A DATA DICTIONARY SYSTEM FOR A HIGH-LEVEL DATA MODEL

A Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Kei-Chang

August, 1983

ACKNOWLEDGEMENTS

I am very grateful to my advisor, Dr. Ramez Elmasri, without his advice, assistance, and kind encouragement, the preparation of this thesis would not have been possible. I am also grateful to my thesis committee members, Dr. Marek Rusinkiewicz and Dr. Richard W.Scamell, for their criticisms and suggestions. Finally, I am grateful to my husband Ben and my parents for their encouragement.

A DATA DICTIONARY SYSTEM FOR A HIGH-LEVEL DATA MODEL

An Abstract of a Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Kei-Chang

August, 1983

ABSTRACT

Data is a resource in both its physical and descriptive aspects. Both should be managed as any other major organizational resource. A data dictionary system is a formal method to handle and control the data information. It enables management to enforce data definition standards; it supplies information about the creation, usage and relationships of data; it eliminates the data redundancy and data inconsistency; it aids the security of sensitive data definitions against unauthorized use. In this paper, we first introduce the data dictionary system concept and discuss the Entity-Category-Relationship (ECR) model. Then a data dictionary design based on the ECR model is presented. Both the method and examples of the schema and the data dictionary files are presented. The data dictionary files creation, contents, function and relationships are discussed. In addition, function procedures which help the users to get the information from the data dictionary files are presented.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Database technology is an important topic in the area of computer and management science. Many thousands of organization have become dependent on the continued and successful operation of a database system [DATE 77]. A database system is a system which records, maintains information and permits users to access the data in a natural and friendly way. Achieving effective database usability and reponsiveness depends on the database model being used in the database system. High-level data models, such as the entity-category-relationship model used in this thesis, allow for easier database design, straightforward definition of queries and transactions, and correct representations of constraints on the data.

The data dictionary is a principal component of a database system. This paper describes a data dictionary design based on the entity-category-relationship model at the conceptual level. The entity-category-relationship model is an extension of Chen's entity-relationship model [CHEN 76] in several ways. The most important is that entities are classified in two ways: (1) according to

similarity of properties. (2) according to roles which they may play in a relationship.

The data dictionary is also an aid to assist the database administrator in cataloging and maintaining the database design. So the data dictionary becomes one of the most important tools in database administration. An integrated data dictionary is also used by the database system software whenever the software needs information on the structure of the data, and hence is an essential part of the database system.

1-1. Data Dictionary Definition, Objectives and Users

The data dictionary is a centralized repository of information about data. It is a basic tool to help the database administrator, system analyst and user to plan, control and use the data. The scope of the data dictionary is quite divergent. It can be narrowed so that it only covers the database definition necessary to support the software of a database management system, or it can be expanded to cover all the data important to an organization [Curtice 81], such as the hardware environment, the user's name list and the database definition. Figure 1 illustrates the logical structure of a typical data dictionary. The

dotted line rectangle includes the narrow meaning of a data dictionary. It includes data files, relationships, records, groups, elements, transactions and reports. An element is the smallest unit of data that can be referenced by a process through a transaction or report. A relationship is an association between one or more files. Group is a grouping of logically related elements and/or groups. For example, the grouping of month, day and year into DATE. Record is the smallest unit of physical data. Transaction is a specific set of input data that triggers the execution of a specific process or job [UHROWCZIK 73]. A report defines the information to be presented to the user. Process is a procedure that accomplishes a specific data task. The extended data dictionary will include the hardware data consisting of processers, lines and terminals as well as information on users of the database system.

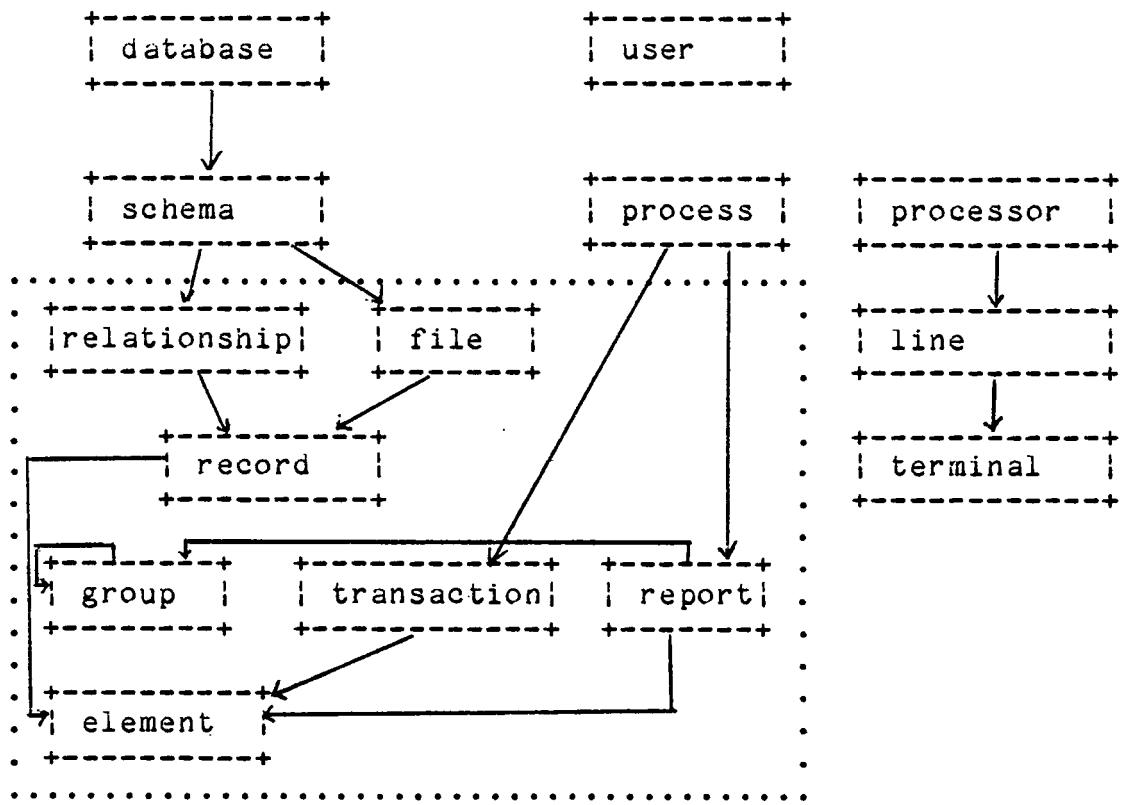


Figure 1. Logical structure of a typical data dictionary

The general data dictionary system has the following objectives:

(1). Achieving control of the data resource, preventing redundancy and inconsistency in the source data collection, having a repository for information.

(2). Reducing implementation lead time and cost. A data dictionary can provide complete and standard data

definitions for use in an application program, so it can help control the implementation cost.

(3). Reducing update lead time and cost.

(4). Establishing the standards for data usage. As a repository of data information, standards for data names, definitions, storage format, locations and access can be easily established and enforced.

The above objectives are very similiar to the objectives of a database management system. But with the aid of a data dictionary, the objectives could be achieved to a much higher degree.

Six groups of users might interface with the data dictionary:

(1) Database administrators: Those who plan, design, monitor and restructure the database use the data dictionary as a tool to store the data description information.

(2) Application system planners: Those who generate the data definition use the data dictionary to avoid duplicating data definition, thus preventing data redundancy and inconsistency.

(3) System analysts, Programmers: They use the data

dictionary to reduce coding effort, store the design of systems and support analysis of dictionary system changes [FRANK 82].

(4) Data processing management: Those who prepare the queries use the data dictionary system to create job control language parameters; control of different versions of program libraries and data files.

(5) End user: Those who obtain their view of data from the data dictionary.

(6) Software programs: Includes compilers, interpreters and DBMS software which generate macro definitions for use by specific database management systems or generate structured source-data definition statements for inclusion in source code by programmers for all programming languages.

1-2 . The Need for a Data Dictionary System

When the enterprise expands, the data of the enterprise grows rapidly. Management becomes aware that the data is an important and valuable corporate resource. The data should be managed and controlled with much care as the other resources. A data dictionary system enables management to

enforce data definition standards, supplies information about data creation, usage and relationships, eliminates the data redundancy and errors caused by data inconsistency, and the security of sensitive data definition against unauthorized user [DATA BASE 77]. This section will show that a data dictionary system is a valuable tool for other users in the enterprise.

(A). Management of the data resource

The data always exists in its files, on forms passing between individuals and departments. The orderly flow and presentation of those data to the interested parties depends on the data dictionary system. Because the data dictionary system:

(1) knows what data exists and where and how it is used.

(2) controls modifications to existing data or processes using new data.

(3) controls plans for new use of data and the acquisition of new types of data.

When a large amount of data in an installation is used by more than one application, it becomes necessary to establish central control of data definition and use. This is accomplished by the function of data administration. The

data administrator must ensure that all applications are using the data correctly and it is being properly maintained. A data dictionary system helps the data administrator in specifing the storage media, backup requirements and availability criteria.

(B). System Control

The information systems of an enterprise may change because of

(1) normal development and growth of the enterprise.

(2) changes in enterprise structure from reorganization or merging.

(3) changes in the software environment, such as introduction of a DBMS.

As an enterprises' information system develops in size and complexity, standards alone fail to ensure an optimum use of the data resource. The amount of data information becomes large and unwieldy. This needs a formal method to handle and control the information. The formal methods are embodied in the data dictionary system which provides a means for effectively recording and handling the new information.

A data dictionary system is ideally suited to these system changes since it can provide the following forms of support:
(1) It records all details of the current system and therefore provides a baseline from which to plan changes.
(2) It can record details of all proposed changes and can report any inconsistencies in these proposals.
(3) It can report on some of the major cost factors involved in implementing specific changes and from which a cost-effective implementation sequence can be devised [DATA BASE 77].

(C) Relations with users

As an enterprise develops, the functional specialization becomes important. As a result, the users become unaware of the significance of their part in the enterprise structure and fail to appreciate the effect they have on it. The data dictionary system serves as a central easily accessible source of information on any aspect of the enterprise's information system that the user may feel is relevent to him. It increases the user's confidence in making them aware of their roles in the system. The data dictionary system performs a valuable public relations role.

It is clear from the above discussion that a data dictionary system is an important tool to support the data administration. Although a data dictionary system will create new tasks, in return it will reduce the effort required by such activities as documentation, coding of programs, creation of test data files, checking and auditing of output files.

1-3 . Integrated Data Dictionary System

An ideal data dictionary can operate in any environment without loosing its efficiency and functionality. However, at present this ideal is far from attainable. According to the dependency degree which the data dictionary system relies on DBMS, the data dictionary system can be separated into three approaches: Independent approach, DBMS application approach and embedded approach [FRANK 82]. In this paper, a data dictionary with the embedded approach is presented,i.e, the data dictionary is a component of the DBMS. The data dictionary is the only source of data information. DBMS provides the management service on data dictionary, such as recovery or concurrency control. On the other hand, the DBMS accesses the database via the data dictionary. So this is an integrated data dictionary

system. In Figure 2, we can see the relationship between the data dictionary system and the DBMS.

In the independent approach of the data dictionary system the data dictionary is completely independent of DBMS. The DBMS maintains its own data dictionary and accesses the database via its own dictionary. The data dictionary system may provide facilities description, software interface between different DBMS [FRANK 82].

The data dictionary system in the DBMS application approach is just like another database. The DBMS maintains its own data dictionary for each database. The data dictionary system interfaces dynamically with only one DBMS and its components but it may interface statically with other DBMSs that operate in the same hardware environment.

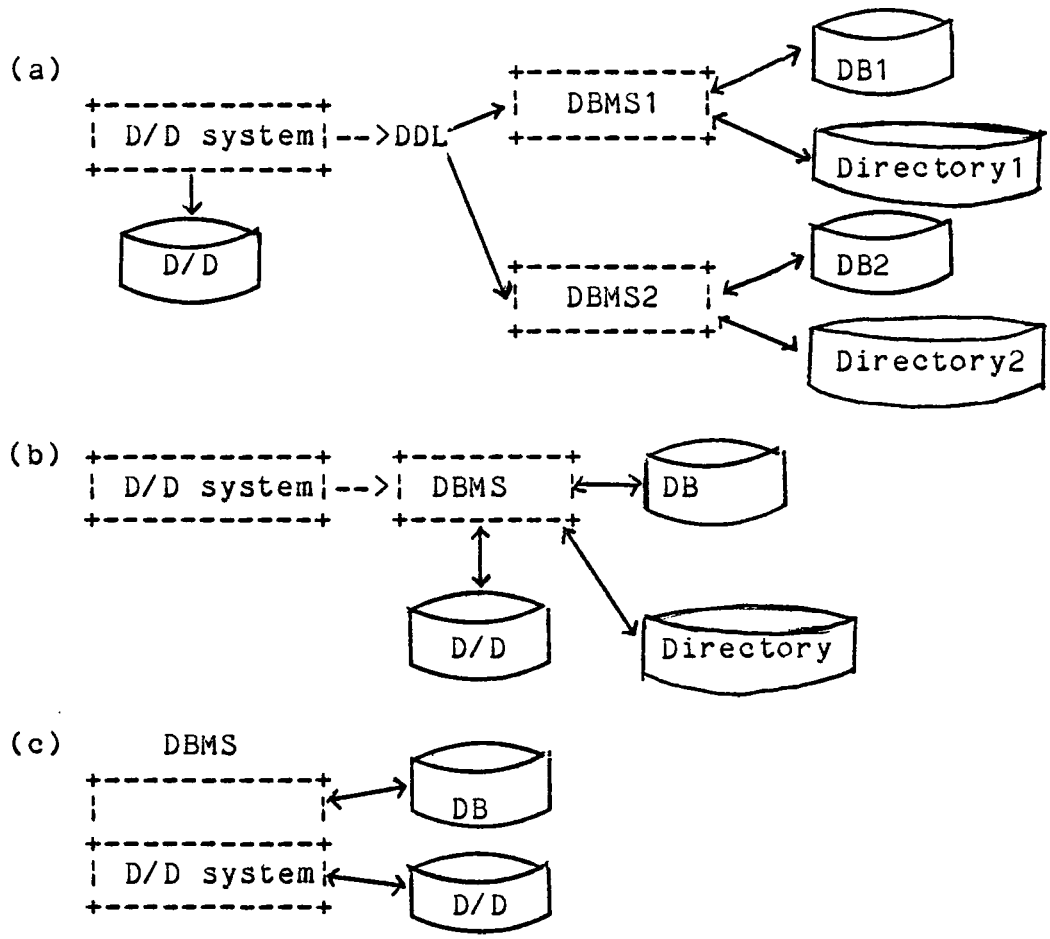


Figure 2. Data dictionary system:DBMS relationships

(a) Independent approach (b) DBMS application approach
(c) Embedded approach

1-4. Data Dictionary Future Directions

The capabilities of data dictionaries are expanding in many directions. Some of them will increase their capabilities in the database design process. They are

becoming useful for defining and standardizing all data in an organization, not only DBMS stored data, or even all computerized data. Some of data dictionaries increase their usability in process and program-oriented data instead of just data about data. In some cases, the documentation of a system is also done by data dictionary.

