Database Processing

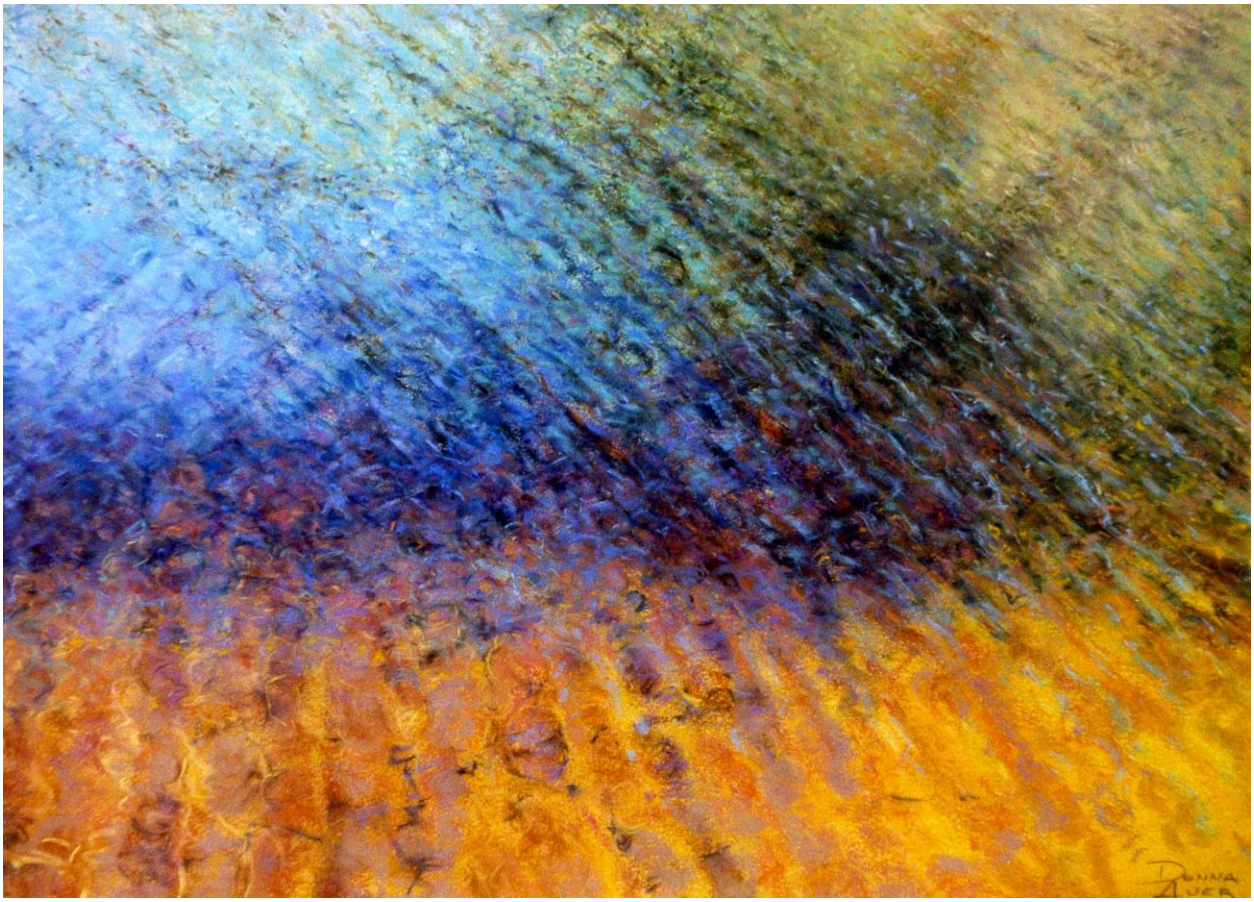
Fundamentals, Design, and Implementation

