

AD-A181 577

INTEGRATED INFORMATION SUPPORT SYSTEM (IIS) VOLUME 5

1/4

COMMON DATA MODEL 5 (U) GENERAL ELECTRIC CO

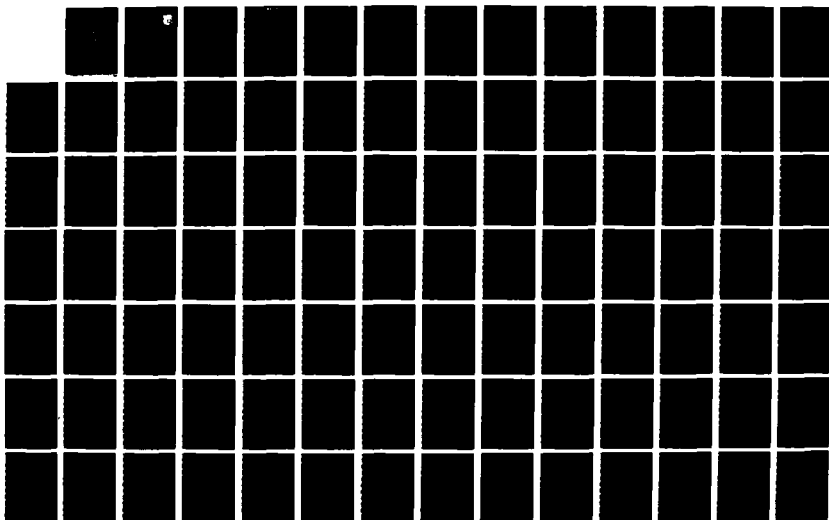
SCHENECTADY NY PRODUCTION RESOURCES CONSU

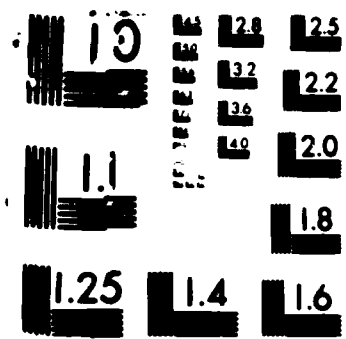
UNCLASSIFIED

D ROLLINS ET AL 81 NOV 85 U--628141881

F/G 5/2

NL





DTIC FILE COPY

2

**AFWAL-TR-86-4006
Volume V
Part 1**



**INTEGRATED INFORMATION
SUPPORT SYSTEM (IISS)
Volume V - Common Data Model Subsystem
Part 1 - CDM Administrator's Manual**

AD-A181 577

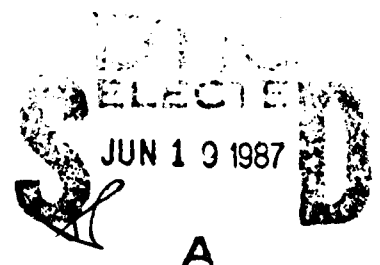
**General Electric Company
Production Resources Consulting
One River Road
Schenectady, New York 12345**

**Final Report for Period 22 September 1980 - 31 July 1985
November 1985**

Approved for public release; distribution is unlimited.

PREPARED FOR:

**MATERIALS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AFB, OH 45433-6533**

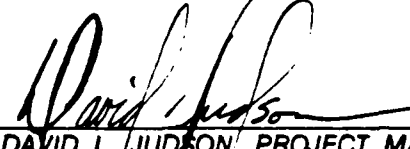


NOTICE

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; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Office of Public Affairs (ASD/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

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



DAVID L. JUDSON, PROJECT MANAGER
AFWAL/MLTC
WRIGHT PATTERSON AFB OH 45433

5 Aug 1986

DATE

FOR THE COMMANDER:



GERALD C. SHUMAKER, BRANCH CHIEF
AFWAL/MLTC
WRIGHT PATTERSON AFB OH 45433

7 Aug 86

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 AFWAL/MLTC, WPAFB, OH 45433 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

AD-A181577
REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS									
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)		5. MONITORING ORGANIZATION REPORT NUMBER(S) AFVAL-TR-86-4006 Vol V, Part 1									
6a. NAME OF PERFORMING ORGANIZATION General Electric Company Production Resources Consulting	6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION AFVAL/MLTC									
6c. ADDRESS (City, State and ZIP Code) 1 River Road Schenectady, NY 12345		7b. ADDRESS (City, State and ZIP Code) WPAFB, OH 45433-6533									
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Materials Laboratory Air Force Systems Command, USAF	8b. OFFICE SYMBOL (If applicable) AFVAL/MLTC	8. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-80-C-5155									
9a. ADDRESS (City, State and ZIP Code) Wright-Patterson AFB, Ohio 45433		10. SOURCE OF FUNDING NOS. <table border="1"><thead><tr><th>PROGRAM ELEMENT NO.</th><th>PROJECT NO.</th><th>TASK NO.</th><th>WORK UNIT NO.</th></tr></thead><tbody><tr><td>78011F</td><td>7500</td><td>62</td><td>01</td></tr></tbody></table>		PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.	78011F	7500	62	01
PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.								
78011F	7500	62	01								
11. TITLE (Include Security Classification) (See Reverse)											
12. PERSONAL AUTHOR(S) Rollins, D. Loomis, M. Hogan, J. Leifeste, B.											
13a. TYPE OF REPORT Final Technical Report	13b. TIME COVERED 22 Sept 1980 - 31 July 1985	14. DATE OF REPORT (Yr., Mo., Day) 1985 November	15. PAGE COUNT 302								
16. SUPPLEMENTARY NOTATION ICAM Project Priority 6201		The computer software contained herein are theoretical and/or references that in no way reflect Air Force-owned or -developed computer software.									
17. COSATI CODES <table border="1"><thead><tr><th>FIELD</th><th>GROUP</th><th>SUB GR</th></tr></thead><tbody><tr><td>1308</td><td>0905</td><td></td></tr></tbody></table>		FIELD	GROUP	SUB GR	1308	0905		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)			
FIELD	GROUP	SUB GR									
1308	0905										
19. ABSTRACT (Continue on reverse if necessary and identify by block number) The Common Data Model Processor (CDMP) is a mechanism by which application programs can retrieve and update data without knowing where or how the data are stored. The CDM (Common Data Model) is a database where schemas and mappings for data access are stored. This is a user manual for the CDM Administrator. It describes the philosophical and practical objectives of the CDM Administrator, the design of the CDM, and the steps needed to enter and maintain data in the CDM.											
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐		21. ABSTRACT SECURITY CLASSIFICATION Unclassified									
22a. NAME OF RESPONSIBLE INDIVIDUAL David L. Judson		22b. TELEPHONE NUMBER (Include Area Code) 813-255-8976	22c. OFFICE SYMBOL AFVAL/MLTC								

11. Title

Integrated Information Support System (IISS)
Vol V - Common Data Model Subsystem
Part 1 - CDM Administrator's Manual



Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	
Justification	
By _____	
Distribution/	
Availability Code	
Dist	Availability
	Special

A-1

PREFACE

This user's manual covers the work performed under Air Force Contract F33615-80-C-5155 (ICAM Project 6201). This contract is sponsored by the Materials Laboratory, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Gerald C. Shumaker, ICAM Program Manager, Manufacturing Technology Division, through Project Manager, Mr. David Judson. The Prime Contractor was Production Resources Consulting of the General Electric Company, Schenectady, New York, under the direction of Mr. Alan Rubenstein. The General Electric Project Manager was Mr. Myron Hurlbut of Industrial Automation Systems Department, Albany, New York.

Certain work aimed at improving Test Bed Technology has been performed by other contracts with Project 6201 performing integrating functions. This work consisted of enhancements to Test Bed software and establishment and operation of Test Bed hardware and communications for developers and other users. Documentation relating to the Test Bed from all of these contractors and projects have been integrated under Project 6201 for publication and treatment as an integrated set of documents. The particular contributors to each document are noted on the Report Documentation Page (DD1473). A listing and description of the entire project documentation system and how they are related is contained in document FTR620100001, Project Overview.

The subcontractors and their contributing activities were as follows:

TASK 4.2

<u>Subcontractors</u>	<u>Role</u>
Boeing Military Aircraft Company (BMAC)	Reviewer.
D. Appleton Company (DAGOM)	Responsible for IDEF support, state-of-the-art literature search.
General Dynamics/ Ft. Worth	Responsible for factory view function and information models.

Subcontractors

Role

Illinois Institute of
Technology

Responsible for factory view
function research (IITRI)
and information models of
small and medium-size business.

North American Rockwell

Reviewer.

Northrop Corporation

Responsible for factory view
function and information
models.

Pritsker and Associates

Responsible for IDEF2 support.

SofTech

Responsible for IDEFO support.

TASKS 4.3 - 4.9 (TEST BED)

Subcontractors

Role

Boeing Military Aircraft
Company (BMAC)

Responsible for consultation on
applications of the technology
and on IBM computer technology.

Computer Technology
Associates (CTA)

Assisted in the areas of
communications systems, system
design and integration
methodology, and design of the
Network Transaction Manager.

Control Data Corporation
(CDC)

Responsible for the Common Data
Model (CDM) implementation and
part of the CDM design (shared
with DACOM).

D. Appleton Company
(DACOM)

Responsible for the overall CDM
Subsystem design integration
and test plan, as well as part
of the design of the CDM
(shared with CDC). DACOM also
developed the Integration
Methodology and did the schema
mappings for the Application
Subsystems.

UM 620141001
1 November 1985

<u>Subcontractors</u>	<u>Role</u>
Digital Equipment Corporation (DEC)	Consulting and support of the performance testing and on DEC software and computer systems operation.
McDonnell Douglas Automation Company (McAuto)	Responsible for the support and enhancements to the Network Transaction Manager Subsystem during 1984/1985 period.
On-Line Software International (OSI)	Responsible for programming the Communications Subsystem on the IBM and for consulting on the IBM.
Rath and Strong Systems Products (RSSP) (In 1985 became McCormack & Dodge)	Responsible for assistance in the implementation and use of the MRP II package (PIOS) that they supplied.
SofTech, Inc.	Responsible for the design and implementation of the Network Transaction Manager (NTM) in 1981/1984 period.
Software Performance Engineering (SPE)	Responsible for directing the work on performance evaluation and analysis.
Structural Dynamics Research Corporation (SDRC)	Responsible for the User Interface and Virtual Terminal Interface Subsystems.

Other prime contractors under other projects who have contributed to Test Bed Technology, their contributing activities and responsible projects are as follows:

<u>Contractors</u>	<u>ICAM Project</u>	<u>Contributing Activities</u>
Boeing Military Aircraft Company (BMAC)	1701, 2201, 2202	Enhancements for IBM node use. Technology Transfer to Integrated Sheet Metal Center (ISMC).

UM 620141001
1 November 1985

<u>Contractors</u>	<u>ICAM Project</u>	<u>Contributing Activities</u>
Control Data Corporation (CDC)	1502, 1701	IISS enhancements to Common Data Model Processor (CDMP).
D. Appleton Company (DACOM)	1502	IISS enhancements to Integration Methodology.
General Electric	1502	Operation of the Test Bed and communications equipment.
Hughes Aircraft Company (HAC)	1701	Test Bed enhancements.
Structural Dynamics Research Corporation (SDRC)	1502, 1701, 1703	IISS enhancements to User Interface/Virtual Terminal Interface (UI/VTI).
Systran	1502	Test Bed enhancements. Operation of Test Bed.

TABLE OF CONTENTS

	<u>Page</u>
SECTION 1.0	INTRODUCTION 1-1
1.1	Managing Data as a Corporate Resource 1-1
SECTION 2.0	CDM OVERVIEW 2-1
2.1	The Fundamental Approach 2-1
2.1.1	The Three Schema-Architecture 2-1
2.1.2	Representation of the Three Types of Schemas 2-5
2.1.3	Integration Methodology 2-6
2.1.4	Contributions to IRRIASSPA 2-9
2.2	Basic Components of the Design 2-10
2.2.1	The CDM Database 2-10
2.2.2	CDM1 2-11
2.2.3	The CDM Processor 2-11
SECTION 3.0	RESPONSIBILITIES OF THE CDM ADMINISTRATOR 3-1
3.1	Establishing Data Standards 3-1
3.2	Maintaining the CDM 3-1
3.3	Protecting the CDM 3-1
3.4	Facilitating Use of the CDM 3-2
SECTION 4.0	MAINTAINING THE CONCEPTUAL SCHEMA 4-1
4.1	Methodology Overview 4-1
4.1.1	CS Structure 4-1
4.1.2	Basic Approach 4-4
4.1.3	Modeling Forms 4-4
4.2	Building the Initial CS 4-16
4.2.1	Phase 0: Starting the Project 4-16
4.2.2	Phase 1: Defining Entity Classes .. 4-20
4.2.3	Phase 2: Defining Relation Classes 4-22
4.2.4	Phase 3: Defining Key Classes 4-25
4.2.5	Phase 4: Defining Nonkey Attribute Classes 4-33
4.3	Expanding the CS 4-35
4.3.1	Phase 0: Starting the Project 4-36
4.3.2	Phase 1: Defining Entity Classes .. 4-38
4.3.3	Phase 2: Defining Relation Classes 4-40
4.3.4	Phase 3: Defining Key Classes 4-42

TABLE OF CONTENTS (Continued)

	<u>Page</u>
4.3.5	Phase 4: Defining Nonkey Attribute Classes 4-54
SECTION 5.0	MAINTAINING THE CDM 5-1
5.1	Methodology Overview 5-1
5.1.1	Using NDDL with the CDM Tables 5-1
5.1.2	Direct Loading of the CDM Tables .. 5-3
5.2	Loading the Initial CS Description .. 5-24
5.2.1	Direct Loading of the CDM Tables 5-24
5.2.2	Loading the CS with the NDDL 5-25
5.3	Modifying/Deleting CS Elements 5-26
5.3.1	Entity Class Changes 5-29
5.3.2	Attribute Class Changes 5-30
5.3.3	Attribute Use Class Changes 5-32
5.3.4	Key Class Changes..... 5-33
5.3.5	Key Class Member Changes 5-34
5.3.6	Relation Class Changes 5-34
5.3.7	Inherited Key Class Changes 5-36
5.3.8	Inherited Attribute Class Changes 5-37
5.3.9	Summary 5-38
5.4	Updating the CS Tables in the CDM ... 5-38
5.4.1	Direct Updating of the CS CDM Tables 5-39
5.4.2	Updating the CS CDM Tables with the NDDL 5-46
SECTION 6.0	MAINTAINING INTERNAL SCHEMAS AND MAPPINGS 6-1
6.1	Methodology Overview 6-1
6.1.1	IS and CS-IS Mapping Structure 6-1
6.1.2	Basic Approach 6-2
6.1.3	IS Modeling Forms 6-16
6.1.4	NDDL Commands for Internal Schema 6-25
6.1.5	Loading the CDM Tables 6-30
6.2	Modifying/Deleting IS Elements and CS-IS Mappings 6-40
6.2.1	Database Changes 6-43
6.2.2	Database Area Changes 6-46
6.2.3	Record Type Changes 6-47

TABLE OF CONTENTS (Continued)

	<u>Page</u>
6.2.4	Database Area Assignment Changes .. 6-51
6.2.5	Data Field Changes 6-53
6.2.6	Record Set Changes 6-60
6.2.7	Record Set Member Changes 6-62
6.2.8	Summary 6-64
6.3	CODASYL Databases 6-65
6.3.1	CODASYL-Specific Considerations ... 6-65
6.3.2	Building a CODASYL IS and CS-IS Mapping 6-67
6.3.3	Building a CODASYL IS and CS-IS Mapping with NDDL 6-74
6.3.4	Modifying a CODASYL IS and CS-IS Mapping Objective 6-75
6.4	Relational Databases 6-80
6.4.1	Relational-Specific Considerations 6-80
6.4.2	Building a Relational Table IS and CS-IS Mapping 6-84
6.4.3	Building a Relational Table IS and CS-IS Mapping with NDDL 6-86
6.4.4	Modifying a Relational Table IS and CS-IS Mapping 6-87
6.4.5	Modifying a Relational Table IS and CS-IS Mapping with NDDL 6-90
6.5	IMS Databases 6-91
6.5.1	IMS Specific Considerations 6-91
6.5.2	Building an IMS IS and CS-IS Mapping 6-94
6.5.3	Building an IMS IS and CS-IS Mapping with NDDL 6-99
6.5.4	Modifying an IMS IS and CS-IS Mapping 6-100
6.5.5	Modifying an IMS IS and CS-IS Mapping with NDDL 6-109
6.6	VSAM Files 6-111
6.6.1	VSAM-Specific Considerations 6-111
6.6.2	Building a VSAM IS and CS-IS Mapping 6-112
6.6.3	Modifying a VSAM IS and CS-IS Mapping 6-114
6.7	Sequential Files (Flat Files) 6-117

TABLE OF CONTENTS (Continued)

		<u>Page</u>
6.7.1	Sequential-Specific Considerations	6-117
6.7.2	Building a Sequential File IS and CS-IS Mapping	6-117
6.7.3	Modifying a Sequential File IS and CS-IS Mapping	6-120
SECTION 7.0	MAINTAINING EXTERNAL SCHEMAS	
	MAPPINGS	7-1
7.1	Methodology Overview	7-1
7.1.1	ES and CS-ES Mapping Structure	7-1
7.1.2	Basic Approach	7-1
7.1.3	Modeling Forms	7-11
7.1.4	CDM Tables and ES NDDL	7-12
7.2	Building an ES and CS-ES Mapping	7-14
7.3	Modifying/Deleting ES Elements and CS-ES Mappings	7-16
7.3.1	User View (ES) Changes	7-18
7.3.2	Data Item Changes	7-19
7.3.3	Summary	7-20
APPENDIX A	GLOSSARY	A-1
APPENDIX B	REFERENCES	B-1

LIST OF ILLUSTRATIONS

<u>Figure</u>	<u>Title</u>	<u>Page</u>
1-1	Data as an Integral Part of the Decision-Making Process	1-3
2-1	Two Fundamentally Different Views of Data: Logical and Physical ..	2-3
2-2	Direct Mapping of Logical and Physical Views	2-3
2-3	The Three-Schema Architecture	2-5
4-1	Relation Classes Form	4-7

LIST OF ILLUSTRATIONS (Continued)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
4-2	Relation Classes Form Example	4-8
4-3	Owned Attribute Classes Form	4-11
4-4	Owned Attribute Classes Form Example	4-12
4-5	Inherited Attribute Classes Form	4-14
4-6	Inherited Attribute Classes Form Example	4-15
4-7	Refinements of Nonspecific Relation Classes Example	4-57
4-8	Triads and Other Dual-Path Structure Example	4-58
4-9	Migration Through Two Relation Classes Example	4-59
4-10	Guidelines for Determining Key Classes of Dependent Entity Classes ..	4-60
5-1	NDDL Commands	5-2
5-2	Owned Attribute Classes Form Example ..	5-5
5-3	Inherited Attribute Classes Form Example	5-8
5-4	Entity Class Glossary Form Example	5-11
5-5	Inherited Attribute Classes Form Example	5-14
5-6	Owned Attribute Classes Form Example ..	5-17
5-7	Inherited Attribute Classes Form Example	5-18
5-8	Owned Attribute Classes Form Example ..	5-19
5-9	Inherited Attribute Classes Form Example	5-20
5-10	Relation Classes Form Example	5-23
5-11	Impact of Conceptual Schema Changes ...	5-28
6-1	Entity Class/Record Type Mapping	6-3
6-2	Join Examples	6-10
6-3	Join Structures	6-11
6-4	Record Type/Entity Class Mapping Form	6-18
6-5	Record Type/Entity Class Mapping Form Example	6-19
6-6	Record Type Join Structures Diagram ...	6-20
6-7	Record Type Join Structures Diagram Example	6-21
6-8	Data Field/Attribute Use Class Mapping	6-23

LIST OF ILLUSTRATIONS (Continued)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
6-9	Data Field/Attribute Use Class Mapping Example	6-24
6-10	Set Type/Relation Class Mapping	6-26
6-11	Set Type/Relation Class Mapping Example	6-27
6-12	Data Field/Attribute Use Class Mapping Example	6-32
6-13	Record Type Join Structure Diagram Example	6-35
6-14	Record Type Entity Class Mapping Example	6-37
6-15	Set Type/Relation Class Mapping Example	6-39
6-16	Impact of Internal Schema Changes	6-42
6-17	Incomplete Join Structure Example	6-73
6-18	Relational Implementation of the Conceptual Model	6-83
7-1	Data Item/Attribute Use Class Mappings	7-3
7-2	ES-CS Join Examples	7-7
7-3	ES-CS Join Structures	7-9
7-4	Impact of External Schema Changes	7-18

SECTION 1

INTRODUCTION

The purposes of this document are several and include:

- a) Describing the philosophical and practical objectives of the Common Data Model (CDM) Administrator;
- b) Discussing the CDM itself, its underlying design, and its role in the IISS environment;
- c) Describing in detail the steps necessary in entering and maintaining data kept in the CDM.

After reading and understanding this document, the CDM Administrator should not only be able to collect, enter, and maintain CDM-related data, but also be able to understand the reasons why such activities are performed.

The NDDL statements used to perform the actual CDM maintenance activities are described in detail in the NDDL User Guide.

1.1 Managing Data as a Corporate Resource

Managing data as a corporate resource is a philosophy about the importance of data to an organization. The approach recognizes that data are assets to be managed along with the other more generally recognized resources of an enterprise, including its personnel, inventories, capital, and so forth. Organizations spend tremendous sums of money collecting and manipulating data, trying to extract information needed to support decision making. The CDM Administrator has as one of his or her primary objectives the preservation of that continuing, substantial investment in data resources. The CDM Administrator plays a major role in protecting and properly managing that investment by managing common data rather than just managing applications that access data.

Data management includes all the activities that ensure that quality data are available to produce needed information and knowledge. The objective of data management is to keep data assets resilient, flexible, and adaptable to supporting decision-making activities in the business. Data management responsibilities include: 1) the representation, storage, and

organization of data so that they can be selectively and efficiently accessed, 2) the manipulation and presentation of data so that they support the user environment effectively, and 3) the protection of data so that they retain their value.

The philosophy of the CDM recognizes that data are absolutely necessary to the decision-making cycles of organizations (Figure 1-1). Individuals must not only be able to collect and retain data for their own use, but also be able to share data and pool their knowledge resources. The ability to correlate information across traditional applications boundaries and to provide information that supports all levels of decision making, from operational through tactical through strategic, is increasingly important as management at all levels is becoming more aware of the potential power of information systems.

The CDM provides the capability to pull the enterprise's database resources together to form an integrated, common source of information to support decision making.

The objectives of data management include the following:

- Independence of data access from data descriptions
- Increased data accessibility
- Improved data integrity
- Improved data shareability
- Improved data resiliency
- Improved data administration and control
- Improved data security
- Improved performance

The CDM Administrator needs to understand each of these objectives.

Independence between data access and data descriptions improves control over the data descriptions, facilitates standardization of data-naming conventions, and reduces the programming effort required to accommodate modified data descriptions. Data independence is perhaps the single most important factor in determining the long-range success of a data-driven environment.

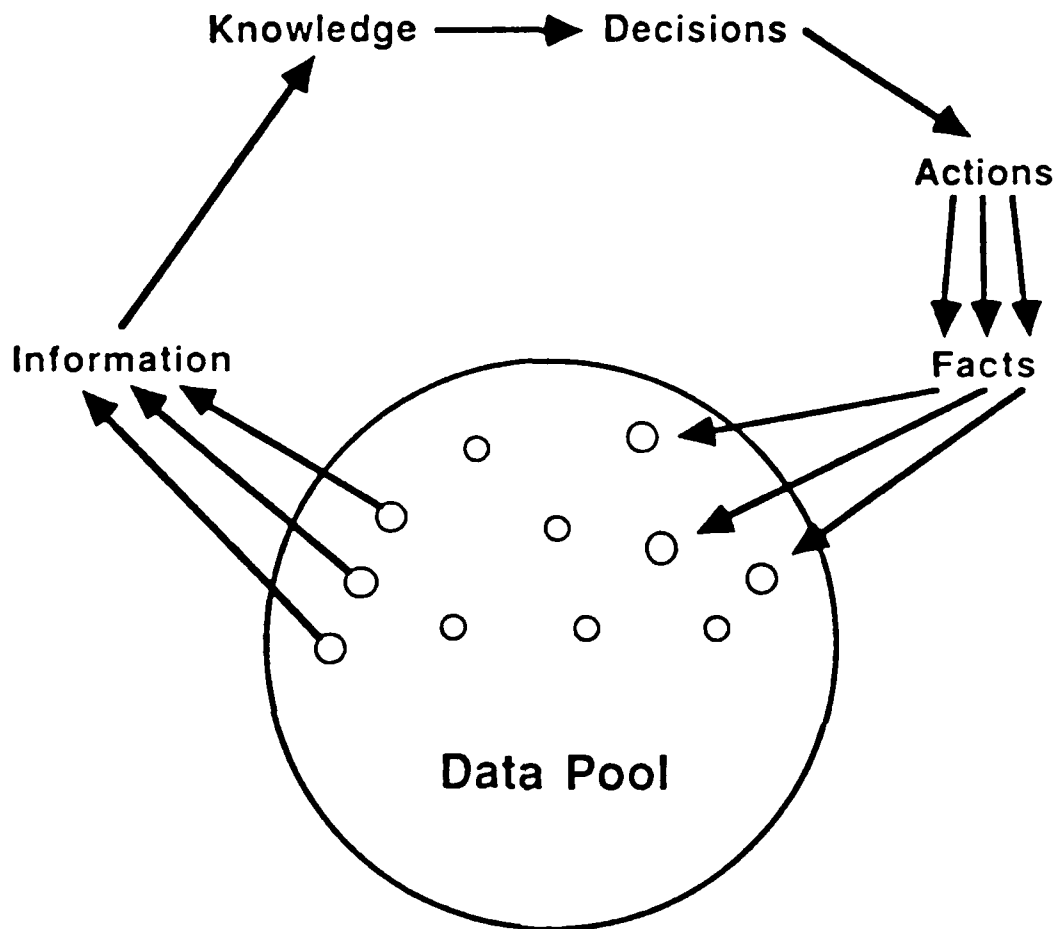


Figure 1-1. Data as an Integral Part of the Decision Making Process

Data accessibility refers to the capability for a user to extract needed information from the data resource. Data accessibility is enhanced by user-friendly interface languages and well designed screens. Good accessibility is characterized by being able to relate data in many different ways to produce information, and by being able to represent that information in a variety of suitable forms. Data accessibility is improved by the CDM in its support of multiple access paths and retrieval sequences through the physical databases. Programming effort for data manipulation is decreased and cost-effective, general-purpose query facilities such as the NDML become possible.

Data integrity is essential to maintain the quality of the

data resource. Data integrity is measured by the completeness and consistency of the data resource. Does it contain the data that are relevant to the decision-making needs of the user? Does it contain all required interrelationships among types of data, and are all consistency constraints satisfied?

Data shareability is needed to keep common data truly common. Without shareability, data proliferate and their quality becomes uncontrollable. Without shareability, data are private and personal; their quality is each individual user's responsibility. The main difficulty with this distribution and redundancy of control is that it results in no control at all. Improved shareability can be achieved by supporting multiple access paths through the physical databases, thereby enabling them to serve many diverse needs. Shareability is also achieved by separating individual user's views of the data resource from the actual physical implementation of databases.

Data shareability refers not just to database contents, but also to logic that accesses and manages data. Reduced data duplication streamlines data access, reduces the programming effort required for updating data, and reduces the potential for inconsistent data. Reduced redundancy in the data management effort improves the productivity of data processing personnel.

Data recoverability is needed to keep the data resource resilient in the wake of errors. Error conditions need to be detected and corrected. Better yet, errors should be prevented from occurring in the first place. Part of the difficulty in providing a resilient data resource is continuing to make the data available to users while recovering from errors.

The CDM Administrator should help to ensure that the data resource continues to satisfy users' information needs, even as those needs change through time. Many organizations have successfully established data administration functions to help develop and protect data assets. The CDM Administrator plays a similar role for the integrated, overall data resource.

Data security is essential to prevent unauthorized access to data. Certainly not all environments require the same, elaborate security schemes, but nearly all organizations' data assets need to have some degree of access protection. Some data are wide open to public retrieve-only access; others require strict authentication to provide retrieval. Many databases have more stringent restrictions on accesses that will change database contents than on accesses that only read database

contents.

Performance of the data resource has two facets: efficiency and effectiveness. Efficiency is a measure of how well the data system utilizes physical computer support, while effectiveness is a measure of how well the data system meets users' information needs. The characteristics are closely related; for example, a user may be totally dissatisfied with the system if response time is measured in hours rather than seconds. Response time is generally considered to be an efficiency measure, but it certainly has an impact on effectiveness.

SECTION 2

CDM OVERVIEW

2.1 The Fundamental Approach

2.1.1 The Three-Schema Architecture

A key to implementing effective data-oriented environments lies in a framework that is called the Three-Schema Architecture. This approach was proposed in the mid-1970s, then developed, and finally published in 1977 in a report from a committee of the American National Standards Institute - "The ANSI/X3/SPARC DBMS Framework: Report of the Study Group on Data Base Management Systems."

The basic concepts proposed in the report have the power to lead us to more effective information resource management. They are implemented in the CDM.

The Three-Schema Architecture is based upon several fundamental facts:

- Computers and users need to be able to view the same data in different ways
- Different users need to be able to view the same data in different ways
- It is (more or less) frequently desirable for users and computers to change the ways they view data
- It is undesirable for the computer to dictate or constrain the ways that users view data

Thus, it is necessary to be able to support different types of views of a data resource. Users need to be able to work with logical representations of data, which are independent of any physical considerations of how the data are actually stored and managed on computer facilities. Users view data in terms of high-level entities, e.g., staff members, tools, vehicles, products, orders, and customers. Meanwhile, computer facilities, e.g., access methods, operating systems, and DBMSs, need to be able to work with more physical representations. They view data in terms of records and files, with index

structures, B-trees, linked lists, pointers, addresses, pages, and so forth.

These requirements lead us to conclude first that there are two fundamentally different types of data views: logical and physical. The logical views are user-oriented, while the physical views are computer-oriented (Figure 2-1).

A second conclusion is that there must be a mapping or transformation between the logical and physical views. After all, the ultimate objective is to enable users to gain access to their data that reside on computerized media. This mapping might be simple if there were only one user view and one database, but that is not the real-world situation. Rather, there are multitudes of user views and commonly many (sometimes hundreds or thousands) databases in an enterprise.

Each user view could be mapped directly to the underlying databases (Figure 2-2). This solution suffers, however, when change is introduced in either type of view. If a physical database is restructured on a disk to provide more efficient performance, then the mapping to each of the user views that references that database can be affected. If a logical view is revised to present information in a somewhat different way, then the mapping to each of the referenced databases may be affected. Independence of logical and physical considerations would not have been achieved, and we would find that physical computer factors would constrain the ways that users logically view their data. This is undesirable.

Using three-schema architecture terminology, "external schemas" represent user views of data, while "internal schemas" represent physical implementations of databases. Schemas are metadata, i.e., they are data about data. As a simple example, CUSTOMER-NAME and CHARACTER (17) are metadata describing the data value CHRISTOPHER ROBIN.

To enable multiple users to share a data resource that is implemented on potentially many physical databases, we insert between the users' views and the physical views a neutral, integrated view of the data resource. This view is called a "conceptual schema" in three-schema architecture terminology. Others sometimes call it an "enterprise view."

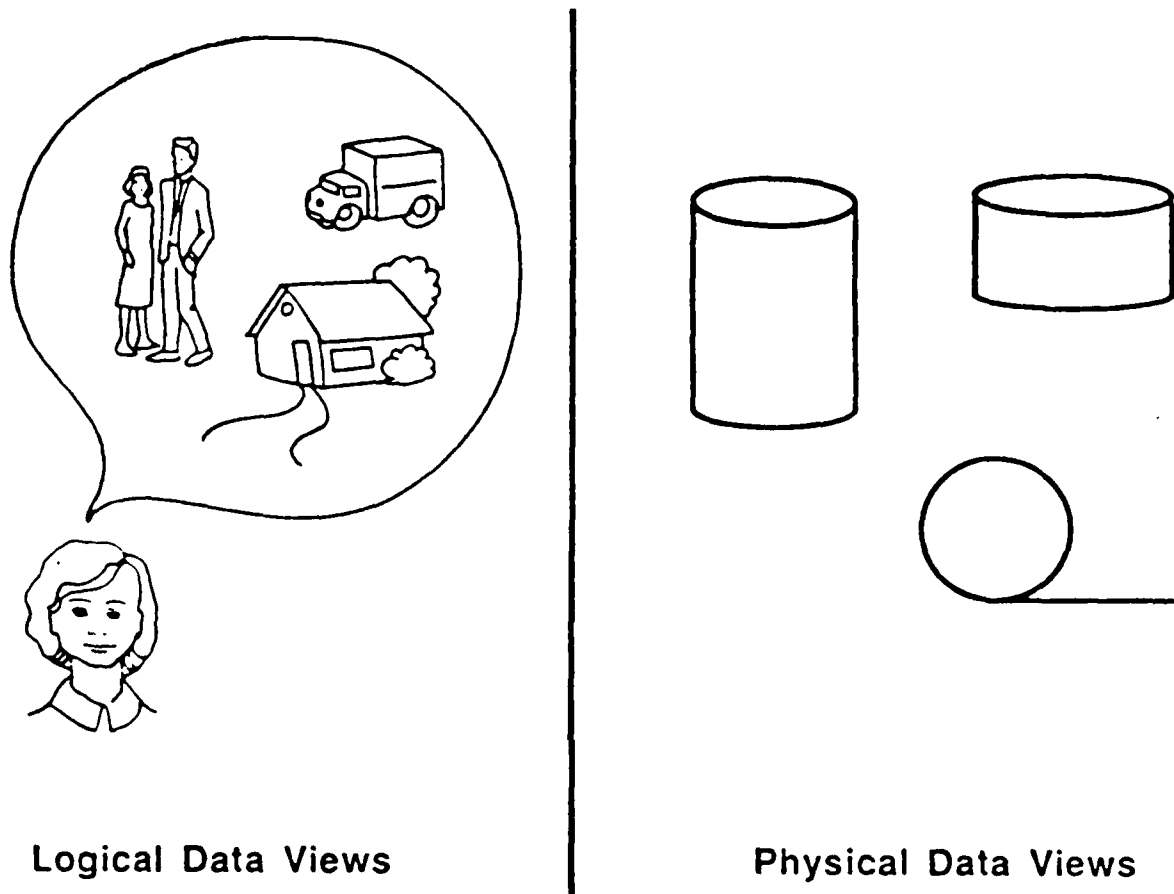


Figure 2-1. Two Fundamentally Different Views of Data: Logical and Physical

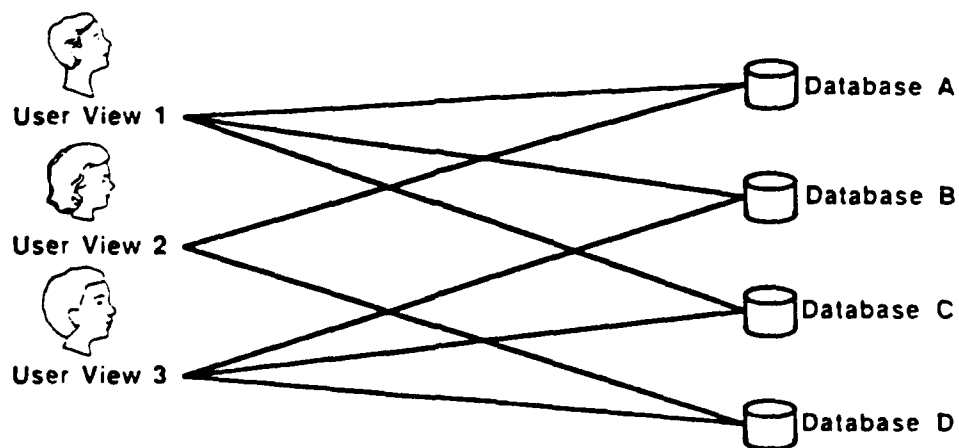


Figure 2-2. Direct Mapping of Logical and Physical Views

As the vehicle for data integration and sharing, the conceptual schema also carries metadata for enforcement of data integrity rules. It is extensible, consistent, accessible, shareable, and enables the data resource to evolve as needs change and mature.

Figure 2-3 illustrates the relationships between the three types of schemas. The schemas and the mappings between them are the mechanism for achieving both data independence and support of multiple views. An internal schema can be changed to improve efficiency and take advantage of new technical developments without altering the conceptual schema.

The conceptual schema represents knowledge of shareable data. There may be access controls and security restrictions placed upon these common data, but they are not restricted to access by only one user. The conceptual schema does not describe personal data.

The scope of the conceptual schema expands through time. The conceptual schema extension methodology continually expands the conceptual schema to include knowledge of more shared data. The external-conceptual mappings protect the external schemas and the transactions/programs that depend on them from most modifications incurred in evolving the conceptual schema.

Adding data to the integrated, common resource does not start over in defining the data resource, nor does it create another stand-alone database. Rather, development of its database must examine questions of how those data relate to what is already known by the conceptual schema. The result will be an integrated data resource whose scope is expanded gradually. It is absolute folly to approach integration of the data resources of an organization all at once; the job must be taken on piecemeal. The conceptual schema is the integrator.

The CDM contains all three types of schemas, as well as the interschema mappings. It not only documents these metadata, but also supplies appropriate metadata to support transaction processing.

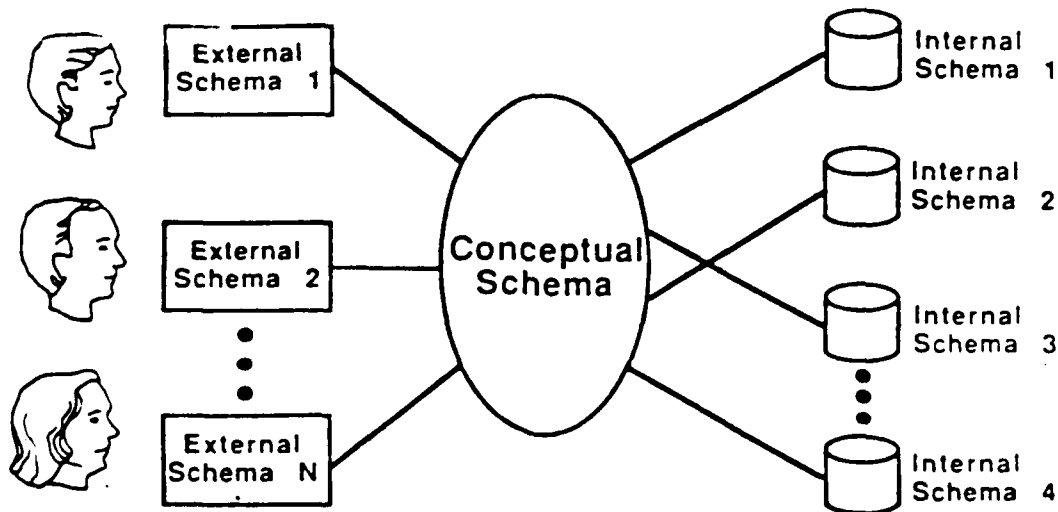


Figure 2-3. The Three-Schema Architecture: One Conceptual Schema That Provides for Integration and Independence of Many External Schemas and Many Internal Schemas

2.1.2 Representation of the Three Types of Schemas

In the IISS, the Three-Schema Architecture is implemented through the CDM facilities to store each of the three types of schemas and the interschema mappings. An appropriate representation mode has been selected for each of the three types of schemas.

The conceptual schema is represented by an IDEF1 model. The CDM stores this model in terms of entity classes, attribute classes, and relation classes.

The external schemas are represented by tables. The user views the common data resource in terms of flat, simple tables. The mappings between these tables and the IDEF1 model of the conceptual schema are part of the CDM database.

The internal schemas are represented in terms of physical database components, including record types and inter-record relationships. The CDM Processor routines convert the users' data access requests, which are phrased in terms of tables, into requests against the conceptual schema IDEF1 model, then into requests against the physical database structures described in the internal schema part of the CDM.

2.1.3 Integration Methodology

The Integration Methodology is the set of procedures and guidelines that are used to expand the conceptual schema and to increase the sphere of common data available to support users and applications. The schemas and schema mappings in the CDM are built, maintained, and accessed using the Integration Methodology and the CDMP.

The Integration Methodology is intended to guide the CDM Administrator in building and maintaining the conceptual schema and in keeping its mappings to the internal and external schemas highly accurate. This methodology consists of a set of techniques for building the conceptual schema in gradual increments, for building external and internal schemas from portions of the conceptual schema, for developing schema mappings, and for keeping these various CDM components current.

The first step in populating the CDM is to select a portion of the data and to document it in the conceptual schema. Then external and internal schemas for those data are built and mapped to the conceptual schema. Subsequently, other portions of the data resource are incorporated into the conceptual schema, and new external and internal schemas and mappings are developed. The CDM is populated gradually, in increments, rather than all at once. It evolves through time.

A conceptual schema is represented by a semantic data model. The IISS uses the IDEF1 methodology, with certain extensions from DACOM's Data Modeling Technique. (Subsequent to the development of CDM subsystem, IDEF1 was formally extended. See Appendix C for references.) The data model reflects business policy, provides a rigorous view of the meaning of the data resource, and is independent of the physical implementation of the data resource.

Building a data model is a rigorous procedure, whose objective is to discover and document the semantic data structure in its most fundamental terms. The modeling is a multi-step process that requires substantial input from users who are expert in the subject area.

The fundamental steps of the CDM Integration Methodology are as follows:

1. Identify the scope of the initial increment of the conceptual schema.

2. Develop the data model for that initial increment of the conceptual schema.
3. Load the data model into the CDM database.
4. Identify any physical databases or files within the scope of data in the conceptual schema.
5. Load their internal schemas into the CDM database.
6. Build the conceptual-to-internal schema mappings for the internal schemas loaded in Step 5.
7. Load the conceptual-to-internal schema mappings into the CDM database.
8. Determine which users/application programs should have external schemas mapped from the conceptual schema.
9. Design the external schemas identified in Step 8, and their mappings to the conceptual schema.
10. Load the external schemas and external-to-conceptual schema mappings into the CDM database.
11. Identify the scope of the next increment to the conceptual schema.
12. Develop the data model for the next increment of the conceptual schema.
13. Integrate the data model from Step 12 with the data model of the existing conceptual schema.
14. Load the integrated data model into the CDM database.
15. Verify that the conceptual-to-internal and external-to-conceptual schema mappings are still valid, correcting them as needed.
16. Identify any additional physical databases or files that are now within the scope of the extended conceptual schema.
17. Load their internal schemas into the CDM database.

18. Build the conceptual-to-internal schema mappings for the incremented portions of the conceptual schema.
19. Load the conceptual-to-internal schema mappings into the CDM database.
20. Identify any additional users or application programs that should be supported by the extended conceptual schema.
21. Design external schemas to support the users/application programs identified in Step 20, and develop their external-to-conceptual schema mappings.
22. Load the external schemas and external-to-conceptual schema mappings from Step 21 into the CDM database.
23. Repeat Steps 11 through 22 for each increment to the conceptual schema.

The evolutionary strategy for the conceptual schema should be developed early in the life of the above cycle. The strategy should ensure that the common data resource evolves in a manner that serves the enterprise's need for controlled, shared data. One tactic is to define the initial scope by that of an existing database that has a corresponding data model. Ideally, that database would contain core information of high interest to the target user community.

Perhaps the most important point to understand about the CDM Integration Methodology is that the incorporation of additional data into the common data resource MUST be done in conjunction with the existing conceptual schema. No data can be accessed using the CDM integrated facilities, including the Neutral Data Manipulation Language, unless they are known to the CDM. Adding data causes the conceptual schema to expand in a consistent manner that enables integration to occur. By contrast, adding data to an environment that does not use conceptual schema technology just adds more fragmentation to what is probably already at best an interfaced (not integrated) system.

Applying the CDM Integration Methodology is not like swallowing a pill. It requires precise knowledge of the meanings of the data that are to be available in the integrated common data resource. It means not just building IDEF1 models for those databases, but also analyzing the models for overlap.

synonyms, homonyms, and all the incipient anomalies and quirks that somehow have crept into our database structures over the years. The cost is measured in man-months of effort; the benefits are integration and a knowledge base that can be built on and evolved in the future.

2.1.4 Contributions to IRRIASSPA

The use of the Common Data Model and the Three-Schema Architecture allows an organization to benefit from contributions to IRRIASSPA, which are part of the objectives of the USA's Integrated Computer Aided Manufacturing (ICAM) project to develop the Integrated Information Support System (IISS). The contributions can best be summarized as follows:

Independence - the IISS allows the separation of the description and manipulation of logical data structures from the actual physical data representations and isolates implementation changes from user views and programs.

Relatability - the NDDL used in building the CDM allows the CDM Administrator to define, modify, and maintain relationships among data.

Resiliency/Recoverability - although not specifically addressed by the CDM, the design of the CDM Processor provides the ability to recover from failures without damage to the data resource.

Integrity - is provided through the use of data integrity constraints, which the application may specify and the CDM Processor enforces.

Accessibility - the NDDL allows the definition of data that resides not only in different databases but also on different computers.

Security - not expressly addressed by the CDM.

Shareability - is provided by support of multiple user views (i.e., external schemas) of the data resource.

Performance - the NDML, by use of the CDM, allows data from multiple resources to be addressed in a cost-effective manner in a distributed environment.

Administration - by providing a means of documenting the

meanings in the data resource and of providing a vehicle by which consistency can be maintained even as the scope of the CDM is extended. It also allows the maintenance of information about data in different databases.

2.2 Basic Components of the Design

The Common Data Model(CDM) subsystem is comprised of three components:

1. The CDM database, which is the database dictionary of the IISS
2. A logical model of the CDM database called CDM
3. The CDM Processor (CDMP), which is the distributed database manager of the IISS

This section will briefly discuss each of these basic components and show how they interrelate, one with another.

2.2.1 The CDM Database

The CDM database is the database dictionary of the IISS. It captures knowledge of the locations, characteristics, and interrelationships of all shared data in the system. The most significant feature of the CDM database is that it implements the ANSI/X3/SPARC concepts of the three-schema approach to data management. These three types of schemas are the conceptual schema (CS), the internal schemas (IS), and the external schemas (ES).

The conceptual schema describes a neutral, integrated view of the shared data resource. There is one conceptual schema in an enterprise. It is independent of physical database structures and boundaries and is neutral to biases of individual applications. Each external schema represents a user or application view of data. Requests are made against external schemas. Each internal schema represents an external schema to the local DBMS.

The CDM database is implemented as a relational database, which presently resides on a VAX 11/780 computer. It is accessed by the CDMP at compile-time to generate appropriate local DBMS calls against internal schemas to process a user's NDML request against an external schema.

The CDM database is represented logically using a semantic data modeling technique called IDEF1. This method of data modeling is a hybrid of the entity-relationship approach, the relational model, and the Smith's 2D data abstraction approach. This logical model of the CDM database is called CDM1.

2.2.2 CDM1

CDM1 is a model of metadata, i.e., data about data. It gives the logical structure of the CDM database which maintains the metadata. These metadata describe the meanings and characteristics of user data.

The conceptual schema portion of the CDM1 model is related to portions that describe internal and external schemas. An internal schema describes a local database structure in just enough detail to give the CDMP adequate information to generate code that can be processed by the pertinent local DBMS. Because one of the requirements of the IISS is that it provide integration of data in existing databases, the mappings between the conceptual schema metadata and the internal schema metadata are not simple. IISS does not have the luxury of supporting only certain clean database structures. It is very likely that an attribute may be represented by one or more data files, which may be in different databases and even on different computers, or by relationships between record types.

