

1194

**AN INTEGRATED INTERFACE TO OBJECT
ORIENTED MODEL (OOM) AND RELATIONAL
MODEL (RM) FROM EXTENDED ENTITY
RELATIONSHIP
MODEL (EERM)**

*Dissertation submitted to the
Jawaharlal Nehru University
in partial fulfilment of the requirements
for the award of the degree of*

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE & TECHNOLOGY

by

DIWAN HAUYM KHAN

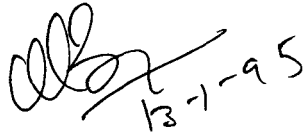
**SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI -110 067
(INDIA)**

JANUARY 1995.

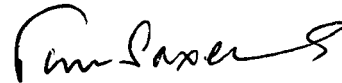
CERTIFICATE

This is to certify that the dissertation titled "*An Integrated Interface between Object Oriented Model (OOM) and Relational Model from Extended Entity Relationship Model (EERM)*" being submitted by **DIWAN HAUYM KHAN** to Jawaharlal Nehru University, New Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology (M.Tech) in Computer Science and Technology is a record of the original work done by him under the supervision of Dr. P.C. Saxena, Professor, School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, during the year 1994-95, Monsoon Semester.

The results reported in this dissertation have not been submitted in part or in full to any other university or institution for the award of any degree or diploma.

Handwritten signature of Prof. K.K. Bharadwaj, dated 13-1-95.

Prof. K.K. Bharadwaj
Dean,
School of Computer &
Systems Sciences,
Jawaharlal Nehru University,
New Delhi.

Handwritten signature of Prof. P.C. Saxena.

Prof. P.C. Saxena
Professor,
School of Computer and
Systems Sciences,
Jawaharlal Nehru University,
New Delhi.

ACKNOWLEDGEMENTS

I bestow my gratitude to my supervisor Dr.P.C.Saxena, Professor, School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi for suggesting me this topic. I very much indebted to him for his personal involvement during the period of my work and his eloquent guidance which has been indispensable in bringing about a successful completion of this dissertation.

My thanks are due to Prof. G.V. Singh and Prof. K.K. Nambiar for their kind support and constant help during the M.Tech programme.

I extend my sincere gratitude to Prof. K.K. Bharadwaj, Dean School of Computer and Systems Sciences, Jawaharlal Nehru University for providing me with the environment and all the facilities required for the completion of my dissertation.

My sincere thanks to my friends Sanjeev Kumar Sinha, D.K.Lobiyal and Manoj Kumar Sarangi for giving me company in the long hours of nights during the period of this project.

Last but not the least, I thank Prabhat Kumar Pandey, Ravi Gupta, Sunil Kumar Verma, Vineet Kumar Sharma and Punam Raju for helping me in all possible ways towards the completion of the dissertation.

Hauym
DIWAN HAUYM KHAN

CONTENTS

CHAPTER-ONE : INTRODUCTION

1 REQUIREMENTS FOR AN INTERFACE

- 1.1 VIEW MODELLING**
- 1.2 VIEW INTEGRATION**
 - 1.2.1 CONFLICT ANALYSIS**
 - 1.2.2 CONFLICT RESOLUTION**
 - 1.2.3 VIEW MERGING**
- 1.3 VIEW TRANSLATION**

CHAPTER-TWO : BASIC CONCEPTS IN EER MODEL, RELATIONAL MODEL AND OBJECT ORIENTED MODEL

- 2.1 EXTENDED ENTITY RELATIONSHIPS MODEL**
 - 2.1.1 ORIGINAL CLASSES OF OBJECTS (ER MODEL)**
 - 2.1.2 EXTENDED CLASSES OF OBJECTS (EER MODEL)**
 - 2.1.3 FUNDAMENTAL EER CONSTRUCTS**
- 2.2 RELATIONAL MODEL**
- 2.3 OBJECT ORIENTED DATA MODEL**
 - 2.3.1 WHAT IS OBJECT ORIENTED ?**
 - 2.3.2 OBJECT ORIENTED DATABASE SYSTEM**
 - 2.3.3 FEATURES OF OBJECT ORIENTED MODEL**
 - 2.3.4 OBJECT ORIENTED PROGRAMMING**

CHAPTER-THREE : VIEW MODELLING, VIEW INTEGRATION IN DATABASE DESIGN

- 3.1 VIEW MODELLING
- 3.2 VIEW INTEGRATION
- 3.3 RULES FOR VIEW INTEGRATION
- 3.4 VIEW INTEGRATION ALGORITHM
- 3.5 EXAMPLES ILLUSTRATING THE VIEW INTEGRATION
ALGORITHM

CHAPTER-FOUR : AN INTERFACE BETWEEN EER MODEL TO RELATIONAL MODEL

- 4.1 VIEW TRANSLATION
- 4.2 ALGORITHM FOR VIEW TRANSLATION
 - 4.2.1 TRANSFORMATION RULES FOR EER MODEL
- 4.3 EXAMPLE ILLUSTRATING THE VIEW TRANSLATION
ALGORITHM

CHAPTER-FIVE : AN INTERFACE FROM EER MODEL TO OBJECT ORIENTED MODEL

- 5.1 DEVELOPMENT OF RULES FOR THE TRANSLATION
- 5.2 EXAMPLE ILLUSTRATING THE TRANSLATION FROM EER
MODEL TO OBJECT ORIENTED MODEL
- 5.3 THE OBJECT CLASS DECLARATION IN OBJECT ORIENTED
LANGUAGE C++

CHAPTER-SIX : CONCLUSION

REFERENCES

CHAPTER-ONE

INTRODUCTION

Object Oriented Technology is fast emerging as the favorite of software designers. Most of the software designer are shifting from the relational approach to object oriented approach due to its several advantages over relational approach. Some of them are as follows:

- (i) In OODB the complex structural objects can be artificially represented.
- (ii) In relational Database properties of object cannot be modeled.
- (iii) In relational Database operational semantics of complex structured object is not expressible.
- (iv) OODB supports inheritance, modularity, easy upgrade and various other features.

Extended entity relationship approach is one of the most popular and fundamental approach which have been followed by designers throughout the globe. "Although the OODB Technology is emerging but the building of fundamental blocks are extended entity relationships with modifications to support the OODB philosophy.

At the same time, logical database design research is at its zenith. Logical database design is concerned with determining the structure of a database independent of implementation consideration. Its purpose is to

transform real world requirements into a good logical database. There are number of alternatives for accomplishing the transformations. The OODB technology has get impetus in research and development from the same logical database design.

1. Requirement for an interface

We require a database system that is efficient for operations on individual data objects without losing functionality provided by a relational DBMS. Typically our database operations cannot be expressed as a single query in SQL or another high-level relational language and must be decomposed into many simple "object-oriented" queries. The queries are invoked by programs rather than directly by an end-user, response time must be at least of order of the magnitude faster than conventional DBMS. We might get the performance required from an "object-oriented" database system but we would lose the powerful relational facilities. Instead of using either an object-oriented language or the high-level SQL languages. If we make interface which is the topic of this dissertation, it will give the best characteristic of both kinds of database in one system.

In the dissertation the adopted approach is towards developing a common platform for the interface to the extended entity relational model with the object oriented model and relational model. This interface will convert EERM to RM and OOM. This is accomplished in following two steps:

STEP 1 : Building an interface between extended entity relationship model and relational model.

STEP 2 : Building an interface between extended entity relationship model and object oriented model.

Accomplishment of first step requires View Modeling, View Integration and View Translation.

1.1 View Modeling

In view modeling, the informations and processing requirement of each of the group of users are analyzed and modeled using extended entity relation data model.

1.2 View Integration

View Integeration is done for the EERM. This gives a global view to user as a part of the view integration the following are studied.

1.2.1 Conflict Analysis

The main goal of this step is to detect and resolve all types of inconveniences that exists in representing same classes of concepts in the two view. During conflict analysis several types of conflicts are discovered.

1.2.2 Conflict Resolution

In this step assertion specifying the precise relationship between the domain of pair of entities and relationship from different view are generated. Then in order to resolve the conflicts, the integration of entity types and integration of relationship type are made

1.2.3 View merging

Once the assertion specifying the correspondence between entities of the two views are generated, similar entities and relationships are integrated and unrelated entities and relationships are simply merged in partial integrated view.

1.3 View Translation

In this step the view is transformed from EER model into relational model. A relational schema is created that include all the attributes that can have only atomic attribute values. By analyzing the functional dependencies in valid relational schema, the set candidate key is chosen as a primary key. Foreign key attribute on relationship attribute will be added during subsequent steps so that the translation process is carried out.

The next step is to develop interface from EERM to OOM. Here all properties of object oriented model are to be studied and the ternary associations are converted into binary associations. Further, properties like inheritance, aggregation and polymorphism will be incorporated.

An object oriented design is based on entity and class relationships. This method concentrates on establishing relationships between entities, and between classes and on representing and refining the relationship. It also provides information and specific procedure in order to identify classes, objects, attributes and operations.

The final phase involves developing interface for the proposed system. This involves integrating the interfaces developed in earlier stages.

CHAPTER-TWO

BASIC CONCEPTS IN EER MODEL, RELATIONAL MODE AND OBJECTED ORIENTED MODEL

This chapter briefly discusses the basic concepts used in extended entity relationship model, relational model and object oriented model.

2.1 EXTENDED ENTITY RELATIONSHIPS MODEL

The entity-relationship approach initially proposed by Chen, although modified and extended by others, still remains the premier model for conceptual design. It is used to represent information in terms of entities, their attributes, and associations among entity occurrences called relationships.

2.1.1 Original Classes of objects (ER model)

Initially, Chen proposed three classes of objects: entities, attributes, and relationships (Figure 2.1). Entity sets were the principal objects about which information was to be collected and usually denoted a person place, thing, or event or informational interest. Attributes were used to detail the entities by giving them descriptive properties such as name, color, and weight. Finally, relationship (formerly called relationship sets)

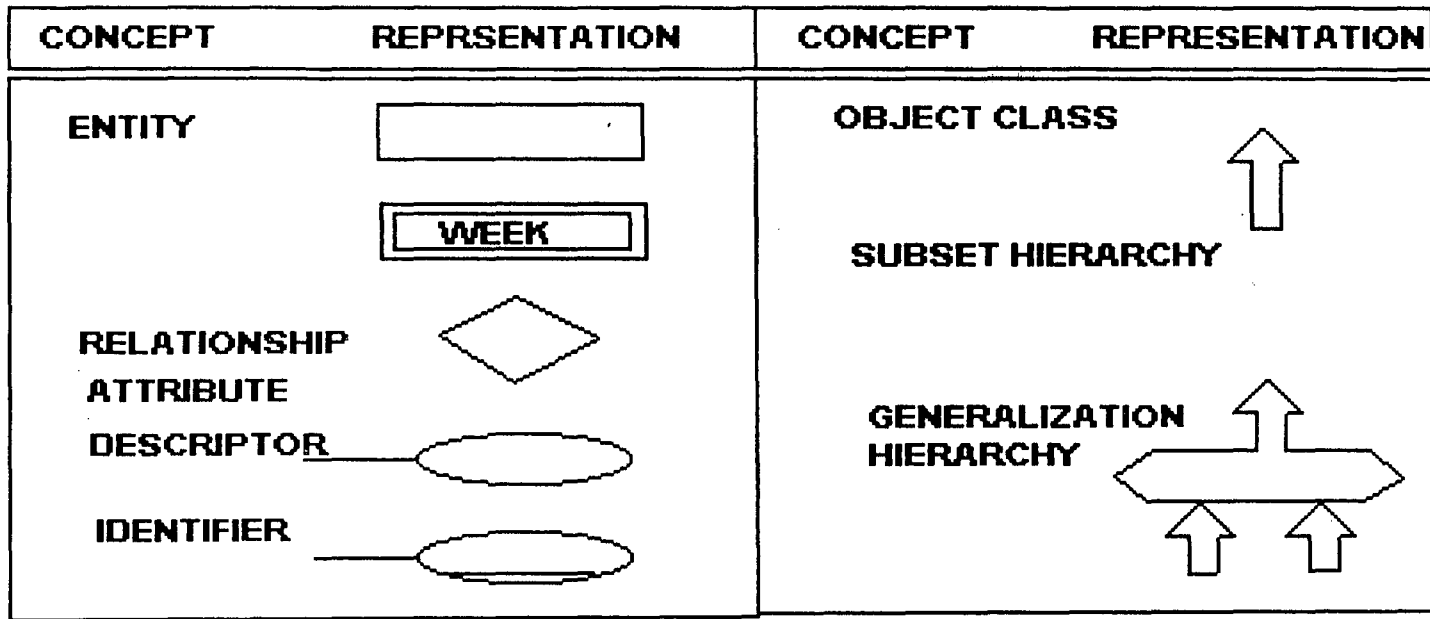


Fig. 2.1

Fig. 2.2

EXTENDED ER (EER) MODEL REPRESENTATION

represented by real-world associations among one or more entities.

