
Database Processing

11th Edition

David M. Kroenke and David J. Auer

APPENDIX E

The Semantic Object Model



Executive Editor: Bob Horan
Editorial Director: Sally Yagan
AVP/Editor-in-Chief: Eric Svendsen
Assistant Editor: Kelly Loftus
Director of Marketing: Patrice Lumumba Jones
Senior Marketing Manager: Anne Fahlgren
Marketing Assistant: Susan Osterlitz
Senior Managing Editor: Judy Leale
Production Project Manager: Kelly Warsak
Senior Operations Supervisor: Arnold Vila
Operations Specialist: Ilene Kahn
Senior Art Director: Janet Slowik

Design Director: Christy Mahon
Cover and Interior Design: Karen Quigley
Manager, Rights and Permissions: Charles Morris
Manager, Cover Visual Research & Permissions: Karen Sanatar
Cover Illustration: Stockbyte/Veer/Corbis
Media Director: Lisa Rinaldi
Lead Media Project Manager: Denise Vaughn
Full-Service Project Management: Jennifer Welsch/BookMasters, Inc.
Composition: Integra Software Services
Printer/Binder: Quebecor World-Versailles
Cover Printer: Lehigh-Phoenix Color/Hagerstown
Text Font: 10/12 Kepler MM

Credits and acknowledgments borrowed from other sources and reproduced, with permission, in this textbook appear on appropriate page within text.

Microsoft® and Windows® are registered trademarks of the Microsoft Corporation in the U.S.A. and other countries. Screen shots and icons reprinted with permission from the Microsoft Corporation. This book is not sponsored or endorsed by or affiliated with the Microsoft Corporation.

MySQL®, the MySQL GUI Tools® (MySQL Query Brower® and MySQL Administrator®), the MySQL Command Line Client®, and MySQL Workbench® are registered trademarks of Sun Microsystems, Inc. in the U.S.A and other countries. Screenshots and icons reprinted with permission of Sun Microsystems, Inc. This book is not sponsored or endorsed by or affiliated with Sun Microsystems.

Eclipse® and The Eclipse PHP Development Tools (PDT) Project® are trademarks of the Eclipse Foundation, Inc. The Eclipse platform is copyright Eclipse copyright holders and others, 2000, 2007. Screenshots reprinted under the terms of the Eclipse Public License v1.0 available at www.eclipse.org/legal/epl-v10.html. This book is not sponsored or endorsed by or affiliated with the Eclipse Foundation, Inc.

PHP is copyright The PHP Group 1999–2008, and is used under the terms of the PHP Public License v3.01 available at http://www.php.net/license/3_01.txt. This book is not sponsored or endorsed by or affiliated with The PHP Group.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Copyright © 2010, 2006, 2004, 2000, 1998 Pearson Education, Inc., publishing as Prentice Hall, One Lake Street, Upper Saddle River, New Jersey 07458. All rights reserved. Manufactured in the United States of America. This publication is protected by Copyright, and permission should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission(s) to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458.

Many of the designations by manufacturers and seller to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed in initial caps or all caps.

Library of Congress Cataloging-in-Publication Data

Kroenke, David.

Database processing: fundamentals, design, and implementation/David M. Kroenke, David J. Auer.—11th ed.
p. cm.

Includes bibliographical references and index.

ISBN 978-0-13-230267-8 (casebound : alk. paper)

1. Database management. I. Auer, David J. II. Title.

QA76.9.D3K7365 2009
005.74—dc22

2009011144

10 9 8 7 6 5 4 3 2 1

Prentice Hall
is an imprint of



www.pearsonhighered.com

ISBN 10: 0-13-230267-5
ISBN 13: 978-0-13-230267-8

Appendix

E

The Semantic Object Model

Chapter Objectives

- To define the term semantic object.
- To define and illustrate the use of simple, group, and object link attributes.
- To demonstrate the use of semantic object diagrams to build a data model.
- To define and illustrate the seven basic types of semantic objects.
- Compare the semantic object model to the entity-relationship model.

This appendix discusses the semantic object model, which is used to create data models like the E-R model discussed in Chapters 5 and 6. As shown in Figure E-1, the development team interviews users; analyzes the users' reports, forms, and queries; and constructs a model of the users' data from these reports. This data model is later transformed into a database design.

The particular form of the data model depends on the constructs used to build it. If an E-R model is used, the model will have entities, relationships, and the like. If a semantic model is used, the model will have semantic objects and related constructs, which are discussed here.

The E-R model and the semantic object model are like lenses through which the database developers look when studying and documenting the users' data. Both lenses work, and they both ultimately result in a database design. They use different lenses to form that design, however; and because the lenses create different images, the designs they produce may not be exactly the same.

When developing a database, you must decide which approach to use, just as a photographer needs to decide which lens to use. Each approach has strengths and weaknesses, which we discuss at the end of this appendix.

The semantic object model was first presented in 1988, in the third edition of this text. It is based on concepts that were developed and published by Codd and by Hammer and McLeod.¹ The semantic object model is a data model. It differs from **object-oriented database processing**, which was discussed in Chapter 1. Whereas object-oriented databases concern the storage of OOP objects, the semantic object model is a data model and is an alternative to the E-R model.

¹ E. F. Codd, "Extending the Relational Model to Capture More Meaning," *ACM Transactions on Database Systems*, December 1976, pp. 397–424; and Michael Hammer and Dennis McLeod, "Database Description with SDM: A Semantic Database Model," *ACM Transactions on Database Systems*, September 1981, pp. 351–386.

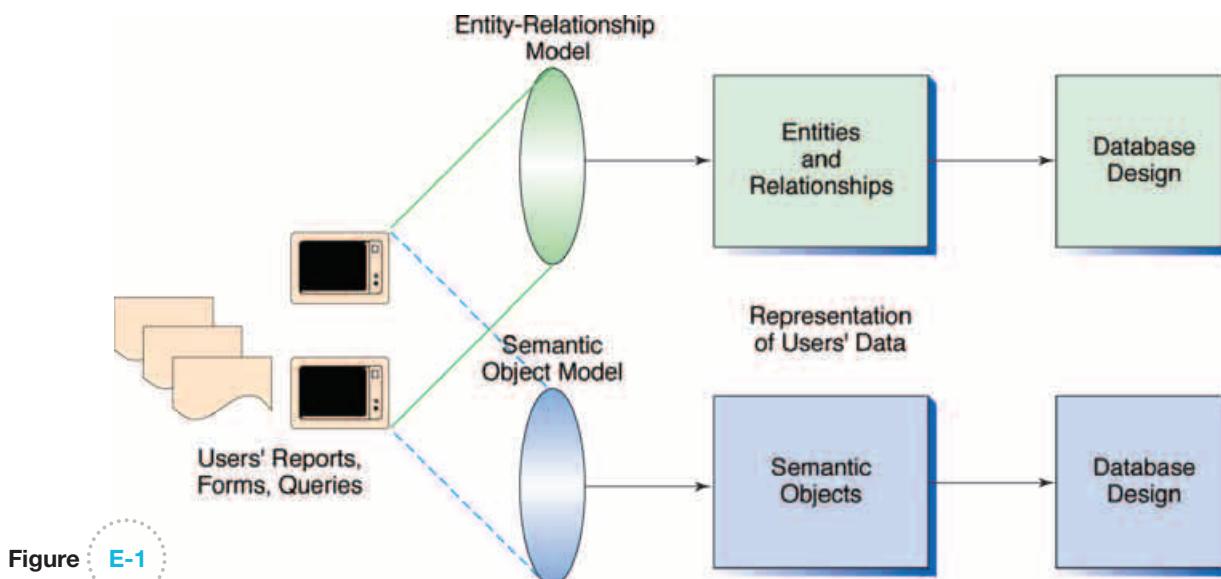


Figure E-1

Using Different Data Models
for Database Designs

Semantic Objects

The purpose of a database application is to provide forms, reports, and queries so that the users can track entities or objects important to their work. The goals of the early stages of database development are to determine the things to be represented in the database, to specify the characteristics of those things, and to establish the relationships among them.

In Chapters 5 and 6, we referred to these things as **entities**. In this appendix, we refer to them as **semantic objects**, or sometimes as just **objects**. The word **semantic** means “meaning,” and a semantic object is one that models, in part, the “meaning” of the users’ data. Semantic objects model the users’ perceptions more closely than does the E-R model. We use the adjective *semantic* with the word *object* to distinguish the objects discussed in this appendix from the objects defined in object-oriented programming (OOP) languages.

Defining Semantic Objects

Entities and objects are similar in some ways, but they differ in others. We begin with the similarities. A semantic object is a representation of some identifiable thing in the users’ work environment. More formally, a semantic object is a *named collection of attributes that sufficiently describes a distinct identity*.

Like entities, semantic objects are grouped into **object classes**. An object class has an **object class name** that distinguishes it from other classes and that corresponds to the names of the things it represents. Thus, a database that supports users who work with student records has an object class called STUDENT. Note that object class names, like entity class names, are spelled with capital letters. A particular semantic object is an **instance** of the class. Thus, ‘William Jones’ is an instance of the STUDENT class, and ‘Accounting’ is an instance of the DEPARTMENT class.

Like entities, an object has a **collection of attributes**. Each attribute represents a characteristic of the identity being represented. For instance, the STUDENT object can have attributes such as Name, HomeAddress, CampusAddress, DateOfBirth, DateOfGraduation, and Major. This collection of attributes also is a **sufficient description**, which means that the attributes represent all of the characteristics that the users need in order to do their work. As we stated in Chapter 5, things in the world have an infinite set of characteristics; we cannot represent all of them. Instead, we represent those necessary for the users to satisfy their information needs so that they can successfully perform their jobs. Sufficient description also means that the objects are complete in and of themselves. All of the data required about a

CUSTOMER, for example, is located in the CUSTOMER object, so we need not look anywhere else to find data about CUSTOMERS.

Objects represent **distinct identities**—they are something that users recognize as independent and separate and that they want to track and report. These identities are the nouns about which the information is to be produced. To understand the term *distinct identity*, recall that there is a difference between objects and object instances. CUSTOMER is the name of an object, and ‘CUSTOMER 12345’ is the name of an instance of an object. When we say that an object represents a distinct identity, we mean that users consider each *instance* of an object to be unique and identifiable in its own right.

Finally, note that the identities that the objects represent may or may not have a **physical existence**. For example, EMPLOYEEs physically exist, but ORDERs do not. Orders are models of a contractual agreement to provide certain goods or services under certain terms and conditions. They are not physical things, but rather representations of agreements. Thus, something need not be physical to be considered an object; it need only be identifiable in its own right in the minds of the users.

Attributes

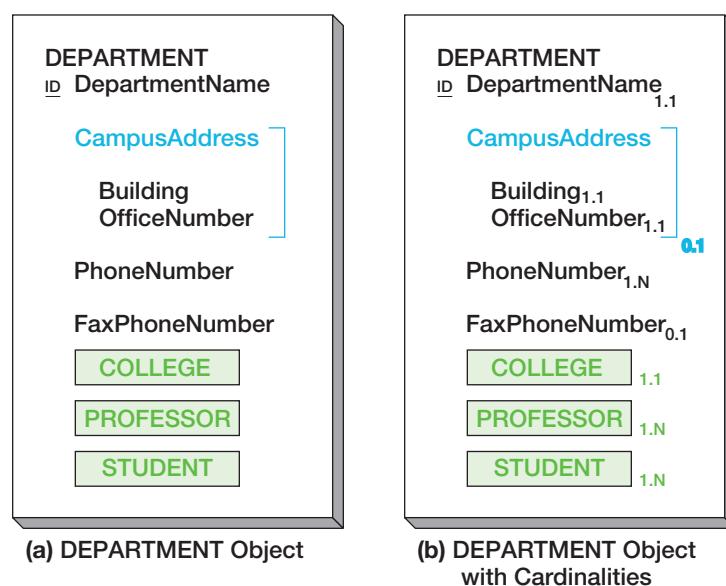
Semantic objects have attributes that define their characteristics. There are three types of attributes. **Simple attributes** have a single element. Examples are DateOfHire, InvoiceNumber, and SalesTotal. **Group attributes** are composites of other attributes. One example is Address, which contains the attributes {Street, City, State, Zip}; another example is FullName, which contains the attributes {FirstName, MiddleInitial, LastName}. **Semantic object attributes** are attributes that establish a relationship between one semantic object and another.

To understand these statements better, look at Figure E-2(a), which is an example of a **semantic object diagram**, or **object diagram**. Such diagrams are used by development teams to summarize the structures of objects and to present them visually. Objects are shown in portrait-oriented rectangles. The name of the object appears at the top, and attributes are written in order after the object name.

The DEPARTMENT object contains an example of each of the three types of attributes. DepartmentName, PhoneNumber, and FaxPhoneNumber are simple attributes, each of which represents a single data element. CampusAddress is a group attribute containing the simple attributes Building and OfficeNumber. Finally, COLLEGE, PROFESSOR, and STUDENT each are semantic object attributes, which means that those objects are connected to and logically contained in DEPARTMENT.

The object attributes, or **object links** as they are sometimes called, mean that when a user thinks about a DEPARTMENT, he or she thinks not only about DepartmentName, CampusAddress,

Figure E-2
DEPARTMENT Object
Diagram



PhoneNumber, and FaxPhoneNumber, but also about the COLLEGE, PROFESSORs, and STUDENTs that are related to that department. Because COLLEGE, PROFESSOR, and STUDENT also are objects, the complete data model contains object diagrams for them, too. The COLLEGE object contains attributes of the college; the PROFESSOR object contains attributes of the faculty; and the STUDENT object contains attributes of the students.

