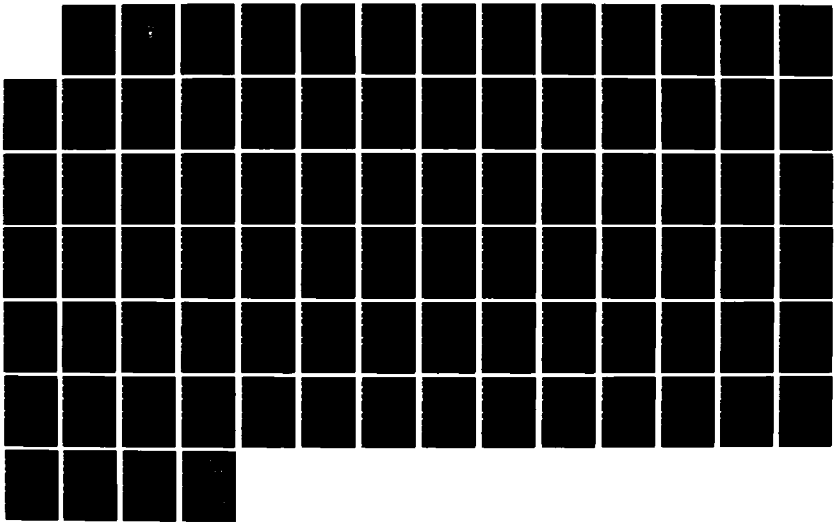
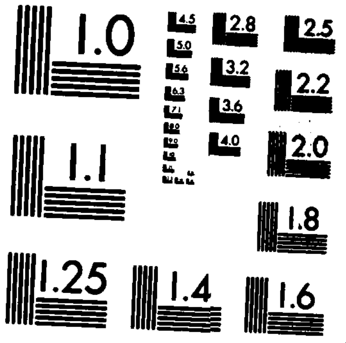
AD-A164 998 RELATIONAL MODEL OF A DATA DICTIONARY(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA M G DEDEOGLU DEC 85 1/1

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

RELATIONAL MODEL OF A DATA DICTIONARY

by

M. Gokhan Dedeoglu

December 1985

Thesis Advisor: Daniel R. Dolk

SECURITY CLASSIFICATION OF THIS PAGE AD-A164998

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)			
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b. OFFICE SYMBOL *(If applicable)* 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School			
6c. ADDRESS *(City, State, and ZIP Code)* Monterey, CA 93943-5100		7b. ADDRESS *(City, State, and ZIP Code)* Monterey, CA 93943-5100			
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER			
8c. ADDRESS *(City, State, and ZIP Code)*		10. SOURCE OF FUNDING NUMBERS			
		PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
RELATIONAL MODEL OF A DATA DICTIONARY

12. PERSONAL AUTHOR(S)
Dedeoglu, M. Gokhan

13a. TYPE OF REPORT Master's Thesis	13b. TIME COVERED FROM ____ TO ____	14. DATE OF REPORT *(Year, Month, Day)* 1985 December	15. PAGE COUNT 85

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	Data Dictionary, Relational Model, ORACLE, Prolog Expert Systems

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*
The data dictionary system is an important tool for supporting information resource management. It facilitates the management and control of data. This thesis will develop a relational model of a data dictionary and implement it on the ORACLE relational data base management system. Then, this data dictionary model will be implemented using the logic-oriented Prolog language. The Prolog model of a data dictionary will demonstrate that logic programming can be used for relational data base applications and that it provides more powerful dictionary capabilities than the relational model. [illegible]

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Prof. Daniel R. Dolk	22b. TELEPHONE *(Include Area Code)* 408-646-2260	22c. OFFICE SYMBOL 54Dk

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Relational Model of a Data Dictionary

by

M. Gokhan Dedeoglu
Lieutenant(j.g.) Turkish Navy
B.S., Turkish Naval Academy , 1979
B.S., Technical University of Istanbul , 1983

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1985

Author: Gokhan Dedeoglu
M. Gokhan Dedeoglu

Approved by: Daniel R. Dolk
Daniel R. Dolk, Thesis Advisor

David K. Hsiao
David H. Hsiao, Second Reader

[illegible]
Vincent Y. Lum, Chairman,
Department of Computer Science

K.T. Marshall
Kneale T. Marshall,
Dean of Information and Policy Sciences

ABSTRACT

The data dictionary system is an important tool for supporting information resource management. It facilitates the management and control of data.

This thesis will develop a relational model of a data dictionary and implement it on the ORACLE relational data base management system. Then, this data dictionary model will be implemented using the logic-oriented Prolog language. The Prolog model of a data dictionary will demonstrate that logic programming can be used for relational data base applications and that it provides more powerful dictionary capabilities than the relational model.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

The author wishes to gratefully acknowledge his thesis advisor, Prof. Daniel R. Dolk, for suggesting the basis of this thesis, and for his invaluable advice and guidance during the course of this work.

The author would also like to express his appreciation to Prof. David H. Hsiao for his constructive critism as a second reader.

I. INTRODUCTION

Data is a resource to be managed. Data is processed to produce information. Data must be administered and controlled to coordinate data usage in order to produce information. The transformation of data into information is the primary function of an information system. Information supports the enterprise's structure and generates its business processes. Enterprises need to manage their information by centrally defining and storing their data resource.

The data dictionary system (DDS) plays an active and central role in the management and control of the corporate data resource. It is the repository of the information needed by the enterprise. A central source of documentation helps improve communication between involved personnel and offers improved system development.

The DDS has become basic to all phases of data administration. The DDS is used to satisfy requirements for information resource management to aid in database design and to provide the information needed for the effective auditing of the data. The DDS can support many aspects of the information resource management environment involving the management and use of data.

The first part of Chapter 2 of this thesis explains the concept of data dictionary system. It explains metadata, metadatabase, data dictionary system, and the functions of the data dictionary system.The second part of Chapter 2 surveys seven commercially available data dictionary systems. The general characteristics of these dictionary systems are discussed in this survey. In Chapter 3, a relational model data dictionary will be developed and implemented on the ORACLE relational database management system. Although a relational DBMS has many advantages over other systems, it has limited dictionary capabilities.The first part of the Chapter 4 will explain the general characteristics of expert systems. In the second part of the Chapter 4 a Prolog model of a data dictionary system will be developed. Because of the characteristics of Prolog, the rules about the information resource management data can be defined more easily than in the relational DBMS environment. By using the Prolog model we can implement information resource management effectively and efficiently.

II. DATA DICTIONARY SYSTEM

A. METADATA

In order to manage data as a resource, it is essential that data about data be clearly specified by data objects. These data objects are called entities. In a data base environment the entities are represented in the form of metadata entities such as data elements, records, files, or data bases.

Metadata entities are described by means of metadata,that is, data about data. Metadata and user data are different from each other. Metadata is used to describe the characteristics of user data.

An example of the metadata for a data element in a metadatabase is given in Fig. 2.1 . In this example, for the data element "USNAME", the data dictionary contains the attributes like description, length, and relationship, but not the actual name of user. Thus, metadata contains descriptive and definitional information about the data.

```
Name of metadata entity: data element
Identification: USNAME
Description: user name is entered as last name,
             first initial, middle initial.
Length/size: 30 characters, only alphanumeric
             characters allowed.
Relationship/usage: this entity is used in files A, B,
             and programs AA, and BB.
```

Figure 2.1 Example of the metadata for a data element.

B. DICTIONARY AND DIRECTORY METADATA

The dictionary metadata is used by the system users. In contrast, the directory metadata is used by the system components. Directory metadata provides information about the physical location of the data. It shows how the data can be accessed, and it contains information about the internal representation of the data entity.

The system which contains directory metadata is called data dictionary/ directory system. Current systems do not separate dictionary and directory functions. They offer only a partial independence between these functions.

C. METADATABASE

A metadata database is a collection of managed, controlled, and related metadata and is referred to as a metadatabase. The characteristics of the metadatabase are the same as those of a user database which include data sharing, data integrity, and data independence. The metadatabase is shared among the user groups, processes, and automated systems such as database management systems, report generators, and query processors.

D. METADATABASE MANAGEMENT

The metadata needs a metadatabase management system, just as the user data needs a database management system for organization, access, and control. The data dictionary system supports the management and control of the metadata

The DDS is a metadatabase management system which provides user/system interface functions, such as query processing and report generation required for the data usage. The DDS also supports many administration and control activities required for metadata management.

E. THE DATA DICTIONARY SYSTEM

A data dictionary system is a centralized repository of data about data. A DDS is used for management and control of data resources. It is a data base about the data bases, and users of the data bases. The centralization of data suggests that there is enterprisewide coordination and control of the metadata. A DDS provides a wide range of facilities and capabilities to support metadata management.

A DDS can be implemented in different forms. The scope of a data dictionary can be narrow. For example, it can consist of simple programs that cover only the data base definitions to support a DBMS. On the other hand, it can be implemented as a very sophisticated data resource management and control tool that cover all the data important to an organization.

A DDS can use a DBMS in its implementation, that is, it can be a DBMS-dependent system, or it can be an independent system. Thus, it can have a passive role by producing information about the data base, or it can force other software to manage and control the data.

F. ACTIVE AND PASSIVE DATA DICTIONARY SYSTEMS

There are two important implementation strategies for integrating the DDS into the operating environment : an active DDS, and a passive DDS.

In an active DDS, processes or system components are fully dependent upon the DDS for its metadata, that is, the only source of metadata is in the DDS. By contrast, in a passive DDS, processes and system components are not dependent upon the DDS for its metadata. The required metadata is obtained from other resources.

The active DDS has several advantages. It eliminates redundant metadata definition, insures consistency in the metadata, controls the metadata usage and metadata changes.

Also, it achieves a great data independence by separating the physical view from the logical view.

Although active DDS has several advantages, it has some drawbacks. It introduces an overhead when binding time is accomplished during execution. Also, the dependency of processing components to the DDS causes bottlenecks in some systems. [Ref. 1]

G. FUNCTIONS OF A DATA DICTIONARY SYSTEM

The functions of a typical DDS are the following :

1. Maintenance Function

This function enables entities, relationships, and attributes to be added, modified, and deleted from the dictionary. The DDSs provide dictionary maintenance commands to perform this function. Execution of these commands is subject to security and restrictions. There are several maintenance methods. Some systems offer batch input to enter the new data. Other systems allow extraction of dictionary data from existing file and database descriptions as well as entering this data directly into the system.

2. Extensibility Function

The extensibility function allows a user to modify the standard dictionary schema to suit specific enterprise needs. The user can add new entities and attributes to the dictionary structure, and establish new relationships. Some DDSs offer a specific meta-entity for extensibility feature. Others offer a generic mechanism to allow the user to add new entities and attributes.

3. Report Processor Function

Report processor function provides reports on a number of dictionary entities. Report facilities are invoked by means of specific commands. The common categories of reports are :

1. Reports of some or all entities of a given type.
2. Reports on all attributes for a specified entity of any type.
3. Usage reports which show either how a given entity is used by other entities, or how other entities use a given entity.
4. A keyword-in context (KWIC) or keyword-out-of-context (KWOC) facility that is used to search specified attributes for a given keywords.

4. Query Processor Function

This function provides information about the usage of dictionary entities, keyword searches, and synonym searches. It allows English-like queries of the DDS. This function is most often used in an interactive mode.

5. Convert Functions

The convert functions scan application programs, library files, and dictionary schemata to generate metadata from these sources to be input to the DDS maintenance function. There are several options for a convert function like changing names, selecting lines to scan, selecting types of transactions.

6. Software Interface Function

This function provides metadata to other software systems such as DDL processors and compilers. Software systems access the DDS either statically or dynamically by means of software interfaces. Static interfaces produce formatted statements for the software packages or create encoded control files for their use. Dynamic interfaces provide direct access to other software systems and use high level interface commands.

7. Exit Facility

The exit facility enables the system user to extend the routines delivered by the DDS vendor. For example, a user can code a new security check for accessing an entity.

All DDSs do not contain the exit facility because of possible side effects of user-written routines.