14th Edition

David M. Kroenke • David J. Auer

Online Appendix C

E-R Diagrams and the IDEF1X Standard



Vice President, Business Publishing: Donna Battista
Editor in Chief: Stephanie Wall
Acquisitions Editor: Nicole Sam
Program Manager Team Lead: Ashley Santora
Program Manager: Denise Weiss
Editorial Assistant: Olivia Vignone
Vice President, Product Marketing: Maggie Moylan
Director of Marketing, Digital Services and Products: Jeanette Koskinas
Executive Product Marketing Manager: Anne Fahlgren
Field Marketing Manager: Lenny Ann Raper
Senior Strategic Marketing Manager: Erin Gardner
Product Marketing Assistant: Jessica Quazza
Project Manager Team Lead: Jeff Holcomb
Project Manager: Ilene Kahn
Operations Specialist: Diane Peirano
Senior Art Director: Janet Slowik
Text Designer: Integra Software Services Pvt. Ltd.
Cover Designer: Integra Software Services Pvt. Ltd.
Cover Art: Donna Auer
Vice President, Director of Digital Strategy & Assessment: Paul Gentile
Manager of Learning Applications: Paul Deluca
Digital Editor: Brian Surette
Digital Studio Manager: Diane Lombardo
Digital Studio Project Manager: Robin Lazrus
Digital Studio Project Manager: Alana Coles
Digital Studio Project Manager: Monique Lawrence
Digital Studio Project Manager: Regina DaSilva
Full-Service Project Management and Composition: Integra Software Services Pvt. Ltd.
Printer/Binder: RRD Willard
Cover Printer: Phoenix Color/Hagerstown
Text Font: 10/12 Mentor Std Light

Library of Congress Cataloging-in-Publication Data

Kroenke, David M.
Database processing: fundamentals, design, and implementation/David M. Kroenke, David J. Auer.–Fourteenth edition.
pages cm
Includes bibliographical references and index.
ISBN 978-0-13-387670-3 (student edition)–ISBN 978-0-13-387676-5
(instructor's review copy)
1. Database management. I. Auer, David J. II. Title.
QA76.9.D3K76 2016
005.74–dc23

2015005632

10 9 8 7 6 5 4 3 2 1

PEARSON

ISBN 10: 0-13-387670-5
ISBN 13: 978-0-13-387670-3

Appendix C – 10 9 8 7 6 5 4 3 2 1

Chapter Objectives[0]

- To understand IDEF1X-standard E-R diagrams.
- To be able to model nonidentifying connection relationships, identifying connection relationships, nonspecific relationships, and categorization relationships using the IDEF1X E-R model.
- To understand the differences between E-R generalization/subtype relationships and IDEF1X categorization relationships.
- To understand the use of domains in the IDEF1X E-R model.

What Is the Purpose of This Appendix?

IDEF1X (Integrated Definition 1, Extended) is a variation of the entity-relationship (E-R) model. IDEF1X, which was announced as a national standard in 1993, is based on earlier work done for the U.S. military in the mid-1980s. IDEF1X assumes that a relational database is to be created. IDEF1X is complex and unwieldy and is not as popular as the crow's foot model used throughout this text. In this appendix, we are primarily interested in understanding the IDEF1X notation, and how it relates to the IE Crow's Foot notation.

Why Should I Learn to Use IDEF1X?

Because IDEF1X is a national standard, all vendors of entity-relationship data modeling products that wish to sell to the U.S. government must comply with it. Accordingly, most popular data modeling products have the capability to produce at least rudimentary IDEF1X diagrams. Further, if you work for a U.S. government organization or for an organization that does business with the U.S. government, you may be required to use it. Consequently, we present an overview of IDEF1X here.

[0] Scott Vandenberg of Siena College contributed material to this chapter.

What Will This Appendix Teach Me?

IDEF1X includes entities, relationships, and attributes, but it tightens the meanings of these terms and qualifies them with more specific terminology. In addition, IDEF1X includes the definition of domains, a component not present in the extended E-R model. Finally, IDEF1X changed the E-R graphical symbols, using a dot instead of a crow's foot and adding a new structure called *categories,* a version of generalization hierarchies and subtypes. Differences between the extended E-R model and the IDEF1X model are summarized in Figure C-1. In this appendix, we will learn to use IDEF1X to create data models.

What Are IDEF1X Entities?

IDEF1X entities are the same as the entities in the extended E-R model. They represent something that the users want to track—something about which users want to keep data. Like the extended E-R model,

Extended E-R Model Term	Corresponding IDEF1X E-R Version Term	Remarks
Entity	Entity	Same.
Attribute	Attribute	Same.
Relationship	Relationship	Same.
1:1 and 1:N relationships	Nonidentifying connection relationship	Connection relationship is the same as HAS-A relationship.
N:M relationship	Nonspecific relationship	
ID-dependent relationship	Identifying connection relationship	
Weak entity, but not ID dependent	None	
Supertype entity	Generic entity	Generic entity has an IS-A relationship to a category cluster.
Subtype entity	Category entity	Category entities are mutually exclusive in their category cluster.
None	Domain	

Figure C-1: Correspondence of Terms between the Extended E-R Model and the IDEF1X Version

- Nonidentifying Connection Relationships
- Identifying Connection Relationships
- Nonspecific Relationships
- Categorization Relationships

Figure C-2: IDEF1X Relationship Terms

entities are shown with either square or rounded corners, although the meaning of entities with rounded corners is slightly different in IDEF1X, as you will learn when we discuss identifying relationships.