Attribute Cardinality

Each attribute in a semantic object has both a minimum cardinality and a maximum cardinality. The **minimum cardinality** indicates the number of instances of the attribute that must exist for the object to be valid. Usually, this number is either zero or one. If it is zero, the attribute is not required to have a value; if it is one, it must have a value. Although it is unusual, the minimum cardinality can sometimes be larger than one. For example, the attribute PLAYER in an object called BASKETBALL-TEAM might have a minimum cardinality of five because this is the smallest number of players required to make up a basketball team.

The **maximum cardinality** indicates the maximum number of instances of the attribute that the object may have. It is usually either one or N. If it is one, the attribute can have no more than one instance; if it is N, the attribute can have many values, and the absolute number is not specified. Sometimes, the maximum cardinality is a specific number, such as five, meaning that the object can contain no more than exactly five instances of the attribute. For example, the attribute PLAYER in BASKETBALL-TEAM might have a maximum cardinality of 15, which would indicate that no more than 15 players could be assigned to a team's roster.

Cardinalities are shown as subscripts of attributes in the format $n.m$, where n is the minimum cardinality and m is the maximum cardinality. In Figure E-2(b), the minimum cardinality of DepartmentName is one and the maximum is also one, which means that exactly one value of DepartmentName is required. The cardinality of PhoneNumber is 1.N, meaning that a DEPARTMENT is required to have at least one PhoneNumber, but may have many. The cardinality of 0.1 in FaxPhoneNumber means that a DEPARTMENT may have either zero or one FaxPhoneNumber.

The cardinalities of groups and the attributes in groups can be subtle. Consider the attribute CampusAddress. Its cardinalities are 0.1, meaning that a DEPARTMENT need not have an address and has at most one. Now, examine the attributes inside CampusAddress. Both Building and OfficeNumber have the cardinalities 1.1. You might be wondering how a group can be optional if the attributes in that group are required. The answer is that the cardinalities operate only between the attribute and the container of that attribute. The minimum cardinality of CampusAddress indicates that there need not be a value for address in DEPARTMENT, but the minimum cardinalities of Building and OfficeNumber indicate that both Building and OfficeNumber must exist in CampusAddress. Thus, a CampusAddress group need not appear; but if one does, it must have a value for both Building and OfficeNumber.

Object Instances

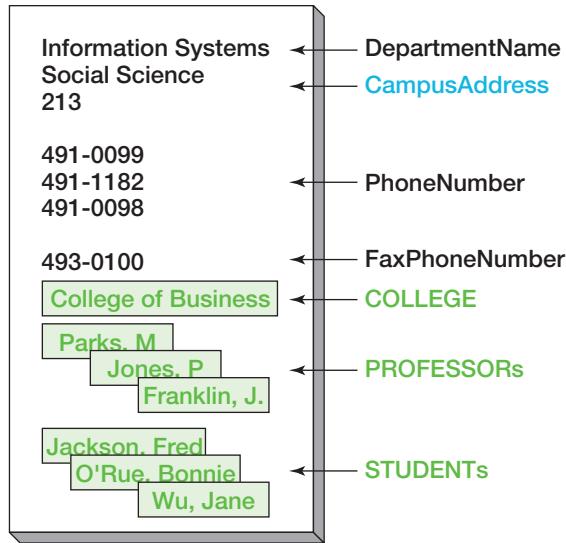
The object diagrams for DEPARTMENT shown in Figure E-2 are a format, or general structure, that can be used for any department. An instance of the DEPARTMENT object is shown in Figure E-3, with each attribute's value for a particular department.

The DepartmentName is Information Systems, and it is located in Room 213 of the Social Science Building. Observe that there are three values for PhoneNumber—the Information Systems Department has three phone lines in its office. Other departments may have fewer or more, but every department has at least one.

Furthermore, there is one instance of COLLEGE—the College of Business, and there are multiple values for the PROFESSOR and STUDENT object attributes. Each of these object attributes is a complete object; each has all the attributes defined for an object of that type. To keep this diagram simple, only the identifying names are shown for each of the instances of object attribute.

An object diagram is a picture of the user's perception of an object in the work environment. Thus, in the user's mind, the DEPARTMENT object includes all of these data. A DEPARTMENT logically contains data about the COLLEGE in which it resides as well as the PROFESSORs and STUDENTs who are related to that department.

Figure E-3
An Instance of the
DEPARTMENT Object
in Figure E-2



Paired Attributes

The semantic object model has no one-way object relationships. If an object contains another object, the second object will contain the first. For example, if DEPARTMENT contains the object attribute COLLEGE, then COLLEGE will contain the matching object attribute DEPARTMENT. These object attributes are called **paired attributes**, because they always occur as a pair.

Why must object attributes be paired? The answer lies in the way in which human beings think about relationships. If object A has a relationship with object B, then object B will have a relationship with object A. At the very least, B is related to A in the relationship of “things that are related to B.” If this argument seems obscure, try to envision a one-way relationship between two objects. It cannot be done.

Object Identifiers

An **object identifier** is one or more object attributes that the users employ to identify object instances. Such identifiers are potential names for a semantic object. In CUSTOMER, for example, possible identifiers are CustomerID and CustomerName. These are attributes that users consider to be valid names of CUSTOMER instances. Compare these identifiers with attributes such as DateOfFirstOrder, StockPrice, and NumberOfEmployees. Such attributes are not identifiers because the users do not think of them as names of CUSTOMER instances.

A **group identifier** is an identifier that has more than one attribute. Examples are {FirstName, LastName}, {FirstName, PhoneNumber}, and {State, License Number}.

Object identifiers may or may not be unique, depending on how the users view their data. For example, InvoiceNumber is a unique identifier for ORDER, but StudentName is not a unique identifier for STUDENT. For example, two students may be named “Mary Smith.” If so, the users will employ StudentName to identify a group of one or more students and then, if necessary, use values of other attributes to identify a particular member of that set.

In semantic object diagrams, object identifiers are denoted by the letters *ID* in front of the attribute. If the identifier is unique, these letters will be underlined. In Figure E-2(b), for example, the attribute DepartmentName is a unique identifier of DEPARTMENT.

Normally, if an attribute is to be used as an identifier, its value is required. Also, generally there is no more than one value of an identifier attribute for a given object. In most cases, therefore, the cardinality of an ID attribute is 1.1, so we use this value as a default.

However, in a few, rare cases the cardinality of an identifier may be other than 1.1. Consider, for example, the attribute Alias in the semantic object PERSON. A person need not have an alias, or he or she may have several aliases. Hence, the cardinality of Alias is 0.N. Showing the

subscripts of all attributes clutters the semantic object diagram. To simplify, we will assume that the cardinalities of simple-value identifier attributes are 1..1 and that the cardinalities of other simple-value attributes are 0..1. If the cardinality of the simple-value attribute is other than these assumptions, we will show it on the diagram. Otherwise, subscripts on simple-value attributes will be omitted.

Attribute Domains

The **domain** of an attribute is a description of an attribute's possible values. The characteristics of a domain depend on the type of the attribute. The domain of a simple attribute consists of both a physical and a semantic description. The physical description indicates the type of data (for example, numeric versus string), the length of the data, and other restrictions or constraints (for example, the first character must be alphabetic or the value must not exceed 9999.99). The semantic description indicates the function or purpose of the attribute—it distinguishes this attribute from other attributes that might have the same physical description. For example, the domain of DepartmentName can be defined as “the set of strings of up to seven characters that represent names of departments at Highline University.” The phrase *strings of up to seven characters* is the physical description of the domain, and the phrase *that represent names of departments at Highline University* is the semantic description. The semantic description differentiates strings of seven characters that represent names of departments from similar strings that represent names of courses, buildings, or some other attribute.

In some cases, the physical description of a simple attribute domain is an **enumerated list**, the set of an attribute's specific values. The domain of the attribute PartColor, for example, might be the enumerated list {‘Blue’, ‘Yellow’, ‘Red’}.

The domain of a group attribute also has a physical and a semantic description. The physical description is a list of all of the attributes in the group and the order of those attributes. The semantic description is the function or purpose of the group. Thus, the physical domain description of CampusAddress (shown in Figure E-2) is the list {Building, OfficeNumber}; the semantic description is *the location of an office at Highline University*.

The domain of an object attribute is the set of object instances of that type. In Figure E-2, for example, the domain of the PROFESSOR object attribute is the set of all PROFESSOR object instances in the database. The domain of the COLLEGE object is the set of all COLLEGES in the database. In a sense, the domain of an object attribute is a dynamically enumerated list; the list contains all of the object instances of a particular type.

Semantic Object Views

Users access the values of object attributes through database applications that provide data entry forms, reports, and queries. In most cases, such forms, reports, and queries do not require access to all of an object's attributes. For example, Figure E-4 shows two application views of DEPARTMENT. Some attributes of DEPARTMENT (its DepartmentName, for example) are visible in both application views. Other attributes are visible in only one. For example, STUDENT is seen only in the StudentListing View, but PROFESSOR is visible only in the Staff View.

The portion of an object that is visible to a particular application is called the **semantic object view**, or simply the **view**. A view consists of the name of the object plus a list of all of the attributes visible from that view.

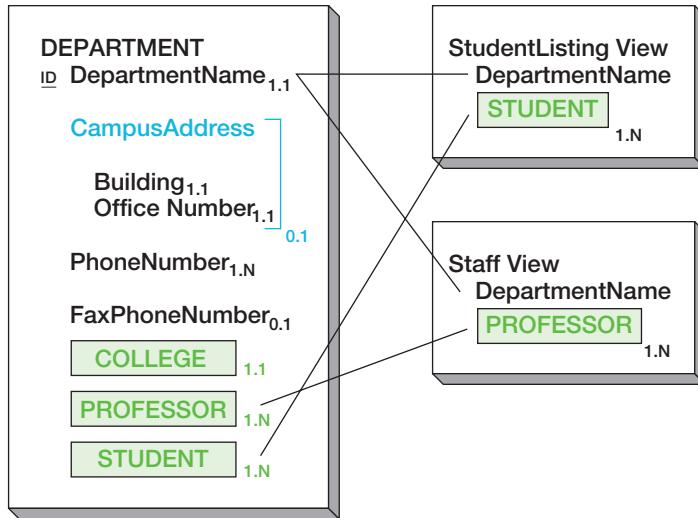
Views are used in two ways. First, when developing the data model, the database and application developers can use views to work backward. That is, they begin with the forms, reports, and queries that the users say they need and then work backward to the database design.

To do this, the team selects a required form, report, or query and determines the view that must exist in order for the form, report, or query to be created. Then, the team selects the next form, report, or query and does the same. These two views are then integrated. This process is repeated until the structure of the entire database has been created.

The second way in which views are used occurs after the database structure has been created. At this point, views are constructed to support new forms, reports, and queries based on the existing database structure. Examples of this second use were shown for SQL Server 2008, Oracle Database 11G, and MySQL 5.1 in Chapters 10, 10A, and 10B, respectively.

Figure E-4

StudentListing and Staff Views of the DEPARTMENT Semantic Object



Types of Objects

This section describes and illustrates seven types of objects. For each type, we examine a report or form and show how to model that report or form with an object. Then, we transform each of these types of objects into database designs.

Three new terms are used in this section: A **single-value attribute** is an attribute whose maximum cardinality is one; a **multivalue attribute** is one whose maximum cardinality is greater than one; and a **nonobject attribute** is a simple or group attribute.

Simple Objects

A **simple object** is a semantic object that contains only single-value, simple, or group attributes. An example is shown in Figure E-5. Figure E-5(a) shows two instances of a report called an *Equipment Tag*. Such tags are applied to items of office equipment to help keep track of inventory. These tags can be considered a report.

Figure E-5(b) shows a simple object, EQUIPMENT, that models Equipment Tag. The attributes of the object include the items shown on the tag: EquipmentNumber, Description, AcquisitionDate, and PurchaseCost. Note that none of these attributes are multivalue, and none are object attributes. Hence, EQUIPMENT is a simple object.

Figure E-5(c) is an example of a simple object, EQUIPMENT, which can be represented by a single relation, as shown in Figure E-5(c). Each attribute of the object is defined as an

Figure E-5

Examples of a Simple Object

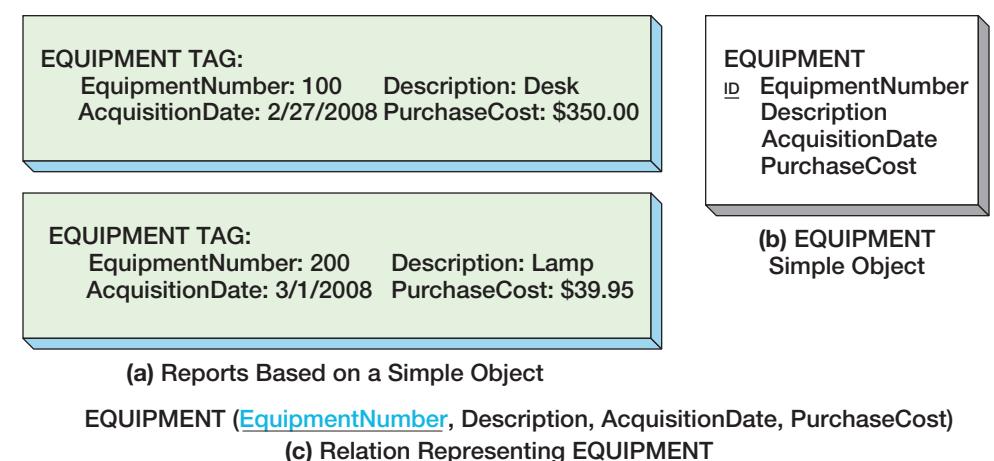
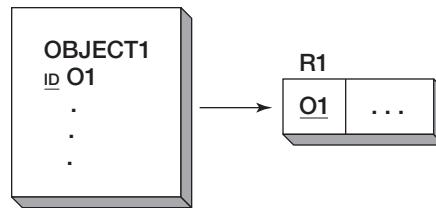


Figure E-6

General Transformation
of a Simple Object into
a Relation



attribute of the relation. The identifying attribute, EquipmentNumber, becomes the key attribute of the relation, denoted by underlining EquipmentNumber in Figure E-5(c).

