRPSTMP	N
	AUC_IS_MAPPING	SELECT	DRPDFMP	N
	AUC_IS_MAPPING	SELECT	SELAIMP	N
	AUC_IS_MAPPING	SELECT	DRPALTG	N
	AUC_IS_MAPPING	DELETE	DEL1AIM	N
	AUC_IS_MAPPING	DELETE	DEL1PRF	N
	AUC_IS_MAPPING	SELECT	DRPPRF1	N
	AUC_IS_MAPPING	DELETE	DELAISM	N
	AUC_ST_MAPPING	SELECT	DRPSTMP	N
	AUC_ST_MAPPING	DELETE	DELASM	N
	AUC_ST_MAPPING	DELETE	DEL1ASM	N
	DATA BASE	SELECT	SELDBNM	N
	DISTRIBUTED RULES	DELETE	DRPRULE	N
	EC_RT_MAPPING	SELECT	ECRTALL	N
	EC_RT_MAPPING	DELETE	DRPECRT	N
	EC_RT_MAPPING	DELETE	DEL1ERT	N
	ENTITY_CLASS	SELECT	VERENT	N
	ENTITY_NAME	SELECT	VERENT	N
	ENTITY_NAME	SELECT	VERENT	N
	PROJECT_DATA_			
	FIELD	DELETE	DELPDF	N
	PROJECT_DATA_			
	FIELD	DELETE	DEL1PDF	N
	PROJECT_DATA_			
	FIELD	SELECT	DRPDFMP	N
	RC_BASED_REC_SET	DELETE	DELRCST	S
	RC_BASED_REC_SET	SELECT	VERRCST	N
	RC_BASED_REC_SET	SELECT	VERRCDB	N
	RECORD_TYPE	SELECT	VERRPDB	N
	RELATION_CLASS	SELECT	VERRC	N
	RP_MAIN	SELECT	VERRPM	N
	RP_SUBROUTINE	SELECT	VERRPS	N

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
DROP MODEL	AC_KEYWORD	DELETE	DELACKW	S
	ATTRIBUTE_CLASS	SELECT	FNDACM	N
	ATTRIBUTE_CLASS	DELETE	DELAC	S
	ATTRIBUTE_NAME	SELECT	SELACNM	N
	ATTRIBUTE_NAME	DELETE	DELACNM	S
	ATTRIBUTE_USE_CL	SELECT	DLMDAUC	N
	ATTRIBUTE_USE_CL	DELETE	DELAUCL	S
	AUC_CONSTRAINT	DELETE	DELAON	N
	COMPLETE_RELATION	DELETE	DELCMPR	S
	CONSTRAINT_INPUT	DELETE	DELCONI	N
	DESC_TEXT	DELETE	DELTEXT	S
	EC_CONSTRAINT	DELETE	DELECON	N
	EC_KEYWORD	DELETE	DELECKW	S
	ENTITY_CLASS	SELECT	FNDECM	N
	ENTITY_CLASS	DELETE	DELEC	S
	ENTITY_NAME	SELECT	SELECNM	N
	ENTITY_NAME	DELETE	DELECNM	S
	INHERITED_ATT_USE	DELETE	DELIAUK	S
	KEY_CLASS	SELECT	DELMDKC	N
	KEY_CLASS	DELETE	DELKC	S
	KEY_CLASS_MEMBER	DELETE	DELKCMT	S
	MODEL_CLASS	SELECT	VERMOD	N
	MODEL_CLASS	DELETE	DELMOD	S
	OWNED_ATTRIBUTE	DELETE	DELOACE	S
	RC_KEYWORD	DELETE	DELRCWK	S
	RELATION_CLASS	SELECT	DELMDRC	N
	RELATION_CLASS	DELETE	DELRC	S
	VIEW_QUAL_XREF	SELECT	VERVWQU	N
DROP MODULE	MODULE_PARAMETER	DELETE	DELPARM	N
	SOFTWARE_MODULE	SELECT	VERSMOD	N
	SOFTWARE_MODULE	DELETE	DELSMOD	N
DROP PARTITION	ENTITY_CLASS	SELECT	VERECN	N
	ENTITY_NAME	SELECT	VERECN	N
	HORIZONTAL_PART	SELECT	VERHPN	N
	HORIZONTAL_PART	DELETE	DELHPNA	N
	MODEL_CLASS	SELECT	VERECN	N