An external schema describes the portion of the conceptual schema that is within the purview of a user or application. An external schema is equivalent to a view in the relational model. The conceptual-to-external schema mapping part of the CDM1 is straightforward. The present implementation of the CDM subsystem supports any external schema that can be formed by joining conceptual schema entities and selecting attributes.

Thus, the CDM1 model is a semantic data model that describes the logical structure of the CDM database. The CDM1 represents the conceptual schema, the internal schemas and their mappings from the conceptual schema, and the external schemas and their mappings from the conceptual schema.

2.2.3 The CDM Processor

The CDMP is the distributed database manager of the IISS. It builds on top of local DBMS services to provide data access. The CDMP plays both a compile-time and a run-time role in the processing of transactions. The compile-time component is

called the CDMP Precompiler. The run-time components are called the CDMP Distributed Request Supervisor (DRS) and the CDMP Aggregator.

2.2.3.1 CDMP Precompiler

The CDMP Precompiler performs the following functions for each data request:

1. Parses the request
2. Transforms the request from an external schema access to a conceptual schema access
3. Decomposes the request into subrequests, each of which accesses one internal schema
4. Determines an appropriate access path for each subrequest generating code that can be processed by the pertinent local DBMS
5. Generates code to transform any data to be extracted from local databases from internal to conceptual schema format (this code is called a Request Processor Packet or RPP)
6. Generates code to transform any data results from conceptual to external schema format (this code is called a C/E Transformer or CEX)
7. Generates code to invoke appropriate RPPs and CEXs at run-time, via calls to the NTM Subsystems

The CDMP Precompiler accesses the CDM database to find metadata for the interschema transforms and integrity constraints for update requests.

After successful precompilation of a user's program, which contains imbedded data requests in a SQL-like language called the Neutral Definition/Manipulation Language (NDML), the CDMP has produced the following code modules:

1. Modified user program, which now contains calls to the NTM, which will activate appropriate processes at run-time.
2. One Request Processor (RP) per DBMS that manages data

to be accessed by the user program. Each RP contains one or more RPPs.

3. One Conceptual-to-External Transformer (CEX), which will deliver query results to the modified user program at run-time.

2.2.3.2 Distributed Request Supervisor

There is presently one CDMP Distributed Request Supervisor (DRS), which has responsibility for scheduling and coordinating the various subrequests of user transactions. The DRS uses request graphs produced by the CDMP Precompiler to determine which operations are to be performed where. The DRS also uses knowledge of communications costs and intermediate result volumes in its algorithm for scheduling RPPs.

Request Processors always deliver results as relations. The relations are operated upon by the Aggregators.

2.2.3.3 Aggregators

An Aggregator is called to perform a single function, e.g., a union or a join, on two sets of data, each of which exists in a single sequential file. These data sets are the results of an RPP or another Aggregator.

An Aggregator always deals with data in conceptual schema format.

SECTION 3

RESPONSIBILITIES OF THE CDM ADMINISTRATOR

The role that the CDM Administrator plays in the IISS environment is not unlike that of the database administrator in that the CDMA is responsible for the following:

1. Establishing Data Standards
2. Maintaining the CDM
3. Protecting the CDM
4. Facilitating Use of the CDM

Each of these areas is of major importance to the organization and a failure to properly administer either of these areas of responsibility can cost the organization dearly.

3.1 Establishing Data Standards

One of the early roles of the CDMA is the establishment of data standards. Part of this work has already been initiated during the development of the CDM1. The work that remains is to determine what types of standards to implement and to gain acceptance for the use of these standards. It should be noted that, without acceptable standards, it will be difficult, if not impossible, for the CDMA to enforce any level of standardization.

3.2 Maintaining the CDM

The CDMA must maintain the CDM. This entails the building of the initial conceptual schema (CS), internal schemas (IS), CS to IS mappings, external schemas (ES), and ES to CS mappings, as well as extending the model and modifying and deleting elements as needed. It is to be expected that the need for extending and modifying the CDM will grow over time, slowly at first, then growing rapidly as the benefits of the concept are proved before leveling off after several years.

3.3 Protecting the CDM

One of the most important responsibilities of the CDMA is the protection of the CDM against loss, theft, and corruption.

UM 620141001
1 November 1985

be it intentional or not. At issue is the substantial investment that went into the development of the CDM and the potential damage that can be caused to the enterprise should the data fall into the wrong hands.

3.4 Facilitating Use of the CDM

The CDMA must make the CDM available to all those who can potentially gain from the use of the CDM and have legitimate reason to do so. This may involve making the CDM available on other computers in the network. It also involves communicating with the CDM user and potential users as to the contents and performance of the CDM, as well as the usability of the data. Part of this communication will involve solving problems and answering questions and reporting the status of the CDM.

SECTION 4

MAINTAINING THE CONCEPTUAL SCHEMA

4.1 Methodology Overview

This section and its subsections (4.2 - 4.3) introduce the methodology for building and updating a conceptual schema. The portion of the CDM database that contains a conceptual schema is described, and the basic approach to developing a conceptual schema is presented. Detailed instructions for filling out the modeling forms are included.

4.1.1 CS Structure

A conceptual schema is essentially a single IDEF1 model that describes all of the common data in an enterprise. Consequently, its components are those of any IDEF1 model:

- Entity Classes
- Relation Classes
- Attribute Classes
- Attribute Use Classes
- Inherited Attribute Use Classes
- Key Classes
- Key Class Members

Detailed explanations of these can be found in the IDEF1 documentation. (Extensions to the IDEF1 language, referenced in Appendix C, simplify the IDEF1 terminology used here.)

In addition to the usual metadata (data about data) contained in any IDEF1 model, the conceptual schema requires certain new elements of metadata. Key class numbers are assigned to enable alternate key classes for the same entity class to be distinguished from one another. Tag numbers, tags (names), and tag labels are assigned to enable attribute use classes within the same entity class to be distinguished from one another. Data types and sizes are identified for all attribute classes.

The conceptual schema must conform to several rules that cause the data relationships and descriptions to be as explicit as possible. (Note: In these rules the phrase "any number" includes the possibility of zero.)

1. Single-Owner Rule: An entity class can own any number of attribute classes. Every attribute class is owned by exactly one entity class.
2. Every entity class contains one or more attribute use classes. Every attribute use class is contained in exactly one entity class.
3. Every attribute class appears as exactly one attribute use class in its owner entity class. An attribute class can also appear as any number of attribute use classes in any number of other entity classes. Every attribute use class corresponds to exactly one attribute class.
4. Every entity class has one or more key classes. Every key class is for exactly one entity class.
5. Every key class is composed of one or more key class members. Every key class member is in exactly one key class.
6. An attribute use class can be used as a member of any number of key classes for the entity class in which it is contained. An attribute use class cannot be used as more than one member of the same key class; i.e., every member of a key class must be a different attribute use class. An attribute use class in one entity class cannot be used as a member of a key class for any other entity class. Every key class member is exactly one attribute use class.
7. An entity class can be independent in any number of relation classes and dependent in any number. An entity class cannot be both independent and dependent in the same relation class. Every relation class has exactly two entity classes: one independent, one dependent.
8. A key class can migrate through any number of relation classes in which its entity class is independent. A key class cannot migrate through a relation class in which its entity class is dependent or one in which its entity class is not involved. Every relation class has exactly one key class from the independent entity class migrating through it into the dependent entity class.

9. Every relation class is a migration path for one or more inherited attribute use classes, one for each member of the key class that migrates through it. Every inherited attribute use class has exactly one relation class as its migration path.
10. Every member of the key class that migrates through a relation class creates exactly one inherited attribute use class in the dependent entity class for that relation class. Every inherited attribute use class is created from exactly one key class member.
11. Every attribute use class in an entity class represents either one attribute class that is owned by that entity class or one inherited attribute use class that migrated into that entity class. Every inherited attribute use class is represented by exactly one attribute use class.
12. Unique-Key Rule: No two entity instances in an entity class can have identical values in the same key class for that entity class. For a multi-member key class, instances can have identical values for some members, but not for all.
13. No-Null Rule: Every entity instance in an entity class has a value in each attribute use class in that entity class.
14. No-Repeat Rule: No entity instance in an entity class can have more than one value in any attribute use class in that entity class. This rule is equivalent to the first normal form in the relational database model.
15. Full-Functional-Dependency Rule: No entity instance in an entity class can have a value in an owned, nonkey attribute use class that can be identified by less than the entire key value for that entity instance. This rule applies only to entity classes with multi-member key classes and is equivalent to the second normal form in the relational database model.
16. No-Transitive-Dependency Rule: No entity instance in an entity class can have a value in an owned, nonkey attribute use class that can be identified by the value in another owned or inherited, nonkey attribute use class in that entity class. This rule is equivalent to

the third normal form in the relational database model.

17. **Smallest-Key-Class Rule:** No entity class with a multi-member key class can be split into two or more entity classes, each with fewer members in its key class, without losing some information. This rule is a combination and extension of the fourth and fifth normal forms in the relational database model.

4.1.2 Basic Approach (Onion Concept)

The complete conceptual schema for an enterprise contains thousands of entity classes and a corresponding number of relation classes, attribute classes, etc. It is much too large to be built all at once. Instead, it must be built in increments -- each one building on the prior ones, until the conceptual schema is complete. The increments are like the layers of an onion; as each layer is added, the onion gets a little larger.

The process of "growing" the conceptual schema involves two procedures, both of which are enhanced versions of the IDEF1 modeling procedure. The first is used to build the initial increment only. The second is used to build each additional increment. The only difference between the two is that the second must be concerned about the integration of the new increment with the existing conceptual schema. This involves being continually aware of which components of the conceptual schema are within the scope of the new increment and how any of those components will be affected by the addition of the new increment. These two procedures are in Sections 4.2 and 4.3, respectively.

4.1.3 Modeling Forms

Because the methodology for maintaining the conceptual schema is based on the IDEF1 information modeling methodology, it uses most of the IDEF1 forms:

- Source Material Log
- Source Data List
- Entity Class Pool
- Entity Class Definition
- Relation Class Matrix
- Attribute Class Pool
- Kit Cover Sheet
- Entity Class Diagram

(optional)

Relation Class Definition	(optional)
Attribute Class Diagram	(optional)
Entity Class/Attribute Class Matrix	(optional)
Attribute Class Migration Index	(optional)
Author Page Control Log	(optional)
Index Control Log	(optional)
Kit Control Log	(optional)
Text Control Log	(optional)
FEO Control Log	(optional)
Entity Class Set Control Log	(optional)
Entity Class/Function View Matrix	(optional)

Please refer to the IDEF1 documentation for detailed descriptions of these forms.

A few of the regular IDEF1 forms have certain shortcomings that make them unsuitable for use in directly loading the conceptual schema tables into the CDM database. The forms listed below were designed to eliminate those shortcomings:

- Relation Classes
- Owned Attribute Classes
- Inherited Attribute Classes

The rest of this section contains a detailed description and two samples (one blank, one filled in) of each of these forms.

NOTE:

When using the NDDL (see Neutral Data Definition Language Users Guide - UM 620141100) for maintaining the conceptual schema in the CDM database, names should be substituted for any/all numbers on the modeling forms. A discussion of the NDDL can be found in Section 5.1.1.

Relation Classes Form

Purpose:

To provide a single source of information about relation classes that are to be described in the conceptual schema.

Instructions:

Fill in one or more pages for each entity class that is independent in a relation class. List only those relation

classes in which the entity class is independent; do not list any relation classes in which it is dependent. Do not fill in a page for an entity class that is dependent in all of its relation classes.

<u>Form Area</u>	<u>Explanation</u>
1. Independent Entity Class Name	Name of the entity class that is independent in the relation class. This will be the same for all relation classes entered on a page. It is included only to make the entry readable; it is not used in loading the conceptual schema.
2. Relation Class Label	Label of the relation class. This is part of the unique identification of a relation class.
3. R.C. Card.	Symbol for the cardinality of the relation class.
4. Dependent Entity Class Name	Name of the entity class that is dependent in the relation class. It is included only to make the entry readable; it is not used in loading the conceptual schema.
5. Dep. E.C. No.	Number of the entity class that is dependent in the relation class.
6. Ind. K.C. No.	Number of the key class in the independent entity class that migrates through the relation class into the dependent entity class.
7. Node	Number of the entity class that is independent in all of the relation classes listed on the page.

All other form areas correspond to areas on the regular

UM 620141001
1 November 1985

USED AT	AUTHOR PROJECT										DATE REV		WORKING	REVIEW	DATE		CONTEXT	
	NOTES 1 2 3 4 5 6 7 8 9 10											DRAFT						
Independent Entity Class Name												Relation Class Label		R.C. Card	Dependent Entity Class Name		Dep. E.C. No.	Ind. K.C. No.
①												②	③	④	⑤	⑥		
MODE ⑦												TITLE		Relation Classes		NUMBER		

Figure 4-1. Relation Classes Form

UM 620141001
1 November 1985

USED AT	AUTHOR PROJECT 6201M MCM	DATE Aug 1983	WORKING EVAL RECOMMENDED FINANCIAL	REASON	DATE	CONTEXT
NOTES 1 2 3 4 5 6 7 8 9 10						
Independent Entity Class Name	Relation Class Label	R.C. Card	Dependent Entity Class Name	Dep. E.C. No.	Ind K.C. No.	
Op Exec Plan	Is	Y	OEP Group Member	E13	K1	
Op Exec Plan	Has	D	OEP Stored Item Req	E61	K1	
Op Exec Plan	Is Used To Manufacture	D	Op Exec Plan Part	E15	K1	
Op Exec Plan	Is	Y	Op Exec Plan Comp	E14	K1	
Op Exec Plan	Has	D	Operation	E10	K1	
Op Exec Plan	Has	D	Op Exec Plan Obsl	E71	K1	
NOX	E11	Relation Classes			NUMBER	62

Figure 4-2. Relation Classes Form Example

Owned Attribute Classes Form

Purpose:

To provide a single source of information about owned attribute use classes that are to be described in the conceptual schema.

Instructions:

Fill in one or more pages for each entity class that owns an attribute use class, either key or nonkey. List only those attribute use classes that are owned by the entity class; do not list any attribute use classes that are inherited by the entity class. Do not fill in a page for an entity class that contains only inherited attribute use classes.

Form Area

Explanation

1. Tag No.	Tag number for the attribute use class.
2. A.C. Name & Label	Name, label, and any synonyms of the attribute use class. The name is listed first. The label is enclosed in parentheses and placed on the line below the name. If the name and label are identical, the label can be omitted. If the attribute use class has any synonyms, the term "Synonyms:" is placed below the name and label and the synonyms are listed under it.
3. A.C. No.	Attribute class number for the attribute use class.
4. A.C. Definition	Definition of the attribute use class.
5. Type ID.	Format description for the attribute use class indicating data type (numeric, character, etc.), length, and decimal length (if applicable). The data type

must be one from the CDM Data
Type Table.

6.	Mbr. of K.C. No.	Number(s) of the key class(es) to which the attribute use class belongs, if any.
7.	Node	Number of the entity class that owns all of the attribute use classes listed on the page.

All other form areas correspond to areas on the regular IDEF1 forms. Please refer to the IDEF1 documentation for details about those areas.

Inherited Attribute Classes Form

Purpose:

To provide a single source of information about inherited attribute use classes that are to be described in the conceptual schema.

Instructions:

Fill in one or more pages for each entity class that inherits an attribute use class. List only those attribute use classes that are inherited by the entity class; do not list any attribute use classes that are owned by the entity class. Do not fill in a page for an entity class that contains only owned attribute use classes.

Form Area

Explanation

1.	Tag No.	Tag number for the attribute use class.
2.	Tag & Label	Name, label, and any synonyms of the attribute use class. The name is listed first. The label is enclosed in parentheses and placed on the line below the name. If the name and label are identical, the label can be omitted. If the attribute use

UM 620141001
1 November 1985

USED AT	AUTHOR PROJECT										DATE	REVISION		DATE	CONTENT														
	NOTES 1 2 3 4 5 6 7 8 9 10											WORKING	DEWIT																
												IN COMMANDED																	
												PUBLICATION																	
Tag No.	A.C. Name & Label										A.C. No.	A.C. Definition			Type ID	Mbr of K C No													
①	②										③	④			⑤	⑥													
<table border="1"> <tr> <td>MODE</td> <td>⑦</td> <td>TITLE</td> <td colspan="10">Owned Attribute Classes</td> <td>NUMBER</td> </tr> </table>																MODE	⑦	TITLE	Owned Attribute Classes										NUMBER
MODE	⑦	TITLE	Owned Attribute Classes										NUMBER																

Figure 4-3. Owned Attribute Classes Form

UM 620141001
1 November 1985

USED AT		AUTHOR		DACOM (CEM, DRR)		DATE		Aug 1983		X WORKING		DATE		CONTEXT		
		PROJECT		6201M MCNM		REV				DRAFT						
		NOTES		1 2 3 4 5 6 7 8 9 10						RECOMMENDED						
Tag No.	A.C. Name & Label						A.C. No.		A.C. Definition						Type ID	Mbr. of K.C. No.
T37	Operation Execution Plan Group Identification (OEP Group ID)						A10		A unique identifier assigned to identify groups of operation execution plans						N(4)	K01
T134	Status						A34		A code that indicates where a group of operation execution plans is within its life cycle						C(8)	
T135	Total Operation Execution Plans (Total OEPs)						A35		The total number of operation execution plans that make up the group						N(4)	
MODE	E12		TITLE		Owned Attribute Classes						NUMBER		69			

Figure 4-4. Owned Attribute Classes Form Example

class has any synonyms, the term "Synonyms:" is placed below the name and label, and the synonyms are listed under it.

3.	A.C. No.	Attribute class number for the attribute use class.
4.	Ind. E.C. No.	Number of the independent entity class from which the attribute use class was inherited.
5.	Ind. K.C. No.	Number of the key class in the independent entity class that migrated through the relation class named in the "Migration Path R.C. Label" area.
6.	Ind. Tag No.	Tag number of the attribute use class in the independent entity class that migrated to become this attribute use class.
7.	Migration Path	Label of the relation class through which the attribute use class was inherited.
8.	Mbr. of K.C. No.	Number(s) of the key class(es) to which the attribute use class belongs, if any.
9.	Node	Number of the entity class that contains all of the attribute use classes listed on the page.

All other form areas correspond to areas on the regular IDEF1 forms. Please refer to the IDEF1 documentation for details about those areas.

UM 620141001
1 November 1985

USED AT	AUTHOR PROJECT										DATE REV.		WORKING		REVISION		DATE		CONTEXT	
	NOTES 1 2 3 4 5 6 7 8 9 10												DRAFT							
Tag No.	Tag & Label	A.C. No.	Ind. E.C. No.	Ind. K.C. No.	Ind. Tag No.	Migration Path R.C. Label					Mbr. of K.C. No.									
①	②	③	④	⑤	⑥	⑦					⑧									
Inherited Attribute Classes																				
MODE: ⑨	TITLE										NUMBER									

Figure 4-5. Inherited Attribute Classes Form

UM 620141001
1 November 1985

USED AT		AUTHOR PROJECT 6201M MCM		DATE: Aug 1983 REV.		X WORKING DRAFT RECOMMENDED PUBLICATION		DATE		CONTEXT	
NOTES 1 2 3 4 5 6 7 8 9 10											
Tag No.	Tag & Label	A.C. No.	Ind. E.C. No.	Ind. K.C. No.	Ind. Tag No.	Migration Path R.C. Label	Mbr. of K.C. No.				
T73	Requisition Number (Req No)	A09	E20	K01	T28	Is For	K01				
T191	Issuing Manufacturing Area Identification (Iss Mfg Area ID)	A07	E20	K01	T182	Is For	K01				
T192	Destination Manufacturing Area Identification (Dest Mfg Area ID)	A07	E24	K01	T40	Is					
NODE E67		TITLE					Inherited Attribute Classes		NUMBER 139		

Figure 4-6. Inherited Attribute Classes Form Example

4.2 Building the Initial CS

This section and its subsections (4.2.1 - 4.2.5) describe the procedure for initiating an enterprise's conceptual schema. The procedure is concerned with creating a detailed description (an information model) of a portion of the enterprise's common data and with collecting the data required to place that description in the CDM database as the first piece of the conceptual schema (the first layer of the onion). It is not concerned with deciding which portion of the common data to describe nor with setting up the CDM database and its utilities; these things must be done before starting the procedure. The procedure consists of six phases, the first five of which are patterned after those in IDEF1. The five IDEF1 phases are as follows:

- Phase 0 - Starting the Project
- Phase 1 - Defining Entity Classes
- Phase 2 - Defining Relation Classes
- Phase 3 - Defining Key Classes
- Phase 4 - Defining Nonkey Attribute Classes

The procedure for the sixth phase, which consists of populating the CDM database with the conceptual schema, is described in Section 5. Each IDEF phase is described in a subsequent subsection.

4.2.1 Phase 0: Starting the Project

Objectives:

- State the purpose, scope, and viewpoint for the information model.
- Establish the project team.
- Develop a phase-level project schedule.
- Collect and catalog relevant source material.

This phase is patterned after Phase 0 of IDEF1, and the description presented here is less detailed than the one in the IDEF1 documentation. Please refer to that documentation for

further information.

Tasks:

1. The CDM Administrator appoints a project manager.

Usually, this will be the CDM Administrator.

2. The project manager states the purpose for building the information model.

This explains why the model is needed, i.e., what it will be used for. A model built with this procedure is primarily used to initiate the enterprise's conceptual schema. (It is not necessary to explain why the conceptual schema is needed.) If the model has other purposes, they should be mentioned also.

3. The project manager states the scope of the information model.

This sets the boundary of the model. It should be specific enough to be useful in deciding whether or not a particular element of common data should be included in the model. Some of the things that can be used as the basis for scoping a model are the following:

- Information subjects: parts, employees, sales orders, etc.
- Functions: engineering release, shop floor control, etc.
- Existing computer files or databases
- Existing computer application systems

4. The project manager states the viewpoints for the information model.

This explains the mental attitude or role that people should adopt when looking at and thinking about the model, i.e., in whose place they should put themselves. Usually, this will be the job title of someone who is intimately involved with the common data being modeled.

5. The project manager appoints the project team members.

The four roles to be filled are as follows:

- Modeler - one or two IDEF1 experts.
- Source - several subject experts, i.e., people who have in-depth knowledge about some or all of the common data being modeled.
- Reviewer - several subject experts; some sources may also serve as reviewers. The CDM Administrator must also serve as a reviewer to ensure that the model, as it is developed, is properly documented for loading into the CDM database tables.
- Librarian - a person who is trained and experienced in coordinating kit reviews and in maintaining files of model documentation; a modeler may also serve as the librarian.

6. The project manager appoints the acceptance review committee members.

This committee should consist of subject experts from the area being modeled and from other, related areas.

7. The project manager schedules the project phases.

Estimate the amount of effort needed to complete each phase (usually in man-weeks or man-months) and then convert those estimates to elapsed times and milestones based on the availability of the project team members. At this point, only the phases are scheduled; the individual tasks within a phase will be scheduled when that phase is started.

8. The project manager schedules the remaining Phase 0 tasks.

Estimate the amount of effort needed to perform each remaining task in this phase (usually in man-hours or man-days) and then convert those estimates to elapsed times and milestones based on the availability of the project team members who will perform those tasks. The schedules for the subsequent phases should be adjusted if they are inconsistent with these task schedules.

9. The modeler develops a data collection plan.

Determine what kinds of source material are needed and where and how to get that material.

10. The project manager conducts a project kick-off meeting attended by the project team members.

The objectives of the meeting are as follows:

- To introduce the team members to one another and to the roles they will be performing.
- To determine which members need IDEF1 training.
- To present, discuss, and finalize the statements of purpose, scope, and viewpoint.
- To present and discuss the project schedule.
- To present, discuss, and finalize the data collection plan.

11. The modeler collects source material from the sources.

Gather the documents, policies, procedures, database designs, etc., and interview the sources in accordance with the data collection plan (Task 9).

12. The modeler catalogs the source material.

Prepare Source Material Log Forms and Source Data List Forms. If a database design is among the source material, the record names and data field names should be included in the source data list.

13. The modeler explains any author conventions.

These are deviations from or additions to the regular IDEF1 methodology. Mention the use of the three specially designed modeling forms: Relation Classes Form, Owned Attribute Classes Form, and InheritedAttribute Classes Form.

Deviation from IDEF1:

Usually, kits are not used to accomplish the review of the Phase 0 model documentation; the essentials are reviewed during the kick-off meeting (Task 10). However, the project manager may require that kits be used to supplement or replace the kick-off meeting.

4.2.2 Phase 1: Defining Entity Classes

Objective:

- Identify and define the apparent entity classes that are within the scope of the model.

This phase is patterned after Phase 1 of IDEF1, and the description presented here is less detailed than the one in the IDEF1 documentation. Please refer to that documentation for further information.

Tasks:

1. The project manager decides what method to use to review the Phase 1 model.

The options are to distribute review kits, to hold a walk-through meeting, or to do both. The factors to consider are the following:

- Some team members may have to travel to attend a walk-through. How many trips can the project budget afford?
- A review can usually be accomplished faster with a walk-through than with kits. Is there enough time to circulate kits, perhaps two or three times?
- Some reviewers may have very limited time to spend on the project. How can their time be used most effectively, by reviewing a kit or by attending a walk-through? Will they devote time to reviewing a kit on their own?

2. The project manager schedules the Phase 1 tasks.

Estimate the amount of effort needed to perform each task in this phase (usually in man-hours or man-days) and then convert those estimates to elapsed times and

milestones based on the availability of the project team members who will perform those tasks. The schedules for the subsequent phases should be adjusted if they are inconsistent with these task schedules.

3. The modeler builds an entity class pool.

Examine the entries in the source data list and deduce what sort of thing each entry identifies, describes, refers to, etc. For example:

- Employee number, name, birth date, and salary are data elements about an employee; hence, an "Employee" entity class.
- Part number, description, and dimensions are all about a part; hence, a "Part" entity class.

Each sort of thing is represented by an entity class. Talk to the sources when additional information is needed. The entity instances within an entity class should be distinguishable from one another by some unique identifier. Assign an entity class number to each entity class, and record it on an Entity Class Pool Form.

When examining record names from a database design, be careful to think about the "real-world thing" that each kind of record represents. Realize that several kinds of records may represent the same thing or, conversely, that one kind of record may represent several different things. Also, realize that certain kinds of records may be present for technical reasons only (performance, backup/recovery, etc.). Such records do not represent "real-world things" and should not result in entity classes being added to the pool.

4. The modeler defines each entity class.

Fill out an Entity Class Definition Form for each entity class in the pool. Talk to the sources when additional information about an entity class is needed. Check off each pool entry as it is dealt with.

Watch for synonyms (different names for the same thing) and homonyms (same name for different things). When there are synonyms for something, there is only one

entity class to define. Use the most commonly used name as the "official" entity class name, and record it and the corresponding entity class number on an Entity Class Definition Form. Record the other names as synonyms on the form. In the pool, add a note to each synonym entry referring to the official name or number.

For a homonym, there are two or more entity classes to define, one for each thing that the term represents. Pick a new name for each thing to clarify the differences. Record the new names in the entity class pool along with a new entity class number for each, and fill out Entity Class Definition Forms. For example, if an order can be either something received by an enterprise from a customer, or something sent by an enterprise to a vendor, call the first a sales order and the second a purchase order, and fill out two definition forms.

5. The modeler, reviewers, and librarian participate in reviewing the Phase 1 model.

The method of review was selected in Task 1. The modelers prepare the review materials (kits or walk-through handouts), the reviewers read and comment on the materials, and the modelers respond to the comments. If kits are used, the librarian coordinates their circulation. The CDM Administrator reviews the model to ensure that all model documents are prepared properly for loading the CDM database tables.

4.2.3 Phase 2: Defining Relation Classes

Objective:

- Identify and define the apparent relation classes that are within the scope of the model.

This phase is patterned after Phase 2 of IDEF1, and the description presented here is less detailed than the one in the IDEF1 documentation. Please refer to that documentation for further information.

Tasks:

1. The project manager decides what method to use to

review the Phase 2 model.

See Phase 1, Task 1, for the options and factors to consider.

2. The project manager schedules the Phase 2 tasks.

See Phase 1, Task 2, for details.

3. The modeler builds a relation class matrix.

List all of the entity classes across the top and down the left side of Relation Class Matrix Forms or on a large sheet of grid paper; the matrix is easier to work with when it is all on one sheet of paper. Then, determine which pairs of entity classes are related to each other. Look for data about one thing that is also data about another. For example:

- Customer and Sales Order

A sales order has some data about the customer that placed it, such as customer number, name, address, etc.

- Part and Purchase Order

A purchase order contains some data about the parts being ordered, such as part numbers, descriptions, dimensions, etc.

- Department and Employee

One element of data about an employee is the department to which he/she is assigned, such as department number, name, etc.

- Manufacturing Order and Employee

A manufacturing order has some data about the employees who performed its operations, such as employee numbers, names, etc.

Such sharing of data implies a relationship of some sort. Talk to the sources when additional information about such sharing of data is needed. If a database design is among the source material, the relationships

depicts may be useful. Place an "X" in the matrix at the intersection of each pair of related entity classes.

4. The modeler prepares overview diagrams (FEOs).

These diagrams are intended to show all of the entity and relation classes on just a few pages. Reviewers can usually understand overview diagrams better than individual entity class diagrams, so they will be the primary (or sole) depiction of the model. Each diagram should focus on a particular subject with which the reviewers will be comfortable (e.g., major activities), and each should contain about 10-to-20 entity classes and their relation classes. Use large sheets of paper (e.g., 11x17) and photo-reduction, if necessary.

Every entity and relation class in the matrix must appear in at least one diagram. Use some author convention to signify the entity classes that appear in more than one diagram (e.g., by broadening or double-lining the entity class boxes) and to identify which other diagrams they are in (e.g., by listing the diagram numbers near the entity class boxes). For example, if entity class E27 is in diagrams F1, F3, and F4:

- List F3 and F4 near E27's box on F1.
- List F1 and F4 near E27's box on F3.
- List F1 and F3 near E27's box on F4.

Add the appropriate cardinality and a meaningful label to each relation class as it is drawn in a diagram. Talk to the sources when additional information about a relation class label and cardinality is needed. Cardinalities may be either specific or nonspecific; derived entity classes should not be introduced yet to avoid getting ahead of the reviewers. Check off each relation class in the matrix as it is drawn in a diagram (e.g., by circling the X in the matrix).

5. The modeler defines any additional entity classes that are introduced during this phase.

Whenever a new entity class is introduced, immediately document it by performing the tasks in Phases 1 and 2

that are needed to:

- Update the entity class pool.
- Prepare an Entity Class Definition Form.
- Update the relation class matrix if it has been started.
- Update the overview diagrams if they have been started.

6. The modeler, reviewers, and librarian participate in reviewing the Phase 2 model.

See Phase 1, Task 5 for details.

Deviation from IDEF1:

Usually, individual entity class diagrams are not prepared because the overview diagrams are easier to understand and review, and Relation Class Definition Forms are not filled out because the relation class labels are supposed to be self-descriptive. Also, the Related Entity Class Node Cross-Reference Form is replaced by the specially designed Relation

Classes Form, which is called for in Phase 3. However, the project manager may require the use of any or all of these to supplement the model documentation called for above.

4.2.4 Phase 3: Defining Key Classes

Objectives:

- Refine all nonspecific relation classes in the model.
- Identify the apparent attribute classes that are within the scope of the model.
- Identify and define a key class for each entity class in the model.
- Validate every relation class in the model via key class migration.

This phase is patterned after Phase 3 of IDEF1, and the description presented here is less detailed than the one in the IDEF1 documentation. Please refer to that documentation for further information. Also, please refer to Section 5.2.2.1 for

details on how to fill out the Relation Classes, Owned Attribute Classes, and Inherited Attribute Classes Forms.

Tasks:

1. The project manager decides what method to use to review the Phase 3 model.

See Phase 1, Task 1, for the options and factors to consider.

2. The project manager schedules the Phase 3 tasks.

See Phase 1, Task 2, for details.

3. The modeler refines the nonspecific relation classes.

Introduce a derived entity class for each nonspecific relation class and convert that relation class to a pair of specific relation classes as shown in Figure 4-7 at the end of this section. Assign entity class numbers to the derived entity classes, record them in the entity class pool, and fill out Entity Class Definition Forms. The sources may be able to recommend appropriate names and definitions for some derived entity classes.

Remove the nonspecific relation classes from the relation class matrix and the overview diagrams. Add the derived entity classes and the specific relation classes to the matrix and the diagrams. Retain the same focus for each diagram unless the reviewers suggested a change.

Also, update any optional documents that are affected.

4. The modeler eliminates any unneeded triads or other dual-path structures.

A dual-path structure is one composed of two or more related entity classes in which:

- There are two paths connecting one entity class to another
- One path is a single relation class

- The other path is a series of relation classes (unless the structure has only two entity classes in which case the second path is a single relation class also)

See the examples in Figure 4-8 at the end of this section. Talk to the sources to determine whether the two paths are equal, unequal, or indeterminant. The paths are equal if, for each dependent entity instance, they both lead to the same independent entity instance. The paths are unequal if, for each dependent entity instance, they each lead to a different independent entity instance. The paths are indeterminant if they are equal for some dependent entity instances and unequal for others. If the paths are equal, the single-relation-class path is redundant and must be removed from the relation class matrix and the overview diagrams (and from any optional documents in which appears).

5. The modeler fills out Relation Class Forms.

Record each relation class on a Relation Classes Form. Leave the Ind. K.C. No. column blank for now. As each relation class is recorded on a form, check it off on a copy of each overview diagram in which it appears (e.g., by circling the relation class labels).

6. The modeler builds an attribute class pool.

Examine the entries in the source data list and deduce what sort of characteristic each represents, where a characteristic is a data element that identifies, describes, refers to, etc., a thing being modeled. Each sort of characteristic is represented by an attribute class. Talk to the sources when additional information is needed. Assign an attribute class number to each attribute class, and record it on an Attribute Class Pool Form.

When examining data field names from a database design, realize that several data fields may represent the same kind of "real-world characteristic" or, conversely, that one data field may represent several different characteristics. For example:

- SALES-ORDER-CUSTOMER-NUMBER, INVOICE-CUSTOMER-

NUMBER, and ACCOUNTS-RECEIVABLE-CUSTOMER-NUMBER all represent the same characteristic of a customer, i.e., customer number.

- SALESMAN-ASSIGNMENT-CODE may represent both the territory and the product for which the salesman is responsible.

Also, realize that certain data fields may be present for technical reasons only (e.g., record codes) and should not be included in the attribute class pool.

7. The modeler defines the key classes of the totally independent entity classes.

A totally independent entity class is one that is not dependent in any relation class. Select any one and find the attribute classes in the pool that make up its key class. Watch for attribute class synonyms and homonyms, and handle them like those for entity classes (Phase 1, Task 4). A few totally independent entity classes have two or more alternate key classes (e.g., employees can be uniquely identified by either employee numbers or Social Security Numbers). Be sure to identify all key classes for such an entity class. Also, be sure each key class conforms to the following rules:

- Single-Owned Rule
- Unique-Key Rule
- No-Null Rule
- No-Repeat Rule
- Smallest-Key-Class-Rule

See Section 4.1.1 for explanations of these rules. Define any new entity and relation classes needed to resolve rule violations. See Tasks 11 and 12 for details. Talk to the sources when additional information about a key class is needed.

Assign a key class number to each key class of the entity class (K1 for the first; K2 for the second, if any, etc.) and a tag number to each key class member. Fill out an Owned Attribute Classes Form, and record the key classes in the overview diagrams. Check off each attribute class in the pool as it is used.

8. The modeler migrates the key classes of the totally independent entity classes.

One of the key classes of the entity class from Task 7 must migrate through every relation class in which the entity class is independent. If it has two or more alternate key classes, only one can migrate through each relation class. The same one need not migrate through all of them however; one can migrate through some, another through others. The sources should be able to indicate which key class to use for each relation class. Record the number of the key class that migrates through a relation class in the Ind. K.C. No. column of the Relation Classes Form from Task 7.

Each member of the key class that migrates through a relation class becomes an inherited attribute class in the entity class that is dependent in that relation class. Fill out an Inherited Attribute Classes Form for each dependent entity class, i.e., those listed in the Dep. E.C. No. and Name columns of the Relation Classes Form. Record each inherited attribute class as follows:

- Tag No. column: Assign a new tag number to each inherited attribute class.
- Tag and Label column: Use the name and label of the key class member except in the following two situations:
 - If the key class member migrates through two relation classes into the same dependent entity class, it will appear as two inherited attribute classes, each of which must have a distinct name and label within the entity class. In this case, assign a new name and label to each. See Figure 4-9 at the end of this section for an example.
 - If a new name and label would be more descriptive, they may be used.
- A.C. No. column: Use the attribute class number of the key class member even if a new name and label were assigned.

- Ind. E.C. No. column: Use the number of the entity class that the key class member migrated from.
- Ind. K.C. No. column: Use the key class number of the key class member.
- Ind. Tag No. column: Use the tag number of the key class member.
- Migration Path R.C. Label column: Use the label of the relation through which the key class member migrated.
- Mbr. of K.C. No. column: Leave blank for now.

On copies of the overview diagrams, keep track of which relation classes have been used for key class migration (e.g., by circling the relation class labels).

Repeat Tasks 7 and 8 for each totally independent entity class.

9. The modeler defines the key classes of the remaining entity classes.

The remaining entity classes are those that are not totally independent, i.e., those that are dependent in at least one relation class. Key classes have migrated through some relation classes to appear as inherited attribute classes in some of these entity classes. Some have received all of their inherited attribute classes; others have not. One way to determine whether an entity class has is to examine the copies of the overview diagrams that were used to keep track of key class migration in Task 8. If each relation class in which the entity class is dependent has been used for key class migration, then the entity class has received all of its inherited attribute classes; otherwise, it has not.

Select any one entity class that has received all of its inherited attribute classes, and define its key class(es). The members of its key class(es) may include some of its inherited attribute classes or some

new attribute classes from the pool or both. See Figure 4-10 at the end of this section for guidelines. Handle any synonyms and homonyms in the attribute class pool in the same way as those for entity classes (Phase 1, Task 4). Remember that the entity class may have two or more alternate key classes; be sure to identify all of them. Be sure each key class conforms to the following rules:

- Single-Owner Rule
- Unique-Key Rule
- No-Null Rule
- No-Repeat Rule
- Smallest-Key-Class-Rule

See Section 4.1.1 for explanations of these rules. Define new entity and relation classes needed to resolve rule violations. See Tasks 11 and 12 for details. Talk to the sources when additional information about a key class is needed.

If a key class member comes from the attribute class pool, assign a tag number to it, check it off in the pool, and record it on an Owned Attribute Classes Form. Assign a key class number to each key class (K1 for the first; K2 for the second, if any, etc.), and record it in the Mbr. of K.C. No. column on the Owned Attribute Classes Form or the Inherited Attribute Classes Form where each key class member appears. If an attribute class, either owned or inherited, is a member of more than one key class, record the key class number of each. Also, record the key classes and any nonkey inherited attribute classes in the overview diagrams.

10. The modeler migrates the key classes of the remaining entity classes.

If the entity class from Task 9 is not independent in any relation classes, its key class does not migrate; see the last paragraph of this task. If it is independent in one or more relation classes, record the number of the key class that migrates through each one in the Ind. K.C. No. column of the Relation Classes Form. If the entity class has alternate key classes, record only one key class number for each relation class, although not all relation classes have to get the same number; the sources should be able to indicate

which key class to use for each.

For each entity class that is listed in the Dep. E.C. No. and Name columns of the Relation Classes Form, fill out an Inherited Attribute Classes Form as described in Task 8. Also, as each relation class is used for key class migration, mark it on the overview diagram copies from Task 8.

Repeat Tasks 9 and 10 until key classes for all remaining entity classes have been defined and migrated.

11. The modeler defines any additional entity classes that are introduced during this phase.

Whenever a new entity class is introduced, immediately document it by performing the tasks in Phases 1 - 3 that are needed to:

- Update the entity class pool.
- Prepare an Entity Class Definition Form.
- Update the relation class matrix.
- Define the relation classes in which it is involved. See Task 12 for details.
- Update the overview diagrams.
- Define and migrate its key class(es) at the appropriate time during Tasks 7 - 10.
- Update any optional documents that are affected.

12. The modeler defines any additional relation classes that are introduced during this phase.

Whenever a new relation class is introduced, immediately document it by performing the tasks in Phases 2 and 3 that are needed to:

- Update the relation class matrix.
- Update the overview diagrams.
- Refine it if it is nonspecific.
- Eliminate any unneeded dual-path structures.
- Record it on a Relation Classes Form.
- Validate it via key class migration at the appropriate time during Task 8 or 10.
- Update any optional documents that are affected.

13. The modeler, reviewers, and librarian participate in

reviewing the Phase 3 model.

See Phase 1, Task 5, for details.

Deviation from IDEF1:

The specially designed Relation Classes, Owned Attribute Classes, and Inherited Attribute Classes forms are used in place of the regular IDEF1 forms: Related Entity Class Node Cross-Reference, Attribute Class Definition (2), and Inherited Attribute Class Cross-Reference. The forms used are designed to facilitate loading the conceptual schema. Also, the following IDEF1 forms are not called for, but may be used at the discretion of the project manager:

- Attribute Class Diagram
- Entity Class/Attribute Class Matrix
- Attribute Class Migration Index
- Refinement Alternative Diagram
- Entity Class/Function View Matrix

4.2.5. Phase 4: Defining Nonkey Attribute Classes

Objectives:

- Identify and define the nonkey attribute classes that are within the scope of the model.
- Identify the entity class that owns each nonkey attribute class.

This phase is patterned after Phase 4 of IDEF1, and the description presented here is less detailed than the one in the IDEF1 documentation. Please refer to that documentation for further information. Also, please refer to Section 4.1.3 for details on how to fill out Owned Attribute Classes Forms.

Tasks:

1. The project manager decides what method to use to review the Phase 4 model.

See Phase 1, Task 1, for the options and factors to consider.

2. The project manager schedules the Phase 4 tasks.

See Phase 1, Task 2, for details.

3. The modeler populates the model with the nonkey attribute classes.

The nonkey attribute classes are those that were not used as members of any key classes in Phase 3, i.e., those that have not been checked off in the attribute class pool. Find the entity class that owns each of these according to the following rules:

- Single-Owner Rule
- No-Null Rule
- Full-Functional-Dependency Rule
- No-Transitive-Dependency Rule

See Section 4.1.1 for explanations of these rules. Define any new entity and relation classes needed to resolve any rule violations. See Tasks 4 and 5 for details. Talk to the sources when additional information about a nonkey attribute class is needed.

Assign a tag number to each nonkey attribute class, and record it on an Owned Attribute Classes Form. Check off each in the pool as it is used.

4. The modeler defines any additional entity classes that are introduced during this phase.

Whenever a new entity class is introduced, immediately document it by performing the tasks in Phases 1 - 3 that are needed to:

- Update the entity class pool.
- Prepare an Entity Class Definition Form.
- Update the relation class matrix.
- Define the relation classes that it is involved in. See Task 5 for details.
- Update the overview diagrams.
- Define and migrate its key class(es).
- Update any optional documents that are affected.

5. The modeler defines any additional relation classes that are introduced during this phase.

Whenever a new relation class is introduced, immediately document it by performing the tasks in

Phases 2 and 3 that are needed to:

- Refine it if it is nonspecific.
- Eliminate any unneeded dual-path structures.
- Update the relation class matrix.
- Record it on a Relation Classes Form.
- Update the overview diagrams.
- Validate it via key class migration.
- Update any optional documents that are affected.

6. The modeler, reviewers, and librarian participate in reviewing the Phase 4 model.

See Phase 1, Task 5, for details.

Deviation from IDEF1:

The specially designed Owned Attribute Classes Form is used instead of the regular Attribute Class Definition Forms to facilitate loading the conceptual schema. Also, the following IDEF1 forms are not called for, but may be used at the discretion of the project manager:

- Attribute Class Diagram
- Entity Class/Attribute Class Matrix

See Section 5.2.1 for instructions on how to load.

4.3 Expanding the CS

This section and its subsections describe the procedure for expanding an enterprise's conceptual schema. The procedure is concerned with creating a detailed description (an information model) of a portion of the enterprise's common data, some or all of which is not already described in the conceptual schema, and with collecting the data required to place that description in the CDM database as an additional piece of the conceptual schema (another layer of the onion).

The procedures described in the following subsections correspond to the five IDEF1 phases discussed in the previous section.

4.3.1 Phase 0: Starting the Project

Objectives:

- State the purpose, scope, and viewpoint for the information model.
- Establish the project team.
- Develop a phase-level project schedule.
- Collect and catalog relevant source material.

This phase is patterned after Phase 0 of IDEF1, and the description presented here is less detailed than the one in the IDEF1 documentation. Please refer to that documentation for further information. Also, please refer to Section 4.1.2 for details on how to fill out the Relation Classes, Owned Attribute Classes, and Inherited Attribute Classes forms.

Tasks:

1. The CDM Administrator appoints a project manager.
Usually, this will be the CDM Administrator
2. The project manager states the purpose for building the information model

See Task 2 of Section 4.2.1
3. The project manager states the scope of the information model

See Task 3 of Section 4.2.1
4. The project manager states the viewpoint for the information model

See Task 4 of Section 4.2.1
5. The project manager appoints the project team members

See Task 5 of Section 4.2.1
6. The project manager appoints the acceptance review committee members

This committee should consist of subject experts from the area being modeled and from other, related areas.

7. The project manager schedules the project phases.

See Task 7 of Section 4.2.1.

8. The project manager schedules the remaining Phase 0 tasks.

See Task 8 of Section 4.2.1.

9. The modeler develops a data collection plan.

Determine what kinds of source material are needed and where and how to get that material.

10. The project manager conducts a project kick-off meeting attended by the project team members.

See Task 10 of Section 4.2.1.

11. The modeler collects source material from the sources.

Gather the documents, policies, procedures, database designs, etc., and interview the sources in accordance with the data collection plan (Task 9).

12. The modeler catalogs the source material.

Prepare Source Material Log Forms and Source Data List Forms. If a database design is among the source material, the record names and data field names should be included in the source data list.

13. The modeler examines the existing conceptual schema.

Identify the entity, relation, and attribute classes in the existing conceptual schema that appear to be within the scope of the model. Fill out the following forms from the descriptions in the conceptual schema.

- Entity Class Definition Form
- Relation Classes Form
- Owned Attribute Classes Form
- Inherited Attribute Classes Form
- Relation Class Matrix Form

To distinguish these elements of the conceptual schema from the new ones that will be documented during the course of this modeling project, prefix all of the identification numbers with the letter "C." For example:

- Entity Class Number = CE12
- Attribute Class Number = CA94
- Tag Number = CT156
- Key Class Number = CK1

14. The modeler explains any author conventions.

These are deviations from or additions to the regular IDEF1 methodology. Mention the use of the three specially designed modeling forms: Relation Classes Form, Owned Attribute Classes Form, and Inherited Attribute Classes Form. Also, explain that in order to distinguish between model elements that are already in the conceptual schema and those that are not, the identification numbers of the former will be prefixed with the letter "C" for conceptual while those of the latter will be prefixed with the letter "N" for new.

Deviation from IDEF1:

Usually, kits are not used to accomplish the review of the Phase 0 model documentation; the essentials are reviewed during the kick-off meeting (Task 10). However, the project manager may require that kits be used to supplement or replace the kick-off meeting.

4.3.2 Phase 1: Defining Entity Classes

Objective:

- Identify and define the apparent entity classes that are within the scope of the model.

This phase is patterned after Phase 1 of IDEF1, and the description presented here is less detailed than the one in the IDEF1 documentation. Please refer to that documentation for further information.

Tasks:

1. The project manager decides what method to use to review the Phase 1 model.

See Task 1 of Section 4.2.2.

2. The project manager schedules the Phase 1 tasks.

See Task 2 of Section 4.2.2.