8. Management Function

The management function is responsible for security, integrity, concurrency control and internal access for the DDS. In DBMS-dependent dictionary systems, some of these functions may be subsumed by the DBMS itself. [Ref. 2]

H. SURVEY OF DATA DICTIONARIES

There are several commercially available DDS packages in the DDS marketplace. Most of them have introduced by the DBMS software vendors. In this survey, the characteristics of the following DDSs will be explained :

1. DB/DC Data Dictionary(IBM).
2. Datamanager (MSP,Inc.).
3. Integrated Data Dictionary (IDD) (Cullinet Software, Inc.).
4. Extended Data Dictionary (XDD) (Intel Systems Corporation).
5. Datadictionary (Applied Data Research).
6. UCC Ten (University Computing Company).
7. Data Control System (DCS) (Cincom Systems, Inc.).

The following information has been ontained from " Information Resource/ Data Dictionary Systems - Henry C. Lefkovits, Edgar H. Sibley, Sandra L. Lefkovits, 1983, QED Information Sciences Inc., Wellesley, Massachusetts 02181 ". The more information about these dictionary systems can be obtained from this reference. [Ref. 3]

1. DB/DC Data Dictionary

Vendor	: International Business Machines (IBM)
Hardware	: IBM 360, 370, 30xx, 43xx
Source language	: Assembler language.
Dependent DBMS	: IMS or DOS PL/I.
Entity names	: Database, Segment, Element, System,

		Job, Program, Module, Transaction, PSB, PCB, SYSDEF.
Extensibility	:	New entity-types, relationship-types, and attribute-types can be defined. Dictionary schema has Extensibility Control Information entity-types : CATEGORY, RELTYPE, and ATTRTYPE.
Maintenance	:	Three different maintenance ways : 1. Keyword driven commands. 2. 3270 Interactive Forms. 3. The Batch Forms Input facility.
Reports and queries	:	Reporting commands : REPORT, SCAN, PUNCH. Reports : 1. Entity-Specific reports. 2. Display form equivalent reports. 3. Indirect entity reference reports. 4. Glossary reports. 5. GUIDE reports.
Status facility	:	Every entity have a status which is being expressed by a status code. Status codes : Test, Production, Installed and user-defined.
Security facility	:	Access characteristics can be assigned as follows : 1. Status. 2. Entity-type. 3. SIGN-ON command. DDUSER entity-type is used to establish authorized users.
Bridge facility	:	DBD-IN, PSB-IN, COBOL-IN, PLI-IN commands are used for reading these descriptions and creating dictionary

entities and relationships corresponding to them.

2. Datamanager

Vendor	: Management Systems and Programming.
Hardware	: IBM 360, 370, 30xx, 43xx, and plug compatible machines.
Source language	: Assembler language.
Dependent DBMS	: Independent.
Entity-names	: File, Group, Item, System, Program, Module.
Extensibility:	: By using User Defined Syntax facility user can define extensibility entity-types and attribute types.
Maintenance	: A number of commands are available for adding new entities, modifying and deleting them. These commands can be executed either on-line or in batch mode. Also there are commands for manipulating definitions of entities.
Reports and queries	: The report commands : Print, List, Report, Glossary, Bulk Print, Bulk Report, Switch, Skip, Space, Text. The query commands : What, Which, Whose, Who, Does, Show. Both sets of commands can be used in both batch and interactive modes and they contain facilities for selecting categories of entities.
Status facility	: The dictionary administrator can define up to 256 statuses, each one of which has a name. There exist two types of statuses : non-frozen and frozen.

Security facility	: For entering the system a password is supplied by the AUTHORITY command. Individual entities can be assigned levels of protection by the PROTECT command. A user can also be assigned a specific security level. Also, the dictionary itself can be assigned an insertion security level and protection security level.
Bridge facility	: Three bridge facilities available : 1. The User Interface facility. 2. The Source Language Generation Facility. 3. Interfaces to DBMSs and the MARK IV File Management System.

3. <u>Integrated Data Dictionary</u>

Vendor	: Cullinet Software, Inc.
Hardware	: IBM 360, 370, 30xx, 43xx.
Source language	: Assembler language.
Dependent DBMS	: IDMS.
Extensibility	: System supports a full range of schema extensibility features that allow to perceive and use additional entity-types, relationship-types, and attribute-types. There are two mechanisms which are used for extensibility : 1. The CLASS/ATTRIBUTE declaration. 2. The definition of relational keys.
Maintenance	: SET OPTIONS command is used for control of the default processing options. DDDL statements may be

		executed either in a batch or on-line. For on-line usage an option exists for either full screen or line mode entry of statements. Three main maintenance commands are : ADD, MODIFY, DELETE.
Reports and queries	:	Four different ways for data retrieving : 1. Standard reports of the DDR (Dictionary/Directory Reporter). 2. Facilities of the CULPRIT system. 3. Using DISPLAY/PUNCH command. 4. Using OLQ, accessing QFILEs. The specific DDR reports : 1. Detail reports. 2. Key reports. 3. Summary reports. 4. Cross-reference reports. 5. Special purpose reports.
Status facility	:	VERSION mechanism is the major way to provide different environments for development, test, etc. By using VERSION clause, a number appended to the entity name.
Security facility	:	This facility consists of both global and local mechanism. The dictionary administrator can establish security for the entity-types and the following functionality : 1. CLASS and ATTRIBUTE security. 2. LOAD MODULE security. 3. IDMS security. 4. IDMS-DC security. 5. IDD security.

	6. OLQ security.
	7. CULPRIT security.
Bridge facility	: There exist bridges from IDMS-DB/DC to IDD as well as in the other direction, from IDD to IDMS-DB/DC.

4. Datadictionary

Vendor	: Applied Data Research (ADR).
Hardware	: IBM 360, 370, 30xx, 43xx.
Source language	: Assembler language.
Dependent DBMS	: DATACOM/DB.
Entity-names	: Element, Key, Field, Record, File, Report, Area, Database, Dataview, System, Program, Module, Job, Step, Library, Member, Node, Authorization, Panel, Person.
Extensibility	: New entity-types, relationship-types, and attribute-types can be introduced
Maintenance	: Two primary maintenance ways : 1. On-line maintenance facility. 2. Batch execution transactions. The Input Creation Facility analyzes COBOL record descriptions and creates corresponding entities.
Reports and queries	: System has two facilities to extract data from the dictionary : 1. The Batch Reporting facility. 2. The On-line Maintenance facility.
Status facility	: Two status mechanisms : 1. An entity may be assigned a version number. 2. Every entity has an attribute of type STATUS.
Security facility	: Two security mechanisms :

	1. The use of Passwords. 2. The use of Locks and Override Codes. Additionally, through the use of on-line interface a user can be assigned a password, and an authorization level.
Bridge facility	: Two bridge facilities : 1. The Service facility. 2. The Source Language Generation facility.

5. Extended Data Dictionary

Vendor	: Intel Systems Corporation.
Hardware	: IBM 360, 370, 30xx,43xx.
Source language	: Assembler language.
Dependent DBMS	: System 2000.
Entity names	: Data Base, File, Work Area, Schema, Record, Subschema Record, File Record, Work Structure, Item, User Application, Work Unit, Program.
Extensibility	: New entity-types, attribute-types, or relationship-types can be defined. The master password is required for this process.
Maintenance	: Three maintenance ways : 1. Use of the XDD update and utility strings. 2. Use of SCF (Self-Contained Facility). 3. Use of QueX facility of System 2000.
Reports and queries	: The Report Generation Procedures provide five reports : CAT, DES,

	EXC, EXS, and EXI. Also, the PRODUCE command provides explosion and impact reports.
Status facility	: The status facility consists of the use of multiple versions for entities of all types. Every version of an entity has a unique status. Multiple versions can exist which have the same status.
Security facility	: A Master Password can be selected. This password allows secondary passwords to be assigned to the users. Each such password has associated authorities that control dictionary operations.
Bridge facility	: Two bridge facilities : 1. The dictionary mat be preloaded using COBOL and COBOL PLEX program Data Collection facilities to extract meta data from the COBOL or COBOL PLEX programs. 2. The XDD COBOL Generation Bridge may be used to distribute structures and logical views to COBOL or COBOL PLEX programs.

6. UCC Ten

Vendor	: University Computing Company.
Hardware	: IBM 360, 370, 30xx, 43xx.
Source language	: 90 % COBOL, 10 % Assembler language.
Dependent DBMS	: IMS HIDAM databases.
Entity names	: Field, List, Segment, Data, Group, Set, File/Data Base, Transaction, Module, Program, Job Application,

	Program Specification Block (PSB), ID, and 23 more communication oriented, message oriented, format oriented and ADF oriented entity-types.
Extensibility	: None.
Maintenance	: Three maintenance interfaces : 1. Transactions that are submitted in one of the following modes : On-line at terminal, On-line Queue, Batch Queue, Batch DBA. 2. Input to the dictionary using preformatted screens via 3270 terminals. 3. Fixed format input for adding entities and certain relationships.
Reports and queries	: Three types of reports : 1. Entity reports. 2. Text reports. 3. Keyword reports.
Status facility	: System provides Text and Production status with 255 sides.
Security facility	: System contains a security user exit. This exit is used for the password protection of the DSTRUCTURE and DABSOLUTE commands. IMS security facilities are also available.
Bridge facility	: System contains facilities whereby the contents of the dictionary can be used to generate source statements that can be used by other software processors. These actions can be invoked through the GENERATE transaction.

7. Data Control System (DCS)

Vendor	: Cincom Systems, Inc.
Hardware	: IBM 370, 30xx, 43xx.
Source language	: COBOL and MANTIS.
Entity names	: Element, File, Database, Report, Source Document, Transaction, User Application System, Program.
Extensibility	: None.
Maintenance	: Entities and relationships are added, modified, and deleted by the use of predefined screens. The types of screens are : 1. Facility Selection Menu. 2. Entity Screens. 3. Relationship Screens. 4. Table Definition Screens. 5. Mini Menus.
Reports and queries	: System provides two facilities : 1. The Interactive Screen Interface which can be used to display information about entities and relationships. 2. The Batch Reporting facility which can be used to produce preformatted reports.
Status facility	: None.
Security facility	: System contains a special identification of a user called the Master User. This Master User assigns passwords to other users.
Bridge facility	: The DCS Generation Facility which consists of CSIDBI01 and CSIDBI02 programs, can be used to control database access to TOTAL databases.

III. RELATIONAL DICTIONARY MODEL

A. THE RELATIONAL MODEL

The relational model views a logical data base as a collection of tables. These tables are two-dimensional and are called relations. Relations contain single-valued entries but no repeating groups or arrays. The columns of a relation are called attributes, and the rows are called tuples. Each column contains the same kind of data (e.g.:dates), but the entries in rows are not identical. The rows and columns can be ordered in any sequence without affecting the information content. [Ref. 4]

The logical relationships are inherent in the data with the relational model. Two tuples can have a relationship if they have two attributes that arise from the same domain. Users can access and combine data using data values.

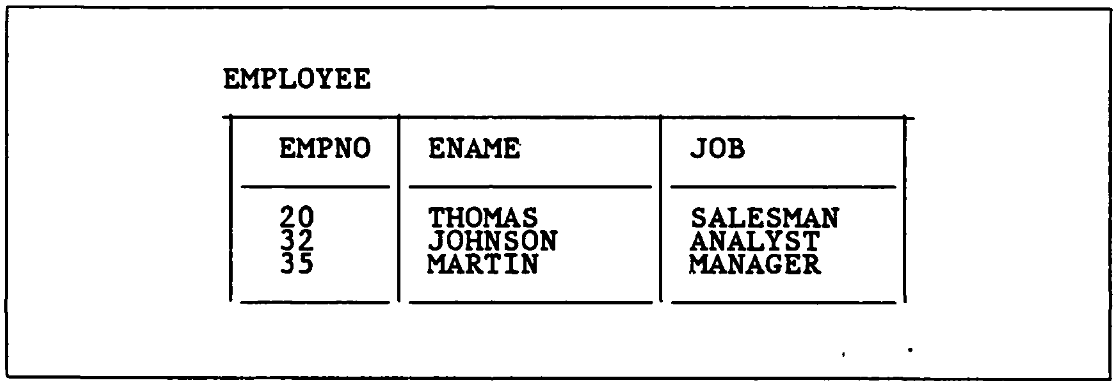
The relational data base approach provides many advantages in ease of use and simplicity, data independence, user friendliness, flexibility, data base processing power, and security controls. Also, it provides a good theoretical foundation grounded in the mathematical theory of relations.