Proper management of the data resource can be achieved via a data dictionary which provides a variety of services to a number of users. In addition, the use of data dictionary can be extended in the future to compilers and DBMS for further productivity improvements. However, the effective use of a data dictionary system depends on a strong data management commitment. This support is difficult to obtain in many organizations [FRANK 82]. How to improve the cooperation between DBMS and data dictionary system is the most important issue in the near future.

CHAPTER 2

ENTITY-CATEGORY-RELATIONSHIP MODEL

In database design, the choice of a data model is a very important issue. The data model must be able to capture the data meaning, the data independence, and the semantics of each user application, and also provide a straightforward means of data definition and data manipulation. The currently popular data models are the relational model, the hierarchical model, and the DBTG network model. The relational model can achieve a high degree of data independence but it may lose some important semantic information [CHEN 76]. The DBTG network model provides a more natural view of data by separating entities and relationships (to a certain extent), but its capability to achieve data independence has been challenged [CHEN 76]. In this paper, we use the entity- category-relationship model of data at the conceptual level which is a generalization of Chen's entity-relationship (ER) model. First we present the ANSI/SPARC three level-schema architecture for DBMS to understand the different views of a database.

The ANSI/SPARC three-schema architecture supports three views of a database [ANSI 77]: (see figure 3)

External schema : user view of the database. It is a set of rules describing how information is viewed by a program or user [NIJSSEN 77]. There will be numerous external schema in general.

Conceptual schema: the enterprise view of the database. It functions as a 'stable platform' to which internal and external schemas may be mapped. It will provide a semantic description of the database, free of implementation concerns [WEEL 80,HR-80-251]. It is a set of rules describing which information may enter and reside in the database [NIJSSEN 77].

Internal schema: the system view of the database. It is a set of rules describing how information is physically represented on storage media [NIJSSEN 77].

Because of the function of the conceptual schema, a change to the user view of the database or a change to the system view of the database won't impact the conceptual schema .

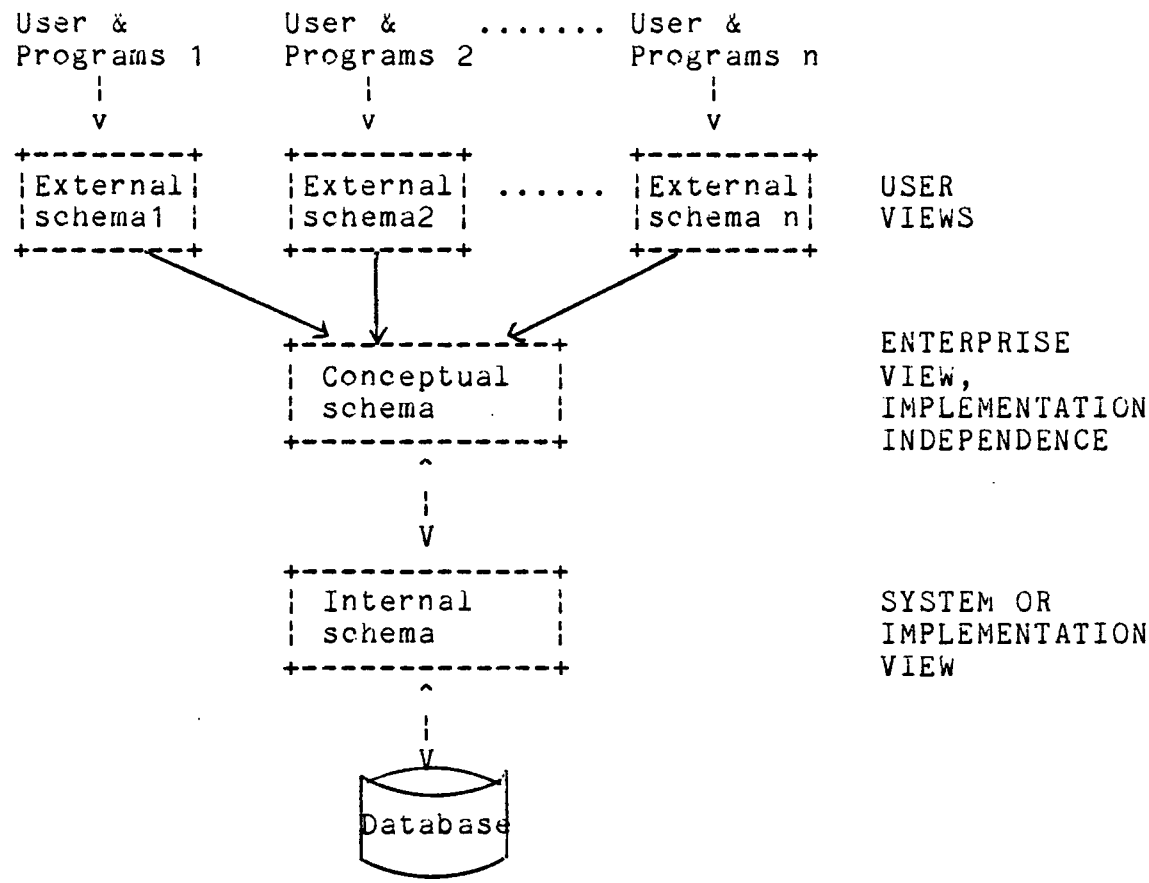


Figure 3. ANSI/SPARC 3-schema architecture

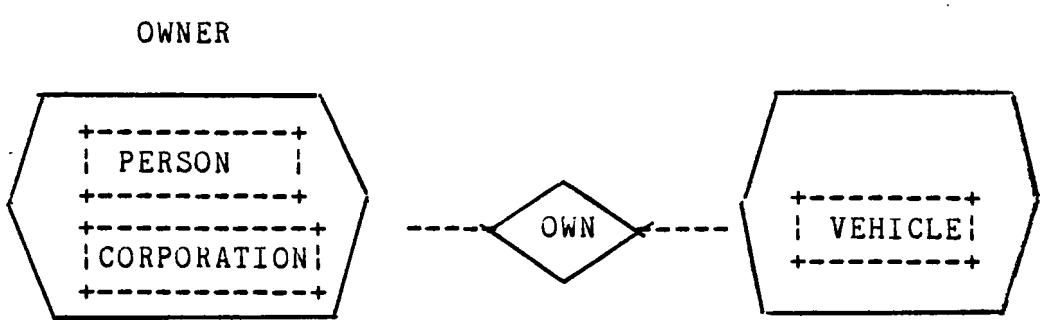
2-1. Introduction to Entity-Category-Relationship Model

The entity-category-relationship model (ECR model) is a valuable design tool for user view modeling and integration because of its rich semantics. Once the views are integrated into one conceptual schema, the need arises for

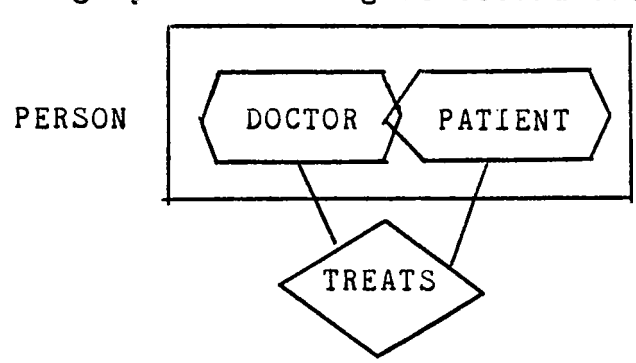
straightforward means of data definition and data manipulation. The entity-category-relationship model was developed at the Honeywell Corporate Technology Center. It is the extension of the entity-relationship model (ER model). The primary weaknesses of the ER model are in the areas of (1) completeness, (2) multiple kinds of connections among entity sets, (3) implementation independence, and (4) the distinction between entities and their indentifiers [WEEL 80,HR-80-250].

In the ECR model, two kinds of entity sets, types and categories are distinguished to remedy the first two weaknesses in the ER model. Entities are classified into types according to similarity of fundamental properties. Types are disjoint, an entity belongs to one and only one type. Entities are grouped into categories according to the roles they play in the relationships. Categories are not disjoint in that an entity may be a member of several categories. For example, John Smith is of type person, but may be a member of categories employee, student and vehicle owner. Category also provides for the grouping of dissimilar entities according to the relationship roles they play ,i.e, the multiple-member set concept provided by the DBTG model. In the ER model one can't directly model

subsets of entity sets nor higher order collections of entities (sets of entity sets). But in the ECR model, subsets and higher order collections of entities are defined based upon the need to group entities according to their possible relationship roles. For example, in figure 4, category OWNER functions as a second order collection of PERSON and CORPORATION. Categories DOCTOR and PATIENT are two subsets of the PERSON entity type.



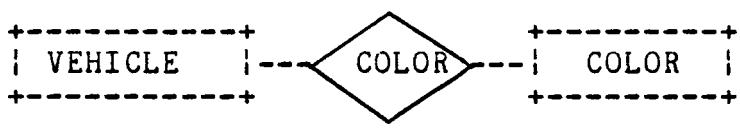
(a) Category functioning as second order collection



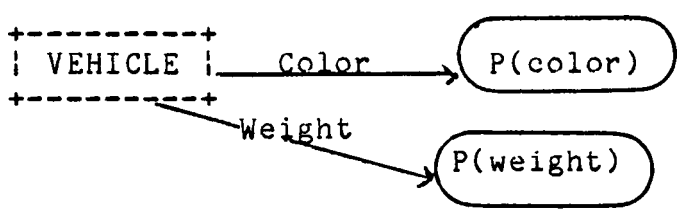
(b) Category functioning as subsets

Figure 4. Using categories to represent subsets and higher order collections of entities.

The ER model restricts attributes of entities and relationships to be single valued. Any multiple valued attributes would be treated as separate entities and relationships. In the ER model, attributes are defined as mathematical functions mapping entity sets or relationship sets into value sets, i.e, for every entity in an entity set, there is exactly one value in the corresponding value set for an attribute. But in the ECR model, attributes are defined as a function mapping a set of entities into the power set of a value set. Then both single-valued and multi-valued attributes are modeled in a uniform manner. Figure 5 is a example.



(a). ER Model



(b). ECR Model

Figure 5. Representing the multiple valued attribute

With regards to the fourth weakness, the ER model relies upon key attribute values to represent an entity. If unfortunately, some of the key attribute values should change, it would result in deletion of an entity and insertion of a new one although the new entity is the same as the old in reality. Within the ECR model, an entity represents itself. It is not necessary to specify key attribute values to identify an entity.

2-2. Entity, Category, Attribute and Relationship

The ECR Model views the world as consisting of entities, relationships among them and attributes which provide information about both entities and relationships.

ENTITY,CATEGORY AND TYPE

An entity is a 'thing' which can be distinctly identified. Persons, companies, vehicles are examples of entities. Entities are grouped into sets according to two criteria:(1) into entity types according to similarity of fundmental properties. Types are disjoint. An entity may belong to one and only one entity type. For example, a man with social security number 123-45-6789 is an entity, a set

of men is an entity type. (2) into categories according to the roles they play within relationships. Categories are not necessarily disjoint. An entity may belong to any number of categories in the database. For example, in Figure 6 members of the AUTOMOBILE type and of the TRUCK type may be categorized as VEHICLE in an owner-vechicle relationship.

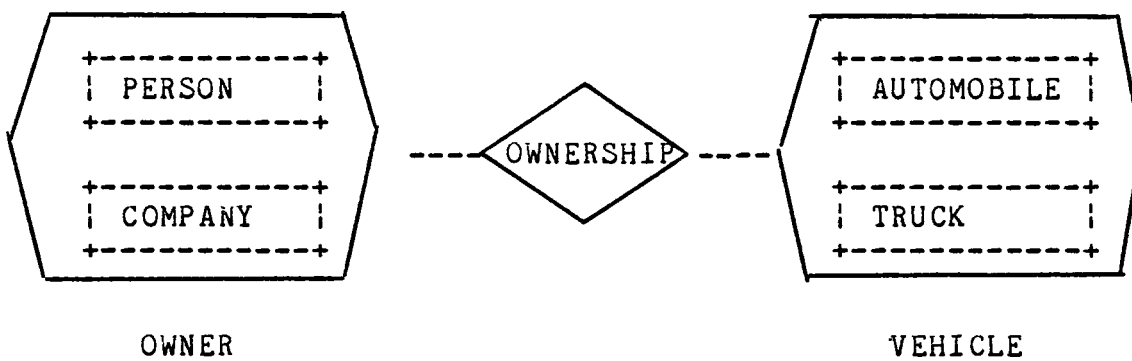


Figure 6. Vehicle Registration Database

The entities of a category C are specified as follows:

C=T1[S1] U T2[S2] U.....U Tn[Sn]

where each Ti is an entity type or category which are called the defining entity types and categories, each Si is a predicate which are called defining predicates. The predicate specifies the set of entities from Ti that belong to C to those entities that satisfy Si [ELMASRI 81]. The

predicate is optional. If no predicate is given with an entity type specification, the members of the category are the entities from that entity type that are explicitly inserted in the category. If the predicate Si is TRUE, then all the entities in Ti belong to C.

For example, in Figure 7, Job, Name and Number are the attribute names of EMPLOYEE. Category SECRETARY is defined on the entity type EMPLOYEE with the predicate Job='SECRETARY'. Category ENGINEER is defined on the entity type EMPLOYEE with the predicate Job='ENGINEER'.

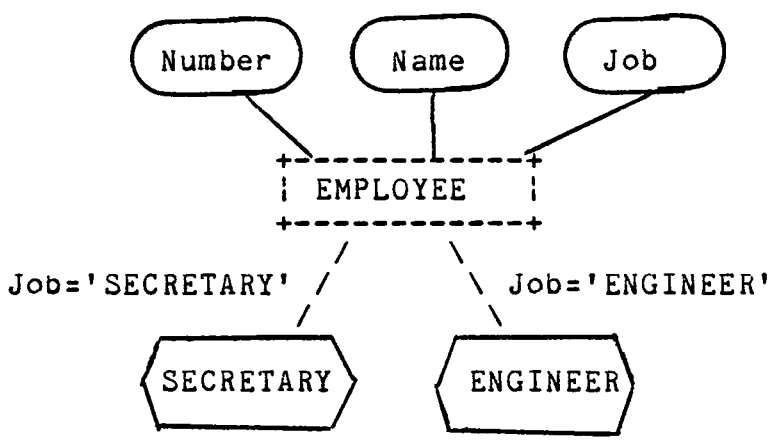


Figure 7. Category defined with a predicate

RELATIONSHIP

Relationships are mathematical relations over

categories of entities. Each of the categories is said to participate in the relationship R, i.e, R is a relationship set or relationship over the not necessarily distinct categories C1,C2 ...Cn if it is a subset of C1XC2X...XCn. A relationship instance within R is a tuple of entities <x1,x2 ...xn>, where xi is an element of Ci. The relationship may be 1 to 1, 1 to many, many to 1 and many to many .