3. The modeler builds an entity class pool.

Examine the entries in the source data list and deduce what sort of thing each entry identifies, describes, refers to, etc. For example:

- Employee number, name, birth date, and salary are data elements about an employee; hence, an "Employee" entity class.
- Part number, description, and dimensions are all about a part; hence, a "Part" entity class.

Each sort of thing is represented by an entity class. Determine whether any of these entity classes are already in the conceptual schema and, if so, whether modeling forms were prepared for them in Phase 0, Task 13. Rely on the entity class definitions more than the names or labels in deciding whether a conceptual schema entity class represents the same sort of thing as an entity class deduced from the source data list. If any entity class is in the conceptual schema, but modeling forms were not prepared, prepare them now; see Phase 0, Task 13 for details. Talk to the sources when additional information is needed. The entity instances within an entity class should be distinguishable from one another by some unique identifier. Assign an entity class number, prefixed with "N," to each new entity class, and record them on an Entity Class Pool Form. Do not record any conceptual schema entity classes in the pool.

When examining record names from a database design, be careful to think about the "real-world thing" that each kind of record represents. Realize that several kinds of records may represent the same thing or, conversely, that one kind of record may represent several different things. Also, realize that certain kinds of records

may be present for technical reasons only (performance, backup/recovery, etc.). Such records do not represent "real-world things" and should not result in entity classes being added to the pool.

4. The modeler defines each entity class.

See Task 4 of Section 4.2.2.

Also, review the names, labels, and definitions of the conceptual schema entity classes, record any changes that are required on the Entity Class Definition Forms, and write "UPDATED" below the entity class number in the lower left corner.

5. The modeler, reviewers, and librarian participate in reviewing the Phase 1 model.

The method of review was selected in Task 1. The modelers prepare the review materials (kits or walk-through handouts), the reviewers read and comment on the materials, and the modelers respond to the comments. If kits are used, the librarian coordinates their circulation.

6. The CDM Administrator reviews the model to ensure that it is compatible with the conceptual schema.

Definitions are compared to see whether any entity, relation, or attribute classes that are identified as new in the model are really the same as those that are already in the conceptual schema, possibly with different names or labels. Also, each proposed conceptual schema update is evaluated to gauge its impact on the existing CS/ES and CS/IS mappings.

4.3.3. Phase 2: Defining Relation Classes

Objective:

- Identify and define the apparent relation classes that are within the scope of the model.

This phase is patterned after Phase 2 of IDEF1, and the description presented here is less detailed than the one in the IDEF1 documentation. Please refer to that documentation for further information.

Tasks:

1. The project manager decides what method to use to review the Phase 2 model.

See Phase 1, Task 1, for the options and factors to consider.

2. The project manager schedules the Phase 2 tasks.

See Phase 1, Task 2, for details.

3. The modeler builds a relation class matrix.

See Task 3 of Section 4.2.3.

4. The modeler prepares overview diagrams (FEOs).

See Task 4 of Section 4.2.3.

5. The modeler defines any additional entity classes that are introduced during this phase.

Whenever a new entity class is introduced, double-check the conceptual schema to see if it is already there. Rely on the entity class definitions more than the names or labels in deciding whether a conceptual schema entity class represents the same sort of thing as a new entity class. If a new entity class is already described in the conceptual schema, prepare the modeling forms listed in Phase 0, Task 13. If it is not, immediately document it by performing the tasks in Phases 1 and 2 that are needed to:

- Update the entity class pool
- Prepare an Entity Class Definition Form
- Update the relation class matrix if it has been started
- Update the overview diagrams if they have been started

6. The modeler, reviewers, and librarian participate in reviewing the Phase 2 model

See Task 5 of this section for details.

Deviation from IDEF1:

Usually, individual entity class diagrams are not prepared because the overview diagrams are easier to understand and review, and Relation Class Definition Forms are not filled out because the relation class labels are supposed to be self-descriptive. Also, the Related Entity Class Node Cross-Reference Form is replaced by the specially designed Relation Classes Form, which is called for in Phase 3. However, the project manager may require the use of any or all of these to supplement the model documentation called for above.

4.3.4 Phase 3: Defining Key Classes

Objectives:

- Refine all nonspecific relation classes in the model.
- Identify the apparent attribute classes that are within the scope of the model.
- Identify and define a key class for each entity class in the model.
- Validate every relation class in the model via key class migration.

This phase is patterned after Phase 3 of IDEF1, and the description presented here is less detailed than the one in the IDEF1 documentation. Please refer to that documentation for further information. Also, please refer to Section 4.1.3 for details on how to fill out the Relation Classes, Owned Attribute Classes, and Inherited Attribute Classes Forms.

Tasks:

1. The project manager decides what method to use to review the Phase 3 model.

See Task 1 of Section 4 2 1.
2. The project manager schedules the Phase 3 tasks

See Task 2 of Section 4 2 1
3. The modeler refines the nonspecific relation classes.

Introduce a derived entity class for each nonspecific relation class and convert that relation class to a pair of specific relation classes as shown in Figure 4-7 at the end of this section. Assign entity class numbers, prefixed with "N," to the derived entity classes, record them in the entity class pool, and fill out Entity Class Definition Forms. The sources may be able to recommend appropriate names and definitions for some derived entity classes.

Remove the nonspecific relation classes from the relation class matrix and the overview diagrams. Add the derived entity classes and the specific relation classes to the matrix and the diagrams. Retain the same focus for each diagram unless the reviewers suggested a change. Also, update any optional documents that are affected.

4. The modeler eliminates any unneeded triads or other dual-path structures.

A dual-path structure is one composed of two or more related entity classes in which:

- There are two paths connecting one entity class to another
- One path is a single relation class
- The other path is a series of relation classes (unless the structure has only two entity classes in which case the second path is a single relation class also)

See the examples in Figure 4-8 at the end of this section. Talk to the sources to determine whether the two paths are equal, unequal, or indeterminant. The paths are equal if, for each dependent entity instance, they both lead to the same independent entity instance. The paths are unequal if, for each dependent entity instance, they each lead to a different independent entity instance. The paths are indeterminant if they are equal for some dependent entity instances and unequal for others. If the paths are equal, the single-relation-class path is redundant and must be removed from the model, i.e., from the relation class matrix and the overview

diagrams (and from any optional documents in which it appears).

If the relation class that must be removed is already described in the conceptual schema, it should already be listed on a Relation Classes Form from Phase 0, Task 13. Write "DELETE" in the margin next to it and write "UPDATED" below the entity class number in the lower left corner.

If the dependent entity class in that relation class is from the conceptual schema, the inherited attribute classes that it received via key class migration through that relation class must be removed also. Write "DELETE" in the margin next to each one on the Inherited Attribute Classes Form, and write "UPDATED" below the entity class number in the lower left corner. If any of them is a key class member in the dependent entity class, that key class is now incomplete and must be removed; see Task 13 for details.

5. The modeler fills out Relation Class Forms.

See Task 5 of Section 4.2.4.

6. The modeler builds an attribute class pool.

Examine the entries in the source data list and deduce what sort of characteristic each represents, where a characteristic is a data element that identifies, describes, or refers to, a thing being modeled. Each sort of characteristic is represented by an attribute class. Determine whether any of the attribute classes are already in the conceptual schema and, if so, whether modeling forms were prepared for them in Phase 0, Task 13. Rely on the attribute class definitions more than the names or labels in deciding whether a conceptual schema attribute class represents the same sort of characteristic as an attribute class deduced from the source data list. If an attribute class is in the conceptual schema, but modeling forms were not prepared, prepare them now; see Phase 0, Task 13, for details. Talk to the sources when additional information is needed. Assign an attribute class number, prefixed with "N," to each new characteristic deduced from the source data list, and record them on Attribute Class Pool Forms.

When examining data field names from a database design, realize that several data fields may represent the same kind of "real-world characteristic" or, conversely, that one data field may represent several different characteristics. For example:

- SALES-ORDER-CUSTOMER-NUMBER, INVOICE-CUSTOMER-NUMBER, and ACCOUNTS-RECEIVABLE-CUSTOMER-NUMBER all represent the same characteristic of a customer, i.e., customer number.
- SALESMAN-ASSIGNMENT-CODE may represent both the territory and the product for which the salesman is responsible.

Also, realize that certain data fields may be present for technical reasons only (e.g., record codes) and should not be included in the attribute class pool.

7. The modeler defines the key classes of the totally independent entity classes.

A totally independent entity class is one that is not dependent in any relation classes. Select any one and find the attribute classes in the pool that make up its key class. If the entity class is already in the conceptual schema, at least one key class has already been defined for it. However, others may be discovered here because of new owned attribute classes. Watch for attribute class synonyms and homonyms, and handle them like those for entity classes (Phase 1, Task 4). A few totally independent entity classes have two or more alternate key classes (e.g., employees can be uniquely identified by either Social Security or employee numbers). Be sure to identify all key classes for such an entity class. Also, be sure each key class conforms to the following rules:

- Single-Owned Rule
- Unique-Key Rule
- No-Null Rule
- No-Repeat Rule
- Smallest-Key-Class-Rule

See Section 4.1 for explanations of these rules. Define any new entity and relation classes needed to

resolve rule violations. See Tasks 11 and 12 for details. If an attribute class that is needed as a key class member for a new entity class is already owned by a conceptual schema entity class, a relationship exists between those two entity classes. If it is not already documented as a new relation class, it must be before the key class of the new entity class can be defined; see Task 12 for details. If the new entity class is dependent in the new relation class, it is no longer totally independent, so its key class cannot be defined until Task 9. If the new entity class is independent in the relation class, the ownership of the attribute class must be changed; it is owned by the new entity class, not by the one in the conceptual schema. Record it on an Owned Attribute Classes Form for the new entity class, using the same name, label, definition, domain (type and size), and attribute class number, prefixed with "C," but assign a new tag number, prefixed with "N." Write "DELETE" in the margin next to the attribute class on the form for the conceptual schema entity class and write "UPDATED" below the entity class number in the lower left corner. If it is a key class member in the conceptual schema entity class, that key class is now incomplete and must be removed; see Task 13 for details. Talk to the sources when additional information about a key class is needed.

Assign a key class number, prefixed with "N," to each new key class of the entity class (NK1 for the first; NK2 for the second, if any, etc.). Assign a tag number, prefixed with "N," to each new attribute class that is a key class member; record it on an Owned Attribute Classes Form, and check it off in the attribute class pool. Record the key classes, both new ones and ones from the conceptual schema, in the overview diagrams.

Also, review the name, label, and definition of each conceptual schema attribute class that is a key class member; record any changes that are required on the Owned Attribute Classes Form where it appears, write "CHANGE" in the margin next to it, and write "UPDATED" below the entity class number in the lower left corner

8. The modeler migrates the key classes of the totally

independent entity class.

One of the key classes of the entity class from Task 7 must migrate through every relation class in which the entity class is independent. A key class has already migrated through every conceptual schema relation class, but some may have had that migration undone in Task 13, i.e., those with a circled key class number in the Ind. K.C. No. column of a Relation Classes Form and with "OMIT" written in the margin. Only these and the new relation classes, i.e., those without a key class number in that column, need to be considered here. If the entity class has two or more alternate key classes, only one can migrate through each relation class. The same one need not migrate through all of them, however, one can migrate through some, another through others. The sources should be able to indicate which key class to use for each relation class. For a new relation class, record the key class number in the Ind. K.C. No. column of the Relation Classes Form from Task 7. For a conceptual schema relation class that is having its key class migration redone, if the key class number is the same as the one that is already in Ind. K.C. No. column, erase the circle around it and erase "OMIT" in the margin. If the key class numbers are different, replace the circled one with the new one and change "OMIT" to "CHANGE" in the margin.

Each member of the key class that migrates through a relation class becomes an inherited attribute class in the entity class that is dependent in that relation class. Fill out an Inherited Attribute Classes Form for each dependent entity class, i.e., those listed in the Dep. E.C. No. and Name columns of the Relation Classes Form. If the dependent entity class is already in the conceptual schema, use the Inherited Attribute Classes Form that was prepared in Phase 0, Task 13. Record each new inherited attribute class as follows:

- Tag No. column: Assign a new tag number, prefixed with "N", to each inherited attribute class. If an inherited attribute class replaces an owned attribute class whose ownership was changed in Task 8, use the tag number

prefixed with "C" that was assigned to that owned attribute class, and change "DELETE" to "NEW OWNER" in the margin next to that owned attribute class on the Owned Attribute Classes Form.

- Tag and Label column: Use the name and label of the key class member except in the following two situations:
 - If the key class member migrates through two relation classes into the same dependent entity class, it will appear as two inherited attribute classes, each of which must have a distinct name and label within the entity class. In this case, assign a new name and label to each. See Figure 4-9 at the end of this section for an example.
 - If a new name and label would be more descriptive, they may be used.
- A.C. No. column: Use the attribute class number of the key class member, even if a new name and label were assigned.
- Ind. E.C. No. column: Use the number of the entity class from which the key class member migrated.
- Ind. K.C. No. column: Use the key class number of the key class member.
- Ind. Tag No. column: Use the tag number of the key class member.
- Migration Path R.C. Label column: Use the label of the relation class through which the key class member migrated.
- Mbr. of K.C. No. column: Leave blank for now.

If an inherited attribute class that was removed from a conceptual schema entity class in Task 4 or 13 is being reestablished, do not record it as described above. Instead, reuse the one that is already recorded on the Inherited Attribute Classes Form.

Erase "DELETE" from the margin. If any of the values in the following columns need to be changed, replace them with the new values and write "CHANGE" in the margin:

- Tag and Label Column
- Ind. E.C. No. Column
- Ind. K.C. No. Column
- Ind. Tag No. Column
- Migration Path R.C. Label Column

If the Mbr. of K.C. No. column contains any key class numbers, circle each and write "OMIT" in the margin.

On copies of the overview diagrams, keep track of which relation classes have been used for key class migration, including those from the conceptual schema that had already been used (e.g., by circling the relation class labels).

Repeat Tasks 7 and 8 for each totally independent entity class.

9. The modeler defines the key classes of the remaining entity classes.

The remaining entity classes are those that are not totally independent, i.e., those that are dependent in at least one relation class. Key classes have migrated through some relation classes to appear as inherited attribute classes in some of these entity classes. Some have received all of their inherited attribute classes; others have not. One way to determine whether an entity class has is to examine the copies of the overview diagrams that were used to keep track of key class migration in Task 8. If each relation class that the entity class is dependent in has been used for key class migration, then the entity class has received all of its inherited attribute classes; otherwise it has not.

Select any one entity class that has received all of its inherited attribute classes, and define its key class(es). If the entity class is already in the conceptual schema, at least one key class has already been defined for it. However, if one was removed in Task 13, it must be reestablished or a new one must

be defined. Also, other key classes may be discovered here because of new owned or inherited attribute classes. The members of its key class(es) may include some of its inherited attribute classes or some of the new attribute classes from the pool or both. See Figure 4-10 at the end of this section for guidelines. Handle any synonyms and homonyms in the attribute class pool in the same way as those for entity classes (Phase 1 Task 4). Remember that the entity class may have two or more alternate key classes. Be sure to identify all of them. Be sure each key class conforms to the following rules:

- Single-Owner Rule
- Unique-Key Rule
- No-Null Rule
- No-Repeat Rule
- Smallest-Key-Class-Rule

See Section 4.1 for explanations of these rules. Define new entity and relation classes needed to resolve rule violations. See Tasks 11 and 12 for details. If an attribute class that is needed as a key class member for a new entity class is already owned by a conceptual schema entity class, a relationship exists between those two entity classes. If it is not already documented as a new relation class, it must be before the key class of the new entity class can be defined. See Task 12 for details. If the new entity class is dependent in the relation class, its key class cannot be defined until one from the independent entity class has migrated through the new relation class. If the new entity class is independent in the relation class, the ownership of the attribute class must be changed; it is owned by the new entity class, not by the one in the conceptual schema. Record it on an Owned Attribute Classes Form for the new entity class, using the same name, label, definition, domain (type and size), and attribute class number, prefixed with "C," but assign a new tag number, prefixed with "N." Write "DELETE" in the margin next to the attribute class on the form for the conceptual schema entity class and write "UPDATED" below the entity class number in the lower left corner. If it is a key class member in the conceptual schema entity class, that key class is now incomplete and must be removed; see Task 13 for

details. Talk to the sources when additional information about a key class is needed.

Assign a key class number, prefixed with "N," to each new key class (NK1 for the first, NK2 for the second, if any, etc.). If a key class that was removed in Task 13 is being reestablished, reuse its original key class number, prefixed with "C." Assign a tag number, prefixed with "N," to each new key class member that comes from the attribute class pool, check it off in the pool, and record it on an Owned Attribute Classes Form.

Also, review the name, label, and definition of each conceptual schema attribute class that is a key class member, record any changes that are required on the Owned Attribute Classes Form where it appears, write "CHANGE" in the margin next to it, and write "UPDATED" below the entity class number in the lower left corner.

Identify each new key class member by recording its key class number in the Mbr. of K.C. No. column on either the Owned Attribute Classes Form or the Inherited Attribute Classes Form. If an attribute class, either owned or inherited, is a member of more than one key class, record the key class number of each. If an attribute class is being reestablished as a member of a key class that was removed in Task 13, erase the circle around the key class number in the Mbr. of K.C. No. column of the Owned or Inherited Attribute Classes Form and erase "OMIT" from the margin. Also, record the key classes and any nonkey inherited attribute classes, both new ones and ones from the conceptual schema, in the overview diagrams.

10. The modeler migrates the key classes of the remaining entity classes.

If the entity class from Task 9 is not independent in any relation classes, its key class does not migrate; see the last paragraph of this task. If it is independent in one or more relation classes, one of its key classes must migrate through each. A key class has already migrated through every conceptual schema relation class, but some may have had that

migration undone in Task 13, i.e., those with a circled key class number in the Ind. K.C. No. column of a Relation Classes Form and with "OMIT" written in the margin. Only these and the new relation classes, i.e., those without a key class number in that column, need to be considered here. Record the number of the key class that migrates through each new relation class in the Ind. K.C. No. column of the Relation Classes Form. If the entity class has alternate key classes, record only one key class number for each relation class, although not all relation classes have to get the same number; the sources should be able to indicate which key class to use for each. For a conceptual schema relation class that is having its key class migration redone, if the key class number is the same as the one that is already in Ind. K.C. No. Column, erase the circle around it and erase "OMIT" in the margin. If the key class numbers are different, replace the circled one with the new one and change "OMIT" to "CHANGE" in the margin.

For each entity class that is listed in the Dep. E.C. No. and Name columns of the Relation Classes Form, fill out an Inherited Attribute Classes Form as described in Task 8. Also, keep track of which relation classes have been used for key class migration, including those from the conceptual schema, by marking them on the overview diagram copies from Task 8.

Repeat Tasks 9 and 10 until key classes for all remaining entity classes have been defined and migrated.

11. The modeler defines any additional entity classes that are introduced during this phase.

Whenever a new entity class is introduced, double-check the conceptual schema to see if it is already there. Rely on the entity class definitions more than the names or labels in deciding whether a conceptual schema entity class represents the same sort of thing as a new entity class. If a new entity class is already described in the conceptual schema, prepare the modeling forms listed in Phase 0, Task 13. If it is not, immediately document it by

performing the tasks in Phases 1 - 3 that are needed to:

- Update the entity class pool.
 - Prepare an Entity Class Definition Form.
 - Update the relation class matrix.
 - Define the relation classes in which it is involved. See Task 12 for details.
 - Update the overview diagrams.
 - Define and migrate its key class(es) at the appropriate time during Tasks 7 - 10.
 - Update any optional documents that are affected.
12. The modeler defines any additional relation classes that are introduced during this phase.
- See Task 12 of Section 4.2.4.
13. The modeler removes any incomplete key classes and all resulting inherited attribute classes.

Either the removal of a relation class that is already in the conceptual schema (Task 4) or the change in ownership of an attribute class that is already in the conceptual schema (Tasks 7 and 9) can cause a key class member to be removed from a conceptual schema entity class, either temporarily (until Task 8 or 10) or permanently. When this happens, the key class that lost the member becomes incomplete, so it can no longer fulfill its function. Consequently, it must be removed also. The other attribute classes that are members of that key class, if any, can remain in the entity class, but their membership in that key class must be removed. Circle the key class number in the Mbr. of K.C. No. column on the Owned or Inherited Attribute Classes Form where each member appears, write "OMIT" in the margin next to it, and write "UPDATED" below the entity class number in the lower left corner.

If the key class migrated to other conceptual schema entity classes, that migration must be undone. Circle the key class number in the Ind. K.C. No. column of the Relation Classes Form for each relation class that is affected, write "OMIT" in the margin next to each, and write "UPDATED" below the entity class number in the lower left corner. If any of the

affected dependent entity classes are not already in the model, add them now: see Phase 0, Task 13 for details. Write "DELETE" in the margin of the Inherited Attribute Classes Forms next to each inherited attribute class that resulted from the migration of that key class, and write "UPDATED" below the entity class number in the lower left corner.

If any of these inherited attribute classes is a key class member itself, this task must be repeated, and it must continue to be repeated until all key classes and all inherited attribute classes that are contingent on the original key class have been marked for removal. Key classes and inherited attribute classes of all affected entity classes will be reestablished in Tasks 8 - 10, but they may not be exactly the same.

14. The modeler, reviewers, and librarian participate in reviewing the Phase 3 model.

See Task 5 of Section 4.2.4.

Deviation from IDEF1:

The specially designed Relation Classes, Owned Attribute Classes, and Inherited Attribute Classes Forms are used in place of the following regular IDEF1 forms: Related Entity Class Node Cross-Reference, Attribute Class Definition (2), and Inherited Attribute Class Cross-Reference. The forms used are designed to facilitate loading the conceptual schema. Also, the IDEF1 forms listed below are not called for, but may be used at the discretion of the project manager:

- Attribute Class Diagram
- Entity Class/Attribute Class Matrix
- Attribute Class Migration Index
- Refinement Alternative Diagram
- Entity Class/Function View Matrix

4.3.5 Phase 4: Defining Nonkey Attribute Classes

Objectives:

- Identify and define the nonkey attribute classes that are within the scope of the model.

- Identify the entity class that owns each nonkey attribute class

This phase is patterned after Phase 4 of IDEF1, and the description presented here is less detailed than the one in the IDEF1 documentation. Please refer to that documentation for further information. Also, please refer to Section 4.1.3 for details on how to fill out Owned Attribute Classes Forms.

Tasks:

1. The project manager decides what method to use to review the Phase 4 model.

See Task 1 of Section 4.2.1.

2. The project manager schedules the Phase 4 tasks.

See Task 2 of Section 4.2.1.

3. The modeler populates the model with the nonkey attribute classes.

See Task 3 of section 4.2.1.

Assign a tag number, prefixed with "N," to each nonkey attribute class, and record it on an Owned Attribute Classes Form. Check off each in the pool as it is used.

Also, review the name, label, and definition of each conceptual schema attribute class, record any changes that are required on the Owned Attribute Classes Form where it appears, write "CHANGE" in the margin next to it, and write "UPDATED" below the entity class number in the lower left corner.

4. The modeler defines any additional entity classes that are introduced during this phase.

Whenever a new entity class is introduced, double-check the conceptual schema to see if it is already there. Rely on the entity class definitions more than the names or labels in deciding whether a conceptual schema entity class represents the same sort of thing as a new entity class. If a new entity class

is already described in the conceptual schema, prepare the modeling forms listed in Phase 0, Task 13. If it is not, immediately document it by performing the tasks in Phases 1-3 that are needed to:

- Update the entity class pool.
- Prepare an Entity Class Definition Form.
- Update the relation class matrix.
- Define the relation classes that it is involved in. See Task 5 for details.
- Update the overview diagrams.
- Define and migrate its key class(es).
- Update any optional documents that are affected.

5. The modeler defines any additional relation classes that are introduced during this phase.

See Task 5 of Section 4.2.1.

6. The modeler, reviewers, and librarian participate in reviewing the Phase 4 model.

See Task 5 of Section 4.2.1.

Deviation from IDEF1:

The specially designed Owned Attribute Classes Form is used instead of the regular Attribute Class Definition Forms to facilitate loading the conceptual schema. Also, the following IDEF1 forms are not called for, but may be used at the discretion of the project manager:

- Attribute Class Diagram
- Entity Class/Attribute Class Matrix

See Section 5.2 for instructions on how to update the CS tables.

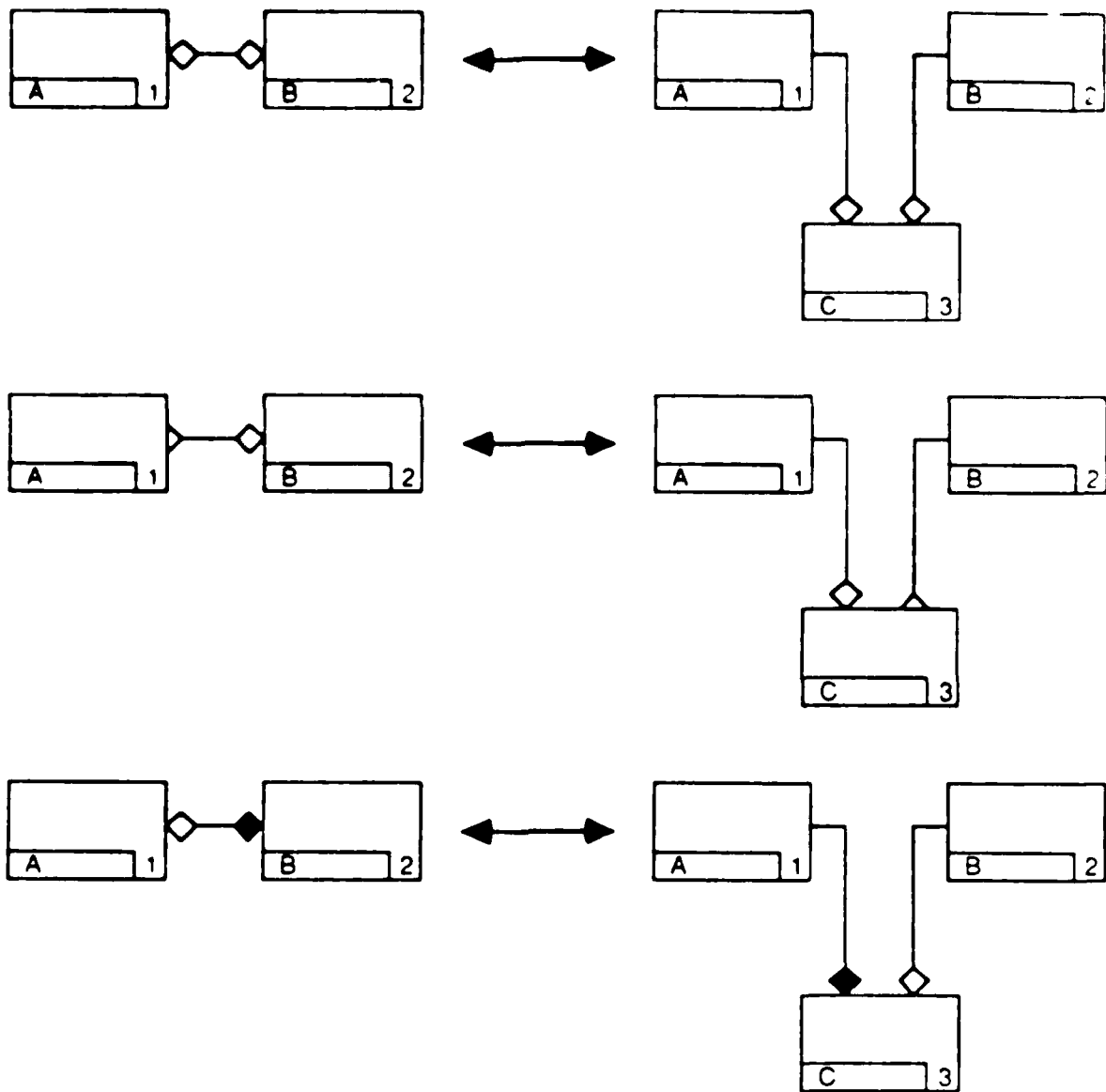


Figure 4-7. Refinements of Nonspecific Relation Classes Example

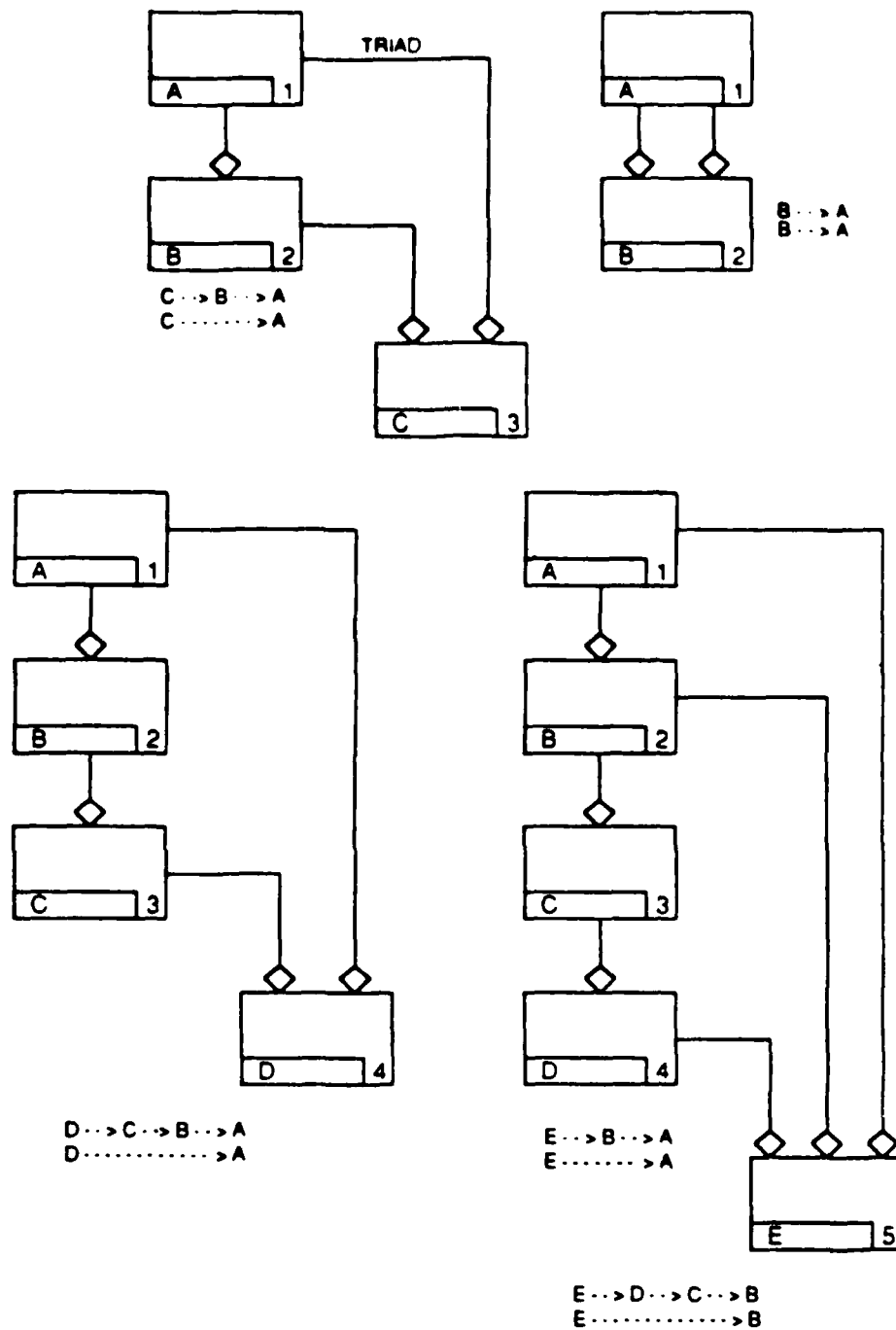


Figure 4-8. Triads and Other Dual-Path Structure Examples

Part Number is the key class of Part. It migrates through each relation class to appear twice in Component Part. The inherited attribute class that results from the left relation class could be named "Assembly Part Number" and the one from the right could be called "Component Part Number" to associate each with the appropriate relation class.

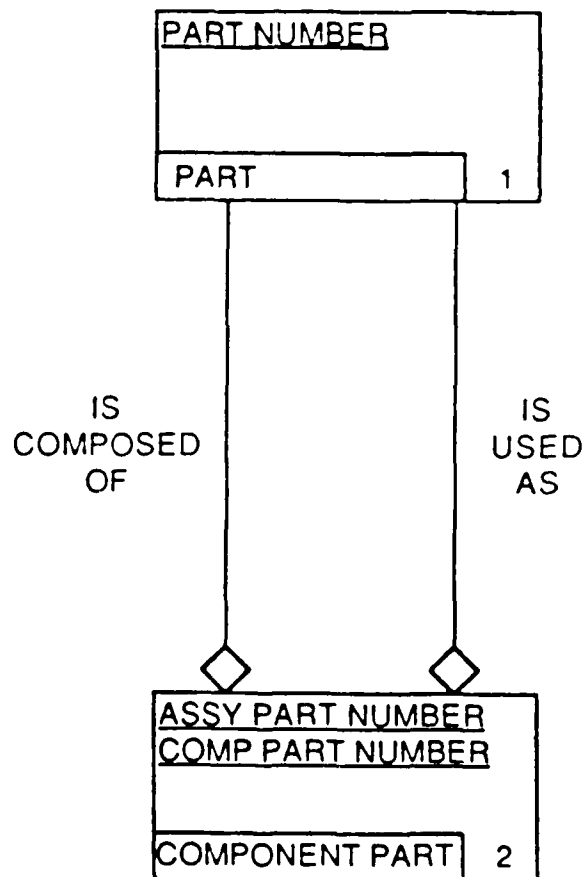


Figure 4-9. Migration Through Two Relation Classes Example

- A. In a one-to-zero-or-one relation class the key class of the dependent is usually the same as that of the independent:

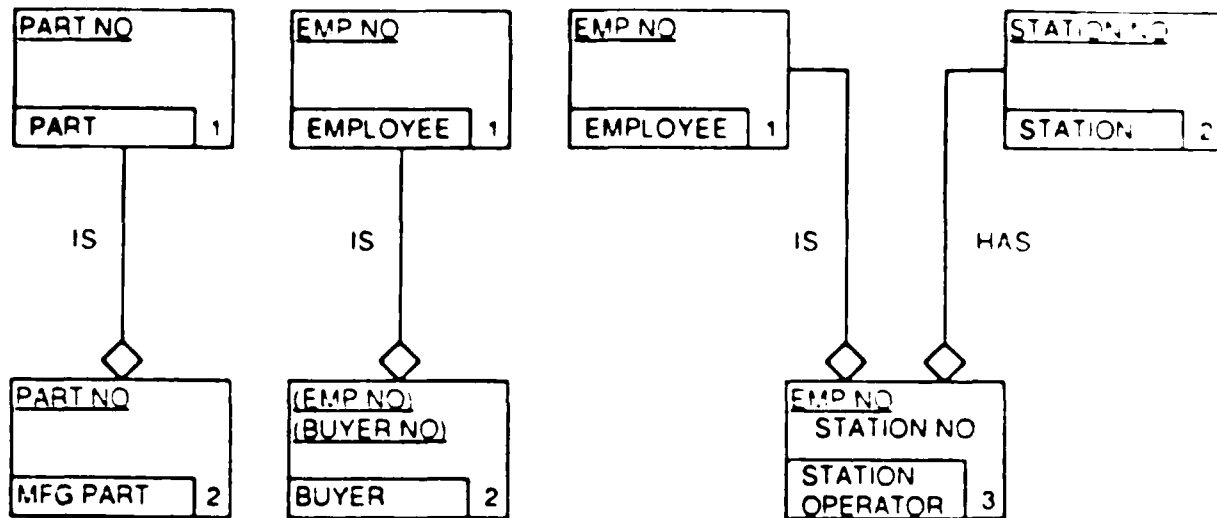


Figure 4-10. Guidelines for Determining Key Classes of Dependent Entity Classes

- B. The key class of an entity class that was derived to refine a many-to-many relation class is usually composed of attribute classes inherited from the two independent entity classes:

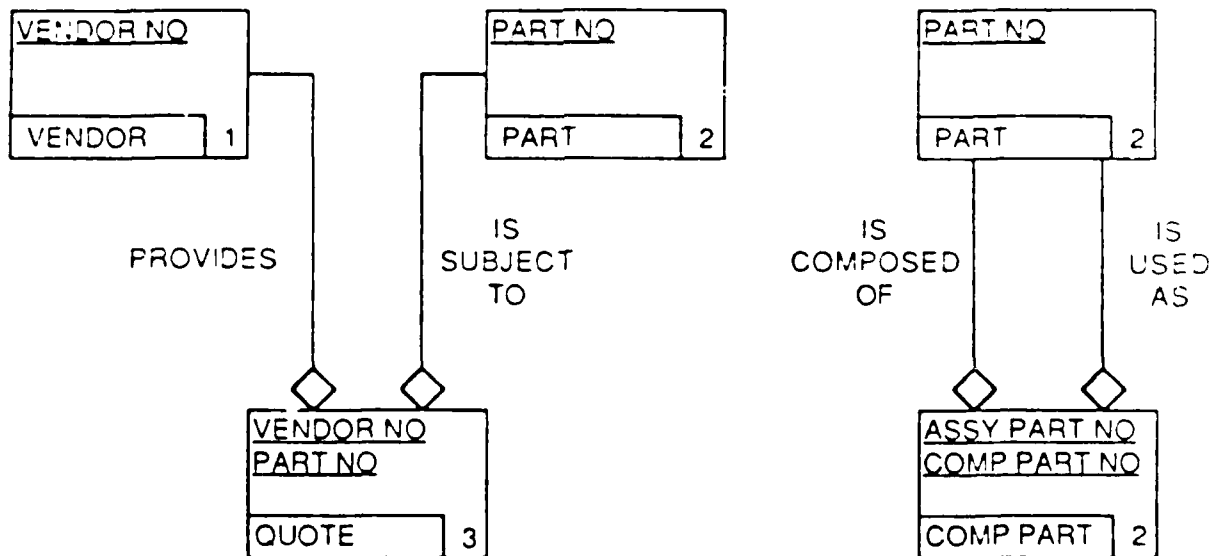


Figure 4-10. Guidelines for Determining Key Classes of Dependent Entity Classes (Continued)

- C. In this example, Bin Wrhs No and Item Wrhs No always have the same value so only one must be in the key.

- D. In this example, Proj Plan No and Tool Plant No do not always have the same value, so both must be in the key class.

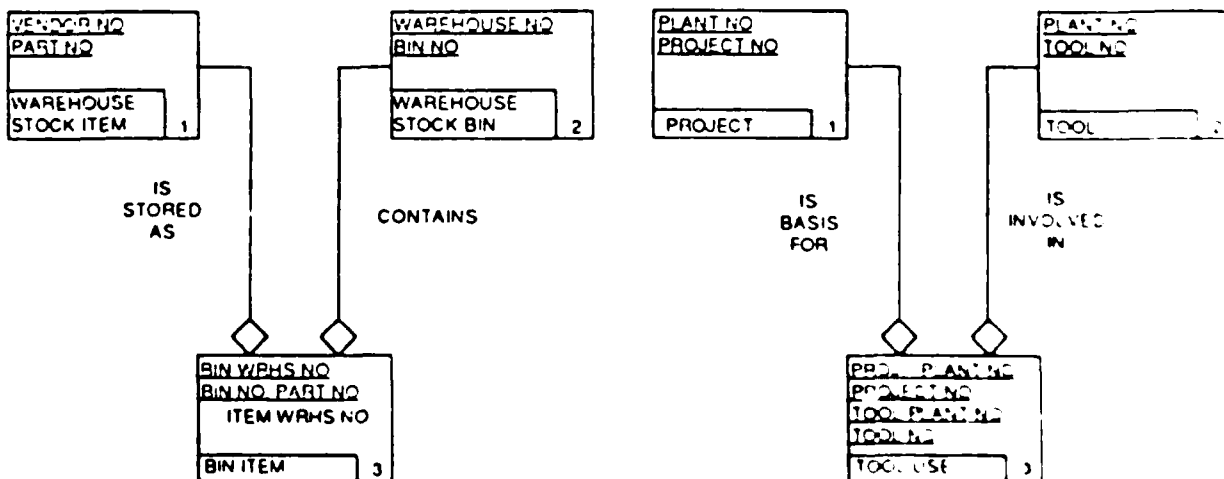


Figure 4-10. Guidelines for Determining Key Classes of Dependent Entity Classes (Continued)

AD-A181 577

INTEGRATED INFORMATION SUPPORT SYSTEM (IIS) VOLUME 3

2/4

COMMON DATA MODEL 5 (U) GENERAL ELECTRIC CO

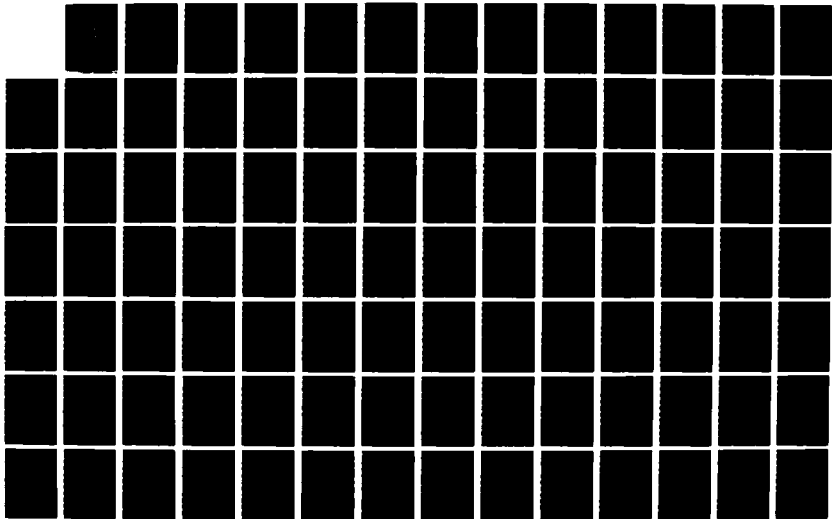
SCHENECTADY NY PRODUCTION RESOURCES CONSU

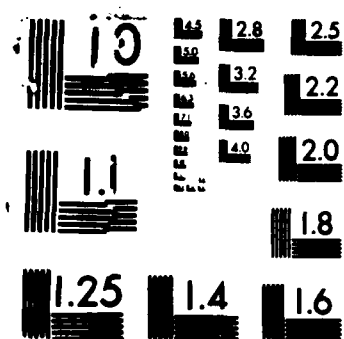
D ROLLINS ET AL 01 NOV 85 U--620141001

F/G 5/2

NL

UNCLASSIFIED





SECTION 5

MAINTAINING THE CDM

5.1 Methodology Overview

The CDM database can be maintained by either or both of the following two methods:

1. Using the NDDL Commands
2. Directly loading the CDM Tables

Direct loading of the CDM Tables is discussed in Section 5.1.2. The use of the NDDL is the recommended approach and is discussed in the next section.

5.1.1 Using NDDL with the CDM Tables

The Neutral Data Definition Language, hereafter NDDL, is an interpretive language that was developed to populate and maintain the CDM database. Figure 5-1 contains a list of NDDL commands.

As detailed instructions on the use of NDDL are provided in the NDDL User's Guide UM620141100, this manual will not describe NDDL syntax and only references NDDL commands, within the context of the methodology.

NDDL Commands

Alter Alias	Define Database
Alter Attribute	Define Record
Alter Domain	Define Set
Alter Entity	Describe
Alter Map	Drop Alias
Alter Model	Drop Attribute
Alter Relation	Drop Database
Check Model	Drop Domain
Combine Entity	Drop Entity
Compare Model	Drop Field
Copy Attribute	Drop Keyword
Copy Description	Drop Map
Copy Entity	Drop Model
Copy Model	Drop Record
Create Alias	Drop Relation
Create Attribute	Drop Set
Create Domain	Drop View
Create Entity	Halt
Create Map	Merge Model
Create Model	Rename
Create Relation	
Create View	

Figure 5-1. NDDL Commands

5.1.2 Direct Loading of the CDM Tables

This section explains how to load each of the following tables in the conceptual schema portion of the CDM database:

Attribute Class Table

Attribute Use Class Table

Data Type Table

Entity Class Table

Inherited Attribute Use Class Table

Key Class Table

Key Class Member Table

Relation Class Table

The following paragraphs are arranged alphabetically by table name.

5.1.2.1 Attribute Class Table

Source Documents:

Owned Attribute Classes forms from the IDEF1 model.

Instructions:

Create one table entry for each page entry.

<u>Table Field</u>	<u>Source Field</u>
AC No	A.C. No. column. Use the number following the "A"; do not include the "A" itself.
AC Name	A.C. Name & Label column. Use the noun phrase that is not in parentheses.

<u>Table Field</u>	<u>Source Field</u>
AC Label	A.C. Name & Label column. Use the noun phrase that is in parentheses; do not include the parentheses themselves. If there is no noun phrase in parentheses, use the A.C. Name noun phrase.
EC No	Node (lower left corner). Use the number following the "E"; do not include the "E" itself.
Type ID	Type ID column. Use the letter to the left of the parenthesis.
Max Att Value Len	Type ID column. Use the number that is in parentheses. If there is a decimal point in the number, only use the portion to the left (i.e., the integer portion).
Max Att Dec Len	Type ID column. If there is a decimal point in the number in parentheses, use the portion to the right (i.e., the decimal portion). Otherwise, leave blank.

Example:

From the sample form in Figure 5-2, the resulting Attribute Class Table would be as follows:

AC No	AC Name	AC Label	EC No	Type ID	Max Att Value Len	Max Att Dec Len
--	-----	-----	--	----	-----	----
24	Item Identification	Item ID	8	C	10	
25	Item Description	Item Desc	8	C	50	
26	Item Name	Item Name	8	C	24	
27	Item Length	Item Length	8	N	8	2
28	Item Quantity	Item Qty	8	N	6	

UM 620141001
1 November 1985

USED AT		AUTHOR PROJECT		DATE REV.		WORKING DRAFT		RECOMMENDED PUBLICATION		DATE		CONTEXT		
		NOTES 1 2 3 4 5 6 7 8 9 10												
Tag No.	A.C. Name & Label	A.C. No.	A.C. Definition										Type ID	Mbr. of K.C. No.
T48	Item Identification (Item ID)	A24	~~~~~										C(10)	K1
T49	Item Description (Item Desc)	A25	~~~~~										C(50)	
T50	Item Name	A26	~~~~~										C(24)	
T51	Item Length	A27	~~~~~										N(8.2)	
T52	Item Quantity (Item Qty)	A28	~~~~~										N(6)	
MODE: E8	TITLE: Owned Attribute Classes											NUMBER: 11		

Figure 5-2. Owned Attribute Classes Form Example

5.1.2.2 Attribute Use Class Table

Source Documents:

1. Owned Attribute Classes forms from the IDEF1 model.
2. Inherited Attribute Classes forms from the IDEF1 model.

Instructions:

Create one table entry for each entry on either type of page.

<u>Table Field</u>	<u>Source Field</u>
Tag No	Tag No. column. Use the number following the "T"; do not include the "T" itself.
Tag Label	A.C. Name & Label column on an Owned Attribute Classes page; Tag & Label column on an Inherited Attribute Classes page. Use the noun phrase that is enclosed in parentheses; do not include the parentheses themselves. If there is no noun phrase in parentheses, use the noun phrase that is not enclosed in parentheses.
EC No	Node (lower left corner). Use the number following the "E"; do not include the "E" itself.
AC No	A.C. No. column. Use the number following the "A"; do not include the "A" itself.