There are two types of attributes: identifiers and descriptors. The former is used to uniquely distinguish among the occurrences of an entity, whereas the latter is used to describe an entity occurrence. Entities can be distinguished by the "strength" of their identifying attributes. Strong entities have internal identifiers that uniquely determine the existence of entity occurrence. Weak entities derive their existence from the identifying attributes (sometimes called external attributes) of one or more "parent" entities. Relationships have semantic meaning which is indicated by the connectivity between entity occurrences (one to one, one to many, and many to many), and the participation in this connectivity by the member entities may be either optional or mandatory. For example, the entity "person" may or may not have a spouse. Finally, each of the entities may have one or more synonyms associated with it. The diagrams for representing entities, relationships, and attributes are shown in Figure 2.1.

2.1.2 Extended classes of objects (EER model)

The original ER model has long been effectively used for communicating fundamental data and relationship definitions with the end user. Using the ER model as a conceptual schema representation, however, has proved difficult because of the inadequacy of the initial modeling constructs. View integration, for example, requires the use of abstraction concepts such as generalization [Navathe et al. 1986]. Data integrity involving null attribute values requires defining relationships

such that a null set on either side of the relationship is either allowed or disallowed. Also, certain relationship of degree higher than 2 (binary) may be present and are awkward (or incorrect) when represented in binary form. The extended ER model provides simple representations for these commonly used concepts and is compatible with the simplicity of the original ER model.

The introduction of the category abstraction into the ER model resulted in two additional types of objects: subset hierarchies and generalization hierarchies [Navathe and Cheng 1983; Elmasri et al. 1986]. The subset hierarchy specifies possibly overlapping subsets, while the generalization hierarchy specifies strictly non overlapping subsets. Both subset objects will transform equivalently to a relational data model scheme, but they will differ significantly with regard to update (integrity) rules.

(i) Subset Hierarchy Definition

An entity E_1 is a subset of another entity E_2 if every occurrence of E_1 is also an occurrence of E_2 .

A subset hierarchy is the case in which every occurrence of the generic entity may also be an occurrence of other entities that are potentially overlapping subsets (Figure 2.2). For example, the entity EMPLOYEE may include "employees attending college," "employees who hold political office," or "employees who are also shareholders" as specialized classifications.

(ii) Generalization Hierarchy Definition

An entity E is generalization of the entities E_1, E_2, \dots, E_n if each occurrence of E is also an occurrence of one and only one of the entities E_1, E_2, \dots, E_n .

A generalization hierarchy occurs when an entity (which we call the generic entity) is partitioned by different values of a common attribute (Figure 2.2). For example, the entity EMPLOYEE is a generalization of ENGINEER, SECRETARY and TECHNICIAN. The generalization object (EMPLOYEE) is called an "IS-A" exclusive hierarchy because each occurrence of the entity EMPLOYEE is an occurrence of one and only one of the entities ENGINEER, SECRETARY and TECHNICIAN.

2.1.3 Fundamental EER Constructs

The following classification of EER constructs is defined to facilitate development of a concise and easy to understand EER diagram.

(i) Degree of a Relationship

The degree of relationship is a number of entities associated with the relationship. An n -ary relationship is of degree n . Unary, binary, and ternary relationships are special cases in which the degree is 1, 2, and 3 respectively. This is indicated in Figure 2.3.

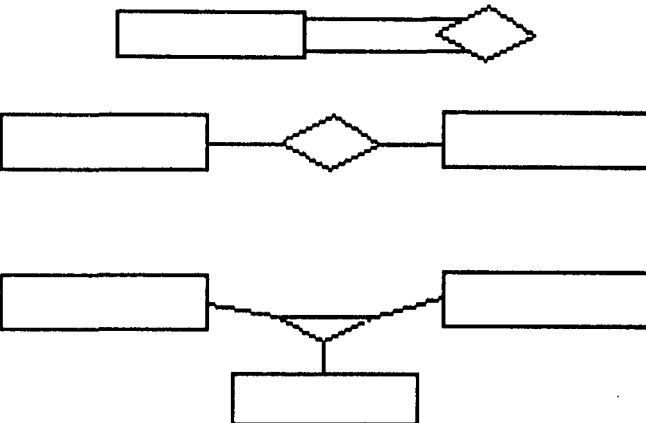


CONCEPT	REPRESENTATION
DEGREE UNARY BINARY TERNERY	
CONNECTIVITY 1 : 1 1 : n m : n	
MEMBERSHIP CLASS mandatory optional	

Fig. 2.3 FUNDAMENTAL EER CONSTRUCTS RELATIONSHIP TYPES.

(ii) Connectivity of a Relationship

The connectivity of a relationship specifies the mapping of the associated entity occurrence in the relationship. Values for connectivity are either "one" or "many". For a relationship among entities $E_1, E_2, \dots, E_i, \dots, E_m$ a connectivity of "one" for entity E_i means that given all entities except E_i , there is at most one related entity occurrence of E_i .

The actual number associated with the term "many" is called the cardinality of the connectivity. Cardinality may be represented by upper and lower bounds. Figure 2.3 shows the basic constructs for connectivity; one to one (unary or binary relationship), one to many (unary to binary relationship), and many to many (unary or binary relationship). The shaded area in the unary or binary relationship diamond represents the "many" side, while the unshaded area represents the "one" side [Reiner et al. 1985].

We use an n -sided polygon to represent n -ary relationships for $n > 2$ in order to show explicitly each entity associated with the relationship to be either "one" or "many" related to the other entities. Each corner of the n -sided polygon connects to an entity. A shaded area denotes "many" and an unshaded corner denotes "one". The ternary relationship (see in Figure 2.3) illustrates this type of association, which is much more complex than either a unary or binary relationship.

(iii) Membership Class in a Relationship

Membership class specifies whether either the "one" or "many" side in a relationship is mandatory or optional. If an occurrence of the "one" side entity must always exist for the entity to be included in the system, then it is mandatory. When an occurrence of that entity need not exist, it is considered optional. The "many" side of a relationship is similarly mandatory if at least one entity occurrence must exist, and optional otherwise. The optional membership class, defined by a "O" on the connectivity line between an entity and a relationship, is shown in Figure 2.3. Membership class is implied by existence dependency in the real-world system; for example, an independent (strong) entity associated with a dependent (weak) entity cannot be optional, but the weak entity may be optional. Weak entities are sometimes depicted with a double-bordered rectangle (Figure 2.1).

(iv) Object class of entities and relationships

The basic objects are the n-ary relationships with their associated entities. Object resulting from abstraction are the generalization hierarchy and the subset hierarchy (Fig. 2.1 and 2.2). The generalization hierarchy implies that the subsets are a full partition, such that the subsets are disjoint and their combination makes up the full set. The subset hierarchy implies that the subsets are potentially overlapping.

2.2 RELATIONAL MODEL

The relational model of data was introduced by Codd [1970]. The relational model represents the data in database as a collection of relations. Informally, each relation resembles a table or to some extent, a simple file.

When a relation is thought of as a **table** of values, each row in the table represents a collection of related data values. These values can be interpreted as a fact describing an entity or a relation instance.

A **relation** (or **relation instance**) r of the relation schema $R (A_1, A_2, \dots, A_n)$ also denoted by $r(R)$, is set of n -tuples $r = \{t_1, t_2, \dots, t_m\}$. Each n -tuple t is an ordered list of n values $t = \langle V_1, V_2, \dots, V_n \rangle$, where each value V_i , $1 \leq i \leq n$, is an element of $\text{dom}(A_i)$ or is a special **null** value. The terms **relation intention** for the schema R and **relation extension** for a relation instance $r(R)$ are also commonly used.

2.3 OBJECT ORIENTED DATA MODEL

Object-oriented modeling design is a technique for new way of thinking about problems using model organized around real-world concepts. The fundamental construct is object, which combines both data structure and behavior in a single entity. Object-oriented models are useful for understanding problem, communication with application experts, modeling enterprises, documentation, and designing programs and database.

2.3.1 What is Object-Oriented ?

The term "Object-Oriented" mean that we organize software as a collection of discrete objects that incorporate data structure and behavior.

The basic idea behind the object-oriented approach is very simple. We perceive the world around us as a variety of objects. When we look at plant, we see a plant, a mass of individual atoms. We can divide the plant into leaves, flower, stems and root, but we still see those items as units, as objects.

2.3.2 Object Oriented Database System

The unit of storage in OODB is the object-class. It requires that encapsulated combination of data and methods (i.e. procedure) must be managed as an integral unit. The goal of the OODBS is to provide logical and physical independence of the objects from application object oriented programs that form systems. Thus multiple systems can query, retrieve and update objects concurrently and independently while the OODBS manages, all details of object consistency, concurrency, security, recovery, and integrity.

Definition of Object Oriented Database Systems

Any object oriented database system should satisfy two criteria, it should be a database management system, and it should be an object oriented system. The first criteria translates into six features, persistence, secondary storage management, data sharing (concurrency), data reliability transaction management and recovery, adhoc query facility, and schema

modification. The second translates into eight features. The type/class, encapsulation/data abstraction, inheritance, polymorphism/late binding, computational completeness, object identity, complex object, and extensibility.

2.3.3 Features of Object Oriented Model

Following are basic features of the Object Oriented Model:

(i) Methods and Messages

Only the methods of an object have access to its state, and a method can only be invoked by sending the object a message. The distinction between a message and a method is subtle but important. Since a method is part of an object and not a global entity, there is no problem with two deferent objects having a same method name.

(ii) Polymorphism

It is quite common in object oriented systems to code multiple classes of an object that respond to the same messages. The ability of different objects to respond differently to the same messages is known as polymorphism.

(iii) Classification

Classification means that objects with the same data structure (attributes) and behavior (operations) are grouped into a class.

(iv) Inheritance

Inheritance is the sharing of attributes and operations among classes based on hierarchical relationship. A class can be define broadly and refined into successively finer subclasses. Each subclass incorporates, or inherits, all the properties of its superclass and adds its own unique properties. The properties of the superclass need not be repeated in each subclass. For example, if we are writing a banking application, we would define a Saving Account object that is just like the existing banking Banking Account object but with a few extras. Saving Account will inherit all Bank Account's state and methods.

(v) Encapsulation

An object consists of an encapsulated representation (state) and set of messages (operations or procedures) that can be applied to that object. Encapsulation is technical name for information hiding. Instead of organizing programs into procedures that share global data, the data is packaged with the procedures that access that data. This concept is often called data abstraction or modular design.