The structural properties of relationships specify constraints on each category Ci which participates in the relationship R by two numbers (I1,I2); 0<=I1<=I2 and I2>0. The numbers mean that each entity that is a member of category Ci must exist in at least I1 and in at most I2 relationship instances in R. If I1=0, a relationship R is partial with respect to the participation of Ci, and if I1>=1 a relationship R is total. If I1=I2=1, the participation is unique which means exactly one appearance of an entity x in Ci exists in R at a given moment [ELMASRI 81].

A relationship R is specific with respect to the participation of Cj if the participation is total and once an entity e from Cj is related by some relationship instance in R, that relationship can't be deleted unless e itself is deleted [ELMASRI 81].

ATTRIBUTES

Attributes are the properties which describe the entity types, categories and relationships. An attribute is a function which maps an entity, category or relationship instance into the power set of a value set. The power set of a set is the collection of all possible subsets of that set, including the empty set. As a result, both single-valued and multi-valued attributes are modeled in a uniform manner [Weel 80,HR-80-250]. As the example in Figure 5(b). Each entity instance that is a member of the category inherits the attribute values from its entity type [Hevner 80].

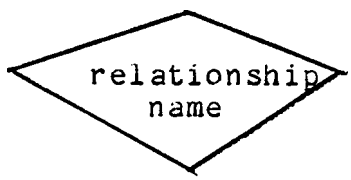
2-3. ECR Diagram

The ECR diagrammatic technique is an extension of the diagrams for the ER model . Rectangular boxes represent entity types, diamond boxes represent relationships, ovals represent attributes which are connected by lines to the entity type, relationship or category of which they are a property, hexagonal boxes represent categories.

Partial participation is indicated by joining the

category and relationship by a single line, total participation by a double line, and specific participation by a triple line. The inclusion of an entity type in a category is indicated by drawing a line between the two and placing a set inclusion symbol,C, on the line. An arrowhead in the many to one direction represents a functional relationship [ELMASRI 81].

We will use following convention for ECR names:
Attribute names will be all in capitalized.
Entity type, category, relationship and valueset names will be all in uppercase.
Participation names will be all in lowercase.



RELATIONSHIP

ENTITY TYPE WHICH ALSO IS A CATEGORY

Figure 8. ECR diagram

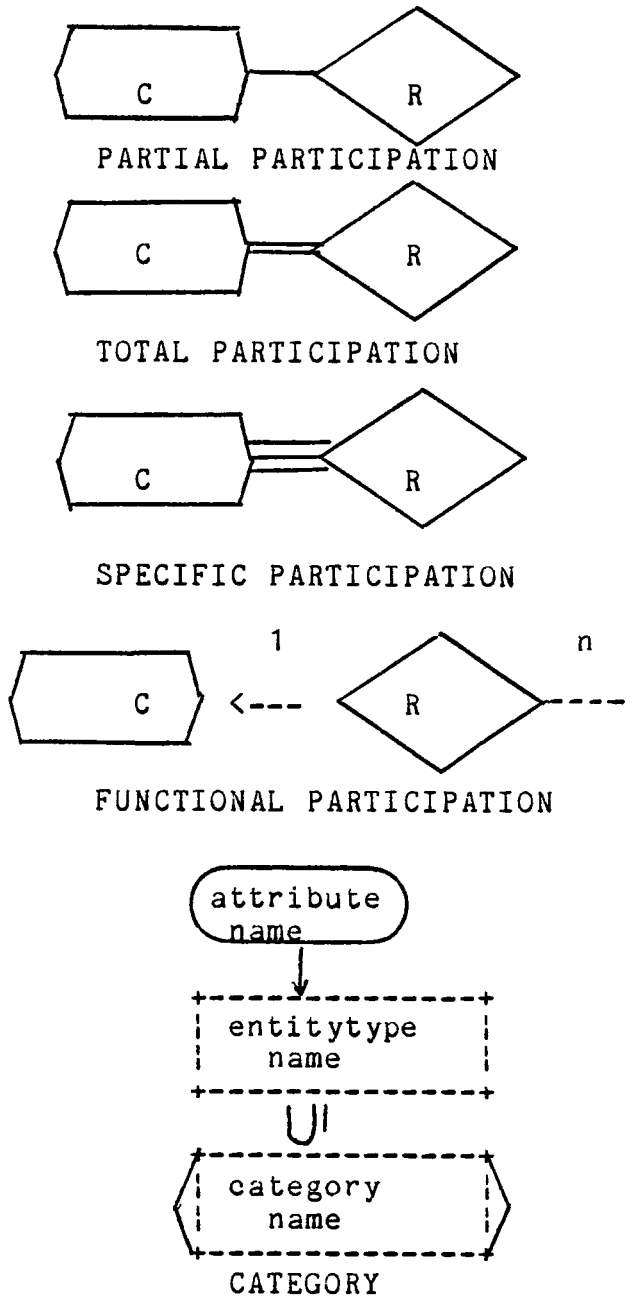


Figure 8.(cont.) ECR Diagram

Here we take an example to explain ECR model. As in Figure 9, it is an ECR schema for a company database. EMPLOYEE is an entity type name and also is a category name. The attribute names of EMPLOYEE are Name, Address, Hphone, Ssn, and Ophone. Category name SCIENTIST is defined on EMPLOYEE, Sgrade is its attribute name and it is the inclusion of entity type name EMPLOYEE. WORKS-ON is a relationship name between PROJECT and EMPLOYEE and the relation is partial participation. The participation names for the category PROJECT participates the relationship WORKS-ON instance are (employees,projects) and the participation names for the category EMPLOYEE participates the relationship WORKS-ON instance are (projects, employees). The relationship name ASSIGNED is the relationship instance between DIVISION and FULL-TIMEEMPLOYEE and the relation is total participation between ASSIGNED and FULL-TIMEEMPLOYEE.

In this example, entity type and also category names are EMPLOYEE, PROJECT and DIVISION. Category names are SCIENTIST, TECHNICIAN and FULL-TIMEEMPLOYEE. Relationship names are ASSIGNED, MANAGES, WORKS-ON. Attribute names are Name, Number, Location, Pstart, Sal, Address, Hours, Hstart, Ssn, Hphone, Ophone, Sgrade, Tgrade and Union.

Participation names are employees, division, manager and proj-managed.

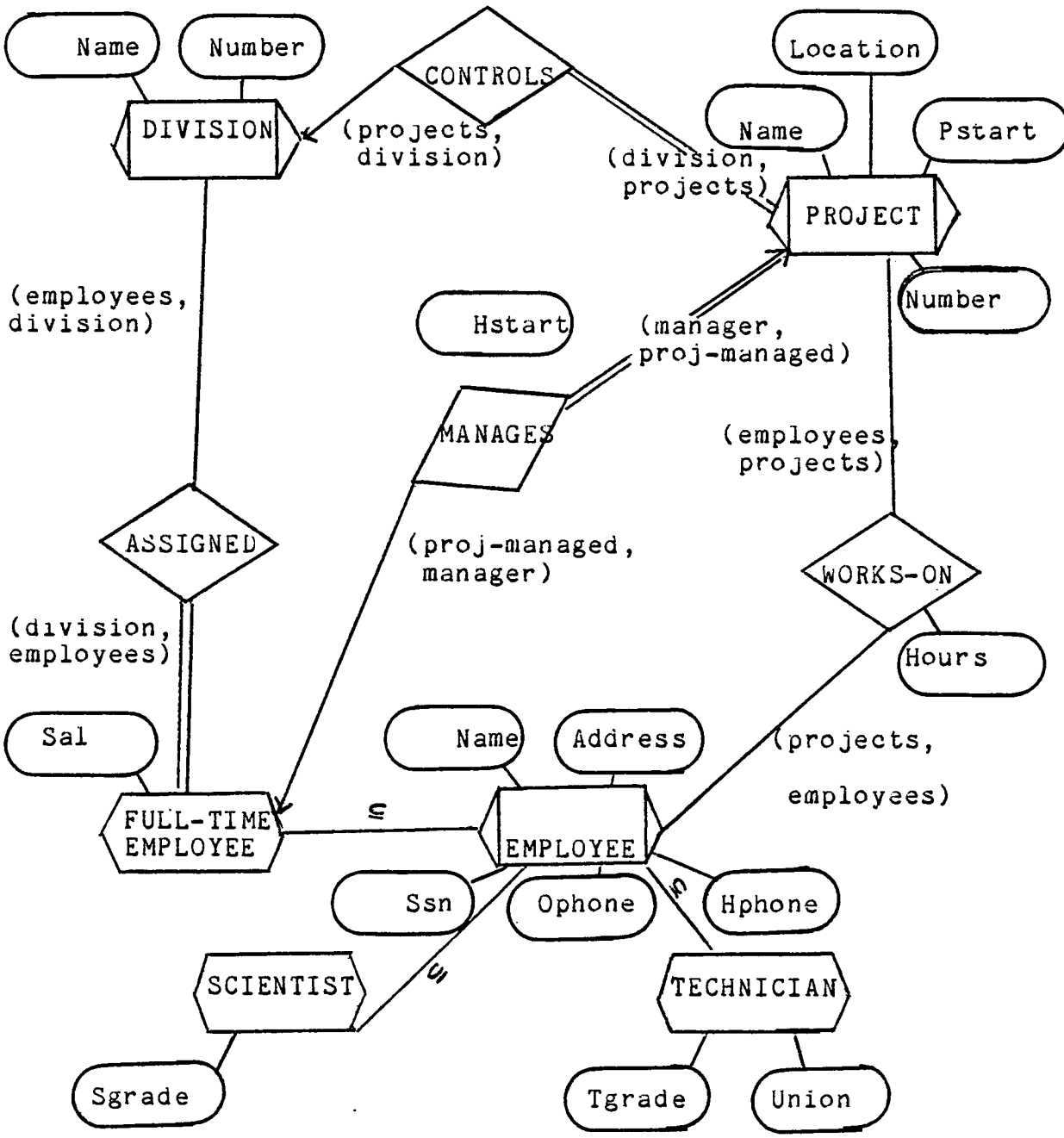


Figure 9. ECR schema for a company database

CHAPTER 3

DATA DEFINITION LANGUAGE

The statement for defining the constructs of an ECR schema is the DEFINE statement. The define statement is composed of five types of statements: those for defining value set, entity type, category, relationship and data file. The BNF symbols used in data definition language are:

[x]: x occurs 0 or 1 times.

{x}: x occurs 0 or more times.

x|y: x or y.

(x|y|.....|z): x or y or ...or z, used for grouping alternatives.

'x': x is a literal metasymbol.

BNF for DEFINE statement:

< define statement >::=DEFINE (<valueset>|< entitytype >|<category>|<relationship>| < file >)

In figure 10 , an example of an ECR schema is given for a specific application. The application manages data about

the students, departments and courses in a university. We will use this example to illustrate the definition of value set, entitytype, category, relationship and file. The ECR diagram of the university database is as follows:

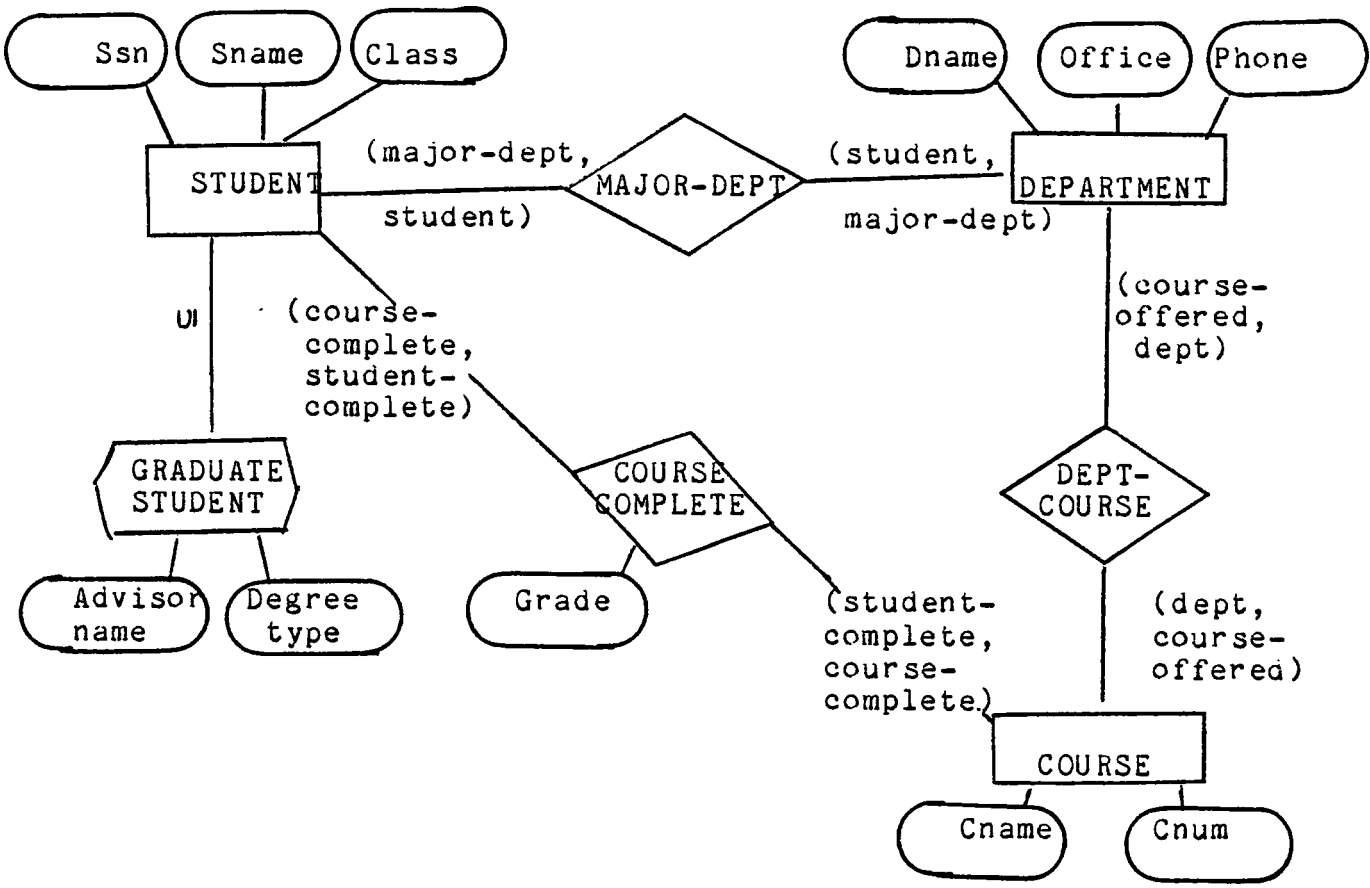


Figure 10. ECR diagram of university database application

3-1 . Value Set Definition

A value set is defined by its name and description. The value set names must be unique because they are used to refer to the value set when an attribute is defined. The valueset descriptions can be one of the following:

(1). A set of explicit values. This specification is used for small value sets with values of irregular structure.