UM 620141001
1 November 1985

Example:

From the example form in Figures 5-2 and 5-3, the resulting Attribute Use Class Table would be as follows:

<u>Tag No</u>	<u>Tag Label</u>	<u>EC No</u>	<u>AC No</u>
48	Item ID	8	24
49	Item Desc	8	25
50	Item Name	8	26
51	Item Length	8	27
52	Item Qty	8	28
105	Op Plan ID	20	68
106	Iss Resource ID	20	76
107	Ben Resource ID	20	76
108	Stock Area ID	20	83
109	Item ID	20	24

UM 620141001
1 November 1985

USED AT:		AUTHOR: PROJECT:		DATE: REV:		WORKING		READER		DATE		CONTEXT:	
		NOTES: 1 2 3 4 5 6 7 8 9 10				DRAFT		RECOMMENDED					
						PUBLICATION							
Tag No.	Tag & Label	A.C. No.	Ind. E.C. No.	Ind. K.C. No.	Ind. Tag No.	Migration Path R.C. Label			Mbr. of K.C. No.				
T105	Operation Plan Identification (OP Plan ID)	A68	E11	K1	T34	Initiates							
T106	Issuing Resource Identification (Iss Resource ID)	A76	E21	K1	T59	Issues							
T107	Benefitting Resource Identification (Ben Resource ID)	A76	E21	K1	T59	Will Benefit From							
T108	Stock Area Identification (Stock Area ID)	A83	E30	K1	T42	Is Depleted By							
T109	Item Identification (Item ID)	A24	E30	K1	T43	Is Depleted By							
MODE E20	TITLE					Inherited Attribute Classes					NUMBER		

Figure 5-3. Inherited Attribute Classes Forms Example

5.1.2.3 Data Type Table

Source Documents:

None

Instructions:

Create one table entry for each data type.

<u>Table Field</u>	<u>Source Field</u>
Type ID	Assign a letter or numeral to identify the data type.
Type Desc	Briefly describe the data type.

Example:

<u>Type ID</u>	<u>Type Desc</u>
C	Character
N	Numeric
D	Date
T	Time

5.1.2.4 Entity Class Table

Source Documents:

Entity Class Glossary forms from the IDEF1 model.

Instructions:

Create one table entry for each glossary form.

<u>Table Field</u>	<u>Source Field</u>
EC No	Node (lower left corner). Use the number following the "E"; do not include the "E" itself.
EC Label	Second line in central area.
EC Name	First line in central area.

UM 620141001
1 November 1985

Example:

From the example form in Figure 5-4, the Entity Class Table would be as follows:

<u>EC No</u>	<u>EC Label</u>	<u>EC Name</u>
35	User Assign	User Assignment

USED AT	AUTHOR PROJECT	A C Nowlin	DATE REV	23 Dec 82	X	WORKING	REVIEW	DATE	CONTEXT
	NOTES 1 2 3 4 5 6 7 8 9 10					DRAFT	RECOMMENDED		
						PURIFICATION			

Entity Class Name	User Assignment
Entity Class Label	User Assign
Entity Class Definition:	Specific individuals are assigned responsibilities for a variety of roles. Such assignment allows decision to be made in a controllable manner. Each assigned individual is uniquely identified by a user identifier

MODE	E35 G1	TITLE	User Assignment	NUMBER	

Figure 5-4. Entity Class Glossary Form Example

5.1.2.5 Inherited Attribute Use Class Table

Source Documents:

Inherited Attribute Classes forms from the IDEF1 model.

Instructions:

Create one table entry for each page entry.

<u>Table Field</u>	<u>Source Field</u>
Tag No	Tag No. column. Use the number following the "T"; do not include the "T" itself.
KC No	Ind. K.C. No. column. Use the number following the "K"; do not include the "K" itself.
Tag No of Key Mem	Ind. Tag No. column. Use the number following the "T"; do not include the "T" itself.
Ind EC No	Ind. E.C. No. column. Use the number following the "E"; do not include the "E" itself.
Dep EC No	Node (lower left corner). Use the number following the "E"; do not include the "E" itself.
RC Label	Migration Path R.C. Label column.

UM 620141001
1 November 1985

Example:

From the example form in Figure 5-5, the resulting table would be as follows:

<u>Tag No</u>	<u>KC No</u>	<u>Tag No of Key Mem</u>	<u>Ind EC No</u>	<u>Dep EC No</u>	<u>RC Label</u>
105	1	34	11	20	Initiates
106	1	59	21	20	Issues
107	1	59	21	20	Will benefit from
108	1	42	30	20	Is depleted by
109	1	43	30	20	Is depleted by

UM 620141001
1 November 1985

USED AT:		AUTHOR PROJECT		DATE REV		WORKING		REVIEW		DATE		CONTEXT	
		NOTES 1 2 3 4 5 6 7 8 9 10				IN WAST		RECOMMENDED					
Tag No.	Tag & Label	A.C. No.	Ind. E.C. No.	Ind. K.C. No.	Ind. Tag No.	Migration Path R.C. Label						Mbr. of K.C. No.	
T105	Operation Plan Identification (OP Plan ID)	A68	E11	K1	T34	Initiates							
T106	Issuing Resource Identification (Iss Resource ID)	A76	E21	K1	T59	Issues							
T107	Benefiting Resource Identification (Ben Resource ID)	A76	E21	K1	T59	Will Benefit From							
T108	Stock Area Identification (Stock Area ID)	A83	E30	K1	T42	Is Depleted By							
T109	Item Identification (Item ID)	A24	E30	K1	T43	Is Depleted By							
MODE E20	Inherited Attribute Classes											NUMBER	

Figure 5-5. Inherited Attribute Classes Form Example

5.1.2.6 Key Class Table

Source Documents:

Key Class Member Table in the CDM (i.e., the table created with the prior set of instructions).

Instructions:

Create one table entry for each entry in the Key Class Member table that has a different entity class or key class number than any prior entry. If two or more entries have the same entity class and key class numbers, create only one entry in this table.

<u>Table Field</u>	<u>Source Field</u>
Key Class EC No	EC No
KC No	KC No

Example:

Sample Key Class Member Table
(from the example in the prior set of instructions):

<u>EC No</u>	<u>KC No</u>	<u>Tag No</u>
8	1	17
8	1	16
14	1	70
14	2	71
14	2	72
18	1	101

Resulting Key Class Table:

<u>Key Class EC No</u>	<u>KC No</u>
8	1
14	1
14	2
18	1

5.1.2.7 Key Class Member Table

Source Documents:

1. Owned Attribute Classes form from the IDEF1 model.
2. Inherited Attribute Classes forms from the IDEF1 model.

Instructions:

Create one table entry for each key class member entry on either type of page. A key class member entry is one that has a number preceded by "K" in the Mbr. of K.C. No. column (right-most column on either type of page).

<u>Table Field</u>	<u>Source Field</u>
EC No	Node (lower left corner). Use the number following the "E"; do not include the "E" itself.
KC No	Mbr. of K.C. No. column. Use the number following the "K"; do not include the "K" itself.
Tag No	Tag No. column. Use the number following the "T"; do not include the "T" itself.

Example:

From the forms shown in Figures 5-6, 5-7, 5-8, and 5-9, the resulting table is as follows:

<u>EC No</u>	<u>KC No</u>	<u>Tag No</u>
8	1	17
8	1	16
14	1	70
14	2	71
14	2	72
18	1	101

UM 620141001
1 November 1985

USED AT		AUTHOR PROJECT		DATE REV.		WORKING		READER		DATE		CONTEXT:	
		NOTES 1 2 3 4 5 6 7 8 9 10				DRAFT							
						RECOMMENDED							
						PUBLICATION							

Tag No.	A.C. Name & Label	A.C. No.	A.C. Definition	Type ID.	Mbr. of K.C. No.
T17	Location Identification (Loc ID)	A08	Unique Identification Assigned To Each Location Where Items Are Stored	C(8)	K01

MODE E8	TITLE	Owned Attribute Classes	NUMBER

Figure 5-6. Owned Attribute Classes Form Example

UM 620141001
1 November 1985

USED AT:		AUTHOR PROJECT:		DATE REV:		WORKING		DATE		CONTEXT:		
		NOTES: 1 2 3 4 5 6 7 8 9 10				DRAFT RECOMMENDED PUBLICATION						
Tag No.	Tag & Label	A.C. No.	Ind. E.C. No.	Ind. K.C. No.	Ind. Tag No.	Migration Path R.C. Label	Mbr. of K.C. No.					
T16	Storage Area Identification (Star Area ID)	A07	E29	K01	T13	Is Composed Of	K01					
T194	Storage Location Status (Loc Status)	A16	E72	K01	T178	Is Assigned To						
MODE E8	TITLE						Inherited Attribute Classes		NUMBER			

Figure 5-7. Inherited Attribute Classes Form Example

UM 620141001
1 November 1985

USED AT		AUTHOR PROJECT		DATE REV.		WORKING DRAFT RECOMMENDED PUBLICATION		DATE		CONTEXT	
		NOTES 1 2 3 4 5 6 7 8 9 10									
Tag No.	A.C. Name & Label	A.C. No.	A.C. Definition					Type I.D.	Mbr. of K.C. No.		
T101	Employee Identification (EmpID)	A16	A unique Identifier assigned to each employee					C (12)	K01		
MODE E18	TITLE		Owned Attribute Classes					NUMBER			

Figure 5-8. Owned Attribute Classes Form Example

UM 620141001
1 November 1985

USED AT		AUTHOR PROJECT										DATE REV		WORKING		REVIEW		DATE		CONTENT	
		NOTES 1 2 3 4 5 6 7 8 9 10												DRAFT							
														RECOMMENDED							
														PURIFICATION							
Tag No.	Tag & Label	A.C. No.	Ind. E.C. No.	Ind. K.C. No.	Ind. Tag No.	Migration Path R.C. Label	Mbr. of K.C. No.														
T70	Operation Execution Plan Component Identification (OEP COMP ID)	A11	E11	K01	T39	Is	K01														
T71	Operation Execution Plan Identification (OEP ID)	A11	E101	K01	T52	Has	K02														
T72	Operation Number (Oper No.)	A14	E10	K01	T53	Has	K02														
MODE	E14	Inherited Attribute Classes																NUMBER			

Figure 5-9. Inherited Attribute Classes Form Example

5.1.2.8 Relation Class Table

Source Documents:

Relation Classes pages from the IDEF1 model.

Instructions:

Create one table entry for each page entry.

<u>Table Field</u>	<u>Source Field</u>
Ind EC No	Node (lower left corner). Use the number following the "E"; do not include the "E" itself.
Dep EC No	Dep. E.C. No. column. Use the number following the "E"; do not include the "E" itself.
RC Label	Relation Class Label column.
Min No Dep Ent	R.C. Card. column. If a filled-in diamond () is shown, enter 1 (one); otherwise, enter 0 (zero).
Max No Dep Ent	R.C. Card. column. If a half diamond () is shown, enter 1 (one); otherwise, leave blank.
KC No	Ind. K.C. No. column. Use the number following the "K"; do not include the "K" itself.

Example:

From the example shown in Figure 5-10, the resulting table is as follows:

Ind EC No	Dep EC No	RC Label	Min No Dep Ent	Max No Dep Ent	KC No
11	13	Is	0	1	1
11	61	Has	0		1
11	15	Is used to manufacture	1		1
11	14	Is	0	1	1
11	10	Has	1		1
11	71	Has	0		1

UM 620141001
1 November 1985

USED AT	AUTHOR PROJECT	DATE REV	WORKING		DATE	CONTEXT
			DEFT	RECOMMENDED PUBLICATION		
NOTES 1 2 3 4 5 6 7 8 9 10						
Independent Entity Class Name	Relation Class Label	R.C. Card	Dependent Entity Class Name	Dep. E.C. No.	Ind. K.C. No.	
Op Exec Plan	Is		OEP Group Member	E13	K1	
Op Exec Plan	Has		OEP Stored Item Req	E61	K1	
Op Exec Plan	Is Used To Manufacture		Op Exec Plan Part	E15	K1	
Op Exec Plan	Is		Op Exec Plan Comp	E14	K1	
Op Exec Plan	Has		Operation	E10	K1	
Op Exec Plan	Has		Op Exec Plan Obsl	E71	K1	

MODE	E11	TITLE	Relation Classes	NUMBER

Figure 5-10. Relation Classes Form Example

5.2 Loading the Initial CS Description

Objective:

- Load the descriptions of the entity, relation, attribute, and key classes contained in the model into the following tables in the CDM database:

Entity Class Table
Relation Class Table
Attribute Class Table
Attribute Use Class Table
Key Class Table
Key Class Member Table
Inherited Attribute Use Class Table

If the CDM tables are to be updated with the NDDL commands, skip to Section 5.2.2.

5.2.1 Direct Loading of the CS CDM Tables

Please refer to Section 5.1.2 for details on how to load these tables.

Tasks:

1. The CDM Administrator loads descriptions from the Entity Class Definition Forms.

Create one entry in the Entity Class Table from each form.

2. The CDM Administrator loads descriptions from the Relation Classes Forms.

Create one entry in the Relation Class Table from each line on each form.

3. The CDM Administrator loads descriptions from the Owned Attribute Classes Forms.

Create one entry in the Attribute Class Table from each line on each form.

4. The CDM Administrator loads descriptions from the Inherited Attribute Class Forms.

Create one entry in the Inherited Attribute Use Class Table from each line on each form.

Create one entry in the Attribute Use Class Table from each line on each form.

Create one entry in the Key Class Table for each key class number in the Mbr. of K.C. No. column, if any.

Create one entry in the Key Class table for each key class number in the Mbr. of K.C. No. column, if any, unless a duplicate entry is already in the table.

5.2.2 Loading the CS with the NDDL

Objective:

- Load the descriptions of the entity, relation, attribute, and key classes contained in the model into the CDM database with NDDL Commands.

Task:

1. The CDM Administrator loads the domains for the attribute classes from the Owned Attribute Classes Forms.

For each attribute, use the NDDL CREATE DOMAIN command to load domain(s) and data types. Data type is indicated by the Type ID column on the form.

2. The CDM Administrator loads the attribute classes from the Owned Attribute Classes Forms.

For each attribute class, use the NDDL CREATE ATTRIBUTE command.

3. The CDM Administrator loads the descriptions for the attribute classes from the Owned Attribute Classes Forms.

For each attribute, use the NDDL DESCRIBE ATTRIBUTE command.

4. The CDM Administrator loads the entity classes from the Entity Definition Forms.

For each entity class, use the NDDL CREATE ENTITY command.

Note:

The attributes for independent entity classes come from the Owned Attribute Form. The attributes for dependent entity classes come from both the Owned Attribute and Inherited Attribute Forms.

5. The CDM Administrator loads the descriptions for the entity classes from the Entity Definition Form.

For each entity class, use the NDDL DESCRIBE ENTITY command.

6. The CDM Administrator loads the relation classes from the Relation Classes Forms.

For each relation class, use the NDDL CREATE RELATION command.

7. The CDM Administrator loads the descriptions for the relation classes.

Use the NDDL DESCRIBE RELATION command.

5.3 Modifying/Deleting CS Elements

Prior to modifying or deleting elements of the CS, the CDM Administrator must assess the impact of the proposed change on the other components of the CDM. The objective of this section is to provide the CDM Administrator with an approach to the analysis of the impact that a change in the CS might have upon the other areas of the CDM or on software modules, such as user APs and generated APs.

The approach that is taken in analyzing the impact that a change to the CS might have to other areas of the CDM, or to a software module, is to list the changes that might be made, and then for each of those changes, to identify the other changes that would have to be made -- either in the CS, or another schema, or in an ES-CS, or an IS-CS mapping, or in a software module. Changes that do not impact any other areas are omitted.

A similar section appears in the discussions that follow on the Internal Schemas and the IS-CS Mappings and on the External Schemas and the ES-CS Mappings, Sections 6 and 7 respectively.

The following assumptions about the nature of the changes to the Conceptual Schema and the sequence in which they are made have been taken in order to perform the analysis:

1. If the updating the CDM tables directly, components of the conceptual schema are added in the following sequence:

- Entity classes
- Attribute classes
 - An owned attribute class for each
 - An attribute use class in the owner entity class for each
- Key classes
- Key class members
- Relation classes
 - An inherited key class for each
 - An inherited attribute class for each member of the key class that migrates to become the inherited key class
 - An attribute use class for each inherited attribute class

2. If using the NDDL, components of the conceptual schema are added in the following sequence:

- Data types
- Attribute classes
- Attribute domains
- Entity classes
- Relation classes

3. All changes in the conceptual schema that are needed to support a change in an external or internal schema are made before the external or internal schema is changed.

4. All changes in the conceptual schema that are needed to support a change in an ES-CS or IS-CS mapping are made before the ES-CS or IS-CS mapping is changed.

5. A change in the name or definition of a component of the conceptual schema is for cosmetic purposes only and does not alter the basic meaning of that component.

Finally, a note of explanation about how the changes and their impacts are organized. Only the direct impacts of a change are listed with it. If one change results in a cascade of other changes, only the first in the cascade is listed with the initial change. Each subsequent change is listed as an impact of the one immediately before it. So to find the total extent of the impact of a change, one must trace from the initial change to each change that it results in and then to each in which that change impacts.

Figure 5-11 shows the relationship between the change and the part of the CDM that may be impacted by the change.

OVERVIEW MATRIX	A change to:							
	Ent.	Att.	Att.	Key	KeyCl	Rel.	Inh.	Inh.
Can impact:	Class	Class	UseCl	Class	Mbr.	Class	KeyCl	AttCl
Att. Class	X							
Att. Use Cl.		X						X
Key Class	X				X	X		
Key Class Mbr			X	X				
Relation Cl.	X							
Inh. Key Cl.				X		X		
Inh. Att. Cl.					X		X	
Own. Att. Cl.	X	X						
User View							X	
Data Item		X	X					
EC-UV Join							X	
AUC-DI Map.			X					
Record Type							X	
Data Field		X	X					
EC-RT Join							X	
EC-RT Map.	X							
AUC-DF Map.			X					
AUC-Set Map.			X					
DF Index			X					
RC-Set Map.							X	
Software Mod.		X						

Figure 5-11. Impact of Conceptual Schema Changes

5.3.1 Entity Class Changes

- Add a new entity class.
No other impact.
- Change an entity class name.
No other impact.
- Change an entity class definition.
No other impact.
- Change an entity class keyword.
No other impact.

- Delete an entity class.

If the entity class has any owned attribute classes, either delete them and the corresponding attribute classes, or change their ownership to another entity class.

Note:

If using NDDL, the DROP ENTITY command will:

- Delete any owned attribute class occurrences.
- Remove attribute use, inherited attribute, key class member, and key classes.
- Delete all relation classes for the entity.
- Delete any keywords associated with the entity.
- Delete all description texts for the entity.

Delete all the key classes for the entity class.

Delete all the relation classes in which the entity class is either independent or dependent.

Delete any EC-RT mappings in which the entity class is involved and any corresponding horizontal partitions and constraint statements and others.

5.3.2 Attribute Class Changes

- Add a new attribute class.

Add an owned attribute class to specify the entity class that owns the attribute class. Add an attribute use class in that entity class to represent the attribute class.

Note:

If using NDDL, the CREATE ATTRIBUTE command will:

- Add a new attribute class
 - Associate a domain with the attribute class.
 - Create keyword references for the attribute.
- The NDDL CREATE ENTITY command is used to add key classes and owned attributes.

Add an attribute class data description for the attribute class.

- Change an attribute class name.

No other impact.

- Change an attribute class definition.

No other impact.

- Change an attribute class keyword.

No other impact.

- Change the owner entity class of an attribute class.

This is the same as deleting the attribute class and then re-adding it with a different owner entity class.

- Change the data description of an attribute class.

This requires at least the recompilation of all

software modules that access the attribute class. It may also involve changing the data descriptions of the data items and data fields that map to the attribute class.

1. Identify all the attribute use classes that correspond to the attribute class.
2. Identify any data field to which those attribute use classes map.
3. If the data descriptions of any of those data fields are incompatible with the new data description of the attribute class, change those data field data descriptions.
4. Identify any data items which map to the attribute use classes from Step 1.
5. If the data descriptions of any of those data items are incompatible with the new data description of the attribute class, change those data item data descriptions.
6. Identify all the software modules that use any of those data items.
7. Recompile all those software modules, even if none of the data item data descriptions or data field data descriptions were changed.

Note:

This section can be ignored when using the NDDL.

- Delete an attribute class.

Delete the owned attribute class for the attribute class.

Delete the attribute use class that represents the attribute class in its owner entity class.

Delete the attribute class data description for the attribute class.

5.3.3 Attribute Use Class Changes

Note:

If using the NDDL, the ALTER ATTRIBUTE command will:

- Change domains
- Add keywords
- Drop keywords

- Add a new attribute use class.

The addition of an attribute use class is never initiated on its own; it is always the result of one of the following conceptual schema changes:

1. The addition of an attribute class
2. The change in ownership of an attribute class
3. The addition of an inherited attribute class

No other impact.

- Change an attribute use class name.

No other impact.

- Delete an attribute use class.

The deletion of an attribute use class is never initiated on its own; it is always the result of one of the following conceptual changes:

1. The deletion of an attribute class
2. The change in ownership of an attribute class
3. The deletion of an inherited attribute class

Delete any key class members that the attribute use class is used as.

If the attribute use class maps to any data items, either delete the AUC-DI mappings and the data items, or change them to map to other attribute classes.

If the attribute use class maps to any data fields, either delete the AUC-DF mappings and the data fields, or delete only the mappings leaving the data fields unmapped, or change them to map to other attribute use classes.

Delete any repeating data field indexes that the attribute use class is used as.

Delete any AUC-Set mappings for the attribute use class.

5.3.4 Key Class Changes

Note:

If using the NDDL, the DROP ATTRIBUTE command will:

- Delete the attribute; if owned, all occurrences of the attribute are removed from owned attribute, attribute use class, key class member, and inherited attribute use class.
- The attribute use class is deleted from the model.
- Those key class occurrences with no remaining key class members are deleted.
- If a key class is deleted, the occurrence of a complete relation is also deleted.
- All keywords associated with the attribute class will be dropped.
- The primary name and all aliases for the attribute class will be deleted.
- All description texts for the attribute class will be deleted.

- **Add a new key class**

Add a key class member for each attribute use class that is part of the key class.

- **Delete a key class.**

Delete all the key class members that belong to the key class.

Delete any inherited key classes that the key class

has migrated to become.

5.3.5 Key Class Member Changes

Note:

If using the NDDL, the CREATE ENTITY or ALTER ENTITY commands are used to add or delete key classes.

- Add a new key class member.

If the new member is being added to a key class that has migrated to become one or more inherited key classes, add a corresponding inherited attribute class to each of those inherited key classes.

- Delete a key class member.

Delete any inherited attribute classes that the key class member has migrated to become

If the key class member is the last or only member of the key class, delete the key class also.

5.3.6 Relation Class Changes

Note:

If using the NDDL, the CREATE ENTITY or ALTER ENTITY commands are used to add or delete key members.

- Add a new relation class.

Add the inherited key class for which the relation class is the migration path.

- Change a relation class name.

No other impact.

- Change a relation class definition.

No other impact.

- Change a relation class keyword.

No other impact.

- Change the cardinality of a relation class.

Only the cardinality for the entity class that is dependent in the relation class can be changed, not that for the independent entity class; i.e., a relation class cannot be changed from specific to nonspecific. When the cardinality changes from 1:0-1 to 1:M, usually one or more members must be added to the key class of the dependent entity class. When the cardinality changes from 1:M to 1:0-1, usually one or more members must be deleted from the key class of the dependent entity class.

- Change which key class migrates through a relation class.

This is the same as deleting the inherited key class for which the relation class is the migration path and then re-adding it for a different migrating key class.

- Change which entity class is independent in a relation class.

This is the same as deleting the relation class and then re-adding it for a different independent entity class.

- Change which entity class is dependent in a relation class.

This is the same as deleting the relation class and then re-adding it for a different dependent entity class.

- Delete a relation class.

Delete the inherited key class for which the relation class is the migration path.

Note:

If using the NDDL, the CREATE RELATION, ALTER RELATION, and DROP RELATION commands are used for relation class changes. The CREATE RELATION command can:

- Add a new relation to the model.
- Add cardinalities for the entities of the relation class.

- Record key class migration.
- Add keyword references for the relation.
- Create an attribute use and an inherited attribute use class for the dependent entity class for each key class member of the independent entity migrated to the dependent entity.

The ALTER RELATION command can:

- Change cardinalities.
- Change key class migration.
- Change keyword references.

The DROP RELATION command will:

- Remove attribute use and inherited attributes from the dependent entity and any other entities to which they have migrated.
- Delete the relation class from the model.
- Drop all keywords associated with the relation
- Delete all description text for the relation.

5.3.7 Inherited Key Class Changes

- Add a new inherited key class.

The addition of an inherited key class is never initiated on its own; it is always the result of one of the following conceptual schema changes:

1. The addition of a new relation class
2. The change of which key class migrates through a relation class

Add an inherited attribute class for each member of the key class to which the inherited key class corresponds.

- Delete an inherited key class.

The deletion of an inherited key class is never initiated on its own; it is always the result of one of the following conceptual schema changes:

1. The deletion of a relation class
2. The change of which key class migrates through a relation class

Delete all the inherited attribute classes that belong to the inherited key class.

Delete any EC-UV joins for which the inherited key class is the basis, and change the corresponding user views as necessary to account for the deleted joins or delete those user views entirely.

Delete any EC-RT joins for which the inherited key class is the basis, and change the corresponding record types as necessary to account for the deleted joins or delete those record types entirely.

Delete any RC-Set mappings for which the inherited key class is the basis.

Note:

If using the NDDL, inherited key classes are added, changed, or deleted with the CREATE RELATION, ALTER RELATION, and DROP RELATION commands.

5.3.8 Inherited Attribute Class Changes

- Add a new inherited attribute class.

The addition of an inherited attribute class is never initiated on its own; it is always the result of one of the following conceptual schema changes:

1. The addition of an inherited key class
2. The addition of a key class member

Delete the attribute use class that represents the inherited attribute class in the dependent entity class.

Note:

If using the NDDL, inherited attribute classes and key class members are added, changed, or deleted with the CREATE RELATION, ALTER RELATION, and DROP RELATION commands.

5.3.9 Summary

The following points are offered in summary

1. A change in the conceptual schema can result in additional changes in that schema, in external and internal schemas, in ES-CS and IS-CS mappings, and in software modules.
2. The information in the CDM database and the CDM1 model is inadequate for identifying the software modules that are impacted by most schema changes. Specifically, the following information needs to be added:
 - The data items that are accessed by a software module that contains user views.
 - The databases, record types, data fields, record sets, record set members, and database areas that are accessed by a software module that accesses databases directly.
 - The record types, data fields, record sets, record set members, and database areas that are accessed by a generated AP.

5.4 Updating the CS Tables in the CDM

Objectives:

- Load the descriptions of the new entity, relation, attribute, and key classes contained in the model into the following tables in the CDM database:
 - Entity Class Table
 - Relation Class Table
 - Attribute Class Table
 - Attribute Use Class Table
 - Key Class Table
 - Key Class Member Table
 - Inherited Attribute Use Class Table
- Change or delete entries in these tables to reflect updates to the descriptions of the entity, relation, attribute, and key classes that are already in the conceptual schema.

If the CDM tables are to be updated with the NDDL commands, skip to Section 5.4.2

5.4.1 Direct Updating of the CS CDM Tables

Please refer to Section 5.1.2 for details on how to load and update these tables.

Tasks:

1. The CDM Administrator rennumbers the new model elements.

Usually, some of the new entity class, key class, attribute class, and tag numbers that were assigned in Phases 1-4, i.e., those prefixed with "N," are the same as some that are already in the conceptual schema. No attempt is made to prevent this, because to do so would impede the development of the model. Consequently, all of the new numbers must now be examined, and any duplicates must be replaced.

List all of the new entity class numbers from the Entity Class Definition Forms in numeric sequence. Compare each to those in the Entity Class Table. If a match is found, select another number that is not in the list or the table, and write it on the list next to the one to be replaced.

List all of the new attribute class numbers from the Owned Attribute Classes Forms in numeric sequence. Compare each to those in the Attribute Class Table. If a match is found, select another number that is not in the list or the table, and write it on the list next to the one to be replaced.

List all of the new tag numbers from the Tag No. column of the Owned Attribute Classes Forms and the Inherited Attribute Classes Forms in numeric sequence. Compare each to those in the Attribute Use Class Table. If a match is found, select another number that is not in the list or the table, and write it on the list next to the one to be replaced.

List only the new key class numbers for conceptual schema entity classes, i.e., those in the Mbr. of K.C. No. column of the Owned Attribute Classes Forms

and Inherited Attribute Classes Forms that have an entity class number prefixed with "C" in the lower left corner. List both the entity class number and the new key class number; do not list the same pair more than once. Compare each pair to those in the Key Class Table. If a match is found, select another key class number that is not paired with that entity class number in the list or the table, and write it on the list next to the one to be replaced. Do not assign another entity class number.

These replacement numbers will be used to update the conceptual schema tables rather than the duplicated numbers on the forms.

2. The CDM Administrator updates descriptions from the Entity Class Definition Forms for conceptual schema entity classes.

Each of these forms has an entity class number prefixed with "C" in the lower left corner.

If "UPDATED" is written in the lower left corner, change the corresponding entry in the Entity Class Table.

3. The CDM Administrator loads or updates descriptions from the Relation Classes Forms for conceptual schema entity classes.

Each of these forms has an entity class number prefixed with "C" in the lower left corner.

When creating or changing table entries in this task, use the replacement numbers from Task 1 rather than the duplicated ones prefixed with "N" on the forms.

If a line has an entity class number prefixed with "N" in the Dep. E.C. No. column, create one entry in the Relation Class Table.

If "UPDATED" is written in the lower left corner and:

"CHANGE" is written next to a line, change the corresponding entry in the Relation Class Table.

"DELETE" is written next to a line:

- Remove the corresponding entry from the Relation Class Table.
- Modify any CS-IS mappings and any CS-ES mappings that are affected; see Sections 6 and 7 for details.

"OMIT" is written next to a line, a key class did not migrate through that relation class. That must be done before the conceptual schema tables are updated.

4. The CDM Administrator loads or updates descriptions from the Owned Attribute Classes Forms for conceptual schema entity classes.

Each of these forms has an entity class number prefixed with "C" in the lower left corner.

When creating or changing table entries in this task, use the replacement numbers from Task 1 rather than the duplicated ones prefixed with "N" on the forms. If a line has an attribute class number prefixed with "N" in the A.C. No. column:

- Create one entry in the Attribute Class Table.
- Create one entry in the Attribute Use Class Table.
- Create one entry in the Key Class Member Table for each key class number in the Mbr. of K.C. No. column, if any.
- Create one entry in the Key Class Table for each key class number in the Mbr. of K.C. No. column, if any, unless a duplicate entry is already in the table.

If a line has an attribute class number prefixed with "C" in the A.C. No. column:

- Create one entry in the Key Class Member Table for each key class number prefixed with "N" in the Mbr. of K.C. No. column, if any.
- Create one entry in the Key Class Table for each key class number prefixed with "N" in the Mbr.

of K.C. No. column, if any, unless a duplicate entry is already in the table.

If "UPDATED" is written in the lower left corner and "CHANGE" is written next to a line, change the corresponding entry in the Attribute Class Table.

If "UPDATED" is written in the lower left corner and "DELETE" is written next to a line:

- Remove the corresponding entry from the Attribute Use Class Table.
- Remove the entry from the Key Class Member Table that corresponds to each key class number in the Mbr. of K.C. No. column, if any.
- Remove the entry from the Key Class Table that corresponds to each key class number in the Mbr. of K.C. No. column, if any, unless there is still an entry with the same entity class number and key class number in the Key Class Member Table.
- Modify any CS-IS mappings and any CS-ES mappings that are affected; see Sections 6 and 7 for details.

If "UPDATED" is written in the lower left corner and "OMIT" is written next to a line:

- Remove the entry from the Key Class Member Table that corresponds to each circled key class number in the Mbr. of K.C. No. column.
- Remove the entry from the Key Class Table that corresponds to each circled key class number in the Mbr. of K.C. No. column unless there is still an entry with the same entity class number and key class number in the Key Class Member Table.

If "UPDATED" is written in the lower left corner and "NEW OWNER" is written next to a line; make no table updates from the line. The necessary updates will be made from lines on other forms.

5. The CDM Administrator loads or updates descriptions from the Inherited Attribute Classes Forms for conceptual schema entity classes.

Each of these forms has an entity class number prefixed with "C" in the lower left corner.

When creating or changing table entries in this task, use the replacement numbers from Task 1 rather than the duplicated ones prefixed with "N" on the forms.

If a line has a tag number prefixed with "N" in the Tag No. column:

- Create one entry in the Attribute Use Class Table.
- Create one entry in the Inherited Attribute Use Class Table.
- Create one entry in the Key Class Member Table for each key class number in the Mbr. of K.C. No. column, if any.
- Create one entry in the Key Class Table for each key class number prefixed with "N" in the Mbr. of K.C. No. column, if any, unless a duplicate entry is already in the table.

If a line has a tag number prefixed with "C" in the Tag No. column:

- And a tag number prefixed with "N" in the Ind. Tag No. column, create one entry in the Inherited Attribute Use Class Table.
- Create one entry in the Key Class Member Table for each key class number prefixed with "N" in the Mbr. of K.C. No. column, if any.
- Create one entry in the Key Class Table for each key class number prefixed with "N" in the Mbr. of K.C. No. column, if any, unless a duplicate entry is already in the table.

If "UPDATED" is written in the lower left corner and "CHANGE" is written next to a line, change the

corresponding entry in the Inherited Attribute Use Class Table.

If "UPDATED is written in the lower left corner and DELETE" is written next to a line:

- Remove the corresponding entry from the Attribute Use Class Table.
- Remove the corresponding entry from the Inherited Attribute Use Class Table.
- Remove the entry from the Key Class Member Table that corresponds to each key class number in the Mbr. of K.C. No. column, if any.
- Remove the entry from the Key Class Table that corresponds to each key class number in the Mbr. of K.C. No. column, if any, unless there is still an entry with the same entity class number and key class number in the Key Class Member Table.
- Modify any CS-IS mappings and any CS-ES mappings that are affected; see Sections 6 and 7 for details.

If "UPDATED is written in the lower left corner and "OMIT" is written next to a line:

- Remove the entry from the Key Class Member Table that corresponds to each circled key class number in the Mbr. of K.C. No. column.
- Remove the entry from the Key Class Table that corresponds to each circled key class number in the Mbr. of K.C. No. column unless there is still an entry with the same entity class number and key class number in the Key Class Member Table.

6. The CDM Administrator loads descriptions from the Entity Class Definition Forms for new entity classes.

Each of these forms has an entity class number prefixed with "N" in the lower left corner.

When creating table entries in this task, use the replacement numbers from Task 1 rather than the duplicated ones prefixed with "N" on the forms.

Create one entry in the Entity Class Table from each form.

7. The CDM Administrator loads descriptions from the Relation Classes Forms for new entity classes.

Each of these forms has an entity class number prefixed with "N" in the lower left corner.

When creating table entries in this task, use the replacement numbers from Task 1 rather than the duplicated ones prefixed with "N" on the forms. Create one entry in the Relation Class Table from each line on each form.

8. The CDM Administrator loads or updates descriptions from the Owned Attribute Classes Forms for new entity classes.

Each of these forms has an entity class number prefixed with "N" in the lower left corner.

When creating or changing table entries in this task, use the replacement numbers from Task 1 rather than the duplicated ones prefixed with "N" on the forms.

Create one entry in the Attribute Use Class Table from each line on each form.

Create one entry in the Key Class Member Table for each key class number in the Mbr. of K.C. No. column, if any.

Create one entry in the Key Class Table for each key class number in the Mbr. of K.C. No. column, if any, unless a duplicate entry is already in the table.

If a line has an attribute class number prefixed with "N" in the A.C. No. column, create one entry in the Attribute Class Table.

If a line has an attribute class number prefixed with "C" in the A.C. No. column, change the entity class number and anything else that is different in the

corresponding entry in the Attribute Class Table.

9. The CDM Administrator loads descriptions from the Inherited Attribute Classes Forms for new entity classes.

Each of these forms has an entity class number prefixed with "N" in the lower left corner.

When creating table entries in this task, use the replacement numbers from Task 1 rather than the duplicated ones prefixed with "N" on the forms.

Create one entry in the Attribute Use Class Table from each line on each form.

Create one entry in the Inherited Attribute Use Class Table from each line on each form.

5.4.2 Updating the CS CDM Tables with the NDDL

Objectives:

- Update the descriptions of the new entity, relation, attribute, and key classes continued in the model into the CDM database with the NDDL.

Tasks:

1. The CDM administrator loads or updates CS attribute classes.

For each new attribute class, use the NDDL CREATE ATTRIBUTE, and DESCRIBE ATTRIBUTE commands.

For each modified attribute, use the NDDL ALTER ATTRIBUTE command. The DESCRIBE ATTRIBUTE command should also be considered.

For each deleted attribute, use the NDDL DROP ATTRIBUTE command.

Note:

All occurrences of the attribute are removed from the owned attribute, attribute use class, key class member, and inherited attribute use class.

2. The CDM administrator loads or updates conceptual schema entity classes.

For each new entity class, use the NDDL CREATE ENTITY and DESCRIBE ENTITY commands.

For each modified entity class, use the NDDL ALTER ENTITY command. The DESCRIBE ENTITY command should also be considered.

For each deleted entity class, use the NDDL DROP ENTITY command.

Note:

Any owned attribute class occurrences are deleted. Also removed are attribute use, inherited attribute, key class member, and key classes. All relation classes involving the entity are deleted, as are any keywords associated with the entity class. The primary name and all aliases for the entity class are deleted. All description texts for the entity class are deleted.

3. The CDM Administrator updates conceptual schema relation classes.

For each new relation class, use the NDDL CREATE RELATION and DESCRIBE RELATION commands.

For each modified relation class, use the NDDL ALTER RELATION command. The DESCRIBE RELATION command should also be considered.

For each deleted relation class, use the NDDL DROP RELATION command.

Note:

If a key class has been migrated, the attribute use and inherited attributes are removed from the dependent entity and from any other entities to which they have migrated. The relation class and complete relation are deleted from the model. The keywords associated with the relation are dropped. All description texts for the relation are deleted.

SECTION 6

MAINTAINING INTERNAL SCHEMAS AND MAPPINGS

6.1 Methodology Overview

This section and its subsections (6.1.1 - 6.1.4) introduce the methodology for building and updating internal schemas (IS) and for mapping them to the conceptual schema (CS). The portion of the CDM database that contains internal schemas and CS-IS mappings is described, and the basic approach to developing both is presented. Detailed instructions for filling out the modeling forms and loading the pertinent CDM database tables are included.

6.1.1 IS and CS-IS Mapping Structure

There are various generic database models (CODASYL, relational, hierarchic, etc.), and most database management systems (DBMSs) are based on one or another of them. In addition, a particular model may be modified or extended for a particular DBMS. Each of these models generates its own style of internal schema. While many internal schema components are common to all styles, some are peculiar to only one or a few. The CDM does not contain a separate structure for each style of internal schema. Instead, a single, composite structure that can support any style is provided. Each internal schema component is represented by one entity class, regardless of how many styles that component is in. The relevant entity classes for each style are listed in the appropriate "Specific Considerations" section 6.3.1, 6.4.1, etc.). In general, they cover:

- DBMSs
- Databases
- Record Types
- Data Fields
- Record Keys
- Record Relationships

The mapping between the conceptual schema and an internal schema has three levels:

- Entity class to record type
- Relation class to record relationship
- Attribute use class to data field

6.1.2 Basic Approach

This methodology addresses the following subjects:

- Describing existing physical database designs in the internal schema portion of the CDM.
- Determining the mappings between internal schemas and the conceptual schema and describing them in the CS-IS mapping portion of the CDM.
- Updating these descriptions to reflect changes in either the physical database designs or the conceptual schema.

This methodology does not address the creation of physical database designs. DBMS vendors, books, classes, etc., offer much more guidance in this area than can be provided here.

A CS-IS mapping is intended to show which components of an internal schema correspond to those of the conceptual schema. A record type maps to an entity class if they both represent the same kind of "real-world" things. For example, in Figure 6-1, the EMP-MAST record type maps to the Employee entity class because both represent employees. There is a one-for-one correspondence between the record type and the entity class; each employee is represented by one instance of the record type and by one instance of the entity class. Notice that even though the record type contains data fields (DIV-NO, DEPT-NAME, SPOUSE-NAME) that correspond to attribute use classes in other entity classes, the record type does not map to those other entity classes. It represents a different kind of "real-world" things than any of those entity classes and is not in a one-for-one correspondence with any of them. For example, one instance of the Department entity class exists for each department while several instances of the EMP-MAST record type exist, one for each employee in a department, or if a department has no employees, no record type instances exists for that department. As another example, the Married Employee entity class has an instance for each employee who is married, while the EMP-MAST record type has an instance for every employee, married or not.

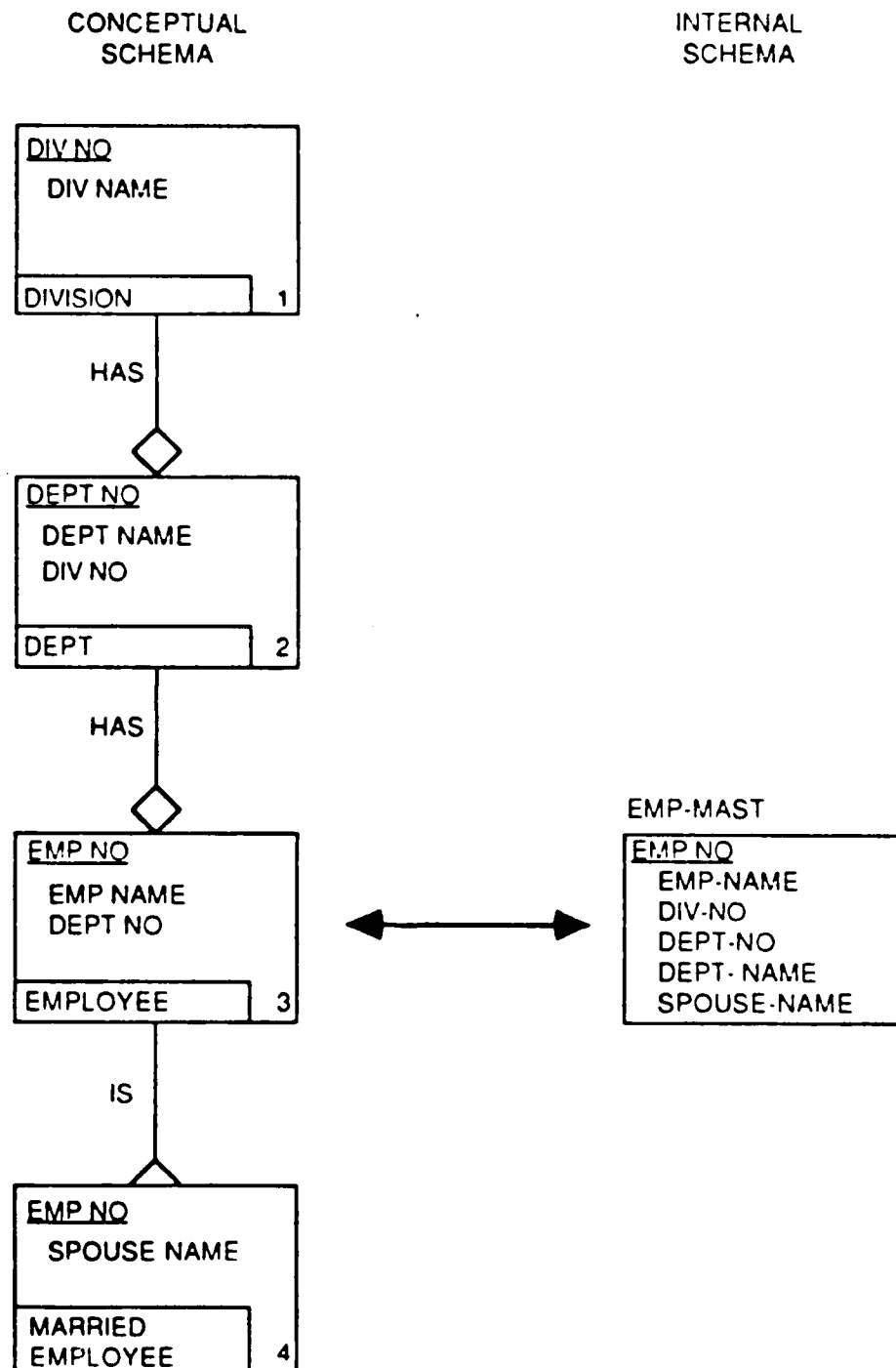


Figure 6-1. Entity Class/Record Type Mapping

In a similar manner, a data field maps to an attribute use class if they both represent the same kind of data about "real-world" things. Using the example in Figure 6-1 again, the EMP-NO, EMP-NAME, and DEPT-NO data fields in the EMP-MAST record type map to attribute use classes in the Employee entity class; DIV-NO and DEPT-NAME, to those in the Dept entity class; and SPOUSE-NAME, to one in the Married Employee entity class. Notice that some data fields could map to more than one attribute use class. For example, EMP-NO and DEPT-NO could have mapped to attribute use classes in the Married Employee and Department entity classes, respectively, instead of those in the Employee entity class. They map to those in the Employee entity class is because the record type maps to that entity class. DIV-NO is another example; it could have mapped to an attribute use class in the Division entity class rather than to one in the Department entity class. The reason it maps to the one in the latter is that the Department entity class is more closely related to the Employee entity class than the Division entity class is. Notice also that all these examples involve attribute use classes that belong to key classes. This is because only they can migrate to other entity classes; an owned, nonkey attribute use class appears only in its owner entity class. These situations are summarized in the following mapping rules:

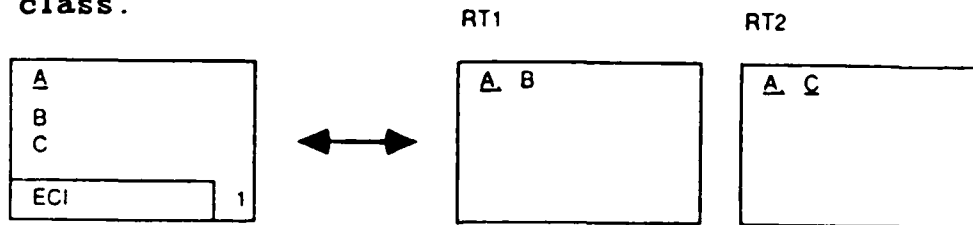
- If a data field could map to either an attribute use class in the entity class to which the record type maps or to one in another entity class, it always maps to the former (e.g., EMP-NO and DEPT-NO).
- If a data field could map to more than one attribute use class, none of which are in the entity class to which the record type maps, it always maps to the one in the entity class that is most closely related to the entity class to which the record type maps (e.g., DIV-NO).

Finally, a record set maps to a relation class if they both represent the same kind of association between "real-world" things. This implies that the two record types in the record set, the owner and the member, map to the two entity classes in the relation class, the independent and the dependent, respectively.

The following subsections (6.1.2.1 - 6.1.2.6) present various subjects to consider when dealing with CS-IS mappings. None of them are mutually exclusive; each can be combined with one or more of the others.

6.1.2.1 Vertical Partitions

An entity class is vertically partitioned when some of its attribute use classes map to data fields in one record type and others map to those in another. An entity class can have several vertical partitions. Each record type maps to the entity class.