(vi) Class Versus Instance

In object oriented languages, the definition of type is often called a class. A class definition defines both the instance variables (stage or representation) and the methods (operation) for objects of that class.

(vii) Message Passing

The use of the word message versus procedure suggests a looser connection between the object and its user. In most cases, the messages are organized into public and private categories. Private messages, however, can only be executed by the object itself. These are not available (visible) to outside users.

(viii) Static and Dynamic Binding

Static binding means that all variables are bound to types at compilation time. Dynamic binding waits until execution time before binding variables to types. An advantage to dynamic binding is that a variable may represent one of the many different objects from different classes.

(ix) Object Identity

In object oriented database systems, the concept of object identity plays an important role among others. It is used as a surrogate to distinguish an object from all others. Whenever a new entity instance is created, the system will automatically generate an internal identifier "oid" for it and this identifier will not be reused for other entity instances. Although objects in an OODB can be uniquely identified by object identifiers, the concept of key which is used in relational DBMS as a link of object identifier still has its place in OODBS.

2.3.4 Object Oriented Programming

Object oriented programming encourages code reuse rather than reinvention. It encourages prototyping and code polishing. It rewards the development of generic function. Object oriented programming enables you to create software that can be readily comprehended and shared with others. Some of the popular OOP languages are SIMULA, Ada, C++ etc. Among these C++ has received a tremendous attention.

CHAPTER-THREE

VIEW MODELING, VIEW INTEGRATION IN DATABASE DESIGN

3.1 VIEW MODELING

View modeling is the process of eliciting user view by analyzing the user requirements and the information needs in an organization. The user view can be defined as the perception of user about what a database should contain [Navathe, 1980]. According to Navathe and Sehkolnick, there are two major tasks in the view modeling.

- (i) Extracting from user or from person incharge of application development, the relevant part of the real world.
- (ii) Abstracting this information in a form that completely represents the user view so that it can be subsequently used in the design. The view modeling methodology is described below in a step-by-step manner:

STEP 1: Classification of Entities and Attributes

The first step in view modeling is to identify entities and their corresponding attributes. It is not easy to define entities attributes and

relationship constructs, and also to distinguish them in the database. The following are the guidelines for classifying entities and attributes.

- (i) Entities have descriptive information but identifying attributes do not have. If there is descriptive information about an object, the object should be classified as an entity. For example, STUDENT is an entity in the STUDENT VIEW.
- (ii) Attributes should be attached to entities that they describe most. For example, the attribute Faculty-Name should be an attribute of Faculty instead of the entity STUDENT.
- (iii) Multivalued attribute should be classified as entities.

STEP 2: Identify The Key of Entities

STEP 3: Identify Missing Entities

STEP 4: Identify Generalization and Subset Hierarchies

If there is a generalization or subset hierarchy among entities, then reattach attributes to the relevant entities. For example, suppose the following entities were identified in the EER model:

EMPLOYEE (with identifier EMP-NO and descriptors, EMP-NAME, HOME-ADDRESS, SALARY, DATE-OF-BIRTH
JOB-TITLE, SKILL).

ENGINEER (with identifier EMP-NO and descriptors, EMP-NAME, HOME-ADDRESS, SPECIALTY).

SECRETARY (with identifier EMP-NO and descriptors,
EMP-NAME, DATE-OF-BIRTH,
SALARY,SPEED-OF-TYPING).

TECHNICIAN (with identifier EMP-NO and descriptors,
EMP-NAME, SKILL, YEARS).

Here EMPLOYEE is identified as a generalization of ENGINEER,
SECRETARY and TECHNICIAN.

STEP 5: Define Relationships

After identifying the entities and attributes, the next step is to define relationship among these entities. For each relationship the following should be specified.

Degree (Whether, Unary, Binary, or Ternary).

Cardinalates (the connectivity of each entity should be expressed as either '1' or 'n').

Attributes (If additional attribute are required then they should be identified).

STEP 6: Identify Missing Relationship

To ensure that the view created by the designer completely represents the user's information requirements, it is necessary to check that all the required concepts are represented by the view. These view's must eventually be consolidated into a single global view to eliminate redundancy and inconsistency from the model.

3.2 VIEW INTEGRATION

View integration is a database design technique that offers to synthesize an integrated conceptual schema by combining the previously obtain individual requirements.

For the development of large database it is very difficult to design the whole conceptual database schema at once. So if the database requirement are stated on the basis of individual documents, there is a need for integration.

The main goal of view integration is to find all parts in the individual requirement documents that refer to the same concept in reality, and unify their integration. The unification is very important because often the same portion of reality may be represented in different ways in each individual requirement documents.

We assume the individual requirements are represented as EER schemas and show that the EER model serves as a very good basis for the integration of individual requirement into a single conceptual schema.

View integration may be divided into three main phases: the conflict analysis, the conflict resolution and the actual view-merging phase. In the following we will refer to conflict analysis and conflict resolution as preintegration because, if conflicts do not occur, then these two phase are

STEP 1: Conflict Analysis

The main goal of this step is to detect and resolve all types of incoherences that exist in representing same classes of concepts in the two view. During conflict analysis, several types of conflict may be discovered.

(i) Naming Conflicts

During this step names of concepts are analyzed and compared in order to discover homonyms and synonyms. Synonyms occur when representation in the individual EER schemas refer to different concepts in the reality, and homonyms occurs when the name are the same but different concepts are represented.

(ii) Type Conflicts

This arises when the same concept is represented by different modeling constraints in the two view.

(iii) Domain Conflicts

It occurs, when in different schema the same attribute is associated with different domain. For example, the attribute I-Card-No. may be declared as an integer in one schema and as a character string in another schema.

(iv) Conflicts Among Constraints

It occurs, when two schemas contain different constraints on the same concept.

STEP 2: Assertion Generation

In this step assertions specifying the precise relationship between the domain of pair of entities, and relationship from different view are generated. These assertions form the basis for integration of similar entities and relationship of the two view.

1. Identical domains : $\text{DOM}(E_1) = \text{DOM}(E_2)$
2. Subset domains : $\text{DOM}(E_1) \subset \text{DOM}(E_2)$
3. Superset domains : $\text{DOM}(E_1) \supset \text{DOM}(E_2)$
4. Overlapping domains : $\text{DOM}(E_1) \cap \text{DOM}(E_2) = \phi$
5. Disjoint domains : $\text{DOM}(E_1) \cap \text{DOM}(E_2) = \phi$

3.3 RULES FOR VIEW INTEGRATION

This section discusses the rules used for view integration.

RULES 1: Elements Integration Rule

Let X_1, X_2 be elements in two distinct views, $X_1 \in V_1, X_2 \in V_2$ such that $X_1 \equiv X_2$.

If we denote by X , the element in the integrated schema resulting from the integration of X_1 and X_2 , then:

- If X_1 and X_2 are not of the same type, X is an entity type;
- If X_1 and X_2 are of the same type but are not attributes, X is of the same type as X_1, X_2 .

This rule considers only equivalence assertions. Our approach would be similar to Jardin's.

RULES 2: Links Integration Rule

Let A_1, B_1 be two linked elements in view V_1 , A_2 and B_2 be two linked elements in V_2 and the following correspondence assertions

$$A_1 \equiv A_2$$

$$B_1 \equiv B_2$$

$$A_1-B_1 \equiv A_2-B_2$$

Let A be the integrated element in IS (Integrated Schema) corresponding to A_1 and A_2 , let B be the integrated element in IS corresponding to B_1 and B_2 then the integration of A_1-B_1 and A_2-B_2 links are:

- A role links if A and B are an entity type and a relationship type respectively.
- An attribute links if A and B are an element and an attribute respectively.
- A links relationship type with its two roles (Standard name, no attribute) if A and B are two entity types.

The cardinalities of the integrated link or path, are:

$$\text{Cardmin}(A) = \text{Cardmin}(A_1) = \text{Cardmin}(A_2)$$

$$\text{Cardmax}(A) = \text{Cardmax}(A_1) = \text{Cardmax}(A_2)$$

$$\text{Cardmin}(B) = \text{Cardmin}(B_1) = \text{Cardmin}(B_2)$$

$$\text{Cardmax}(B) = \text{Cardmax}(B_1) = \text{Cardmax}(B_2)$$

RULE 3: Path Integration Rule

Let E_1, E_2, \dots, E_n be elements in view V_1 . Let F_1, F_2, \dots, F_p be elements in view V_2 , with the following correspondence assertion:

$$E_1 \equiv F_1.$$

Let G_1 be the integrated element in IS corresponding to E_1 and F_1 .

Then:

- The correspondence assertion between a link and a path,

$$E_1-E_2 \equiv F_1-F_2-\dots-F_p$$

with $E_2 \equiv F_p$ generating G_p in IS, generates in IS a path $G_1-F'_2-\dots-F'_{p-1} G_p$ where F'_2, \dots, F'_{p-1} are elements of IS corresponding to F_2, \dots, F_{p-1} , and each link of the path is created according to the concepts modeling the link elements as in Rule 2.

- The correspondence assertion between two composite paths

$$E_1-E_2-\dots-E_n \equiv F_1-F_2-\dots-F_p \quad n>2, p>2.$$

with $E_n \equiv F_p$ generating G_p in IS, generates in IS two paths and an integrity constraint. The two paths are:

$$G_1-E'_2-\dots-E'_{n-1}-G_p$$

$$G_1-F'_2-\dots-F'_{p-1}-G_p$$

Where E'_2, \dots, E'_{n-1} are elements of IS corresponding to E_2, \dots, E_{n-1} , and F'_2, \dots, F'_{p-1} are elements corresponding to F_2, \dots, F_{p-1} , and each of the paths is created according to the modeling concepts of the linked elements, as in Rule 2. The integrity constraint states that the two paths link the same occurrences.

RULE 4: Integration of Attributes of Corresponding Elements

Let E_1 be an element in view V1, and E_2 , an element in view V2, with the following correspondence assertion:

$$E_1 \equiv E_2,$$

with corresponding attributes

$$A_{11} = A_{21}, A_{12} = A_{22}, \dots, A_{1n} = A_{2n}$$

then, the integrated element E in IS corresponding to E_1 and E_2 will have:

- An attribute A_i for each attribute correspondence $A_{1i} = A_{2i}$.
 A_i 's domain and cardinalities are equal to those of A_{1i} and A_{2i}
i.e.

$$\text{Cardmin}(A_i) = \text{Cardmin}(A_{1i}) = \text{Cardmin}(A_{2i})$$

$$\text{Cardmax}(A_i) = \text{Cardmax}(A_{1i}) = \text{Cardmax}(A_{2i})$$

- An attribute B'_j for each attribute B_i of E_1 (or of E_2) has no correspondent. B'_j 's domain and cardinalities are equal to B_j 's ones.

$$\text{Cardmin}(B'_j) = \text{Cardmin}(B_j)$$

$$\text{Cardmax}(B'_j) = \text{Cardmax}(B_j)$$

RULE 5: Attribute With Path Integration Rule

Let E_1, E_2, \dots, E_n be elements and A an attribute in view V_1 ; and let F_1, F_2, \dots, F_p be elements and B an attribute in view V_2 with the following correspondence assertions:

$$E_1 \equiv F_1$$

$$A \equiv B$$

Let G_1 be the integrated element in IS corresponding to E_1 and F_1 ;
then:

- The correspondence assertions

$$E_1-A \equiv F_1-F_2-\dots-F_p-B$$

generates in IS an attribute B' , which is the attribute of F'_p where F'_p is the element corresponding to F_p . The domain and cardinalities of B' are

the same as those of B.

- The correspondence assertion

$$E_1-E_2-\dots-E_n-A \equiv F_1-F_2-\dots-F_p-B \quad n \geq 2, p \geq 2$$

generates in IS two attributes and an integrity constraint. The attributes are: A', which is an attribute of E'_n, where E_n is the element corresponding E_n; and B', which is an attribute of F'_p, where F'_p is the element corresponding to F_p.