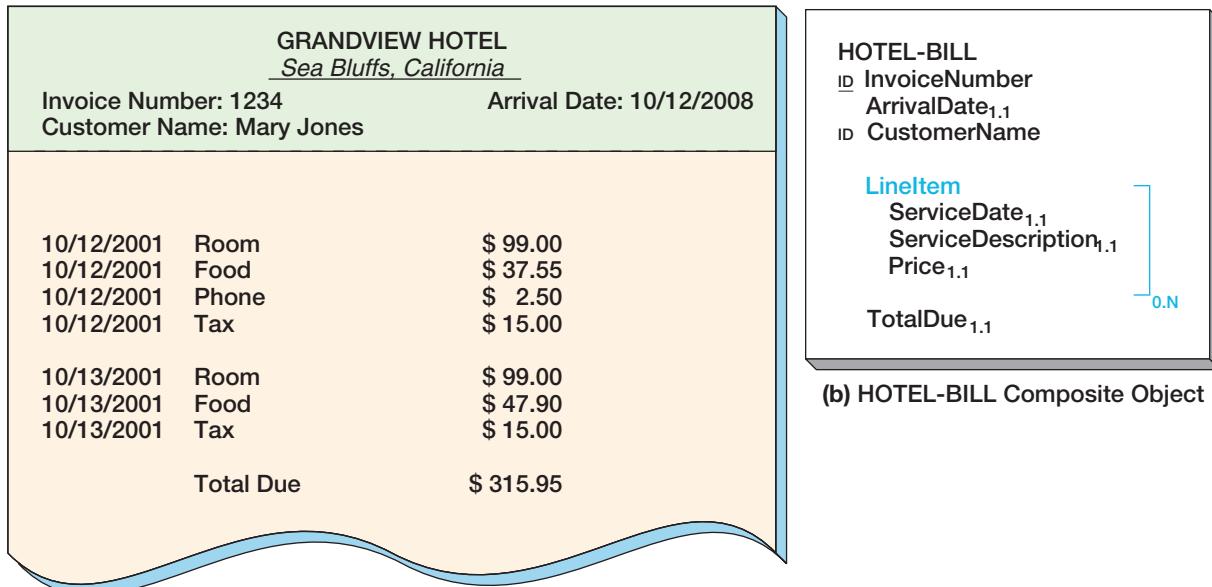
The general transformation of simple objects is illustrated in Figure E-6. Object OBJECT1 is transformed into relation R1. The attribute that identifies OBJECT1 instances is O1; it becomes the key of relation R1. Nonkey data are represented in this and subsequent figures with ellipses (...).

Because a key is an attribute that uniquely identifies a row of a table, only unique identifiers—those with the ID underlined—can be transformed into keys. If there is no unique identifier in the object, one must be created by combining the existing attributes to form a unique identifier or by defining a surrogate key.

Composite Objects

Figure E-7

Examples of a Composite
Object



HOTEL-BILL (InvoiceNumber, ArrivalDate, CustomerName, TotalDue)

LINEITEM (InvoiceNumber, ServiceDate, ServiceDescription, Price)

Referential integrity constraint:

InvoiceNumber in LINEITEM must exist
in InvoiceNumber in HOTEL-BILL

(c) Relational Representation of HOTEL-BILL

ArrivalDate, CustomerName, and TotalDue. It also contains a group of attributes that is repeated for services provided to the guest. Each group includes ServiceDate, ServiceDescription, and Price.

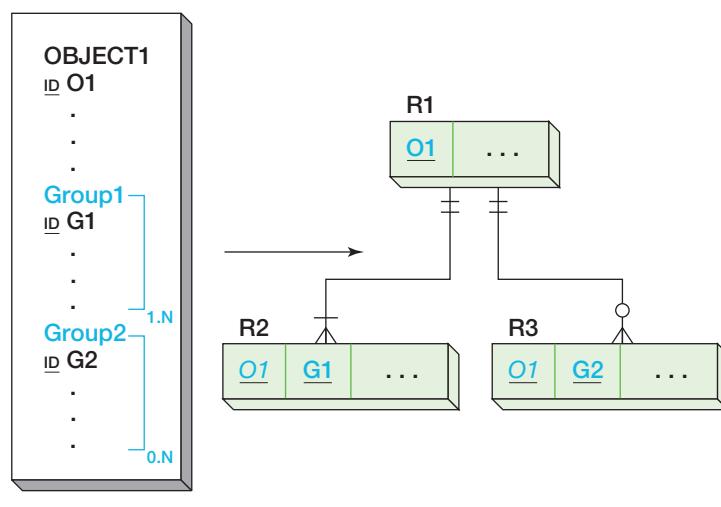
Figure E-7(b) shows an object diagram for the HOTEL-BILL object. The attribute LineItem is a group attribute having a maximum cardinality of N, which means that the group ServiceDate, ServiceDescription, Price can occur many times in an instance of the HOTEL-BILL semantic object.

LineItem is not modeled as an independent semantic object; instead, it is considered to be an attribute within a HOTEL-BILL. This design is appropriate because the hotel does not view one line of a guest's charges as a separate thing, so line items on the guest's bill do not have identifiers of their own. No employee attempts to enter a LineItem except in the context of a bill. The employee enters the data for bill number 1234 and then, in the context of that bill, enters the charges. Or the employee retrieves an existing bill and enters additional charges in the context of that bill.

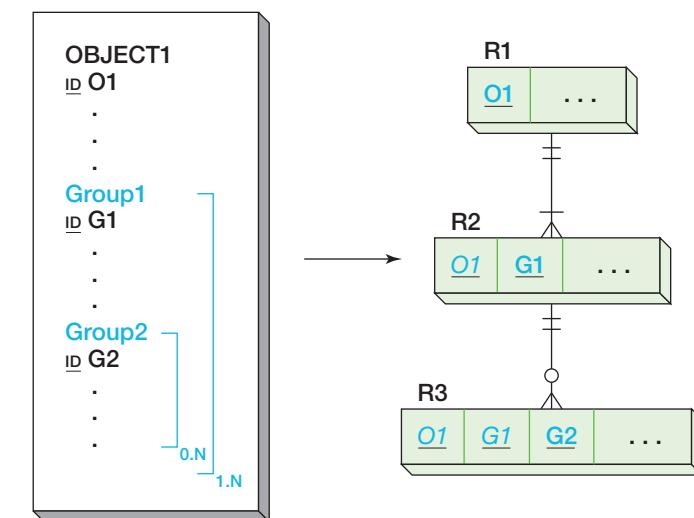
The minimum cardinality of LineItem is zero, which means that a HOTEL-BILL object can exist without any LineItem data. This allows a bill to be started when the customer checks in

Figure E-8

General Transformation of Composite Objects



(a) Composite Objects with Separate Groups



(b) Composite Objects with Nested Groups

and before there are any charges. If the minimum cardinality were one, no HOTEL-BILL could be started until there was at least one charge. This design decision must be made in light of the business rules. It may be that the hotel's policy is not to start the bill until there has been a charge. If so, then the minimum cardinality of LineItem should be one.

To represent this object, one relation is created for the base object, HOTEL-BILL, and an additional relation is created for the repeating group attribute, DailyCharge. This relational design is shown in Figure E-7(c).

In the key of DAILY-CHARGE, InvoiceNumber is underlined because it is part of the key of DAILY-CHARGE, and it is italicized because it is also a foreign key. (It is a key of HOTEL-BILL.) ChargeDate is underlined because it is part of the key of DAILY-CHARGE, but it is not italicized because it is not a foreign key.

In general, composite objects are transformed by defining one relation for the object itself and another relation for each multivalue attribute. In Figure E-8(a), object OBJECT1 contains two groups of multivalue attributes, each of which is represented by a relation in the database design. The key of each of these tables is the composite of the identifier of the object plus the identifier of the group. Thus, the representation of OBJECT1 is a relation R1 with key O1, a relation R2 with key (O1, G1), and a relation R3 with key (O1, G2).

The minimum cardinality from the object to the group is specified by the minimum cardinality of group attribute. In Figure E-8(a), the minimum cardinality of Group1 is one and that of Group2 is zero. These cardinalities are shown as a hash mark (on R2) and an oval (on R3) in the data structure diagram. The minimum cardinality from the group to the object is always one by default because a group cannot exist if the object that contains that group does not exist. These minimum cardinalities are shown by hash marks on the relationship lines into R1.

Groups can be nested. Figure E-8(b) shows an object in which Group2 is nested within Group1. When this occurs, the relation representing the nested group is made subordinate to the relation that represents its containing group. In Figure E-8(b), relation R3 is subordinate to relation R2. The key of R3 is the key of R2, which is (O1, G1) plus the identifier of Group2, which is G2; thus the key of R3 is (O1, G1, G2).

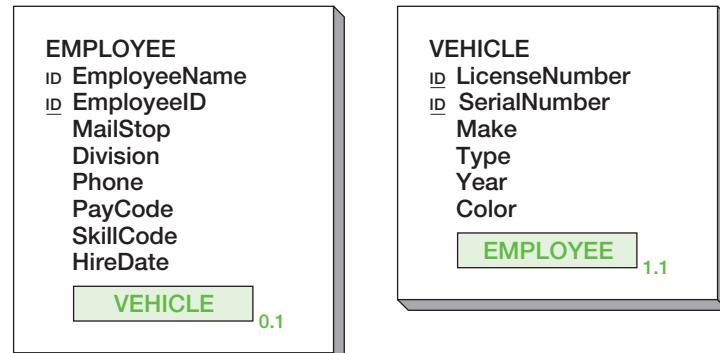
Figure E-9

Compound Objects with 1:1 Paired Properties

VEHICLE DATA			
License number		Serial number	
Make	Type	Year	Color
Employee assignment			

EMPLOYEE WORK DATA			
Employee name		Employee ID	
MailStop		Division	Phone
Pay code	Skill code	Hire date	Auto assigned

(a) Example Vehicle and Employee Data Entry Forms



(b) EMPLOYEE and VEHICLE Compound Objects

Make sure that you understand why the keys in Figure E-8(b) are constructed as they are. Also note that some attributes are underlined and italicized and some are simply underlined because some attributes are both local and foreign keys and some are just local keys.

Compound Objects

A **compound object** contains at least one object attribute. Figure E-9(a) shows two different data entry forms. One form, used by the company's motor pool, is used to keep track of the vehicles. The second form is used to maintain data about the employees. According to these forms, a vehicle is assigned to at most one employee, and an employee has at most one auto assigned.

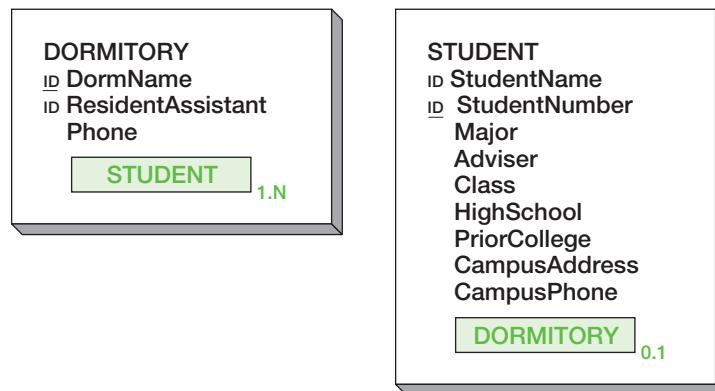
Figure E-10

Compound Objects with 1:N Paired Properties

DORMITORY OCCUPANCY REPORT		
Dormitory	Resident Assistant	Phone
Ingersoll	Sarah and Allen French	3-5567
<u>Student name</u>	<u>Student Number</u>	<u>Class</u>
Adams, Elizabeth	710	SO
Baker, Rex	104	FR
Baker, Brydie	744	JN
Charles, Stewart	319	SO
Scott, Sally	447	SO
Taylor, Lynne	810	FR

StudentName	Horan, Bob
StudentNumber	345
Major	Accounting
Adviser	Julian Jackson
Class	SO
HighSchool	St Andrews, Jacksonville
PriorCollege	None
CampusAddress	Ingersoll #308
CampusPhone	3-7782

(a) Example Dormitory Report and Student Data Entry Forms



(b) DORMITORY and STUDENT Compound Objects

We cannot tell from these forms whether an auto must be assigned to an employee or whether every employee must have an auto. To obtain that information, we have to ask the users in the motor pool or the human resources department. Assume that we find out that an EMPLOYEE need not have a VEHICLE, but that a VEHICLE must be assigned to an employee.

Figure E-9(b) shows object diagrams for EMPLOYEE and VEHICLE. An EMPLOYEE contains VEHICLE as one of its attributes, and VEHICLE in turn contains EMPLOYEE as one of its attributes. Because both EMPLOYEE and VEHICLE contain object attributes, they both are compound objects. Furthermore, because neither attribute is multivalue, the relationship from EMPLOYEE to VEHICLE is one to one, or 1:1.

In Figure E-9(a), the Employee and Vehicle forms contain each other. That is, VEHICLE DATA has a field Employee assignment, and EMPLOYEE WORK DATA has a field Auto assigned. However, this is not always the case; sometimes the relationship can appear in only one direction. Consider the report and form in Figure E-10(a), which concern two objects: DORMITORY and STUDENT. From the Dormitory Occupancy Report, we can see that users think of a dorm as having attributes regarding the dorm (Dormitory, ResidentAssistant, Phone) and also attributes regarding the students (StudentName, StudentNumber, Class) who live in the dorm.