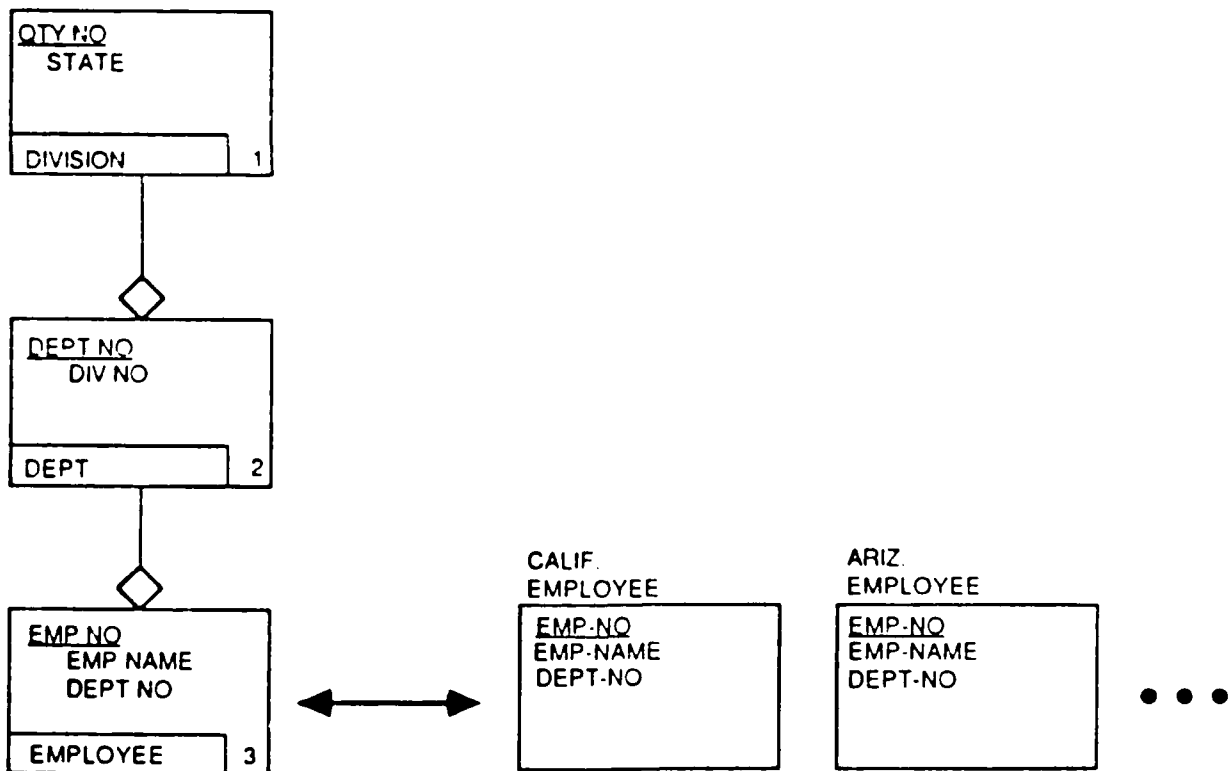
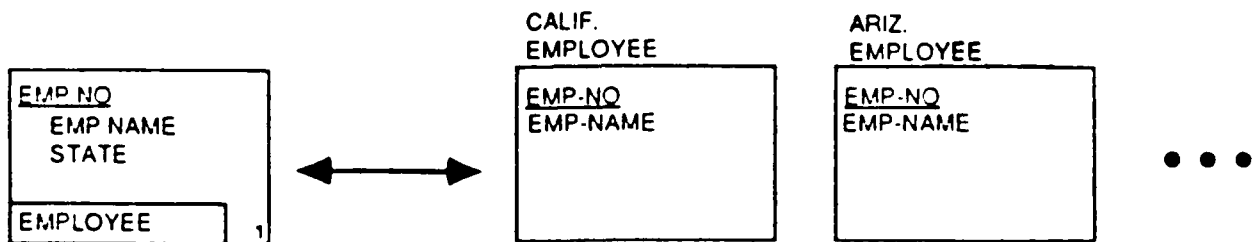


6.1.2.2 Horizontal Partition

An entity class is horizontally partitioned when some of its entity instances map to instances of one record type and others map to instances of another record type. Usually, the horizontal partitioning of an entity class is governed by the values in a particular attribute use class.

The attribute use class may be in the entity class being partitioned (as in the example below), or it may be in one that that entity class is dependent on, either directly or indirectly.

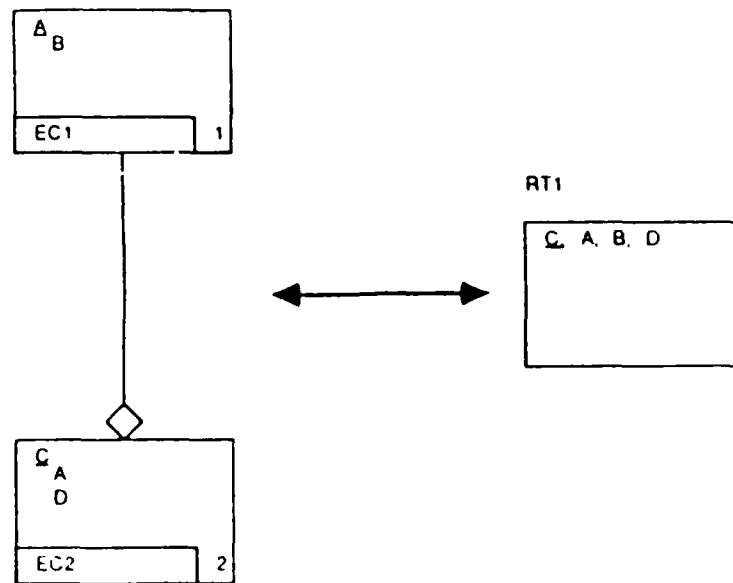
UM 620141001
1 November 1985



An entity class can have several horizontal partitions. Each record type maps to the entity class.

6.1.2.3 Joins

If some data fields in a record type map to attribute use classes in one entity class and others map to those in another, the two entity classes must be combined to form that record type. This is done with a relational "join" operation, which concatenates the entity instances of one entity class with those of the other. The two entity classes must be directly related by a relation class so that their entity instances can be matched using the key class of the independent and the corresponding inherited attribute use class(es) of the dependent.



If the relation class cardinality is one-to-many, each independent entity instance is concatenated with each entity instance that is dependent on it. In the first example in Figure 6-2, each PO-HEADER instance is formed by concatenating a Vendor instance with a PO instance based on identical values in Vendor No. If a Vendor instance has no dependent PO instances, it is not represented by a PO-HEADER instance. This produces one record instance for each instance in the dependent entity class, so the mapping must be to that entity class. If the mapping was to the independent entity class, i.e., if there was one record instance for each Vendor instance, P. O. NO. and any

other attribute use classes from the P. O. entity class could occur multiple times in each record instance. Since a relational join cannot form record instances with repeating data fields, this situation is prohibited.

If the relation class cardinality is one-to-zero-or-one, the mapping can be to either the independent or the dependent entity class because neither can cause a repeating data field. The second and third examples in Figure 6-2 show these two situations. In the second, there is one BUYER record instance for an employee who is not a buyer. In the third example, there is one EMP-MAST instance for each Employee instance. If an employee is not married, the SPOUSE-NAME data field in the record instance for that employee is null.

If a record type has data fields that map to attribute use classes in several entity classes, they must all be combined to form the record type. This is done with a series of the join operations described above, each of which combines two of the entity classes. All of the entity classes must be inter-related such that they form one of the following (See Figure 6-2):

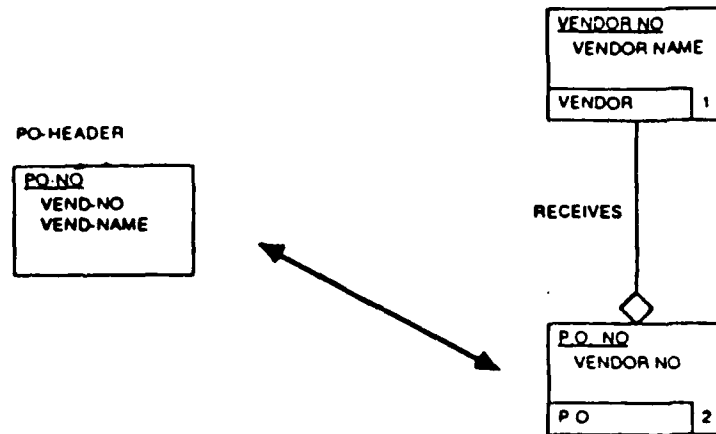
1. A regular hierarchy, i.e., a structure in which:
 - One entity class, called the apex, is not dependent on any of the others (e.g., EC1)
 - Every other entity class is dependent on exactly one entity class (not necessarily the same one for all)
 - Every relation class cardinality is one-to-zero-or-one
2. A confluent hierarchy (an upside-down hierarchy), i.e., a structure in which:
 - One entity class, called the apex, has none of the others dependent on it (e.g., EC14)
 - Every other entity class has exactly one entity class dependent on it (not necessarily the same one for all)
 - Any specific relation class cardinality is permitted

3. A combination of:

- One confluent hierarchy and
- One or more regular hierarchies, each of whose apex entity classes are also in the confluent hierarchy (e.g., EC15, EC20, and EC25).

Each hierarchy is called a join structure. As shown in the examples in Figure 6-3, the record type must map to the apex entity class of the regular or confluent hierarchy. If a combination of hierarchies exists, the mapping must be to the apex of the confluent hierarchy.

ONE-TO-MANY RELATION CLASS



ONE-TO-ZERO-OR-ONE RELATION CLASS

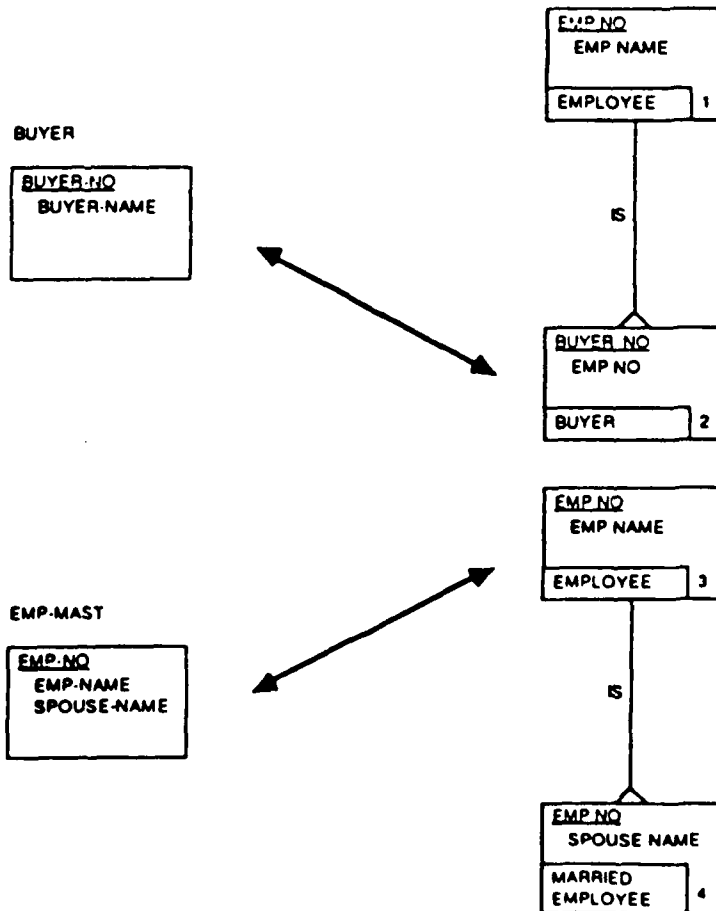
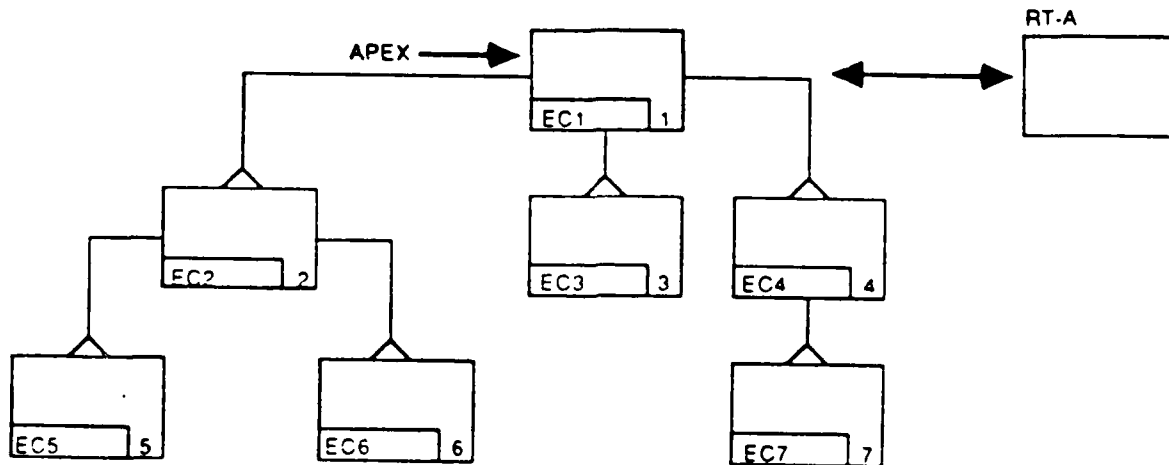


Figure 6-2. Join Examples

REGULAR HIERARCHY:



CONFLUENT HIERARCHY:

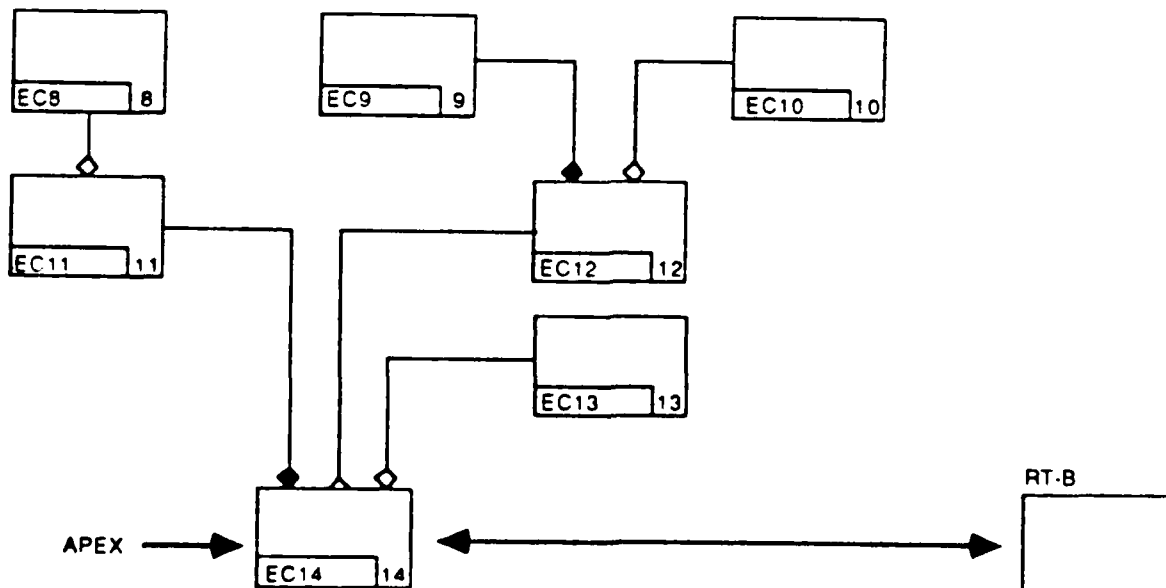


Figure 6-3. Join Structures

COMBINATION.

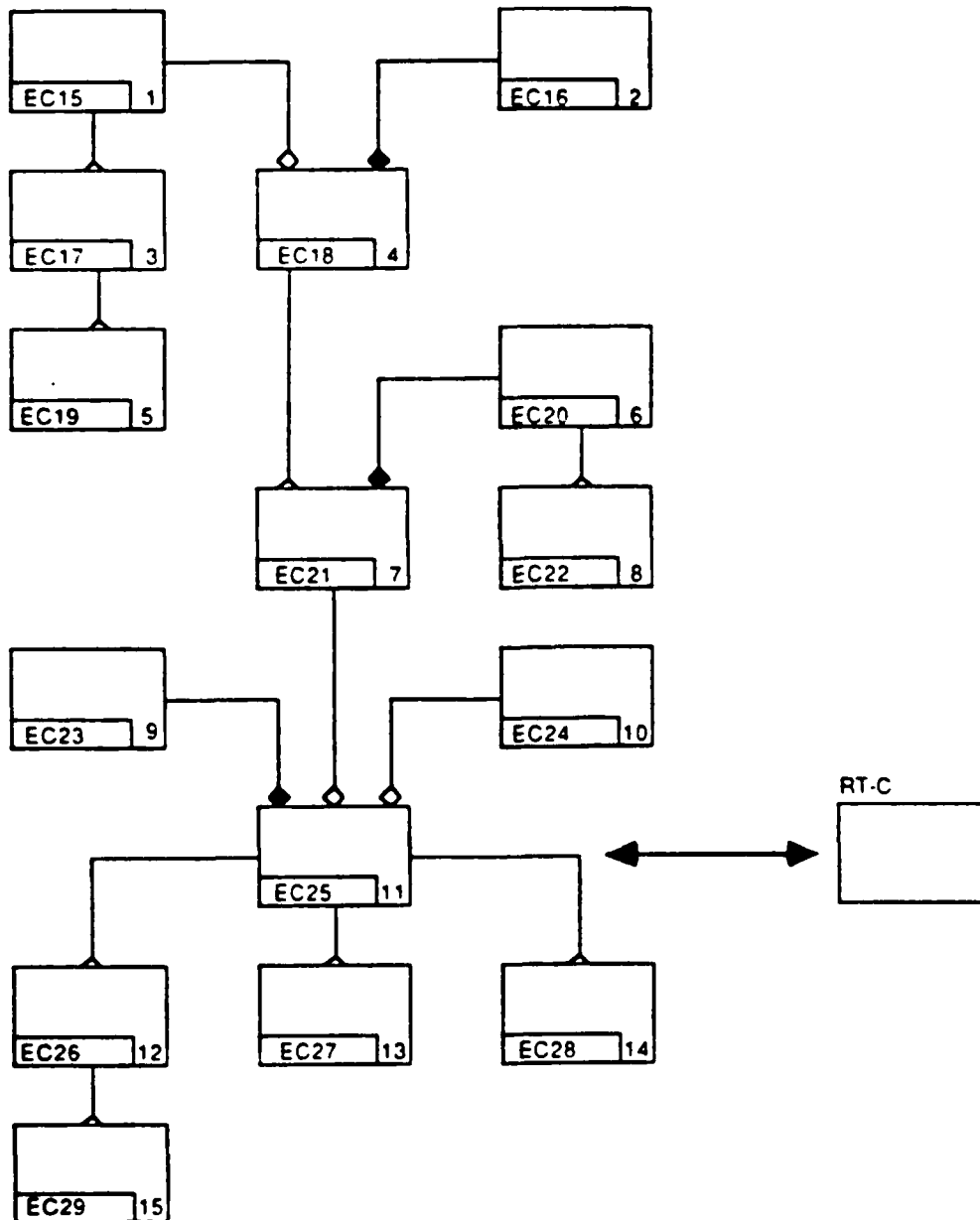
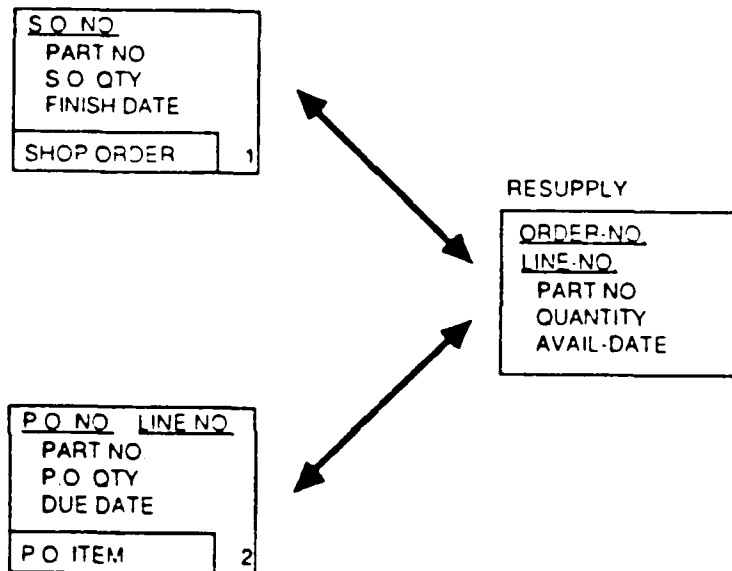


Figure 6-3. Join Structures (Continued)

6.1.2.4 Unions

A record type can map to two or more entity classes. This is the case when there is one record instance for each entity instance in one entity class and one for each in another. In the example below, each RESUPPLY record instance corresponds to either one Shop Order instance or one P. O. Item instance.



The creation of this sort of record type involves the use of the relational "union" operation. This allows the instances from both entity classes to be treated as if they were all the same kind of entity instances. Each data field can map to an attribute use class in each entity class, but that is not required. A data field can map to an attribute use class in one entity class without mapping to one in another. In the example above, these mappings are:

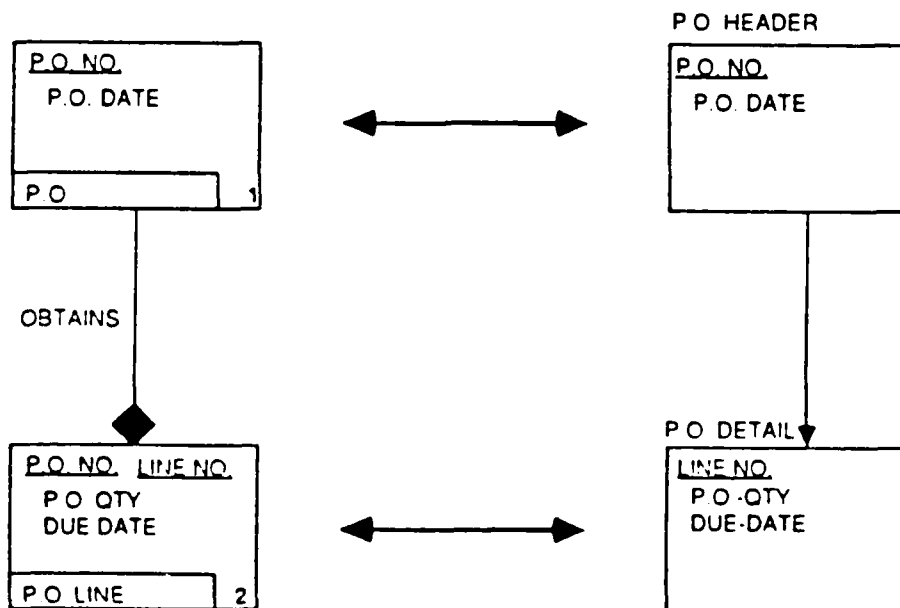
RESUPPLY		SHOP ORDER		P.O. LINE
ORDER-NO	maps to	S.O. No.	and	P.O. No.
LINE-NO	maps to			Line No.
PART-NO	maps to	Part No.	and	Part No.
QUANTITY	maps to	S.O. Qty.	and	P.O. Qty.
AVAIL-DATE	maps to	Finish Date	and	Due Date

LINE-NO does not map to an attribute use class in the Shop Order entity class. Consequently, each record instance that corresponds to an instance of that entity class is null in that data field.

Usually, the entity classes involved in a union are not directly related, although this is not a requirement.

6.1.2.5 Phantoms

Some DBMSs discourage the creation of data fields that would map to inherited attribute use classes. In the example below, P.O. No. in the P.O. Line entity class has no corresponding data field in the PO-DETAIL record type. Instead, when the purchase order number for an instance of that record type is needed, the one in the related PO-HEADER record instance is used.



In this example, the P.O. No. in the P.O. Line entity class is called a "phantom" attribute use class because values for it can be retrieved from the database even though it does not map to any data field.

6.1.2.6 Duplications (Replications vs Redundancy)

Data duplication exists when the value in an attribute use class for a particular entity instance is stored in two or more data fields. In general, this is when an attribute use class

maps to more than one data field. However, there are exceptions. When an entity class is horizontally partitioned, some or all of its attribute use classes map to more than one data field. If the partitions do not overlap though, i.e., if each entity instance corresponds to only one record instance, then each value is stored only once. Then, there is no data duplication. To summarize, data duplication exists when an attribute use class maps to two or more data fields unless all of those mappings result from a non-overlapping horizontal partition.

There are two types of data duplication:

Data Redundancy: The values in one of the data fields to which an attribute use class maps are not kept synchronized with those in another to which it maps.

Data Replication: The values in one of the data fields to which an attribute use class maps are updated and controlled to be kept synchronized with those in another to which it maps. Data replication may be used for performance purposes or for purposes of joining across physical record (but not for joining entity classes).

As indicated by these definitions, data replication can be useful, but data redundancy is always undesirable. With data replication, updates to those multiple copies are controlled from a single source. The multiple copies are kept synchronized such that they reflect the same history of updates. By contrast, with data redundancy, updates to the multiple copies can be controlled by multiple sources, e.g., by different applications. The result is that the copies may reflect different histories of updates and carry different values. When a user accesses redundant data, there can be no guarantee of the integrity or quality of that data. Depending on which copy is accessed, the user may receive very different results.

Whenever an attribute use class maps to more than one data field, the CDM Administrator must specify which is the "preferred copy." This is the data field that the CDM Processor

will use for retrievals and qualifications in all NDML requests, regardless of application. The others will be used for joining across physical records when necessary. If an entity class has been horizontally partitioned, there should be a preferred copy designated in each partition. If a particular application wishes to use other than the preferred copy, it must bypass the CDM Processor and access that data field directly. It cannot use NDML for the request.

The CDM Processor treats all duplication as data replication; it must consider the values in all data fields to be synchronized. If, in fact, some duplication is really data redundancy, improper physical record joining may be performed, resulting in spurious responses.

6.1.3 IS Modeling Forms

Most DBMSs provide a language for defining databases, a Data Definition Language (DDL). DDL Statements for each database are used to directly load the internal schema tables of the CDM database or with the appropriate NDDL commands. Consequently, no internal schema modeling forms are needed.

The following forms are used to model the mappings between internal schemas and the conceptual schema:

- Record Type/Entity Class Mapping Form
- Record Type Join Structure Diagram
- Data Field/Attribute Use Class Mapping Form
- Set Type/Relation Class Mapping Form

The rest of this section contains a detailed description and two samples (one blank, one filled in) of each of these forms.

Note: When using NDDL, any references on the above forms to ID's or NO's should be replaced with names.

Record Type/Entity Class Mapping Form

Purpose: To provide a single source of information about the mappings between record types and entity classes.

Instructions:

Fill in one or more pages for each database. List each entity class to which record type maps.

<u>Form Area</u>	<u>Explanation</u>
1. Database ID	Unique identification code assigned to the database by the CDMA.
2. Record Type ID	Name or code that the DBMS uses to identify the record type.
3. Entity Class No.	Number of an entity class to which the record type maps.
4. Entity Class Name	Name of the entity class whose number is in the prior column. It is included only to make the entry readable; it is not used in loading the mapping tables.

Record Type Join Structure Diagram

Purpose: To provide a single source of information about the join structures for a record type.

Instructions:

Fill in one page for each record type that involves joining two or more entity classes.

<u>Form Area</u>	<u>Explanation</u>
1. Database ID	Unique identification code assigned to the database by the CDMA.
2. Record Type ID	Name or code that the DBMS uses to identify the record type.
3. (Diagram Area)	Depiction of the entity classes and relation classes that make up the join structure.

UM 620141001
1 November 1985

USEDAT	AUTHOR PROJECT	DATE REV	WORKING DRAFT	REVIEW DATE	CONTEXT
	NOTES: 1 2 3 4 5 6 7 8 9 10		RECOMMENDED PUBLICATION		

Database ID	Record Type ID	Entity Class No.	Entity Class Name
①	②	③	④

MODE: ⑤	TITLE:	Record Type Entity Class Mapping	NUMBER

Figure 6-4. Record Type/Entity Class Mapping Form

UM 620141001
1 November 1985

USED AT:	AUTHOR PROJECT 6201M MCM	DATE Aug 1983	X	WORKING []	DATE	CONTEXT
	NOTES 1 2 3 4 5 6 7 8 9 10			RECOMMENDED		
				PUBLICATION		

Database ID	Record Type ID	Entity Class No	Entity Class Name
2	OP_PLAN_GROUP	E12	OP EXEC PLAN GROUP
2	OP_PLAN_OP_PLAN	E14	OP EXEC PLAN COMPONENT
2	OP_PLAN_PART	E15	OP EXEC PLAN PART
2	PERSON	E18	EMPLOYEE

MODE	TITLE	Record Type/Entity Class Mapping	NUMBER	2

Figure 6-5. Record Type/Entity Class Mapping Form Example

UM 620141001
1 November 1985

USED AT	AUTHOR PROJECT	DATE REV	WORKING DRAFT RECOMMENDED PUBLICATION	REVIEW	DATE	CONTEXT
	NOTES: 1 2 3 4 5 6 7 8 9 10					

Database ID: ①	Record Type ID: ②
③	

NODE	TITLE	Record Type Join Structure Diagram	NUMBER

Figure 6-6. Record Type Join Structure Diagram

USED AT	AUTHOR PROJECT	DACOM (CEM, DRR) 6201M MCMM	DATE REV	Aug 1983	X	WORKING PRIVATE	READY TO RECOMMEND PUBLICATION	DATE	CONTEXT
	NOTES 1 2 3 4 5 6 7 8 9 10								

Database ID.: 2	Record Type ID.: Material Req


```

graph TD
    ITEM[ITEM 7] --- IS1[IS] --- PRODUCT_ITEM[PRODUCT ITEM 41]
    PRODUCT_ITEM --- IS2[IS] --- MATERIAL[MATERIAL 40]
    PRODUCT_ITEM --- IS_REF[IS REFERENCED BY] --- PRODUCT_ITEM_ROMT[PRODUCT ITEM ROMT 9]
  
```


NOTE	TITLE	Record Type Join Structure Diagram	NUMBER	34

Figure 6-7. Record Type Join Structure Diagram Example

Data Field/Attribute Use Class Mapping Form

Purpose: To provide a single source of information about the mappings between data fields and attribute use classes.

Instructions:

Fill in one or more pages for each record type in a database. List each attribute use class to which each data field maps.

<u>Form Area</u>	<u>Explanation</u>
1. Database ID	Unique identification code assigned to the database by the CDMA.
2. Record Type ID	Name or code that the DBMS uses to identify the record type.
3. Data Field ID	Name or code that the DBMS uses to identify the data field.
4. Entity Class No.	Number of the entity class that contains the attribute use class whose number and tag are in the next two columns.
5. Att. Use Cl. Tag No.	Tag number of an attribute use class to which data field maps.
6. Attribute Use Class	Name of the attribute use class Tag whose tag number is in the prior column. It is included only to make the entry readable; it is not used in loading the mapping tables.

Relation Class/Set Type Mapping Form

Purpose: To provide a single source of information about the mappings between CODASYL set types or IMS paths and relation classes.

USED AT	AUTHOR PROJECT	DATE REV	WORKING STATUS	IN APT DATE	CONTEXT
	NOTES: 1 2 3 4 5 6 7 8 9 10		RECOMMENDATION		

Database ID: ①	Record Type ID: ②				
Data Field ID	Entity Class No.	Alt. Use Cl Tag No.	Attribute Use Class Tag		
③	④	⑤	⑥		

MODE	TITLE	Data Field/Attribute Use Class Mapping	NUMBER

Figure 6-8. Data Field/Attribute Use Class Mapping

UM 620141001
1 November 1985

USED AT	AUTHOR PROJECT	DACOM (CEM, DRR)	DATE REV	Aug 1983	TEACHER	DATE	CONTEXT
	NOTES 1 2 3 4 5 6 7 8 9 10				WORKING		
					INPAET		
					INCOMMEMED		
					INFORMATION		

Database ID:		Record Type ID : Resource	
Data Field ID:	Entity Class No.	Alt. Use Cl. Tag No	Attribute Use Class Tag
RES_ID	E21	T33	MFG AREA ID
RES_STATUS	E21	T197	MFG AREA STATUS
RES_NEXT_PM_DATE	E21	T152	NEXT PM DATE
RES_NEXT_PMR	E21	T150	NEXT REQ NO
RES_MAX_PMR	E21	T148	MAX REQ NO
RES_NEXT_OPG	E21	T151	NEXT GROUP NO
RES_MAX_OPG	E21	T149	MAX GROUP NO
RES_TONNAGE_RATING	E21	T154	TONNAGE RATING
RES_TYPE	E21	T155	MFG AREA TYPE

MODE	TITLE	Data Field/Attribute Use Class Mapping	NUMBER	20

Figure 6-9. Data Field/Attribute Use Class Mapping Example

Instructions:

Fill in one or more pages for each database. List all of the set types that map to relation classes and all of the record types that are members in each set type.

<u>Form Area</u>	<u>Explanation</u>
1. DB ID	Unique identification code assigned to the database by the CDMA.
2. Set Type ID	Name or code that the DBMS uses to identify the set type.
3. Member Record Type ID	Name or code that the DBMS uses to identify a record type that is a member in the set type.
4. Ind. E.C. No.	Number of the entity class that is independent in the relation class to which the set type maps.
5. Relation Class Label	Label of the relation class to which the set type maps.
6. Dep. E.C. No.	Number of the entity class that is dependent in the relation class to which the set type maps.

6.1.4 NDDL Commands for Internal Schema

The following is a summary of the NDDL commands required to load and maintain database definitions (internal schemas) and their mappings to the integrated model (conceptual schema):

- The DEFINE DATABASE command is used to add a new database. The command will:
 - Define the DBMS.
 - Define the database name.
 - Describe the host.
 - Record passwords.
 - Associate schemas, subschemas, areas, and files.
 - Supply PSB name and feedback length for IMS databases.

UM 620141001
1 November 1985

USED AT		AUTHOR PROJECT		DATE REV		WORKING		REVISION		DATE		CONTEXT	
		NOTES 1 2 3 4 5 6 7 8 9 10				INVEST		RECOMMENDATION					
DB ID	Set Type ID	Member Record Type ID	Ind. E.C. No.	Relation Class Label	Dep. E.C. No.								
①	②	③	④	⑤	⑥								
TITLE		Set Type/Relation Class Mapping		NUMBER									
NOTE													

Figure 6-10. Set Type/Relation Class Mapping

UM 620141001
1 November 1985

USEDAT		AUTHOR PROJECT		DACOM (CEM, DIRM) 6201M MCM/M		DATE 1983 JULY		X		WORKING		IS AFTER		DATE		CONTEXT	
		NOTES		1 2 3 4 5 6 7 8 9 10													
DB ID		Set Type ID		Member Record Type ID		Ind. E.C. No.		Relation Class Label		Dep. E.C. No.							
2	CAN_BE_PERFORMED_BY	NORTHROP_ALT_RESOURCE		E56		Is Performed By		E57									
2	CARRIED_OUT_BY	ALT_RESOURCE		E10		Is Performed By		E4									
2	CONTAINS	RESOURCE_RESOURCE		E21		Has		E24									
2	CONSISTS_OF	OPERATION		E11		Has		E10									
2	CONTROLLED_BY	STOCK_AREA_ITEM		E7		Is		E30									
2	CONTROLS_STORAGE	STOCK_AREA_ITEM		E29		Stores		E30									
2	DISPATCHES	OP_PLAN_GROUP		E21		Has		E12									
2	GROUPS	OPERATION_PLAN		E12		Has		E13									
2	HOLDS	PERSON_SKILL		E18		Has		E19									
2	IDENT_FOR_PROD	OP_PLAN_PART		E11		Is Used To Manufacture		E15									
2	INITIATES	PROD_MAT_REQ		E11		Has		E61									
2	INITIATES_FOR_GROUP	PROD_MAT_REQ		E12		Has		E64									
2	ISSUES	PROD_MAT_REQ		E21		Issues		E20									
2	IS_ALSO_KNOWN_BY	ALIAS		E21		Is Known By		E3									
2	IS_CONTROLLED_BY	USER_ASSIGNMENT		E38		Has		E23									
MODE		TITLE		Set Type/Relation Class Mapping				NUMBER		30							

Figure 6-11. Set Type/Relation Class Mapping Example

- Once a database is defined to the CDM, any change in the parameters referenced above requires that the database be deleted from the CDM database (DROP DATABASE command) and redefined.

The NDDL DROP DATABASE command will:

- Delete the database from the CDM. All associated record types, record sets, data fields and mappings are also deleted.
 - All descriptive texts for the database will be removed.
- Database record/segment definitions are added to the CDM with the DEFINE RECORD command. This command is used to:
 - Describe a table/record/segment for a previously defined database.
 - Define the key field and whether or not it must be unique.
 - Define what area a record physically resides in.
 - Provide an IMS segment's length.
 - Define the fields/columns/elements/items within tables/records/segments.
- Once a record has been defined to the CDM, any changes require that the record be deleted from the CDM database (NDDL DROP RECORD command) and redefined.

The NDDL DROP RECORD command will:

- Delete the record and all associated fields, areas, and sets.
- If any of the datafields for the deleted record were mapped to the integrated model, their mapping will be dropped. If it was a primary mapping, any secondary mapping, even to other databases, will also be dropped. If it was a secondary mapping, the primary mapping would not be dropped.
- All description texts will be deleted for the record and any fields.

- The DEFINE SET command is used to create sets (paths in IMS). This command can:
 - Define a particular set of a database
 - Relate database records.
 - Define TOTAL LINK fields.
- Any modifications to a set that has already been defined to the CDM database require that the set be deleted with the DROP SET command and redefined.

The DROP SET command is used to delete sets (paths in IMS). This command will:

- Delete the set specified, and all associated mappings.
- Remove all description texts for the set.
- Mappings between the integrated model (entity classes, attribute classes and relation classes) and databases are defined with the NDDL CREATE MAP command. This command can:
 - Map tag names (attribute class) from a conceptual schema entity class to datafields in a Record/Table/Segment.
 - Map an attribute use class to a set.
 - Map a relation class to a set member.
- Mappings between the integrated model (entity classes, attribute classes and relation classes) and databases can be changed with the NDDL ALTER MAP command. This command can:
 - Add a new mapping between a tag name and a datafield.
 - Drop a mapping between a tag name and a datafield. A primary datafield map will not be dropped if secondary maps exist for a particular attribute use class.
 - Modify the primary-secondary indicator and/or the datatype name.
- Mappings between the integrated model (entity classes, attribute classes and relation classes) and databases can be deleted with the NDDL DROP MAP command. This command will delete all mappings.

6.1.5 Loading the CDM Tables

The following tables in the internal schema portion of the CDM database must be loaded directly with NDDL commands:

- Component Data Field Table
- Database Table
- Database Area Table
- Database Area Assignment Table
- Data Field Table
- Data Field Redefinition Table
- Record Set Table
- Record Set Member Table
- Record Type Table
- Repeating Data Field Occurrence Counter Table

In addition, the following tables in the CS-IS portion of the CDM database must also be loaded:

- Attribute Use Class/Data Field Mapping Table
- Entity Class/Record Type Join Table
- Entity Class/Record Type Mapping Table
- Relation Class/Set Type Mapping Table

The loading of the CS-IS mapping tables from the analysis forms is discussed in the following paragraphs.

Attribute Use Class/Data Field Mapping Table

Source Documents:

Data Field/Attribute Use Class Mapping pages from the CS-IS mapping model.

Instructions:

For each record type, use the NDDL CREATE MAP command to map entity class tag names to the record's datafields.

UM 620141001
1 November 1985

<u>Table Field</u>	<u>Source Field</u>
EC No	Entity Class No. column. Use the number following the "E"; do not include the "E" itself.
Tag No	Att. Use Cl. Tag No. column. Use the number following the "T"; do not include the "T" itself.
DB ID	Database ID area near the top of the page.
RT ID	Record Type ID area near the top of the page.
DF ID	Data Field ID column.

Example:

The following Attribute Use Class/Data Field Mapping Table results from the example in Figure 6-12:

<u>EC No</u>	<u>Tag No</u>	<u>DB ID</u>	<u>RT ID</u>	<u>DF ID</u>
8	53	2	Locatn	Loc ID

UM 620141001
1 November 1985

USED AT.	AUTHOR PROJECT	DATE REV	WORKING DRAFT	REVISION	DATE	CONTEXT
	NOTES 1 2 3 4 5 6 7 8 9 10		RECOMMENDED	PUBLICATION		
Database ID.: 2		Record Type ID.: LOCATN				
Data Field ID.		Entity Class No.	Alt. Use Cl. Tag No.	Attribute Use Class Tag		
Loc ID		E8	T53	Stock Area Location ID		
MODE	TITLE	Data Field/Attribute Use Class Mapping			NUMBER	

Figure 6-12. Data Field/Attribute Use Class Mapping Example

Entity Class/Record Type Join Table

Source Documents:

Record Type Join Structure Diagrams from the CS/IS mapping model.

New Instructions:

For each appropriate record type, use the NDDL CREATE MAP command to map the entity class/record type joins.

<u>Table Field</u>	<u>Source Field</u>
EC No	Number in the upper left corner of the entity class box that maps to the record type.
DB ID	Database ID area near the top of the diagram.
RT ID	Record Type ID area near the top of the diagram.
Ind EC No	Number in the upper left corner of the independent entity class box.
Dep EC No	Number in the upper left corner of the dependent entity class box.
RC Label	Verb phrase connected to the relation class line by a squiggle (see sample diagram page).

UM 620141001
1 November 1985

Example:

The following Entity Class/Record Type Join Table results from the example in Figure 6-13:

EC No	DB ID	RT ID	Ind EC No	Dep EC No	RC Label
18	4	Pegging Record	23	5	Is
18	4	Pegging Record	23	12	Is
18	4	Pegging Record	10	18	Is Satisfied By
18	4	Pegging Record	23	18	Is Treated As

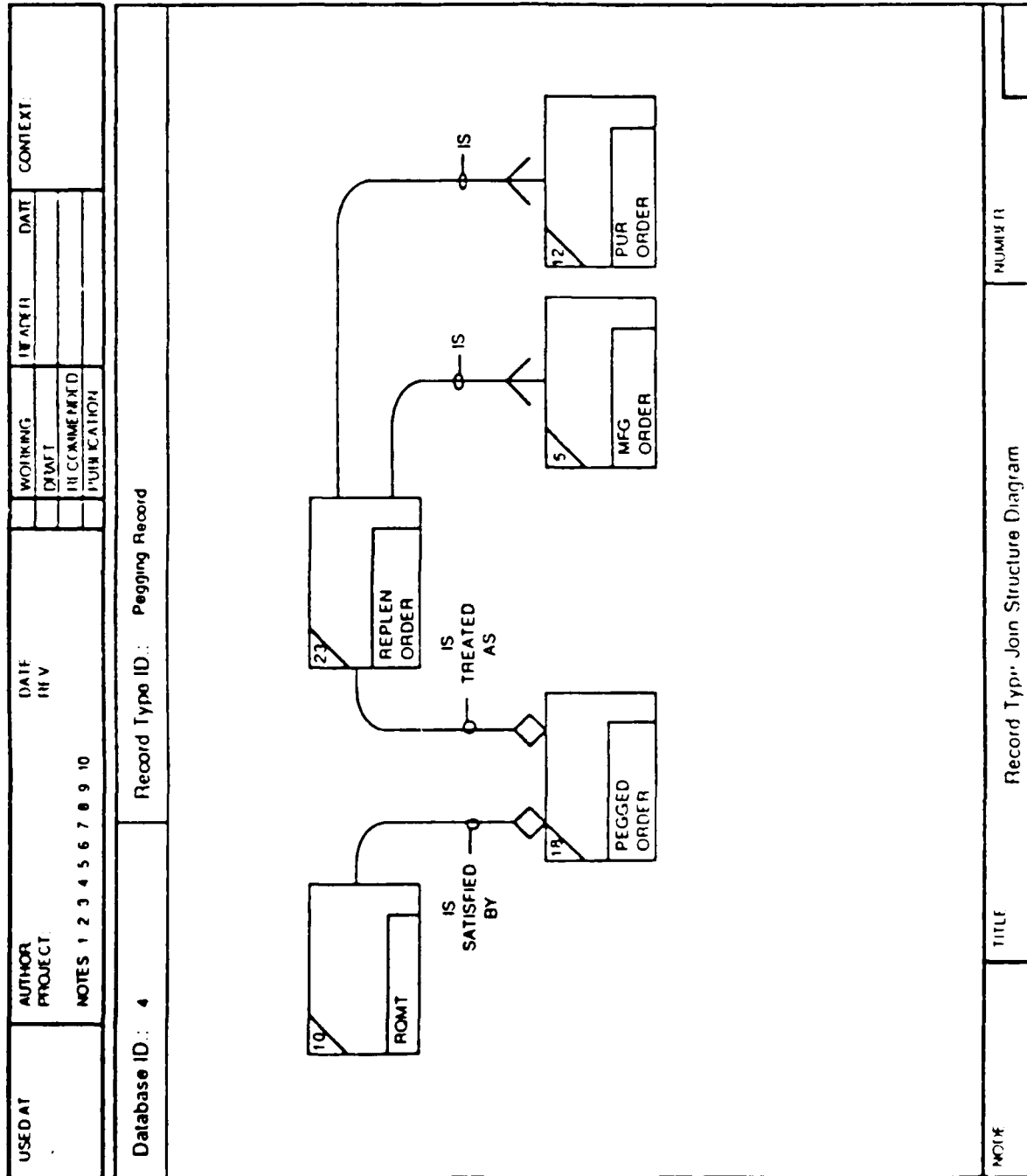


Figure 6-13. Record Type Join Structure Diagram Example

Entity Class/Record Type Mapping Table

Source Documents:

Record Type/Entity Class Mapping pages from the CS/IS mapping model.

Instructions:

For each record type, use the NDDL CREATE MAP command to map the entity class to the record.

<u>Table Field</u>	<u>Source Field</u>
EC No	Entity Class No. column. Use the number following the "E"; do not include the "E" itself.
DB ID	Database ID column.
RT ID	Record Type ID column.

Example:

The following Entity Class/Record Type Mapping Table results from the example in Figure 6-14:

<u>EC No</u>	<u>DB ID</u>	<u>RT ID</u>
8	2	Locatn

UM 620141001
1 November 1985

USE DAT	AUTHOR PROJECT	DATE REV	WORKING DATE	REVIEW DATE	CONTEXT
	NOTES 1 2 3 4 5 6 7 8 9 10		RECOMMENDED PUBLICATION		

Database ID	Record Type ID	Entity Class No.	Entity Class Name
2	LOCATN	E8	Stock Area Location

NOT#	TITLE	Record Type/Entity Class Mapping	NUMBER

Figure 6-14. Record Type/Entity Class Mapping Example

Relation Class/Record Set Mapping Table

Source Documents:

Set Type/Relation Class Mapping pages from the CS/IS mapping model.

Instructions:

For each set, use the NDDL CREATE MAP command to map a relation class to a set (path in IMS).

<u>Table Field</u>	<u>Source Field</u>
Ind EC No	Ind E.C. No. column. Use the number following the "E"; do not include the "E" itself.
Dep EC No	Dep E.C. No. column. Use the number following the "E"; do not include the "E" itself.
RC Label	Relation Class Label column.
DB ID	Database ID column.
Set ID	Set Type ID column.
Member RT ID	Member Record Type ID column.

Example:

The following Relation Class/Record Set Mapping Table results from the example in Figure 6-15:

<u>Ind EC No</u>	<u>Dep EC No</u>	<u>RC Label</u>	<u>DB ID</u>	<u>Set Id</u>	<u>Member RT ID</u>
29	8	Is Composed of	2	Physically Controls	Locatn

UM 620141001
1 November 1985

USED AT		AUTHOR PROJECT		DATE		WORKING		DATE		CONTEXT	
		NOTES 1 2 3 4 5 6 7 8 9 10		DATE		DATE		DATE		DATE	
DB ID	Set Type ID	Member Record Type ID	Ind. E.C. No.	Relation Class Label	Dep. E.C. No.						
2	Physically Controls	LOCATN	E29	Is Composed Of	E8						
MODE	TITLE		Set Type/Relation Class Mapping		NUMBER						

Figure 6-15. Set Type/Relation Class Mapping Example

6.2 Modifying/Deleting IS Elements and CS-IS Mappings

Prior to modifying or deleting elements of the IS or the CS-IS, the CDM Administrator must assess the impact of the proposed change on the other components of the CDM. The objective of this section is to provide the CDM Administrator with an approach to the analysis of the impact that a change in the IS or CS-IS might have upon the other areas of the CDM or on software modules, such as user APs and generated APs.