Domain and cardinalities of A' and B' are, respectively, the same as those of A and B.

The integrity constraint states that the two paths link the same values.

RULE 6: Add Rule

Any element (entity type or relationship type) that exists in a view and has no corresponding element in any other view is added to the integrated schema with all its attributes without modification:

Let X₁-Y₁ be a link (role or attribute link) that exist in view V1 and has no corresponding link nor path in view V2. Let X and Y be the elements of IS corresponding to X₁ and Y₁. Then, a link or a link relationship type X-Y is added to IS, according to the modeling concepts of X and Y.

Cardinalities of X-Y are define as follows:

If X_1 is equivalent to X ($X_1 \equiv X$), then

$$\text{Cardmin}(X) = \text{Cardmin}(X_1)$$

$$\text{Cardmax}(X) = \text{Cardmax}(X_1)$$

If not (X_1 is only a subset of X), then

$$\text{Cardmin}(X) = 0$$

$$\text{Cardmax}(X) = \text{Cardmax}(X_1)$$

Cardinalities of Y are define in the same way.

The following diagrams sketch how integration rules are applied to usual cases. Cardinalities of links are not shown.

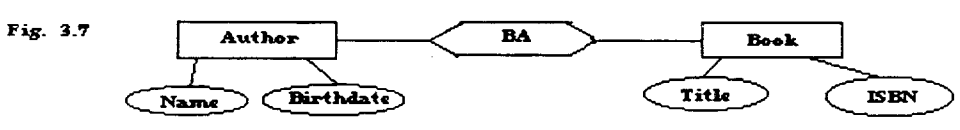
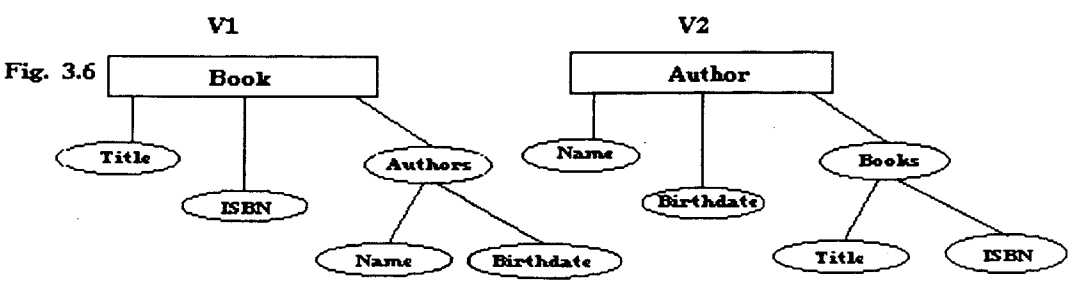
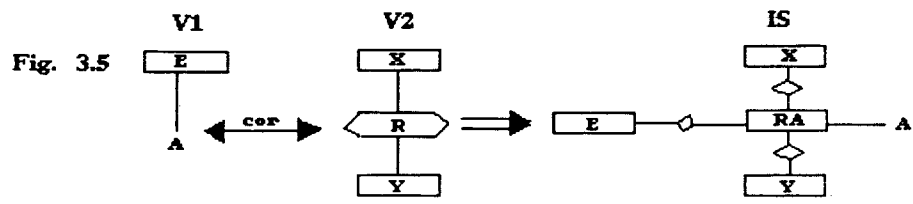
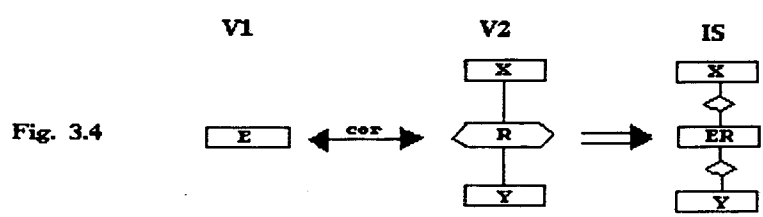
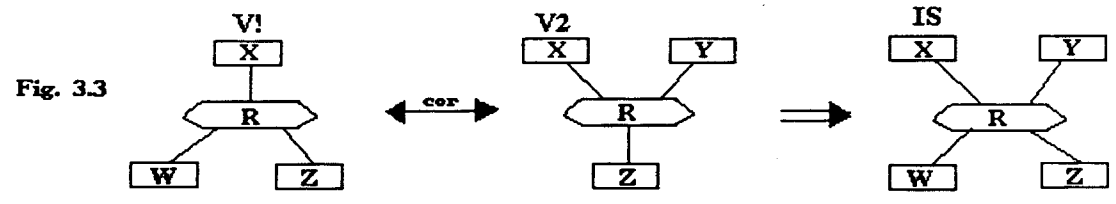
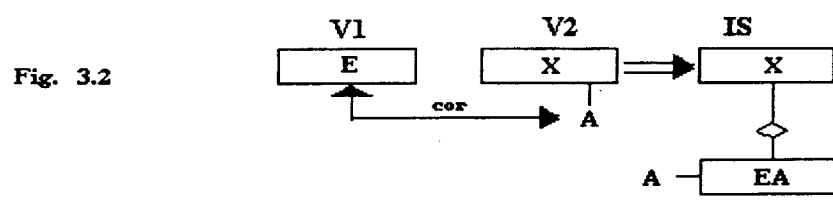
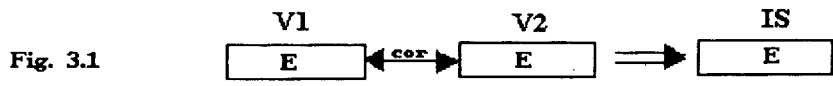
Case 1: Equivalence of Two Entity Types

An equivalence between two entity types generates an entity type (Rule 1 plus Rule 4 for attributes) in Fig. 3.1.

In every case, each time Rule 1 is run, Rule 4 is also activated in order to add attributes to integrated elements.

Case 2: Equivalence of an Entity Type and an Attribute

An equivalence between an entity type E and an attribute A, of some element X, generates an entity type, say EA (Rule 1). Rule 6 generates for the X-A attribute link a relationship type linking EA to the entity type representing the element to which the original attribute is attached in the view (X in our example). The name for the generated relationship type may be automatically generated by the integrator, or be specified by the DBA (Fig. 3.2).



Case 3: Equivalence of Two Relationship Types

In this case, the correspondence assertions are

$$R \equiv R, X \equiv X, Z \equiv Z$$

$$X-R \equiv X-R, Z-R \equiv Z-R \text{ (Fig. 3.3).}$$

An equivalence between relationship types generates a relationship type (rule 1). Link integration (Rule 2) generates one role link for the two X-R links and one for the two Z-R links. Rule 6 adds role links without correspondent: Y-R and W-R. The integrated relationship type will therefore link all entity resulting from the integration of the participating entity types in the views.

Case 4: Equivalence of an Entity Type and a Relationship Type

An equivalence between an entity Type E and a relationship type R generates an entity type, say, ER (Rule 1). Rule 6 generates, for each role of R, a relationship type linking ER with the entity types resulting from the integration of the entity types participating in R (here, X and Y, Fig. 3.4).

Case 5: Equivalence of a Relationship Type and an Attribute

An equivalence between an attribute A and a relationship type R generates an entity type, say, RA (Rule 1). Rule 6 generates the same substitution for R as in the immediately previous case, plus a relationship type linking E and RA. This last relationship type results from adding the E-A attribute link of view V1 to the integrated schema (Fig. 3.5).

3.4 VIEW INTEGRATION ALGORITHM

The algorithms are designed to perform view integration based on the six rules. Certainly, algorithm will vary depending on which integration processing strategy is chosen: N-ary versus binary N-ary strategies integrate n view in one shot. In this case, the integrator will have to sum up all correspondences involving the same object through all views. With that global knowledge, it will be able to decide which construct is to be build in the integrated schema, and to generate the appropriate mappings between each view and the integrated schema.

Algorithm 1.

Integration of two views (V1 and V2):

Input: V1, V2 and the correspondence assertions in between.

Output: An IS and the V1-IS, V2-IS correspondences.

A. Deferring Attribute Correspondences

/* The corresponding attribute are included in a "with corresponding attributes" clause of corresponding elements, must be processed after integration of the other elements and the paths. The initial step by the algorithm is intended to isolate correspondences whose integration is to be deferred.*/

- Remove from the set of elements, correspondences assertions all attributes correspondence, and put them aside.
- Remove these attributes from the views. (temporarily)

- Remove from the set of path correspondence assertions all correspondences where the paths terminate on corresponding attributes, and put them aside.

B. Element Integration

/ Phase 1 (Integrate Corresponding Elements)*/*

For each element correspondence assertion $X_1\langle\text{cor}\rangle X_2$

do:

- Execute Rule 1 (element Integration rule)
- Execute Rule 4 (Integration of attribute) for the integrated elements.
- Mark as already processed, in V1 and in V2, X_1, X_2 , their attributes and their attribute links.
- Generate the correspondence assertions $x_1\langle\text{cor}\rangle X$ in V1-IS and $X_2\langle\text{cor}\rangle X$ in V2-IS

enddo.

/ Phase 2 (Add Noncorresponding Elements) */*

For each V1 element and for each V2 element that has not been marked as processed in phase 1.

do:

- Execute Rule 6 (add rule)
- Mark this element as processed, and its attributes and its attribute link (in V1 or in V2)
- Generate the correspondence assertion in V1-IS or in V2-IS

enddo.

C. Path Integration

/* Phase 1 (Add Path Correspondence Assertion for Roles of Equivalent Relationship Types) */

For each pair of corresponding equivalent relationship types, R_1 in $V1$, R_2 in $V2$ such that

R_1 links E_1, F_1, \dots, G_1 R_2 links E_2, F_2, \dots, H_2

$R_1 \equiv R_2, \quad E_1 \equiv E_2, \quad F_1 \equiv F_2 \dots$

do:

- Add to the set of path correspondence assertions, the following assertion.

$E_1-R_1 \equiv E_2-R_2, \quad F_1-R_1 \equiv F_2-R_2 \dots$

enddo.

/* Phase 2 (Integrate Corresponding Links and Paths)*/

For each path correspondence assertion $X_1, \dots, Z_1 \langle \text{cor} \rangle X_2, \dots, Z_2$

do:

- Execute the appropriate integration rule: Rule 2 if two link are involved, Rule 3 if composite paths are involved.
- Mark as processed in $V1$ and in $V2$, the corresponding links.
- Generate the path correspondence assertion in $V1$ -IS and in $V2$ -IS

enddo.

/* Phase 3 (Add Noncorresponding Links)*/

For each $V1$ link and for each $V2$ link that has not been marked as processed.

do:

- Execute Rule 6 (add rule)
- Mark this link as processed (in V1 or in V2)
- Generate the path correspondence assertion in V1-IS or in V2-IS

enddo.

D. Integration of Attribute correspondences

/* We now consider attribute and path correspondence assertions */

For each attribute correspondence assertion, $A_1 <cor> A_2$

do:

- If there is a path correspondences assertion involving A_1 and A_2 then execute rule 5 (attribute with path integration rule) else add both attributes, A_1 and A_2 , to IS

endif.

- Generate the path correspondence assertions in VI-IS and/or in V2-IS

enddo.

end. (end of algorithm 1)

Algorithm 2

Refinement of an Integrated Schema

Input : an integrated schema

Output: an equivalent integrated schema IS and the IS-IS' correspondences.

