CHAPTER 10

Database System Development Lifecycle

Chapter Objectives

In this chapter you will learn:

- The main components of an information system.
- The main stages of the database system development lifecycle (DSDLC).
- The main phases of database design: conceptual, logical, and physical design.
- The types of criteria used to evaluate a DBMS.
- How to evaluate and select a DBMS.
- The benefits of Computer-Aided Software Engineering (CASE) tools.

Software has now surpassed hardware as the key to the success of many computer-based systems. Unfortunately, the track record for developing software is not particularly impressive. The last few decades have seen the proliferation of software applications ranging from small, relatively simple applications consisting of a few lines of code to large, complex applications consisting of millions of lines of code. Many of these applications have required constant maintenance. This maintenance involved correcting faults that had been detected, implementing new user requirements, and modifying the software to run on new or upgraded platforms. The effort spent on maintenance began to absorb resources at an alarming rate. As a result, many major software projects were late, over budget, unreliable, and difficult to maintain, and performed poorly. These issues led to what has become known as the **software crisis**. Although this term was first used in the late 1960s, the crisis is still with us. As a result, some authors now refer to the software crisis as the **software depression**. As an indication of the crisis, a study carried out in the UK by OASIG, a Special Interest Group concerned with the Organizational Aspects of IT, reached the following conclusions about software projects (OASIG, 1996):

- 80–90% do not meet their performance goals;
- about 80% are delivered late and over budget;
- around 40% fail or are abandoned;
- under 40% fully address training and skills requirements;

- less than 25% properly integrate enterprise and technology objectives;
- just 10–20% meet all their success criteria.

There are several major reasons for the failure of software projects, including:

- lack of a complete requirements specification;
- lack of an appropriate development methodology;
- poor decomposition of design into manageable components.

As a solution to these problems, a structured approach to the development of software was proposed called the **Information Systems Lifecycle** (**ISLC**) or the **Software Development Lifecycle** (**SDLC**). However, when the software being developed is a database system the lifecycle is more specifically referred to as the **Database System Development Lifecycle** (**DSDLC**).

Structure of this Chapter In Section 10.1 we briefly describe the information systems lifecycle and discuss how this lifecycle relates to the database system development lifecycle. In Section 10.2 we present an overview of the stages of the database system development lifecycle. In Sections 10.3 to 10.13 we describe each stage of the lifecycle in more detail. In Section 10.14 we discuss how Computer-Aided Software Engineering (CASE) tools can provide support for the database system development lifecycle.

10.1 The Information Systems Lifecycle

Information system The resources that enable the collection, management, control, and dissemination of information throughout an organization.

Since the 1970s, database systems have been gradually replacing file-based systems as part of an organization's Information Systems (IS) infrastructure. At the same time, there has been a growing recognition that data is an important corporate resource that should be treated with respect, like all other organizational resources. This resulted in many organizations establishing whole departments or functional areas called Data Administration (DA) and Database Administration (DBA) that are responsible for the management and control of the corporate data and the corporate database, respectively.

A computer-based information system includes a database, database software, application software, computer hardware, and personnel using and developing the system.

The database is a fundamental component of an information system, and its development and usage should be viewed from the perspective of the wider requirements of the organization. Therefore, the lifecycle of an organization's information system is inherently linked to the lifecycle of the database system that supports it. Typically, the stages in the lifecycle of an information system include: planning, requirements collection and analysis, design, prototyping, implementation, testing, conversion, and operational maintenance. In this chapter we review these stages

from the perspective of developing a database system. However, it is important to note that the development of a database system should also be viewed from the broader perspective of developing a component part of the larger organization-wide information system.

Throughout this chapter we use the terms "functional area" and "application area" to refer to particular enterprise activities within an organization such as marketing, personnel, and stock control.

10.2 The Database System Development Lifecycle

As a database system is a fundamental component of the larger organization-wide information system, the database system development lifecycle is inherently associated with the lifecycle of the information system. The stages of the database system development lifecycle are shown in Figure 10.1. Below the name of each stage is the section in this chapter that describes that stage.

It is important to recognize that the stages of the database system development lifecycle are not strictly sequential, but involve some amount of repetition of previous stages through *feedback loops*. For example, problems encountered during database design may necessitate additional requirements collection and analysis. As there are feedback loops between most stages, we show only some of the more obvious ones in Figure 10.1. A summary of the main activities associated with each stage of the database system development lifecycle is described in Table 10.1.