The approach that is taken in analyzing the impact that a change to the IS or CS-IS might have to other areas of the CDM or to a software module is to list the changes that might be made and then for each of those changes to identify the other changes that would have to be made either in the CS or another schema or in an ES-CS or an IS-CS mapping or in a software module. Changes that do not impact any other areas are omitted.

A similar section appears in the discussions on the Conceptual Schema and on the External Schemas and the ES-CS Mappings, Sections 5 and 7 respectively.

The following assumptions about the nature of the changes to the Internal Schema and the CS-IS Mappings and the sequence in which they are made have been taken in order to perform the analysis:

1. Components of an internal schema are added in the following sequence
 - Databases
 - a database password
 - each database area
 - a schema and subschema name
 - a PCB for an IMS database
 - Record types
 - each database area assignment
 - an IMS segment for a record type in an IMS database
 - Data fields
 - a data field redefinition for a data field that redefines another
 - an elementary data field for a data field that is not part of a group data field
 - a component data field for a data field that is part of a group data field

- a segment data field for a data field in an IMS database
 - a data field/record set linkage for the data field in a TOTAL variable file that is used as a linkpath to a TOTAL master file
 - Record sets
 - Record set members
2. All changes in the internal schema that are needed to support a change in an ES-CS or IS-CS mapping are made before the ES-CS or IS-CS mapping is changed.
 3. A change in the name or definition of a component of the internal schema is for cosmetic purposes only and does not alter the basic meaning of that component.

Finally, a note of explanation about how the changes and their impacts are organized. Only the direct impacts of a change are listed with it. If one change results in a cascade of other changes, only the first in the cascade is listed with the initial change. Each subsequent change is listed as an impact of the one immediately before it. So to find the total extent of the impact of a change, one must trace from the initial change to each change that it results in and, then to each in which that change impacts.

Figure 6-16 shows the relationship between the change and the possible impacts upon other parts of the CDM that the change may affect.

Overview Matrix	A change to:						
	Data- base	D/B Area	Rec. Type	Area Assgn	Data Field	Rec. Set	Set Mbr.
Can impact:							
D/B Area	X						
Record Type	X						
Area Assgn		X	X				
Data Field			X		X		X
Record Set			X				
Set Member			X		X	X	
EC-RT Join			X				
EC-RT Map.			X		X		
AUC-DF Map.			X		X		
AUC-Set Map.						X	
DF Index			X		X		
RC-Set Map.			X				X
Software Mod	X	X	X	X	X	X	X

Figure 6-16. Impact of Internal Schema Changes

6.2.1 Database Changes

Please see section 6.1.4 for a description of NDDL IS Commands. Note that, any modification to a database that has already been defined to the CDM requires the following:

- The database must be deleted from the CDM with the NDDL DROP DATABASE command.
- The revised database must be defined to the CDM with the NDDL DEFINE DATABASE command.
- All datafields, records, sets and mappings must also be defined.

The following is a list of data base related changes and their potential impact:

- Add a new database.

Add all the record types that the database contains.

Use the NDDL DEFINE DATABASE command.

- Change a database name.

Change the database name in any generated APs that access the database and recompile those generated APs.

Change the database name in any software modules that directly access the database and recompile those modules.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a database.

- Change a database definition.

No other impact.

- Change a database keyword.

No other impact.

- Change the DBMS name of a database.

Depending on which two DBMSs are involved, this could have a dramatic impact on the entire structure of an internal schema. A description of this impact is beyond the scope of this report. Suffice it to say that in such cases it would be advisable to delete the database and then to read it using the new DBMS name.

Regardless of the extent of the impact on the internal schema itself, recompile any software modules that resulted in generated APs that access the database.

Also, change and recompile any software modules that directly access the database.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a database.

- Change the host identification for a database.

Recompile any software modules that resulted in generated APs that access the database.

- Change a database password.

Recompile any software modules that resulted in generated APs that access the database.

Change and recompile any software modules that directly access the database.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a database.

- Change a database schema name.

Recompile any software modules that resulted in generated APs that access the database.

Change and recompile any software modules that directly access the database.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a database.

- Change a database subschema name.

Recompile any software modules that resulted in generated APs that access the database.

Change and recompile any software modules that directly access the database.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a database.

- Change the PSB name for an IMS database.

Recompile any software modules that resulted in generated APs that access the database.

Change and recompile any software modules that directly access the database.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a database.

- Change the PCB name for an IMS database.

Recompile any software modules that resulted in generated APs that access the database.

Change and recompile any software modules that directly access the database.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a database.

- Change the key feedback length for an IMS database.

Recompile any software modules that resulted in generated APs that access the database.

Change and recompile any software modules that directly access the database.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a database.

- Delete a database.

Delete all the record types in the database.

Delete any database password for the database.

Delete any schema names for the database.

Delete any database areas in the database.

Delete the PCB if this is an IMS database.

Use the NDDL DROP DATABASE command.

Recompile any software modules that resulted in generated APs that access the database.

Change and recompile all the software modules that directly access the database or discard them entirely.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a database.

6.2.2 Database Area Changes

The following is a summary of the impact of charger to database areas: (See 6.1.4 for the NDDL.)

- Add a new database area.

No other impact.

- Delete a database area.

Delete any database area assignments for the database area.

Recompile any software modules that resulted in generated APs that use area searches to access record types in the database area.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that use database area searches to access record types.

Change and recompile any software modules that use area searches to access record types in the database area.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that use database area searches to access record types.

6.2.3 Record Type Changes

The following is a summary of the impact of changes to record types:

- Add a new record type. Add all the data fields that the record type contains. Use the NDDL DEFINE RECORD command.

Add any EC-RT joins from which the record type results.

Add an EC-RT mapping from for any entity class to which the record type maps. Use the NDDL CREATE MAP command.

If the record type is assigned to one or more database areas, add a database area assignment for each.

Note: Any change to a record type requires that it first be deleted with the NDDL DROP RECORD command then added with the DEFINE

RECORD command. The set must be added with the DEFINE SET command and then all mappings must be added with the CREATE MAP command.

- Change a record type name.

Change the record type name in all the following in which it appears:

- Data fields
- Data field redefinitions
- Elementary data fields
- Component data fields
- Record sets
- Record set members
- Database area assignments
- IMS segment
- Segment data fields
- Data field/record set linkages
- EC-RT mappings
- Horizontal partitions
- EC-RT union discriminators
- AUC-DF mappings
- Repeating data field indexes
- RC-Set mappings
- EC-RT joins

Recompile any software modules that resulted in generated APs that access the record type.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access a record type.

Change and recompile any software modules that directly access the record type.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a record type.

- Change a record type definition.
No other impact.
- Change a record type keyword.
No other impact.
- Change the segment name of a record type in an IMS database.

Recompile any software modules that resulted in generated APs that access the record type.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access a record type.

Change and recompile any software modules that directly access the record type.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a record type.

- Change the segment size of a record type in an IMS database.

Recompile any software modules that resulted in generated APs that access the record type.

Note: Neither the CDM database nor the CDM1 model

AD-A181 577

INTEGRATED INFORMATION SUPPORT SYSTEM (IIS) VOLUME 5

3/4

COMMON DATA MODEL 5 (U) GENERAL ELECTRIC CO

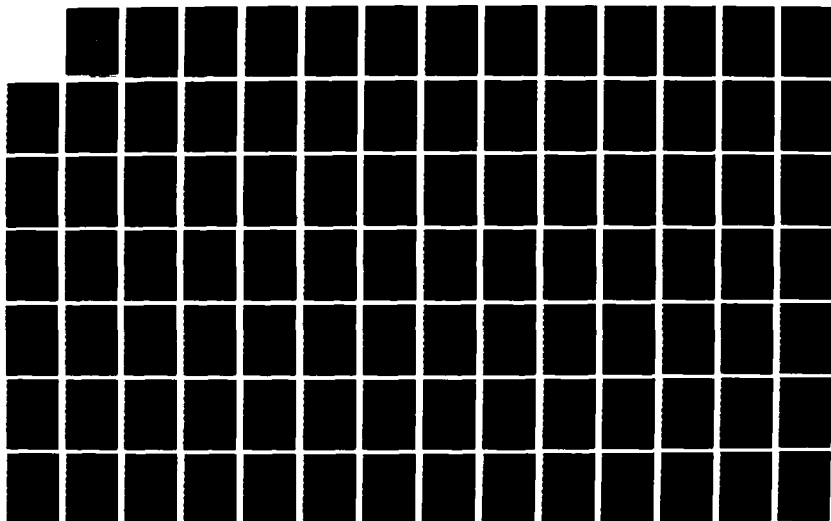
SCHENECTADY NY PRODUCTION RESOURCES CONSU

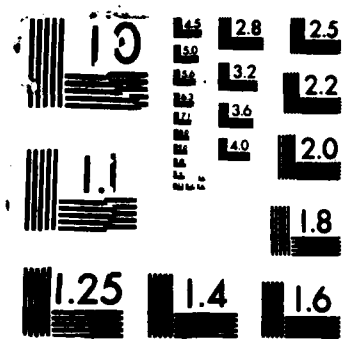
UNCLASSIFIED

D ROLLINS ET AL 81 NOV 85 U--628141881

F/G 5/2

NL





contains the information needed to identify the generated APs that access a record type.

Change and recompile any software modules that directly access the record type.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a record type.

- Change the mapping between a record type and the entity classes to which it corresponds.

Add, change, and delete any of the following as necessary:

EC-RT mappings

Horizontal partitions and constraint statements,
EC-RT union discriminators

AUC-DF mappings

Use the Alter Map command.

Recompile any software modules that resulted in generated APs that access the record type.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access a record type.

- Change the entity class joins that must be done to form a record type.

Add, change, and delete EC-RT joins as necessary.

Use the ALTER MAP command.

Recompile any software modules that resulted in generated APs that access the record type.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify

the generated APs that access a record type.

- Delete a record type.

Delete all the data fields that the record type contains.

Delete any record sets in which the record type is owner.

Delete any record set members that the record type is used as.

Delete any database area assignments for the record type.

Delete the IMS segment if the record type is in an IMS database.

Delete any EC-RT mappings for the record type.

Delete any horizontal partitions for the record type.

The NDDL DROP RECORD command will accomplish all of the above.

Recompile any software modules that resulted in generated APs that access the record type.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access a record type.

Change and recompile any software modules that directly access the record type or discard them entirely.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a record type.

6.2.4 Database Area Assignment Changes

The following is a summary of the impact of changes to

database area assignments. (See 6.2.1 for the NDDL statements.)

- Add a new database area assignment.

Recompile any software modules that resulted in generated APs that use area searches to access the record type referenced in the database area assignment.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that use database area searches to access a record type.

Change and recompile any software modules that use area searches to directly access the record type referenced in the database area assignment.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a record type.

- Delete a database area assignment.

Recompile any software modules that resulted in generated APs that use area searches to access the record type referenced in the database area assignment.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that use database area searches to access a record type.

Change and recompile any software modules that use area searches to directly access the record type referenced in the database area assignment.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a record type.

6.2.5 Data Field Changes

Note that a change to the structure of a record type requires that the record first be deleted with the NDDL DROP RECORD command. Then the modified record type is added again with the NDDL DEFINE RECORD command. All mappings must also be added with the DEFINE SET and CREATE MAP commands.

The following is a summary of the impact of changes to data fields:

- Add a new data field.

If the data field redefines another data field, add a data field redefinition.

If the data field is not a group of component data fields, add an elementary data field.

If the data field is a component of another data field, add a component data field.

If the data field is in an IMS database, add a segment data field.

If the data field is used as a linkpath from a TOTAL variable file to a TOTAL master file, add a data field/record set linkage.

Add an AUC-DF mapping for any attribute use class to which the data field maps.

Add an EC-RT union discriminator for any EC-RT mapping that the data field is used to distinguish among.

If the data field occurs more than once in the record type, add a repeating data field index for any attribute use class that is used to access the data field.

- Change a data field name.

Use the NDDL DROP RECORD, DEFINE RECORD, DEFINE SET, and CREATE MAP commands.

Change the data field name in all the following in

which it appears:

Data field redefinitions

Elementary data field

Component data fields

EC-RT union discriminators

AUC-DF mappings

Repeating data field indexes

Segment data field

Data field/record set linkage

Recompile any software modules that resulted in generated APs that access the data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access a data field.

Change and recompile any software modules that directly access the data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a data field.

- Change a data field definition.

No other impact.

- Change a data field keyword.

No other impact.

- Change the record key code of a data field.

Recompile any software modules that resulted in generated APs that access the data field.

Note: Neither the CDM database nor the CDM1 model

contains the information needed to identify the generated APs that access a data field.

Change and recompile any software modules that directly access the data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a data field.

- Change the number of occurrences of a data field.

If the number of occurrences is being changed from one to something else, add a repeating data field index for any attribute use class that can be used to access the data field.

If the number of occurrences is being changed to one from something else, delete any repeating data field indexes for the data field.

Recompile any software modules that resulted in generated APs that access the data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access a data field.

Change and recompile any software modules that directly access the data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a data field.

- Change which data field is redefined by a data field.

Recompile any software modules that resulted in generated APs that access the data field, i.e., the one that redefines another, not the one being redefined.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access a data

field.

Change and recompile any software modules that directly access the data field, i.e., the one that redefines another, not the one being redefined.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a data field.

- Change which group data field of which a component data field is part.

Recompile any software modules that resulted in generated APs that access the component data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access a data field.

Change and recompile any software modules that directly access the component data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a data field.

- Change the data description of a data field.

Use the NDDL ALTER MAP command.

Recompile any software modules that resulted in generated APs that access the data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access a data field.

Change the data description of the data field in any software modules that directly access it and recompile those software modules.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a

data field.

- Change the segment start byte of an IMS data field.

Use the NDDL DROP SEGMENT, DEFINE SEGMENT, DEFINE SET and CREATE MAP commands.

Recompile any software modules that resulted in generated APs that access the data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access a data field.

Change and recompile any software modules that directly access the data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a data field.

- Change the IMS data field indicator of a data field.

Recompile any software modules that resulted in generated APs that access the data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access a data field.

Change and recompile any software modules that directly access the data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a data field.

- Change the record set name of a TOTAL data field.

Use the NDDL DROP SET, DEFINE SET and CREATE MAP commands.

Recompile any software modules that resulted in generated APs that access the data field.

Note: Neither the CDM database nor the CDM1 model

contains the information needed to identify the generated APs that access a data field.

Change and recompile any software modules that directly access the data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a data field.

- Change the linkage type code of a TOTAL data field.

Recompile any software modules that resulted in generated APs that access the data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access a data field.

Change and recompile any software modules that directly access the data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a data field.

- Change the mapping between a data field and the attribute use classes to which it corresponds.

Add, change, and delete AUC-DF mappings as necessary.

Use the NDDL ALTER MAP commands.

Recompile any software modules that resulted in generated APs that access the data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access a data field.

- Delete a data field.

Delete any data field redefinition that the data

field is used as.

Delete the data field redefinition by which the data field is redefined. If there is more than one, delete only one of them and replace the redefined data field name in all the others with the redefining data field name from the one deleted.

Delete any elementary data field for the data field. Delete any component data fields of which the data field is a group.

Delete any component data field that the data field is used as.

Delete any segment data field for the data field.

Delete any data field/record set linkage that the data field appears as.

Delete any AUC-DF mappings in which the data field is involved.

Delete any EC-RT union discriminators that the data field is used as.

Delete any repeating data field indexes by which the data field is accessed.

Use the NDDL DROP RECORD, DEFINE RECORD, DEFINE SET and CREATE MAP commands.

Recompile any software modules that resulted in generated APs that access the data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access a data field.

Change and recompile any software modules that directly access the data field.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a data field.

6.2.6 Record Set Changes

The following is a summary of the impact of change to record sets:

- Add a new record set.

Add all the record set members that the record set has.

Add an AUC-Set mapping if the record set maps to a value for an attribute use class.

Use the NDDL DEFINE SET and CREATE MAP command.

- Change a record set name.

Change the record set name in all the following in which it appears:

Record set members

AUC-Set mappings

Use the NDDL DROP SET, DEFINE SET and CREATE MAP commands.

Recompile any software modules that resulted in generated APs that access record types via the record set.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access record types via a record set.

Change and recompile any software modules that directly access record types via the record set.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access record types via a record set.

- Change a record set definition.

No other impact.

- Change a record set keyword.

No other impact.

- Change the total number of members in a record set.

The change of the total number of members in a record set is never initiated on its own; it is always the result of either the addition or deletion of a record set member.

No other impact.

- Change which record type is owner in a record set.

Use the NDDL DROP SET, DEFINE SET and CREATE MAP commands.

Recompile any software modules that resulted in generated APs that access record types via the record set.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access record types via a record set.

Change and recompile any software modules that directly access record types via the record set.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access record types via a record set.

- Change the mapping between a record set and the attribute use class to which it corresponds.

Add, change, and delete AUC-Set mappings as necessary.

Use the NDDL ALTER MAP command.

Recompile any software modules that resulted in generated APs that access record types via the record set.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access record types via a record set.

- Delete a record set.

Delete all the record set members that the record set has.

Delete any AUC-Set mapping for the record set.

Use the NDDL DROP SET command.

Recompile any software modules that resulted in generated APs that access record types via the record set.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access record types via a record set.

Change and recompile any software modules that directly access record types via the record set.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access record types via a record set.

6.2.7 Record Set Member Changes

The following is a summary of the impact of changes to record set members:

- Add a new record set member.

Add an RC-Set mapping for any relation class that is the basis for the record set member.

Increase the total number of members in the record set by one.

Use the NDDL DROP SET, DEFINE SET, and ALTER MAP commands.

- Change the required membership indicator of a record set member.

Use the NDDL DROP SET, DEFINE SET and CREATE MAP commands.

Recompile any software modules that resulted in generated APs that update the record type that is used as the record set member.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that update record types.

Change and recompile any software modules that directly update the record type that is used as the record set member.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly update record types.

- Change the mapping between a record set member and the inherited key classes on which it is based.

Add and delete RC-Set mappings as necessary.

Use the NDDL ALTER MAP command.

Recompile any software modules that resulted in generated APs that access the record set member.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access record set members.

- Delete a record set member.

Decrease the total number of members in the record set by one.

Delete any RC-Set mappings on which the record set member is based.

Delete any data field/record set linkage for the

record set member.

Use the NDDL ALTER MAP command.

Recompile any software modules that resulted in generated APs that access the record type that is used as the record set member.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the generated APs that access a record type.

Change and recompile any software modules that directly access the record type that is used as the record set member.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that directly access a record type.

6.2.8 Summary

The following points are offered in summary:

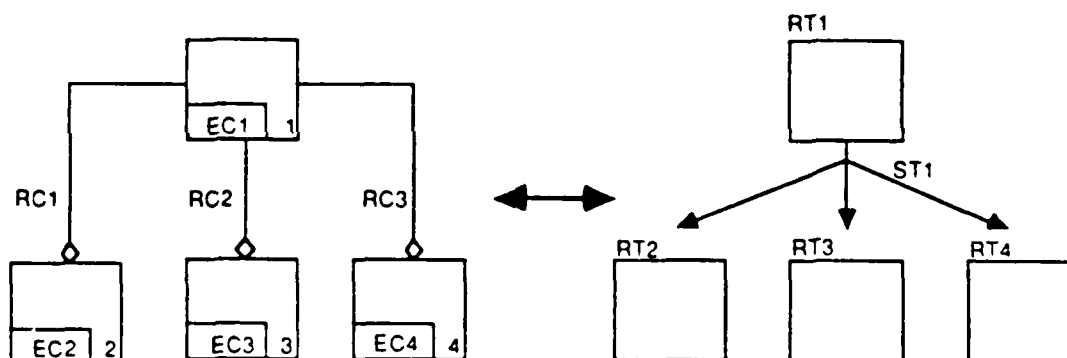
1. A change in an internal schema can result in additional changes in that schema, in its IS-CS mapping, and in software modules. However, it cannot impact other internal schemas or IS-CS mappings, nor any external schemas or ES-CS mappings, nor the conceptual schema.
2. A change in IS-CS mapping is always the result of another change to either the corresponding internal schema or to the conceptual schema.
3. The information in the CDM database and the CDM1 model is inadequate for identifying the software modules that are impacted by most schema changes. Specifically, the following information needs to be added:
 - The data items that are accessed by a software module that contains user views.
 - The databases, record types, data fields, record sets, record set members, and database areas that are accessed by a software module that

- accesses databases directly.
The record types, data fields, record sets, record set members, and database areas that are accessed by a generated AP.

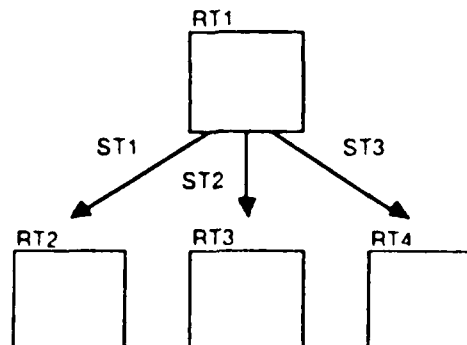
6.3 CODASYL Databases

6.3.1 CODASYL-Specific Considerations

CODASYL DBMSs offer two database design features that are not available in most others: multi-member set types and optional membership set types. The first is a single set type that has one owner record type and two or more member record types. An owner instance can be associated with any number of instances of each type of member; they are not mutually exclusive. In essence, several logical relationships (relation classes) are combined into one physical relationship (set type).



RATHER THAN



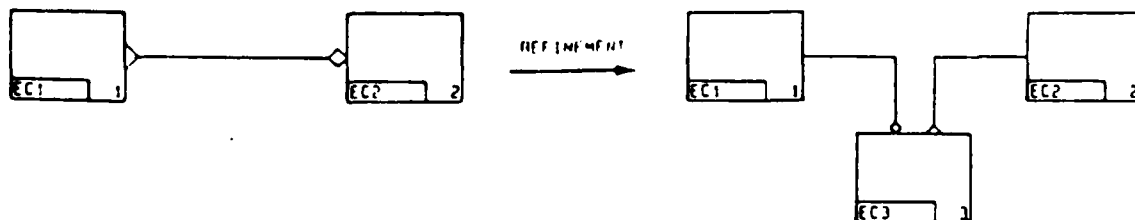
The CS-IS mapping for a multi-member set type involves the following:

- The owner and member record types each have a primary mapping to a different entity class. Any of them can have secondary mappings also.
- The set type maps to several relation classes, one per member.
- The entity class that the owner maps to is independent in all of these relation classes.
- Each entity class that a member maps to is dependent in one of these relation classes.

In the example above:

RT1 maps to EC1.
RT2 maps to EC2.
RT3 maps to EC3.
RT4 maps to EC4.
ST1 maps to RC1, RC2, and RC3.

An optional membership set type is one in which an instance of the member record type is allowed to exist without being associated with an instance of the owner record type. This is in contrast with any other set type in which every member instance must be associated with an owner instance. An optional membership set type is equivalent to a nonspecific relation class whose cardinality is zero-or-one-to-zero-one-or-many. Such a relation class is refined, as shown below, before it is incorporated into the conceptual schema.



Consequently, the CS-IS mapping for an optional membership set type involves the following:

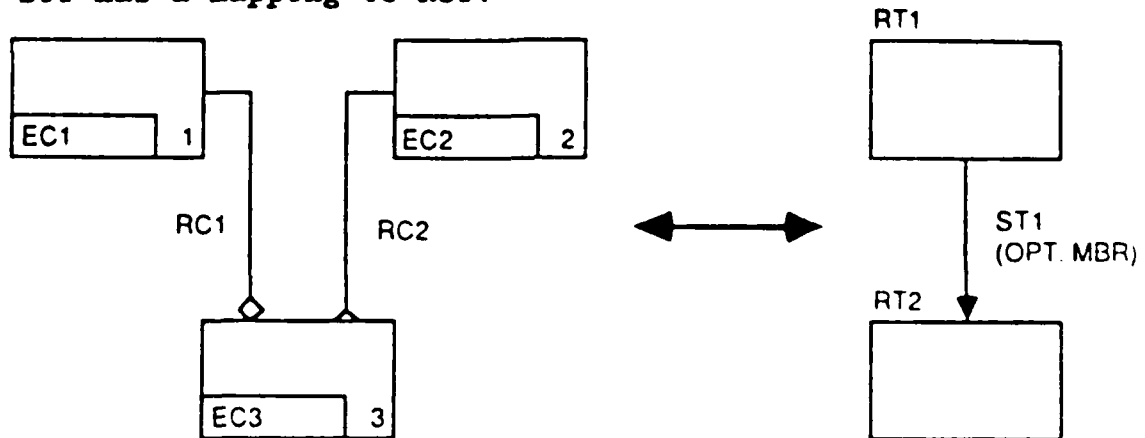
- The owner record type has a primary mapping to one entity class, and the member record type has a

primary mapping to another, and a secondary mapping to a third. Either one can have additional secondary mappings also.

- The set type maps to a one-to-many relation class.
- The entity class that the owner maps to is independent in that relation class.
- The secondary entity class for the member is dependent in that relation class.
- The primary and secondary entity classes for the member are independent and dependent, respectively, in a one-to-zero-or-one relation class, which is a join linkage for the member.

In the example above:

RT1 has a primary mapping to EC1.
RT2 has a primary mapping to EC2.
RT2 has a secondary mapping to EC3.
ST1 has a mapping to RC1.



6.3.2 Building a CODASYL IS and CS-IS Mapping

Objectives:

- Load the description of a CODASYL database into the following tables in the CDM database:

Database Table
Record Type Table

Database Area Table
Database Area Assignment Table
Data Field Table
Component Data Field Table
Data Field Redefinition Table
Repeating Data Field Occurrence Counter Table
Record Set Table
Record Set Member Table

- Build a model of the mapping between the CODASYL database and the conceptual schema.
- Load the descriptions of the CS-IS mapping into the following tables in the CDM database:

Entity Class/Record Type Mapping Table
Entity Class/Record Type Join Table
Relation Class/Set Type Mapping Table
Attribute Use Class/Data Field Mapping Table

Refer to Section 6.13 for details on how to fill out the CS-IS mapping forms.

If the CDM tables are to be loaded with NDDL commands, skip to Section 6.3.3. Building a CODASYL IS and CS-IS Mapping with the NDDL.

Tasks:

1. The CDM Administrator loads descriptions from the database DDL statements.

Create one entry in the Record Type Table for each record type in the database.

Create one entry in the Database Area Assignment Table for each record type that is assigned to a database area. If a record type is assigned to more than one area, create a table entry for each.

Create one entry in the Data Field Table for each data field in each record type in the database.

Create one entry in the Component Data Field Table for each data field that is part of another data field.

Create one entry in the Data Field Redefinition Table for each data field that redefines another data field.

Create one entry in the Repeating Data Field Occurrence Counter Table for each data field that occurs more than once in a record type.

Create one entry in the Record Set Table for each set type in the database.

Create one entry in the Record Set Member Table for each record type that is a member of a set type. If a record type is a member of more than one set type, create a table entry for each. If a set type has more than one member record type, create a table entry for each.

2. The CDM Administrator determines the mapping for each record type.

Usually, it is easier to map the record types that are not members in any set types first. Those that are set type members should not be mapped until all of their owner record types have been mapped.

Determine what sort of "real-world" thing the record type represents. Each instance of a record type contains data about a specific person, place, object, etc., that is significant to the enterprise. Usually, all of the instances of the same type are about the same sort of thing. This is not always the case, however. An instance of the RESUPPLY-ORDER record type could represent either an order to the production department to make a certain quantity of parts (i.e., a manufacturing order) or an order to a vendor to furnish a certain quantity of parts (i.e., a purchase order). This is similar to defining an entity class. The data fields in the record type, especially those that uniquely identify its instances, and the set types that it participates in, especially as a member, can all be useful in determining what the record type represents.

A few record types do not represent real-world things; they exist to improve database performance. Examples include SYSTEM-OWNER and entry points. Such

record types do not map to any entity classes and can be ignored.

Determine which entity class in the conceptual schema represents the same sort of thing as the record type. This primarily involves finding the entity class whose definition corresponds to the intent of the record type. Comparing the key classes, attribute use classes, and relation classes of the entity classes to the keys, data fields, and set types of the record types can be helpful also. If the record type represents several sorts of things, it will map to several entity classes, one for each sort of thing; see Section 6.1.2.4 regarding relational unions. If none of the entity classes represent what the record type does, either the record type exists only to improve database performance or the conceptual schema must be expanded; see Section 4.3.

Fill out a line on a Record Type/ Entity Class Mapping Form for each entity class to which the record type maps.

3. The CDM Administrator determines the mapping for each data field.

Determine what sort of data about real-world things that the data field contains. If the record type that contains the data field represents more than one sort of thing, i.e., if it has more than one mapping, the data field may contain several sorts of data. All of these must be identified.

A few data fields do not contain data about real-world things; they exist for technical reasons only. Examples include record codes and record activity dates. Such data fields do not map to any attribute use classes and can be ignored.

Determine which attribute use classes in the conceptual schema represent the same sort of data as the data field. This involves finding the attribute use class whose definition or migration path corresponds to the intent of the data field. The first place to look is the entity class to which the record type maps. If the record type maps to more

than one entity class, the data field may map to an attribute use class in each. The value in the data field in each instance of the record type must be the same as the one in the attribute use class in the corresponding instance of the entity class. If two or more inherited attribute use classes that come from the same owned attribute use class have identical values in every entity instance, the data field may map to some or all of them.

If none of the attribute use classes in the mapped-to entity class(es) correspond to the data field, the next places to look are the entity classes that are related to those entity class(es). Again, the value in each record instance must be the same as the value in the corresponding entity instance. If the attribute use class is not in any of these entity classes, the search must be widened to include the entity classes that are related to them. This continues until the proper attribute use class is found or until it is determined that a new attribute class must be added to the conceptual schema; see Section 4.3.

Fill out a line on a Data Field/ Attribute Use Class Mapping Form for each attribute use class to which the data field maps.

4. The CDM Administrator determines any joins that are needed for each record type.

Determine whether any of the data fields in the record type map to attribute use classes that are not in the entity class(es) to which the record type maps. This can be done by comparing the entity class numbers that are entered on the Data Field/Attribute Use Class Mapping Forms for the record type to those that are entered on the Record Type/Entity Class Mapping Form for the record type. If an entity class number is on the first form but not on the second, that entity class must be joined with the one to which the record type maps.

Determine whether any other entity classes are needed to complete the join structure(es). The entity classes that must be joined to form the record type must form one or more join structures as described in

Section 6.1.2.3. If the join structures are not contiguous, one or more additional joins may be needed. For example, if the record type in Figure 6-17 maps to EC4 and involves joins with EC1 and EC3, it must also have a join with EC2. Without it, EC1 cannot be joined to the EC3-EC4 join result. The join must involve EC2 even though none of its attribute use classes map to data fields in the record type.

Prepare Record Type Join Structure Diagrams for the record types involve joins.

5. The CDM Administrator determines the mapping for each record set.

Determine what sort of relationship between "real-world" things the set type represents. If the set type has more than one member record type, each must be considered separately. If either the owner or the member record type has no mapping to an entity class, the set type will have no mapping to a relation class, so it can be ignored.

Determine which relation class in the conceptual schema represents the same sort of relationship as the set type. Usually, this is the relation class whose independent entity class maps to the owner record type and whose dependent entity class maps to the member record type.

Fill out a line on a Set Type/ Relation Class Mapping Form for the relation class to which the set type maps.

6. The CDM Administrator loads descriptions from the Record Type/Entity Class Mapping Forms.

Create one entry in the Entity Class/Record Type Mapping Table from each line on each form.

7. The CDM Administrator loads descriptions from the Field/Attribute Use Class Mapping Forms.

Create one entry in the Entity Class/Record Type Join Table for each relation class in a diagram.

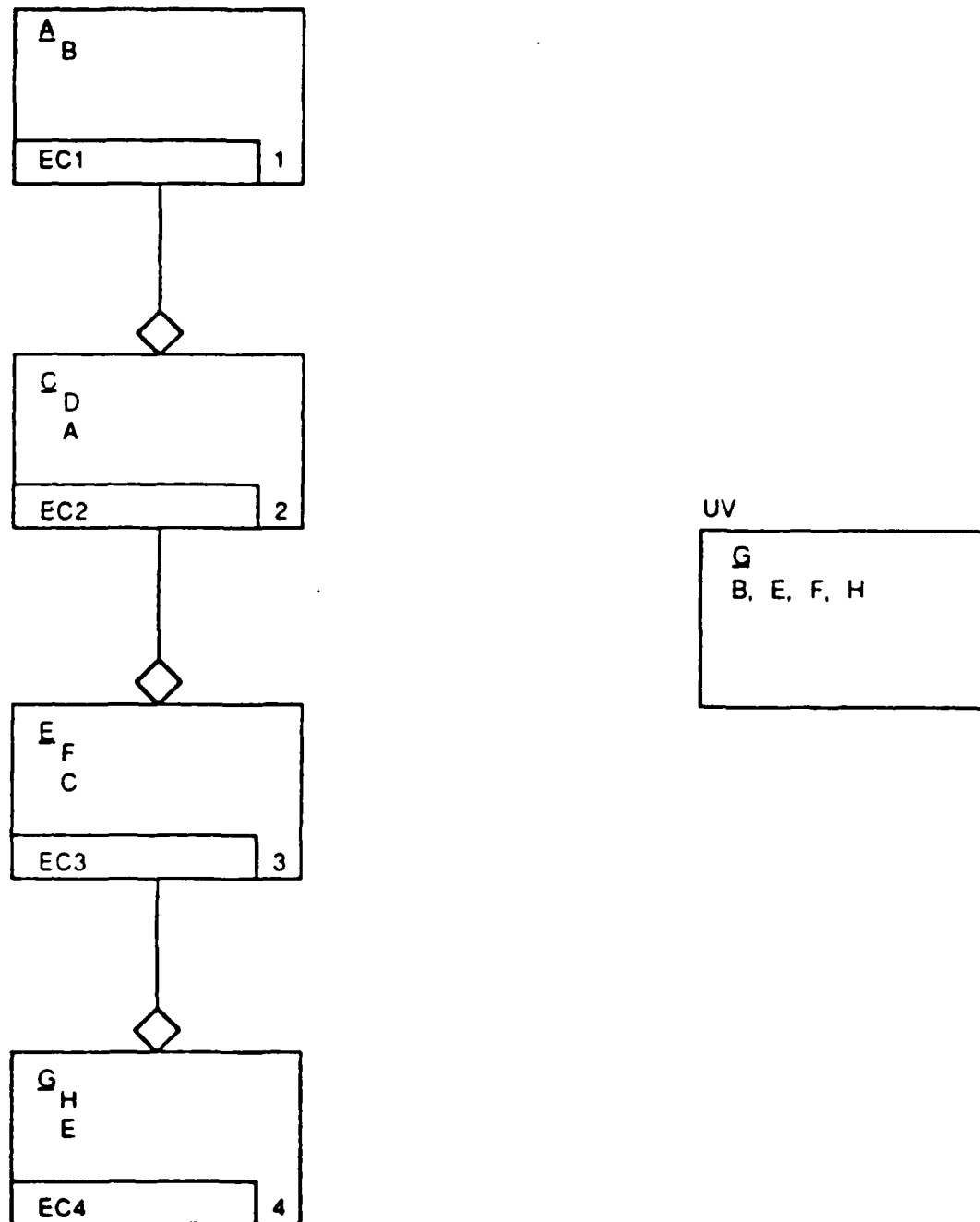


Figure 6-17. Incomplete Join Structure Example

8. The CDM Administrator loads descriptions from the Data Field/Attribute Use Class Mapping Forms.

Create one entry in the Attribute Use Class/Data Field Mapping Table from each line on each form.

9. The CDM Administrator loads descriptions from the Set Type/Relation Class Mapping Forms.

Create one entry in the Relation Class/Set Type Mapping Table from each line on each form.

6.3.3 Building a CODASYL IS and CS-IS Mapping with NDDL

A summary of the NDDL commands required to load and maintain an IS and CS-IS Mappings is contained in Section 6.1.4.

Tasks:

1. Load descriptions from the database DDL statements.

- Use the NDDL DEFINE DATABASE command to:

- Define the database name.
- Define the DBMS.
- Describe the host.
- Record password(s).
- Define schemas, subschemas, and areas.

- For each record in the database, Use the NDDL DEFINE RECORD command to:

- Attach record to database.
- Define the key field and state whether or not it must be unique.
- Define the datafields within the record.
- Define what area a record is to physically reside in.

Note: The DEFINE RECORD syntax does not support repeating groups, component data fields, or redefined data fields.

- For each set in the database, use the NDDL DEFINE SET command to:

- Define the set.
 - Relate database records.
2. Determine the primary mapping for each record type.
 - See Task 2 of Section 6.3.2.
 3. Determine the primary mapping for each data field.
 - See Task 3 of Section 6.3.2
 4. Determine any secondary mappings for each record type.
 - See Task 4 of Section 6.3.2.
 5. Determine any secondary mappings for each data field.
 - See Task 5 of Section 6.3.2.
 6. Determine the mapping for each set type.
 - See Task 6 of Section 6.3.2.
 7. Load descriptions from the Record Type/Entity Class Mapping Form, the Record Type Join Structure Diagrams, the Data Field/Attribute Use Class Mapping Forms, and the Set Type/Relation Class Mapping Forms.
 - For each record, use the NDDL CREATE MAP command to:
 - Map tag names (attribute classes) from a conceptual schema entity class to a datafield.
 - Map attribute use class to a set.
 - For each set, use the NDDL CREATE MAP command to:
 - Map a conceptual schema relation class to a set.

6.3.4 Modifying a CODASYL IS and CS-IS Mapping Objective:

Add one or more new record type(s) as follows:

- Load the description of a new record type(s) within a

previously mapped CODASYL database into the following tables in the CDM database:

Record Type Table
Database Area Table
Database Area Assignment Table
Data Field Table
Component Data Field Table
Data Field Redefinition Table
Repeating Data Field Occurance Counter Table
Record Set Table
Record Set Member Table

- Build a model of the new mapping between the CODASYL database and the conceptual schema.
- Load the description of the new CS-IS mapping into the following tables in the CDM database:

Entity Class/Record Type Mapping Table
Entity Class/Record Type Join Table
Relation Class/Set Type Mapping Table
Attribute Use Class/Data Field Mapping Table

Refer to Section 6.1.3 for details on how to fill out the CS-IS mapping forms.

If the CDM tables are to be modified with the NDDL commands, skip to Section 6.3.5: Add One or More New Record Type(s) to a CODASYL Database with the NDDL.

Tasks:

1. The CDM Administrator loads descriptions for the new record type from the database DDL statements.

Create one entry in the Database Area Table for each new area (if any) in the database.

Create one entry in the Record Type Table for each new record type added to the database.

Create one entry in the Database Area Assignment Table for each new record type that is assigned to a database area.

Create one entry in the Data Field Table for each

data field in each new record type added to the database.

Create one entry in the Component Data Field Table for each data field that is part of another data field.

Create one entry in the Data Field Redefinition Table for each data field that redefines another data field.

Create one entry in the Repeating Data Field Occurrence Counter Table for each data field that occurs more than once in a record type.

Create one entry in the Record Set Table for each set type the new record type participates in as an owner.

Create one entry in the Record Set Member Table for each new record type that is a member of a set type. If a new record type is a member of more than one set create a table entry for each.

2. The CDM Administrator determines the mapping for each new record type.

Usually, it is easier to map the record types that are not members in any set types first. Those that are set type members should not be mapped until all of their owner record types have been mapped.

Determine what sort of "real-world" thing the new record type represents. Each instance of a record type contains data about a specific person, place, object, etc., that is significant to the enterprise. Usually, all of the instances of the same type are about the same sort of thing. This is not always the case, however. An instance of the RESUPPLY-ORDER record type could represent either an order to the production department to make a certain quantity of parts (i.e., a manufacturing order) or an order to a vendor to furnish a certain quantity of parts (i.e., a purchase order). This is similar to defining an entity class. The data fields in the record type, especially those that uniquely identify its instances, and the set types that it participates in, especially as a member, can all be useful in

determining what the record type represents.

A few record types do not represent real-world things; they exist to improve database performance. Examples include SYSTEM-OWNER and entry points. Such record types do not map to any entity classes and can be ignored.

Determine which entity class in the conceptual schema represents the same sort of thing as the new record type. This primarily involves finding the entity class whose definition corresponds to the intent of the record type. Comparing the key classes, attribute use classes, and relation classes of the entity classes to the keys, data fields, and set types of the record type can be helpful also. If the record type represents several sorts of things, it will map to several entity classes, one for each sort of thing; see Section 6.1.2.4 regarding relational unions. If none of the entity classes represent what the record type does, either the record type exists only to improve database performance or the conceptual schema must be expanded; see Section 4.3.

Fill out a line on a Record Type/Entity Class Mapping Form for each entity class to which the new record type maps.

3. The CDM Administrator determines the mapping for each data field in the new record type.

Determine what sort of data about real-world things that the data field contains. If the record type that contains the data field represents more than one sort of thing, i.e., if it has more than one primary mapping, the data field may contain several sorts of data. All of these must be identified.

A few data fields do not contain data about real-world things; they exist for technical reasons only. Examples include record codes and record activity dates. Such data fields do not map to any attribute use classes and can be ignored.

Determine which attribute use classes in the conceptual schema represent the same sort of data as

the data field. This involves finding the attribute use class whose definition or migration path corresponds to the intent of the data field. The first place to look is the entity class to which the record type maps. If the record type maps to more than one entity class, the data field may map to an attribute use class in each. The value in the data field in each instance of the record type must be the same as the one in the attribute use class in the corresponding instance of the entity class. If two or more inherited attribute use classes that come from the same owned attribute use class have identical values in every entity instance, the data field may map to some or all of them.

If none of the attribute use classes in the mapped-to entity class(es) correspond to the data field, the next places to look are the entity classes that are related to those entity class(es). Again, the value in each record instance must be the same as the value in the corresponding entity instance. If the attribute use class is not in any of these entity classes, the search must be widened to include the entity classes that are related to them. This continues until the proper attribute use class is found or until it is determined that a new structure class must be added to the conceptual schema; See Section 4.3.

Fill out a line on a Data Field/Attribute Use Class Mapping Form for each attribute use class to which the data field maps.

4. The CDM Administrator determines any joins that are needed for each new record type.

Determine whether any of the data fields in the record type map to attribute use classes that are not in the entity class(es) to which the record type maps. This can be done by comparing the entity class numbers that are entered on the Data Field/Attribute Use Class Mapping Forms for the record type to those that are entered on the Record Type/Entity Class Mapping Form for the record type. If an entity class number is on the first form but not on the second, that entity class must be joined with the one to which the record type maps.

Determine whether any other entity classes are needed to complete the join structure(s). The entity classes that must be joined to form record type must form one or more join structures as described in Section 6.1.2.3. If the join structures are not contiguous, one or more additional joins may be needed. For example, if the record type in Figure 6-18 maps to EC4 and involves joins with EC1 and EC3, it must also have a join to EC2. Without it, EC1 cannot be joined to the EC3-EC4 join result. The join must involve EC2 even though none of its attribute use classes map to data fields in the record type.

Prepare Record Type Join Structure Diagrams for each new record type that has any secondary mappings.

5. The CDM Administrator determines the mapping for each record set in which the new record type is a member.

6.4 Relational Databases

6.4.1 Relational-Specific Considerations

A relational DBMS provides a simple, uniform way of looking at data, which is completely independent of actual storage structures and of access techniques used to retrieve the data.

All relationships between data are expressed in terms of the actual data values, not by pointers or storage adjacency. The ability to relate common fields of data found in more than one table is provided by the data access language. This enables a user or program to specify the desired data in terms of properties the data possess.

In a hierarchical or network DBMS data model, access paths are predefined in the data structure definition. A user or program can use only the predefined paths to navigate through the data structure. This could limit the use of the data. However, it is a strength if only those paths are needed, because the system can provide quick access through the predefined paths.

In a relational data model, paths need not be predefined. Data requests are not expressed in terms of access paths. All access is done by matching field values.

Relational DBMS(s) represent data as relations or two-dimensional tables. These tables consist of attributes (columns) and tuples (rows). Each entity in a database will have a corresponding relation defined, which consists of a set of attributes (a set of values for one attribute type is referred to as a domain). Each occurrence of the entity within the database can be thought of as a tuple within the relation. These two-dimensional tables have the following properties:

- (1) Each entry in a table-column represents one data item (attribute). There are no repeating groups.
- (2) They are column-homogeneous (in any column, all items are of the same kind).
- (3) Each column is assigned a distinct name.
- (4) All rows are distinct. Duplicate rows are not allowed.

Figure 6-18 is an illustration of a relational implementation of the conceptual data model.

The mapping from the Conceptual Schema to a relational database is very straightforward where:

- Nonspecific relationships have been resolved.
- Keys have been migrated.
- No role names are used.

In mapping to a relational DBMS:

- Each entity class becomes a table (relation).
- Each attribute of an entity becomes a data item (column or field) in the corresponding table.
- The key of each entity becomes the primary key in the corresponding table.
- A relationship is represented by foreign keys in the dependent entity.

In Figure 6-18, each entity becomes a relational table

UM 620141001
1 November 1985

where:

E1 maps to Supplier Table
E2 maps to Order Table
E3 maps to Line-item Table
E4 maps to Quotation Table
E5 maps to Part Table

The relationships from the CS in Figure 6-18 are represented by attributes within tables and map as follows:

RT1 maps to Supplier # in the QUOTATION Table
RT2 maps to Supplier # in the ORDER Table
RT3 maps to Supplier # in the LINE-ITEM Table
RT4 maps to Part # in the QUOTATION Table
RT5 maps to Part # in the LINE-ITEM Table

SUPPLIER

SUPPLIER-NO	SUPPLIER-NAME	BILL-TO-ADDRESS	SHIP-TO-ADDRESS
KEY			

PART

PART-NO	PART-NAME	PART-DESCRIPTION	QTY-ON-HAND
KEY			

QUOTATION

SUPPLIER-NO	PART-NO	QUOTE-PRICE	LEAD-TIME
KEY			

ORDER

ORDER-NO	SUPPLIER-NO	ORDER-DATE	DELIVERY-DATE
KEY			

LINE-ITEM

ORDER-NO	LINE-ITEM-NO	PART-NO	QUANTITY	PRICE
KEY				

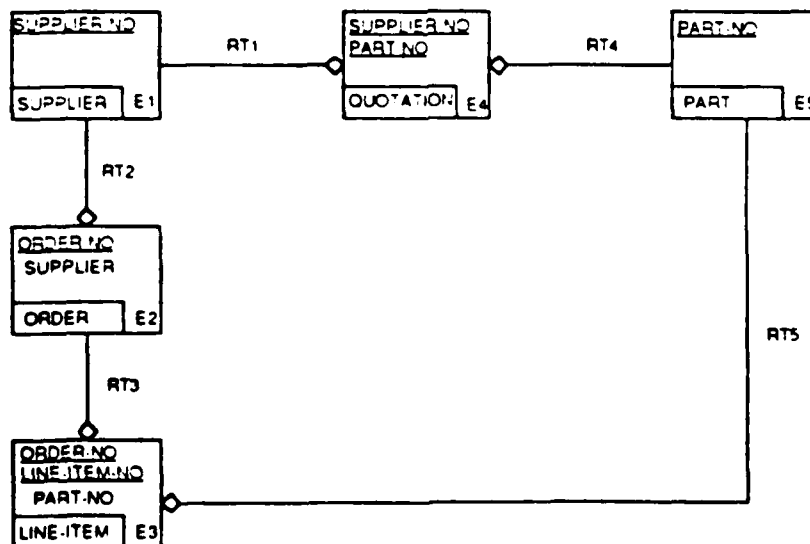


Figure 6-18. Relational Implementation of the Conceptual Model

6.4.2 Building a Relational Table IS and CS-IS Mapping

Objectives:

- Load the description of a relational database table into the following tables in the CDM database:

Database Table
Record Type Table
Database Area Table
Database Area Assignment Table
Data Field Table
Component Data Field Table

- Build a model of the mapping between the relational table and the conceptual schema.
- Load the descriptions of the CS-IS mapping into the following tables in the CDM database:

Entity Class/Record Type Mapping Table
Attribute Use Class/Data Field Mapping Table

Refer to Section 6.1.3 for details on how to fill out the CS-IS mapping forms.