The relations are easy to understand by users. Relationships between relations are easily expressed. With relations, a high degree of data independence can be achieved. A wide variety of relations can be derived easily by using algebraic operations to satisfy different user needs. By using these algebraic operations, users can be constrained to specific instances of relations and attributes.

B. OVERVIEW OF ORACLE SYSTEM

The ORACLE relational data base management system is a computer program that manages data. Users access data via the SQL nonprocedural data sublanguage which is a structured query language with English keywords.

An ORACLE data base consists of tables which in turn consist of columns and rows. A row is made up of fields which contain data values. An example of a table is given in Fig. 3.1 .


EMPLOYEE

EMPNO	ENAME	JOB
20	THOMAS	SALESMAN
32	JOHNSON	ANALYST
35	MARTIN	MANAGER

Figure 3.1 An ORACLE Table.

A user can create tables via the SQL Data Definition Language commands. The CREATE TABLE command is used to create a new table. The column names and the data types of the columns are specified for each table created. The DROP TABLE statement deletes a table from the data base schema.

After a table is created, data can be entered into the table via an INSERT command. Users can remove a row from a table using the DELETE command. The UPDATE command allows a user to modify a field in a row of a table.

The most common operation in ORACLE is to retrieve data from tables by means of queries. The SELECT command is used for this purpose. By using this command, we can select all the columns or specific columns from a table. We can also control the order columns are displayed, and prevent the selection of duplicate rows. The syntax of SELECT command as following :

```
SELECT  some columns
FROM    some tables
WHERE   certain conditions are met.
```

SQL provides a powerful join operator as part of the SELECT command. Two or more tables can be merged on the basis of common fields, resulting in a single table. We can list the tables to be joined in the FROM clause and the relationships between the tables in the WHERE clause. [Ref. 5]

C. A RELATIONAL DATA DICTIONARY MODEL

A relational data dictionary (RDD) model was developed and implemented using ORACLE. This RDD system runs on VAX-VMS 11/780 computer.

The National Bureau of Standards (NBS) has developed dictionary standards which capture the common features of systems such as extensibility, maintenance, and report processing. The dictionary system-standard schema has specific entity-types, relationship-types, and attribute types which are developed by the NBS. The system-standard schema constitutes the core of the logical structure of this dictionary. The entity-types, attribute-types, and relationship-types are as shown in Tables I, II, and III.

1. <u>Description of Entity-types</u>

1. USER, describes a person or an organization that uses the DDS.
2. SYSTEM, describes a collection of programs and/or modules associated with a major function of the enterprise.

TABLE I

ENTITY-TYPES OF NBS SYSTEM-STANDARD SCHEMA

USER
SYSTEM
PROGRAM
MODULE
FILE
DOCUMENT
RECORD
ELEMENT
BIT-STRING
CHARACTER-STRING
FIXED-POINT
FLOAT

TABLE II

RELATIONSHIP-TYPES OF NBS SYSTEM-STANDARD SCHEMA

CONTAINS
PROCESSES
RESPONSIBLE_FOR
RUNS
GOES_TO
DERIVED_FROM
CALLS
REPRESENTED_AS
STANDARD_FOR
HAS_SORT_KEY
HAS_ACCESS_KEY

3. PROGRAM, represents information about a collection of executable code.
4. MODULE, describes the parts of programs which are logically associated with each other.
5. FILE, describes collections of records.
6. DOCUMENT, describes instances of data to document to the user.
7. RECORD, describes instances of logically associated data.

TABLE III

ATTRIBUTE-TYPES OF NBS SYSTEM-STANDARD SCHEMA

ADDED_BY
ALLOWABLE_RANGE
ALLOWABLE_VALUE
CLASSIFICATION
CODE_LIST_LOCATION
COMMENTS
DATA_CLASS
DESCRIPTION
DURATION_TYPE
DURATION_VALUE
ENTITY_NAME
ENTITY_TYPE
LAST_MODIFICATION_DATE
LAST_MODIFIED_BY
LOCATION
NUMBER_OF_LINES_OF_CODE
NUMBER_OF_MODIFICATIONS
NUMBER_OF_RECORDS
RECORD_CATEGORY
SECURITY

8. ELEMENT, describes an instance of data.
9. BIT_STRING, describes a string of binary codes.
10. CHARACTER_STRING, describes a string of characters.
11. FIXED_POINT, describes the representation of numeric values.
12. FLOAT, describes the representation of approximate numeric values.

2. Description of Attribute-types

1. ADDED_BY, describes the person who inserts data into a relation.
2. ALLOWABLE_RANGE, describes the allowable range of a data element.
3. ALLOWABLE_VALUE, describes the allowable value for a data element.
4. CLASSIFICATION, describes the area of responsibility or interest of an entity.
5. CODE_LIST_LOCATION, describes the hardware location of

codes of a program or module.

6. COMMENTS, gives information about the characteristics of an entity.
7. DATA_CLASS, describes the class of a data element.
8. DATE_ADDED, describes the insertion date of a data element into the data base.
9. DURATION_TYPE, describes the type of duration of a process.
10. DURATION_VALUE, describes the duration value required for a process.
11. ENTITY_NAME, represents the name of an entity in the data base.
12. ENTITY_TYPE, describes the type of an entity in the data base.
13. LAST_MODIFICATION_DATE, describes the last modification date of a data element in the data base.
14. LAST_MODIFIED_BY, describes the user who makes the last modification to a data element.
15. LOCATION, describes the hardware location of data in the data base.
16. NUMBER_OF_LINES_OF_CODE, represents the number of codes of a program or a module.
17. NUMBER_OF_MODIFICATIONS, represents the number of modifications of a data element.
18. NUMBER_OF_RECORDS, represents the number of records of a file.
19. RECORD_CATEGORY, describes the category of a record in a file.
20. SECURITY, describes the security class of an entity for explaining the authority level of a user to use it.

3. Description of Relationship-types

1. CONTAINS, describes a relation where an entity-type contains other entity-types.
2. PROCESSES, describes a relation where an entity-type processes other entity-type.
3. RESPONSIBLE_FOR, describes the responsibility of a user to process a DDS element.
4. RUNS, describes an association between user and system elements.
5. GOES_TO, describes a relation where a process transfers control to another one.
6. DERIVED_FROM, describes a relation where an entity is derived from another one.
7. CALLS, describes a relation where an entity calls another one.
8. REPRESENTED_AS, describes the entities that document a data element.
9. STANDARD_FOR, describes the standard elements used to describe an element.
10. HAS_SORT_KEY, describes the sort key element of a file.
11. HAS_ACCESS_KEY, describes the access key element of a file.

4. The Relations of Dictionary

The dictionary has several different relations. The general of these relations is as follows :

USER_X (*user_name,* description, classification, date_ added,added_by, last_modification_date, last_ modified_by, number_of_modifications, location, comments, security)

SYSTEM (*system_name,* description, classification, date_ added, added_by, last_modification_date, last_ modified_by, number_of_modifications, location, duration_value, duration_type, comments, security)

PROGRAM (*program_name,* description, number_of_lines_of_ code, classification, date_added, added_by, last_ modification_date, last_modified_by, number_of_ modifications, location, duration_value, duration _type, comments, security)

MODULE (*module_name,* description, classification, date_ added, added_by, last_modification_date, last_ modified_by, location, number_of_lines_of_code, number_of_modifications, comments, security)

FILE_X (*file_name,* description, classification, date_ added, added_by, last_modification_date, last_ modified_by, location, number_of_modifications, number_of_records, comments, security)

DOCUMENT (*document_name,* description, classification, date_added, added_by, last_modification_date, last_modified_by, location, number_of_ modifications, comments, security)

RECORD (*record_name,* description, classification, date_ added, added_by, last_modification_date, last_ modified_by, number_of_modifications, record_ category, comments, security)

ELEMENT (*element_name,* description, classification, date_ added, added_by, last_modification_date, last_ modified_by, number_of_modifications, allowable_ range, allowable_value, comments, code_list_ location, data_class, security)

CONTAINS (*entity_name1, entity_type1, entity_name2, entity _type2*)

PROCESSES (*entity_name1, entity_type1,entity_name2, entity_type2*)

RESPONSIBLE_FOR (*entity_name1, entity_type1, entity_name2, entity_type2*)

RUNS (*entity_name1, entity_type1, entity_name2, entity _type2*)

GOES_TO (*entity_name1, entity_type1, entity_name2, entity_type2*)

DERIVED_FROM (*entity_name1, entity_type1, entity_name2, entity_type2*)

CALLS (*entity_name1, entity_type1, entity_name2, entity_type2*)

REPRESENTED_AS (*entity_name1, entity_type1, entity_name2 entity_type2*)

STANDARD_FOR (*entity_name1, entity_type1, entity_name2, entity_type2*)

HAS_ACCESS_KEY (*entity-name1, entity_type1, entity_name2, entity_type2*)

HAS_SORT_KEY (*entity_name1, entity_type1,entity_name2, entity_type2*)

There are two special relations in the dictionary schema : ALIAS and CATEGORY. The ALIAS relation is used to record synonyms. Synonyms are two or more names for the same data item. In an enterprise every department can use different names for the same data item in the data base. In this case, the synonyms are recorded as aliases. The ALIAS relation in the data dictionary is defined as :

ALIAS (*entity_name, entity_type, alias_name*)

The CATEGORY relationship provides a key word in context (KWIC) capability which allows different entities to be arbitrarily categorized by user-defined terms. For example it may be desirable to classify certain files, programs, reports, users, etc. as being PERSONNEL-related. This can be done via CATEGORY by associating each such entity with the PERSONNEL category. The format of CATEGORY relation is defined as :

CATEGORY (*entity_name, entity_type, category_name*)

The dictionary model has a specific relationship by which we can represent the dictionary entities and the specific relationships in which they participate. This relationship makes the dictionary model self-descriptive. Thus, it can describe its schema structure. The format of this relationship as following :

RELATIONSHIP (*entity_name1, entity_type1, entity_name2, entity_type2,* relation)

Another type of relationship of the dictionary model is ENTITY by which we represent the entity_types such as 'SYSTEM', 'FILE', etc. . The format of this relationship as following :

ENTITY (*entity_name,* description, classification, date_ added, added_by, last_modification_date, last_ modified_by, number_of_modifications, location, comments, security)

D. RELATIONSHIPS BETWEEN ENTITY-TYPES

The pairs of entity-types belong to a specific relationship are as shown in Table IV (entity-type1 and entity-type2 are both assumed to be 'ENTITY' and are omitted for the sake of clarity).