/* Replace entity types, which are only bound to link relationship type by EER relationship type */

For each entity type E in IS such that all its roles are bound to link relationship types R_1, R_2, \dots, R_n whose cardinalities are 1:1.

do:

- Substitute a new relationship type (say, R) for E together with all its roles and related relationship types. R links all entity types that were bound by link relationship types to E . R 's attribute are the attribute of E .
- Generate the following correspondence assertion in $IS-IS'$

$$E \equiv R, R_1 \equiv R, R_2 \equiv R, \dots, R_n \equiv R.$$

enddo. (end of Algorithm 2)

3.5 EXAMPLES ILLUSTRATING THE VIEW INTEGRATION ALGORITHM

Here, we illustrated the main aspects of our algorithm. We take library information system which shows how different modeling constructs (entity types and attributes) are integrated and the importance of links integration. The second one, about customers' orders, requires integration of a composite path and of corresponding attributes of noncorresponding elements. Last, we discuss a new example, taken from (Larson [1989]), which involves integration of an entity type and relationship type, and refinement of the integrated schema.

A. Example I: Library Information System

Let us consider the views V_1, V_2 (Fig. 3.6). The set of correspondence assertions between V_1 and V_2 consists of two assertions about elements and one about paths:

Book \equiv Author.books with corresponding attributes;

title = title, ISBN = ISBN

Book.authors \equiv Author with corresponding attributes;

name = name, birthdate = birthdate

Book-authors \equiv book-Author

Step 1 of the integration algorithm is not required: There is no attribute correspondence assertion.

Step 2, phase 1, will start building the integrated schema by considering the first assertion: Book \equiv Author.books. As the two elements are not of the same type, an entity type, say Book, is generated in IS. Book has title and ISBN as attributes. The following correspondences are generated:

A) V1-IS: Book \equiv Book with corresponding attributes:

title = title, ISBN = ISBN.

B) V2-IS: Author.Books \equiv Book with corresponding attributes:

title = title, ISBN = ISBN.

Similarly, the next correspondence will be dealt with:

Book.authors \equiv Author. An entity type, say Author, is generated in IS, with name and birthdate as attributes. The following correspondences are generated:

A) V1-IS: Book.aurthors \equiv Author with corresponding attributes:

name = name, birthdate = birthdate.

B) V2-IS: Author \equiv Author with corresponding attributes:

name = name, birthdate = birthdate.

Step 2, phase 2: This phase is not needed. There is no corresponding element.

Step 3, Phase 1: This phase is not needed. There is no relationship type.

Step 3, phase 2: Deals with unique path correspondence:

Book-authors \equiv book-author

Book Book-authors (V1) and books-Author(V2) are direct links. According to Rule 2, their integration consists in inserting a link in IS between Book and Author. As these are two entity types, the new link will conform to the following pattern: role-relationship type-role. Assuming the new relationship type is named BA, two more correspondences are generated:

-V1-Is :Book - authors \equiv Book - BA - Author

-V2-IS : Author - Book \equiv Author - BA - Book

Nothing else is left to be done. The integration has produced the integrated schema shown in Fig. 3.7.

B. Example 2: Customers' Orders

Let us now consider the customers' orders example. The views to be integrated are illustrated in Fig.3.8.

The set of correspondence assertions between V1 and V2 consist of the following assertions:

Customer \equiv ustomer with corresponding attributes:

name = name

ordered \equiv Ordline with corresponding attributes:

quantity = qty

Product \equiv Product with corresponding attributes:

P# = P#

Fig. 3.8

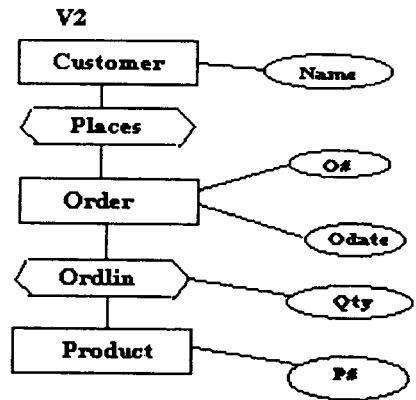
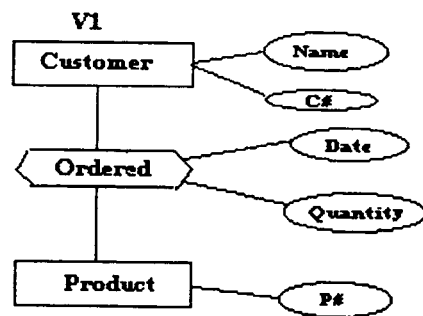
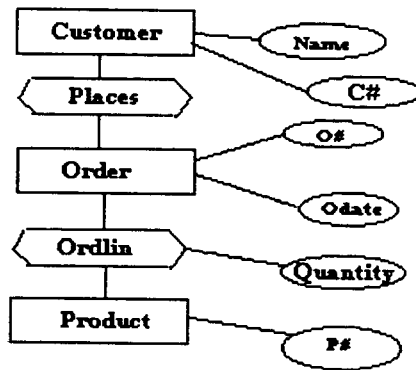
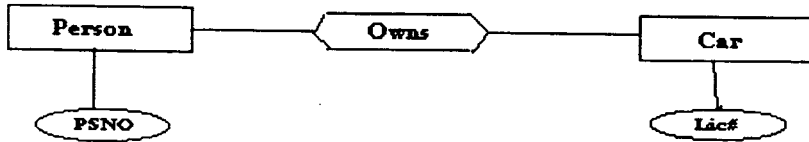


Fig. 3.9



V3



V4

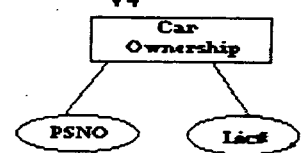


Fig. 3.11

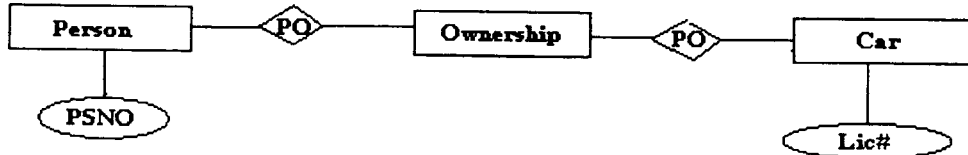
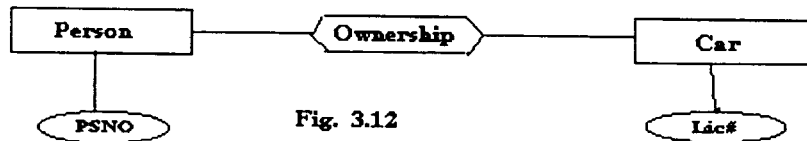


Fig. 3.12



ordered-date \equiv Order-Odate

ordered-date \equiv ordline-Order-Odate

Customer-ordered \equiv Customer-places-Order-ordline

Step 1 puts aside the two correspondences involving attributes date and Odate, and removes date and Odate from the views.

Phase 1 of step 2 then proceeds with the first three assertions, inserting into IS:

- an entity type Customer, with name and C# attributes,
- a relationship type, say oline, with an attribute quantity.
- an entity type Product, with the P# attribute, and generating the appropriate correspondence, assertions V1-IS and V2-IS.

Phase 2 of step 2 adds to IS the places and Order elements from V2.

- a relationship type places, without attributes.
- an entity type Order, with the O# attribute, and generates additional correspondence assertions V1-IS and V2-IS.

Phase 1 of step 3 adds the following path correspondence assertion between V1 and V2.

Product-ordered \equiv Product-ordline

Phase 2 of step 3 integrates the following paths:

Customer-ordered \equiv Customer-places-Order-ordline and generates Is three lins for the composite path:

Customer-places, places-Order, Order-oline. Then, the paths

Product-ordered \equiv Product-ordline

are integrated, generating in IS the Product-oline link.

Adequate correspondences are generated for V1-Is and V2-IS.

The next step 4, deals with the correspondences involving the date and Odate attributes (which were put aside by step 1):

$\text{ordered.date} \equiv \text{Order.Odate}$

$\text{ordered-ordered.date} \equiv \text{ordline-Order} - \text{Order.Odate}.$

The path correspondence includes a direct link: ordered-date. Therefore, the other link is chosen for integration in IS. The ordline-Order link already is in Is. The algorithm has only to add the Order-Odate link, which implies creating Odate in IS as attribute of Order. Some more assertions go into V1-IS and V2-Is.

As no refinement is needed, the final integrated schema is as shown in Fig. 3.9.

C. Example 3: Car's Ownerships

Our last example was proposed by (Larson [1989]) to show a case of entity-type/relationship-type integration. It is based on the views illustrated in Fig.3.10.

Correspondence assertions are:

- $\text{Person} \equiv \text{Carownership.PSNO}$ with corresponding attributes:
 $\text{PSNO} = \text{PSNO}.$
- $\text{Car} \equiv \text{Carownership.Lic\#}$ with corresponding attributes:
 $\text{Lic\#} = \text{Lic\#}.$
- $\text{owns} \equiv \text{Carownership}.$
- $\text{owner-Person} \equiv \text{Carownership-PSNO}.$

- owner-Car \equiv Carownership-Lic#.

Integration starts with step 2 which generates three entity types in IS:

- Person (integration of Person and PSNO), with attribute PSNO.
- Car (integration of Car and Lic#), with attribute Lic#.
- Ownership (integration on owns and Carownership).

Path integration, step 3 adds the Person-Ownership and Car-Ownership links. This process implies that two new relationships are added-one, say PO, between Car and Ownership. Cardinalities for the PO-Ownership and CO-Ownership roles are 1:1.

After this step, the integrated schema is as shown in Fig. 3.11. In this case, if the refinement algorithm is run, the rule applied and the PO-Ownership-CO structure is replaced by a simpler Ownership relationship type (Fig. 3.12), which is the final expected result.

STEP 3: MERGING

Once the assertion specifying the correspondence between entities of the two views are generated, similar entities and relationships are integrated and unrelated entities and relationships are simply merged in the partial integrated view.

CHAPTER-FOUR

AN INTERFACE BETWEEN EER MODEL TO RELATIONAL MODEL

4.1 VIEW TRANSLATION

In this step the view is transformed from EER Model into Relational Model. For each entity type E of the EER schema, we create a relational schema that includes all the attributes that can have only atomic attribute values. By analyzing the FDs (Functional Dependencies) valid in RS (Relational Schema), the set candidate key is chosen as a primary key. Foreign key attribute or relationship attribute will be added during subsequent steps so that the translation process is carried out in the following steps.

STEP 1: Assume Globally Unique Key Attributes. In this step, to distinguish the key of two entities, the label of the entity E is prefixed to the label of its key attributes.

STEP 2: Determine the key and other attributes of the relationships.

STEP 3: Define relational schema and key dependencies.

STEP 4: Determine the key and total attribute of the entities.

STEP 5: For each weak entity create a relational schema and include the attributes with atomic domain of WE in RS.

- STEP 6:** For each multivalued or composite attribute A we have to create a new RS, that includes an entity set X corresponding to A plus the primary key K of E. The primary key of RS formed from the attribute set X.
- STEP 7:** Create a single relation RS_i for each subclass S_i and include in RS_i the set of attributes that are specific for the subclass.
- STEP 8:** If only a few specific attributes are defined for a subclass, it is possible to represent a generalization or subset hierarchy by creating only a single relational schema to represent the superclass and all its subclasses.

4.2 ALGORITHM FOR VIEW TRANSLATION

The EER model includes all the modeling concept of the ER model so that the algorithm for the ER model and then for the EER model can be developed.

4.2.1 Transformation Rules for EER model

The rules for transformation are described as :

- RULE 1:** For each regular entity type E in the ER schema, we create a relation R that includes all the attributes of E. For a composite attribute we include only the simple component attributes. We also choose one of the key attributes of E as primary key for R. If the chosen key of E is composite, then the set of simple attributes that form it will together form the primary key of R.