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
DROP PSB	IISS_PSB	SELECT	VERPSB	N
	IISS_PSB	DELETE	DELPSB	N
DROP RECORD	AUC_IS_MAPPING	SELECT	VERAIM	N
	AUC_ST_MAPPING	SELECT	FNDASA	N
	DATA_BASE	SELECT	VERDBAS	N
	DATA_FIELD	SELECT	DELDFL2	S
	DESC_TEXT	DELETE	DELTEXT	S
	DF_SET_LINKAGE	DELETE	DELDSL2	S
	ECRTUD	SELECT	SELURT	N
	HORIZONTAL PART	SELECT	VERHORZ	N
	RC BASED REC_SET	SELECT	VERRCBS	N
	RECORD_SET	SELECT	SELRSET	N
	RECORD_SET	DELETE	DELIRST2	S
	RECORD_SET USAGE	SELECT	VERRSUS	N
	RECORD_TYPE	SELECT	VERRT	N
	RECORD_TYPE	DELETE	DELIRST2	S
	SET_TYPE_MEMBER	SELECT	SELSTM	N
DROP RELATION	ATTRIBUTE USE CL	DELETE	DELAUCL	S
	AUC_CONSTRAINT	DELETE	DELAON	N
	AUC_IS_MAPPING	SELECT	VERTAUC	N
	COMPLETE_RELATION	SELECT	VERRCC	N
	COMPLETE_RELATION	DELETE	DELCPRC	S
	COMPLEX_MAPPING_			
	PARM	SELECT	VERTAP	N
	CONSTRAINT INPUT	DELETE	DELCONI	N
	ENTITY_CLASS	SELECT	VERENT	N
	ENTITY_NAME	SELECT	VERENT	N
	INHERITED_ATT_USE	SELECT	DELMIGRC	S
	INHERITED_ATT_USE	DELETE	DELIAUC	S
	KEY_CLASS_MEMBER	SELECT	DRPMGRC	N
	KEY_CLASS_MEMBER	DELETE	DELKCMT	S
	PROJECT_DATA_ITEM	SELECT	VERTPDI	N
	RC_BASED PEC_SET	SELECT	VERRCB	N
	RC_KEYWORD	DELETE	DELRCWK	S
	RELATION_CLASS	SELECT	VERRC	N
	RELATION_CLASS	DELETE	DELRC	S
	VIEW_QUAL_XREF	SELECT	VERVWQU	N
	VIEW_RC_COMPONENT	SELECT	VERSRC	N

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
DROP SET	AUC_ST_MAPPING	SELECT	FNDASA	N
	DATA_BASE	SELECT	VERDBAS	N
	DF_SET_LINKAGE	DELETE	DELDSL2	S
	RC_BASED_REC-SET	SELECT	VERRCBS	N
	RECORD_SET	SELECT	VERRSET	N
	RECORD_SET	DELETE	DELIRST2	S
	RECORD_SET_USAGE	SELECT	VERRSUS	N
	SET_TYPE_MEMBER	DELETE	DELSTM2	S
DROP UNION	DATA_BASE	SELECT	VERDB	N
	ECRTUD	SELECT	SELURT	N
	ECRTUD	DELETE	DELURT	N
	RECORD_TYPE	SELECT	VERRT	N
DROP VIEW	DATA_ITEM	SELECT	DRPDIV	N
	DATA_ITEM	DELETE	DELDIV	N
	DI_PARM	SELECT	VERDIPA	N
	PROJECT_DATA_ITEM	DELETE	DELPDI	N
	USER_VIEW	SELECT	VERVIEW	N
	USER_VIEW	DELETE	DELVIEW	N
	VIEW_EC_XREF	DELETE	DELECRF	N
	VIEW_QUALIFY_			
	CRITERIA	DELETE	DELQCRF	N
VIEW_QUAL_XREF	DELETE	DELQCTG	N	
MERGE MODEL	ATTRIBUTE_CLASS	SELECT	CMBOA	N
	ATTRIBUTE_CLASS	SELECT	VERATT	N
	ATTRIBUTE_NAME	SELECT	CMBOA	N
	ATTRIBUTE_NAME	SELECT	VERATT	N
	ATTRIBUTE_USE_CL	SELECT	BLKCLST	N
	DOMAIN_CLASS	SELECT	CMBOA	N
	ENTITY_CLASS	SELECT	MRGNODE	S
	ENTITY_CLASS	SELECT	BLKCLST	N
	ENTITY_CLASS	SELECT	VERENT	N
	ENTITY_NAME	SELECT	MRGMOD2	S
	ENTITY_NAME	SELECT	VERENT	N
	ENTITY_NAME	SELECT	SELECNM	N
	MODEL_CLASS	SELECT	MRGNODE	S
	MODEL_CLASS	SELECT	VERMOD	N
	MODEL_CLASS	SELECT	MRGMOD2	S
	OWNED_ATTRIBUTE	SELECT	CMBOA	N
	RELATION_CLASS	SELECT	MRGNODE	S
	RELATION_CLASS	SELECT	MRGMOD2	S