What Are IDEF1X Relationships?

Figure C-2 lists the four types of IDEF1X relationships. Although the terminology is awkward, it is very specific—just what you would expect from a national standard. We will consider each type in turn.

Nonidentifying Connection Relationships

Nonidentifying connection relationships are 1:1 or 1:N relationships between two non-ID-dependent (hence, *nonidentifying*) entities. Connection relationships are the same as what are called HAS-A relationships in the extended E-R model. Figure C-3 shows examples of nonidentifying connection relationships.

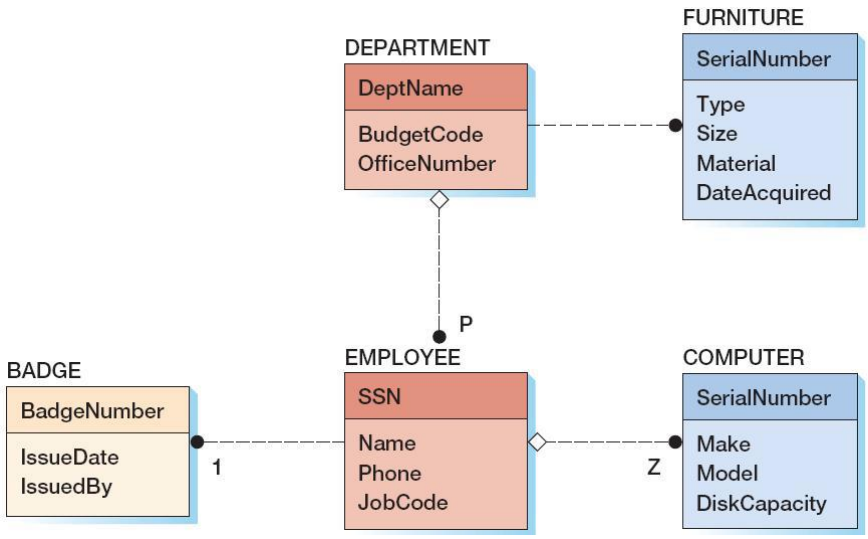


Figure C-3: Nonidentifying Connection Relationships

According to the standard, nonidentifying relationships are represented with a dashed line. Furthermore, in IDEF1X, connection relationships are always drawn from a parent to a child entity. For 1:1 relationships, either entity can be considered the parent, but IDEF1X forces you to pick one of the entities for that role. As shown in Figure C-3, a filled-in circle is placed on the relationship line adjacent to the child entity in an IDEF1X E-R diagram.

In IDEF1X, the default cardinality of a nonidentifying connection relationship is one-to-many, with a mandatory parent and an optional child. Such relationships are shown by a dashed line with a filled-in circle by the child and no other notation. Thus, in Figure C-3 the relationship from DEPARTMENT to FURNITURE is 1:N; no DEPARTMENT is required to have any FURNITURE, and every FURNITURE entity must be assigned to a DEPARTMENT. If the relationship cardinality is different from this default, additional notation is added to the diagram. If the child is required, a *P* is placed near the circle, indicating that one or more child entities are required (the *P* stands for "positive," as in *positive number*). If the parent is optional, a **diamond** is added to the line, adjacent to the parent. In Figure C-3, the relationship from DEPARTMENT to EMPLOYEE is 1:N; a DEPARTMENT must have at least one EMPLOYEE, and an EMPLOYEE is not required to be related to a DEPARTMENT.

The 1:1 relationships are denoted by adding notation near the circle on the relationship line. A one indicates that exactly one child is required; a *Z* indicates that zero or one child is allowed. Thus, in Figure C-3 an EMPLOYEE is connected to exactly one BADGE entity and is also connected to zero or one COMPUTER entity. The diamond indicates that a COMPUTER need not be related to an EMPLOYEE. Because there is no diamond on the EMPLOYEE side of the BADGE:EMPLOYEE relationship, a BADGE must be connected to an EMPLOYEE.