For small database systems, with a small number of users, the lifecycle need not be very complex. However, when designing a medium to large database systems with tens to thousands of users, using hundreds of queries and application programs, the lifecycle can become extremely complex. Throughout this chapter, we concentrate on activities associated with the development of medium to large database systems. In the following sections we describe the main activities associated with each stage of the database system development lifecycle in more detail.

10.3 Database Planning

Database planning

The management activities that allow the stages of the database system development lifecycle to be realized as efficiently and effectively as possible.

Database planning must be integrated with the overall IS strategy of the organization. There are three main issues involved in formulating an IS strategy, which are:

- identification of enterprise plans and goals with subsequent determination of information systems needs;
- evaluation of current information systems to determine existing strengths and weaknesses;
- appraisal of IT opportunities that might yield competitive advantage.

The methodologies used to resolve these issues are outside the scope of this book; however, the interested reader is referred to Robson (1997) and Cadle & Yeates (2007) for a fuller discussion.

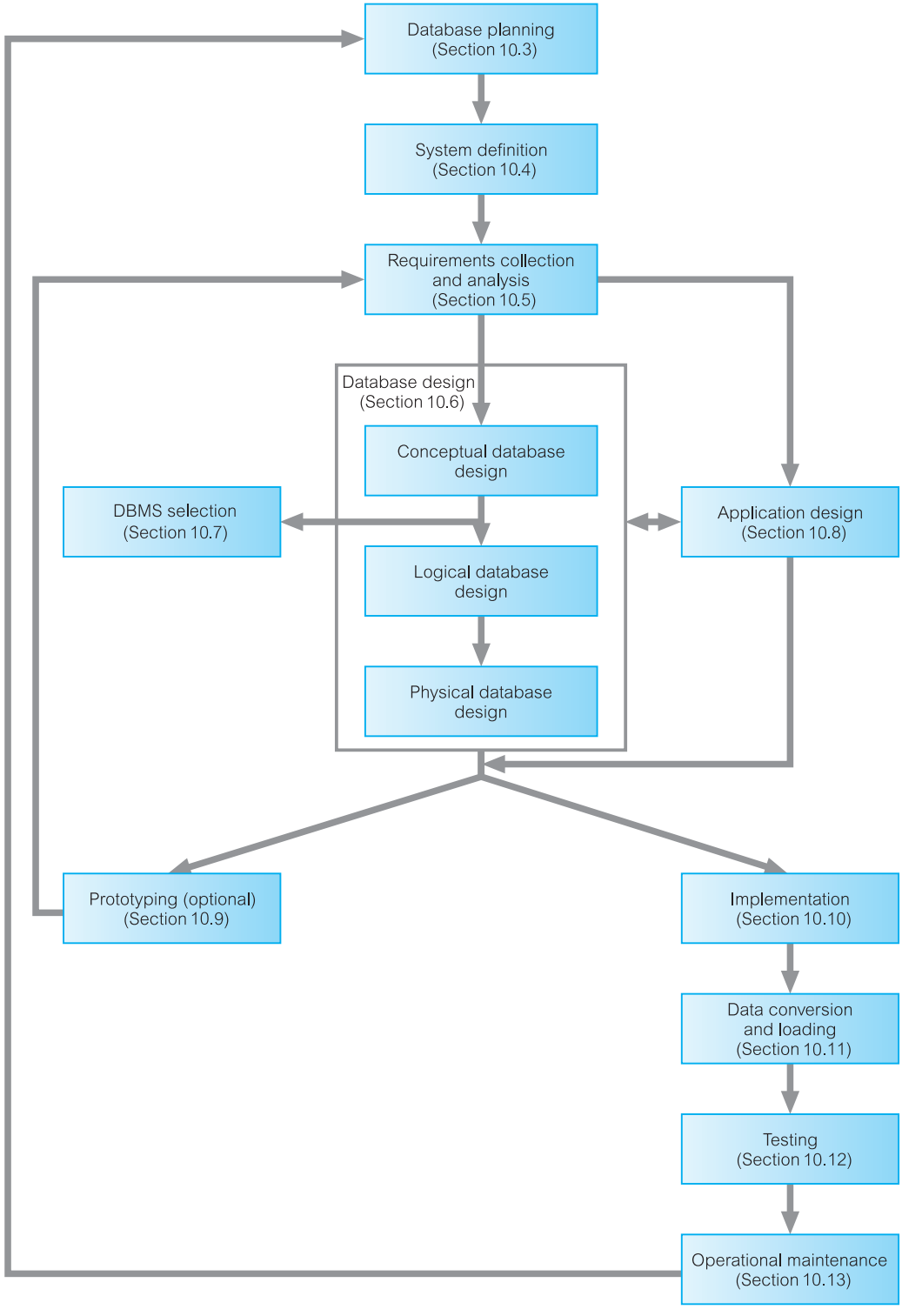


Figure 10.1 The stages of the database system development lifecycle.

TABLE 10.1 Summary of the main activities associated with each stage of the database system development lifecycle.

STAGE	MAIN ACTIVITIES
Database planning	Planning how the stages of the lifecycle can be realized most efficiently and effectively.
System definition	Specifying the scope and boundaries of the database system, including the major user views, its users, and application areas.
Requirements collection and analysis	Collection and analysis of the requirements for the new database system.
Database design	Conceptual, logical, and physical design of the database.
DBMS selection	Selecting a suitable DBMS for the database system.
Application design	Designing the user interface and the application programs that use and process the database.
Prototyping (optional)	Building a working model of the database system, which allows the designers or users to visualize and evaluate how the final system will look and function.
Implementation	Creating the physical database definitions and the application programs.
Data conversion and loading	Loading data from the old system to the new system and, where possible, converting any existing applications to run on the new database.
Testing	Database system is tested for errors and validated against the requirements specified by the users.
Operational maintenance	Database system is fully implemented. The system is continuously monitored and maintained. When necessary, new requirements are incorporated into the database system through the preceding stages of the lifecycle.