COMMAND	TABLE NAME	FUNCTI	MODULE	(S=SQL N=NDML)
RENAME	AC_KEYWORD	UPDATE	MODACKW	N
	APPLICATION	UPDATE	UPDAPP	S
	ATTRIBUTE_CLASS	SELECT	VERATT	N
	ATTRIBUTE_NAME	SELECT	VERATT	N
	ATTRIBUTE_NAME	MODIFY	UPDACNM	S
	AUC_IS_MAPPING	UPDATE	UPDRAUC	S
	AUC_ST_MAPPING	UPDATE	UPDAUC	S
	DATA_BASE	SELECT	VERSTID	N
	DATA_BASE	SELECT	VERDB	N
	DATA_BASE	UPDATE	UPDDBN	S
	DATA_BASE	UPDATE	UPDDB	S
	DATA_BASE	SELECT	VERDFID	N
	DATA_BASE	SELECT	VERRTID	N
	DATA_FIELD	SELECT	VERDFID	N
	DATA_FIELD	UPDATE	UPDFDF	S
	DATA_FIELD	UPDATE	UPDDF	S
	DATA_ITEM	SELECT	VERDIID	N
	DATA_ITEM	UPDATE	UPDDI	S
	DATA_ITEM	UPDATE	UPDDDI	S
	DBMS_ON_HOST	UPDATE	UPDDBH	S
	DB_AREA			
	ASSIGNMENT	UPDATE	UPDDAA	S
	DF_SET_LINKAGE	UPDATE	UPDDFU	S
	DF_SET_LINKAGE	UPDATE	UPDFDFU	S
	DF_SET_LINKAGE	UPDATE	UPDRDFU	S
	DOMAIN_CLASS	SELECT	VERDOM	N
	DOMAIN_CLASS	MODIFY	UPDTDOM	S
	EC_KEYWORD	UPDATE	MODECKW	N
	ENTITY_CLASS	SELECT	VERENT	N
	ENTITY_NAME	SELECT	VERENT	N
	ENTITY_NAME	MODIFY	UPDECNM	S
	IISS_HOST	SELECT	VERHST	N
	IISS_HOST	UPDATE	UPDHST	S
	IISS_KEYWORD	SELECT	VERKW	N
	IISS_KEYWORD	MODIFY	UPDTKW	S
	IISS_PSB	UPDATE	UPDPSB	S
	MODEL_CLASS	SELECT	VERMOD	N
	MODULE_PARAMETER	UPDATE	UPDMP	S
	PROJECT_DATA_			
	FIELD	UPDATE	UPDPDF	S
	PROJECT_DATA_			
	FIELD	UPDATE	UPDFPDF	S
	PROJECT_DATA_ITEM	UPDATE	UPDPDI	S
	RC_BASED_REC_SET	UPDATE	UPDRCB	S
	RC_BASED_REC_SET	UPDATE	UPDRCB	S



COMMAND	TABLE NAME	FUNCTI	MODULE	(SE=SQL N=NDML)
RENAME (continued)				
	RC_KEYWORD	UPDATE	MODRCKW	N
	RECORD_SET	SELECT	VERSTID	N
	RECORD_SET	UPDATE	UPDRRS	S
	RECORD_SET	UPDATE	UPDRSS	S
	RECORD_TYPE	SELECT	VERRTID	N
	RECORD_TYPE	UPDATE	UPDRT	S
	RELATION_CLASS	SELECT	VERRC	N
	RELATION_CLASS	MODIFY	UPDRCNM	S
	SET_TYPE_MEMBER	UPDATE	UPDSTM	S
	SET_TYPE_MEMBER	UPDATE	UPDRSTM	S
	USER_DEF_DATA_			
	TYPE	SELECT	VERDTN	N
	USER_DEF_DATA_			
	TYPE	UPDATE	UPDUDT	S
	USER_VIEW	SELECT	VERDIID	N
	USER_VIEW	MODIFY	UPDVIEW	S
	USER_VIEW	SELECT	VERVIEW	N