Identifying Connection Relationships

Identifying connection relationships are the same as ID-dependent relationships in the extended E-R model. The identifier of the parent is always part of the identifier of the child. In Figure C-4, a BUILDING entity is the parent of an ID-dependent relationship to the OFFICE entity.

Note that the identifier of BUILDING, BuildingNumber, is part of the identifier of OFFICE. The notation (FK) means that this attribute is a foreign key of another entity (here, BUILDING). Identifying relationships are portrayed with solid lines, and child entities in identifying relationships are shown with rounded corners.

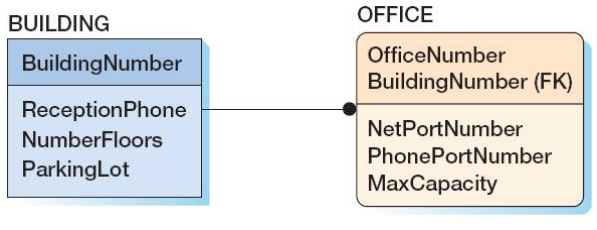


Figure C-4: Identifying Connection Relationship

In this example, there is no additional notation beside the filled-in circle, so the default cardinality is assumed. Thus, a BUILDING may be connected to zero, one, or many OFFICEs. If a *1, Z,* or *P* were placed by the circle, a BUILDING would connect to at most one, to zero or one, or to one or more OFFICE entities, respectively. There can never be a diamond on the parent side of an identifying relationship because children in identifying relationships always require a parent.

Like the crow's foot model, IDEF1X allows for weak entities that are not identifying. However, it provides no special notation for such entities. The minimum cardinality for the parent in such relationships is one, but no other means exists for documenting the fact that they are logically dependent on their parent.

Nonspecific Relationships

A **nonspecific IDEF1X relationship** is simply a many-to-many relationship. Nonspecific relationships are shown with a filled-in circle on each end of the solid relationship line, as shown in Figure C-5. In IDEF1X, a P can be used at either solid dot to set minimum cardinalities of a nonspecific relationship.

Like the crow's foot model, IDEF1X treats many-to-many relationships as poor stepchildren of nonidentfying connection relationships. This is done because such relationships have no direct expression in the relational model. As discussed in Chapter 6, such relationships must be converted to two 1:N relationships before they can be processed in a relational database.

To some people, this is a glaring fault in IDEF1X, because it confuses conceptual schema ideas with internal schema ideas. These people would say that one should be able to fully document N:M relationships, including N:M relationships with attributes, in the conceptual model. What happens to that relationship later in designing the internal schema should not be a factor in conceptual design.

What Are Categorization Relationships?

Categorization relationships are a specialization of generalization/subtype relationships in the extended E-R model. Specifically, a categorization relationship is a relationship between a **generic entity** and another entity called a **category entity**. Category entities are grouped into **categorization clusters**. For example, in Figure C-6 EMPLOYEE is a generic entity; PROGRAMMER, PQA_ENGINEER, and TECH_WRITER are category entities; and the category cluster, represented by the filled-in circle over two horizontal lines, is the collection of PROGRAMMER, PQA_ENGINEER, and TECH_WRITER.