If the CDM tables are to be loaded with the NDDL commands, skip to Section 6.4.3.

Tasks:

1. The CDM Administrator loads descriptions from the database DDL statements.

Create one entry in the Database Table for the relational table.

Create one entry in the Database Area Table for the portion of the storage pool the table occupies.
(DBSPACE Statements in SQL/DB2.)

Create an entry in the Record Type Table for the relational table record (row).

Create an entry in the Database Area Assignment Table for the table recrd.

Create one entry in the Data Field Table for each data field (column) in the table.

Create one entry in the Component Data Field Table for each data field that is part of another data field.

2. The CDM Administrator determines the mapping for each table.

Determine what sort of "real-world" thing the table represents. Each instance of a record type within a table contains data about a specific person, place, object, etc., that is significant to the enterprise. With a relational DBMS, all of the instances of the same type are about the same sort of thing and map directly to an entity.

Determine which entity class in the conceptual schema represents the same sort of thing as the table. This primarily involves finding the entity whose definition corresponds to the intent of the table.

Fill out a line on a Record Type/Entity Class Mapping Form for the entity class to which the table maps.

3. The CDM Administrator determines the mapping for each column.

Determine what sort of data about real-world things that the data field contains. There should always be a one-for-one mapping between the attributes of an entity and the data fields of its corresponding table. A table, however, could contain data fields that exist only to maintain a relationship. These data fields will not have a mapping.

A few data fields may not contain data about real-world things; they exist for technical reasons only. Examples include record codes and record activity dates. Such data fields do not map to any attribute use classes and can be ignored.

Determine which attribute use classes in the conceptual schema represent the same sort of data as the data field. This involves finding the attribute use class whose definition or migration path

corresponds to the intent of the data field.

Fill out a line on a Data Field/Attribute Use Class Mapping Form for each attribute use class to which the data field maps.

4. The CDM Administrator loads descriptions from the Record Type/Entity Class Mapping Forms.

Create one entry in the Entity Class/Record Type Mapping Table from each line on each form.

5. The CDM Administrator loads descriptions from the Data Field/Attribute Use Class Mapping Forms.

Create one entry in the Attribute Use Class/Data Field Mapping Table from each line on each form.

6.4.3 Building a Relational Table IS and CS-IS Mapping with NDDL

A summary of the NDDL commands required to load and maintain on IS and CS-IS mappings is contained in Section 6.1.4.

Tasks:

1. The CDM Administrator loads descriptions from the database DDL statements.

Use the NDDL DEFINE DATABASE command to+

- Define the database name.
- Define the DBMS.
- Describe the host.
- Record the password(s).

For each table in the database, use the NDDL DEFINE TABLE command to:

- Define the table name.
- Define the columns.

Note: The DEFINE TABLE command does not support repeating groups, component datafields, or redefined datafields.

2. The CDM administrator determines the primary mapping

for each new table.

See Task 2 of Section 6.4.2.

3. The CDM Administrator determines the primary mapping for each field.

See Task 3 of Section 6.4.2.

4. The CDM Administrator loads the descriptions from the Data Field/Attribute Use Class Mapping Forms.

For each new table, use the NDDL CREATE MAP command to:

- Map tag names (attribute classes) from a conceptual schema entity class to column names.
- Map attribute use classes to a column name.

6.4.4 Modifying a Relational Table IS and CS-IS Mapping

6.4.4.1 Modify a Mapped Table by Adding, Modifying, and/or Deleting Columns

Objective:

- Modify the description of a previously defined relational table within the tables in the CDM database:

Data Field Table
Component Data Field Table

- Build a model of the new CS-IS mapping between the relational table and the conceptual schema.
- Load the description of the new CS-IS mapping into the following table in the CDM database:

Attribute Use Class/Data Field Mapping Table

Refer to Section 6.1.3 for details on how to fill out the CS-IS mapping forms.

If the CDM tables are to be modified with the NDDL commands, skip to Section 6.4.5.

Tasks:

1. The CDM Administrator loads descriptions for the modified table from the database DDL statements.

Create one entry in the Data Field Table for each new data field added to the table.

Delete the entry for each data field deleted from the table. Modifications to previously defined data fields are made in the Data Field Table as appropriate.

Create one entry in the Component Data Field Table for each new data field that is part of another data field.

Delete the entry in the Component Data Field Table for each data field deleted that is part of another data field.

Create a new entry in the Record Key Table for each new data field or set of new data fields that is designated as the key of the table.

Delete the entry(s) in the Record Key Table for each data field deleted or set of data fields that is designated as part of the key of a table. Remember, a table must have a key, and it must be unique.

2. The CDM Administrator determines the mapping for each new column in the table.

NOTE: This task is to be omitted when modifying or deleting a previously defined data field.

Determine what sort of data about real-world things that the data field contains. See Task 3 of Section 6.5.2.

Determine which attribute use class in the conceptual schema represents the same sort of data as the data field. See Task 3 of Section 6.5.2.

Fill out a line on a Data Field/Attribute Use Class Mapping Form for the attribute use class to which the

new data field maps.

3. The CDM Administrator loads descriptions from the Data Field/Attribute Use Class Mapping Forms.

Create one entry in the Attribute Use Class/Data Field Mapping Table from each line on each form that references a new data field.

Delete each entry in the Attribute Use Class Data Field Mapping Table that references a deleted data field.

6.4.4.2 Delete a Previously Defined Relational Table by Modifying the CS-IS Mapping.

Objective:

- Delete the description of a previously defined relational table from the tables in the CDM database:

Record Type Table
Database Area Table
Database Area Assignment Table
Data Field Table
Component Data Field Table

- Load the description of the new CS-IS mapping into the following tables in the CDM database:

Entity Class/Record Type Mapping Table
Attribute Use Class/Data Field Mapping Table

Refer to Section 6.1.3 for details on how to fill out the CS-IS mapping forms.

Tasks:

1. The CDM Administrator deletes descriptions from the tables in the CDM database for the deleted relational table.

Delete the entry in the Record Type Table for the deleted table.

Delete the entry in the Database Area Assignment Table for the deleted table.

Delete the entry in the Data Field Table for each data field in the table to be deleted.

Delete the entry in the Component Data Field Table for each data field that is part of another data field for the deleted table.

2. The CDM Administrator deletes descriptions from the Record Type/Entity Class Mapping Forms.

Delete the entry in the Record Type Component Table from each line on each form that references a deleted table.

3. The CDM Administrator deletes descriptions from the Data Field/Attribute Use Class Mapping Forms.

Delete the entry in the Attribute Use Class/Data Field Mapping Table for each data field in a deleted table.

6.4.5 Modifying a Relational Table IS and CS-IS Mapping with NDDL

A summary of the NDDL commands required to load and maintain on IS and CS-IS mapping is contained in Section 6.1.4.

Tasks:

1. The CDM Administrator loads descriptions for the modified table from the database DDL statements.

Use the NDDL DROP TABLE command to delete the table that is to be modified.

Use the NDDL DEFINE TABLE command to define the columns that were added, changed, or deleted.

2. The CDM Administrator determines the mapping for each new column in the table.

See Task 2 of Section 6.4.4.

3. The CDM Administrator loads descriptions from the Data Field/Attribute Use Class Mapping Form.

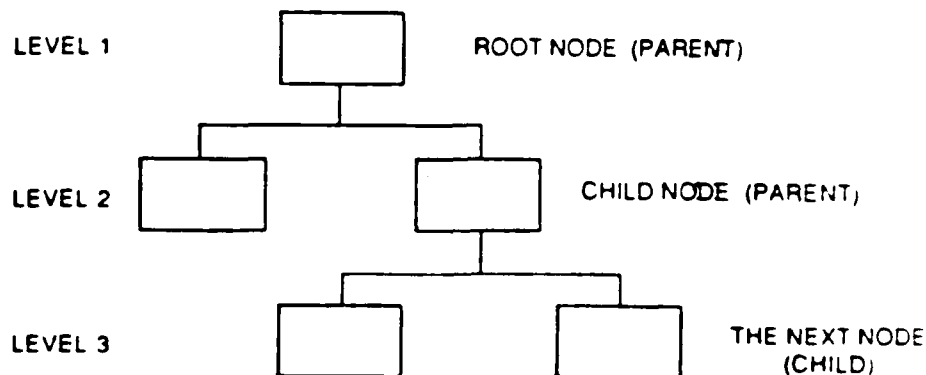
For each modified table, use the NDDL CREATE MAP command to:

- Recreate the mappings between tag names (attribute classes) from a conceptual schema entity class to column names.
- Recreate the mappings between attribute use classes and column names.

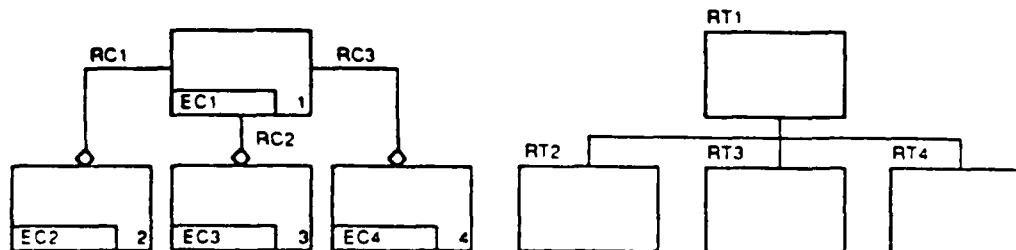
6.5 IMS Databases

6.5.1 IMS Specific Considerations

Whereas the basic construct of the CODASYL model is a set and complicated structures can be built from sets, the IMS model represents data in the form of a tree structure. A tree consists of different levels of entities referred to as nodes. A node can have many occurrences, that is, sets of data values for its data items. Each higher level implies dominance over the levels below it, thus creating a hierarchy. The highest level contains only one node called a root node. All nodes, with the exception of the root, must be connected at a level above it. The node at the higher level is called a parent node and "owns" all of the lower level nodes in the limb. The node at the lower level is called a child node. A child node must have one and only one parent node. A parent node can have none, one, or many nodes connected to them as children. There can be many occurrences of a specific child node under a single parent. A parent and its children at each level are considered a physical tree. A database may consist of many of these physical trees. Even though the set construct is not supported by the IMS model, a parent and all of its children are to be considered analogous to a set. The following example depicts the IMS hierarchical model.



A node is called an IMS segment. A logical record in the database consists of a root and all of its children. A database record can consist of a tree with up to 15 levels. In essence, many logical relations (relation classes) could be combined into one physical hierarchy. This is called a regular hierarchy and is defined via an IMS Database Definition (DBD).



The CS-IS mapping for a regular hierarchy involves the following:

- Each parent-child relationship within the hierarchy maps to a relation class.
- The parent in each relationship maps to the entity class that is independent in that relation class.
- The child in each relationship maps to the entity class that is dependent in that relation class.

In the example above:

RT1 maps to EC1
RT2 maps to EC2
RT3 maps to EC3
RT4 maps to EC4
RT1:RT2 maps to RC1
RT1:RT3 maps to RC2
RT1:RT4 maps to RC3

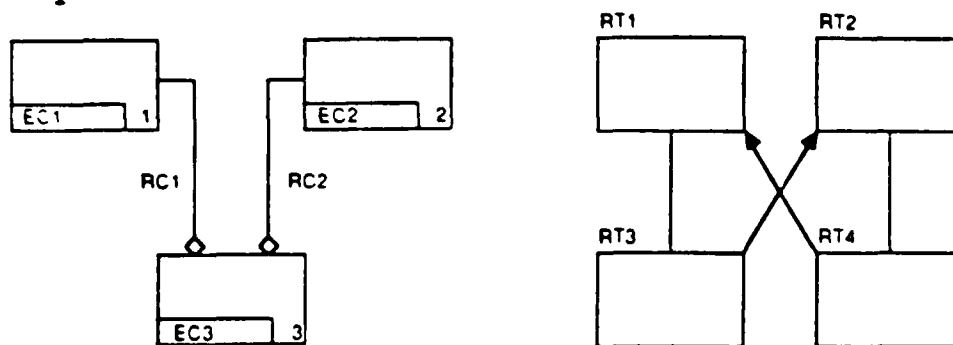
The previous diagram illustrates that each dependent segment has a parent segment and exists as one element in a child-parent relationship. These relationships can have both a physical and a logical form. The physical form of the parent-child relationship is a consequence of (1) the definition of a given data base and (2) the method by which the data elements are stored. The logical relationship is established solely by express definition and exists externally to any

physical organization constraints.

IMS has an additional relationship called a "twin." As with the parent-child relationship, two forms of twins exist: physical and logical twins. Physical twin segments are multiple occurrences of a common segment format. At the root segment level, the set of physical twins is the set of all root segment occurrences of a given database. At the dependent segment level, a set of physical twins is the set of physical child occurrences for a given segment format within a hierarchy. At the logical level, twins are multiple occurrences of a common segment format having a common logical parent. The physical and logical concepts give IMS the capability of storing network type relationships (sets) between entities. These network physical structures are viewed by users and programmers as one or more hierarchical views.

Every logical relationship involves the use of three segments; no more, no less. Two of these segment types (the physical parent and the logical parent) can exist in separate databases or they may exist in the same physical database. The third segment type (the logical child) is used to construct the logical linkage. This segment type is very special since it has two parents.

A nonspecific membership set type whose cardinality is many- to-many, is one in which (1) each member of entity class "1" is related to zero, one, or many members of entity class "2" and (2) each member of entity class "2" is related to zero, one, or many members of entity class "1." Such a relation class is refined, as shown in Figure 4-7, before it is incorporated into the conceptual schema.



The CS-IS mapping for a many-to-many membership set type involves the following.

- Each parent segment has a primary mapping to one entity class.
- The child segments have a primary mapping to a single entity class (RT4 may or may not exist physically on the database depending on the IMS options that were chosen).
- Two IMS DBDs are required and map to one-to-many relation classes.
- The entity classes to which the parents map are independent in their respective relation classes.
- The entity class to which the children map is dependent in that relation class.

In the example above:

RT1 maps to EC1
RT2 maps to EC2
RT3 maps to EC3
RT4 maps to EC3
RT1:RT3 maps to RC1
RT2:RT4 maps to RTC2

6.5.2 Building an IMS IS and CS-IS Mapping

Objectives:

- Load the description of an IMS database into the following tables in the CDM database:

Database Table
Record Type Table
Database Area Table
Data Field Table
Component Data Field Table
Data Field Redefinition Table
Repeating Data Field Occurance Counter Table
Record Set Table
Record Set Member Table
- Build a model of the mapping between the IMS database and the conceptual schema.
- Load the descriptions of the CS-IS mapping into the

following tables in the CDM database:

Entity Class/Record Type Mapping Table
Entity Class/Record Type Join Table
Relation Class/Set Type Mapping Table
Attribute Use Class/Data Field Mapping Table

Refer to Section 6.1.3 for details on how to fill out the CS-IS mapping forms.

Skip to Section 6.5.3 if the IMS description is to be loaded with the NDDL.

Tasks:

1. The CDM Administrator loads descriptions from the database DDL (DBD) statements.

Create one entry in the Database Table for the database.

Create one entry in the Record Type Table for each segment type in the database.

Create one entry in the Data Field Table for each data element in each segment type in the database.

Create one entry in the Component Data Field Table for each data element that is part of another data field.

Create one entry in the Data Field Redefinition Table for each data element that redefines another data field.

Create one entry in the Repeating Data Field Occurrence Counter Table for each data element that occurs more than once in a segment type.

Create one entry in the Set Type Table for each parent: child relationship in the database.

Create one entry in the Set Type Member Table for each segment type that is a child (member of a set type). If a segment type is a member of more than one set type (this can only occur if it is a logical child), create a table entry for each. If a parent

segment type has more than one child segment type, create a table entry for each.

2. The CDM Administrator determines the mapping for each segment type.

Usually, it is easier to map an IMS database by starting with the root segment and working down each limb of the tree. A child segment type should not be mapped until its parent segment type has been mapped.

Determine what sort of "real-world" thing the segment type represents. Each instance of a segment type contains data about a specific person, place, object, etc., that is significant to the enterprise. With IMS, all of the instances of the same type are about the same sort of thing. Usually, all of the instances of the same type are about the same sort of thing.

A few segment types do not represent real-world things; they exist to provide database pointers between logically related segments. Such segment types do not map to any entity classes and can be ignored.

Determine which entity class in the conceptual schema represents the same sort of thing as the segment type. This primarily involves finding the entity class whose definition corresponds to the intent of the segment type. Comparing the key classes, attribute use classes, and relation classes of the entity classes to the keys, data elements, and parent-child relationships of the segment types can be helpful also. If the segment type represents several sorts of things, it will map to several entity classes, one for each sort of thing; see Section 6.1.2.4 regarding relational unions. If none of the entity classes represent what the segment type does, either the segment type exists only to provide a logical relationship or the conceptual schema must be expanded. (See Section 4.3.)

Fill out a line on a Record Type/Entity Class Mapping Form for each entity class to which the segment type maps.

3. The CDM Administrator determines the mapping for each data element.

Determine what sort of data about real-world things that the data element contains. If the segment type that contains the data element represents more than one sort of thing, i.e., if it has more than one mapping, the data field may contain several sorts of data. All of these must be identified.

A few data elements do not contain data about real-world things; they exist for technical reasons only. Examples include segment codes and segment activity dates. Such data elements do not map to any attribute use classes and can be ignored.

Determine which attribute use classes in the conceptual schema represent the same sort of data as the data element. This involves finding the attribute use class whose definition or migration path corresponds to the intent of the data element. The first place to look is the entity class to which the segment type maps. If the segment type maps to more than one entity class, the data element may map to an attribute use class in each. The value in the data element in each instance of the segment type must be the same as the one in the attribute use class in the corresponding instance of the entity class. If two or more inherited attribute use classes that come from the same owned attribute use class have identical values in every entity instance, the data element may map to some or all of them.

If none of the attribute use classes in the mapped to entity class(es) correspond to the data element, the next places to look are the entity classes that are related to the primary entity class(es). Again, the value in each segment instance must be the same as the value in the corresponding entity instance. If the attribute use class is not in any of these entity classes, the search must be widened to include the entity classes that are related to them. This continues until the proper attribute use class is found or until it is determined that a new attribute class must be added to the conceptual schema; see Section 4.3.

Fill out a line on a Data Field/Attribute Use Class

Mapping Form for each attribute use class to which the data element maps.

4. The CDM Administrator determines any joins that are needed for each segment type.

Determine whether any of the data elements in the segment type map to attribute use classes that are not in the entity class(es) to which the segment type maps. This can be done by comparing the entity class numbers that are entered on the Data Field/Attribute Use Class Mapping Forms for the segment type to those that are entered on the Record Type/Entity Class Mapping Form for the segment type. If an entity class number is on the first form but not on the second, that entity class must be joined with the one to which the segment type maps.

Determine whether any other entity classes are needed to complete the join structure(s). The entity classes that must be joined to form the segment type must form one or more join structures as described in Section 6.1.2.3. If the join structures are not contiguous, one or more additional joins may be needed. For example, if the segment type in Figure 6-17 maps to EC4 and involves joins with EC1 and EC3, it must also have a join with EC2. Without it, EC1 cannot be joined to the EC3-EC4 join result. The join must involve EC2 even though none of its attribute use classes map to data elements in the segment type.

Prepare Record Type Join Structure Diagrams for the segment types that involve joins.

5. The CDM Administrator determines the mapping for each parent-child relationship.

Determine what sort of relationship between "real-world" things the set type represents. If the set type has more than one child segment type, each must be considered separately. If either the parent or the child segment type has no mapping to an entity class, the set type will have no mapping to a relation class, so it can be ignored.

Determine which relation class in the conceptual

schema represents the same sort of relationship as the set type. Usually, this is the relation class whose independent entity class maps to the parent segment type and whose dependent entity class maps to the child segment type.

Fill out a line on a Set Type/ Relation Class Mapping Form for the relation class to which the set type maps.

6. The CDM Administrator loads descriptions from the Record Type/Entity Class Mapping Forms.

Create one entry in the Entity Class/Record Type Mapping Table from each line on each form.

7. The CDM Administrator loads descriptions from the Record Type Join Structure Diagrams.

Create one entry in the Entity Class/Record Type Join Table for each relation class in a diagram.

8. The CDM Administrator loads descriptions from the Data Field/Attribute Use Class Mapping Forms.

Create one entry in the Attribute Use Class/Data Field Mapping Table from each line on each form.

9. The CDM Administrator loads descriptions from the Set Type/Relation Class Mapping Forms.

Create one entry in the Relation Class/Set Type Mapping Table from each line on each form.

6.5.3 Building an IMS IS and CS-IS Mapping with NDDL

Tasks:

1. The CDM Administrator loads descriptions from the database DDL (DBD) statements.

Use the NDDL DEFINE DATABASE command to:

- Define the DBMS
- Define the database name
- Describe the host
- Record password(s)

- Define the PBS name
- Define database areas (dataset)
- Supply key feedback length

Use the DEFINE SEGMENT command to:

- Attach the segment(s) to the database
- Define the key fields and state whether or not it must be unique
- Define what area a segment is to physically reside in
- Define the elements within the segment

Use the DEFINE PATH command for each new parent/child relationship.

2. The CDM Administrator determines the mapping for each segment type.

See Task 2 of Section 6.5.2.

3. The CDM Administrator determines the mapping for each data field (element).

See Task 3 of Section 6.5.2.

4. The CDM Administrator determines any joins that are needed for each segment type.

See Task 4 of Section 6.5.2.

5. The CDM Administrator determines the mapping for each parent/child relationship.

See Task 5 of Section 6.5.2

6. The CDM Administrator loads the mappings.

Use the NDDL CREATE MAP command for each segment.

6.5.4 Modifying an IMS IS and CS-IS Mapping

6.5.4.1 Add One or More New Segment Type(s)

Objective:

- Load the description of a new segment type(s) within

an IMS database into the following tables in the CDM database:

Record Type Table
Database Area Table
Data Field Table
Component Data Field Table
Data Field Redefinition Table
Repeating Data Field Occurance Counter Table
Record Set Table
Record Set Member Table

- Build a model of the new mapping between the IMS database and the conceptual schema.
- Load the description of the new CS-IS mapping into the following tables in the CDM database:

Entity Class/Record Type Mapping Table
Entity Class/Record Type Join Table
Relation Class/Set Type Mapping Table
Attribute Use Class/Data Field Mapping Table

Refer to Section 6.1.3 for details on how to fill out the CS-IS mapping forms.

Skip to Section 6.5.5 if the IMS description is to be loaded with NDDL.

Tasks:

1. The CDM Administrator loads descriptions for the new segment type from the database DDL (DBD) statements.

Create one entry in the Record Type Table for each new segment type added to the database.

Create one entry in the Data Field Table for each data field in each new segment type added to the database.

Create one entry in the Component Data Field Table for each data element that is part of another data field.

Create one entry in the Data Field Redefinition Table for each data element that redefines another data

field.

Create one entry in the Repeating Data Field Occurrence Counter Table for each data field that occurs more than once in a segment type.

Create one entry in the Record Set Table for each parent-child relationship ("set type") the new segment type participates in as a parent.

Create one entry in the Record Set Member Table for each new segment type that is a child (member of a set type). If a segment type is a member of more than one set type (this can only occur if it is a logical child), create a table entry for each. If adding a parent segment with multiple children, create a table entry for each child.

2. The CDM Administrator determines the mapping for each new segment type.

See Task 2 of Section 6.5.2.

Fill out a line on a Record Type/Entity Class Mapping Form for each entity class to which the segment type maps.

3. The CDM Administrator determines the mapping for each element in the new segment type.

See Task 3 of Section 6.5.2.

Fill out a line on a Data Field/Attribute Use Class Mapping Form for each attribute use class to which the data field maps.

4. The CDM Administrator determines any joins that are needed for each new segment type.

See Task 4 of Section 6.5.2.

Prepare Record Type Join Structure Diagrams for each new segment type that has any secondary mappings.

5. The CDM Administrator determines the mapping for each parent-child relationship.

See Task 5 of Section 6.5.2.

Fill out a line on a Set Type/Relation Class Mapping Form for the relation class to which the set type maps.

6. The CDM Administrator loads descriptions from the Record Type/Entity Class Mapping Form.

Create one entry in the Entity Class/Record Type Mapping Table from each line on each form.

7. The CDM Administrator loads descriptions from the Record Type Join Structure Diagrams.

Create one entry in the Entity Class/Record Type Join Table for each relation class in a diagram.

8. The CDM Administrator loads descriptions from the Data Field/Attribute Use Class Mapping Forms.

Create one entry in the Attribute Use Class/Data Field Mapping Table from each line on each form.

9. The CDM Administrator loads descriptions from the Set Type/Relation Class Mapping Forms.

Create one entry in the Relation Class/Set Type Mapping Table from each line on each form.

6.5.4.2 Modify an Existing Segment Type Database by Adding Modifying, and/or Deleting Data Fields

Objective:

- Modify the description of a previously defined segment type within the tables in the CDM database:
 - Data Field Table
 - Data Field Redefinition Table
 - Component Data Field Table
 - Repeating Data Field Occurance Counter Table
- Build a model of the new CS-IS mapping between the IMS database and the conceptual schema.
- Load the description of the new CS-IS mapping

into the following table in the CDM database:

Attribute Use Class/Data Field Mapping Table

Refer to Section 6.1.3 for details on how to fill out the CS-IS mapping forms.

Skip to Section 6.5.5 if the IMS description is to be modified with NDDL.

Tasks:

1. The CDM Administrator loads descriptions for the modified segment type from the database DDL statements.

Create one entry in the Data Field Table for each new data element added to the database.

Delete the entry for each data element deleted from the database. Modifications to previously defined data elements are made in the Data Field Table as appropriate.

Create one entry in the Component Data Field Table for each new data element that is part of another data field.

Delete the entry in the Component Data Field Table for each data element deleted that is part of another data element.

Create one entry in the Data Field Redefinition Table for each new data element that redefines another data element.

Delete the entry in the Data Field Redefinition Table for each deleted data element that redefines another data element. Care must be taken to ensure the deletion of the redefined field also.

Create one entry in the Repeating Data Field Occurrence Counter Table for each new data element that occurs more than once in a segment type.

Delete the entry in the Repeating Data Occurrence Counter Table Field for each deleted data element that occurs more than once in a segment type.

2. The CDM Administrator determines the mapping for each new data element in the segment type.

NOTE: This task is to be omitted when modifying or deleting a previously defined data element.

See Task 2 of Section 6.5.2.

Fill out a line on a Data Field/Attribute Use Class Mapping Form for each attribute use class to which the new data field maps.

3. The CDM Administrator loads descriptions from the Data Field/Attribute Use Class Mapping Forms.

Create one entry in the Attribute Use Class/Data Field Mapping Table from each line on each form that references a new data field.

Delete each entry in the Attribute Use Class/Data Field Mapping Table that references a deleted data field.

6.5.4.3 Delete a Previously Defined Segment Type by Modifying the IMS and CS-IS Mapping

Objective:

- Delete the description of a previously defined segment type from the tables in the CDM database:
 - Record Type Table
 - Database Area Table
 - Database Area Assignment Table
 - Data Field Table
 - Component Data Field Table
 - Data Field Redefinition Table
 - Repeating Data Field Occurance Counter Table
 - Record Set Table
 - Record Set Member Table
- Build a model of the new mapping between the IMS database and the conceptual schema.
- Load the description of the new CS-IS mapping into

the following tables in the CDM database:
Entity Class/Record Type Mapping Table
Entity Class/Record Type Join Table
Relation Class/Set Type Mapping Table
Attribute Use Class/Data Field Mapping Table

Refer to Section 6.1.3 for details on how to fill out the CS-IS mapping forms.

Skip to Section 6.5.5 if the IMS description is to be modified with the NDDL.

Tasks:

1. The CDM Administrator deletes descriptions from the tables in the CDM database for the deleted segment types.

Delete the entry in the Record Type Table for each segment type deleted from the database.

Delete the entry in the Database Area Assignment Table for each deleted segment type that is assigned to a database area.

Delete the entry in the Data Field Table for each data field in each segment type deleted from the database.

Delete the entry in the Component Data Field Table for each data field that is part of another data field for each segment type deleted from the database.

Delete the entry in the Redefined Data Field Table for each data field that redefines another data field for each segment type deleted from the database.

Delete the entry in the Repeating Data Field Table for each data field that occurs more than once in a deleted segment type.

Delete the entry in the Set Type Table for each deleted segment type that is a parent. If a deleted segment type is the parent of more than one child, delete the table entry for each.

Delete the entry in the Set Type Member Table for each deleted segment type that is a child.

2. The CDM Administrator deletes descriptions from the Record Type/Entity Class Mapping Forms. Delete the entry in the Entity Class/Record Type Mapping Table from each line on each form that references a deleted segment type.
3. The CDM Administrator deletes descriptions from the Record Type Join Structure Diagrams.

Delete the entry in the Entity Class/Record Type Join Table for each relation class in a diagram that references a deleted segment type.

4. The CDM Administrator deletes descriptions from the Data Field/Attribute Use Class Mapping Forms.

Delete the entry in the Attribute Class/Data Field Mapping Table for each data field in a deleted segment type.

5. The CDM Administrator deletes descriptions from the Set Type/Relation Class Mapping Forms.

Delete all entries in the Relation Class/Set Type Mapping Table for each segment type deleted.

6.5.4.4 Delete an IMS Database from the CS-IS Mapping

Objective:

- Delete the description of a previously mapped IMS database from the following tables in the CDM database:

Database Table
Record Type Table
Data Field Table
Component Data Field Table
Data Field Redefinition Table
Repeating Data Field Occurance Counter Table
Record Set Table
Record Set Member Table

- Delete the description of the CS-IS mapping from

the following tables in the CDM database:

Entity Class/Record Type Mapping Table
Entity Class/Record Type Join Table
Relation Class/Set Type Mapping Table
Attribute Use Class/Data Field Mapping Table

Refer to Section 6.1.3 for details on how to fill out the CS-IS mapping forms.

Skip to Section 6.5.5 if the database is to be deleted with NDDL.

Tasks:

1. The CDM Administrator deletes descriptions from the CDM database table.

Delete the entry in the Record Type Table for each segment in the database.

Delete the entry in the Data Field Table for each data element in each segment type in the database.

Delete the entry in the Component Data Field Table for each data element that is part of another data field.

Delete the entry in the Data Field Redefinition Table for each data element that redefines another data field.

Delete the entry in the Repeating Data Field Occurrence Counter Table for each data element that occurs more than once in a segment type.

Delete the entry in the Record Set Table for each set type in which a segment type participates.

Delete the entry in the Record Set Member Table for each segment type that is a member of a set type. If a segment type is a member of more than one set type, delete the table entry for each.

2. The CDM Administrator deletes descriptions from the Record Type/Entity Class Mapping Forms.

Delete the entry(s) in the Entity Class/Record Type

Mapping Table for each segment type in the deleted database.

3. The CDM Administrator deletes descriptions from the Record Type Join Structure Diagrams. Delete the entry in the Entity Class/Record Type Join Table for each relation class that is referenced by a deleted segment type.

4. The CDM Administrator deletes descriptions from the Data Field/Attribute Use Class Mapping Forms.

Delete the entry in the Attribute Use Class/Data Field Mapping Table for all data elements to be deleted.

5. The CDM Administrator deletes descriptions from the Set Type/Relation Class Mapping Forms.

Delete the entry in the Relation Class/Set Type Mapping Table for all set types within the database to be deleted.

6.5.5 Modifying an IMS IS and CS-IS Mapping with the NDDL

6.5.5.1 Add One or More Segment Types

Tasks:

1. The CDM Administrator loads descriptions for the new segment type(s) from the database DDL (DBD) statements.

See Task 1 of Section 6.5.4.

Use the NDDL DEFINE SEGMENT to describe each new segment type.

2. The CDM Administrator determines the mapping for each new segment type.

See Task 2 of Section 6.5.4.

3. The CDM Administrator determines the mapping for each element in the new segment type.

See Task 3 of Section 6.5.4.

4. The CDM Administrator determines any joins that are needed for each new segment type.

See Task 4 of Section 6.5.4.

5. The CDM Administrator determines the mapping for each parent-child relationship.

See Task 5 of Section 6.5.4.

Use the DEFINE PATH command for each new parent/child relationship.

Use the CREATE MAP command for each new segment to be described.

6.5.5.2 Modify and Existing IMS Segment Type by Adding, Modifying or Deleting Data Fields

Tasks:

1. The CDM Administrator loads descriptions for the modified segment type from the database DDL (DBD) statements.

Each segment to be modified must be deleted and redefined. Use the NDDL DROP SEGMENT command. This will also delete all mappings. These must be added later.

Use the DEFINE SEGMENT command for each segment to be modified.

Use the DEFINE PATH command to add the path again.

2. The CDM Administrator determines the mapping for each new data element in the segment type.

Note: This task is to be omitted when modifying or deleting a previously defined data element.

See Task 2 of Section 6.5.4.

3. The CDM Administrator loads descriptions from the Data Field/Attribute Use Class Forms.

Use the CREATE MAP command to redefine each modified

segment. Don't forget to include all mappings.

6.5.5.3 Delete a Previously Defined Segment Type by Modifying the IMS and CS-IS Mapping

Tasks:

1. Delete the segment type with the NDDL DROP SEGMENT command .

6.5.5.4 Delete an IMS Database from the CS-IS Mapping

Tasks:

1. The CDM Administrator deletes descriptions from the CDM database tables.

Delete each segment in the database with the DROP SEGMENT command.

Delete the database with the DROP DATABASE command.

6.6 VSAM Files

6.6.1 VSAM-Specific Considerations

The Virtual Storage Access Method (VSAM) is a component of the IBM operating system's data management services. VSAM supports both direct and sequential processing. VSAM data sets cannot be accessed by any other access method.

VSAM support consists of the following:

- Three data sets organizations: Entry-Sequenced Data Sets (ESDS), Key-Sequenced Data Sets (KSDS), and Relative-Record Data Sets (RRDS). They are supported on DASD (Direct Access Storage Devices) only.
 - A VSAM ESDS is a sequential data set (similar to a SAM data set).
 - A VSAM KSDS is a sequential data set with an index (similar to an ISAM data set).
 - A VSAM RRDS is a data set with preformatted slots for fixed length records to be accessed by a record number (similar to a DAM data set).

NOTE: As VSAM RRDSs are rarely used, they are excluded from this document.

As the mappings for an ESDS and KSDS are identical, the following discussions will not make any differentiations.

The mapping from the Conceptual Schema to a VSAM file is very straightforward where:

- Nonspecific relationships have been resolved.
- Keys have been migrated.
- No role names are used.

In mapping to a VSAM file:

- Each entity class becomes a record.
- Each attribute of an entity becomes a data item in the corresponding VSAM record.
- The key of each entity becomes the primary key in the corresponding record.

If relationships between VSAM files are implied (foreign keys have been migrated), please refer to Section 6.4 for mapping instructions.

6.6.2 Building a VSAM IS and CS-IS Mapping

Objectives:

- Load the description of a VSAM file into the following tables in the CDM database:
 - Database Table
 - Record Type Table
 - Data Field Table
 - Component Data Field Table
- Build a model of the mapping between the VSAM file and the conceptual schema.
- Load the descriptions of the CS-IS mapping into the following tables in the CDM database:

Entity Class/Record Type Mapping Table
Attribute Use Class/Data Field Mapping Table

Please refer to Section 6.1.3 for details on how to fill out the CS-IS mapping forms. Note that, NDDL does not support VSAM data sets.

Tasks:

1. The CDM Administrator loads descriptions from the VSAM file design.

Create one entry in the Database Table for the VSAM file.

Create an entry in the Record Type Table for the VSAM record.

Create one entry in the Data Field Table for each data field in the record.

Create one entry in the Component Data Field Table for each data field that is part of another data field.

2. The CDM Administrator determines the mapping for each record type.

Determine what sort of "real-world" thing the record represents. Each instance of a record type contains data about a specific person, place, object, etc., that is significant to the enterprise. With a VSAM file, all of the instances of the same type are about the same sort of thing and map directly to an entity.

Determine which entity class in the conceptual schema represents the same sort of thing as the VSAM record. This primarily involves finding the entity whose definition corresponds to the intent of the record.

Fill out a line on a Record Type/Entity Class Mapping Form for the entity class to which the record maps.

3. The CDM Administrator determines the mapping for each data field.

Determine what sort of data about real-world things

that the data field contains. There should always be a one-for-one mapping between the attributes of an entity and the data fields of its corresponding record.

A few data fields might not contain data about real-world things; they exist for technical reasons only. Examples include record codes and record activity dates. Such data fields do not map to any attribute use classes and can be ignored.

Determine which attribute use classes in the conceptual schema represent the same sort of data as the data field. This involves finding the attribute use class whose definition or migration path corresponds to the intent of the data field.

Fill out a line on a Data Field/Attribute Use Class Mapping Form for each attribute use class to which the data field maps.

4. The CDM Administrator loads descriptions from the Record Type/Entity Class Mapping Forms.

Create one entry in the Entity Class/Record Type Mapping Table from each line on each form.

5. The CDM Administrator loads descriptions from the Data Field/Attribute Use Class Mapping Forms.

Create one entry in the Attribute Use Class/Data Field Mapping Table from each line on each form.

6.6.3 Modifying a VSAM IS and CS-IS Mapping

6.6.3.1 Modify a Mapped VSAM Record by Adding, Modifying, and/or Deleting Data Fields

Objectives:

- Modify the description of a previously defined relational table within the tables in the CDM database:

Data Field Table
Component Data Field Table

- Build a model of the new CS-IS mapping between the VSAM record and the conceptual schema.
- Load the description of the new CS-IS mapping into the following table in the CDM database:

Attribute Use Class/Data Field Mapping Table

Please refer to Section 6.1.3 for details on how to fill out the CS-IS mapping forms. Note that, NDDL does not support VSAM data sets.

Tasks:

1. The CDM Administrator loads descriptions for the modified record type.

Create one entry in the Data Field Table for each new data field added to the record.

Delete the entry for each data field deleted from the record. Modifications to previously defined data fields are made in the Data Field Table as appropriate.

Create one entry in the Component Data Field Table for each new data field that is part of another data field.

Delete the entry in the Component Data Field Table for each data field deleted that is part of another data field.

2. The CDM Administrator determines the mapping for each new data field in the record type.

NOTE: This task is to be omitted when modifying or deleting a previously defined data field.

Determine what sort of data about real-world things that the data field contains.

Determine which attribute use class in the conceptual schema represents the same sort of data as the data field.

Fill out a line on a Data Field/Attribute Use Class Mapping Form for the attribute use class to which the new data field maps.

3. The CDM Administrator loads descriptions from the Data Field/Attribute Use Class Mapping Forms.

Create one entry in the Attribute Use Class/Data Field Mapping Table from each line on each form that references a new data field.

Delete each entry in the Attribute Use Class Data Field Mapping Table that references a deleted data field.

6.6.3.2 Delete a Previously Defined VSAM Record by Modifying the CS-IS Mapping

Objective:

- Delete the description of a previously defined VSAM record from the tables in the CDM database:

Record Type Table
Data Field Table
Component Data Field Table

- Load the description of the new CS-IS mapping into the following tables in the CDM database:

Entity Class/Record Type Mapping Table
Attribute Use Class/Data Field Mapping Table

Please refer to Section 6.1.3 for details on how to fill out the CS-IS mapping forms.

Tasks:

1. The CDM Administrator deletes descriptions from the tables in the CDM database for the deleted record type.

Delete the entry in the Record Type Table for the record to be deleted.

Delete the entry in the Data Field Table for each data field in the deleted record.

Delete the entry in the Component Data Field Table for each data field that is part of another data field.

2. The CDM Administrator deletes descriptions from the Record Type/Entity Class Mapping Forms.

Delete the entry in the Record Type Component Table from each line on each form that references a deleted record.

3. The CDM Administrator deletes descriptions from the Data Field/Attribute Use Class Mapping Forms.

Delete the entry in the Attribute Use Class/Data Field Mapping Table for each data field in a deleted record.

6.7 Sequential Files (Flat Files)

6.7.1 Sequential-Specific Considerations

The mapping from the Conceptual Schema to a sequential file is very straightforward where:

- Nonspecific relationships have been resolved.
- Keys have been migrated.
- No role names are used.

In mapping to a sequential file:

- Each entity class becomes a record.
- Each attribute of an entity becomes a data item (column or field) in the corresponding record.
- The key of each entity becomes the primary key in the corresponding record.

If relationships between sequential files are implied (foreign keys have been migrated), please refer to Section 6.4 for mapping instructions.

6.7.2 Building a Sequential File IS and CS-IS Mapping

Objectives:

- Load the description of a sequential file into the following tables in the CDM database:

Database Table
Record Type Table
Data Field Table
Component Data Field Table

- Build a model of the mapping between the sequential file and the conceptual schema.
- Load the descriptions of the CS-IS mapping into the following tables in the CDM database:

Entity Class/Record Type Mapping Table
Attribute Use Class/Data Field Mapping Table

Please refer to Section 6.1.3 for details on how to fill out the CS-IS mapping forms. Note that, NDDL does not support sequential files.

Tasks:

1. The CDM Administrator loads descriptions from the sequential file design.

Create one entry in the Database Table for the sequential file.

Create an entry in the Record Type Table for the sequential record.

Create one entry in the Data Field Table for each data field in the record.

Create one entry in the Component Data Field Table for each data field that is part of another data field.

2. The CDM Administrator determines the mapping for each record type.

Determine what sort of "real-world" thing the record represents. Each instance of a record contains data about a specific person, place, object, etc., that is significant to the enterprise.

Determine which entity class in the conceptual schema represents the same sort of thing as the record. This

primarily involves finding the entity whose definition corresponds to the intent of the record.

Fill out a line on a Record Type/Entity Class Mapping Form for the entity class to which the record maps.

3. The CDM Administrator determines the mapping for each data field.

Determine what sort of data about real-world things that the data field contains. There should always be a one-for-one mapping between the attributes of an entity and the data fields of its corresponding record.

A few data fields may not contain data about real-world things; they exist for technical reasons only. Examples include record codes and record activity dates. Such data fields do not map to any attribute use classes and can be ignored.

Determine which attribute use classes in the conceptual schema represent the same sort of data as the data field. This involves finding the attribute use class whose definition or migration path corresponds to the intent of the data field.

Fill out a line on a Data Field/Attribute Use Class Mapping Form for each attribute use class to which the data field maps.

4. The CDM Administrator loads descriptions from the Record Type/Entity Class Mapping Forms.

Create one entry in the Entity Class/Record Type Mapping Table from each line on each form.

5. The CDM Administrator loads descriptions from the Data Field/Attribute Use Class Mapping Forms.

Create one entry in the Attribute Use Class/Data Field Mapping Table from each line on each form.

6.7.3 Modifying a Sequential File IS and CS-IS Mapping

6.7.3.1 Modify a Mapped Record by Adding, Modifying, and/or Deleting Data Fields

Objectives:

- Modify the description of a previously defined relational table within the tables in the CDM database:

Data Field Table
Component Data Field Table

- Build a model of the new CS-IS mapping between the sequential record and the conceptual schema.
- Load the description of the new CS-IS mapping into the following table in the CDM database:

Attribute Use Class/Data Field Mapping Table

Please refer to Section 6.1.3 for details on how to fill out the CS-IS mapping forms. Note that, NDDL does not support sequential fields.

Tasks:

1. The CDM Administrator loads descriptions for the modified record type.

Create one entry in the Data Field Table for each new data field added to the record.

Delete the entry for each data field deleted from the record. Modifications to previously defined data fields are made in the Data Field Table as appropriate.

Create one entry in the Component Data Field Table for each new data field that is part of another data field.

Delete the entry in the Component Data Field Table for each data field deleted that is part of another data field.

2. The CDM Administrator determines the mapping for each new data field in the record type.

NOTE: This task is to be omitted when modifying or deleting a previously defined data field.
Determine what sort of data about real-world things that the data field contains.

Determine which attribute use class in the conceptual schema represents the same sort of data as the data field.

Fill out a line on a Data Field/ Attribute Use Class Mapping Form for the attribute use class to which the new data field maps.

3. The CDM Administrator loads descriptions from the Data Field/Attribute Use Class Mapping Forms.

Create one entry in the Attribute Use Class/Data Field Mapping Table from each line on each form that references a new data field.

Delete each entry in the Attribute Use Class Data Field Mapping Table that references a deleted data field.

6.7.3.2 Delete a Previously Defined Sequential File by Modifying the IMS and CS-IS Mapping

Objectives:

- Delete the description of a previously defined sequential file from the tables in the CDM database:

Record Type Table
Data Field Table
Component Data Field Table

- Load the description of the new CS-IS mapping into the following tables in the CDM database:

Entity Class/Record Type Mapping Table
Attribute Use Class/Data Field Mapping Table

Please refer to Section 6.1.3 for details on how to fill out the CS-IS mapping forms.

Tasks:

1. The CDM Administrator deletes descriptions from the tables in the CDM database for the deleted record type.

Delete the entry in the Record Type Table for the sequential record to be deleted.

Delete the entry in the Data Field Table for each data field in the deleted record.

Delete the entry in the Component Data Field Table for each data field that is part of another data field.

2. The CDM Administrator deletes descriptions from the Record Type/Entity Class Mapping Forms.

Delete the entry in the Record Type Component Table from each line on each form that references a deleted record.

3. The CDM Administrator deletes descriptions from the Data Field/Attribute Use Class Mapping Forms.

Delete the entry in the Attribute Use Class/Data Field Mapping Table for each data field in a deleted record.

SECTION 7

MAINTAINING EXTERNAL SCHEMAS MAPPINGS

7.1 Methodology Overview

This section and its subsections (7.1.1 - 7.1.4) introduce the methodology for building and updating external schemas and for mapping them to the conceptual schema. The portion of the CDM database that contains external schemas and CS-ES mappings is described, and the basic approach to developing both is presented. Detailed instructions for filling out the modeling forms and loading the pertinent CDM database tables are included.

7.1.1 ES and CS-ES Mapping Structure

External schemas are patterned after the relational data model. They are described using only two entity classes: User view and data item. A user view is equivalent to a table in the relational data model; a data item, to a column.

The mapping between the conceptual schema and an external schema has only one level:

Attribute Use Class to Data Item.

7.1.2 Basic Approach

This methodology addresses the following subjects:

- Describing user views and data items in the external schema portion of the CDM.
- Determining the mappings between external schemas and the conceptual schema and describing them in the CS-ES mapping portion of the CDM.
- Updating these descriptions to reflect changes in either the external schemas or the conceptual schema.

A CS-ES mapping is intended to show which components of an external schema correspond to those of the conceptual schema. A data item maps to an attribute use class if they both are the

same kind of data about real-world things. A data item is not to map to more than one attribute use class. In Figure 7-1, the EMP-NAME, DEPT-NAME, and SPOUSE-NAME data items each have only one attribute use class to which to map. If there is more than one attribute use class to which a data item could map, the choice depends on which entity class each attribute use class is in. There must be one entity class that has one entity instance for each row in the user view. If one of the attribute use classes is in that entity class, the data item maps to it. In Figure 7-1, the Employee entity class has one instance for each row in the EMP-MAST user view, so the EMP-NO and DEPT-NO data items map to the equivalent attribute use classes are that entity class. If none of the attribute use classes are in that entity class, the data item maps to the one in the entity class that is most closely related to that entity class. Thus, DIV-NO maps to Div No in the Dept entity class because that entity class is closer to Employee than the Division entity class.