An important first step in database planning is to clearly define the **mission statement** for the database system. The mission statement defines the major aims of the database system. Those driving the database project within the organization (such as the Director and/or owner) normally define the mission statement. A mission statement helps to clarify the purpose of the database system and provide a clearer path towards the efficient and effective creation of the required database system. Once the mission statement is defined, the next activity involves identifying the **mission objectives**. Each mission objective should identify a particular task that the database system must support. The assumption is that if the database system supports the mission objectives, then the mission statement should be met. The mission statement and objectives may be accompanied by some additional information that specifies in general terms the work to be done, the resources with which to do it, and the money to pay for it all. We demonstrate the creation of a mission statement and mission objectives for the database system of *DreamHome* in Section 11.4.2.

Database planning should also include the development of standards that govern how data will be collected, how the format should be specified, what documentation will be needed, and how design and implementation should proceed. Standards can be very time-consuming to develop and maintain, requiring resources to set them up initially, and to continue maintaining them. However, a well-designed set of standards provides a basis for training staff and measuring

quality control, and can ensure that work conforms to a pattern, irrespective of staff skills and experience. For example, specific rules may govern how data items can be named in the data dictionary, which in turn may prevent both redundancy and inconsistency. Any legal or enterprise requirements concerning the data should be documented, such as the stipulation that some types of data must be treated confidentially.

10.4 System Definition

System definition Describes the scope and boundaries of the database system and the major user views.

Before attempting to design a database system, it is essential that we first identify the boundaries of the system that we are investigating and how it interfaces with other parts of the organization's information system. It is important that we include within our system boundaries not only the current users and application areas, but also future users and applications. We present a diagram that represents the scope and boundaries of the *DreamHome* database system in Figure 11.10. Included within the scope and boundary of the database system are the major user views that are to be supported by the database.

10.4.1 User Views

User view Defines what is required of a database system from the perspective of a particular job role (such as Manager or Supervisor) or enterprise application area (such as marketing, personnel, or stock control).

A database system may have one or more user views. Identifying user views is an important aspect of developing a database system because it helps to ensure that no major users of the database are forgotten when developing the requirements for the new database system. User views are also particularly helpful in the development of a relatively complex database system by allowing the requirements to be broken down into manageable pieces.