Figure C-5: Nonspecific Model with N:M Relationship

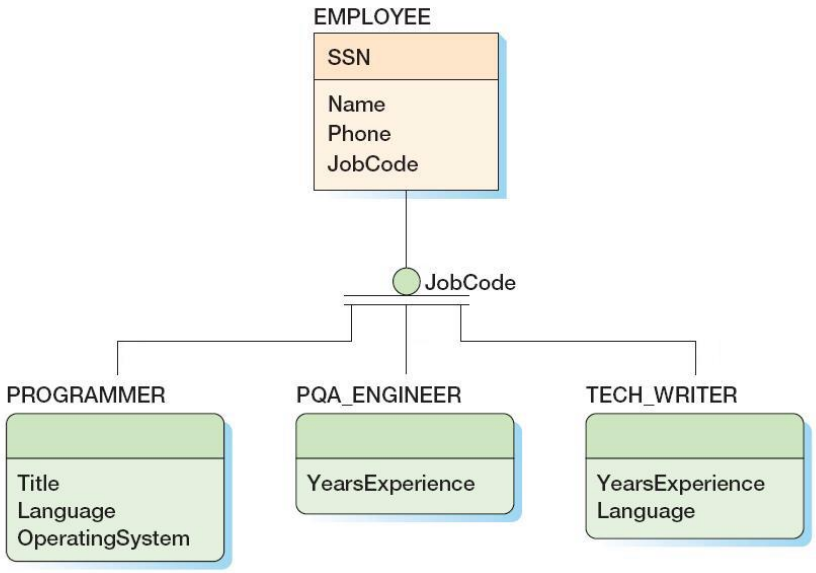


Figure C-6: Categorization Relationship

Categorization relationships are **IS-A relationships**. A programmer is an employee, for example. Because they are IS-A relationships, the primary key of the category entities is the same as the primary key of the generic entity. In this case, the primary key of PROGRAMMER, PQA_ENGINEER, and TECH_WRITER is SSN. Because of this, category entities are shown without primary keys.

In IDEF1X, the entities in a category cluster are **mutually exclusive**. In Figure C-6, an EMPLOYEE can be a PROGRAMMER, a PQA_ENGINEER, *or* a TECH_WRITER. An employee cannot be two or more of these.

Category clusters may have a **discriminator**, which is an attribute of the generic entity that indicates the type of the EMPLOYEE. In Figure C-6, JobCode is the discriminator. This means that the value of JobCode can be used to determine whether the EMPLOYEE is a PROGRAMMER, a PQA_ENGINEER, or a TECH_WRITER.

The means by which this is done are not specified in the conceptual model; we are simply stating here that it can be done. Some category clusters do not have a discriminator; the determination of the category entity is left unspecified.

Category clusters are of two types: complete and incomplete. In a **complete category cluster**, every possible type of category for the cluster is shown. Complete category clusters are denoted by two horizontal lines with a gap in between. The category cluster in Figure C-6 is complete. Because this cluster is complete, all of the categories of EMPLOYEE are shown; there is no missing category. Hence, every EMPLOYEE can be categorized as a PROGRAMMER, a PQA_ENGINEER, or a TECH_WRITER.

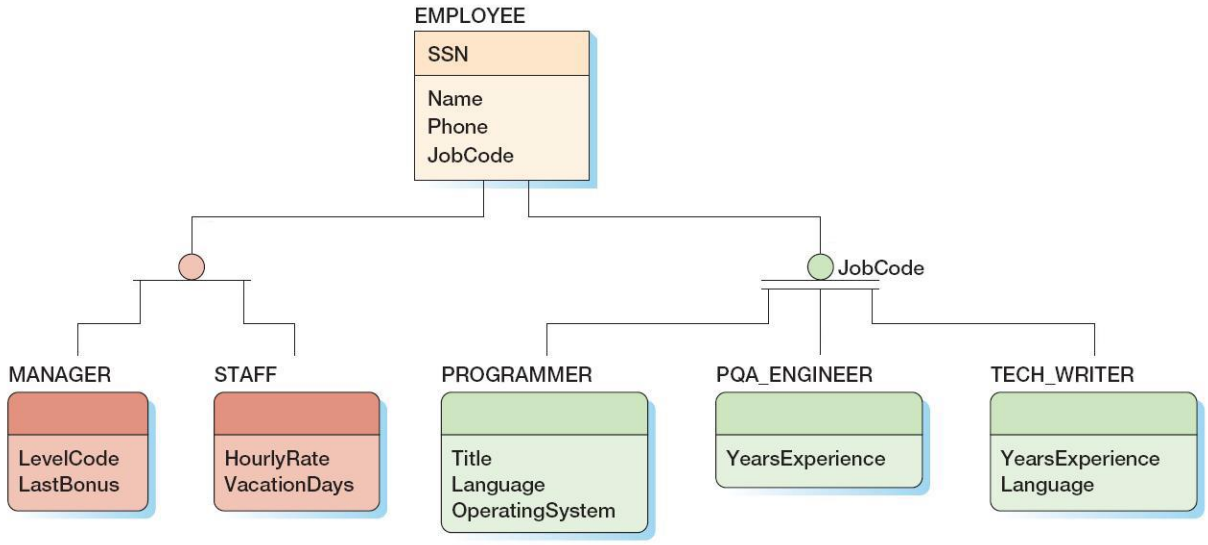


Figure C-7: Incomplete and Complete Category Clusters

BY THE WAY You may be saying, "Wait a minute. Of course, there are other types of employees. What about accountants, for example?" The point is that *according to this model,* there are no other types of employees. Maybe this model is used only in a software development group in which there are no other employee types. Or maybe this model is in error. Whatever the case, however, there are no other types of employees according to this model.