TABLE IV

RELATIONSHIPS BETWEEN ENTITY-TYPES

CONTAINS :

SYSTEM,	SYSTEM
SYSTEM,	PROGRAM
SYSTEM,	MODULE
PROGRAM,	PROGRAM
PROGRAM,	MODULE
MODULE,	MODULE
FILE,	FILE
FILE,	DOCUMENT
FILE,	RECORD
FILE,	ELEMENT
DOCUMENT,	DOCUMENT
DOCUMENT,	RECORD
DOCUMENT,	ELEMENT
RECORD,	RECORD
RECORD,	ELEMENT
ELEMENT,	ELEMENT

RESPONSIBLE_FOR :

USER,	FILE
USER,	DOCUMENT
USER,	RECORD
USER,	ELEMENT
SYSTEM,	FILE
SYSTEM,	DOCUMENT
SYSTEM,	RECORD
SYSTEM,	ELEMENT
PROGRAM,	FILE
PROGRAM,	DOCUMENT
PROGRAM,	RECORD
PROGRAM,	ELEMENT
MODULE,	FILE
MODULE,	DOCUMENT
MODULE,	RECORD
MODULE,	ELEMENT

CALLS :

PROGRAM,	PROGRAM
PROGRAM,	MODULE
MODULE,	MODULE

STANDARD_FOR :

ELEMENT,	ELEMENT

HAS_SORT_KEY &
HAS_ACCESS_KEY :

FILE,	ELEMENT

PROCESSES :

USER,	FILE
USER,	DOCUMENT
USER,	RECORD
USER,	ELEMENT
SYSTEM,	FILE
SYSTEM,	DOCUMENT
SYSTEM,	RECORD
SYSTEM,	ELEMENT
PROGRAM,	FILE
PROGRAM,	DOCUMENT
PROGRAM,	RECORD
PROGRAM,	ELEMENT
MODULE,	FILE
MODULE,	DOCUMENT
MODULE,	RECORD
MODULE,	ELEMENT

RUNS :

USER,	SYSTEM
USER,	PROGRAM
USER,	MODULE

GOES_TO :

SYSTEM,	SYSTEM
PROGRAM,	PROGRAM
MODULE,	MODULE

DERIVED_FROM :

DOCUMENT,	FILE
DOCUMENT,	DOCUMENT
DOCUMENT,	RECORD
ELEMENT,	FILE
ELEMENT,	DOCUMENT
ELEMENT,	RECORD
ELEMENT,	ELEMENT
FILE,	DOCUMENT
FILE,	FILE
RECORD,	DOCUMENT

REPRESENTED_AS :

ELEMENT,	BIT STRING
ELEMENT,	CHARACTER STRING
ELEMENT,	FIXED POINT
ELEMENT,	FLOAT

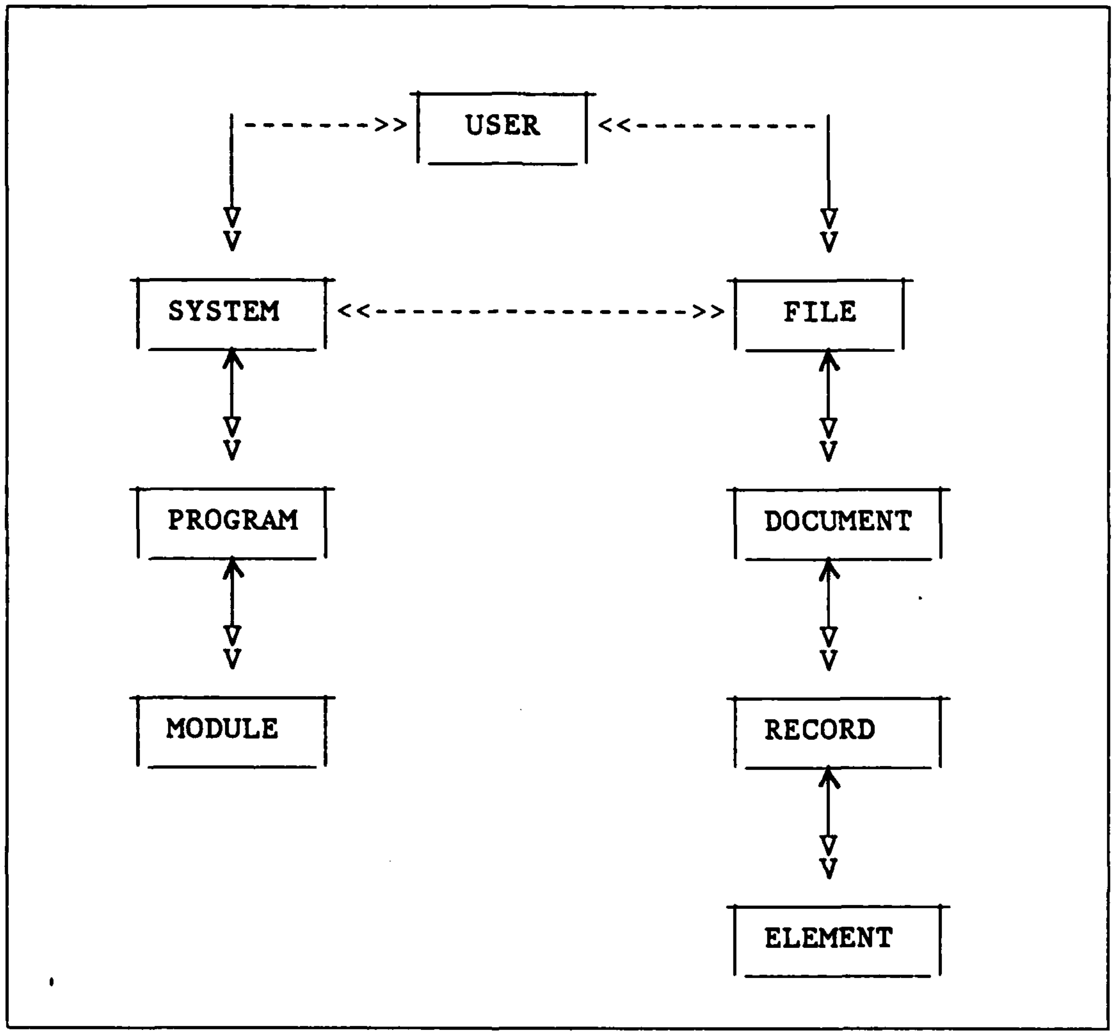


Figure 3.2 Bachman Diagram of entity_types.

The relationships between the entity-types are shown in Fig. 3.2 . In this diagram, we denote the relationship one-to-one by a single- headed arrow (--->), and the relationship one-to-many by a double-headed arrow (--->>).

The implementation of data dictionary model using ORACLE has some shortcomings. We can represent entity-types, and relationships between these entities easily. But, we also need rules about data.

ORACLE implementation is not capable of defining rules. We can only implement immediate data with ORACLE.

E. EXAMPLES OF QUERIES

By using SQL commands, we can represent several different queries. These queries are two types : queries concerning meta-entities and those concerning instances of entities. There are several advantages of having the meta-entity information. The quality of metadata should be monitored by defining and inserting integrity checks. We can do these checking by means of queries concerning meta-entities. These queries also give information about the system-standard schema of the dictionary system. That is, we can describe the entity-types, attribute-types, and relationship-types of the dictionary.

The listing of tuples in the data base is given in Appendix B. The queries in this section are related with these values. Suppose we have a query " Which systems contain ACC5PROG program ? ". The implementation of this query and the answer to this query is given in Fig. 3.3 .

```
UFI> SELECT ENTITY_NAME1
  2  FROM CONTAINS_X
  3  WHERE ENTITY_NAME2='ACC5PROG' AND
  4  ENTITY_TYPE1='SYSTEM';

ENTITY_NAME1
---------------
ACCOUNT-2
ACCOUNT-5
```

Figure 3.3 An Example of Query.

Other types of query examples follow :

Query : " Who is responsible for ACCOUNT-2 system ? " . The implementation of this query is given in Fig. 3.4 .

```
UFI> SELECT ENTITY_NAME1
  2  FROM RESPONSIBLE_FOR
  3  WHERE ENTITY_NAME2='ACCOUNT-2';

ENTITY_NAME1
--------------
JONES H.B.
ALLEN G.M.
SCOTT T.L.
```

Figure 3.4   An Example of Query.

Query : " Which programs process PAYROLL5 record ? " . The implementation of this query is given in Fig. 3.5 .

```
UFI> SELECT ENTITY_NAME1
  2  FROM PROCESSES
  3  WHERE ENTITY_NAME2='PAYROLL5' AND
  4  ENTITY_TYPE1='PROGRAM';

ENTITY_NAME1
--------------
ACCOUNT-2
ACCOUNT-3
ACCOUNT-4
```

Figure 3.5   An Example of Query.

Query : " What elements are contained in the PAYROLL5 record ? " .   The implementation of this query is given in Fig. 3.6 .

Query : " What relationships does FILE participate in ? " . The implementation of this query is given in Fig. 3.7 .

```
UFI> SELECT ENTITY_NAME2
  2  FROM CONTAINS_X
  3  WHERE ENTITY_NAME1='PAYROLL5' AND
  4  ENTITY_TYPE2='ELEMENT';

ENTITY_NAME2
---------------
PRELE1
PRELE2
PRELE3
PRELE4
```

Figure 3.6 An Example of Query.

```
UFI> SELECT ENTITY_NAME1,RELATION,ENTIY_NAME2
  2  FROM RELATIONSHIP
  3  WHERE ENTITY_NAME1='FILE';

ENTITY_NAME1  RELATION          ENTITY_NAME2
------------  ----------        -------------
FILE          CONTAINS          FILE
FILE          CONTAINS          DOCUMENT
FILE          CONTAINS          RECORD
FILE          CONTAINS          ELEMENT
FILE          DERIVED_FROM      FILE
FILE          DERIVED_FROM      DOCUMENT
FILE          HAS_SORT_KEY      ELEMENT
FILE          HAS_ACCESS_KEY    ELEMENT

8 records selected.
```

Figure 3.7 An Example of Query.

Query : " What aliases ACCOUNT-2 program has ? " . The implementation of this query is given in Fig. 3.8 .

Query : " Which entities are in the CONTROL2 category ? " . The implementation of this query is given in Fig. 3.9 .

We are not able to answer some kinds of queries with this dictionary design. For example, a query " What kind of

```
UFI> SELECT ALIAS_NAME
  2  FROM ALIAS
  3  WHERE ENTITY_NAME1='ACCOUNT-2';

ALIAS_NAME
------------
ASEL
APRO
```

Figure 3.8 An Example of Query.

```
UFI> SELECT ENTITY_NAME,ENTITY_TYPE
  2  FROM CATEGORY
  3  WHERE CATEGORY_NAME='CONTROL2';

ENTITY_NAME    ENTITY_TYPE
------------   -----------
ACCOUNT-2      PROGRAM
ACC5FILE       FILE_X
ACC6FILE       FILE_X
PAYROLL5       RECORD
PAYROLL6       RECORD
PAYROLL7       RECORD

6 records selected.
```

Figure 3.9 An Example of Query.

entity is PRELE1 ? " cannot be answered easily. To answer this kind of query, we have to implement every data element as a separate entity in a special relation. This implementation causes overhead in the data base. Also a query " What entities appear in relationships but are not defined as a tuple in any of the entity relations ? " cannot be answered easily. For example PROCESSES ('ACCOUNT-2','PROGRAM','ACCREC','RECORD') is a tuple of PROCESSES relation but 'ACCREC' is not a tuple in RECORD relation. To answer this kind of query, we will have to insert data into a separate relation for every data element we want to represent in the data base.

These types of queries can be answered by means of a different design. We can define two relations:

```
ENTITY (entity_name, entity_type, attr1, attr2,
                                        ... ,attrM )

RELSHIP ( relname, entity_name1, entity_type1,
          entity_name2, entity_type2, attributes )
```

By using these relations we can answer the queries we have asked above like following :

```
SELECT  entity_type
FROM    ENTITY
WHERE   entity_mane = 'PRELE';

SELECT entity_name1
FROM   RELSHIP
WHERE  entity_name1 NOT IN
       ( SELECT entity_name FROM ENTITY );
```

This design method also has disadvantages. For example in the first example we will have null values for some attributes of the relation.

F. USER MANUAL

This user manual explains the necessary procedures to use the ORACLE system. Additional information can be obtained from the ORACLE system manuals.

After entering the VAX-VMS system, you will see the following on the screen:

```
$
```

To start ORACLE type " ORACLE" like following and press the RETURN key on the terminal :

```
$ ORACLE
```

Then type " UFI " like following and press the RETURN key :

```
$ UFI
```

After a few seconds a message will appear :

```
ORACLE Utilities, Copyright (c) 1979, 1980, 1981, 1982,
                                                   RSI
UFI Version 3.5 - on Mon Nov 18 15:39:17  1985
Connecting to ORACLE V 4.2.2 - Interim Release
Enter user-name :
Enter password  :
```

Upon correctly entering the user-name, and password you will receive the following on the screen :

```
UFI>
```

Now, you are ready to enter ORACLE commands into the system. When you want to exit the system type " EXIT " like following :

```
UFI> EXIT
```

Then you will see the following message on the screen :

```
logged off from ORACLE $
```

If you want to log off from VMS system type " log " :

```
$ log
```

Now, you have logged off the VMS system and you will see the following message :

```
logging off the VAX 780 computer
```

1. Creating A Table

A table can be created using the CREATE TABLE command. An example of this command is given in Fig. 3.10 .

```
UFI> CREATE TABLE CONTAINS_X
  2  ( ENTITY_NAME1 CHAR (15),
  3    ENTITY_TYPE1 CHAR (15),
  4    ENTITY_NAME2 CHAR (15),
  5    ENTITY_TYPE2 CHAR (15) );
 Table created.
```

Figure 3.10 Creating a Table.