APPENDIX C

INTERNAL TABLES USED BY NDDL COMMANDS

INTERNAL TABLE NAME	TABLE SIZE	WHY USED	USED BY
ADDTGLS	25	Max # of attributes to be added for a specific key class of a specific entity	Alter Entity
ALGTBL	25	Max # of parameters for a complex mapping algorithm	Define Algorithm
AUPTBL	25	Max # of attributes that are input or output parameters for an algorithm	Copy Map
BOOLST	100	Contains the Boolean operators, parenthesis and pointers to type 2 conditions for an NDML transaction	Create View
CSQCLST	50	Contains entity and tag number for each attribute on the WHERE clause	Create View
DBTBL	25	Max # of databases in the CDM	Copy Database

INTERNAL TABLE NAME	TABLE SIZE	WHY USED	USED BY
DEPECL	25	Max # of entities in the "except" clause	Copy Entity Copy Model
DEPTBL	25	Max # of datafields that are input or output parameters for an algorithm	Copy Map
DFTBL	256	Max # of fields in a record	Define Record, Alter Record, Alter Field, Drop Field
DFPTBL	25	Contains database, record, and data field numbers for Copy Map	Copy Map
DRPTGLS	25	Max # of attributes from a specific key class of a specific entity	Alter Entity
ECLIST	200	Max # of entities in a model	Copy Model, Merge Model, Copy Entity
ECRTBL	100	Contains entities used in entity to record mappings	Copy Map
ECTBL	25	Max # of entity mappings to be copied	Copy Map Copy View
HPTBL	25	Max # of entities that have been horizontally partitioned against a specific record	Copy Record

INTERNAL TABLE NAME	TABLE SIZE	WHY USED	USED BY
KCLIST	20	Max # of keys for a specific entity	Copy Entity
KCNAMET	25	Max # of keys that have been processed for a specific entity	Alter Attribute
KEYLIST	5	Max # of keys for an entity	Copy Model, Merge Model, Copy Entity
KWDTBL	100	Max # of entity keywords in both models	Compare Model
		Max # of attribute keywords in both models	Compare Model
LISTREL	25	Max # of dependent relations per entity	Combine Entity
		Max # of independent relations per entity	Combine Entity
MACDAT	1	Contains WS Variables for Macro Copy Utility	Create Domain
MDTBL	25	Max # of software modules that are used a complex mapping algorithm	Copy Map

INTERNAL TABLE NAME	TABLE SIZE	WHY USED	USED BY
MODTBL	25	Max # of parameters per software module	Define Module, Alter Module
NEWTAGT	25	Contains key migration information of the attribute use class in the new owner entity	Alter Attribute
OLDTAGT	25	Two-dimensional table contains key migration information of the attribute use class in the old owner entity	Alter Attribute
	10	Contains new key class information	
RCDEPKC		Two-Dimensional Table	
	400	Max # of relations in one model	Copy Model, Merge Model, Copy Entity, Combine Entity
	5	Max # of keys per entity	
RCTBL	25	Contains relation class information	Copy Map Copy View
RELTBL	25	Max # of relations where the entity is the dependent entity	Combine Entity, Copy Entity
RENLIST	10	Max # of tags mi- grated per relation mentioned in the Set...clause	Create Relation, Alter Relation

INTERNAL TABLE NAME	TABLE SIZE	WHY USED	USED BY
RTPTBL	25	Max # of records that the input or output parameter to an algorithm	Copy Map
SBSTLST	8	Represents the input table of substitution parameters for Macro expansion routine	Create Domain
SDLIST	25	Contains information needed to validate the legality of the view structure	Create View
TAGTBL	25	Max # of tags per complex mapping algorithm	Drop Algorithm
TDFTBL	256	Max # of fields in a record	Define Record Alter Record, Alter Field, Drop Field
UNTBL	25	Max # of entities that participate in a record union	Copy Map
UPTGLS	25	Contains attribute use class information for updating the inherited attribute use class	Alter Entity
VWDI	25	Max # of data items per view	Create View
VWFROM	25	Max # of entities in From clause per view	Create View

DS 620341100  
30 September 1990

INTERNAL TABLE NAME	TABLE SIZE	WHY USED	USED BY
VWRC	25	Max # of relations in Where clause per view	Create View
VWRETR	25	Max # of tags in Select clause per view	Create View