In contrast, the Student Data Form shows only student data; it does not include any dormitory data. (The campus address might contain a dorm address, but, if so, it is apparently not important enough to document on the form. In a database development project, this possibility should be checked out with the users in an interview. Here, we will assume that the Student Data Form does not include dormitory data.)

As stated earlier, object attributes always occur in pairs. Even if the forms, reports, and queries indicate that only one side of the relationship can be seen, both sides of the relationship always exist. By analogy, a bridge that connects two islands touches both islands and can be used in both directions, even if the bridge is, by custom or law, a one-way bridge.

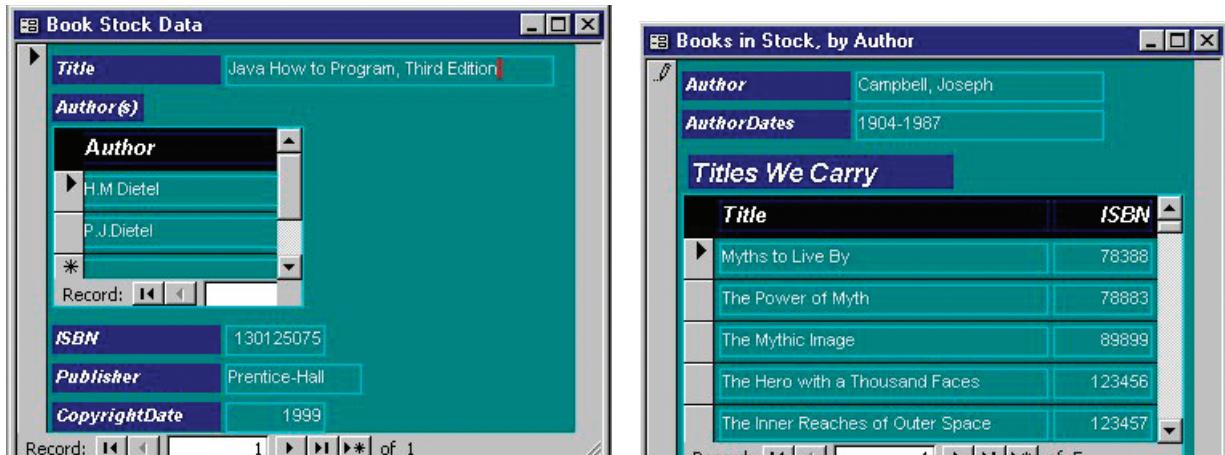
When no form or report can be found to document one side of a relationship, the development team must ask the users about the cardinality of that relationship. In this case, the team needs to find out how many DORMITORY(ies) a STUDENT could have and whether a STUDENT must be related to a DORMITORY. Here, let us suppose that the answers to these questions are that a STUDENT is related to just one DORMITORY and may be related to no DORMITORY. Thus, in Figure E-10(b), DORMITORY contains multiple values of STUDENT, and STUDENT contains one value of DORMITORY. The relationship from DORMITORY to STUDENT is one to many, or 1:N.

A third illustration of compound objects appears in Figure E-11(a). From these two forms, we can deduce that one book can be written by many authors (from the Book Stock Data form) and that one author can write many books (from the Books in Stock, by Author form). Thus, in Figure E-11(b), the BOOK object contains many values of AUTHOR, and AUTHOR contains many values of BOOK. Hence, the relationship from BOOK to AUTHOR is many to many, or N:M. Furthermore, a BOOK must have an AUTHOR, and an AUTHOR (to be an author) must have written at least one BOOK. Therefore, both of these objects have a minimum cardinality of one.

Figure E-12 summarizes the four types of compound objects. In general, OBJECT-1 can contain a maximum of one or many OBJECT-2s. Similarly, OBJECT-2 can contain one or many OBJECT-1s. All of these relationships involve some variation of one-to-one, one-to-many, or many-to-many relationships. Specifically, the relationship from OBJECT1 to OBJECT2 can be 1:1, 1:N, or N:M, whereas the relationship from OBJECT2 to OBJECT1 can be 1:1, 1:M, or M:N. To represent any of these, we need only address the representation of the 1:1, 1:N, and N:M variations.

Representing One-to-One Compound Objects

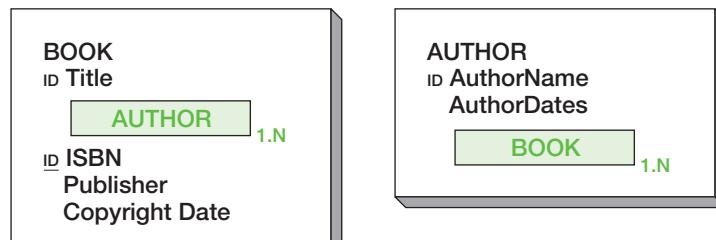
Consider the assignment of a LOCKER to a health club MEMBER. A LOCKER is assigned to one MEMBER, and each MEMBER has one and only one LOCKER. Figure E-13(a) shows the object diagrams. To represent these objects with relations, we define a relation for each object; and, as with 1:1 entity relationships, we place the key of either relation in the other relation. That is, we can place the key of MEMBER in LOCKER or the key of LOCKER in MEMBER. Figure E-13(b) shows the placement of the key of LOCKER in MEMBER. Note that LockerNumber is underlined in LOCKER because it is the key of LOCKER and is italicized in MEMBER because it is a foreign key in MEMBER.



(a) Example Bookstore Data Entry Forms

Figure E-11

Compound Objects with
N:M Paired Properties



(b) BOOK and AUTHOR Compound Objects

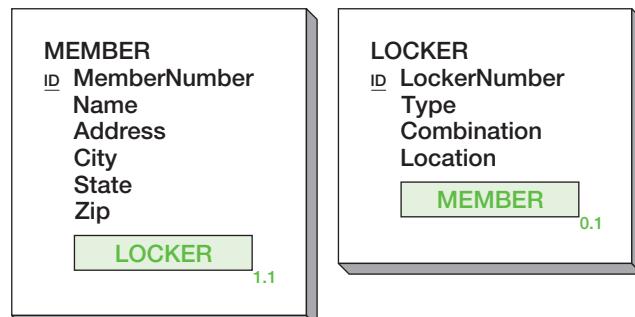
Figure E-12

Four Types of Compound
Objects

Object1 Can Contain		
Object2	One	Many
Can	1:1	1:N
Contain	M:1	M:N

Figure E-13

Example Relational
Representation of 1:1
Compound Objects



(a) Example 1:1 Compound Objects

MEMBER (MemberNumber, Name, Address, City, State, Zip, LockerNumber)

LOCKER (LockerNumber, Type, Combination, Location)

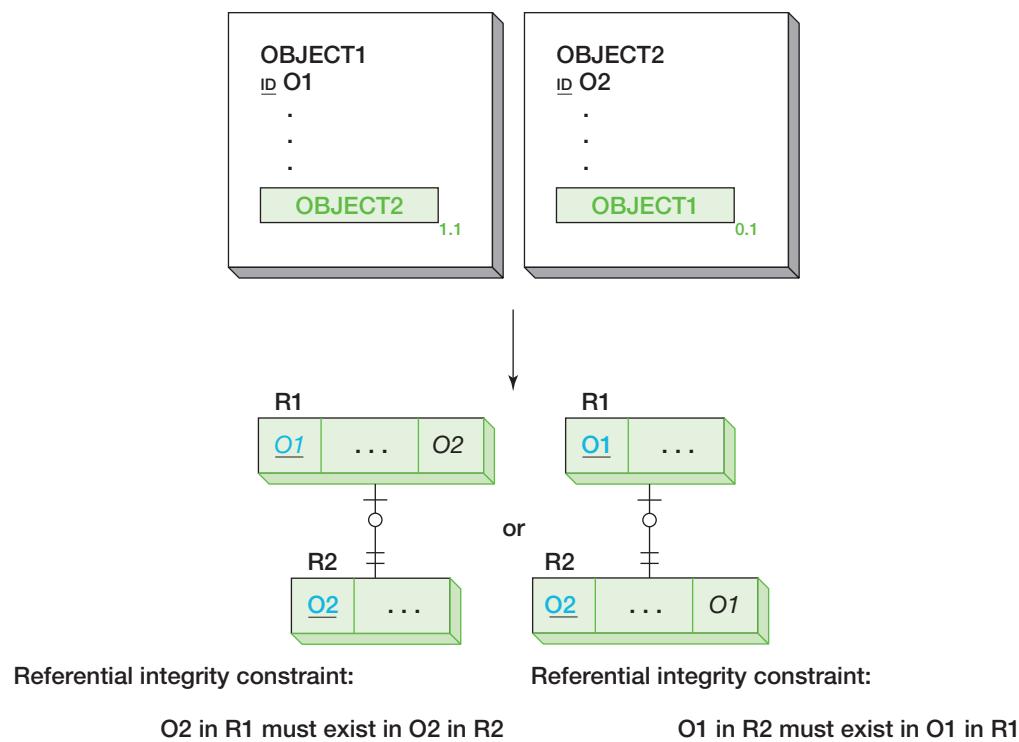
Referential integrity constraint:

LockerNumber in MEMBER must exist in
LockerNumber in LOCKER

(b) Relational Representation

Figure E-14

General Transformation of
1:1 Compound Objects



In general, for a 1:1 relationship between OBJECT1 and OBJECT2 we define one relation for each object, R1 and R2. Then, we place the key of either relation (O1 or O2) as a foreign key in the other relation, as shown in Figure E-14.

Representing One-to-Many and Many-to-One Relationships

Now, consider 1:N relationships and N:1 relationships. Figure E-15(a) shows an example of a 1:N object relationship between EQUIPMENT and REPAIR. An item of EQUIPMENT can have many REPAIRs, but a REPAIR can be related to only one item of EQUIPMENT.

The objects in Figure E-15(a) are represented by the relations in Figure E-15(b). Observe that the key of the parent (the object on the one side of the relationship) is placed in the child (the object on the many side of the relationship).

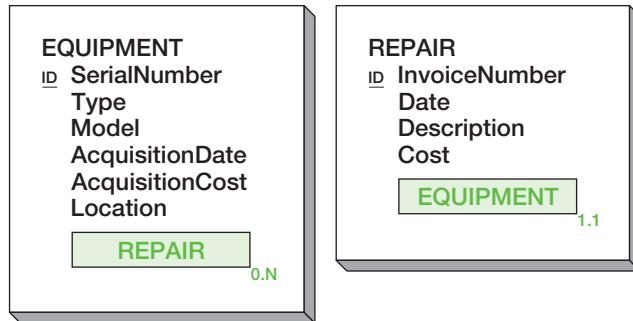
Figure E-16 shows the general transformation of 1:N compound objects. Object OBJECT1 contains many OBJECT2s, and object OBJECT2 contains just one OBJECT1. To represent this structure by means of relations, we represent each object with a relation and place the key of the parent in the child. Thus, in Figure E-16, the attribute O1 is placed in R2.

If OBJECT2 were to contain many OBJECT1s and OBJECT1 were to contain just one OBJECT2, we would use the same strategy, but reverse the role of R1 and R2. That is, we would place O2 in R1.

The minimum cardinalities in either case are determined by the minimum cardinalities of the object attributes. In Figure E-16, OBJECT1 requires at least one OBJECT2, but OBJECT2 does not necessarily require an OBJECT1. These cardinalities are shown in the data structure diagram as an oval on the R1 side of the relationship and as a hash mark on the R2 side of the relationship. These minimum cardinality values are simply examples; either or both objects can have a cardinality of zero, one, or some other number.

Representing Many-to-Many Relationships

Finally, consider M:N relationships. As with M:N entity relationships, we define three relations: one for each of the objects and a third intersection relation. The intersection relation represents



(a) Example 1:N Compound Objects

EQUIPMENT (SerialNumber, Type, Model, AcquisitionDate, AcquisitionCost, Location)

REPAIR (InvoiceNumber, Date, Description, Cost, *SerialNumber*)

Referential integrity constraint:

SerialNumber in REPAIR must exist in
SerialNumber in EQUIPMENT

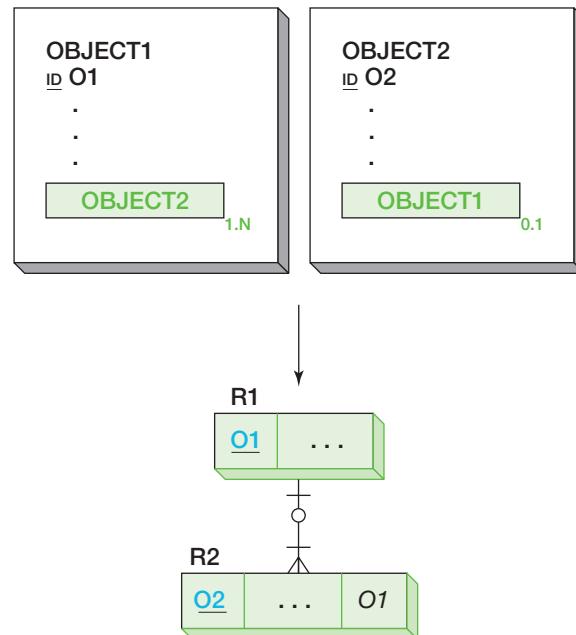
(b) Relational Representation

Figure E-15
Example Relational Representation of 1:N Compound Objects

the relationship of the two objects and consists of the keys of both of its parents. Figure E-17(a) shows the M:N relationship between BOOK and AUTHOR. Figure E-17(b) depicts the three relations that represent these objects: BOOK; AUTHOR; and BOOK-AUTHOR-INT, the intersection relation. Notice that BOOK-AUTHOR-INT has no nonkey data. Both the attributes ISBN and SocialSecurityNumber are underlined and in italics because they are both local and foreign keys.