2. Inserting Data Into a Table

After a table is created, rows can be entered into the table using the INSERT command. An example of this command is given in Fig. 3.11 .

```
 UFI> INSERT INTO CONTAINS_X VALUES
   2  ('ACCOUNT-2','SYSTEM','ACC5PROG','PROGRAM');
ENTITY_NAME1 ENTITY_TYPE1 ENTITY_NAME2 ENTITY_TYPE2
------------ ------------ ------------ ------------
ACCOUNT-2    SYSTEM       ACC5PROG     PROGRAM
 1 record created
```

Figure 3.11 Inserting Data Into a Table.

3. Selecting Data From a Table

The SELECT command is used to retrieve data from a table. An example of this command is given in Fig. 3.12 .

```
UFI> SELECT ENTITY_NAME1, ENTITY_NAME2
  2  FROM CONTAINS_X ;

ENTITY_NAME1   ENTITY_NAME2
------------   ------------
ACCOUNT-2      ACC5PROG
```

Figure 3.12 Selecting Data From a Table.

If we want to select all the columns we can use an asterisk (*) in place of the list of column names. An example is given in Fig. 3.13 . In this example, all the columns of CONTAINS_X table will be selected.

```
UFI> SELECT *
  2  FROM CONTAINS_X ;

ENTITY_NAME1 ENTITY_TYPE1 ENTITY_NAME2  ENTITY_TYPE2
------------ ------------ ------------  ------------
ACCOUNT-2    SYSTEM       ACC5PROG      PROGRAM
```

Figure 3.13 Selecting Data From a Table.

4. Description of the columns of a Table

The DESC command gives the brief description of the columns used in a table. The description returned will contain columns for the number of the column, the maximum size of numeric or formatted data, the type of data, and the name of the column. An example of this command will be given in Fig. 3.14 .

```
UFI> DESC CONTAINS_X ;

  #  size  csize  type                  name
  1    15      1    1 character         ENTITY_NAME1
  2    15      1    1 character         ENTITY_TYPE1
  3    15      1    1 character         ENTITY_NAME2
  4    15      1    1 character         ENTITY_TYPE2
```

Figure 3.14 Description of Columns of a Table.

IV. DATA DICTIONARIES AND EXPERT SYSTEMS

A. OVERVIEW OF EXPERT SYTEMS

Expert systems are the most significant development in the area of artificial intelligence. Expert or knowledge-based systems are computer programs which represent and apply specific knowledge to solve problems. Expert systems use knowledge that is represented in computable form.

The rule-based system paradigm is the most popular problem solving paradigm used for building expert systems. The rule-based system paradigm is built around rules. The rules cover the major situations in a domain and consist of an "if" part and a "then" part :

```
Rule(n)  If     condition 1
                condition 2
                   .
                   .
                condition n
         then   action 1
                action 2
                   .
                   .
                action  n
```

The "if" parts of the rules consist of combinations of known facts. The "then" parts specify new facts to be deduced. We use forward chaining to move from existing conditions to desired actions. Backward chaining hypothesizes a conclusion and use the rules to work backward toward the facts which lead to this conclusion. [Ref. 6]

Expert systems use different methodologies for solving problems.

Some expert systems such as XCON use synthesis oriented forward chaining. Others, such as MYCIN and PROSPECTOR use analysis oriented backward chaining.

XCON's domain concerns the configuration of computer system components. XCON knows the properties of component types for VAX computers and XCON handles orders involving these components. MYCIN aids medical doctors in diagnosing blood and meningitis infections and in recommending antibiotic drug treatment. PROSPECTOR is used by geologists in the exploration of ore deposits.

Expert systems can explain how and why they do things, and they can estimate the quality of their results. They can also demonstrate the stages of the task they have performed as well as any remaining parts to be performed.

An expert system must demonstrate efficient performance and must find effective solutions. These two factors must be traded off in certain cases. Some expert systems make good decisions but very slowly. Some decisions, on the other hand, require rapid response time, possibly at the expense of accuracy. Expert systems must be built to satisfy the particular requirements of each application domain.

B. COMPONENTS OF EXPERT SYSTEMS

The essential components of an expert system are the following :

1. Language Processor : The user and expert system communicate with each other by means of a language processor. The user enters the commands or questions into the system by using the language processor. Conversely, the information generated by the system is presented to the user via the same mechanism.
2. Blackboard : Intermediate decisions are recorded in a blackboard. Generally, blackboards record three types of decisions : plan, agenda, and solution. Plan recommends a general solution methodology to the problem.

Agenda records the actions awaiting the execution. The decisions and hypotheses about the problem are represented by solution elements.

3. Scheduler : The control of the agenda and the control of the order of the rule processing are maintained by a scheduler.
4. Interpreter : The rules contained in the knowledge base get applied to the agenda items by the interpreter.
5. Consistency enforcer : When new data are introduced, the consistency enforcer adjusts the previous solutions to the new data base.
6. Justifier : By using general types of question/ answering plans, the justifier explains the system's behaviour to the user.
7. Knowledge base : The facts and information about the problem and problem solving rules are recorded in the knowledge base. [Ref. 7]

The components of expert systems are given in Fig. 4.1 .

C. EXPERT SYSTEMS AND CONVENTIONAL DATA PROCESSING SYSTEMS

There are many ways in which expert systems differ from both data processing systems and other AI systems.

AI systems involve several features such as symbolic representation, symbolic inference, and heuristic search. AI systems use one of several formal approaches developed for these features. For example, one way to show what a set of antecedent-consequent rules can do is, to draw a network showing how the facts that are the consequents of one rule serve as antecedents to the next. This network is called an inference net in AI systems.

Expert systems perform their tasks in decision making environments. They solve problems in narrow and specialized domains. In contrast, the other AI systems use more general methods.

Expert systems contain self-knowledge, that is knowledge about its own structure and operation. By using

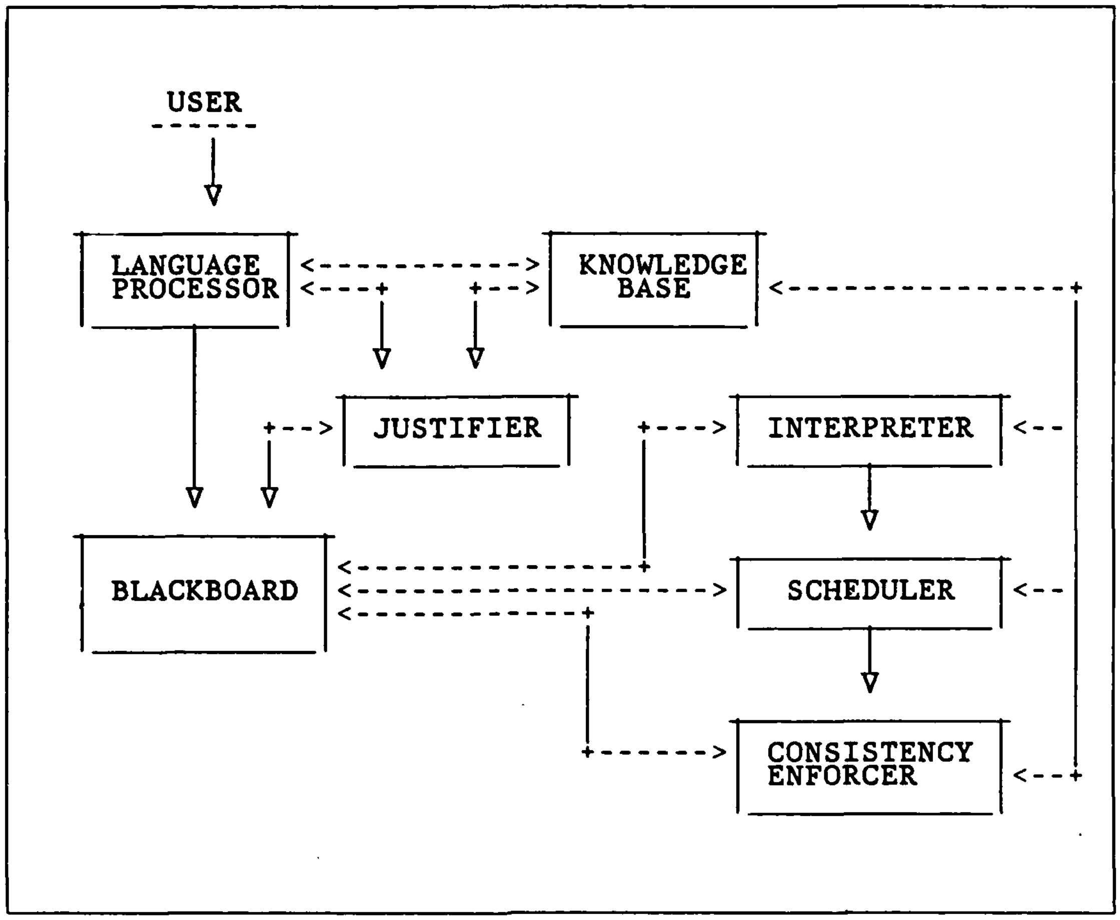


Figure 4.1 Components of an Expert System.

self-knowledge, expert systems provide explanations and justifications about their conclusions. This knowledge is also used for modification and reorganization of the system.

Expert systems solve problems in several areas which can be categorized as follows :

1. Interpretation systems : signal interpretation, speech understanding chemical structure elucidation.
2. Prediction system : weather forecasting, crop estimation.
3. Diagnosis systems : medical, electronic, software diagnosis.
4. Design systems : building design, budgeting.

5. Planning systems : robot, project, communication, military planning problems.
6. Monitoring systems : nuclear power plant, air traffic, disease, fiscal management tasks.
7. Debugging systems : computer aided debugging systems.
8. Repair systems : automotive, network, avionic systems.
9. Instruction systems : Diagnose of students behaviors.
10. Control systems : air traffic control,mission control, business management.

D. KNOWLEDGE REPRESENTATION

There are many different kinds of knowledge. Basically, knowledge can be represented by facts and procedures. Facts are things that are true about the world, and correspond to the meanings of nouns and adjectives. Procedures are sequences of actions that do things and correspond to the meanings of verbs. There are many different ways of representing and manipulating knowledge by computer.

1. Predicate Calculus

Predicate calculus is one of the widely used forms of knowledge representation. The syntax of symbols representing knowledge consists of terms and predicate symbols. In predicate calculus logical connections between entities and functions can be easily represented. Predicate calculus can also express sentences involving universal quantifiers and existential quantifiers. In predicate calculus, new symbol structures can be created from old ones by using rules of inference. Predicate calculus can express the sentence "All parts are large " as

```
( ALL (x) ( ( IS_A  x  PART ) ----> ( LARGE  x ) )  )
```

2. Semantic Networks

Many knowledge representations are built around some form of semantic net. The syntax of a semantic net consists of objects and relationships between pairs of objects. In semantic nets, the objects are represented by labeled circles and the relations are represented by labeled arrows. The semantic nets have a restriction in that they only work well for predicates of two arguments.

3. Control Structures

1. Unordered Control Structures : In a rule based system a set of antecedent-consequent rules can be represented by a network. By using AND/OR/NOT trees this network can be represented. Facts are then input to this system. An AND/OR/NOT tree reaches from base facts at the bottom, through antecedent-consequent rules, to possible conclusion at the top.

2. Backwards Chaining Control Structures : This structure imposes a single sequential ordering on everything that happens. Backwards chaining starts with a hypothesized conclusion and uses rules to work backward toward the facts that support the hypothesis. Backwards chaining works well whenever there are many more facts than goals.