- RULE 2:** For each weak entity type W in the ER schema with owner entity type, we create a relation R and include all simple attributes of W as attributes of R . We include as foreign key attributes of R the primary key attribute(s) of the relation that corresponds to the owner entity type E ; this takes care of identifying relationship type of W . The primary key of R is the combination of the primary key of the owner and partial key of the weak entity type W .
- RULE 3:** For each, binary 1:1 relationship type R in the ER schema, we identify the relations S and T that correspond to the entity type participating in R . We choose one of the relations, say S , and include as foreign key in S the primary key of T . It is better to choose an entity type with total participation in R in the role of S . We include all the attribute of the 1:1 relationship type R as attributes of S .
- RULE 4:** For each regular (nonweak) binary 1: N relationship type R , we identify the relation S that represents the participating entity type at the N -side of the relationship type. We include as foreign key in S the primary key of the relation T that represents the other entity type participating in R ; this is because each entity instance on the N -side is related to at most one entity instance on the 1 -side of the relation type.
- RULE 5:** For each binary M : N relationship type R , we create a new relation S to represent R . We include as foreign key attributes in S the primary keys of the relations that represent the participating entity type; their combination will form the

primary key of S. We also include any simple attribute of the M:N relationship type as attribute of S.

RULE 6: For each multivalued attribute A, we create a new relation R that includes an attribute corresponding to A plus the primary key attribute K of the relation that represents the entity type or relationship type that has A as an attribute. The primary key of R is then the combination of A and K. If the multivalued attribute is composite, we include its simple components.

RULE 7: For each relationship type R, $n > 2$, we create a new relation S to represent R. We include as foreign key attributes in S the primary keys of the relations that represent the participating entity type. We also include any simple attribute of the n _ary relationship type as attribute of S. The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types. However, if the participation constraint (min, max) of one of the entity type E participating in R has $\text{max} = 1$, then the primary key of S can be the single foreign key attribute that refers to the relation E' corresponding to E; this is because, in this case each entity e in E will participate in at most one relationship of R and can hence uniquely identify relationship instance.

RULE 8: Convert each specialization with m subclasses $\{s_1, s_2, \dots, s_m\}$ and (generalized) superclass C, where the attributes of C are $\{k, a_1, \dots, a_n\}$ and k is the primary key, into relation schema

using one of the following options.

- Option 8 A:** Create a relation L for C with attributes $\text{Attrs}(L) = \{k, a_1, \dots, a_n\}$ and $\text{PK}(L) = k$. Also create a relation L_i for each $s_i, 1 \leq i \leq m$, with the attributes $\text{Attrs}(L_i) = \{k\} \cup \{\text{attribute of } S_i\}$ and $\text{PK}(L_i) = K$.
- Option 8B:** Create relation L_i for each subclass $s_i, 1 \leq i \leq m$, with the attributes $\text{Attrs}(L_i) = \{\text{attribute of } s_i\} \cup \{K, a_1, \dots, a_n\}$ and $\text{PK}(L_i) = K$.
- Option 8C:** Create a single relation L with attributes $\text{Attrs}(L) = \{K, a_1, \dots, a_n\} \cup \{\text{attributes of } s_1\} \cup \dots \cup \{\text{attributes of } s_m\} \cup \{t\}$ and $\text{PK}(L) = K$. This option is for a specialization whose subclasses are disjoint and t is the attribute type that indicates the subclass to which each type belongs, if any.
- Option 8D:** Create a single relation schema L with attributes $\text{Attrs}(L) = \{K, a_1, \dots, a_n\} \cup \{\text{attribute of } s_1\} \cup \dots \cup \{\text{attributes of } S_m\} \cup \{t_1, t_2, \dots, t_m\}$ and $\text{PK}(L)=K$. This option for a specialization whose subclasses are overlapping (not disjoint), each $t_i, 1 \leq i \leq m$, is boolean attribute to include whether or not a tuple belongs to subclass S_i . This option has the potential for generating a large number of null values.

4.3 EXAMPLE ILLUSTRATING THE VIEW TRANSLATION ALGORITHM

We define a simple EER Model to illustrate the major steps in the design of relational model.

Suppose that it is desirable to build a company-wide database for a large engineering firm that keeps track of all personnel, their skills and projects assigned, departments worked in, and personal computers allocated. Each employee is given a job title (engineer, technician, secretary, manager). Engineers and technicians work on an average of two projects at one time, and each project would be headquartered at a different location (city). We assume that analysis of the detailed requirements for the data relationships in the company results in global view in EER diagram fig 4.1.

Each relationship in the diagram is based on a verifiable assertion about the actual data in the company.

As an example of view integration, the generalization of EMPLOYEE over JOB_TITLE could represent the consolidation of two views of the database, one based on EMPLOYEE as the basic unit of personnel and the other based on the classification of the employees by job titles and special relationships with those classification, such as the allocation of personal computers (PCs) to engineers. Now we apply the rules of algorithm for translation from EER model to relationship model:

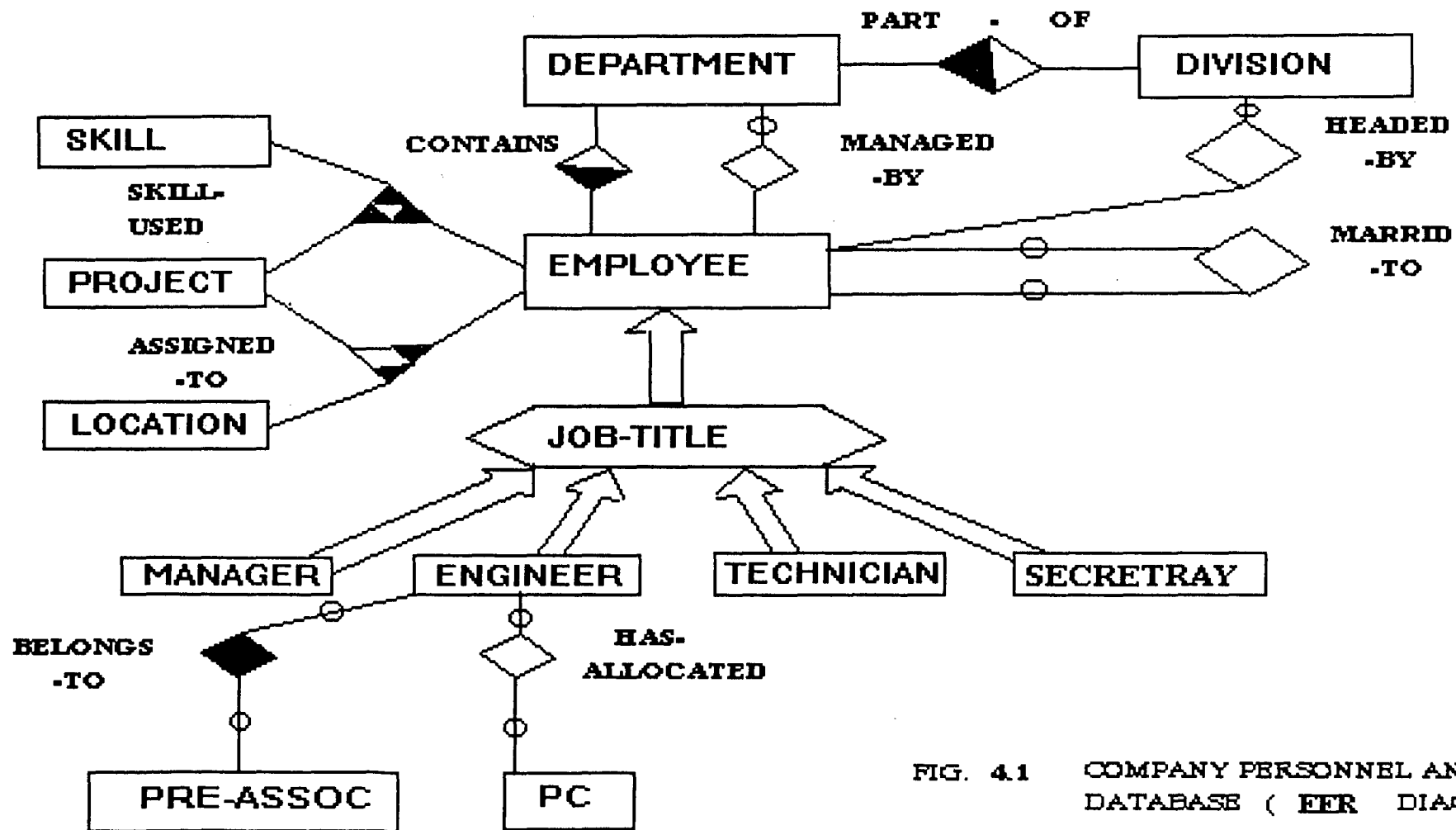


FIG. 4.1 COMPANY PERSONNEL AND PROJECT DATABASE (EER DIAGRAM)

- (i) As the Rule 1 is stating "For each regular entity type E in the ER schema, we create a relation R that include all the attributes of E. For a composite attribute we include only attributes of the simple component.

So in the diagram we have entities "Department, Division, Employee.... etc." for which we are interested in creating the relations. We will also apply Rule 2, once we are done with set of entities.

- (ii) After this we study the association of relationships, which are one-to-one, one-to-many and many-to-many. We proceed in the following steps:

STEP 1: Now we first study the one-to-one relationship between entities. There are two such relationships.

- (a) One such instance is in Figure 4.2. where we have :

Every department must have a manager. An employee can be manager of at most one department.

- (b) In figure 4.3 we have another instance of one_to_one relations. Some personal computers (PCs) are allocated to engineers, but not necessary to all engineers.

So now we apply Rule 3, Rule 4 and Rule 2 to these two cases, then the relation in figure 4.2 can be converted into followings :

Fig. 4.2

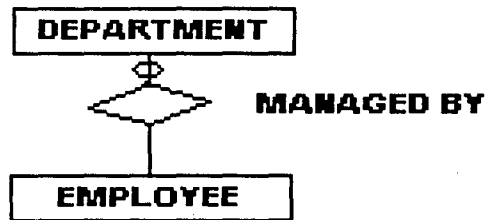


Fig. 4.4

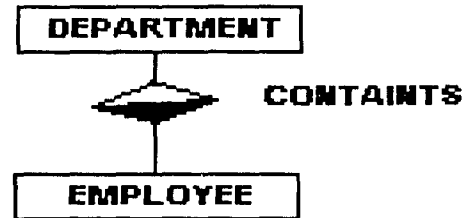


Fig. 4.3

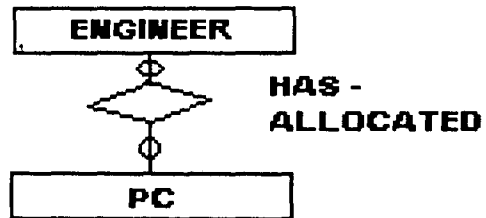


Fig. 4.5

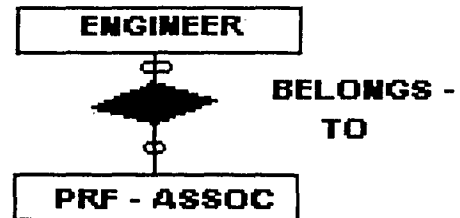


Fig. 4.6



DEPARTMENT (DEPT-NO,.....,EMP-NO)

EMPLOYEE (EMP-NO,...)

Where if EMP_NO is null then it is not allowed in DEPARTMENT.

The relation in Figure 4.3 becomes

ENGINEER (EMP-NO, ...PC-NO)

PC (PC-NO.....)

where If PC-NO is null even then it is allowed in ENGINEER. Now we will study the diagram with perspective of Rule 4 and others earlier Rules, i.e. rule 1, rule 2 and rule 3.