(2). A standard data type. It can be integer, real, string or date. The standard type can be restricted to a subrange.

(3). By reference to an attribute of some given class, such as entitytype, category or relationship. Here we define the value set to be the set of values that currently exists in the database for the given attribute of the given class. No semantic check should be performed on the values of that attribute by the system.

The BNF for valueset definition are as follows:

< value set >::=VALUESET <valueset name > AS <valueset description>

< valueset name >::=< string of characters>

< valueset description >::=< explicit valueset >|< standard valueset >| <reference valueset >

< explicit valueset>::='{'<constant>{,<constant>}'}'

< standard valueset >::=STRING [< integer >]|INTEGER [RANGE [< integer >] [:< integer >]]| REAL [INTERVAL [< real >][: < real >]]

< reference valueset >:==< attribute name > OF < class name

> : < low level domain >

< low level domain >::=<standard valueset>

The definition of value sets in the university application example are as follows:

```
DEFINE VALUESET SOCSECNUM AS STRING 12

DEFINE VALUESET STUDENTNAME AS STRING 15

DEFINE VALUESET CLASSLEVEL AS { 1,2,3,4,5 }

DEFINE VALUESET DEPTNAME AS DNAME OF DEPARTMENT :STRING

DEFINE VALUESET OFFICENUM AS STRING 5

DEFINE VALUESET PHONENUM AS INTEGER 0:9999999999

DEFINE VALUESET COURSENAME AS CNAME OF COURSE :STRING

DEFINE VALUESET COURSENUM AS CNUM OF COURSE:  STRING

DEFINE VALUESET ADVISORNAME AS STRING 15

DEFINE VALUESET DEGREETYPE AS { BS,BA,MS,PHD }

DEFINE VALUESET GRADELEVEL AS { A,B,C,D,E,F,W,I }
```

3-2. Entitytype Definition

An entity type is defined by its name and attribute list. The entity name must be unique, since it is used by the user to specify the entity type when formulating a query or transaction. Each attribute in the list must have a unique name and is associated with a valueset.

The cardinality of an attribute, a , is the number of values from valueset v that can appear in a(x) for any entity x in entitytype T . The cardinality of an attribute is specified by the(optional) integers following MIN (n1) and MAX (n2); 0 =< n1 <= n2 and n2 >0 . The attribute a is single-valued if n2 =1 and multi- valued if n2 >1 . The attribute a is total if n1 >=1 and is partial if n1=0. The default values for (n1, n2) are (1,1), i.e. (single-valued,total). The key word UNIQUE specifies a unique attribute which is used to model key attributes that serve to identify entities of a given type. Here we ignore the compound attributes.

The BNF for entity type definition:

< entity type >::= ENTITYTYPE <entitytype name > ATTRIBUTES < attribute list>

< attribute list >::=< constrained attribute >{,<

constrained attribute>}

< constrained attribute >::=< attribute name > VALUESET < valueset name > [UNIQUE] [MIN < integer >][MAX <integer>]

The definition of entitytypes in the university application are as follows:

```
DEFINE ENTITYTYPE STUDENT ATTRIBUTES
Ssn VALUESET SOCSECNUM UNIQUE,
Sname VALUESET STUDENTNAME,
Class VALUESET CLASSLEVEL MIN 1;

DEFINE ENTITYTYPE DEPARTMENT ATTRIBUTES
Dname VALUESET DEPTNAME UNIQUE,
Office VALUESET OFFICENUM MIN 1 MAX 3,
Phone VALUESET PHONENUM MIN 1 MAX 5;

DEFINE ENTITYTYPE COURSE ATTRIBUTES
Cname VALUESET COURSENAME UNIQUE,
Cnum VALUESET COURSENUMBER UNIQUE;
```

3-3. Category Definition

A category is defined by its name, the entity type specifications which define the category and the attributes

list. There are two ways to define a category. If a category is defined identically as a single entity type, then the name of category can be the same as the name of the entitytype. The other way is to give a new name to the category, then the list of entity type specifications that define the category are given.

Each entity type specification is given by the entitytype name and an optional predicate (logical expression) which selects the subset of entities which are the members of the category. So the category is formed by the union of those subsets of entities. If the optional predicate is not given, any entities from the entitytype can be members of the category. Specific attributes for the category (if they exist) are defined by an attribute list. We call such attributes specific attributes of the entities that participate in the category, while those attributes that are defined on the entitytype of the entity are called the basic attributes of an entity.

The BNF for category definition:

< category >:==CATEGORY < category name > (= < entity type name >| FROM <entity type specifications > [SPECIFIC] ATTRIBUTES <attribute list >])

< entity type specifications >::=< entitytype spec >[:<predicate>]{,<entitytype spec>[:<predicate>]}

< entitytype spec >::=(< entitytype name >|< category name >)

The category definitions of the university application are as follows:

DEFINE CATEGORY GRADUATESTUDENT FROM STUDENT ATTRIBUTES
Advisor VALUESET ADVISORNAME ,
Degree VALUESET DEGREETYPE ;

3-4. Relationship Definition

A relationship is specified by its name, participation list and attribute list. The relationship name is unique from all other relationship names, entitytype names and category names in the schema.

In a relationship there must exist at least two participations. Each participation of a category or entity type in the relationship is specified by the category or entity type name, the structural constraints on the participation (the integers after MIN and MAX), and a pair of names (name1, name2) for the participation. Name1 and

Name2 are used by the query language to refer to the relationship from the category and the category from the relationship.

A relationship is SPECIFIC with respect to the participation of C if when an entity e from C is related by some relationship instance in R, that relationship instance can't be deleted unless e is deleted.

The PD (prohibit deletion) and DLT (delete related instance) are non-default specifications to maintain the basic relationship constraint and the MIN constraint. If both of them are omitted, the default is assumed, which is DLT for the basic relationship constraint, and PD for the MIN constraint.

The attribute names in the attributes list of the relationship must all be unique.

The BNF for the relationship definition:

< relationship >:==RELATIONSHIP < relationship name > FROM < participation list > [ATTRIBUTES <attribute list >]

< participation list >:==< participation >, < participation > {,< participation>}

< participation >::=(< category name|<entity type name >) [SPECIFIC] [PD] '(' < participation name1>, <participation name2>')' [MIN <integer > [DLT]][MAX < integer >]

The relationship definition of the university application are as follows:

```
DEFINE RELATIONSHIP COURSECOMPLETE FROM
STUDENT (coursecomplete,studentcomplete),
COURSE (studentcomplete, coursecomplete) MIN 5,
ATTRIBUTES Grade VALUESET GRADELEVEL;

DEFINE RELATIONSHIP DEPTCOURSE FROM
DEPARTMENT (courseoffered, dept) MIN 1 MAX 1,
COURSE (dept, courseoffered);
```

3-5 . Data File Definition

A data file is defined by its name, file type, field list, primary key and secondary keys. The data file name must be unique. The file type is limited to index sequential file or hash file for our current implementation. The field list is specified by the field name, start location, length, its type (integer, real or string), and a boolean key value for each field in the file.

In the index sequential file , the key can be a primary key or secondary key which is defined by the key start

location and length. Only one primary key can be defined. The number of secondary keys is not limited. In the hashing file, there is only one key, the hash key, which is defined in the field specification. So there is no key specification in the hashing file.

The BNF for data file definition:

< data type >::=FILE < file name > TYPE < file type > FIELDS

< attribute name > < starting location >< type >< length ><key> {, <attribute name > <starting location >< type >< length >< key >} PRIMARY KEY < key spec > [SECONDARY KEY < key spec > {,< key spec >}]CORRESPONDS TO < relationship | category | entitytype > <class name > [JOIN ATTRIBUTES <attribute name > REFERS < attribute name >< file name > {, <attribute name >REFERS <attribute name ><file name>}];

<file name >::=< string of characters>

<class name >::=<string of characters>

< file type >::= indexseq | hash

< starting location >::= < integer >

< type >::= integer | real | string <length>

< length >::=< integer >

< attribute name >::=< string of chars >

< key >::= T | F

< key spec >::= <starting location > < length >

The data file definition of university application are as follows:

```
DEFINE FILE student TYPE indexseq FIELDS
sname 1 string 15 t,
ssn 16 integer 4 t,
class 20 integer 4 f,
advisorname 24 string 15 f,
degreetype 39 string 5 f,
deptname 44 string 15 f
PRIMARY KEY 16 4,
SECONDARY KEY 1 15
CORRESPONDS TO ENTITYTYPE STUDENT;


DEFINE FILE department TYPE hash FIELDS
dname 1 string 15 t,
office 16 string 5 f,
phone 21 integer 4 f
CORRESPONDS TO ENTITYTYPE DEPARTMENT;


DEFINE FILE course TYPE indexseq FIELDS
cname 1 string 15 t,
cnumber 16 string 15 t,
deptname 31 string 15 f
PRIMARY KEY 16 15,
SECONDARY KEY 1 15
CORRESPONDS TO ENTITYTYPE COURSE;


DEFINE FILE studentcourse TYPE indexseq FIELDS
```

```
Scnumber 1 string 15 t,
Scssn 16 integer 4 t,
grade 20 string 1 f
PRIMARY KEY 1 20,
SECONDARY KEY 1 15,16 4
CORRESPONDS TO RELATIOSHIP COURSECOMPLETE
JOIN ATTRIBUTES SCNUMBER REFERS CNUMBER COURSE,
SCSSN REFERS SSN STUDENT;
```

3-6 Naming Restrictions in an ECR Schema

In order to avoid ambiguity in the query language statements, some naming requirements are needed on an ECR schema. These are:

(1) All value set names are unique.

(2) All entity type, category, relationship and data file names are unique.

(3) All attribute names of a particular entity type are unique.

(4) All attribute names of a particular category are unique.

(5) All participation names of a particular category or entity type in the relationships must have unique names for the first participation name.

(6) All participation names for a particular relationship must have unique names for the second participation name.

CHAPTER 4

DATA DICTIONARY AND FUNCTIONS

In this chapter, a data dictionary with the embedded approach for the ECR model is presented. So the data dictionary is the only source of information and the DBMS accesses the database via the data dictionary. A data dictionary is created based on the schema or data definition. The schema is scanned one statement following another. If any syntax error is detected, an error message is given and the schema definition is aborted. Based on the schema, there are eight data dictionary files and a number of database files. The ten data dictionary files are TOTAL, VALUESET, ENTITY, EXPLICT, ATTRIBUTE, CATEGORY, CATEGREL, RELATIONSHIP, DICTIONARY and FILETYPE dictionary files. The contents and format of these files will be illustrated by the example of Figure 10.

4-1 Data Structure for the Data Dictionary Files

Before we insert the data in the data dictionary, we first have to create the dictionary files. A linked list data structure is used and a pointer to the linked list is

passed to the file system. The file system will create the file based on the linked list. The pointer type passed to the file system is called FileChainType which points to a record containing four fields: FileName, FileType, KeyChain and FieldChain. The structure is as follows:

```
FileChainType=^SpecOfFile;

SpecOfFile= RECORD
          FileName: StringType;
          FileType: StringType;
          KeyChain: ChainType;
          FieldChain: ChainType
            END;
```

Filename gives the name of the dictionary file. Filetype is the type of the dictionary file. Currently we have two types of files: Index sequential file and hash file. Both the length of file name and file type are limited to 12 characters. Keychain and FieldChain belong to Chaintype. The structure of ChainType is as follows:

```
ChainType=^List;

List= RECORD
       Pos,Size: Integer;
       Next: Chaintype;
       Case IsField: Boolean of
         TRUE: (FieldName: String15;
                Fieldtype: Char;
                KeySpec: Integer);
         FALSE: ;
         END
       END;
```

KeyChain points to a linked list of records. Each record specifies one key for the file. In the index sequential file, there are three types of keys: primary key, combination key and secondary key. In the hash file, there is only one key, hash key. For a combination key specification, the record which specifies the key contains the start location, the length of the key, the next pointer, IsField (true), the key field name, the field type(C) and keyspec(integer). A combination key requires a name in order to refer to it by that name. A non-combination key is formed of only one field, and we may refer to it by the name of that field. For the other key specifications, the keychain points to a record which only contains the start location, the length of the key, the next pointer, and IsField (false).

FieldChain points to a linked list. Each record in the list specifies one field of the file. The information in each record includes the start location and the length of the field, the next pointer, IsField (true), the field name, the field type and keyspec. The field name is the name of the field and is restricted to 15 characters. The field type is the data type of the field which can be I (Integer), S (String of characters), L (Long integer), R (Real) and C (

Combination key). Keyspec represents the order of the key, if the field is a key and it is the primary key then keyspec is 1, if it is a secondary key then keyspec is 2,...etc. If the field is not a key then keyspec is 0.

Based on the above data structure, we can create the database dictionary files. But the files created are empty until we insert records based on the schema definition. The data structure to insert records in the dictionary files is the same as that for creating the dictionary file. Each field value is placed in a record. The records are linked together, then a pointer to the linked list is passed to the file system. The file system will insert a record based on the field values passed to it. Before going on and introducing the structures of dictionary files for the entity-category-relationship model, we make the following restrictions:

(1). The file name is limited to 12 Characters at most.

(2). The field name is limited to 15 characters at most.

(3). If the field type is 'S', i.e, String of characters, the length of that field occupies 15 bytes.

(4). If the field type is 'I' or 'R',i.e, Integer or Real, the length of that field occupies 4 bytes.

(5). If the field type is 'L',i.e, Long integer, the length

of that field occupies 8 bytes.
(6). If the field type is 'C',i.e, combination key, the length of that field occupies 1 byte. This field is not used, it is only needed by the file system.

4-2 Data Dictionary Files in ECR Model

In order to store the structure of an ECR schema in a way that information needed by the DBMS software or the users may be retrieved, we designed the following files for the data dictionary. For each file, we describe its use, define its type, its fields and keys, and present functions that retrieve data from the file. These functions are used by either the DBMS software or users to access the dictionary information.

1. TOTAL DATABASE DICTIONARY FILE (TOTALDBD)

(A). Dictionary file name: TOTALDBD.
This file is used by the scanner of the queries and transactions for an ECR DBMS. It includes the names and types of all objects (entity types, categories, relationships, attributes, valuesets, participation names) which may be refered to when a DBMS user defines a query or

transaction for the database.

(B). File contents: There are two fields in this dictionary file.

The first field name is 'Name' which is the name of some ECR schema object. This field includes all the names of valuesets, entity types, categories, relationships, attributes and participation names defined in the database schema. The type of this field is 'S'.