3. Forward Chaining Control Structures : In some systems, there are many possible conclusions but just a few facts. For these situations forward chaining is used by starting with the facts and reasoning to conclusions.

E. METAKNOWLEDGE

Metaknowledge can be very important to building, running, and modifying expert systems. Performance can be improved by supplying various sets of metaknowledge that is knowledge about the knowledge in the system.

Metaknowledge guides the location and selection of rules. It records needed facts about knowledge. Metaknowledge enhances the system's explanation abilities by justifying rules.

It facilitates the entry of new terms, facts, and heuristic rules.

Information resource management is a potential domain for the implementation of expert systems. Data dictionary systems are currently used for representing metaknowledge about organizational information resources. Especially, extensibility features of data dictionary systems make it easy to define new data types and relationships to represent metaknowledge.

Information resource management data contains information necessary to manage and control the data. This type of data includes rules for performing its function. The rules are the functions to be performed by the system relative to data. The rules are very important in information resource management, because, they insure effective management control of data. These rules guide the data base activities and provide information about data.

We can store facts about information resources using data dictionary systems, but current data dictionary systems are unable to accommodate rules. Logic-oriented language Prolog can be used to implement these rules. Prolog allows user to define rules which are more compact than a list of facts.

F. KNOWLEDGE REPRESENTATION IN PROLOG

The declarative, logic-based language Prolog is used for solving problems that involve objects and relationships between objects. The Prolog programmer asks what formal relationships and objects occur in the problem, and what relationships are true about the desired solution.

Programming in Prolog consists of declaring some facts about objects and their relationships; defining some rules about objects and their relationships and then asking questions about objects and their relationships subject to these rules. [Ref. 8]

An explanation of an implementation of facts, rules, and a query will be explained with the following example. Suppose we have relations :

```
male(gerry).
male(john).
female(mary).
female(cindy).
parents(gerry,betty,mike).
parents(mary,betty,mike).
brother_of(X,Y):-male(X),parents(X,M,F),
                     parents(Y,M,F).
```

Suppose we want to know if Gerry is the brother of anyone. We can ask this question in Prolog like this :

```
?-brother_of(gerry,X).
```

Prolog prints X= mary. as an answer to this question.

G. A PROLOG MODEL OF A SIMPLE DATA DICTIONARY

The system-standard schema of the dictionary is defined as a specific set of entity-types, relationship types, and attribute types. This system-standard schema satisfies the requirements of many IRDS environments. Also, this schema is a standard schema developed by the National Bureau of Standards.

Data dictionary entity-types, relationship types, and attribute types are as shown in Tables V and VI .

Using Prolog, implementation of a data dictionary with the above entity-types and attribute types can be represented as predicates. For example we can represent the SYSTEM entity-type as :

```
system(name,description,date_created,classification,
last_modified_by,number_of_programs).
```

Integrity constraints can be easily implemented by using predicates involving relationships.

TABLE V

ENTITY-TYPES AND ATTRIBUTE-TYPES OF DICTIONARY MODEL

ENTITY-TYPES :	ATTRIBUTE-TYPES :
USER	ADDED_BY
SYSTEM	CLASSIFICATION
PROGRAM	COMMENTS
MODULE	DATE_ADDED
FILE	DESCRIPTION
DOCUMENT	IDENTIFICATION_NAME
RECORD	LAST_MODIFICATION_DATE
ELEMENT	LAST_MODIFIED_BY
BIT_STRING	NUMBER_OF_MODIFICATIONS
CHARACTER_STRING	NUMBER_OF_RECORDS
FIXED_POINT	NUMBER_OF_CATEGORY
FLOAT	NUMBER_OF_PROGRAMS
	DATE_CREATED
	LOCATION
	STANDARD_FOR
	HAS_SORT_KEY
	HAS_ACCESS_KEY

TABLE VI

RELATIONSHIP-TYPES OF DICTIONARY MODEL

RELATIONSHIP-TYPES :
CONTAINS
PROCESSES
RUNS
RESPONSIBLE_FOR
GOES_TO
DERIVED_FROM
CALLS
REPRESENTED_AS

The general format of these predicates are represented as :

```
relation(entityname1,entitytype1,entityname2,entitytype2).
```

We can represent "contains" relation in Prolog as :

```
contains(system_x,system_t,program_x,program_t).
contains(program_x,program_t,module_x,module_t).
contains(module_x,module_t,record_x,record_t).
contains(record_x,record_t,element_x,element_t).
```

The other types of relationships can be represented as followings :

```
processes(system_x,system_t,file_x,file_t).
responsible_for(user_x,user_t,system_x,system_t).
runs(user_x,user_t,system_x,system_t).
goes_to(system_x,system_t,system_x,system_t).
derived_from(document_x,document_t,file_x,file_t).
calls(program_x,program_t,module_x,module_t).
represented_as(element_x,element_t,bit_string_x,
                                   bit_string_t).
standard_for(element_x,element_t,element_x,element_t).
has_sort_key(file_x,file_t,element_x,element_t).
has_access_key(file_x,file_t,key_x,key_t).
```

In Prolog, rules are used when we want to say that a fact depends on a group of other facts. A rule is a general statement about objects and their relationships.

We can represent queries and information about the data base by using Prolog rules.

A relation in the data base can be derived from a rule in Prolog. By using the relationship predicates we can build rules as followings :

```
contains(X,XX,Z,ZZ):-contains(X,XX,Y,YY),
                                  contains(Y,YY,Z,ZZ).
processes(X,XX,Z,ZZ):-contains(X,XX,Y,YY),
                                  processes(Y,YY,Z,ZZ).
calls(X,XX,Z,ZZ):-contains(X,XX,Y,YY),calls(Y,YY,Z,ZZ).
```

Implementation of queries in Prolog will be explained with the following examples:

Suppose we have following facts and rules about Data Dictionary :

```
contains(system_1,system_t,program_a,program_t).
contains(system_1,system_t,program_b,program_t).
contains(system_1,system_t,program_c,program_t).
contains(program_1,program_t,module_a,module_t).
contains(program_1,program_t,module_b,module_t).
contains(program_1,program_t,module_c,module_t).
contains(program_1,program_t,module_d,module_t).
contains(file_1,file_t,document_1,document_t).
contains(file_1,file_t,document_2,document_t).
contains(file_1,file_t,document_3.document_t).

processes(user_1,user_t,file_1,file_t).
processes(user_1,user_t,file_2,file_t).
processes(user_2,user_t,file_1,file_t).
processes(user_2,user_t,file_2.file_t).

responsible_for(user_1,user_t,system_1,system_t).
responsible_for(user_1,user_t,system_2,system_t).
responsible_for(user_1,user_t,system_3.system_t).
responsible_for(user_2,user_t,system_1,system_t).
responsible_for(user_2,user_t,system_2,system_t).
```

Suppose we have a query : "Which systems contain program_a ?" The implementation of this query and the answer to this query will be as following :

```
?- contains(X,system_t,program_a,program_t).
X= system_1;
X= system_2;
no
```

The other type of query examples are the following :

```
?- contains(program_1,program_t,X,module_t).
X= module_a;
```

```
X= module_b;
X= module_c;
X= module_d;
no


?- processes(X,user_t,file_2,file_t).
X= user_1;
X= user_2;
no


?- responsible_for(user_1,user_t,X,system_t).
X= system_1;
X= system_2;
X= system_3;
no



?- processes(X,user_t,Y,file_t).
X= user_1, Y= file_1;
X= user_1, Y= file_2;
X= user_2, Y= file_1;
X= user_2, Y= file_2;
no
```

We can also define rules about data base and ask questions about these rules :

```
contains(file_1,file_t,record_a,record_t).
contains(file_1,file_t,record_b,record_t).
processes(system_1,system_t,file_1,file_t).
processes(X,XX,Z,ZZ):-processes(X,XX,Y,YY),
                               contains(Y,YY,Z,ZZ).
```

We can ask a question like this : "What does system_1 processes ?"

A query related with the above rule and the answer to this query will be as following :

```
?- processes(system_1,system_t,X,XX).
X= file_1, XX= file_t;
X= record_a, XX= record_t;
X= record_b, XX= record_t;
no
```

If we want to know the records which are processed by system_1, we can ask following question :

```
?- processes(system_1,system_t,X,record_t).
X= record_a;
X= record_b;
no
```

This dictionary model is self-descriptive. That is, we can represent the dictionary entities which participate in a specific relationship. The following facts are used for this purpose :

```
processes(user_x,entity_t,file_x,entity_t).
processes(user_x,entity_t,document_x,entity_t).
processes(user_x,entity_t,record_x,entity_t).
processes(user_x,entity_t,element_x,entity_t).

processes(system_x,entity_t,file_x,entity_t).
processes(system_x,entity_t,document_x,entity_t).
processes(system_x,entity_t,record_x,entity_t).
processes(system_x,entity_t,element_x,entity_t).

processes(program_x,entity_t,file_x,entity_t).
processes(program_x,entity_t,document_x,entity_t).
processes(program_x,entity_t,record_x,entity_t).
processes(program_x,entity_t,element_x,entity_t).

processes(module_x,entity_t,file_x,entity_t).
processes(module_x,entity_t,document_x,entity_t).
processes(module_x,entity_t,record_x,entity_t).
processes(module_x,entity_t,element_x,entity_t).

runs(user_x,entity_t,system_x,entity_t).
runs(user_x,entity_t,program_x,entity_t).
runs(user_x,entity_t,module_x,entity_t).
```

Now, we can ask " Which entities can participate in the 'process' relationship ? " by :

```
?- processes(X,entity_t,Y,entity_t).
```

Prolog lists all the entities which participate this relationship as :

```
X= user_x, Y= file_x;
X= user_x, Y= document_x;
X= user_x, Y= record_x;
X= user_x, Y= element_x;
X= system_x, Y= file_x;
X= system_x, Y= document_x;
X= system_x, Y= record_x;
X= system_x, Y= element_x;
X= program_x, Y= file_x;
X= program_x, Y= document_x;
X= program_x, Y= record_x;
X= program_x, Y= element_x;
X= module_x, Y= file_x;
X= module_x, Y= document_x;
X= module_x, Y= record_x;
X= module_x, Y= element_x;
no
```

As a second example, we can ask " Which entities can participate in the 'runs' relationship ? " by :

```
?- runs(X,entity_t,Y,entity_t).
X= user_x, Y= system_x;
X= user_x, Y= program_x;
X= user_x, Y= module_x;
```

Prolog rules help programmers to modularize knowledge. It's a way of creating new predicates from old predicates without specifying facts explicitly. The representation of facts could become tedious, especially if there are hundreds of facts about the same subject. By using rules, we can represent all of these facts easily since the rules are more compact than a list of facts. Thus, the rules save a great deal of data entry effort. Prolog makes it easy to represent indirect relationships. Prolog creates arbitrary data structures by means of rules which are themselves data. Prolog offers a wide variety of queries. In the structure of Prolog program there are precise representations for these queries. Prolog offers a great extensibility in declaring new facts and rules about the data base.

V. CONCLUSIONS

This thesis has explained the importance of metadata and data dictionary systems in the management and control of the enterprise's data resource. It has shown that the data dictionary system is a central repository of information which helps improve communication between system components of an enterprise.

This thesis has surveyed seven commercially available data dictionary systems. It has explained the characteristics and the capabilities of these systems. Thus, the reader can obtain information about these dictionary systems, compare them, and investigate the needed requirements for a new dictionary system.

This thesis has developed a relational data dictionary model which was implemented on the ORACLE relational database management system. This dictionary model is capable of satisfying the requirements of many IRDS environments. Although the relational model is the most popular data model and it has come to be of great practical significance, its dictionary capabilities are limited.

The ORACLE implementation of the data dictionary model is capable of representing entity-types and relationship-types between these entities. But, it is not capable of representing rules about information resource management data. Since information management data must contain rules for its operational purposes, this is a shortcoming of relational data dictionary models.