STEP 2: For one_to_many (1:N) relationships in the EER diagram, we have one case as shown in Figure 4.4. The figure implies that the every employee works in exactly one department. Every department could contain many employees. So after applying rule 4 we get the relations:

DEPARTMENT (DEPT-NO,...)

EMPLOYEE(EMP-NO,.....DEPT-NO)

Where, if DEPT_NO is null then it is not allowed in EMPLOYEE.

STEP 3: Now we study many_to_many (N:M) relationship of EER diagram shown in figure 4.5. In applying Rule 5 here we get the following relation:

PRF_ASSOC(PA-NO,....)

ENGINEER(EMP-NO,...)

BELONGS_TO(PA-NO,EMP-NO)

Where every professional association could have many members who are engineers, or nonengineers. Every engineer could belong many professional associations, or none.

(iii) Now we study the various extended affairs in ER i.e one entity one relation, n_entity, n_ary, generalization

STEP 1: Now we study one entity, one relation constructs. This is represented by EER diagram in Fig 4.6. Where an employee could have one of the other employee as his or her spouse. After applying rules 6 and rule 8 we have the relation as:

EMPLOYEE (EMP-NO,..SP-EMP-NO)

Where if SP_EMP_NO is null then it is allowed in EMPLOYEE.

STEP 2: We study n-entities, n-ary, relationships where $n > 2$. We found such instance in Fig. 4.7.

Where employees are assigned one or more projects, but can only be assigned at most one project at a given location.

After applying Rule 7 we get the following relations:

EMPLOYEE (EMP-NO,...)

PROJECT (PROJ-NAME,...)

LOCATION (LOC-NAME,...)

ASSIGNED_TO (EMP-NO, LOC-NAME, PROJ-NAME)

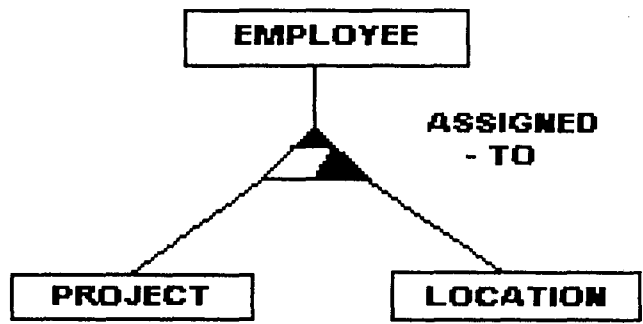


Fig. 4.7

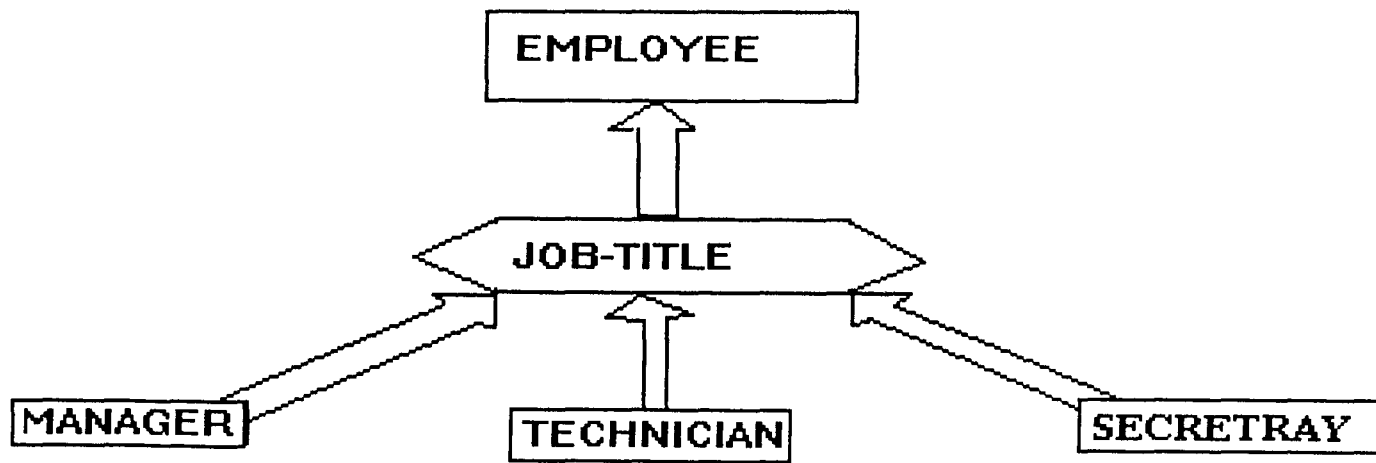


Fig. 4.8

Where functional dependencies are :

EMP-NO, LOC-NAME...>PROJ-NAME

STEP 3: In the search for generalization and subset hierarchy the instances for this are represented in Fig. 4.8. The generalization hierarchy as described in figure 4.8 show that different type of employees are partitioned by values of a common attribute JOB-TITLE. By applying Rule 8, we have the following relations:

EMPLOYEE(EMP-NO,JOB-TITLE, common attributes)

EMP_MANGER(EMP-NO, specific attributes)

EMP_SECRETARY(EMP-NO, specific attributes)

EMP_TECHNICIAN(EMP-NO, Specific attributes)

After this step since there is no instance of Multiple relationship and Aggregation, thus the objective of the chapter is completed.

CHAPTER-FIVE

AN INTERFACE FROM EER MODEL TO OBJECT ORIENTED MODEL

5.1 Development of Rules for the Translation

An extended entity diagram of the database can be viewed into two different perspective, the first is relational and second is object oriented.

In relational database for each entity type record and for each relationship type record, different tables are created which contains data in ordered tuples. The rules used in the view translation in previous chapter will apply here for the translation of EER into relational model. In an object oriented database system, we may represent the same EER diagram, in terms of object model. The translation process will work on following rules.

RULE 1: Each entity in the EER diagram can be translated into different object types.

RULE 2: Simple data and string fields of the entities can be represented similarly as fields of the objects.

RULE 3: The one-to-many relationship between entities may be represented as listed valued fields to the objects, on the one side of the `one_to_many` relationship, or as a pointed or both.

RULE 4: A many-to-many relationship may be represented as listed valued field on both ends.

RULE 5: It may be the situation that an each object at object class uses a linked list valued field to the another subset.

In object oriented database system in which references are represented by pointers, we may require such a representation with pointers in both directions for efficiency. However the same information is represented in two or more different places. This could lead to an update anomalies in the object oriented representation. We could store a relationship between entities in only one object

RULE 6: A set of rules applicable in object oriented design are also applied. In designing, the following points are considered:-

- Identify objects and classes
- Prepare data dictionary
- Identify association (including aggregation) between objects
- Identify attributes of objects and links
- Organise and simplify object classes using inheritance
- Verify that access paths exist for likely queries
- Iterate and refine the model
- Group classes into modules

5.2 Example illustrating the translation from EER Model to Object Oriented Model

Let us take following EER diagram in Figure 5.1 whose relations translation is shown in figure 5.2.

Then object oriented perspective of the same EER diagram after applying all the rules discussed above, are in figure 5.3. In the object translation of the above EER diagram following observation have been made.

By applying rule 1, we have found that there are three object type one for each of the entity types, person, organization and document of then after applying the rule 2 and rule 3 date and strings of fields of the entities are represented as fields of the object these fields include:

name	: string type
birthdate	: date type
pubs	: list of document
phone	: list of area and number the object persons and name of string type
member	: list of persons and role
publs	: list of documents

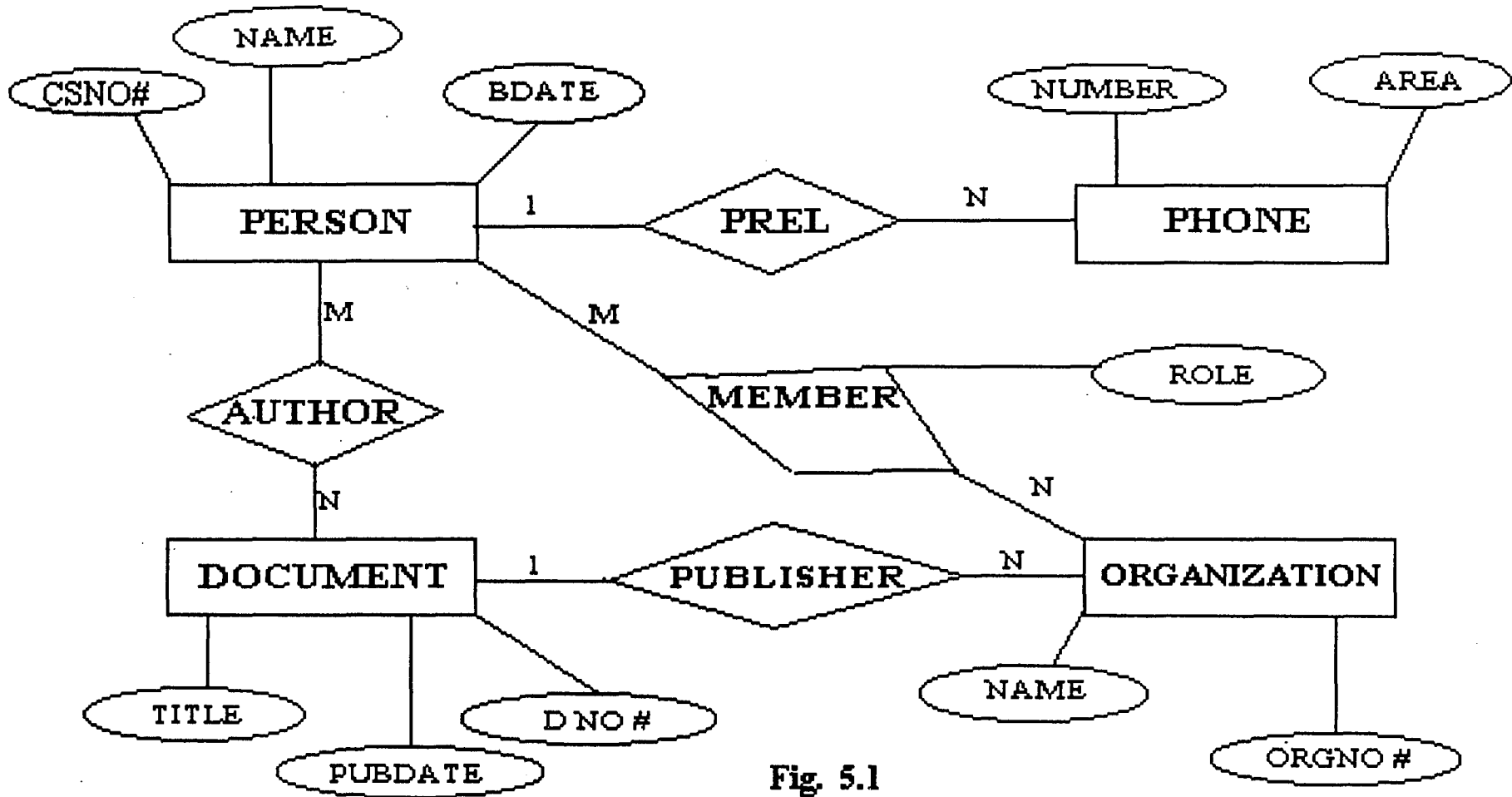


Fig. 5.1

PERSON	CSNO #	NAM	BDATE
ORGANIZATION	ORGNO #	NAME	
DOCUMENT	DNO #	TITLE	PUBDATE
PHONE	CSNO #	NUMBER	AREA
MEMBER	CSNO#	ORGNO #	ROLE
AUTHOR	CSNO #	DNO #	