Figure C-7 shows a second category cluster that consists of MANAGER and STAFF category entities. This is an **incomplete category cluster**, which is indicated by placing the category cluster circle on top of a single line; there is no gap between the horizontal lines. This notation means that at least one category is missing. An employee might be a PART_TIME employee, for example.

We stated that within category clusters, category entities are mutually exclusive. This does not mean, however, that an entity cannot have a relationship to two or more category entities in different clusters. Thus, as shown in Figure C-7, an EMPLOYEE can be a MANAGER and also a PROGRAMMER. An EMPLOYEE cannot, however, be both a MANAGER and STAFF.

As you can see, the IDEF1X model adds structure to the generalization/subtype relationships in the extended E-R model. By further defining these concepts, the IDEF1X model makes them more meaningful, and hence more useful.

What Are Domains?

IDEF1X introduced the concept of domains to the extended E-R model. A **domain** is a named set of values that an attribute can have. A domain can consist of a specific list of values or it can be defined more generally, for example, as a set of strings of maximum length 50. As an example of the former, a university could have a domain called DEPARTMENT_NAMES that consists of the names of all official departments at that university. The domain would be defined by enumerating that list: {Accounting, Biology, Chemistry, Computer Science, Information Systems, Management, Physics, etc.}. As an example of the latter, the domain STUDENT_NAMES could be defined as any character string of length less than 75.

Domains Reduce Ambiguity

Domains are both important and useful. For example, in Figure C-7 notice that both PROGRAMMER and TECH_WRITER have an attribute named Language. Without domains, these attribute names are ambiguous. Do they refer to the same thing or not? It might be that Language for PROGRAMMER is a computer language, whereas Language for TECH_WRITER is a human language. Or they may both be computer languages. This ambiguity can be eliminated by specifying the domain upon which each of these attributes is based.

We can define the domain COMPUTER_LANGUAGE to be the list {'C#', 'C++', 'Java', 'VisualBasic.NET'} and the domain HUMAN_LANGUAGE to be {'Canadian French', 'France French', 'Spanish', 'UK English', 'US English'}. If we now say that Language of PROGRAMMER is based on the COMPUTER_LANGUAGE domain and that Language of TECH_WRITER is based on the HUMAN_LANGUAGE domain, the ambiguity between these two attributes is removed.

Furthermore, named domains eliminate ambiguity from attributes whose values look similar but are not the same. In Figure C-7, suppose that staff employees can accrue only 30 days of vacation. Further, suppose that employees are assumed to never work more than 30 years. In this case, the attributes STAFF.VacationDays, PQA_ENGINEER.YearsExperience, and TECH_WRITER.YearsExperience will all have values from 0 to 30. (In this notation, we identify attributes by appending the entity name to the attribute name with a period in between.) Without domain specification, we cannot tell whether these values refer to the same thing or not. Suppose we define a domain VACATION_DAYS as integers from 0 to 30 and a second domain EXPERIENCE as integers from 0 to 30.

Now, we can state that the attribute STAFF.VacationDays is based on the VACATION_DAYS domain; the attributes PQA_ENGINEER .YearsExperience and TECH_WRITER.YearsExperience are based on the EXPE-RIENCE domain.

Domains Are Useful

Domains not only reduce ambiguity, they are also quite useful. Suppose that in a model of a university database we have an attribute called CampusAddress that is used in many different entities. It could be used in STUDENT, PROFESSOR, DEPARTMENT, LAB, and so forth.

Further suppose that we base all of these CampusAddress attributes on the same domain, called CAMPUS_ADDRESS, and define that domain as all values of the pattern BBB-NNN, in which BBB is a list of building codes and NNN is a list of room numbers. When we define the domain in this way, all of the attributes in the model will inherit this definition.

Now, suppose that as we build our model we discover that some room numbers have four digits. Without a domain definition, we would need to find all of the attributes in the model that use a campus address and change them to have NNNN for room number. With domains, we simply change the domain definition, and all of the attributes based on the domain will inherit the change. This not only reduces work, it eliminates errors that are difficult to find and that are hard to fix when they are found.