The following subsections (7.1.2.1 - 7.1.2.2) present two subjects to consider when dealing with CS-ES mappings. They are not mutually exclusive; a user view can involve neither, either, or both.

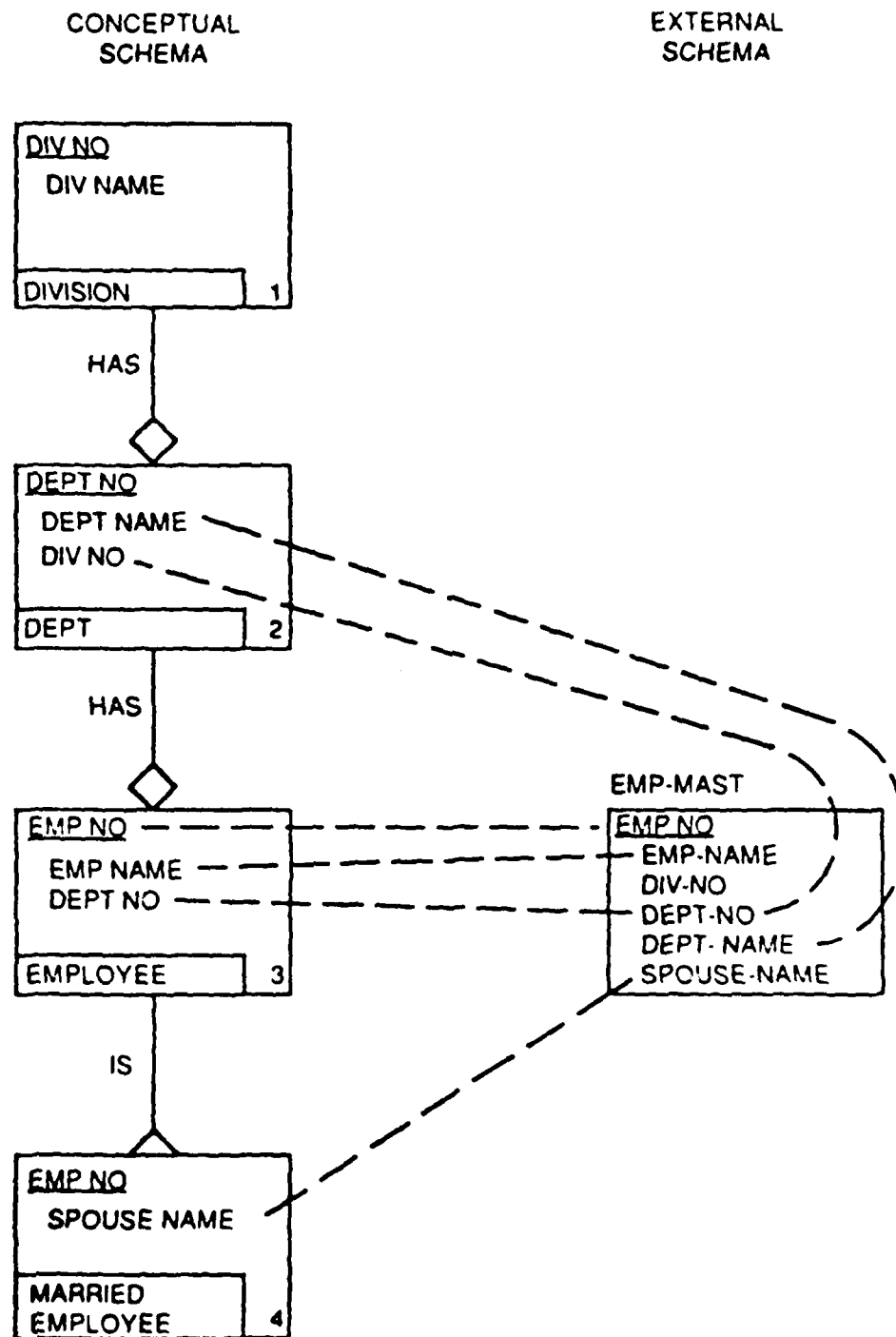
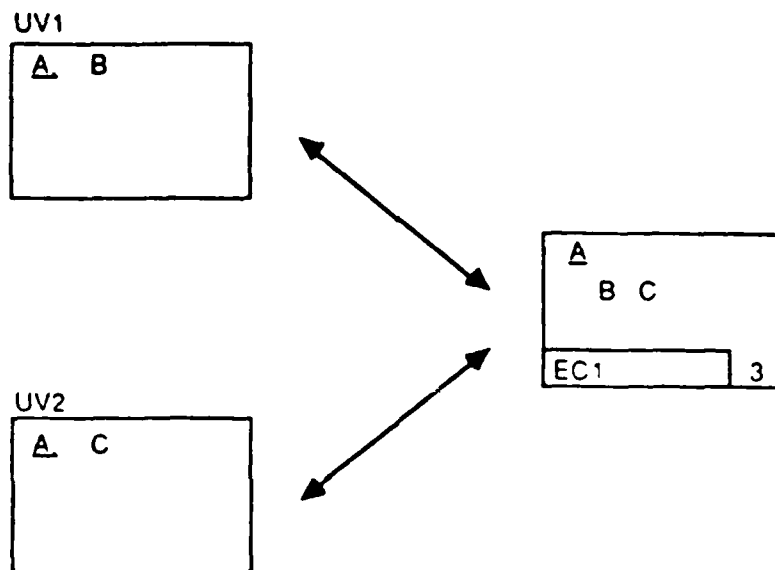


Figure 7-1. Data Item/Attribute Use Class Mappings

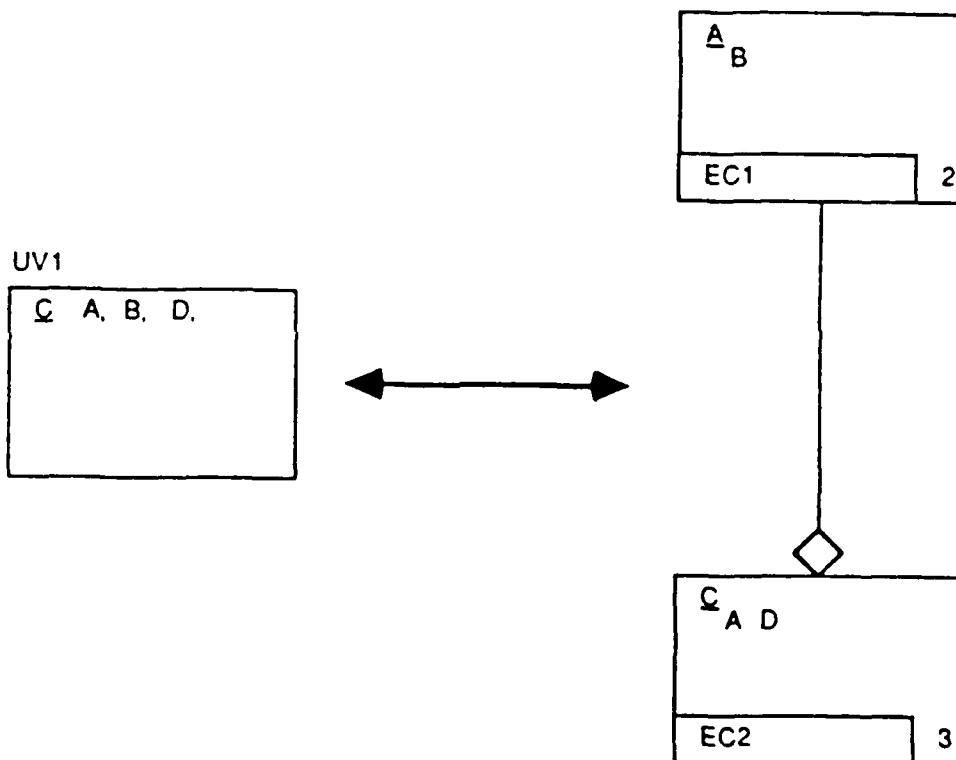
7.1.2.1 Vertical Partitions

An entity class is vertically partitioned when some of its attribute use classes map to data items in one user view and others map to those in another. An entity class can have several vertical partitions.



7.1.2.2 Joins

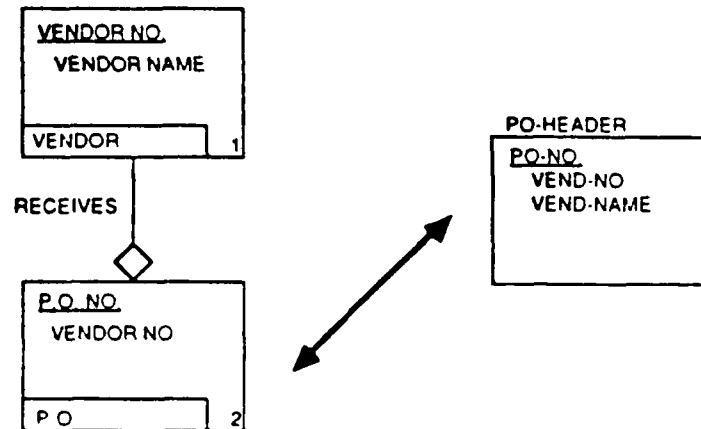
If the data items in a user view map to attribute use classes in two entity classes, those entity classes must be combined to form that user view. This is done with a relational "join" operation, which concatenates the entity instances of one entity class with those of the other. These two entity classes must be directly related by a relation class so that their entity instances can be matched using the key class of the independent and the corresponding inherited attribute use class(es) of the dependent.



If the relation class cardinality is one-to-many, each independent entity instance is concatenated with each entity instance that is dependent on it. In the first example in Figure 7-2, each PO-HEADER instance is formed by concatenating a Vendor instance with a PO instance based on identical values in Vendor No. If a Vendor instance has no dependent PO instances, it is not represented by a PO-HEADER instance. This produces one row in the user view for each instance in the dependent entity class. Since a relational join cannot form user view rows with repeating data items, this concentration cannot be done from dependent to independent.

If the relation class cardinality is one-to-zero-or-one, the concentration can be done in either direction, independent to dependent or dependent to independent, because neither can cause a repeating data field. The second and third examples in Figure 7-2 show these two situations. In the second, there is one BUYER user view row for each Buyer entity instance, and there is no row for an employee who is not a buyer. In the third example, there is one EMP-MAST instance for each Employee instance. If an employee is not married, the SPOUSE-NAME data item in the user view row for that employee contains a null value.

ONE-TO-MANY RELATION CLASS.



ONE-TO-ZERO-OR-ONE RELATION CLASS.

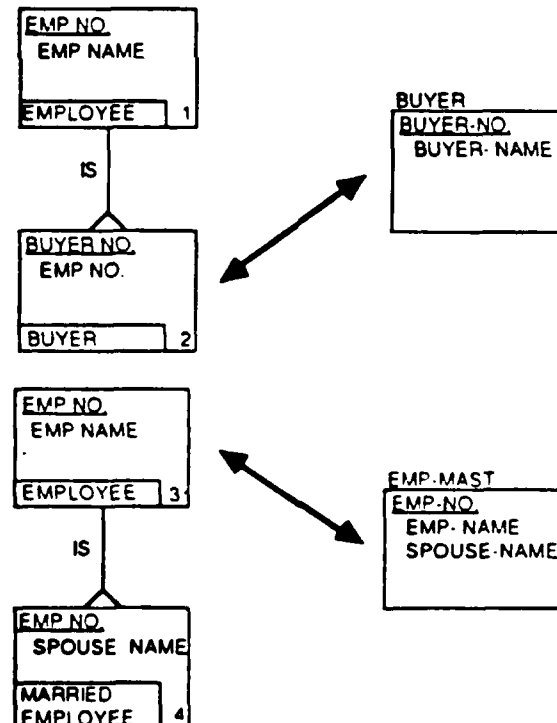


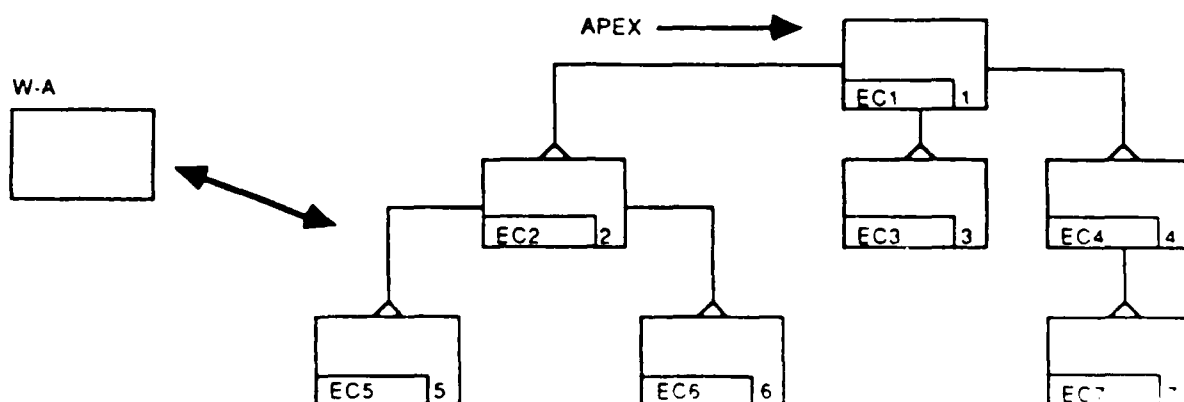
Figure 7-2. ES-CS Join Examples

If the data items in a user view map to attribute use classes in several entity classes, they must all be combined to form the user view. This is done with a series of the join operations described above, each of which combines two of the entity classes. All of the entity classes must be interrelated such that they form one of the following (See Figure 7-3):

1. A regular hierarchy, i.e., a structure in which:
 - One entity class, called the apex, is not dependent on any of the others (e.g., EC1)
 - Every other entity class is dependent on exactly one entity class (not necessarily the same one for all)
 - Every relation class cardinality is one-to-zero-or-one
2. A confluent hierarchy (an upside-down hierarchy), i.e., a structure in which:
 - One entity class, called the apex, has none of the others dependent on it (e.g., EC14)
 - Every other entity class has exactly one entity class dependent on it (not necessarily the same one for all)
 - Any specific relation class cardinality is permitted
3. A combination of:
 - One confluent hierarchy
 - One or more regular hierarchies, each of whose apex entity classes is also in the confluent hierarchy (e.g., EC15, EC20, and EC25).

Each hierarchy is called a join structure. As shown in the examples in Figure 7-3, the user view must have one row for each instance of the apex entity class of the regular or confluent hierarchy. If a combination of hierarchies exists, the user view must have this correspondence to the apex of the confluent hierarchy.

REGULAR HIERARCHY:



CONFLUENT HIERARCHY:

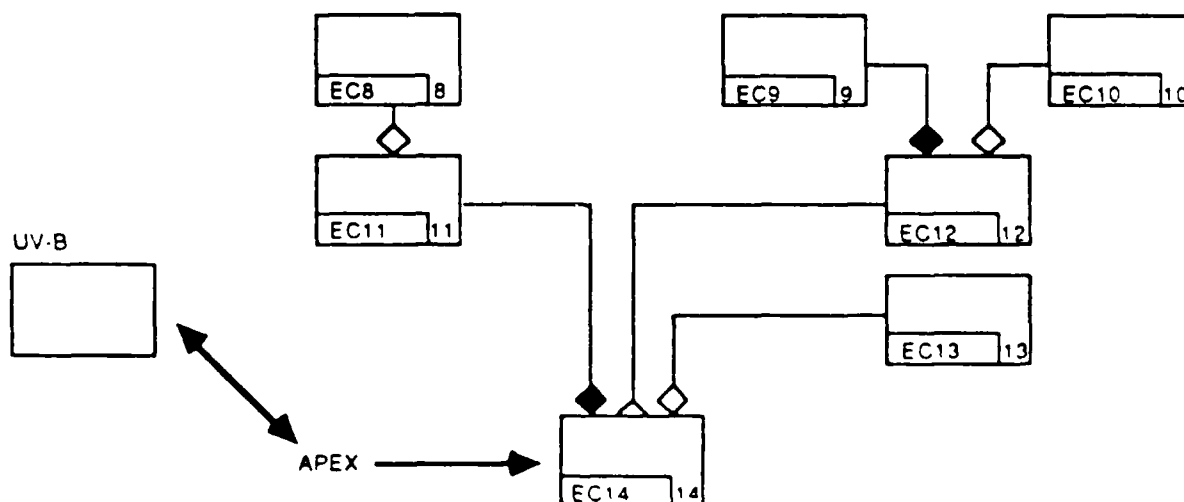


Figure 7-3. ES-CS Join Structures

COMBINATION:

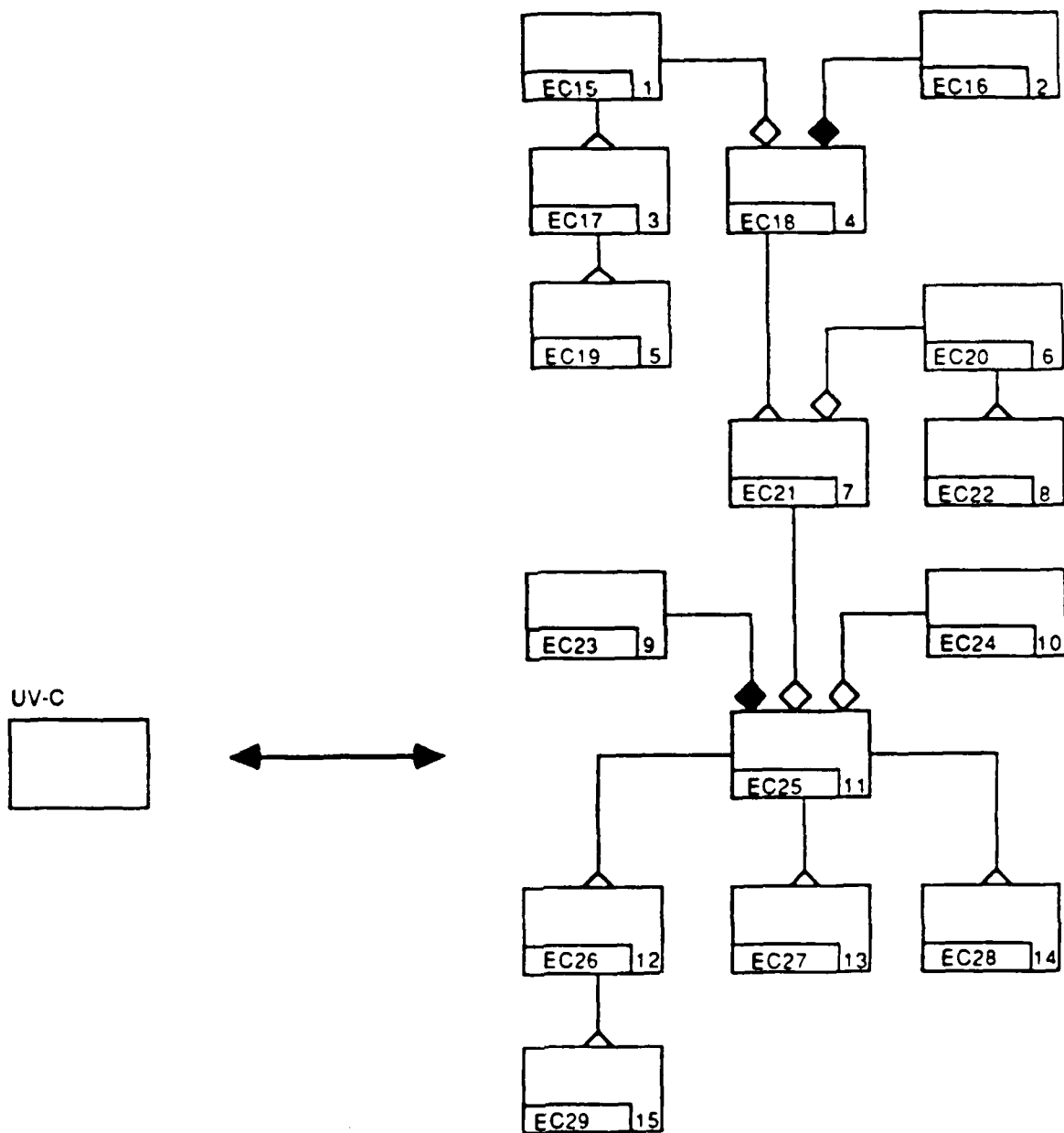


Figure 7-3. CS-ES Join Structures (Continued)

7.1.3 Modeling Forms

The IISS NDDL is used to describe external schemas. Since they are simple in structure, consisting of only user views and data items, no external schema modeling forms are needed.

The following forms are used to model the mappings between external schemas and the conceptual schema:

User View Join Structure Diagram
Data Item/Attribute Use Class Mapping Form

The rest of this section contains a detailed description of each of these forms.

User View Join Structure Diagram

Purpose: To provide a single source of information about the join structures for a user view.

Instructions:

Fill in one page for each join structure for a user view whose data items map to attribute use classes in two or more entity classes.

<u>Form Area</u>	<u>Explanation</u>
1. User View No.	Unique identification code assigned to the user view by the CDMA.
2. User View Name	Name or code by which users identify the user view.
3. (Diagram Area)	Depiction of the entity classes and relation classes that make up the join structure.

Data Item/Attribute Use Class Mapping Form

Purpose: To provide a single source of information about the mappings between external schema data items and conceptual schema attribute use classes.

Instructions:

Fill in one or more pages for each user view

(external schema). List the attribute use class that each data item maps to.

<u>Form Area</u>	<u>Explanation</u>
1. User View No.	Unique identification code assigned to the user view by the CDMA.
2. User View Name	Name or code by which users identify the user view.
3. Data Item Name	Name or code by which user identify the data item.
4. Entity Class No.	Number of the entity class that contains the attribute use class whose number and tag are in the next two columns.
5. Att. Use Cl.Tag No.	Tag number of an attribute use class to which the data item maps.
6. Attribute Use Class	Name of the attribute use class Tag whose tag number is in the prior column. It is included only to make the entry readable; it is not used in loading the mapping tables.

7.1.4 CDM Tables and ES NDDL

This section explains how to load the following tables in the CS-ES portion of the CDM database:

Attribute Use Class/Data Item Mapping Table
Entity Class/User View Join Table

The following pages are arranged alphabetically by table name within these two portions, i.e., in the sequence shown above.

The tables in the external schema portion of the CDM database are loaded with NDDL statements .

The NDDL CREATE VIEW command is used to define the external schema and map data items from the view to attributes (tag-names) in the conceptual schema.

Attribute Use Class/Data Item Mapping Table

Source Documents:

1. Data Item/Attribute Use Class Mapping pages from the CS/ES mapping model.

Instructions:

<u>Table Field</u>	<u>Source Field</u>
UV No.	User View No. area near the top of the page.
DI Name	Data Item Name column.
Tag No.	Att. Use Cl. Tag No. column. Use the number following the "T"; do not include the "T" itself.

Example:

UV No.	DI Name	Tag No.
2	Loc ID	53

User View Join Linkage Table

Source Documents:

1. User View Join Structure Diagrams from the CE/ES mapping model.

Instructions:

<u>Table Field</u>	<u>Source Field</u>
UV No	User View No. area near the top of the diagram.
Ind EC No	Number in the upper left corner of the independent entity class box.
Dep EC No	Number in the upper left corner of the dependent entity class

<u>Table Field</u>	<u>Source Field</u>

box.

RC No.

Verb phrase connected to the
relation class line by a squiggle
(see sample diagram page).

Example:

<u>UV No</u>	<u>Ind Ec No</u>	<u>Dep EC No</u>	<u>RC Label</u>
4	23	5	Is
4	23	12	Is
4	10	18	Is Satisfied By
4	23	18	Is Treated As

7.2 Building an ES and CS-ES Mapping

Objectives:

- Load the description of an external schema into the following tables in the CDM database:

User View Table
Data Item Table

- Build a model of the mapping between the external schema and the conceptual schema.
- Load the descriptions of the CS-ES mapping into the following tables in the CDM database:

User View Join Linkage Table
Attribute Use Class/Data Item Mapping Table

Note: Use the NDDL CREATE VIEW command to describe the
Attribute Use Class to Data Item mappings.

Please refer to Section 7.1.3 for details on how to fill
out the CS-ES mapping forms and to Section 7.1.4 for
details on how to load these CDM tables.

Tasks:

1. The CDM Administrator loads descriptions for an external schema.

Create one entry in the User View Table using NDDL.

Create one entry in the Data Item Table for each data item in the user view using NDDL.

2. The CDM Administrator determines the mapping for each data item.

Determine what sort of data about the real-world things that the data item represents.

Determine which attribute use class in the conceptual schema represents the same sort of data as the data item. This involves finding the attribute use class whose definition or migration path corresponds to the intent of the data field. The first place to look is the entity class that has one entity instance for each row in the user view. The value in the data item in each row of the user view must be the same as the one in the attribute use class in the corresponding instance of the entity class.

If none of the attribute use classes in that entity class correspond to the data item, the next places to look are the entity classes that are related to that entity class. Again, the value in each user view row must be the same as the value in the corresponding entity instance. If the attribute use class is not in any of these entity classes, the search must be widened to include the entity classes that are related to them. This continues until the proper attribute use class is found or until it is determined that a new attribute class must be added to the conceptual schema; see Section 4.3.

Fill out a line on a Data Item/Attribute Use Class Mapping Form for the attribute use class to which the data item maps.

3. The CDM Administrator determines any joins that are needed for the user view.

Determine whether the data items in the user view map to attribute use classes in more than one entity

class. This can be done by comparing the entity class numbers that are entered on the Data Item/Attribute Use Class Mapping Forms for the user view. If all the numbers are the same, the data items all map to attribute use classes in one entity class.

If the data items map to attribute use classes in more than one entity class, prepare a User View Join Structure Diagram. The entity classes must form one or more join structures as described in Section 7.1.2.2. If the join structures are not contiguous, one or more additional entity classes may be needed.

4. The CDM Administrator loads descriptions from the User View Join Structure Diagrams.

Create one entry in the Entity Class/User View Join Table for each relation class in the diagram.

5. The CDM Administrator loads descriptions from the Data Item/Attribute Use Class Mapping Forms.

Create one entry in the Attribute Use Class/Data Item Mapping Table from each line on each form.

Note: The NDDL CREATE VIEW command is used to define the external schema and map data items from the view to attributes (tag-names) in the conceptual schema.

7.3 Modifying/Deleting ES Elements and CS-ES Mappings

Prior to modifying or deleting elements of the ES or the CS-ES, the CDM Administrator must assess the impact of the proposed change on the other components of the CDM. The objective of this section is to provide the CDM Administrator with an approach to the analysis of the impact that a change in the ES or CS-ES might have upon the other areas of the CDM or on software modules, such as user APs and generated APs.

The approach that is taken in analyzing the impact that a change to the ES or CS-ES might have to other areas of the CDM or to a software module is to list the changes that might be made and then for each of those changes to identify the other changes that would have to be made either in the ES or another schema or in an ES-CS or an IS-CS mapping or in a software

module. Changes that do not impact any other areas are omitted.

A similar section appears in the discussions on the Conceptual Schema and on the Internal Schemas and the IS-CS Mappings, Sections 5 and 6 respectively.

The following assumptions about the nature of the changes to the External Schema and the CS-ES Mappings and the sequence in which they are made have been taken in order to perform the analysis:

1. Components of an external schema are added in the following sequence
 - User Views
 - Data Items
2. All changes in the external schema that are needed to support a change in an ES-CS or IS-CS mapping are made before the ES-CS or IS-CS mapping is changed.
3. A change in the name or definition of a component of the external schema is for cosmetic purposes only and does not alter the basic meaning of that component.

Finally, a note of explanation about how the changes and their impacts are organized. Only the direct impacts of a change are listed with it. If one change results in a cascade of other changes, only the first in the cascade is listed with the initial change. Each subsequent change is listed as an impact of the one immediately before it. So to find the total extent of the impact of a change, one must trace from the initial change to each change that it results in and, then to each in which that change impacts.

Figure 7-4 shows the relationship between the change and the possible impacts upon other parts of the CDM that the change may affect.

Overview Matrix	A change to	
	User View	Data Item
User View		X
Data Item	X	
EC-UV Join	X	
AUC-DI Map.		X
Software Mod	X	X
UV Reference	X	

Figure 7-4. Impact of External Schema Changes

7.3.1 User View (ES) Changes:

The potential impact from the changing User Views is as follows:

- Add a new user view.
Add all the data items that the user view contains.
Add any EC-UV joins from which the user view results.
- Change a user view name.
Change the user view name in any software modules in which it appears and recompile those modules.
Change the user view name in any user view references in which it appears.
- Change a user view definition.
No other impact.
- Change a user view keyword.
No other impact.

- Change the entity class joins that must be done to form a user view.

Add, change, and delete EC-UV joins as necessary.

Recompile any software modules in which the user view appears.

- Delete a user view.

Delete all the data items contained in the user view.

Delete any EC-UV joins from which the user view results.

Remove the user view from any software modules in which it appears and recompile those modules, or discard them entirely.

Delete any user view references in which the user view name appears.

7.3.2 Data Item Changes:

The potential impact from the changing of Data Items is as follows:

- Add a new data item.

Add an AUC-DI mapping to specify any attribute use class to which the data item maps.

- Change a data item name.

Change the data item name in any AUC-DI mapping for the data item.

Change the data item name in any software modules that access it and recompile those modules.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that access a data item.

- Change a data item definition.

No other impact.

- Change a data item keyword.

No other impact.

- Change the data description of a data item.

Change the data description of the data item in any software modules that access it and recompile those modules.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that access a data item.

- Change the mapping between a data item and the attribute use classes to which it corresponds.

Add, change, and delete AUC-DI mappings as necessary.

Recompile any software modules that access the data item.

Note: Neither the CDM database nor the CDM1 model contains the information needed to identify the software modules that access a data item.

- Delete a data item.

If the data item is the last or only one in the user view, delete the user view also.

Delete any AUC-DI mapping for the data item. Remove the data item from any software modules that access it and recompile those modules, or discard them entirely.

7.3.3 Summary

The following points are offered in summary:

1. A change in an external schema can result in additional changes in that schema, in its ES-CS mapping, and in

software modules. However, it cannot impact other external schemas or ES-CS mappings, nor any internal schemas or IS-CS mappings, nor the conceptual schema.

2. A change in an ES-CS mapping is always the result of another change to either the corresponding external schema or to the conceptual schema.

Always use the NDDL CREATE VIEW command to describe the ES-CS mappings.

3. The information in the CDM database and the CDM1 model is inadequate for identifying the software modules that are impacted by most schema changes. Specifically, the following information needs to be added:

- The data items that are accessed by a software module that contains user views.
- The databases, record types, data fields, record sets, record set members, and database areas that are accessed by a software module that accesses databases directly.
- The record types, data fields, record sets, record set members, and database areas that are accessed by a generated AP.

APPENDIX A

GLOSSARY

Alpha-Numeric Data Format

A data format for values that can contain characters other than numerals (0-9). Numerals may be permitted also.

Attribute Class

A collection of all the same kind of attributes, i.e., those that have the same meaning. An attribute is a characteristic or fact about an entity. An attribute consists of a name (e.g., employee hire date) and a value (e.g., 15 August 1980). An attribute value may be:

- A. Nondivisible (e.g., state name)
- B. Divisible, i.e., a concatenation of two or more other attribute values (e.g., part number formed by concatenating drawing number and material code).
- C. Computed from one or more other attribute values (e.g., age computed as current date minus birth date).

Attribute Class Data Description

A generic data description that applied to a particular attribute class.

Attribute Use Class

A model attribute class that appears in a model entity class. Each attribute use class represents either an owned attribute class or an inherited attribute class.

Attribute Use Class/Data Field Mapping

Indicates that an attribute use class corresponds to a data item; i.e. that they have the same meaning and that the data item can be used to access the values for the attribute use class.

Attribute Use Class/Data Item Mapping

Indicates that an attribute use class corresponds to a data item; i.e., that they have the same meaning and that the data item can be used to access values for the attribute use class.

Attribute Use Class/Internal Schema Mapping

Indicates that an attribute use class corresponds to some portion of an internal schema.

Attribute Use Class/Record Set Mapping

Certain attribute use classes can be represented in a database by a group of record sets rather than be a data field. For example, Project: Task record sets called Pending, In-Process, On-Hold, and Completed. An attribute use class/record set mapping indicates that a particular record set corresponds to a particular attribute use class value.

Component Data Field

A data field that is part of another data field; e.g., if data field A is made up of data fields B, C, and D, each of these latter data fields is a component of A. A data field cannot be a component of more than one other data field.

Component Domain

An elementary domain that is part of another domain; e.g., a Date domain might be made up of a Month domain, a Day of Month domain, and a Year domain. Each of these latter domains would be a component of the Date domain. An elementary domain can be a component of several other domains.

Component Unit of Measure

An elementary unit of measure that is part of another unit of measure; e.g., the "inch" unit of measure is a component of the "foot-inch" unit of measure. An elementary unit of measure can be a component of several other units of measure.

Conceptual Schema

The description of all the shared data items within an enterprise's databases and of the allowable operations on and integrity constraints for those shared data items. Represented

AD-A101 377

INTEGRATED INFORMATION SUPPORT SYSTEM (IIS) VOLUME 5

4/4

COMMON DATA MODEL 5. (U) GENERAL ELECTRIC CO

SCHEMATIC MY PRODUCTION RESOURCES CONSU..

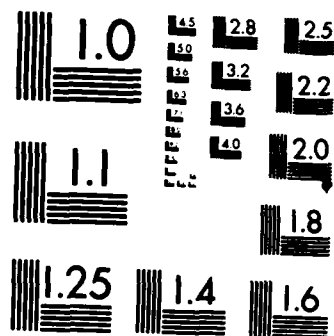
UNCLASSIFIED

D ROLLINS ET AL. 01 NOV 85 UN-620141001

F/G 5/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

by a fully normalized information model in which integrity constraints have been completely specified. Not influenced by any usage or storage considerations. A software module that must be used to access or transform data that is stored in a manner that the CDMP is not designed to handle.

Constraint Statement

One complete NDDL description of either an assertion, a trigger, or a horizontal partition fragment. An assertion is a rule about values for attribute use classes. If an NDML command attempts to violate an assertion, the CDMP rejects the command with an error message. A trigger is a set of conditions and a set of actions, both involving entity classes and attribute use classes. If the conditions are satisfied all the actions are taken. If the conditions are not satisfied, none of the actions are taken. See the definitions of Horizontal Partition and Horizontal Partition Fragment for details about this use of constraint statements.

Database Area

A subdivision of a CODASYL database. This subdivision is a technique for improving the efficiency accessing database record type instances. When a database is subdivided into database areas, some or all of its records types are assigned to particular areas. Instances of these record types are stored only within the assigned areas. Then, these record type instances can be accessed by searching only the appropriate areas rather than the entire database. This access method is only used when the record type instances cannot be located by other means (e.g., by calc keys or record sets).

Database Area Assignment

Indicates that a record type is assigned to a database area.

Database Directory

A software library that must be used when accessing a database.

Database Password

A code that must be supplied when logging on to a DBMS to use a database. The DBMS verifies the password before accepting

any other messages.

Data Field

A portion of a record type in which data values can be stored.

Data Field/Record Set Linkage

A data field in a variable data set in a TOTAL database that is used as the variable control key for a linkpath from a master data set.

Data Field Redefinition

A data field that occupies the same space in a record type as another data field. A record instance cannot contain values in both data fields. One instance can contain a value in one field while another contains a value in the other.

Data Format

The portion of a generic data description that includes the structural characteristics such as data type, length, storage method, etc. If a generic data description is for elementary values (e.g., customer names), it will have only one data format (e.g., Data Type - alphanumeric, Length = 30). If it is for compound values (e.g., part numbers consisting of six numerals followed by three letters followed by four more numerals), it will have more than one data format, one for each elementary portion of the values. For the part number example the data formats would be:

1.	Data Type = numeric	Length = 6
2.	Data Type = alphabetic	Length = 3
3.	Data Type = numeric	Length = 4

A generic data description with a compound unit of measure, i.e., one that is a group of component unit of measures, must have a data format for each component unit of measure.

Data Item

An attribute class as seen by a user in a user view, i.e., a kind of data (e.g., employee hire date), not a particular data value (e.g., 15 August 1980).

Data Management System

Either a database management system or a file management system, i.e., a set of computer programs that must be used to establish and maintain a database or a computer file.

Data Type

The combination of a type of values (e.g., alphanumeric, signed numeric, etc.) and a type of storage (e.g., binary, packed, etc.)

Dependent Entity

The entity class that is dependent in a specific relation class. A dependent entity, i.e., an entity is a dependent entity class, can exist only if it is related to an independent entity. Contrast with independent.

Description Type

A generic object may have several different kinds or styles of description (short, long, technical, nontechnical, etc.). Each is a description type.

DMS on Host

A data management system that is available on a particular host.

Domain

A set of rules about the values that are allowed for a data item, attribute class, or data field. A domain is either an elementary domain or a group of two or more elementary domains, called component domains.

Domain Range

A series of consecutive values that represent all or part of an elementary domain.

Domain Value

A single value within an elementary domain.

Elementary Data Field

A data field that does not have any component data fields.

Elementary Domain

A domain that does not have any component domains. An elementary domain can be expressed as a series of values or value ranges.

Elementary Unit of Measure

A unit of measure that does not have any component units of measure.

Entity Class

A collection of similar entities, i.e., those that have the same kinds of attributes. An entity is a person, place, event, thing, concept, etc.

Entity Class/Record Type Join

A relational join operation that combines two related entity classes as part of the design of a record type.

Entity Class/Record Type Mapping

Indicates that an entity class corresponds to a record type, i.e., that they both have the same meaning and that the record type can be used to store instances of the entity class.

If a record type has more than one EC-RT mapping, some of its instances correspond to instances of one entity class while others correspond to instances of another, i.e., the record type is the relational union of the entity classes. An example is a Replenishment Order record type that maps to both the Purchase Order and Manufacturing Order entity classes. Each record instance represents either a purchase order or a manufacturing order.

Entity Class/Record Type Union Discriminator

If a record type corresponds to more than one entity class, i.e., if it has more than one EC-RT mapping, it is the relational union of those entity classes. Some instances of such a record type correspond to instances of one of the entity

classes, others to those of another. For such a record type there must be a way to determine which record instances correspond to instances of each entity class. An entity class/record type union discriminator provides this by specifying that a given value in a given data field indicates that a given EC-RT mapping should be used.

Entity Class/User View Join

A relational join operation that combines two related entity classes as part of the design of a user view.

External Schema

See User View.

File

A set of stored data that is managed by a file management system (e.g., VSAM).

File/Database

A set of stored data, i.e., either a computer file (e.g., a VSAM or flat file) or a database (e.g., an ORACLE or IMS database).

Generated Request Processor

A software module that was created by the CDMP Precompiler.

Generic Data Description

A detailed description of the values for one or more data items, attribute classes, data fields, and/or module parameter. It includes format, measurement, and domain characteristics of the values.

Generic Data Description Component Unit of Measure

A component unit of measure that is specified as part of a data format. These are only specified for a generic data description that includes a compound unit of measure, i.e., one that is a group of component units of measure.

Generic Data Description Domain

A domain that is specified as part of a generic data description.

Generic Data Description Unit of Measure

A unit of measure that is specified as part of a generic data description.

Generic Object

Anything with a name that distinguishes it from other things of the same type and with a description that explains what it is (e.g., any entity class or attribute class).

Generic Object Description

An explanation of what a particular object is.

Generic Object Description Line

One fixed-length portion of a generic object description.

Generic Object Keyword

A keyword for a particular generic object.

Generic Object Name

An noun or noun phrase by which a generic object is known. Two objects can have the same name.

Horizontal Partition

Indicates that the same record type is not used to store all instances of an entity class, i.e., that one is used to store some instances while another is used to store others. Each record type represents a "fragment" of the entity class. These fragments do not overlap, i.e., no entity instance appears in more than one fragment. An entity class can be partitioned into any number of fragments, usually with each being in a different database or file, although that is not a requirement; some or all may be stored as different record types in the same database or file. A constraint statement defines each fragment, i.e., describes the conditions that must be met by each entity instance that is stored as a given record type. If an entity

class is replicated, i.e., if each of its instances is stored in more than one database instances is stored in more than one database or file, each replication can be horizontally partitioned. For example, for the first replication the instances could be partitioned based on the values in one attribute use class, and for the second replication they could be partitioned based on the values in another.

Horizontal Partition Fragment

A record type that is used to store some, but not all, of the instances of an entity class. A constraint statement describes the conditions that must be met by each entity instance that is stored as the record type. If the conditions are satisfied by the attribute values of an entity instance, it can be stored as an instance of the record type; otherwise, it cannot be.

Host

A computer in the IISS.

IMS Segment

A record type in a database that is controlled by IBM's IMS DBMS.

Independent Entity

The entity class that is not dependent in a specific relation class. An independent entity, i.e., an entity in an independent entity class, can exist without being related to a dependent entity. Contrast with dependent entity class.

Inherited Attribute Class

An attribute class that appears in a dependent entity class because it has migrated from an independent entity class. Must be part of a key class in the independent entity class.

Inherited Attribute Classes Form

Provides a single source of information about inherited attribute use classes that are to be described in the conceptual schema.

Inherited Key Class

A key class in the independent entity class of a relation class that has migrated to appear in the dependent entity class of that relation class.

Internal Schema

A description of the data items in a database. Described from DBMS User's perspective. Usually not fully normalized.

Join

A relational operator that creates a new relation by combining two or more source relations according to specified criteria. A natural join combines the relations by matching tuples with equal values for a common attribute class (column).

Key

An assortment of attributes in an entity that can be used to uniquely identify that entity within its entity class. An entity can have more than one key, e.g., an employee can be uniquely identified by either an employee number or a Social Security Number.

Key Class

A group of one or more of an entity's attributes that can be used to uniquely identify the entity within its entity class. An entity can have more than one key. A key class is a collection of the attribute classes whose member attributes comprise the keys for the entities in an entity class. An entity class has the same number of key classes as each of its member entities has keys. For example, if each entity has three keys, the entity class has three key classes.

Key Class Member

An attribute use class that is part of a key class.

Key Class Migration

The process of moving key classes from independent to dependent entity classes.

Library Module

A software module that is stored in a software library.

Model

A representation of the information requirements of all or part of an enterprise in terms of entity classes, relation classes, and attribute classes.

Model Glossary Name

A name of a model entity class or a model attribute class, either an official name or an alias.

Module Parameter

A means of supplying values to a software module and of receiving results from a module.

Numeric Data Format

A data format for values that can only contain numerals (0-9) and associated punctuation (decimal point, comma, etc.).

Owned Attribute Class

An attribute class that is not an inherited attribute class.

Owned Attribute Classes Form

Provides a single source of information about owned attribute use classes that are to be described in the conceptual schema.

Program Control Block

A portion of a PSB that describes and controls how an IMS database can be accessed.

Program Specification Block

A group of logical views of IMS databases that is used for interacting with the IMS DBMS.

Record Set

An association between one record type, called the owner, and one or more other record types, called the members.

Record Set Member

A record type that is a member of a record set.

Record Type

A group of data values that are stored together as a unit in a computer file or database. A record type is the collection of all the records of the same kind, i.e., all the records that contain the same kind of data values.

Relation Class

An association between an entity in one entity class and one in another. A relationship has a label that describes the association. For example, a customer named ABC Corp. is associated with an order numbered 123 in a manner labeled "placed". A relation class is a collection of the identically labeled relationships between the members of the same two entity classes. Each relation class is either specific or nonspecific.

In a specific relation class, one entity class is "independent" while the other is "dependent"; i.e., entities in the first can exist without being associated with any in the second, but those in the second cannot exist without being associated with one in the first. One key class from the independent entity class "migrates" through each specific relation class to appear in the dependent entity class as inherited attribute classes.

In an nonspecific relation class, neither entity class is dependent on the other; i.e., entities in either entity class can exist without being associated with any in the other. For convenience, one entity class is arbitrarily called "independent" and the other is called "dependent".

Relation Class Form

Provides a single source of information about relation classes that are to be described in the conceptual schema.

Relation Class/Record Set Mapping

Indicates that a record set represents the same association as a relation class. If a record set has more than one member record type, it may represent several relation classes, a different one for each member. Hence, this entity class is only indirectly dependent on record set (via record set member).

Repeating Data Field Occurrence Counter

A data field whose data values indicate how many occurrences of a repeating data field actually contain values.

Segment Data Element

A data field is an IMS segment.

Software Library

A computer file in which software modules can be stored.

Software Module

A set of computer instructions that are treated as a whole (i.e., stored, compiled, and executed together).

Subschema

The description, in the DDL of a CODASYL DBMS, of all or part of a database. For IISS, only one subschema is needed for a CODASYL database, and it must describe all the common data within the database that is to be accessible with NDML.

Unit of Measure

A standard scale for determining the magnitude of something. Examples include inch, foot, foot-inch, meter, ounce, pound, hour, minute, second, etc.

Unit of Measure Conversion

A means of transforming a value expressed in one unit of measure into an equivalent value expressed in another (e.g., transforming inches to feet or feet to meters).

Unit of Measure Conversion Constant

A number in a unit of measure conversion step that is the same every time the conversion is performed. A software module that can be used to perform a unit of measure conversion. A module parameter that is used to supply values to or receive values from a unit of measure conversion module.

Unit of Measure Conversion Step

One of a series of arithmetic steps that can be used to perform a unit of measure conversion. Each step takes the value resulting from the prior step (the first step uses the value to be converted) and adds, subtracts, multiplies, or divides by another value, either a constant or a variable. The result of the last step is the converted value. The processing sequence is always first steps to last; parentheses, branching, and conditional tests are not allowed. Consequently, some unit of measure conversions cannot be performed in this manner (e.g., converting meters to feet-and-inches).

Unit of Measure Conversion Variable

A number in a unit of measure conversion step that can be different every time the conversion is performed. This is only used when the unit of measure being converted from has two or more component units of measure. Each component is a variable and each is involved in a separate step.

User Application Process

A software module that supports business activities rather than data processing activities and that can be executed directly, i.e., a main routine, not a subroutine. A user AP may contain NDML commands for accessing stored data via the CDM, or it may access them directly via DMSs, or it may call subroutines that contain NDML commands or that access stored data directly.

User View

A group of data items that a user wants to deal with as a group. It is similar to an entity class but does not necessarily meet all the conditions for being one, it can be thought of as an unnormalized entity class. A user view is often the result of combining several entity classes via relational join operations and selecting particular attribute use classes as data items via relational project operations.

UM 620141001
1 November 1985

Vertical Partitions

An entity class is vertically partitioned when some of its attribute use classes map to data items in one user view and others map to those in another. An entity class can have several vertical partitions.

APPENDIX B

REFERENCES

ICAM Life Cycle Documents

FTR11021000U	<u>Volume V, Information Modeling Manual</u> <u>(IDEF1)</u>
PRM620141200	<u>Embedded NDML Programmers Reference Manual</u>
UM620141100	<u>Neutral Data Definition Language (NDDL)</u> <u>User's Guide</u>
UM620141002	<u>Information Modeling Manual - IDEF1-</u> <u>Extended (IDEF1X)</u>
TBM62014100	<u>CDM1 - An IDEF1 Model of the Common Data</u> <u>Model</u>

Other References

"The ANSI/X3/SPARC DBMS Framework: Report of the Study Group on Data Base Management System" American National Standards Institute, AFIPS Press, Montrole New Jersey, 1977.

Atre, S., "Data Base, Structure Techniques for Design, Performance, and Management", John Wiley and Sons, Inc., New York, 1980.

Martin, James, "Managing the Data Base Environment", Volumes I and II, Savant Institute, 1981.

END

7-87

DTIC