This thesis has explained the general characteristics of expert systems. It has proposed a Prolog model of a data dictionary as an expert system. Using logic-oriented language Prolog, the rules about the information resource management data can be implemented easily. This model shows that logic programming is suitable for relational database applications. Thus, the user can save a great deal of data entry effort by using rules instead of representing data explicitly. Prolog representation of data provides flexible extensibility features especially when adding new data into the database. Since Prolog is primarily a prototype tool, however, this suggests that more research needs to be done concerning the efficient implementation of rules in a relational environment.

APPENDIX A

ORACLE TABLES OF ENTITY-TYPES AND RELATIONSHIP-TYPES

```
TABLE USER X
    USER_NAME CHAR (15) NOT NULL,
    DESCRIPTION CHAR (60),
    CLASSIFICATION CHAR (10),
    DATE_ADDED CHAR (10),
    ADDED_BY CHAR (15),
    LAST_MODIFICATION_DATE DATE,
    LAST_MODIFIED_BY CHAR (15),
    NUMBER_OF_MODIFICATIONS NUMBER,
    LOCATION CHAR(15),
    COMMENTS CHAR (45),
    SECURITY CHAR (10);

TABLE SYSTEM
    SYSTEM_NAME CHAR (10) NOT NULL,
    DESCRIPTION CHAR (60),
    CLASS_FICATION CHAR (10),
    DATE_ADDED DATE,
    ADDED_BY CHAR (15),
    LAST_MODIFICATION_DATE DATE,
    LAST_MODIFIED_BY CHAR (15),
    NUMBER_OF_MODIFICATIONS NUMBER,
    LOCATION CHAR (15),
    DURATION_VALUE NUMBER,
    DURATION_TYPE CHAR (10),
    COMMENTS CHAR (45),
    SECURITY CHAR (10);

TABLE PROGRAM
    PROGRAM_NAME CHAR (10) NOT NULL,
    DESCRIPTION CHAR (60),
    NUMBER_OF_LINES_OF_CODE NUMBER,
    CLASSIFICATION CHAR (10),
    DATE_ADDED DATE,
    ADDED_BY CHAR (15),
    LAST_MODIFICATION_DATE DATE,
    LAST_MODIFIED_BY CHAR (15),
    NUMBER_OF_MODIFICATIONS NUMBER,
    LOCATION CHAR (15),
    DURATION_VALUE NUMBER,
    DURATION_TYPE CHAR (10),
    COMMENTS CHAR (45),
    SECURITY CHAR (10);

TABLE MODULE
    MODULE_NAME CHAR (10) NOT NULL,
    DESCRIPTION CHAR (60),
    CLASSIFICATION CHAR (10),
    DATE_ADDED DATE,
    ADDED_BY CHAR (15),
    LAST_MODIFICATION_DATE DATE,
    LAST_MODIFIED_BY CHAR (15),
    LOCATION CHAR (15),
    NUMBER_OF_LINES_OF_CODE NUMBER,
    NUMBER_OF_MODIFICATIONS NUMBER,
    COMMENTS CHAR (45),
    SECURITY CHAR (10);

TABLE FILE_X
    FILE_NAME CHAR (10) NOT NULL,
    DESCRIPTION CHAR (60),
    CLASSIFICATION CHAR (10),
    DATE_ADDED DATE,
    ADDED_BY CHAR (15),
    LAST_MODIFICATION_DATE DATE,
    LAST_MODIFIED_BY CHAR (15),
    LOCATION CHAR (15),
    NUMBER_OF_MODIFICATIONS NUMBER,
    NUMBER_OF_RECORDS NUMBER,
    COMMENTS CHAR (45),
    SECURITY CHAR (10);
```

<u>TABLE</u> <u>DOCUMENT</u>

```
    DOCUMENT_NAME CHAR (10) NOT NULL,
    DESCRIPTION CHAR (60),
    CLASSIFICATION CHAR (10),
    DATE_ADDED DATE,
    ADDED_BY CHAR (15),
    LAST_MODIFICATION_DATE DATE,
    LAST_MODIFIED_BY CHAR (15),
    LOCATION CHAR (15),
    NUMBER_OF_MODIFICATIONS NUMBER,
    COMMENTS CHAR (45),
    SECURITY CHAR (10);
```

<u>TABLE</u> <u>RECORD</u>

```
    RECORD_NAME CHAR (10) NOT NULL,
    DESCRIPTION CHAR (60),
    CLASSIFICATION CHAR (10),
    DATE_ADDED DATE,
    ADDED_BY CHAR (15),
    LAST_MODIFICATION_DATE DATE,
    LAST_MODIFIED_BY CHAR (15),
    NUMBER_OF_MODIFICATIONS NUMBER,
    RECORD_CATEGORY CHAR (10),
    COMMENTS CHAR (45),
    SECURITY CHAR (10);
```

<u>TABLE</u> <u>ELEMENT</u>

```
    ELEMENT_NAME CHAR (10) NOT NULL,
    DESCRIPTION CHAR (60),
    CLASSIFICATION CHAR (10),
    DATE_ADDED DATE,
    ADDED_BY CHAR (15),
    LAST_MODIFICATION_DATE DATE,
    LAST_MODIFIED_BY CHAR (15),
    NUMBER_OF_MODIFICATIONS NUMBER,
    ALLOWABLE_RANGE NUMBER,
    ALLOWABLE_VALUE NUMBER,
    COMMENTS CHAR (45),
    CODE_LIST_LOCATION CHAR (15),
    DATA_CLASS CHAR (10),
    SECURITY CHAR (10);

TABLE CONTAINS_X
    ENTITY_NAME1 CHAR (15),
    ENTITY_TYPE1 CHAR (15),
    ENTITY_NAME2 CHAR (15),
    ENTITY_TYPE2 CHAR (15);

TABLE PROCESSES
    ENTITY_NAME1 CHAR (15),
    ENTITY_TYPE1 CHAR (15),
    ENTITY_NAME2 CHAR (15),
    ENTITY_TYPE2 CHAR (15);

TABLE RESPONSIBLE_FOR
    ENTITY_NAME1 CHAR (15),
    ENTITY_TYPE1 CHAR (15),
    ENTITY_NAME2 CHAR (15),
    ENTITY_TYPE2 CHAR (15);

TABLE RUNS
    ENTITY_NAME1 CHAR (15),
    ENTITY_TYPE1 CHAR (15),
    ENTITY_NAME2 CHAR (15),
    ENTITY_TYPE2 CHAR (15);

TABLE GOES_TO
    ENTITY_NAME1 CHAR (15),
    ENTITY_TYPE1 CHAR (15),
    ENTITY_NAME2 CHAR (15),
    ENTITY_TYPE2 CHAR (15);

TABLE DERIVED_FROM
    ENTITY_NAME1 CHAR (15),
    ENTITY_TYPE1 CHAR (15),
    ENTITY_NAME2 CHAR (15),
    ENTITY_TYPE2 CHAR (15);

TABLE CALLS
    ENTITY_NAME1 CHAR (15),
    ENTITY_TYPE1 CHAR (15),
    ENTITY_NAME2 CHAR (15),
    ENTITY_TYPE2 CHAR (15);

TABLE REPRESENTED_AS
    ENTITY_NAME1 CHAR (15),
    ENTITY_TYPE1 CHAR (15),
    ENTITY_NAME2 CHAR (15),
    ENTITY_TYPE2 CHAR (15);

TABLE STANDARD_FOR
    ENTITY_NAME1 CHAR (15),
    ENTITY_TYPE1 CHAR (15),
    ENTITY_NAME2 CHAR (15),
    ENTITY_TYPE2 CHAR (15);

TABLE HAS_SORT_KEY
    ENTITY_NAME1 CHAR (15),
    ENTITY_TYPE1 CHAR (15),
    ENTITY_NAME2 CHAR (15),
    ENTITY_TYPE2 CHAR (15);

TABLE HAS_ACCESS_KEY
    ENTITY_NAME1 CHAR (15),
    ENTITY_TYPE1 CHAR (15),
    ENTITY_NAME2 CHAR (15),
    ENTITY_TYPE2 CHAR (15);

TABLE ALIAS
```

```
    ENTITY_NAME CHAR (15),
    ENTITY_TYPE CHAR (15),
    ALIAS_NAME  CHAR (15);

TABLE CATEGORY
    ENTITY_NAME CHAR (15),
    ENTITY_TYPE CHAR (15),
    CATEGORY_NAME CHAR (15);

TABLE RELATIONSHIP
    ENTITY_NAME1 CHAR (15),
    ENTITY_TYPE1 CHAR (15),
    ENTITY_NAME2 CHAR (15),
    ENTITY_TYPE2 CHAR (15),
    RELATION CHAR (15);

TABLE ENTITY
    ENTITY_NAME CHAR (15) NOT NULL,
    DESCRIPTION CHAR (60),
    CLASSIFICATION CHAR (10),
    DATE_ADDED DATE,
    ADDED_BY CHAR (15),
    LAST_MODIFICATION_DATE DATE,
    LAST_MODIFIED_BY CHAR (15),
    NUMBER_OF_MODIFICATIONS NUMBER,
    LOCATION CHAR (15),
    COMMENTS CHAR (45),
    SECURITY CHAR (10);
```