The second fieldname is 'Type' which means the type of the first field. It ranges from 1 to 6. '1' represents the type of attribute name, '2' represents the type of entity type name, '3' represents the type of relationship name,'4' represents the type of category name, '5' represents the type of participation name and '6' represents the type of valueset name. The type of this field is 'I'.

(C). File type and key field: This file is an indexed sequential file. The primary key field is the first field 'Name'.

(D). Function: From this file, we know all the names of schema objects and their type. This is the most basic repository of schema information. One function procedure is built based on this table:

FUNCTION Schema-type(Sname):Integer;

If the user wishes to know the type of an object name in this schema, he can call this function directly. The input of this function is the data name and the output is an integer which represents the type number. The user can then get more detailed information from the dictionary file which is refered by the type number.

(E). Example: The format and contents of TOTALDBD in the university database example are as follows:

Table 1. TOTALDBD

Name	Type
ADVISORNAME	6
Advisor	1
cdepartment	5
CLASSLEVEL	6
Class	1
Cname	1
Cnum	1
completecourse	5
completstudent	5
COURSECOMPLETE	3
COURSENAME	6
COURSENUM	6
courseoffered	5
COURSE	2
DEGREETYPE	6
Degree	1
DEPARTMENT	2
DEPTCOURSE	3
DEPTNAME	6
Dname	1
GRADELEVEL	6
GRADUATESTUDENT	4
Grade	1
OFFICENUM	6
Office	1
PHONENUM	6
Phone	1
Sname	1
SOCSECNUM	6
Ssn	1
STUDENTNAME	6
STUDENT	2

2. VALUESET DICTIONARY FILE (VALUESTDBD)

(A). Dictionary file name: VALUESTDBD.

This file includes the names, types and range or length of all valuesets. When a user defines a query or transaction, the database may refer this file to do type checking on values of attributes.

(B). File contents : This file contains four fields:

The first field name is 'ValuestName' which is the valueset name defined on schema and the type is 'S'.

The second field name is 'Type' which is the type of the valueset name. It can be 'I' (Integer), 'R' (Real), 'S' (String) or 'E' (Explicit value). The field type is 'S' but occupies only one byte. If the valueset name has the explicit values then this field is 'E' and all explicit values are given by another file named 'EXPLICITDBD'.

The third field name is 'Min' which specifies the minimum range or length of the valueset name. This field type is 'I'.

The fourth field name is 'Max' which specifies the maximum range or length of the valueset name. This field type is 'I'.

(C). File type and key field: This is an indexed sequential file and the primary key field is the first

field, 'valuestname'.

(D). Function: All the valueset names, types, range or length defined in the database are inserted in this file and the valueset name is also inserted in the TOTALDBD. There is one procedure based on this file could be called by the users or DBMS software:

PROCEDURE Valueset-name-type(Attname,Classname; Var valptr);

If the user wishes to know the valueset name and the type for an attribute name, he can call this procedure directly. The input of this procedure is an attribute name and class name. The procedure first finds the valueset name upon with this attribute is defined in ATTRIBUTDBD then finds the valueset type and min, max value in VALUESTDBD. The output is a pointer which points to the valueset name, its type and the min and max information.

(E). Example: The definition of valueset in the university database application:

DEFINE VALUESET SOCSECNUM AS STRING 12

after parsing this definition statement, the valueset name

SOCSECNUM will be inserted in the TOTALDBD and the type is 6. At the same time, SOCSECNUM is inserted in the first field, 'S' is inserted in the second field, '0' is inserted in the third field, and '12' is inserted in the fourth field in the VALUESTDBD.

The VALUESTDBD in the university database is as follows:

Table 2. VALUESTDBD

Valuestname	Type	Min	Max
ADVISORNAME	S	0	15
CLASSLEVEL	E	0	5
COURSENAME	S	0	256
COURSENUM	S	0	256
DEGREETYPE	E	0	4
DEPTNAME	S	0	256
GRADELEVEL	E	0	8
OFFICENUM	S	0	5
PHONENUM	I	0	99999999999
SOCSECNUM	S	0	12
STUDENTNAME	S	0	15

3. EXPLICIT DICTIONARY FILE (EXPLICTDBD)

(A). Dictionary file name : EXPLICTDBD.

When the valueset name is defined by explicit values, the explicit data values are inserted in this file one by one. This file includes the valueset name and its related

explicit values which may be referred to when a DBMS user defines a query or transaction for the database.

(B). File contents: This file includes two fields:
The first field name is 'Valuestname' which is the valueset name. The field type is 'S'.
The second field name is 'Datavalue' which is the data value. The field type is 'S'. The data value is inserted into this field one by one. So the first field is stored redundanctly if the number of data values is more than one.

(C). File type and key field: This file is an indexed sequential file and the primary key is a combination key which is the combination of the two fields. The primary key name is 'ExplicitPrimary'.

(D). Function: From this file we know all the explicit data values of one valueset name. The number of data values is inserted in the VALUESTDBD. This file is related with the file VALUESTDBD.

There is one function procedure based on this file:

PROCEDURE Explicit-Value(Vname; Var Exptr);

This procedure helps the user to get the explicit values of

the valueset name if the valueset values are defined by explicit values. The input is the valueset name and the ouptut are the explicit values stored in a linked list.

(E). Example: As in the Figure 10 example:

DEFINE VALUESET GRADELEVEL AS {A,B,C,D,E,F,W,I}

After parsing the above definition statement, the valueset name GRADELEVEL is inserted in TOTALDBD and its type is 6. At the same time, GRADELEVEL is inserted in the first field, 'E' in the second field, '0' in the third field, '8' in the fourth field of VALUESTDBD. GRADELEVEL is inserted in the first field, 'A','B','C','D','E','F','W','I' are inserted into the second field of EXPLICTDBD one by one. The first field will be repeated 8 times.

The EXPLICTDBD in the university database is as follows:

Table 3. EXPLICTDBD

Valuestname	Datavalue
CLASSLEVEL	1
CLASSLEVEL	2
CLASSLEVEL	3
CLASSLEVEL	4
CLASSLEVEL	5
DEGREETYPE	BS
DEGREETYPE	BA
DEGREETYPE	MS
DEGREETYPE	PHD
GRADELEVEL	A
GRADELEVEL	B
GRADELEVEL	C
GRADELEVEL	D
GRADELEVEL	E
GRADELEVEL	F
GRADELEVEL	W
GRADELEVEL	I

4. ATTRIBUTE DICTIONARY FILE (ATTRIBUTDBD)

(A). Dictionary file name: ATTRIBUTDBD.

All the attributes information about entity types, categories or relationships are included in this file. This file includes the attribute name, the class name entity type, category or relationship it belongs to, the type of class it belongs to, its valueset name, boolean value 'UNIQUE' and the MIN, MAX restrictions of this attribute

name.

(B). File contents: This file includes seven fields:
The first field name is 'Attributname' which is the name of the attributes and the type is 'S'.
The second field name is 'Belongto' which is the entitytype, category or relationship name which the attribute belongs to. The type of this field is 'S'.
The third field name is 'Type' which is the type of the second field, i.e., entity type, category or relationship. The field type is 'S'.
The fourth field name is 'Valuestname' which is the valueset name that the attribute is mapped with. The field type is 'S'.
The fifth field name is 'UNIQUE' which specifies whether the attribute name is unique or not. The field type is 'S'. Since it is a boolean value, it occupies one byte. A unique attribute is a 'key' attribute where no two entities can have the same value for it.
The sixth field name is 'MIN' which means the attribute is total if the value of this field is greater or equal to 1. The attribute is partial if the value of this field is 0. The default value is 1. The field type is 'I'.
The seventh field name is 'MAX' which means the attribute is

single- valued if the value of this field is 1. The attribute is multi-valued if the value of this field is greater than 1. The default value is 1. The field type is 'I'.

(C). File type and key field: This file is an indexed sequential file and the primary key is the combination key which is the combination of first two fields, the type is 'C' and the length is 30 bytes. The primary key name is 'AttributPrimary'. The second key is the first field, 'Attributname' and the third key is the second field, 'Belongto'.

(D). Function: All the attributes information are included in this file. The attribute name is inserted into TOTALDBD and the type is 1. All the attributes of entity types, categories and relationships are included in this file. The valueset name of the attribute is also in this file. So ATTRIBUTDBD has relation with TOTALDBD, VALUESTDBD, ENTITYDBD, CATEGORYDBD and RELATIONDBD.

There are five function procedures based on this file:

FUNCTION Belong-to(Name1, Name2):Boolean;

The user or DBMS software can use this function to check if

an attribute name belongs to an entitytype, category or relationship. The input of this function is name1 which is the attribute name and name2 which is the entity type, category or relationship name. The returned value is a boolean value True or False.

PROCEDURE Attribute-name(Sname; Var Attptr);

If the user or DBMS software wishes to know all the attribute names for a given entity type, category or a relationship, this procedure can be used. The input is the entity type, category or relationship name and the output are all the attribute names which belong to the input object name.

FUNCTION Attribute-Unique(Attname,Classname):Boolean;

If the user or DBMS software wishes to check an attribute is unique or not. This function can be used. The input are the attribute name and the class name it belongs to. The returned value is a boolean value True or False.

PROCEDURE Attribute-restrict(Attname, Classname; Var Minn,Maxn);

This procedure helps the user or DBMS software to find the

restriction number of a given attribute name. The input are the attribute name and the class name, the output are two integers Minn and Maxn which specify the attribute cardinalities.

PROCEDURE Valueset-name-type(Attname, Classname; Var Valptr);

This procedure helps the user or DBMS software to find the valueset name and its type upon giving an attribute name. When given an attribute name, it first finds the valueset name in ATTRIBUTDBD then finds the valueset type based on the valueset name in VALUESTDBD. The input are an attribute name and the class name, the output are valueset name, valueset type and min, max value stored in a linked list and a pointer to that list is returned.

(E). Example: Using the Figure 10 as an example:

After scanning the following definition statement:

```
DEFINE ENTITYTYPE STUDENT ATTRIBUTES
Ssn VALUESET SOCSECNUM UNIQUE,
Sname VALUESET STUDENTNAME,
Class VALUESET CLASSLEVEL MIN 1;
```

Entity type name STUDENT is inserted into TOTALDBD and ENTITYDBD. Attributes name Ssn is inserted into the first field, STUDENT in the second field, 'entity' in the third field, SOCSECNUM in the fourth field, 'T' in the fifth field, '1' in the sixth field and seventh field in ATTRIBUTDBD. The other two attributes informations Sname and Class are inserted in the same way.

The ATTRIBUTDBD file in the university database is as follows:

Table 4. ATTRIBUTDBD

Attribut name	Belongto	Type	Valuest name	Unique	Min	Max
Advisor	GRADUATE-STUDENT	category	ADVISOR-NAME	F	1	1
Class	STUDENT	entity	CLASSLEVEL	F	1	1
Cname	COURSE	entity	COURSENAME	T	1	1
Cnum	COURSE	entity	COURSENUM	T	1	1
Degree	GRADUATE-STUDENT	category	DEGREETYPE	F	1	1
Dname	DEPART-MENT	entity	DEPTNAME	T	1	1
Grade	COURSE-COMPLETE	relatio-ship	GRADELEVEL	F	1	1
Office	DEPART MENT	entity	OFFICENUM	F	1	3
Phone	DEPART MENT	entity	PHONENUM	F	1	5
Sname	STUDENT	entity	STUDENT NAME	F	1	1
Ssn	STUDENT	entity	STUDENTNUM	T	1	1

5. ENTITY DICTIONARY FILE (ENTITYDBD)

(A). Dictionary file name: ENTITYDBD.

This file includes the entity type name, the object names it is in relationships with and, its participation names. When a DBMS user defines a query or transaction the database software will refer this file to check the correctness of query formulation.

(B). File contents: This file includes three fields:
The first field name is 'Entityname' which is the name of the entity type. The field type is 'S'.
The second field name is 'Participation' which is the participation name that this entity type name participates in the relationship instance. This field is helpful to check the validity of the query.
The third field name is 'Relatedto' which is the other entity type name or category name that is related to the first field. The field type is 'S'.
When parsing the entitytype definition statement, only the entitytype name is inserted into ENTITYDBD and TOTALDBD. The second and the third field are blank until the relationship is defined with the first field .

(C). File type and key field: This file is an indexed sequential file and the primary key is a combination key which is the combination of the first two fields. The primary key name is 'EntityPrimary', type is 'C' and is 30 bytes long. The second key is the first field, 'entityname'.

(D). Function: From this file we know all the entity type names in the database and the related entity name or

category name with the first field and their participation names. The entity type name is inserted into the TOTALDBD. After a relationship is defined, the entity type name will appear in RELATIONDBD and the second and the third fields of ENTITYDBD are inserted. If any attribute name is defined, the attribute name and entity type name are also inserted in ATTRIBUTDBD . If any category name is defined on the entity type name or has relationships with the entity type name, the entity type name is inserted into CATEGORYDBD or CATEGRELDBD. So ENTITYDBD has relationships with TOTALDBD, ATTRIBUTDBD, CATEGORYDBD, CATEGRELDBD and RELATIONDBD.

There is one function procedure based on this file:

PROCEDURE Entity-Relate-Name(Sname; var relpt);

The user can find all the related entity type names or category names upon giving an entity name by calling this procedure. The input is an entity type name, the output are all the entity type names or category names and participation names related with the input name.

(E). example: we use the university database application as an example to illustrate how to insert into TOTALDBD, ENTITYDBD and ATTRIBUTDBD.

```
DEFINE ENTITYTYPE STUDENT ATTRIBUTES
Ssn VALUESET SOCSECNUM UNIQUE,
Sname VALUESET STUDENTNAME ,
Class VALUESET CLASSLEVEL MIN 1;
```

After parsing the above statement, the entity name STUDENT will be inserted into TOTALDBD and the type is 2. STUDENT is inserted into the first field of ENTITYDBD, the second and the third are blank temporarily. All the attributes informations are inserted into ATTRIBUTDBD which has been mentioned in 3.

The ENTITYDBD file in the university database application is temporarily as follows:

Table 5. ENTITYDBD

Entityname	Participname	Relatedto
COURSE		
DEPARTMENT		
STUDENT		

6. CATEGORY DICTIONARY FILE (CATEGORYDBD)

(A). Dictionary file name: CATEGORYDBD.

When a category is defined on an entity type name or another category name, this file is used. This file includes the category name, the object name it is defined on and the type.

(B). File contents: This file contains three fields:
The first field name is 'categname' which is the name of the category and its type is 'S'.
The second field name is 'Definedon' which is the name that the category is defined on. The field type is 'S'.
The third field name is 'Type' which is the type of the second field. The category may be defined on an entity type or another category. So the type can be 'entity' or 'category'. The type of this field is 'S'.

(C). File type and key field: This file is an indexed sequential file and the primary key field is a combination key which is the combination of the first two fields. The primary key name is 'CategoryPrimary', the type is 'C' and is 30 bytes. The secondary key field is the first field 'categname'.