A user view defines what is required of a database system in terms of the data to be held and the transactions to be performed on the data (in other words, what the users will do with the data). The requirements of a user view may be distinct to that view or overlap with other views. Figure 10.2 is a diagrammatic representation of a database system with multiple user views (denoted user view 1 to 6). Note that whereas user views (1, 2, and 3) and (5 and 6) have overlapping requirements (shown as hatched areas), user view 4 has distinct requirements.

10.5 Requirements Collection and Analysis

Requirements collection and analysis The process of collecting and analyzing information about the part of the organization that is to be supported by the database system, and using this information to identify the requirements for the new system.

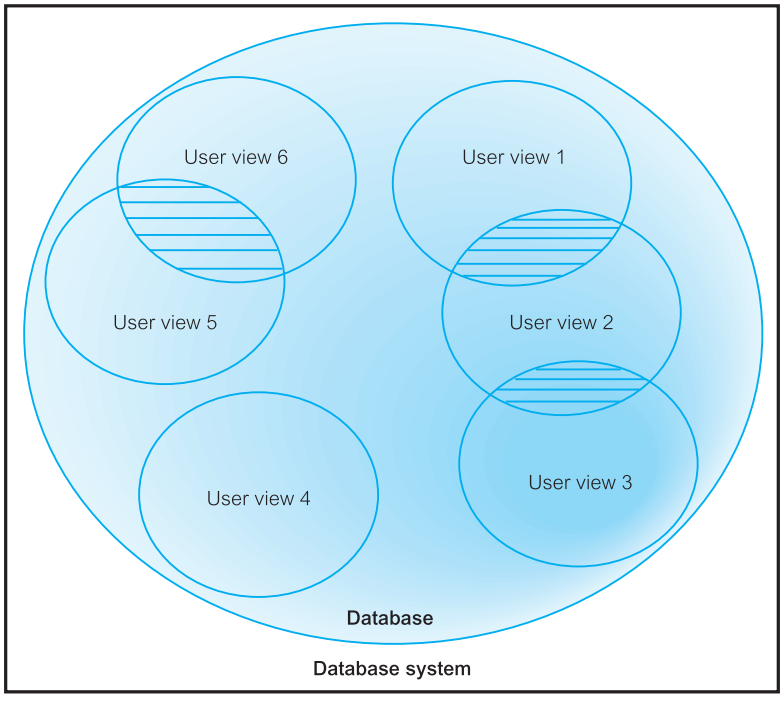


Figure 10.2 Representation of a database system with multiple user views: user views (1, 2, and 3) and (5 and 6) have overlapping requirements (shown as hatched areas), whereas user view 4 has distinct requirements.

This stage involves the collection and analysis of information about the part of the enterprise to be served by the database. There are many techniques for gathering this information, called **fact-finding techniques**, which we discuss in detail in Chapter 11. Information is gathered for each major user view (that is, job role or enterprise application area), including:

- a description of the data used or generated;
- the details of how data is to be used or generated;
- any additional requirements for the new database system.

This information is then analyzed to identify the requirements (or features) to be included in the new database system. These requirements are described in documents collectively referred to as **requirements specifications** for the new database system.

Requirements collection and analysis is a preliminary stage to database design. The amount of data gathered depends on the nature of the problem and the policies of the enterprise. Too much study too soon leads to *paralysis by analysis*. Too little thought can result in an unnecessary waste of both time and money due to working on the wrong solution to the wrong problem.

The information collected at this stage may be poorly structured and include some informal requests, which must be converted into a more structured statement of

requirements. This is achieved using **requirements specification techniques**, which include, for example, Structured Analysis and Design (SAD) techniques, Data Flow Diagrams (DFD), and Hierarchical Input Process Output (HIPO) charts supported by documentation. As you will see shortly, Computer-Aided Software Engineering (CASE) tools may provide automated assistance to ensure that the requirements are complete and consistent. In Section 27.8 we will discuss how the Unified Modeling Language (UML) supports requirements analysis and design.

Identifying the required functionality for a database system is a critical activity, as systems with inadequate or incomplete functionality will annoy the users, which may lead to rejection or underutilization of the system. However, excessive functionality can also be problematic, as it can overcomplicate a system, making it difficult to implement, maintain, use, or learn.

Another important activity associated with this stage is deciding how to deal with the situation in which there is more than one user view for the database system. There are three main approaches to managing the requirements of a database system with multiple user views:

- the **centralized** approach;
- the **view integration** approach;
- a combination of both approaches.