APPENDIX B
LISTING OF THE TUPLES IN THE DATA BASE

```
UFI> SELECT *
  2  FROM USER←X;

USER←NAME  DESCRIPTION          CLASSIFI DATE←ADDE ADDED←BY   LAST←MODI LAST←MODIF
---------- -------------------- -------- --------- ---------- --------- ----------
NUMBER←OF←MODIFICATIONS LOCATION     COMMENTS             SECURITY
----------------------- ------------ -------------------- ----------

JOHNSON    ODS USER             PROGR    22-APR-85 JONES      15-JUL-85 FORD
                      2 DISK                               UNCLASS

THOMAS     ODS USER             SYS ANA  15-JUN-85 JONES      22-SEP-85 FORD
                      3 DISK                               UNCLASS

KING       ODS USER             MANAGER  10-MAY-85 FORD       13-AUG-85 JOHNSON
                      2 DISK                               UNCLASS
```

```
UFI> SELECT *
  2  FROM SYSTEM;

SYSTEM←NAM SYSTEM     DESCRIPTION     CLASSIFI DATE←ADDE ADDED←BY   LAST←MODI
---------- ---------- --------------- -------- --------- ---------- ---------
LAST←MODIF NUMBER←OF←MODIFICATIONS LOCATION     DURATION←VALUE DURATION←T
---------- ----------------------- ------------ -------------- ----------
COMMENTS        SECURITY
--------------- ----------

ACC-2      ACCOUNT    ACCOUNT SYS #2           15-MAR-85 SCOTT      24-SEP-85
JONES                           3 DISK
                UNCLASS

RES-5      RESEARCH   RESEARCH SYS #4          18-MAY-85 WAITE      19-AUG-85
SHIELDS                         5 DISK
                SECRET

SAL←1      SALES      SALES SYS                21-JUL-85 ANDERSON   21-OCT-85
BUCKLEY                        15 DISK
                UNCLASS
```

```
UFI> SELECT *
  2  FROM FILE←X;

FILE←NAME  DESCRIPTION
---------- ------------------------------------------------------------
CLASSIFICA DATE←ADDE ADDED←BY          LAST←MODI LAST←MODIFIED←B LOCATION
---------- --------- --------------- --------- --------------- ---------------
NUMBER←OF←MODIFICATIONS NUMBER←OF←RECORDS
----------------------- -----------------
COMMENTS                                            SECURITY
---------------------------------------------- ----------

ACC5FILE   OUTPUT FILE OF ACC5PROG
ACCOUNT    25-AUG-85 SCOTT T. L.      30-AUG-85 ALLEN G. M.      DISK
                      1                   8
                                                    UNCLASS

ACC6FILE   INPUT FILE OF ACC5PROG
ACCOUNT    25-AUG-85 SCOTT T. L.      05-SEP-85 SCOTT T. L.      DISK
                      1                   5
                                                    UNCLASS

ACC7FILE   DATA AND PROG FILE OF ACC5PROG
ACCOUNT    28-AUG-85 ALLEN G. M.                                 DISK
                      0                   4
                                                    UNCLASS

ACC8FILE   SPECIAL OUTPUT FILE OF ACC5PROG
ACCOUNT    27-AUG-85 SCOTT T. L.      10-SEP-85 SCOTT T. L.      DISK
```

```
FILE+NAME  DESCRIPTION
---------- ----------------------------------------------------------
CLASSIFICA DATE+ADDE ADDED+BY        LAST+MODI LAST+MODIFIED+B LOCATION
---------- --------- --------------- --------- --------------- ---------------
NUMBER+OF+MODIFICATIONS NUMBER+OF+RECORDS
----------------------- -----------------
COMMENTS                                     SECURITY
-------------------------------------------- ----------

                      2                 6
                                             UNCLASS
```

```
UFI> SELECT *
  2  FROM RECORD;

RECORD←NAM DESCRIPTION
---------- ----------------------------------------------------------
CLASSIFICA DATE←ADDE ADDED←BY           LAST←MODI LAST←MODIFIED←B
---------- --------- ------------------ --------- ---------------
NUMBER←OF←MODIFICATIONS RECORD←CAT COMMENTS
----------------------- ---------- ----------------------------------------------------
SECURITY
----------

PAYROLL5   #5 PAYROLL RECORD
ACCOUNT    11-JUL-85 DOE J.H.           15-JUL-85 DOE J. H.
                     6 ACCOUNT
UNCLASS

PAYROLL6   #6 PAYROLL RECORD
ACCOUNT    11-JUL-85 DOE J. H.
                     0 ACCOUNT
UNCLASS

PERSON4    SALESMAN RECORD
PERSONNEL  15-AUG-85 SCOTT T. L.        09-OCT-85 SCOTT T. L.
                     1 PERSONNEL
UNCLASS

PERSON5    PROGRAMMER RECORD
PERSONNEL  10-AUG-85 SCOTT T. L.
```

```
RECORD+NAM DESCRIPTION
---------- --------------------------------------------------------------
CLASSIFICA DATE+ADDE ADDED+BY          LAST+MODI LAST+MODIFIED+B
---------- --------- -------------- --------- --------------
NUMBER+OF+MODIFICATIONS RECORD+CAT COMMENTS
----------------------- ---------- -----------------------------------------
SECURITY
----------

                        0 PERSONNEL
UNCLASS
```

```
UFI> SELECT *
  2  FROM ELEMENT;

ELEMENT←NA DESCRIPTION
---------- ------------------------------------------------------------
CLASSIFICA DATE←ADDE ADDED←BY        LAST←MODI LAST←MODIFIED←B
---------- --------- --------------- --------- ---------------
NUMBER←OF←MODIFICATIONS ALLOWABLE←RANGE ALLOWABLE←VALUE CODE←LIST←LOCAT
----------------------- --------------- --------------- ---------------
DATA←CLASS     COMMENTS                                           SECURITY
-------------- -------------------------------------------------- ----------

ACCE2      INDICATES SALARY OF INDIVIDUAL
ACCOUNT    11-JUL-85 DOE J. H.       28-AUG-85 DOE J. H.
                     3
DISK           ACCOUNT                                            UNCLASS

ACCE3      INDICATES ADDRESS OF INDIVIDUAL
ACCOUNT    15-JUL-85 DOE J. H.
                     0
DISK           ACCOUNT                                            UNCLASS

PRE3       SKILL AREA OF INDIVIDUAL
PERSONNEL  12-JUL-85 ALLEN G. M.     15-AUG-85 DOE J. H.
                     2
DISK           PERSONNEL                                          UNCLASS

PRE4       TERMINATION YEAR OF JOB
PERSONNEL  12-JUL-85 ALLEN G. M.     20-AUG-85 DOE J. H.
```

```
ELEMENT+NA DESCRIPTION
---------- -----------------------------------------------------------
CLASSIFICA DATE+ADDE ADDED+BY        LAST+MODI LAST+MODIFIED+B
---------- --------- --------------- --------- ---------------
NUMBER+OF+MODIFICATIONS ALLOWABLE+RANGE ALLOWABLE+VALUE CODE+LIST+LOCAT
----------------------- --------------- --------------- ---------------
DATA+CLASS       COMMENTS                                          SECURITY
--------------- ------------------------------------------------- ----------

                        3
DISK             PERSONNEL                                         UNCLASS

PRES       TITLE OF JOB
PERSONNEL  23-AUG-85 DOE J. H.        12-SEP-85 ALLEN J. H.
                        2
DISK             PERSONNEL                                         UNCLASS
```

```
UFI> SELECT *
  2  FROM RUNS;

ENTITY←NAME1    ENTITY←TYPE1    ENTITY←NAME2    ENTITY←TYPE2
--------------- --------------- --------------- ---------------
SCOTT T. L.     USER            ACCPG8          PROGRAM
ALLEN           USER            ACCPG9          PROGRAM
ALLEN G. M.     USER            ACC51MOD        MODULE

UFI> SELECT *
  2  FROM GOES←TO;

ENTITY←NAME1    ENTITY←TYPE1    ENTITY←NAME2    ENTITY←TYPE2
--------------- --------------- --------------- ---------------
ACCPG8          PROGRAM         ACCPG9          PROGRAM
ACC51MOD        MODULE          ACC61MOD        MODULE
SAL-1           SYSTEM          ACC-2           PROGRAM

UFI> SELECT *
  2  FROM DERIVED←FROM;

ENTITY←NAME1    ENTITY←TYPE1    ENTITY←NAME2    ENTITY←TYPE2
--------------- --------------- --------------- ---------------
ACC5DOC         DOCUMENT        ACC5FILE        FILE
PER3DOC         DOCUMENT        PERSON4         RECORD
PRE3            ELEMENT         PERSON4         RECORD
ACC8FILE        FILE            ACC8DOC         DOCUMENT
```

```
UFI> SELECT *
  2  FROM ALIAS;

ENTITY←NAME     ENTITY←TYPE     ALIAS←NAME
--------------- --------------- ---------------
ACCOUNT-2       PROGRAM         ASEL
ACCOUNT         PROGRAM         APRO
ACCOUNT-2       PROGRAM         APRO

UFI> SELECT * FROM CATEGORY;

ENTITY←NAME     ENTITY←TYPE     CATEGORY←NAME
--------------- --------------- ---------------
ACCOUNT-2       PROGRAM         TOM
PAYROLL5        PROGRAM         TOM
PAYROLL6        PROGRAM         TOM
ACC5FILE        FILE←X          TOM
ACC6FILE        FILE←X          TOM
```

```
JFI> SELECT *
  2  FROM RELATIONSHIP ;

ENTITY←NAME1    ENTITY←TYPE1    ENTITY←NAME2    ENTITY←TYPE2    RELATION
--------------  --------------  --------------  --------------  --------------
FILE            FILE            FILE            FILE            CONTAINS
FILE            FILE            DOCUMENT        DOCUMENT        CONTAINS
FILE            FILE            RECORD          RECORD          CONTAINS
FILE            FILE            ELEMENT         ELEMENT         CONTAINS
FILE            FILE            DOCUMENT        DOCUMENT        DERIVED←FROM
FILE            FILE            FILE            FILE            DERIVED←FROM
FILE            FILE            ELEMENT         ELEMENT         HAS←SORT←KEY
FILE            FILE            ELEMENT         ELEMENT         HAS←ACCESS←KEY

8 records selected.
```

```
UFI> SELECT *
  2  FROM CALLS;

ENTITY+NAME1     ENTITY+TYPE1     ENTITY+NAME2     ENTITY+TYPE2
--------------   --------------   --------------   --------------
ACCPG8           PROGRAM          ACC51MOD         MODULE
ACCPG8           PROGRAM          ACC61MOD         MODULE
ACC611MOD        MODULE           ACC61MOD         MODULE
ACCPG8           PROGRAM          PERPG3           PROGRAM

UFI> SELECT *
  2  FROM REPRESENTED+AS;

ENTITY+NAME1     ENTITY+TYPE1     ENTITY+NAME2     ENTITY+TYPE2
--------------   --------------   --------------   --------------
PRE3             ELEMENT          CH3              CHAR+STRING
PRE4             ELEMENT          FP5              FIXED+POINT
ACCE3            ELEMENT          CH11             CHAR+STRING
FIN7             ELEMENT          F14              FLOAT
```

LIST OF REFERENCES

1. Leong-Hong, B. W. and Plagman B. H., Data Dictionary/Directory Systems, John Wiley & Sons, Inc., 1982.

2. Allen, F. W. ,Loomis, E. S. , and Manning, M. V., "The Integrated Dictionary/Directory System", Computing Surveys, Vol. 14, No. 2, June 1982.

3. Lefkovits, H. C. , Sibley, E. H. , and Lefkovits, S. L. , Information Resource / Data Dictionary Systems, QED Information Sciences, Inc. , 1983.

4. Kroenke, D. , Database Processing, Science Research Associates, Inc. , 1983.

5. Oracle Corporation, ORACLE Manual Vol. I, 1984.

6. Winston, P. H. , Artificial Intelligence, Addison Wesley Publishing Co. , Inc. ,1984.

7. Waterman, D. A. , and Lenat, D. B. , Building Expert Systems, Addison Wesley Pub. Co. , 1983.

8. Clocksin, W. F. and Mellish, C. S. , Programming in Prolog, Springer - Verlag, Berlin, 1984.

BIBLIOGRAPHY

Berghel, H. L. , "Simplified Integration of Prolog with RDBMS", Data Base, 16, 3 (Spring 1985), 3-12.

Browne, et al, An Evolutionary Data Base Management System. Proceedings of 4th IEEE Compsac 1980, Chicago, IL, October 1980.

Cardenas, A. F. , Data Base Management Systems, 2nd ed. , Allyn and Bacon, Inc. , 1985.

IEEE 1984 International Symposium on Logic Programming 6-9 February 1982.

Kerschberg, L. , Marchand, D. ,Sen, A. , Information System Integration : A Metadata Management Approach, Proceedings of the 4th International Conference on Information Systems, Ross, K. (ed.), Houston, TX, December 1983.

Koltemann, J. E. and Konynski, B. R. , Dynamic Metasystems for Information Systems Development, Proceedings of the 5th International Conference on Information Systems, Maggi, L.,King, J. L. ,and Kraemer, K. L. , (eds.), Tucson, AZ, November 1984.

Mc Carthy, J. L. , Metadata Management for Large Statistical Databases, Proceedings of 8th International Conference on Very Large Data Bases (VLDB), Mexico City, Mexico, September 1982.

NBS Special Publication 500-92, Data Base Directions Information Resource Management - Strategies and Tools, Goldfine A. H. (ed), September 1982 Oracle Corporation, ORACLE Manual Vol. II, 1984.

Sturdza, P. , Data Dictionary Design with an Artificial Intelligence Model, Proceedings of the 16th Hawaii International Conference on System Sciences, 1983.

Ullman, J. D. , Principles of Database Systems, Computer Science Press, Inc. , 1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5100	2
3.	Department Chairman, Code 52MI Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
4.	Prof. Dolk D.R., Code 54Dk Department of Administrative Science Naval Postgraduate School Monterey, California 93943	1
5.	Prof. Hsiao D.H., Code 53Fs Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
6.	Gokhan Dedeoglu Eminalipasa Cad. 91-4 Bostanci, Istanbul, TURKEY	4
7.	Deniz Kuvvetleri Komutanligi Personel Daire Baskanligi Bakanliklar, Ankara TURKEY	5
8.	Deniz Harb Okulu Komutanligi Fen Bilimleri Bolum Baskanligi Tuzla, Istanbul TURKEY	1
9.	Deniz Harb Okulu Komutanligi Kutuphanesi Tuzla, Istanbul TURKEY	1
10.	Istanbul Teknik Universitesi Bilgisayar Bilimleri Fakultesi Kutuphanesi, Istanbul, TURKEY	1
11.	Bogazici Universitesi Bilgisayar Bilimleri Fakultesi Kutuphanesi,Istanbul, TURKEY	1
12.	Ahmet Corapcioglu NPS SMC # 2913 Monterey, CA 93943	2