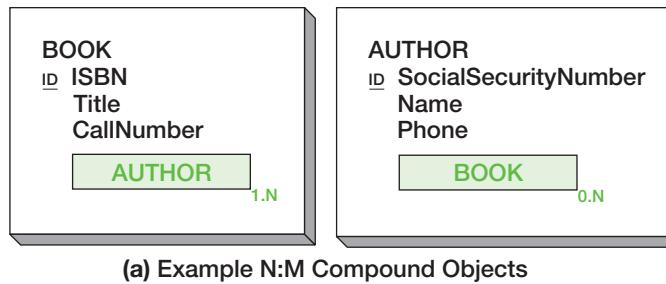
In general, for two objects that have an M:N relationship, we define a relation R1 for object OBJECT1, a relation R2 for object OBJECT2, and a relation R3 for the intersection relation. The general scheme is shown in Figure E-18. Note that the attributes of R3 are only O1 and O2. For M:N compound objects, R3 never contains nonkey data. The importance of this statement will become clear when we contrast M:N compound relationships with association relationships.

Figure E-16
General Transformation of 1:N Compound Objects



Referential integrity constraint:

O1 in R2 must exist in O1 of R1



(a) Example N:M Compound Objects

BOOK (ISBN, Title, CallNumber)AUTHOR (SocialSecurityNumber, Name, Phone)BOOK-AUTHOR-INT (ISBN, SocialSecurityNumber)

Referential integrity constraints:

ISBN in BOOK-AUTHOR-INT must exist in
ISBN in BOOKSocialSecurityNumber in BOOK-AUTHOR-INT must exist in
SocialSecurityNumber in AUTHOR

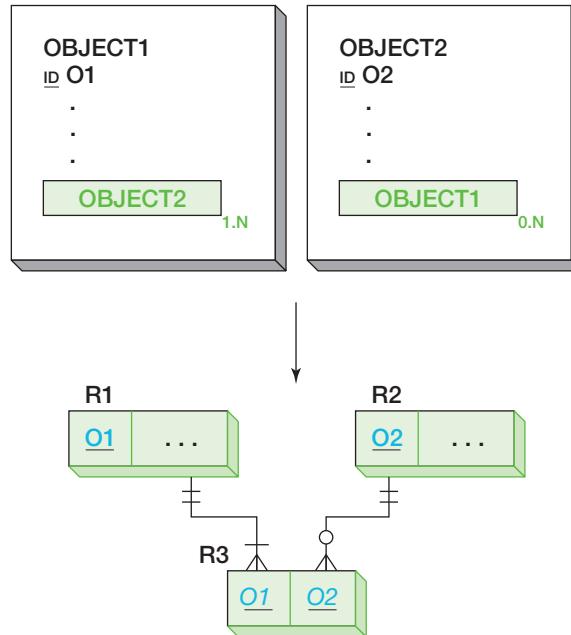
(b) Relational Representation

Figure E-17

Example Relational Representation of N:M Compound Objects

Figure E-18

General Transformation of N:M Compound Objects



Referential integrity constraints:

O1 in R3 must exist in O1 in R1
O2 in R3 must exist in O2 in R2

Hybrid Objects

A **Hybrid object** is a combination of composite and compound objects. In particular, a hybrid object is a semantic object with at least one multivalue group attribute that includes a semantic object attribute.

Figure E-19(a) is a second version of the report about dormitory occupancy shown in Figure E-10(a). The difference is that the third column of the student data contains Rent instead of Class. This is an important difference because Rent is not an attribute of STUDENT, but pertains to the combination of STUDENT and DORMITORY and is an attribute of DORMITORY.

Figure E-1(b) is an object diagram that models this form. DORMITORY contains a multivalue group having the object attribute STUDENT and the nonobject attribute Rent. This means that Rent is paired with STUDENT in the context of DORMITORY.

Now, examine the alternative DORMITORY object shown in Figure E-19(c). This is an *incorrect* model of the report in Figure E-19(a) because it shows that Rent and STUDENT are independently multivalue, which is incorrect, because Rent and STUDENT are multivalue as a pair.

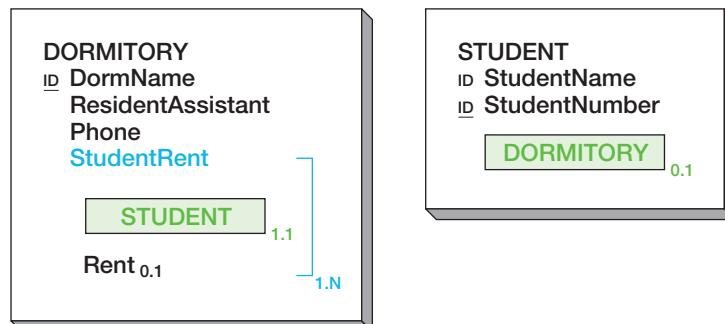
Figure E-20(a) shows a form based on another hybrid object. This Sales Order Form contains data about an order (Sales Order Number, Date, Subtotal, Tax, and Total), data about a CUSTOMER and a SALESPERSON, and a multivalue group that itself contains data about items on the order. Furthermore, ITEM data (Item Number, Description, and Unit Price) appear within the multivalue group.

Figure E-19

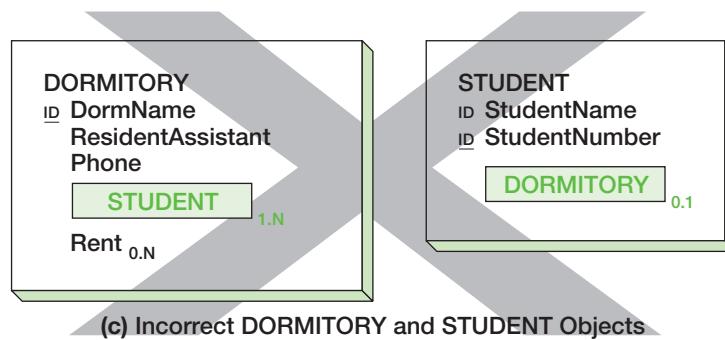
DORMITORY Hybrid Object

DORMITORY OCCUPANCY REPORT		
Dormitory	Resident Assistant	Phone
Ingersoll	Sarah and Allen French	3-5567
<u>Student name</u>	<u>Student Number</u>	<u>Rent</u>
Adams, Elizabeth	710	\$175.00
Baker, Rex	104	\$225.00
Baker, Brydie	744	\$175.00
Charles, Stewart	319	\$135.00
Scott, Sally	447	\$225.00
Taylor, Lynne	810	\$175.00

(a) Dormitory Report with Rent Property



(b) Correct DORMITORY and STUDENT Objects



(c) Incorrect DORMITORY and STUDENT Objects

Carbon River Furniture Sales Order Form

Sales Order Number	10643	Date:	25-Sep-01	
<hr/>				
Customer Name	Carbon River Bookshop			
Address	1145 Elm Street			
City	Carbon River	State	IL	
Phone:	232-0010			
<hr/>				
Salesperson Name	Dodsworth, Anne	Salesperson Code	EZ-1	
<hr/>				
Quantity	Item Number	Description	Unit Price	Extended Price
► 1	78	Executive Desk	\$959.00	\$959.00
► 1	79	Conference Table	\$1,750.00	\$1,750.00
► 4	80	Side Chair	\$99.00	\$396.00
*				
				Subtotal: \$3,105.00
				Tax: \$29.46
				Total: \$3,134.46

(a) Sales Order Form

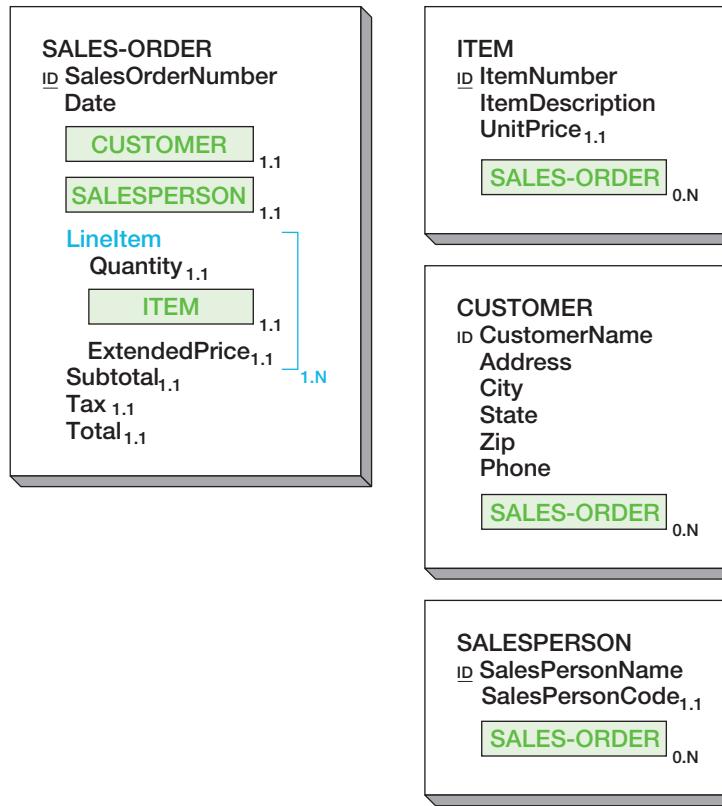


Figure E-20

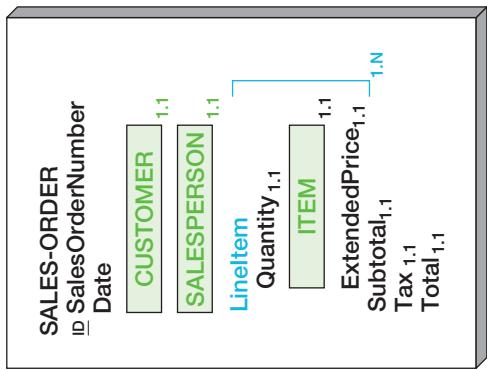
SALES-ORDER Hybrid
Object

(b) Objects to Model Sales Order Form

Figure E-20(b) shows the SALES-ORDER semantic object. It contains the nonobject attributes SalesOrderNumber, Date, Subtotal, Tax, and Total. It also contains the CUSTOMER and SALESPERSON object attributes and a multivalue group that represents each line item on the sales order. The group contains nonobject attributes Quantity and ExtendedPrice and the object attribute ITEM.



(a) ITEM in One ORDER



(b) ITEM in (Possibly) Many LineItems of One ORDER



(c) ITEM in One LineItem of (Possibly) Many ORDERS



(d) ITEM in (Possibly) Many LineItems of (Possibly) Many ORDERS

Figure E-21

Examples of the Four Cases of Maximum Cardinality in a Hybrid Object

The object diagrams shown in Figure E-20(b) are ambiguous in one aspect that may or may not be important, depending on the application. According to the ITEM object diagram, an ITEM can be connected to more than one SALES-ORDER. But because the multivalue group LineItem is encapsulated (hidden within) SALES-ORDER, it is not clear from this diagram whether an ITEM can occur *once or many times* on the same SALES-ORDER.

In general, there are four interpretations of maximum cardinality for the paired attributes in the SALES-ORDER hybrid object:

1. An ITEM can appear on only one SALES-ORDER and in only one of the LineItems within that SALES-ORDER.
2. An ITEM can appear on only one SALES-ORDER, but in many different LineItems within that SALES-ORDER.
3. An ITEM can appear on many different SALES-ORDERS, but in only one LineItem within each of those SALES-ORDERS.
4. An ITEM can appear on many different SALES-ORDERS and in many different LineItems within those SALES-ORDERS.

When it is important to distinguish between these cases, the following notation should be used: If either case 1 or 2 is in force, the maximum cardinality of the hybrid object attribute should be set to one. Thus, for this example, the maximum cardinality of SALES-ORDER in ITEM is set to one. If an ITEM is to appear in only one LineItem of the SALES-ORDER (case 1), it should be marked as having a unique ID in that group. Otherwise (case 2), it need not be marked. These two cases are shown in Figure E-21(a) and E-21(b).

If either case 3 or 4 is in force, the maximum cardinality of the hybrid object attribute is set to N. Thus, for this example, the maximum cardinality of SALES-ORDER in ITEM is set to N. Furthermore, if an ITEM is to appear in only one LineItem of a SALES-ORDER (case 3), it should be marked as having a unique ID in that group. Otherwise (case 4), it need not be marked. These two cases are shown in Figure E-20(c) and E-21(d).

Representing Hybrid Relationships

A general description of these four cases is shown in Figure E-22. Cases 3 and 4 are more common than cases 1 and 2, so we consider them first. OBJECT1 in Figure E-23 shows two groups; Group1 illustrates case 3 and Group2 illustrates case 4.

Group1 has a maximum cardinality of N, which means that there can be many instances of Group1 within an OBJECT1. Furthermore, because OBJECT2 is marked as ID unique, this means that a particular OBJECT2 can appear in only one of the Group1 instances within an OBJECT1. Thus, OBJECT2 acts as an identifier for Group1 within OBJECT1.

Figure E-22
Four Cases of Hybrid Object Cardinality

Case	Description	Example
1	OBJECT2 relates to one instance of OBJECT1 and appears in only one group instance within that object.	ITEM relates to one ORDER and can appear on only one LineItem of that ORDER.
2	OBJECT2 relates to one instance of OBJECT1 and appears in possibly many group instances within that object.	ITEM relates to one ORDER and can appear on many LineItems of that ORDER.
3	OBJECT2 relates to possibly many instances of OBJECT1 and appears in only one group instance within each object.	ITEM relates to many ORDERS and can appear on only one LineItem of that ORDER.
4	OBJECT2 relates to possibly many instances of OBJECT1 and appears in possibly many group instances within those objects.	ITEM relates to many ORDERS and can appear on many LineItems of that ORDER.