(D). Function: This file specifies how a category is defined. The category name is insertd into TOTALDBD. If any attributes are defined on tnis category, all the

attributes informations are included in ATTRIBUTDBD. If the category name participates in any relationship instance, it will appear in RELATIONDBD. The category name is inserted into CATEGRELDBD. So CATEGORYDBD is related to TOTALDBD, ENTITYDBD, ATTRIBUTDBD, CATEGRELDBD and RELATIONDBD. There are two function procedures based on this file:

FUNCTION Defined-on(categname,definename):Boolean;

The user or DBMS software can apply this function to check whether a category name is defined on another category name or entity type name. The input is categname which is the category name and definename which is the entity type or category name. The output is a boolean value.

PROCEDURE Repeat-Definedon(Cname; Var Categptr);

A category may be defined on one entity type or another category and this entity type or category may be defined on another entity type or category. This relationship may be recursively. The user can use this procedure to check this recursive relations. The input is the category name and the output are all the entity types or categories which the given category name is defined. The output are stored in a linked list and a pointer to this list is returned.

(E). Example: Using the example of the university database application:

DEFINE CATEGORY GRADUATESTUDENT FROM STUDENT ATTRIBUTES
Advisor VALUESET ADVISORNAME,
Degree VALUESET DEGREETYPE;

Category name GRADUATESTUDENT is included in TOTALDBD and the type is 4. It will be inserted into the first field of CATEGORYDBD and CATEGRELDBD. Before we insert the STUDENT, we first check if STUDENT is an entity type name or category name. If it is an entity type name then insert STUDENT in the second field and 'entity' in the third field of CATEGORYDBD.

The CATEGORYDBD file in the university database application:

Table 6. CATEGORYDBD

Categname	Definedon	Type
GRADUATE-STUDENT	STUDENT	entity

7. CATEGORY RELATION DICTIONARY FILE (CATEGRELDBD)

(A). Dictionary file name: CATEGRELDBD

The file includes the category name, the object name it is related with and its participation name which may be referred to when a DBMS user defines a query or transaction for the database.

(B). File contents: This file includes three fields:

The first field name is 'Categname' which is the category name defined on the schema. The field type is 'S'.

The second field name is 'Participname' which is the first participation name if this category name participates a relationship instance. The field type is 'S'.

The third field name is 'Relatedto' which is the name of the class that the categroy name related to. The field type is 'S'.

The first field is inserted when the category name is defined. The second and the third fields are blank until the relationship is defined on this category.

(C). File type and key field: This file is an indexed sequential file and the primary key is a combination key which is the combination of the first two fields. The combination key name is 'Categoryprimary', the type is 'C' and occupies 30 bytes. The secondary key is the first field

name 'Categname'.

(D). Function: From this file, we know all the category names and the classes they relate with. When a category statement is defined, the category name is inserted into TOTALDBD, CATEGORYDBD and CATEGRELDBD. If the category participates in any relationships, the second and third fields of this file are filled.

There is one function procedure based on this file:

PROCEDURE Category-Relate-Name(Sname;Var Reltpt);

This procedure helps the user or DBMS software find all the related classes and their participation names upon giving a category name. The input is a category name and the output are all the participation names and the related names.

(E). Example: As in Figure 10 example:

```
DEFINE CATEGORY GRADUATESTUDENT FROM STUDENT ATTRIBUTES
Advisor VALUESET ADVISORNAME,
Degree VALUESET DEGREETYPE;
```

After parsing the above statement, GRADUATESTUDENT is inserted into the first field of TOTALDBD, CATEGORYDBD and CATEGRELDBD. The other information inserted is mentioned in

6. The second and third fields of this file are blank temporarily.

The CATEGRELDBD file in the university database application is as follows:

Table 7. CATEGRELDBD

Categname	ParticipName	Relatedto
GRADUATE-STUDENT		

8. RELATIONSHIP DICTIONARY FILE (RELATIONDBD)

(A). Dictionary file name: RELATIONDBD.
This file includes the relationship name, relation from, the participation names, boolean variable SPECIFIC, PD and DLT, the type and two relationship restrictions. It may be referred by the DBMS to check a query or transaction for correctness of specification.

(B). File contents: This file includes ten fields:
The first field name is 'RelationName' which is the relationship name. The field type is 'S'.
The second field name is 'RelationFrom' which is one of the entity type names or category names that participate in the

relationship. The field type is 'S'.

The third field name is 'Specific'. It means the participation is specific and is a boolean value. The field type is 'S' but occupies only one byte. The default value is 'F'.

The fourth field name is 'PD' which means prohibit deletion. It is a nondefault specification to maintain the min cardinality constraint. If it is omitted, the default value 'F' is assumed. The field type is 'S' but occupies one byte.

The fifth field name is 'DLT' which means delete related instances. It is also a nondefault specification to maintain the basic relationship constraint. If it is omitted, the default value 'F' is assumed. The field type is 'S' but occupies one byte.

The sixth field name is 'ParticipName1' which is the first participation name. It is used to refer to the relationship from the entity type name or category. The field type is 'S'.

The seventh field name is 'ParticipName2' which is the second participation name. It is used to refer to the entity type name or category name from the relationship. The field type is 'S'.

The eighth field name is 'Type' which is the type of the second field, i.e, 'E' or 'C'. the field type is 'S' but occupies one byte.

The ninth field name is 'Min' which is the minimum number of relationship instances that must be related to each entity. The field type is 'I'.

The tenth field name is 'Max' which is the maximum number of relationship instances in R which can be related to each entity. The field type is 'I'.

Because a relationship relates by two entity types or categories in the binary relationship, the relationship names appears at least twice in this file.

(C). File type and key field: This file is an indexed sequential file and the primary key is a combination key which is the combination of the first two fields. The primary key name is 'RelationPrimary' and occupies 30 bytes. The secondary key is the first field 'RelationName' and the third key is the second field 'RelationFrom'.

(D). Function: All the relationship information are contained in this file. When the relationship is defined, first we check if it relates entity types or categories. If it relates an entity type, the type is 'entity' and we go

back to ENTITYDBD to insert the information about that entity. We will illustrate this in the following example. If any attributes are defined on the relationship then all the information is inserted in ATTRIBUTDBD. The relationship name and participation name are inserted in TOTALDBD. The RELATIONDBD has relationship with ENTITYDBD, TOTALDBD, ATTRIBUTDBD, CATEGRELDBD and CATEGORYDBD.

There are three function procedures which could be called by the user:

PROCEDURE Relation-Restrict(Name1,Name2; Var low,High);

This procedure helps the user to find the relationship structural constraints. The input are name1 which is the relationship name, name2 which is the entity type name or category name. The output are 'low' which is the min cardinality and 'high' is the maximum cardinality.

PROCEDURE Relt-Partname(Ename; Var Reltpt);

The user can use this procedure to find the relationship names and the participation names upon giving an entity type name. The input is the entity name, the output are all the relationship names and participation names which are stored in a linked list and a pointer to that list is returned.

PROCEDURE Related-Entity-Partname(Reltname; Var Relpt);

This procedure helps the user to find the class names and participation names upon giving a relationship name. The input is a relationship name and the output are all the class names and participation names stored in a linked list and a pointer to that list is returned.

(E). Example: We can take an example in the Figure 10.

```
DEFINE RELATIONSHIP COURSECOMPLETE FROM
STUDENT (completecourse,completestudent),
COURSE (completestuaent,completecourse),
ATTRIBUTES Grade VALUESET GRADELEVEL MIN 5;
```

After parsing the above statement, the relationship name COURSECOMPLETE is inserted into TOTALDBD and the type is 3. Before we start to insert the information in RELATIONDBD, first we check if the class name STUDENT or COURSE is an entity name or category name. If STUDENT is an entity name then we go back to insert information in ENTITYDBD. We insert STUDENT in the first field, 'coursecomplete' in the second field, COURSE in the third field of ENTITYDBD. At the same time, we insert COURSECOMPLETE in the first field, STUDENT in the second field, 'F' in the third field, 'F' in

the fourth field, 'F' in the fifth field, 'coursecomplete' in the sixth field, 'studentcomplete' in the seventh field, 'E' in the eighth field, '0' in the ninth field, 'max number' in the tenth field of RELATIONDBD. The other related information about COURSE is inserted in the RELATIONDBD in the same way. The attribute information are inserted into ATTRIBUTDBD in the same way we mentioned before.

The RELATIONDBD in the university database is as follows:

Table 8. RELATIONDBD

Relation-name	Relation-from	SPEC-IFIC	PD	DLT	Parti-cipname1	Parti-cipanme2	Type	Min	Max
COURSE-COMPLETE	COURSE	F	F	F	student-complete	course-complete	E	0	M
COURSE-COMPLETE	STUDENT	F	F	F	course-complete	student-complete	E	5	M
DEPT-COURSE	DEPART-MENT	F	F	F	dept	course-offered	E	1	1
DEPT-COURSE	COURSE	F	F	F	course-offered	dept	E	0	M

Note: M means maximum number in PASCAL.

Table 5. ENTITYDBD

EntityName	Particpname	RelatedTo
DEPARTMENT		
DEPARTMENT	courseoffered	COURSE
COURSE		
COURSE	student-complete	STUDENT
STUDENT		
STUDENT	course-complete	COURSE

The above are illustrations of dictionary files. The data files are created using the same data stucture. After parsing the data file definition, the data file is created. They are empty until the DBMS user defines and executes the transaction to load data into database.

9. DICTIONARY FILE (DICTIONARY)

(A). Dictionary file name: DICTIONARY.

This file is the summarization of all the indexed sequential files and hash files. This file includes all the dictionary file names, attribute names, type, location, size and key specification. When a user defines a query or transaction, the database may refer this file to do type or key checking.

(B). File contents: This file contains six fields:

The first field name is 'FileName' which is the name of the file and the type is 'S'.
The second field name is the 'AttributeName' which is the attribute name and the type is 'S'. The attribute names are ordered alphabetically.
The third field name is the 'Type' which is the type of the second field and the type is 'S' but occupies one byte.
The fourth field name is the 'Pos' which is the start position of the attribute name and the type is 'I'.
The fifth field name is the 'Size' which is the size of the attribute name and the type is 'I'.
The sixth field name is the 'keyno' which is the order of the key specification and the type is 'I'.

(C). Function: All the indexed sequential file and hash file information are contained in this file. This file could be called the dictionary of the dictionary files.

There is one function procedure based on this file:

PROCEDURE Field-type(FileName; var valptr);

The user can use this procedure to get all the field names, type, position, size and keyno upon giving a file name. The input is the file name and the output are the field name,

type, position, size and keyno stored in a linked list.

(D). Example: In this thesis, the DICTIONARY includes the above eight dictionary files and some data files. The following is only part of this file.

Table 9. DICTIONARY

FileName	AttributeName	Type	Pos	Size	Keyno
VALUESTDBD	Max	R	21	4	0
VALUESTDBD	Min	R	17	4	0
VALUESTDBD	Type	S	16	1	0
VALUESTDBD	Valuestname	S	1	15	1
ENTITYDBD	EntityName	S	1	15	2
ENTITYDBD	EntityPrimary	C	1	30	1
ENTITYDBD	ParticipName	S	16	15	0
ENTITYDBD	RelatedTo	S	31	15	0

(10). FILETYPE file (FILETYPE)

(A). Dictionary file name: FILETYPE.

This file contained all the dictionary file names and its type. The type of the indexed sequential file is 'I' and the type of the hash file is 'R'. The user may refer this file to do type checking.

(B). File contents: This file contains two fields:

The first field name is the 'FileName' which is the name of the file and the type is 'S'.

The second field name is the 'Type' which is the type of the

file and the type is 'S' but occupies only one byte.

(C). Function: this file contains the name and the type of the files. The user or the system can use this file to check the type of the file.

(D). Example: In this thesis, this file contains all the name and the type of the dicitonary files and some data files. Following is only part of the file.

Table 10. FILETYPE

FileName	Type
ATTRIBUTDBD	I
CATEGRELDBD	I
CATEGORYDBD	I
ENTITYDBD	I
EXPLICTDBD	I
RELATIONDBD	I
TOTALDBD	I
VALUESTDBD	I

CHAPTER 5

CONCLUSION

The purpose of this paper has been to discuss and illustrate the ECR model, the schema definition facilities based on the ECR model and the data dictionary files to store the information required in the database design process and for implementing a database system based on the ECR model. The ECR model provides a semantically rich means of modeling user applications into a database conceptual schema. Because the ECR model is a semantic data model, the data dictionary is more complicated than the other dictionaries based on other models. The data dictionary should contain all the schema objects and divides them into entity types, categories and relationships. Each schema name has relationships with others. Some attributes belong to some entity type, category or relationship. Each attribute has a valueset name associated with it. A relationship relates two or more entity types or categories. All these characterictics are included in the data dictionary files. So these files can be referred to whenever a DBMS user defines a query or transaction on the database. The function procedures help the users or DBMS

software to get the data structures, data format and information from the data dictionary files .

APPENDIX

PROGRAM DESCRIPTION

The programs in this paper are divided into three parts. They are:

1. Program in creating dictionary files.
2. Program for data definition language parser.
3. Program to create dictionary function procedures.

The algorithms and example of these three parts will be illustrated as follows:

1. Program in creating dictionary files.

The dictionary files are the files which includes all the information about the data, i.e, data about the data. Before creating the dictionary file, first we should consider what should be included in each file. In this paper, the data dictionary files are based on the ECR model. So, the ten data dictionary files are created based on the schema and each file contains all the characterictics about that data definition.

(A). The algorithms of creating each file are as follows:

```
BEGIN
   Build key chain;
   Build field chain;
   Build file chain;
   CreateFile(file chain)
END;
```

(B). Data structure used.

Both key chain and field chain are ChainType.

```
ChainType=^List;
List=RECORD
       Pos,Size: Integer;
       Next:ChainType;
       CASE IsField: Boolean of
       True: (FieldName:String15;
              FieldType:Char;
              KeySpec:Integer)
       False: ;
     END
   END;
```

File Chain belongs to FileChainType.

```
FileChainType=^SpecOfFile;
SpecOfFile=RECORD
           FileName:Stringtype;
           Filetype:Stringtype;
           KeyChain:Chaintype;
           FieldChain:Chaintype
           END;
```

(C). Initialize statements for creating dictionary files.