Another practical use for domains is to assess whether two attributes with different names refer to the same thing. For example, an entity named DEPARTMENT might have an attribute named BudgetCode, and a second entity named PROJECT might have an attribute named ExpenseCategory. Using domains, we can readily check to determine whether these two attributes are based on the same domain. Without domains, it would be difficult to know. Even if attribute values look similar, they may have different meanings.

Base Domains and Typed Domains

IDEF1X defines two types of domains. A **base domain** is a domain having a data type and possibly a value list or range definition. The default data types are Character, Numeric, and Boolean. The specification enables users to define additional data types such as Date, Time, Currency, and so forth. A value list is a set of values like that described for the COMPUTER_LANGUAGE domain, and a range definition is like that described for the EXPERIENCE domain.

A **type domain** is a subset of a base domain or a subset of another type domain. Figure C-8 illustrates type domains based on the DEPARTMENT_NAMES domain. Type domains allow the definition of domain hierarchies that can be used for greater specificity.

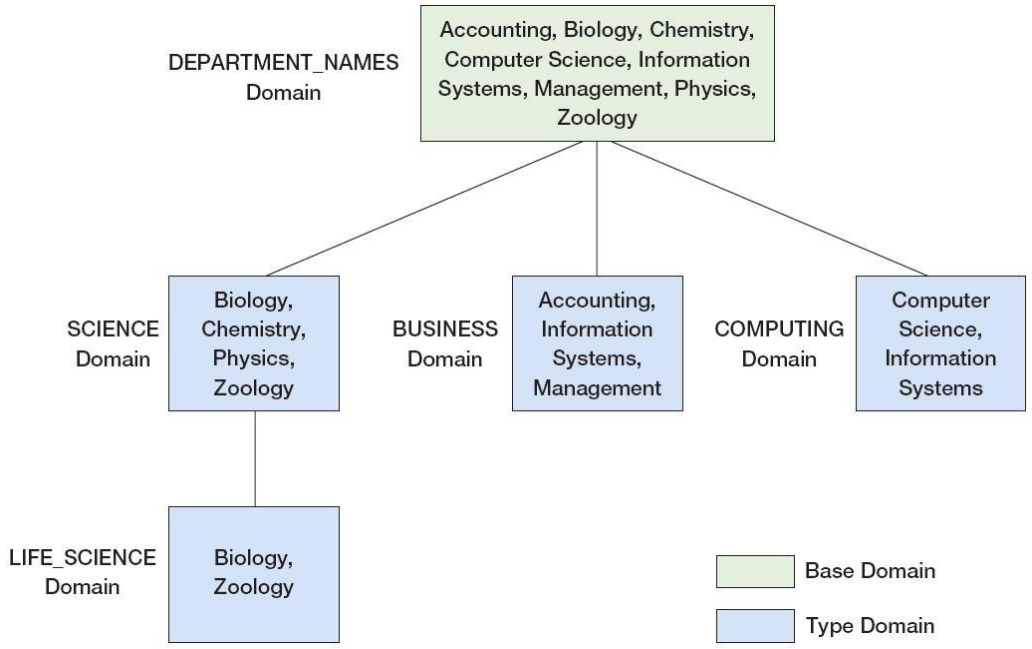


Figure C-8: Example of Domain Hierarchy

Key Terms

base domain	**categorization cluster**
category entity	**complete category cluster**
diamond	**discriminator**
domain	**generic entity**
IDEF1X (Integrated Definition 1, Extended)	**identifying connection relationship**
incomplete category cluster	**IS-A relationship**
mutually exclusive	**nonidentifying connection relationship**
nonspecific IDEF1X relationship	**type domain**

Review Questions

C.1 Why is the IDEF1X model important?

C.2 Name four types of IDEF1X relationships.

C.3 What is a nonidentifying connection relationship? How do such relationships relate to the extended E-R model described in Chapter 5?

C.4 What is an identifying connection relationship? How do such relationships relate to the extended E-R model described in Chapter 5?

C.5 What is a nonspecific relationship? How do such relationships relate to the extended E-R model described in Chapter 5?

C.6 Explain the major differences between categorization relationships and supertype/sub-type relationships in the extended E-R model described in Chapter 5.

C.7 What are the major advantages of the modeling of domains?

C.8 Redraw the E-R diagram in Figure 5-52 using IDEF1X.

C.9 Answer Project Question 5.56, but use IDEF1X instead of the crow's foot model.

C.10 Answer Project Question 5.59, but use IDEF1X instead of the crow's foot model. Redraw Figure 5-57 in IDEF1X notation.