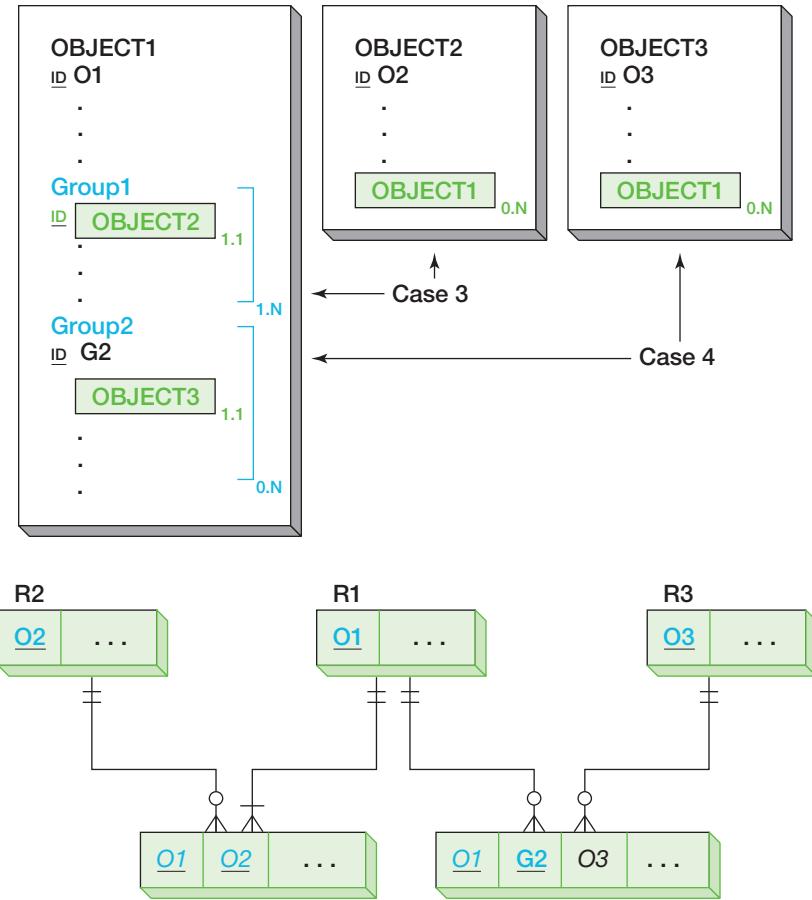


Figure E-23

General Transformations
of Hybrid Objects into
Relations

Referential integrity constraints:

- O1 in R-G1 must exist in O1 in R1
- O2 in R-G1 must exist in O2 in R2
- O1 in R-G2 must exist in O1 in R1
- O3 in R-G2 must exist in O3 of R3

Consider the relational representation of Group1 in Figure E-23. A relation, R1, is created for OBJECT1; and a relation, R2, is created for OBJECT2. In addition, a third relation, R-G1, is created for Group1. The relationship between R1 and R-G1 is 1:N, so we place the key of R1 (which is O1) into R-G1; the relationship between R2 and R-G1 is also 1:N, so we place the key of R2 (which is O2) in R-G1. Because an OBJECT2 can appear with a particular value of OBJECT1 only once, the composite (O1, O2) is unique to R-G1 and can be made the key of that relation.

Now, consider Group2. OBJECT3 does not identify Group2, so OBJECT3 can appear in many Group2 instances in the same OBJECT1. Because OBJECT3 is not the identifier of Group2, we assume that some other attribute, G2, is the identifier.

In Figure E-23, we create a relation R3 for OBJECT3 and another relation R-G2 for Group2. The relationship between R1 and R-G2 is 1:N, so place the key of R1 (which is O1) into R-G2. The relationship between R3 and R-G2 is also 1:N, so place the key of R3 (which is O3) into R-G2.

Unlike Group1, however, (O1, O3) cannot now be the key of R-G2 because an O3 can be paired with a given O1 many times. That is, the composite (O1, O3) is not unique to R-G2, so the key of R-G2 must be (O1, G2).

Case 1 is similar to case 3, except for the restriction that an OBJECT2 can be related to only one OBJECT1. The relations in Figure E-23 still work, but we must add the key of R1 (which is O1) to R2 and establish the restriction that (O1, O2) of R-G1 must equal (O1, O2) of R2.

Case 2 is similar to case 4, except for the restriction that an OBJECT3 can be related to only one OBJECT1. Again, the relations in Figure E-23 will work, but we must add the key of R1 (which is O1) to R3 and establish the restriction that (O1, O3) of R-G2 is a subset of (O1, O3) in R3 (see Review Questions E.21 and E.22).

Association Objects

An **association object** is an object that relates two (or more) objects and stores data that are peculiar to that relationship. Figure E-24(a) shows a report and two data entry screens that give rise to the need for an association object. The report contains data about an airline flight and

Figure E-24

Examples of an Association Object

FLY CHEAP INTERNATIONAL Flight Planning Data Report		
FLIGHT NUMBER	FC-17	DATE 7/30/2008
ORIGINATING CITY	Seattle	DESTINATION Hong Kong
FUEL ON TAKEOFF		
WEIGHT ON TAKEOFF		
AIRPLANE		
Tail Number	N1234FI	
Type	747-SP	
Capacity	148	
PILOT		
Name	Michael Nilson	
FI-ID	32887	
Flight Hours	18,348	

Fly Cheap International Pilot Summary Data Form

FCI_ID	32887	Date Of Last Check Out	7/7/01
Name	Michael Nilson	Hours	18348
SocialSecurityNumber	000-45-0040	Date Of Last Physical	5/19/01
Address	1011 Western		
City	Denver		
State	CO		
Zip	80210		
Phone	555-55-5589		
EmergencyPhone	555-33-9090		

Fly Cheap International Airplane Data Form

TailNumber	N1234FI
Manufacturer	Boeing
Type	747-SP
TotalAirframeHours	11756
TotalEngineHours	11756
EngineHoursSinceOverhaul	756
CurrentCapacity	148
RangeAsConfigured	8900

(a) Example Flight Report and Forms

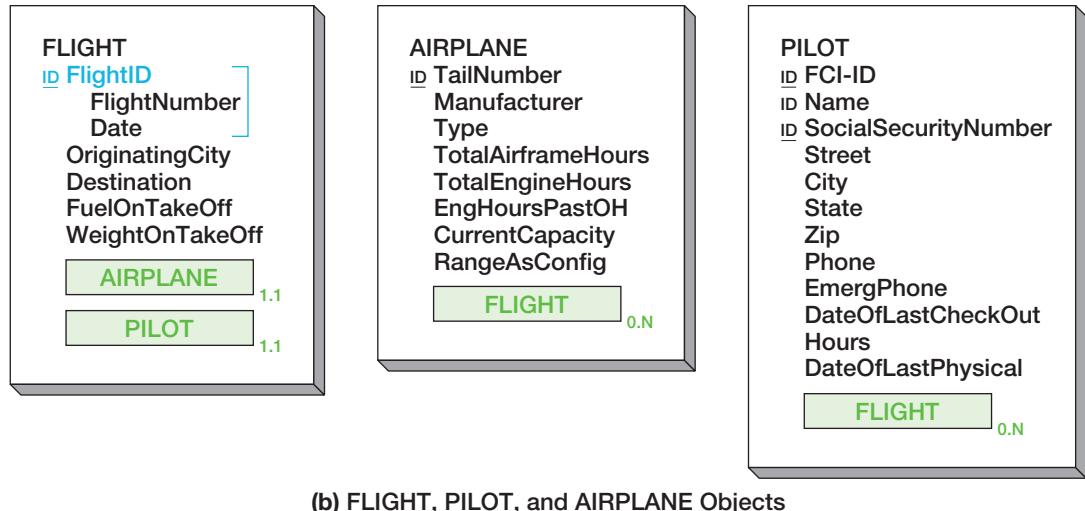


Figure E-24

(Continued)

(b) FLIGHT, PILOT, and AIRPLANE Objects

data about the particular airplane and pilot assigned to that flight. The two data entry forms contain data about a pilot and an airplane.

In Figure E-24(b), the object FLIGHT is an association object that associates the two objects AIRPLANE and PILOT and stores data about their association. FLIGHT contains one each of AIRPLANE and PILOT, but both AIRPLANE and PILOT contain multiple values of FLIGHT. This particular pattern of associating two (or more) objects with data about the association occurs frequently, especially in applications that involve the assignment of two or more things. Other examples are a JOB that assigns an ARCHITECT to a CLIENT, a TASK that assigns an EMPLOYEE to a PROJECT, and a PURCHASE-ORDER that assigns a VENDOR to a SERVICE.

For the example shown in Figure E-24, the association object FLIGHT has an identifier of its own: the group {FlightNumber, Date}. Often, association objects do not have identifiers of their own, in which case the identifier is the combination of the identifiers of the objects that are associated.

To understand this better, consider Figure E-25(a), which shows a report about the assignment of architects to projects. Although the assignment has no obvious identifier, the identifier is the combination {ProjectName, ArchitectName}. These attributes, however, belong to PROJECT and ARCHITECT, not to ASSIGNMENT. The identifier of ASSIGNMENT is thus the combination of those identifiers of the things that are assigned.

Figure E-25(b) shows the object diagrams for this situation. Both PROJECT and ARCHITECT are object attributes of ASSIGNMENT, and the group {PROJECT, ARCHITECT} is the identifier of ASSIGNMENT. This means that the combination of an instance of PROJECT and an instance of ARCHITECT identifies a particular ASSIGNMENT.

Note that the AssignmentID identifier shown in Figure E-25(b) is not unique, thereby indicating that an architect may be assigned to a project more than once. If this is not correct, the identifier should be declared to be unique. Also, if an employee may be assigned to a project more than once, and if for some reason it is important to have a unique identifier for an ASSIGNMENT, the attribute Date or some other time-indicating attribute (Week, Quarter, and so forth) should be added to the group.

In general, when transforming association object structures into relations, we define one relation for each of the objects participating in the relationship. In Figure E-26, OBJECT3 associates OBJECT1 and OBJECT2. In this case, we define R1, R2, and R3, as shown. The key of each of the parent relations, O1 and O2, appears as foreign key attributes in R3, the relation representing the association object. If the association object has no unique identifying attribute, the combination of the attributes of R1 and R2 is used to create a unique identifier.

Note the difference between the association relation shown in Figure E-26 and the intersection relation shown in Figure E-18. The principal distinction is that the association table carries data that represent some aspect of the combination of the objects. The intersection relation carries no data; its only reason for existence is to specify which objects have a relationship with one another.

Project Assignment Report			
Project Name	Abernathy House	Architect Name	Jackson, B.
Project Manager	Smith, J	Phone	232-8878
Project Start	11/11/2008	Office Number	J-1133
Project End			
Assignment Starts	12/15/2008		
Assignment Ends	3/15/2009		
Maximum Budgeted Hours	345		
Maximum Labor Cost	\$27,500		
Maximum Material Cost	\$17,500		

(a) Example Assignment Report

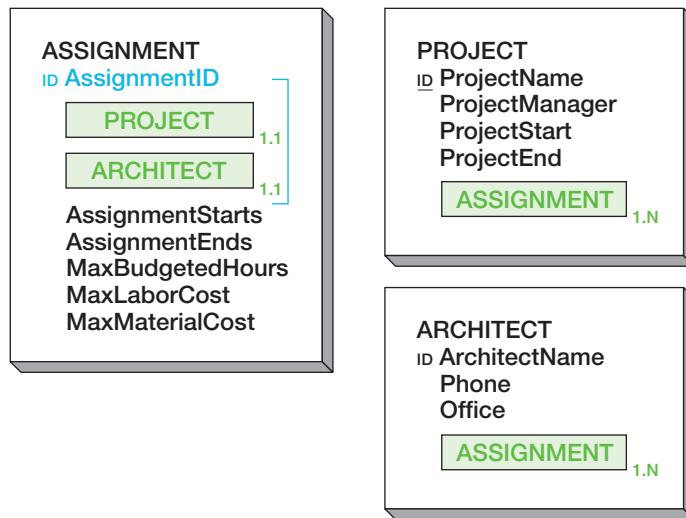


Figure E-25

ASSIGNMENT Association Object

(b) ASSIGNMENT Object with Semantic Object ID

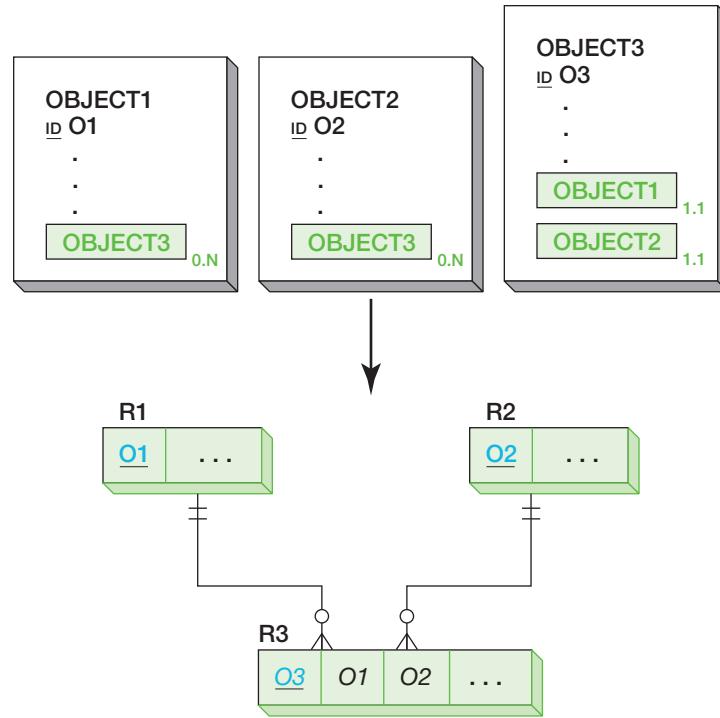
Parent/Subtype Objects

To understand parent and subtype objects, consider the object EMPLOYEE shown in Figure E-27(a). Some of the attributes in EMPLOYEE pertain to all employees, and others pertain only to employees who are managers. The object in Figure E-27(a) is not very precise, because the manager-oriented attributes are not suitable for nonmanager employees.