10.5.1 Centralized Approach

Centralized approach

Requirements for each user view are merged into a single set of requirements for the new database system. A data model representing all user views is created during the database design stage.

The centralized (or one-shot) approach involves collating the requirements for different user views into a single list of requirements. The collection of user views is given a name that provides some indication of the application area covered by all the merged user views. In the database design stage (see Section 10.6), a global data model is created, which represents all user views. The global data model is composed of diagrams and documentation that formally describe the data requirements of the users. A diagram representing the management of user views 1 to 3 using the centralized approach is shown in Figure 10.3. Generally, this approach is preferred when there is a significant overlap in requirements for each user view and the database system is not overly complex.

10.5.2 View Integration Approach

View integration approach

Requirements for each user view remain as separate lists. Data models representing each user view are created and then merged later during the database design stage.

The view integration approach involves leaving the requirements for each user view as separate lists of requirements. In the database design stage (see Section 10.6), we

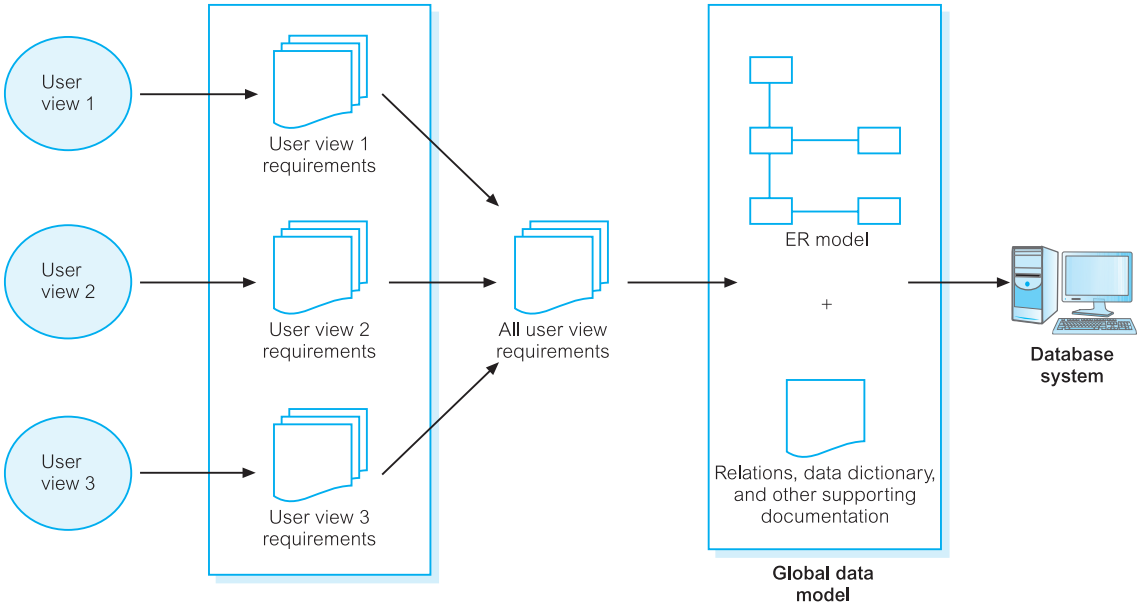


Figure 10.3 The centralized approach to managing multiple user views 1 to 3.

first create a data model for each user view. A data model that represents a single user view (or a subset of all user views) is called a **local data model**. Each model is composed of diagrams and documentation that formally describes the requirements of one or more—but not all—user views of the database. The local data models are then merged at a later stage of database design to produce a **global data model**, which represents *all* user requirements for the database. A diagram representing the management of user views 1 to 3 using the view integration approach is shown in Figure 10.4. Generally, this approach is preferred when there are significant differences between user views and the database system is sufficiently complex to justify dividing the work into more manageable parts. We demonstrate how to use the view integration approach in Chapter 17, Step 2.6.

For some complex database systems, it may be appropriate to use a combination of both the centralized and view integration approaches to manage multiple user views. For example, the requirements for two or more user views may be first merged using the centralized approach, which is used to build a local logical data model. This model can then be merged with other local logical data models using the view integration approach to produce a global logical data model. In this case, each local logical data model represents the requirements of two or more user views and the final global logical data model represents the requirements of all user views of the database system.