The main program to create dictionary files should include the following initialize statements:

```
InitPHLTable;
CreateInitNoQueue;
IstExist:=False;
IsdExist:=False;
PHLIndex:=1;
```

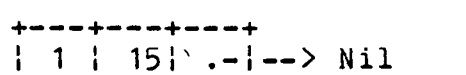
(C). How to run the dictionary files program.

```
$LINK Kinit.obj, Dcreate.obj, Fcreate.obj, OCDFile.obj,
Dbfile.obj, Rwrecord.obj, Utility.obj, FindRec.obj
$Run Kinit
```

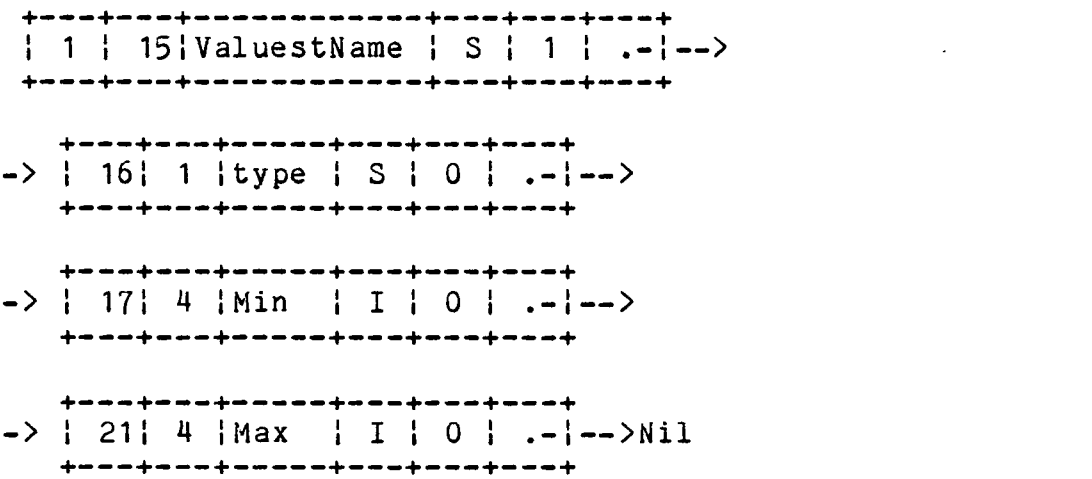
(E). Example

If we create an indexed sequential file named VALUESTDBD, first we decide four fields will be included: Valuestname, Type, Min and Max. The key field will be the first field. The linked list will be as follows:

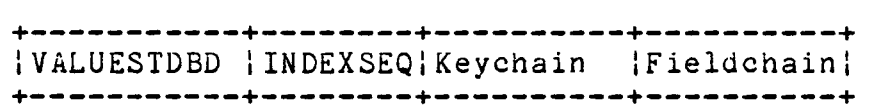
Create key chain



Create field chain



Create filespec



call system procedure CREATEFILE(FileSpec).

2. Program for data definition language parser.

After the data defintion files are created, the data definition records are inserted based on the schema definition. The parser program is used to parse the schema definition language. The input of this program are the schema, they could be input one statement by another or

input as an input file. The output of this program are a set of completely data dictionary files for some database. If any syntex error is detected, an error message is given and the schema definition is aborted. If any error against the naming restrictions in the ECR schema, a warning error is given.

(A). The algorithm to parse the schema definition .

```
BEGIN
   Scan the schema definition;
   Parse the token;
   IF belong to value set definition then
   BEGIN
      Open VALUESTDBD and related files;
      Parse the tokens and collect the data information;
      Link the data records using FieldPtr;
      If no error then INSERTREC(Filename,FieldPtr)
      else write error message;
      Close files
   END
   ELSE IF belong to entity type definition then
   BEGIN
      Open ENTITYDBD and related files;
      Parse the tokens and collect the data information;
      Link the data records using FieldPtr;
      If no error then INSERTREC(Filename,FieldPtr)
      else write error message;
      Close files
   END
   ELSE IF belong to category definition then
   BEGIN
      OPEN CATEGORYDBD and related files;
      Parse the tokens and collect the data information;
      Link the data records using FieldPtr;
      If no error then INSERTREC(Filename,FieldPtr)
      else write error message;
      Close files
   END
   ELSE IF belong to relationship definition then
   BEGIN
      Open RELATIONDBD and related files;
      Parse the tokens and collect the data information;
      Link the data records using FieldPtr;
      If no error then INSERTREC(Filename,FieldPtr)
      else write error message;
      Close files
   END
   ELSE IF belong to file definition then
   BEGIN
      Parse the tokens and collect the data information;
      Create the data files;
   END
END;
```

(B). Data Structure used:

```
StringPtr=^String;
String=RECORD
       Table:Array[1..16] of Char;
       Length:Integer;
       Next:StringPtr;
     END;

FieldPtr=^FieldList;
FieldList=RECORD
            Pos,Size:Integer;
            FieldType:Char;
            Next:FieldPtr;
            CASE typ: Char of
            'I': (IFieldValue:Integer);
            'R': (RFieldValue:Real);
            'L': (LFieldValue:Double);
            'S': (SFieldValue:StringPtr);
            'C': (CFieldValue:Char)
           END
          END;
```

(C). Initialize statements for the driver of parser program.

The main program(driver) to call the DDL parser program must include the following initialize statements:

```
PHLIndex:=1;

IsdExist:=False;

IstExist:=False;

FmsError:=False;

DdlError:=False;

Warning:=False;
```

```
CreateUnitNoQueue;
GI:=0;
Temp-type:=0;
Temp-num:=0;
temp-index:=0;
```

(D). Input file for the DDL parser program.

```
DEFINE VALUESET SOCSECNUM AS INTEGER 0:9999999$
DEFINE VALUESET SALARIES AS REAL 0.0:999999.99$
DEFINE VALUESET LOCATIONCODES AS {MN01,OH20,AZ05}$
DEFINE VALUESET STATES AS {WYOMING,TEXAS,ALASKA,ALABAMA}$
DEFINE VALUESET PEOPLENAMES AS STRING 15$
DEFINE VALUESET PHONENUM AS INTEGER RANGE 0:9999999$
DEFINE VALUESET PROJECTNAMES AS NAME OF PROJECT:  STRING$
DEFINE VALUESET PROJECTNUMS AS NUMBER OF PROJECT:  INTEGER$
DEFINE VALUESET DIVISIONNAMES AS NAME OF DIVISION:  STRING$
DEFINE VALUESET DIVISIONNUMS AS INTEGER$
DEFINE VALUESET STARTDATES AS STRING 8$
DEFINE VALUESET WEEKLYHOURS AS REAL 0.99999:39.99999$
DEFINE VALUESET SCIENTISTGRADES AS {S1,S2,S3,S4,S5,E,F}$
DEFINE VALUESET TECHNICIANGRADE AS
{T1,T2,T3,T4,T5,T6,T7,T8,T9}$
DEFINE VALUESET TRADEUNIONS AS {UAW,TEAMSTERS}$
```

```
DEFINE VALUESET ADDRESSES AS STRING 30$

DEFINE ENTITYTYPE DIVISION ATTRIBUTES
Name VALUESET DIVISIONNAMES UNIQUE,
Number VALUESET DIVISIONNUMS UNIQUE,
Location VALUESET LOCATIONCODES MIN 1 MAX 20;$

DEFINE ENTITYTYPE EMPLOYEE ATTRIBUTES
Name VALUESET PEOPLENAMES,
Ssn VALUESET SOCSECNUM UNIQUE,
Hphone VALUESET PHONENUM MIN 0 MAX 1,
Ophone VALUESET PHONENUM MIN 0 MAX 1,
Address VALUESET ADDRESSES MIN 1 MAX 1;$

DEFINE ENTITYTYPE PROJECT ATTRIBUTES
Name VALUESET PROJECTNAMES UNIQUE,
Number VALUESET PROJECTNUMS UNIQUE,
Locations VALUESET LOCATIONCODES,
Pstart VALUESET STARTDATES;$

DEFINE CATEGORY CDIVISION=DIVISION$

DEFINE CATEGORY FULLTIMEMPLOYEE FROM EMPLOYEE ATTRIBUTES
Sal VALUESET SALARIES;$

DEFINE CATEGORY CONSULTANT FROM EMPLOYEE ATTRIBUTES
```

```
Payscale VALUESET SALARIES;$

DEFINE CATEGORY SCIENTIST FROM EMPLOYEE ATTRIBUTES
Sgrade VALUESET SCIENTISTGRADES,
Societies VALUESET TECHSOCIETIES;$

DEFINE CATEGORY TECHNICIAN FROM EMPLOYEE ATTRIBUTES
Tgrade VALUESET TECHNICIANGRADES,
Tradeunion VALUESET TRADEUNIONS;$

DEFINE RELATIONSHIP ASSIGNED FROM
FULLTIMEMPLOYEE (division,employees) MIN 1 MAX 1,
DIVISION (employees,division),
ATTRIBUTES Sal VALUESET SALARIES;$

DRFINE RELATIONSHIP CONTROLS FROM
DIVISION (projects,division),
PROJECT (division,projects) MIN 1 MAX 1;$

DEFINE RELATIONSHIP MANAGES FROM
FULLTIMEMPLOYEE (proj-managed,manager) MIN 0 MAX 1,
PROJECT (manager,proj-managed) MIN 1 MAX 1,
ATTRIBUTES Mstart VALUESET STARTDATES;$

DEFINE RELATIONSHIP WORKS-ON FROM
EMPLOYEE (projects,employees),
```

```
PROJECT (employees,projects),
ATTRIBUTES Hours VALUESET WEEKLYHOURS;$

DEFINE FILE EMPLOYEE TYPE INDEXSEQ FIELDS
NAME 1 STRING 15 T,
SSN 16 INTEGER 4 T,
HPHONE 20 STRING 12 F,
OPHONE 32 STRING 12 F,
ADDRESS 44 STRING 30 F
PRIMARY KEY 16 4,
SECONDARY KEY 1 15
CORRESPONDS TO ENTITYTYPE EMPLOYEE;$

DEFINE FILE DIVISION TYPE INDESXEQ FIELDS
DNAME 1 STRING 15 F,
DNUMBER 16 INTEGER 4 T,
DLOCATION 20 STRING 15 F
PRIMARY KEY 16 4;$

DEFINE FILE PROJECT TYPE INDEXSEQ FIELDS
PNAME 1 STRING 15 T,
MGRSSN 16 INTEGER 4 T,
DNUMBER 20 INTEGER 4 T,
PNUMBER 24 INTEGER 4 T,
PLOCATION 28 STRING 15 F,
```

```
PSTART 43 STRING 15 F
PRIMARY KEY 24 4,
SECONDARY KEY 1 15,16 4,20 4
CORRESPONDS TO ENTITYTYPE PROJECT
JOIN ATTRIBUTES MGRSSN REFERS SSN FULLTIMEMPLOYEE,
PDNUMBER REFERS DNUMBER DIVISION;$

DEFINE FILE FULLTIMEMPLOYEE TYPE INDEXSEQ FIELDS
FSSN 1 INTEGER 4 T,
FDNUMBER 5 INTEGER 4 T,
SAL 9 REAL 4 F
PRIMARY KEY 1 4,
SECONDARY KEY 5 4
CORRESPONDS TO CATEGORY FULLTIMEMPLOYEE
JOIN ATTRIBUTES FSSN REFERS SSN EMPLOYEE
FDNUMBER REFERS DNUMBER DIVISION;$

DEFINE FILE CONSULTANT TYPE INDEXSEQ FIELDS
CSSN 1 INTEGER 4 T,
PAYSCALE 5 REAL 4 F
PRIMARY KEY 1 4
CORRESPONDS TO CATEGORY CONSULTANT
JOIN ATTRIBUTES CSSN REFERS SSN EMPLOYEE;$

DEFINE FILE SCIENTIST TYPE INDEXSEQ FIELDS
```

```
SSSN 1 INTEGER 4 T,
SGRADE 5 INTEGER 4 F,
SOCIETIES 9 STRING 15 F
PRIMARY KEY 1 4
CORRESPONDS TO CATEGORY SCIENTIST
JOIN ATTRIBUTES SSSN REFERS SSN EMPLOYEE;$

DEFINE FILE TECHNICIAN TYPE INDEXSEQ FIELDS
TSSN 1 INTEGER 4 T,
TGRADE 5 STRING 5 F,
TRADEUNION 10 STRING 10 F
PRIMARY KEY 1 4
CORRESPONDS TO CATEGORY TECHNICIAN
JOIN ATTRIBUTES TSSN REFERS SSN EMPLOYEE;$

DEFINE FILE WORKS-ON TYPE INDEXSEQ FIELDS
WSSN 1 INTEGER 4 T,
WPNUMBER 5 INTEGER 4 T,
SSNPNUM 1 STRING 8 T,
HOURS 9 REAL 4 F
PRIMARY KEY 1 8,
SECONDARY KEY 1 4,5 4
CORRESPONDS TO RELATIONSHIP WORKSON
JOIN ATTRIBUTES WSSN REFERS SSN EMPLOYEE
```

```
WPNUMBER REFERS PNUMBER PROJECT;$
```

(E). How to run DDL parser program.

```
$LINK    Test.obj,DDLParser.obj,Fcreate.obj,    Utility.obj,
OCDFile.obj, Dbfile.obj, Rwrecord.obj, Findrec.obj
$ASSIGN INPUTFILE PAS$INPUT
$RUN Test
```

(F). Example:

For the following schema definition:

```
DEFINE RELATIONSHIP COURSECOMPLETE FROM
STUDENT(completecourse,completestudent),
COURSE(completestudent,completecourse),
ATTRIBUTE Grade VALUESET GRADELEVEL MIN 5;
```

After parsing the above definition then

```
BEGIN
   OpenFile(TOTALDBD,WriteOnly);
   Relationship name COURSECOMPLETE is inserted.
   StringPtr:
   +--------------+-----+-----+
   |COURSECOMPLETE| 14  |  .--|-->Nil
   +--------------+-----+-----+
   FieldPtr:
   +---+---+---------+--+----+    +---+---+---+---+----+
   | 1 | 15|Stringptr|S | .-|--->|16 | 4 | 3 | I | .-|->Nil
   +---+---+---------+--+----+    +---+---+---+---+----+
   InsertRec(TOTALDBD,FieldPtr);
   Participation names :completecourse and completestudent
   are inserted.  (build linked list as above )
   If no error THEN
   BEGIN
   InsertRec(TOTALDBD,FieldPtr1);
   InsertRec(TOTALDBD,FieldPtr2);
   END
   ELSE write error message;
   Attribute name Grade is inserted.
   IF no error THEN
   InsertRec(TOTALDBD,FieldPtr3)
   ELSE write error message;
   CloseFile(TOTALDBD);
   IF STUDENT is an entity type THEN
   BEGIN
     OpenFile(ENTITYDBD,WritOnly);
     Insert STUDENT, completecourse, COURSE into ENTITYDBD;
     Build linked list (FieldPtr4);
     IF no error THEN
     InsertRec(ENTITYDBD,FieldPtr4)
     ELSE write error message;
     CloseFile(ENTITYDBD)
   END
   ELSE IF STUDENT is an category THEN
   BEGIN
     OpenFile(CATEGRELDBD,WritOnly);
     Insert STUDENT, completecourse,COURSE into CATEGRELDBD;
     Build linked list (FieldPtr4);
     IF no error THEN
     InsertRec(CATEGRELDBD,FieldPtr4)
     ELSE write error message;
     CloseFile(CATEGRELDBD);
   END
   ELSE write error message;
```

```
    If COURSE is entity type THEN
    BEGIN
       OpenFile(ENTITYDBD,WritOnly);
       Insert COURSE, completestudent, STUDENT into ENTITYDBD;
       Build linked list (FieldPtr5);
       IF no error THEN
       InsertRec(ENTITYDBD,FieldPtr)
       ELSE write error message;
       CloseFile(ENTITYDBD)
    END
    ELSE COURSE is category THEN
    BEGIN
       OpenFile(CATEGRELDBD,WritOnly);
       Insert COURSE, completestudent, STUDENT into CATEGRELDBD;
       Build linked list (FieldPtr5);
       IF no error THEN
       InsertRec(CATEGRELDBD,FieldPtr)
       ELSE write error message;
       CloseFile(CATEGRELDBD)
    END
    ELSE Write error message;
    OpenFile(RELATIONDBD,WritOnly);
    Insert COMPLETECOURSE, STUDENT, F, F, F, completecourse,
    completestudent, E, 0, Max into RELATIONDBD;
    Build linked list (FieldPtr6);
    Insert COMLETECOURSE, COURSE, F, F, F, completestudent,
    completecourse, E, 0, Max into RELATIONDBD;
    Build linked list (FieldPtr7);
    IF no error THEN
    BEGIN
    InsertRec(RELATIONDBD,FieldPtr6);
    InsertRec(RELATIONDBD,FieldPtr7);
    END
    ELSE write error message;
    CloseFile(RELATIONDBD);
    OpenFile(ATTRIBUTDBD,WritOnly);
    Insert Grade, COURSECOMPLETE, Relationship, GRADELEVEL, F,
    5, 1 into ATTRIBUTDBD;
    Build linked list (FieldPtr8);
    IF no error THEN
    InsertRec(ATTRIBUTDBD,FieldPtr8)
    ELSE write error message;
    CloseFile(ATTRIBUTDBD);
END;
```

(G). Parser error message.