A better model is shown in Figure E-27(b), in which the EMPLOYEE object contains a subtype object: MANAGER. All of the manager-oriented attributes have been moved to the MANAGER object. Employees who are not managers have one EMPLOYEE object instance and no MANAGER object instances. Employees who are managers have both an EMPLOYEE instance and a MANAGER instance. In this example, the EMPLOYEE object is called a **parent object**, or **supertype object**, whereas the MANAGER object is called a **subtype object**.

The first attribute of a subtype, the parent attribute, is denoted by the subscript *P*. Parent attributes are always required. The identifiers of the subtype are the same as the identifiers of the parent. In Figure E-27(b), EmployeeNumber and EmployeeName are identifiers of both EMPLOYEE and MANAGER.

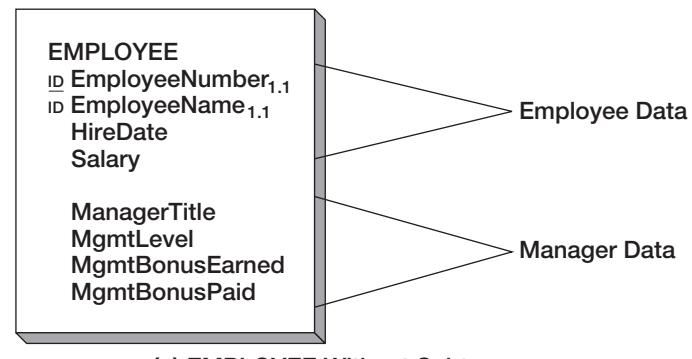
Subtype attributes are shown with the subscript 0.ST or 1.ST. The first digit (0 or 1) is the minimum cardinality of the subtype. If 0, the subtype is optional; if 1, the subtype is required. (A required subtype does not make sense for this example, but it will for the more complicated examples to follow.) The ST indicates that the attribute is a subtype, or an **IS-A attribute**.

**Figure E-26**

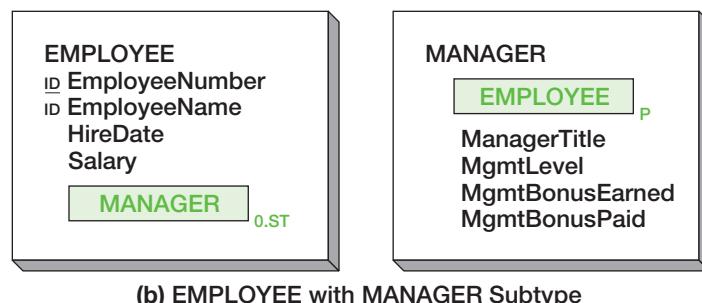
General Transformations of Association Objects into Relations

Referential integrity constraints:

O1 in R3 must exist in O1 in R1
O2 in R3 must exist in O2 in R2

**Figure E-27**

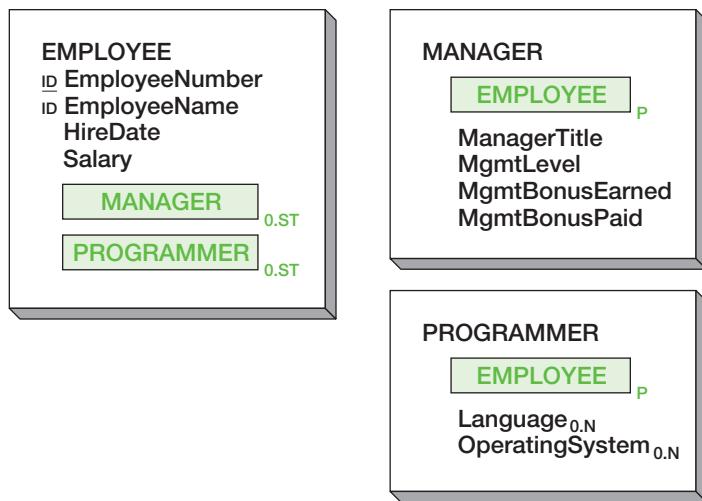
Need for MANAGER Subtype



Parent/subtype objects have an important characteristic called **inheritance**. A subtype acquires, or *inherits*, all of the attributes of its parent; therefore a MANAGER inherits all of the attributes of an EMPLOYEE. In addition, the parent acquires all of the attributes of its subtypes, and an EMPLOYEE who is a MANAGER acquires all of the attributes of MANAGER.

Figure E-28

EMPLOYEE with Two Subtype Properties



A semantic object may contain more than one subtype attribute. Figure E-28 shows a second EMPLOYEE object that has two subtype attributes: MANAGER and PROGRAMMER. Because all of these attributes are optional, an EMPLOYEE can have neither, one, or both of these subtypes. This means that some employees are neither managers nor programmers, some are managers but not programmers, some are programmers but not managers, and some are both programmers and managers.

Subtypes sometimes exclude one another, and are said to be **exclusive subtypes**. That is, a VEHICLE can be an AUTO or a TRUCK, but not both. A CLIENT can be an INDIVIDUAL, a PARTNERSHIP, or a CORPORATION, but only one of these three types. When subtypes exclude one another, they are placed into a subtype group, and the group is assigned a subscript of the format $X.Y.Z$. X is the minimum cardinality and is zero or one, depending on whether the subtype group is required. Y and Z are counts of the number of attributes in the group that are allowed to have a value. Y is the minimum number required, and Z is the maximum number allowed.

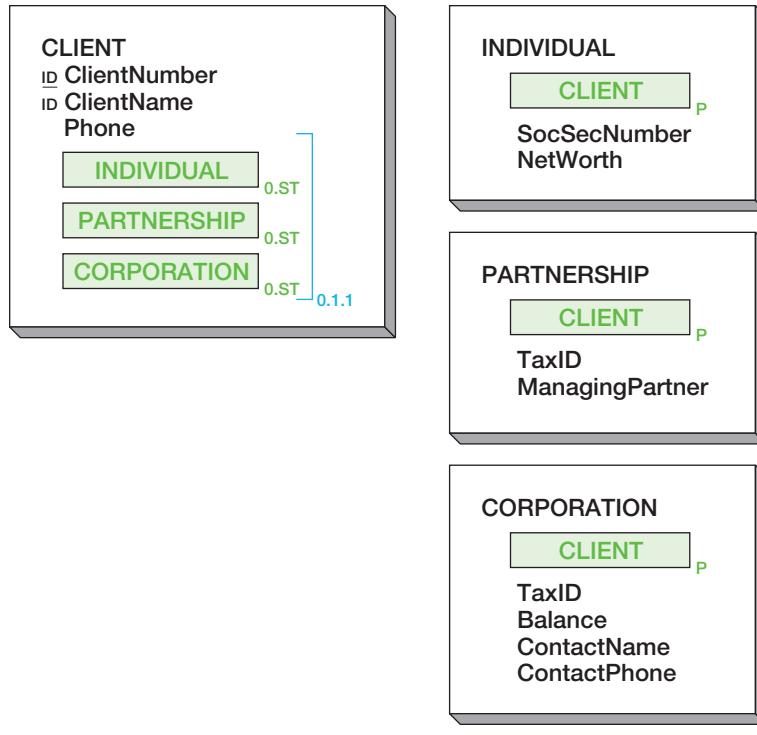
Figure E-29(a) shows three types of CLIENT as a subtype group. The subscript of the group, 0.1.1, means that the subtype is not required; but, if it exists, a minimum of one and a maximum of one (or exactly one) of the subtypes in the group must exist. Note that each of the subtypes has the subscript 0.ST, meaning that they all are optional, as they must be. If they all were required, the maximum count would have to be three, not one. This notation is robust enough to allow for situations in which 3 out of 5 or 7 out of 10 of a list of subtypes must be required.

Even more complex restrictions can be modeled when subtypes are **nested subtypes**. The subtype group in Figure E-29(b) models a situation in which the subtype CORPORATION must be either a TAXABLE-CORP or a NONTAXABLE-CORP. If it is a NONTAXABLE-CORP, it must be either GOV-AGENCY or a SCHOOL. Only a few nonobject attributes are shown in this example. In reality, if such a complex structure were required, there would likely be more attributes.

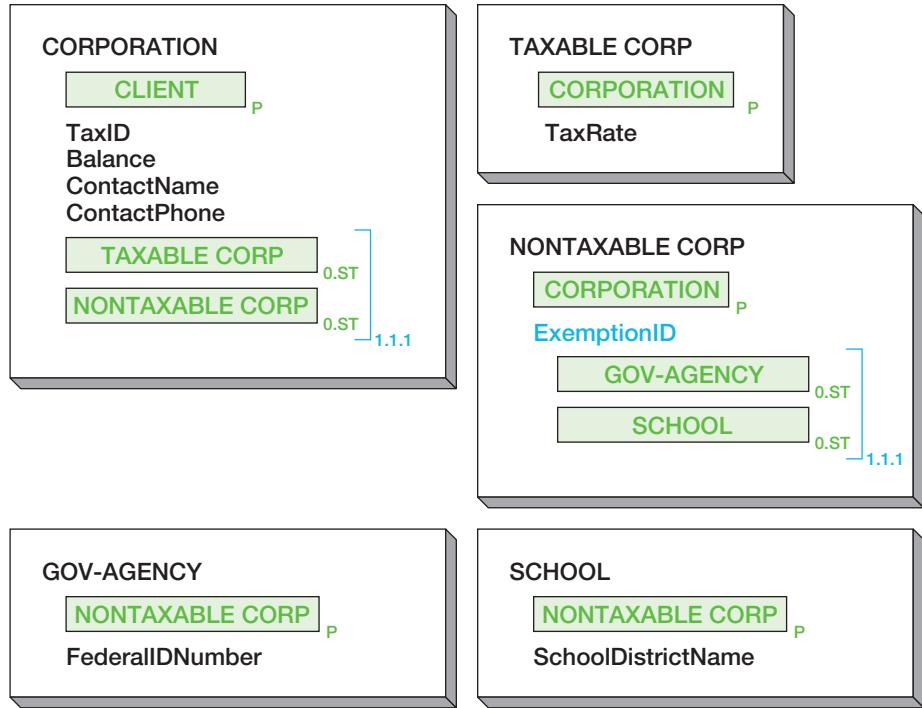
A general scheme for representing subtypes is shown in Figure E-30. One relation is created for the parent and one each for the subtypes. The key of all of the relations is the identifier of the parent. All relationships between the parent and the subtype are 1:1. Note the bar across the relationship lines and the presence of the subtype group's cardinality. The value shown, 0.1.1, means that no subtype is required, but, if present, at most one of the subtypes is allowed.

Archetype/Version Objects

The final type of object is the **archetype/version object**. An archetype object is a semantic object that produces other semantic objects that represent versions, releases, or editions of the



(a) Exclusive Subtypes



(b) Nested Subtypes

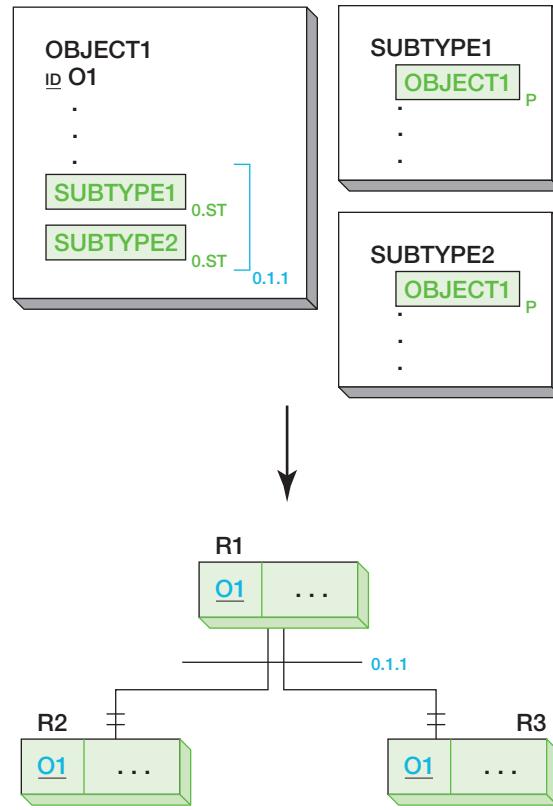
Figure E-29

Exclusive and Nested
Subtypes

archetype. For example, in Figure E-31, the archetype object TEXTBOOK produces the version objects EDITIONS. According to this model, the attributes Title, Author, and Publisher belong to the object TEXTBOOK; the attributes EditionNumber, PublicationDate, and NumberOfPages belong to the EDITION of the TEXTBOOK.