We discuss how to manage multiple user views in more detail in Section 11.4.4, and using the methodology described in this book, we demonstrate how to build a database for the *DreamHome* property rental case study using a combination of the centralized and view integration approaches.

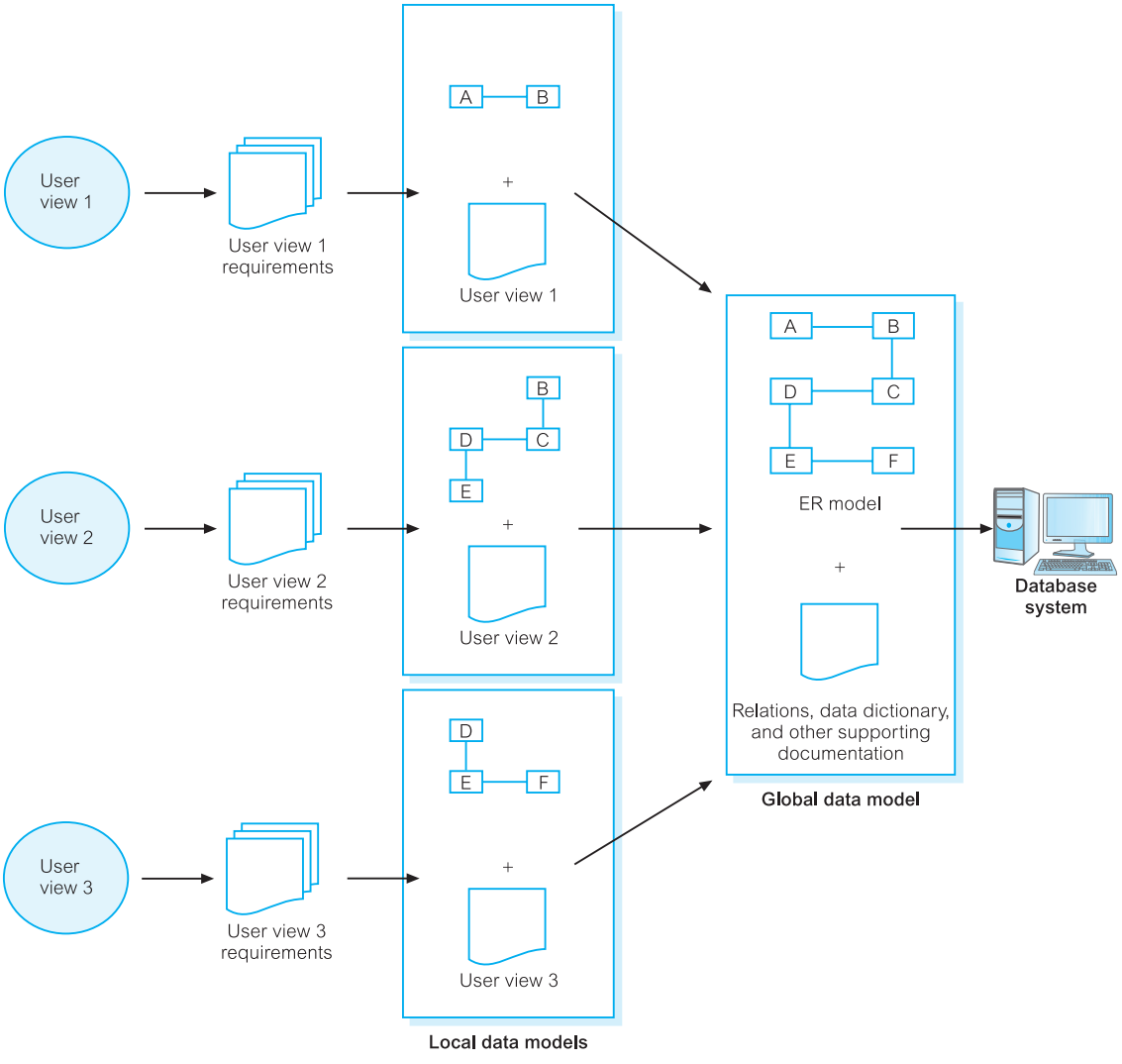


Figure 10.4 The view integration approach to managing multiple user views 1 to 3.