CASE error number of

1:DDERROR: No more token.

2.DDERROR: Schema defined unmatched, should begin with 'DEFINE'.

3.DDERROR: No such schema definition.

4.DDERROR: Should be a constant type.

5.DDERROR: Explicit name should separated by ','.

6.DDERROR: Should be an undefined type.

7.DDERROR: Should be an 'OF' after the undefined type.

8.DDERROR: Should be an 'AS' after VALUESET name.

9.DDERROR: Should be an valueset name.

10.DDERROR: Should have a constant.

11.DDERROR: Should express maximum value.

12.DDERROR: Should express minimum value.

13.DDERROR: Should have a 'VALUESET'.

14.DDERROR: Should have attribute name after entity type.

15.DDERROR: Should have integer, real, string notion.

16.DDERROR: Standard valueset definition unmatched.

17.DDERROR: Undefined name in ENTITYDBD or CATEGORYDBD.

18.DDERROR: Should have an ENTITY name or CATEGORY name.

19.DDERROR: Category definition violation.

20.DDERROR: Should have attributes name after category

name.

21.DDERROR: Should have a category name or 'UNION' after CATEGORY.

22.DDERROR: Should be an integer type.

23.DDERROR: Should be a real type.

24.DDERROR: Should be a string type.

25.DDERROR: Should have FROM after RELATIONSHIP name.

26.DDERROR: Should have RELATIONSHIP name.

27.DDERROR: Should be a '('.

28.DDERROR: Should be a ')'.

29.DDERROR: Should be a ','.

30.DDERROR: Should have ']'.

31.DDERROR: Field name can not exceed 15 characters.

32.DDERROR: Should be a ';'.

33.DDERROR: Should be a FIELDS after file type defined.

34.DDERROR: Should define file type.

35.DDERROR: Should be a TYPE after file name.

36.DDERROR: Should be a boolean type.

37.DDERROR: No such range type in EXPLICIT definition.

38.DDERROR: No this file type, only INDEXSEQ or HASH.

39.DDERROR: Should have PRIMARY KEY.

40.DDERROR: Should have SECONDARY KEY.

41.DDERROR: Should be an integer, real or string.

3. Program to Create Dictionary Function Procedures.

The purpose of writing dictionary function procedures is to help the users or DBMS software to access the dictionary information. Those function procedures could be used by DBMS software or interactive from the terminal. So the user can retrieve data from the data dictionary files very conveniently. There are fifteen function procedures could be used.

(A). Algorithm to create function procedures:

```
BEGIN
   OpenFile(FileName);
   Build a tree pointer which points the data
   record needed;
   FindRec(Filename,Treepoint,found,EndofFile,Buffer);
   IF found THEN ReadRec(Filename,Valueptr,Buffer)
   ELSE the data record is not defined in the
   dictionary file;
   CloseFile(FileName)
END;
```

If you need all the data records after some particular data information, the algorithm is as follows:

```
BEGIN
   OpenFile(FileName);
   Build a tree pointer which points the data record
   needed;
   FindRec(FileName,Treepoint,found,EndofFile,Buffer);
   IF found THEN
   BEGIN
      ReadRec(Filename,Valueptr,Buffer);
      Treepoint:=Nil;  Found:=False;
      While Not EndofFile DO
      BEGIN
         FindRec(FileName,treepoint,found,EndofFile,
         Buffer);
         IF found THEN ReadRec(FileName,Valueptr,Buffer);
      END
   END
   ELSE the data record is not defined;
   CloseFile(Filename)
END;
```

If you try to go back to the beginning of the file after you fine a record, the algorithm is as follows:

```
BEGIN
   OpenFile(FileName);
   Build tree pointer;
   FindRec(FileName,treepoint,found,EndofFile,Buffer);
   IF found THEN
   BEGIN
      ReadRec(Filename,Valueptr,Buffer);
      Treepoint:=nil;  Found:=False;
```

```
        ResetFile(FileName);
        Build another tree pointer;
        FindRec(Filename,treepoint,found,EndofFile,
        Buffer);
        IF found THEN
         Readrec(Filename,Valueptr,Buffer);
           ......................
      END
      ELSE the data record is not defined;
      CloseFile(FileName)
   END;
```

(B). Data Strcture used:

```
TreeType=(OP,CON,DBV);
TreePointer=^TreeNode
TreeNode=RECORD
        left,right:^TreeNode;
        CASE TY: Treetype OF
        OP: (optype:Integer;
             oper:Integer);
        CON: (CASE Contype: Char of
           'I':(Int:Integer);
           'R':(Rea:Real);
           'L':(Long:Double);
           'S':(String:StringPtr);
        DBV:(Attr:String15)
      END;

 ValuePtr=^ValueList;
 ValueList=RECORD
         Pos:Integer;
         Next:ValuePtr;
         Case TYP: CHAR OF
       'I':(Ivalue:Integer);
       'R':(Rvalue:Real);
       'L':(Lvalue:Double);
       'S':(Svalue:StringPtr);
       'C':(Cvalue:Char)
     END;
```

The algorithm to get the information from the valuePtr is as

follows:

```
      BEGIN
         WHILE ValuePtr <> Nil DO
         BEGIN
            CASE Valueptr^.typ OF
            'I': Write(ValuePtr^.Ivalue);
            'R': Write(ValuePtr^.Rvalue);
            'L': Write(ValuePtr^.Lvalue);
            'S': BEGIN
                 FOR I:=1 TO ValuePtr^.Svalue^.length DO
                 Write(ValuePtr^.Svalue^.table[I]);
                 Writeln;
                 END
         P:=ValuePtr;
         ValuePtr:=ValuePtr^.next
         DisPose(P)
      END
    END;
```

(C). Initialize statements for the function procedure program.

The main program to call the function procedures should include the following statements:

```
OpenTypeDic;

PHLindex:=1;

IsdExist:=False;

IstExist:=False;

CreateUnitNoQueue;
```

(D). How to run function procedure program.

$Link Interact.obj, Functionp.obj, Fcreate.obj, Ocdfile.obj, Dbfile.obj, Rwrecord.obj, Findrec.obj, Utility.obj

$Run Interact

(E). Description of function procedure.

(1). FUNCTION SCHEMA-NAME-TYPE (SNAME:STRING25):INTEGER;

PURPOSE: Get the type of an object name.

INPUT: Sname is an object name.

OUTPUT: An integer which is the type of Sname. case Type is

1: Attribute name.

2: Entity type name.

3: Relationship name.

4: Category name.

5: Participation name.

6: Valueset name.

(2). FUNCTION BELONG-TO(NAME1,NAME2:STRING15):BOOLEAN;

PURPOSE: Check if an attribute name belongs to an entity type, category or relationship.

INPUT: Name1 is the attribute name, Name2 is the entity type, category or relationship name.

OUTPUT: A boolean value TRUE or FALSE.

(3). PROCEDURE ENTITY-RELATE-NAME(SNAME:STRING25; VAR RELTPT:VALUEPTR);

PURPOSE: Get all the participation names and the related entity type names or category names upon giving an entity name.
INPUT: Sname is an entity type name.
OUTPUT: All the participation names and related category names stored in a linked list.

(4). FUNCTION DEFINED-ON (CATGNAME,DEFINENAME: STRING25);BOOLEAN;

PURPOSE: Check whether a category name is defined on another category name or entity type name.
INPUT: Categname is a category name, DefineName is an entity type name or another category name which supposed to be defined on.
OUTPUT: A boolean value TRUE or FALSE.

(5). PROCEDURE CATEGORY-RELATE-NAME(CNAME:STRING25; VAR RELTPT:VALUEPTR);

PURPOSE: Get all the participation names and the related classes upon giving a category name.
INPUT: Cname is a category name.
OUTPUT: All the participation names and the related classes stored in a linked list.

(6). PROCEDURE RELATION-RESTRICT(NAME1,NAME2:STRING25; VAR LOW,HIGH:INTEGER);

PURPOSE: Get the relationship structural constraint.
INPUT: Name1 is the relationship name, Name2 is the entity type name or category name.
OUTPUT: Low is the minimum cardinality and High is the maximum cardinality.

(7). PROCEDURE RELT-PARTNAME(ENAME:STRING25; VAR RELTPT:VALUEPTR);

PURPOSE: Get all the relationship names and the participation names upon giving an entity type name.
INPUT: Ename is an entity type name.
OUTPUT: All the relationship names and participation names stored in a linked list.

(8). PROCEDURE ATTRIBUTE-NAME(SNAME:STRING25; VAR ATTPTR:VALUEPTR);

PURPOSE: Get all the attribute names for a given entity type, category or relationship.
INPUT: Sname is an entity type, category or relationship name.
OUTPUT: All the attribute names which belongs to Sname stored in a linked list.

(9). PROCEDURE ATTRIBUTE-RESTRICT(NAME1,NAME2:STRING15; VAR LOW,HIGH:INTEGER);

PURPOSE: Get the restriction number of a given attribute name.
INPUT: Name1 is the attribute name and Name2 is the class name.
OUTPUT: Low is the minimum cardinality and High is the maximum cardinality.

(10). FUNCTION ATTRIBUTE-UNIQUE (NAME1,NAME2: STRING15): BOOLEAN;

PURPOSE: Check if the attribute is unique.
INPUT: Name1 is the attribute name and Name2 is the class

name.

OUTPUT: A boolean value TRUE or FALSE.

(11). PROCEDURE VALUESET-NAME-TYPE(NAME1,NAME2:STRING15; VAR VALPTR:VALUEPTR);

PURPOSE: Get the valueset name, type, min and max value upon giving an attribute name.

INPUT: Name1 is the attribute name and Name2 is the class name.

OUTPUT: Valueset name, type, min and max value stored in a linked list.

(12). PROCEDURE RELATED-ENTITY-PARTNAME(RELTNAME:STRING25; VAR RELTPTR:VALUEPTR);

PURPOSE: Get the class names and participation names upon giving a relationship name.

INPUT: Reltname is a relationship name.

OUTPUT: All the class names and participation names stored in a linked list.

(13). PROCEDURE REPEAT-DEFINEDON(CNAME:STRING25; VAR CATG:VALUEPTR);

PURPOSE: Get all the entity type or category name which a category name defined recursively.
INPUT: Cname is a category name.
OUTPUT: All the entity type or category name stored in a linked list.

(14). PROCEDURE EXPLICIT-VALUE(VNAME:STRING25; VAR EXPTR:VALUEPTR);

PURPOSE: Get all the explicit value upon giving a valueset name.
INPUT: Vname is a valueset name.
OUTPUT: All the explicit values stored in a linked list.

(15). PROCEDURE FIELD-TYPE(FILENAME:STRINGTYPE; VAR FP:VALUEPTR);

PURPOSE: Get all the field name, type, location, size and key number upon giving a file name.
INPUT: FileName is the name of file.
OUTPUT: Field name, type, location, size and key number stored in a linked list.

REFERENCES

[ANSI 77] The ANSI/X3/SPARC DBMS Framework Report of the Study Group on Data Base Management System, Tsichritzis, D and Klug, A., Eds., AFIPS Press.

[CHEN 76] Peter Pin-Shan Chen," The Entity-Relationship Model-Toward a Unifed View of Data", ACM Transaction on Database systems, vol.1,No.1, March 1976. 9-36.

[CURTICE 81] Robert M, Curtice," Data Dictionaries: An Assessment of Current Practice and Problems ", in Proc. 7th Int. Conf. Very Large Database (ACM) Carones. France, 1981). 564-570.

[DATA BASE 77] DATA BASE: A quarterly newsletter of ACM's SIGBDP," The British Computer Society Data Dictionary Systems Working Party Report', Vol. 9,No 2, Fall 1977.

[DATE 77] C.J.DATE, " An Introduction to Database Systems". 3rd edition. Addison-Wesley Publishing Company.p 3.

[ELMASRI 81] Ramez Elmasri," GORDAS: A Data Definition, Query and Update Language for the Entity-Category -Relationship Model of Data', HR-81-250: 17-38, Honeywell Corporate Computer Science Center.

[FRANK 82] Frank W. Allen, Mary E, S.Loomis, and Michael V. Harrnino," The Integrated Dictionary Directory System', Computing Survey. vol. 14 No. 2, June 1982. 245-286.

[HEVNER 80] Alan R. Hevner, " The Use of the Entity-Category -Relationship Model in Database Design', HR-80-264:17-38, Honeywell Corporate Computer Science Center.

[NIJSSEN 77] G.M.Nijssen," Current Issues In Conceptual Schema Concepts " in Architecture and Models in Data Base Management Systems, edited by G.M. Nijssen.North-Holland Publishing Company.

1977. 31-65.

[UHROWCZIK 73] P.P. Uhrowczik," Data Dictionary/Directorids", IBM Syst. J.12,4 (1973). 332-350.

[WEEL 80] James A. Weeldreyer,"The Entity-Category-Relationship Model of Data ",HR-80-250:17-38, Honeywell Corporate Computer Science Center.

[WEEL 80] James A. Weeldreyer,"Structural Aspects of the Entity-Category-Relationship Model of Data', HR-80-251:17-38, Honeywell Corporate Computer Science Center.