Figure E-30

General Transformation of Parent/Subtypes Objects into Relations

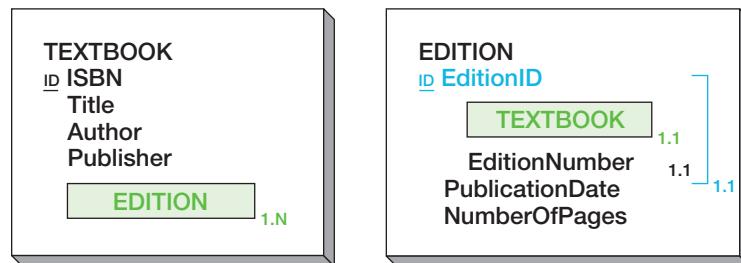


Referential integrity constraints:

O1 in R2 must exist in O1 in R1
O1 in R3 must exist in O1 in R1

Figure E-31

TEXTBOOK/EDITION
Example of an Archetype/Version Object



The ID group in EDITION has two portions: TEXTBOOK and EditionNumber; this is the typical pattern for an ID of a version object. One part of the ID contains the archetype object, and the second part is a simple attribute that identifies the version within the archetype. Figure E-32 shows another instance of archetype/version objects. Figure E-33 shows the general transformation of archetype/version objects. Attribute O1 of R2 is both a local and a foreign key, but O2 is only a local key.

Figure E-32

BUILDING/APARTMENT
Example of an
Archetype/Version Object

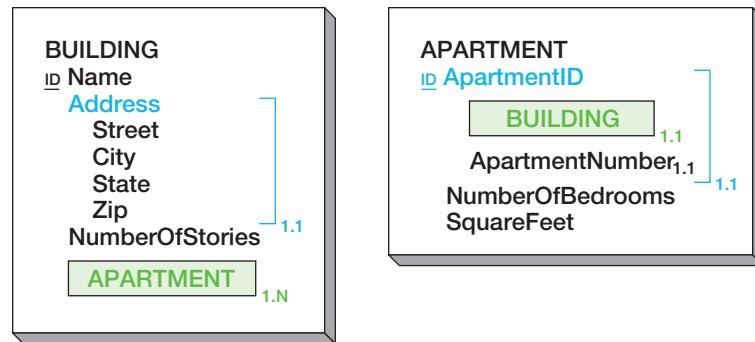
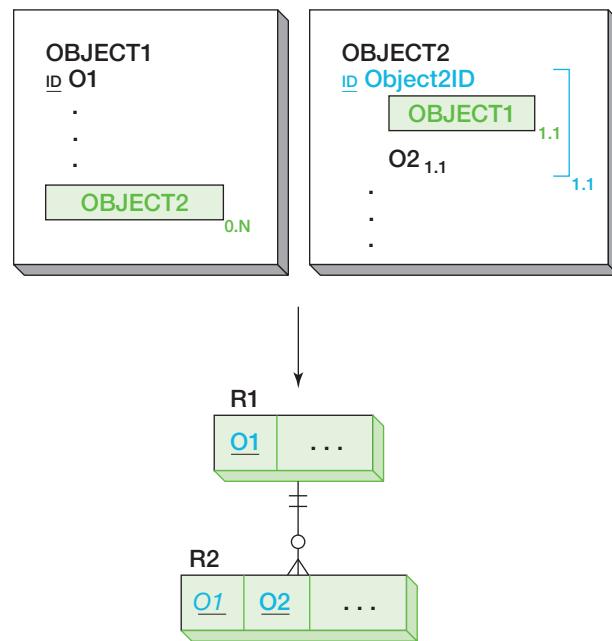


Figure E-33

General Transformation of
Archetype/Version Objects
into Relations



Comparing the Semantic Object and the E-R Models

The E-R model and the semantic object model have both similarities and differences. They are similar in that both are tools for understanding and documenting the structure of the users' data. They both strive to model the structure of the things in the users' world and the relationships among them.

The principal difference between the two models is one of orientation. The E-R model sees the concept of the *entity* as basic. Entities and their relationships are considered the atoms, if you will, of a data model. These atoms can be combined to form what the E-R model calls *user views*, which are combinations of entities whose structures are similar to those of semantic objects.

The semantic object model takes the concept of the *semantic object* as basic. The set of semantic objects in a data model is a map of the essential structure of the things that the user considers to be important. These objects are the atoms of the users' world and are the smallest distinguishable units that the users want to process. They may be decomposed into smaller parts inside the DBMS (or application), but those smaller parts are of no interest or utility to the users.

According to the semantic object perspective, entities, as defined in the E-R model, do not exist. They are only pieces or chunks of the real entities. The only entities that have meaning to users are, in fact, semantic objects. Another way to state this is to say that semantic objects are **semantically self-contained**, or **semantically complete**. Consider an example. Figure E-34 shows four semantic objects: SALES-ORDER, CUSTOMER, SALESPERSON, and ITEM. When a user says, “Show me sales order number 2000,” he or she means show SALES-ORDER, as modeled in Figure E-34. That includes, among other attributes, CUSTOMER data. Because CUSTOMER is part of SALES-ORDER, the SALES-ORDER object includes CUSTOMER.

Figure E-35 is an E-R model of this same data and contains the SALES-ORDER, CUSTOMER, SALESPERSON, LINE-ITEM, and INVENTORY entities. The SALES-ORDER entity includes the attributes OrderNumber, Date, Subtotal, Tax, and Total. Now if a user were to say, “Show me sales order number 2000” and be given only the attributes Date, Subtotal, Tax, and Total, he or she would be disappointed. Most likely, the user’s response would be, “Where’s

Figure E-34
SALES-ORDER and Related Semantic Objects

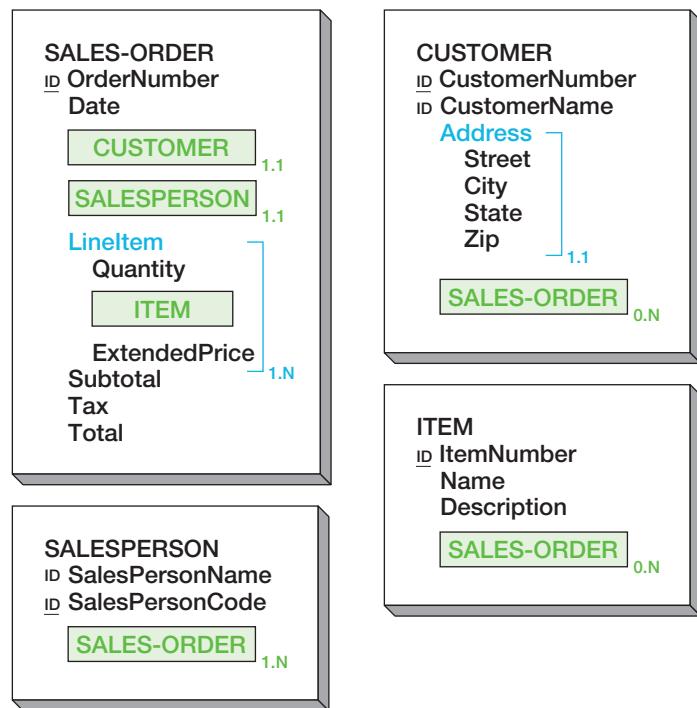
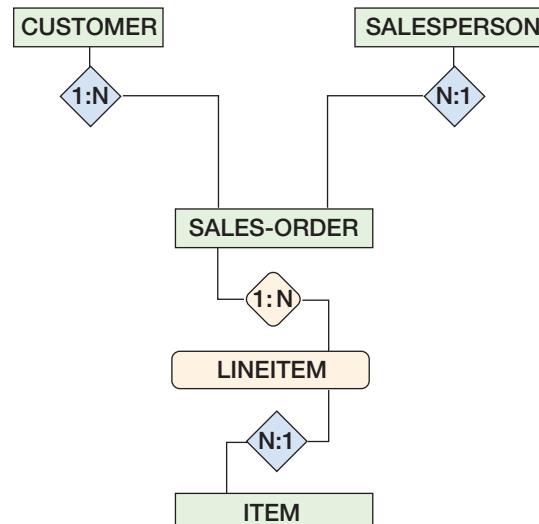


Figure E-35
Entity-Relationship Model of SALES-ORDER and Related Entities



the rest of the data?" That is, the entity SALES-ORDER does not represent the user's meaning of the distinct identity SALES-ORDER. The entity is only a part of SALES-ORDER.

At the same time, when a user (perhaps even the same user) says, "Show me customer 12345," he or she means to show all of the data modeled for CUSTOMER shown in Figure E-34—including CustomerName, all of the attributes of the group Address, and all of the SALES-ORDERS for that CUSTOMER. The entity CUSTOMER shown in Figure E-35 has only the attributes CustomerName, Street, City, State, Zip. If the user were to say, "Show me customer ABC," and be given only this data, he or she again would be disappointed: "No, that's only part of what I want."

According to the semantic object view, E-R entities are unnecessary. Semantic objects can be readily transformed into database designs without ever considering E-R model entities. They are halfway houses, so to speak, constructed in the process of moving away from the paradigm of computer data structures to the paradigm of the user.

Another difference is that the semantic objects contain more metadata than do the entities. In Figure E-34, the semantic object model records the fact that CustomerNumber is a unique identifier in the users' minds. It may or may not be used as an identifier for the underlying table, but that fact is not important to the data model. In addition, CustomerName is a nonunique identifier to the users. Furthermore, the semantic objects represent the fact that there is a semantic group of attributes called *Address*. This group contains other attributes that form the address. The fact that this group exists becomes important when forms and reports are designed. Finally, the semantic objects indicate that an ITEM may relate to more than one SALES-ORDER, but that it can relate to only one LineItem within that SALES-ORDER. This fact cannot be shown on the E-R diagram.

In the final analysis, decide whether Figure E-34 or Figure E-35 gives you a better idea of what the database should contain. Many people find that the boundaries drawn around the semantic objects and the brackets around the group attributes helps them get a better idea of the overall picture of the data model.

Key Terms

archetype/version object	object classes
association object	object diagram
collection of attributes	object identifier
composite object	object links
compound object	object-oriented database processing
distinct identities	paired attributes
domain	parent object
entities	physical existence
enumerated list	semantic
exclusive subtype	semantic object
group attributes	semantic object attributes
group identifier	semantic object diagram
hybrid object	semantic object view
inheritance	semantically complete
instance	semantically self-contained
IS-A attribute	simple attributes
maximum cardinality	simple object
minimum cardinality	single-value attribute
multivalue attribute	subtype object
nested subtype	sufficient description
nonobject attribute	supertype object
object	view
object class name	



Review Questions

- E.1** Explain why the E-R model and the semantic object model are like lenses.
- E.2** Define *semantic object*.
- E.3** Explain the difference between an object class name and an object instance name. Give an example of each.
- E.4** What is required for a set of attributes to be a sufficient description?
- E.5** Explain the phrase *distinct identity* as it pertains to the definition of a semantic object.
- E.6** Explain why a line item of an order is not a semantic object.
- E.7** List the three types of attributes.
- E.8** Give an example of each of the following:
 - a. A simple, single-value attribute
 - b. A group, single-value attribute
 - c. A simple, multivalue attribute
 - d. A group, multivalue attribute
 - e. A simple object attribute
 - f. A multivalue object attribute
- E.9** What is minimum cardinality? How is it used? Which types of attributes have minimum cardinality?
- E.10** What is maximum cardinality? How is it used? Which types of attributes have maximum cardinality?
- E.11** What are paired attributes? Why are they needed?
- E.12** What is an object identifier? Give an example of a simple attribute object identifier and an example of a group attribute object identifier.
- E.13** Define attribute domain. What are the types of attribute domains? Why is a semantic description necessary?
- E.14** What is a semantic object view? Give an example of an object and two views other than those in this text.
- E.15** Give an example of a simple object, other than one in this text. Show how to represent this object by means of a relation.
- E.16** Give an example of a composite object, other than one in this text. Show how to represent this object by means of relations.
- E.17** Give an example of a 1:1 compound object, other than one in this text. Show two ways to represent it by means of relations.
- E.18** Give an example of a 1:N compound object, other than one in this text. Show how to represent it by means of relations.
- E.19** Give an example of an M:1 compound object, other than one in this text. Show how to represent it by means of relations.
- E.20** Give an example of an M:N compound object, other than one in this text. Show how to represent it by means of relations.
- E.21** Give an example of a case 1 (see Figure E-22) hybrid object. Show how to represent it by means of relations.

- E.22** Give an example of a case 2 (see Figure E-22) hybrid object. Show how to represent it by means of relations.
- E.23** Give an example of an association and related objects, other than one in this text. Show how to represent these objects by means of relations. Assume that the association object has an identifier of its own.
- E.24** Do the same as for Review Question E.23, but assume that the association object does not have an identifier of its own.
- E.25** Give an example of a parent object with at least two exclusive subtypes. Show how to represent these objects by means of relations. Use a type indicator attribute.
- E.26** Give an example of a parent object with at least two nonexclusive subtypes. Show how to represent these objects by means of relations. Use a type indicator attribute.
- E.27** Find an example of a form on your campus that would be appropriately modeled with a simple object. Show how to represent this object by means of a relation.
- E.28** Find an example of a form on your campus that would be appropriately modeled with a composite object. Show how to represent this object by means of relations.
- E.29** Find an example of a form on your campus that would be appropriately modeled with one of the types of a compound object. Show how to represent these objects by means of relations.
- E.30** Find an example of a form on your campus that would be appropriately modeled with a hybrid object. Classify the object according to Figure E-22. Show how to represent these objects by means of relations.
- E.31** Find an example of a form on your campus that would be appropriately modeled with an association and related objects. Show how to represent these objects by means of relations.
- E.32** Find an example of a form on your campus that would be appropriately modeled with parent/subtype objects. Show how to represent these objects by means of relations.
- E.33** Find an example of a form on your campus that would be appropriately modeled with archetype/version objects. Show how to represent these objects by means of relations.
- E.34** Explain the similarities between the E-R model and the semantic object model.
- E.35** Explain the major differences between the E-R model and the semantic object model.
- E.36** Explain the reasoning that entities, as defined in the E-R model, do not truly exist.
- E.37** Show how both the E-R model and the semantic object model would represent the data underlying the SALES-ORDER form shown in Figure E-20(a), and explain the main differences.