Fig. 5.2

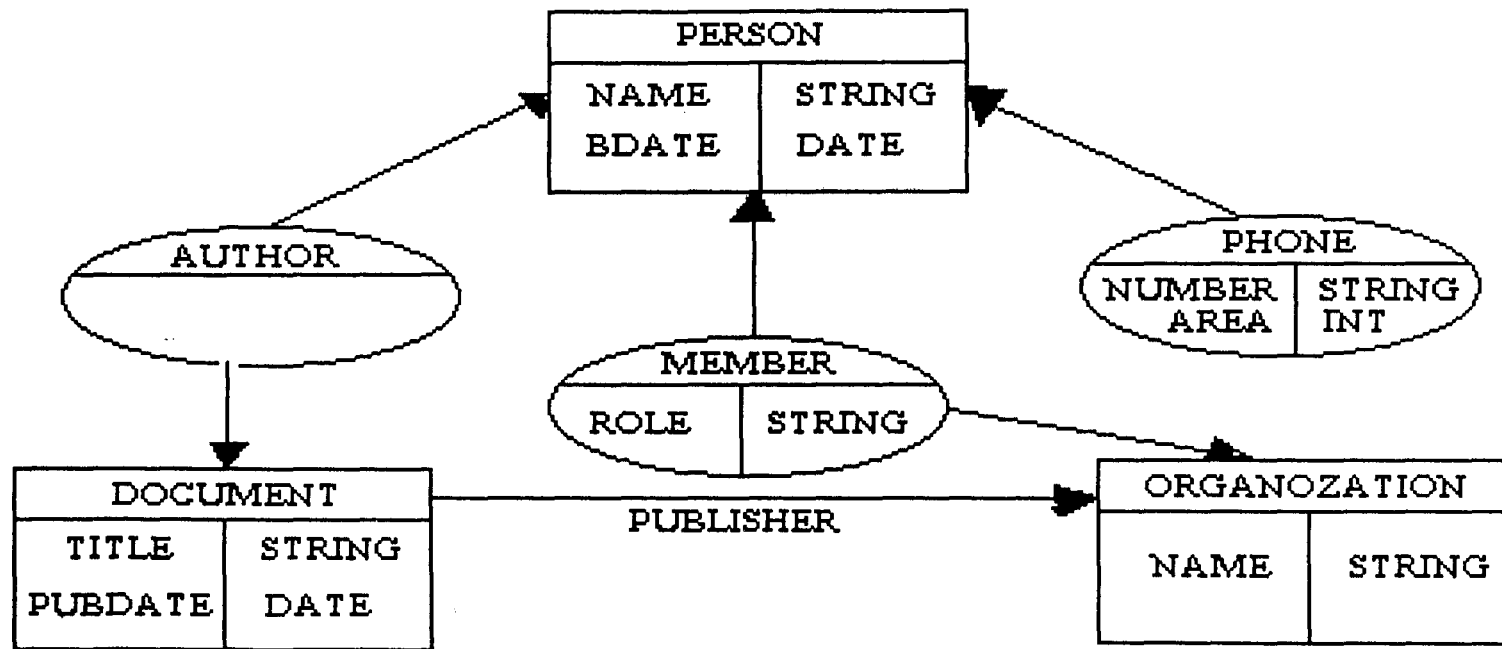


Fig. 5.3

For the object organization

title : string type
pubdate : date type
author : list of persons
publishers : organization type of object document

So our final object classes are mentioned in figure 5.4.

5.3 The Object class declaration in Object Oriented Language C++

Now the perspectives of object oriented are applicable to programming languages like C++ (these languages are called Objected Oriented Language) as follows:

```
class person
{
    char name[];
    char birthdate[];
    document *publs;
    list1 *phone;
}
class organization
{
    char name[];
    list 2 *members;
    publs : document
}
```

PERSON	
NAME	STRING
BDATE	DATE
PUBS	LIST OF DOCUMENT
PHONE	LIST OF [AREA ,NUMBER]

ORGANIZATION	
NAME	STRING
MEMBER	LIST OF [PERSON,ROLE]
PUBLS	LIST OF DOCUMENT

DOCUMENT	
TITLE	STRING
PUBDATE	DATE
AUTHORS	LIST OF PERSON
PULISHER	ORGANIZATION

Fig. 5.4

```
class document
{
    char title[];
    char pubdate[];
    person *authros
    organization publsher
}
```

where list1 is

```
structure list1
{
    char area[];
    struct list1 *next1;
}
```

where list 2 is

```
structure list 2
{
    char person
    struct list2 *next2;
}
```

CHAPTER-SIX

CONCLUSION

In the interface from EER model to relational model, it has been shown that a practical step-by-step methodology for relational database design can be derived using a variety of extensions to the ER conceptual model. The methodology has been illustrated with a simple database design problem, showing each design steps in detail.

A powerful schema integration methodology is the key to successful database design. An integration methodology, designed to meet the objective of support reuse of existing databases and existing application programs, without contradicting the launching of new database services, have been described. The approach employed is based on the following features :

- Automatic resolution of structural conflict
- Conflict resolution performed without modification of initial views.
- Use of a format declarative approaches for user definition of interview and correspondences.
- Applicable to a variety of data models.
- Automatic generation of structured and operational modification between the views and the integrated schema.

The OODBS are getting a lot of attention from various fields. In this work as a interface from EER model to object model, a set of rules have been developed. The methods, discussed provides a procedure for identifying object and classifies how these classes and object are related with each data, and is structured in the EER diagram with the view of OODBS.

Summarizing the discussion, the basic issue arises whether object-oriented concepts are answer to many problems. It is found that object orientation is a new way to tackling database systems but not without use of the fundamental EER model. Coupling of C++ with database functionally is reasonably easy and simple to implement. Since the database interface is based on the EER model, hence it has to be remodeled in terms of EER model. Therefore it is necessary to have some kind of re-engineering, a "backward" mapping from the classical platform data models to the EER model in order to be able to manipulate data with the proposed interface.

Furthermore, some future improvements could be made in the following directions:

- (i) Integration of inclusions, intersection, and exclusion assertions.
- (ii) Consideration of generalization links is correspondence assertions and integration rules.
- (iii) The data model should be enhanced with further modeling concepts. Problems might arise due to their representation in C++.

- (iv) Dynamic schema modifications can be added.
- (v) Support the subdatabases/work spaces i.e. some protection mechanism on type and in balance level. The implementation can be modeled for large objects.

REFERENCES

ATZENI, P., BATINI, C., LENZERINI, M., AND VILLANELLI, F. 1981, INCOD: A system for conceptual design of data and transactions in the entity-relationship model. In Entity- Relationship model. In Entity-Relationship Approach to Information Modeling and Analysis. ER Institute, Saugus, Calif.

Ahmed. R. and Navathe, S.B. (1991). Version Management of Composite Objects in CAD Databases, in "Proc. ACM SIGMOD Conf", pp. 209-108 Denver.

Breitender, C.J. and Muck, T. (1990) A Graph grammar Driven ER case environment, in "Proc. 10th Int'l, Conf. on Entity Relationship Approach", pp. 363-380.

Carswell, J.L., Jr., and Navathe. S.B. (1986) SA-ER: A Methodology That Links Structured Analysis and ER Modeling for Database Design, in "Proc. of the 5th Intl. Entity Relationship Conference", pp 19-35 Dijon, France.

Chen. PP. (1976). The Entity Relationship Model: Towards a Unified View of Data, ACM Trans. on Database System 1(1), 9- 36.

CHUNG, I., NAKAMURA, F., AND CHEN, P. 1981. A decomposition of relations using the entity-relationship approach. In Entity-Relationship Approach to Information Modeling and Analysis, P. Chen. Ed

North-Holland, Amsterdam.

CODD, E. 1970. A relational model for large shared data banks, Commun. ACM 13, 6 (June), 377-387.

C.Batini, M. Lenzerini, and S.B. Navathe, "A comparative analysis of methodologies for database schema integration" ACM Computing Surveys, Vo. 15, no.4 pp. 323-364 Dec. 1986.

Chen. J.Y., and Wang. J.J. Comparing object-oriented design methods experimentally, in Proceedings, 3rd Int. Conf. on Technology of Object-Oriented Languages and Systems-TOOLS '90. Sydney, Australia, November 28-30. 1990.

Coad P., and Yourdan. E. Object Oriented Analysis, Prentice- Hall. Englewood Cliffs. New Jersey., 1990.

Coad, P. Adding to OOA Results. J. Object-Oriented Programming May 64-69(1991b)

Dean H. Object-Oriented Design Using Message Flow Decomposition, J. Object-Oriented Programming 21-31 (1991)

DeAntonellis, V., and Demo. B. (1983) Requirements Collection and Analysis, in S.Ceri, ed., Methodology and Tools for Database Design", pp 9024, North-Holland Amsterdam.

Ehmasri, R., and Navathe, S.B. (1984) "Object Integration in Logical Database Design, in "Proc. Int'l Conf. on Data Engineering", pp 426-433.

ELMASRI, r., HEVNER, A., AND WEELDREVER. J. 1985 The category concept: An extension to the entity-relationship model. Data Knowl. Eng. 1,1 75-116.

Fowler, M., "Which OO analysis and design method" in SCOOP Europe, October, 1991, pp 1-7.

Henderson-Seller B., Edward. J.M. The Object Oriented Systems Life Cycle, Comm. ACM 33, 142-159 (1990)

Jalote, P., "Functional Refinement and Nested Objects for Object-Oriented Design. IEEE Trans. Software Engg. 15, 264- 270(1989)

J.A. Larson, S.B. Navathe, and R. Elmasri, "A theory of attribute equivalence in databases with application to schema integration", IEEE Trans. Software Eng., Vol. 15, no. 4 Apr. 1989.

NAVATHE. S. AND CHENG. A 1983. A methodology for database schema mapping from extended entity relationship models in to the hierarchical model. In the Entity Relationship Approach to software engineering G.C. Davis et al., Eds. Elsevier North-Holland New York.

NAVATHE S. AND GANDHI, S. 1982 A methodology for view integration in logical database design. In Proceedings of the 8th

international conference in very large databases (Maxico City) VLDB Endowment, Saratoga, Calif. pp. 142-152.

NAVATHE S. SASHIDHAR. T., AND ELMASRI R. 1984 Relationship merging in schema integration. In Proceedings of the 10th International Conference Very Large Data Bases (Singapore) VLDB Endowment, Saratoga. Clif., pp. 78-90.

NAVATHE. S. ELMASRI R., AND LARSON J. 1986 Integration user view in database design IEEE Computer 19, 50-62.

Navathe S.B. and Balaraman, A. (1991) A Transaction Architecture for a General Purpose Semantic Data Model Progressive Fragment Allocation, in "Proc. 10th Int'l Conf. on the Entity Relationship Approach" San Mateo. CA.

Pernul. G., and Tjoa, A.M. (1991) A View Integration Approach for the Design of Multilevel Secure Databases, in "Proc. 10th Int'l Conf. on the Entity Relationship Approach" San MATEo, CA.

Smith J., and Smith (1977) Database Abstractions: AGgregation and Generalization ACM Trans. on Database Systems 2(2)

S.B. Navathe. R. Elmasri, and J.A., Larson "Integration user views in database design " IEEE Comput., Vol. 19, no. 1 Jan. 1986. pp. 50-52.

WEBRE, N. 1981 An Extended E.R. Model and its use on a defense project. In Proceedings of the 2nd International Conference on the Entity-Relationship Approach, (Washington D.C.) North-Holland Amsterdam. pp., 175-194.

Ward P.T. How to Integrate Object-Orientation with Structured Analysis and Design, IEEE Software 6, 74-82 (1989)

Wasserman. A.I., An Object-Oriented Structure Design Method for Code Generation, Software Engg. Not. 14, 32-55 (1989)