10.6 Database Design

Database design The process of creating a design that will support the enterprise's mission statement and mission objectives for the required database system.

In this section we present an overview of the main approaches to database design. We also discuss the purpose and use of data modeling in database design. We then describe the three phases of database design: conceptual, logical, and physical design.

10.6.1 Approaches to Database Design

The two main approaches to the design of a database are referred to as "bottom-up" and "top-down." The **bottom-up** approach begins at the fundamental level of attributes (that is, properties of entities and relationships), which through analysis of the associations between attributes are grouped into relations that represent types of entities and relationships between entities. In Chapters 14 and 15 we discuss the process of normalization, which represents a bottom-up approach to database design. Normalization involves the identification of the required attributes and their subsequent aggregation into normalized relations based on functional dependencies between the attributes.

The bottom-up approach is appropriate for the design of simple databases with a relatively small number of attributes. However, this approach becomes difficult when applied to the design of more complex databases with a larger number of attributes, where it is difficult to establish all the functional dependencies between the attributes. As the conceptual and logical data models for complex databases may contain hundreds to thousands of attributes, it is essential to establish an approach that will simplify the design process. Also, in the initial stages of establishing the data requirements for a complex database, it may be difficult to establish all the attributes to be included in the data models.

A more appropriate strategy for the design of complex databases is to use the **top-down** approach. This approach starts with the development of data models that contain a few high-level entities and relationships and then applies successive top-down refinements to identify lower-level entities, relationships, and the associated attributes. The top-down approach is illustrated using the concepts of the Entity-Relationship (ER) model, beginning with the identification of entities and relationships between the entities, which are of interest to the organization. For example, we may begin by identifying the entities PrivateOwner and PropertyForRent, and then the relationship between these entities, PrivateOwner *Owns* PropertyForRent, and finally the associated attributes such as PrivateOwner (ownerNo, name, and address) and PropertyForRent (propertyNo and address). Building a high-level data model using the concepts of the ER model is discussed in Chapters 12 and 13.

There are other approaches to database design, such as the inside-out approach and the mixed strategy approach. The **inside-out** approach is related to the bottom-up approach, but differs by first identifying a set of major entities and then spreading out to consider other entities, relationships, and attributes associated with those first identified. The **mixed strategy** approach uses both the bottom-up and top-down approach for various parts of the model before finally combining all parts together.

10.6.2 Data Modeling

The two main purposes of data modeling are to assist in the understanding of the meaning (semantics) of the data and to facilitate communication about the information requirements. Building a data model requires answering questions about entities, relationships, and attributes. In doing so, the designers discover the semantics of the enterprise's data, which exist whether or not they happen to be recorded in a formal data model. Entities, relationships, and attributes are fundamental to

all enterprises. However, their meaning may remain poorly understood until they have been correctly documented. A data model makes it easier to understand the meaning of the data, and thus we model data to ensure that we understand:

- each user's perspective of the data;
- the nature of the data itself, independent of its physical representations;
- the use of data across user views.

Data models can be used to convey the designer's understanding of the information requirements of the enterprise. Provided both parties are familiar with the notation used in the model, it will support communication between the users and designers. Increasingly, enterprises are standardizing the way that they model data by selecting a particular approach to data modeling and using it throughout their database development projects. The most popular high-level data model used in database design, and the one we use in this book, is based on the concepts of the ER model. We describe ER modeling in detail in Chapters 12 and 13.

Criteria for data models

An *optimal* data model should satisfy the criteria listed in Table 10.2 (Fleming and Von Halle, 1989). However, sometimes these criteria are incompatible with each other and trade-offs are sometimes necessary. For example, in attempting to achieve greater *expressibility* in a data model, we may lose *simplicity*.

10.6.3 Phases of Database Design

Database design is made up of three main phases: conceptual, logical, and physical design.

Conceptual database design

Conceptual database design — The process of constructing a model of the data used in an enterprise, independent of *all* physical considerations.

TABLE 10.2 The criteria to produce an optimal data model.

Structural validity	Consistency with the way the enterprise defines and organizes information.
Simplicity	Ease of understanding by IS professionals and nontechnical users.
Expressibility	Ability to distinguish between different data, relationships between data, and constraints.
Nonredundancy	Exclusion of extraneous information; in particular, the representation of any one piece of information exactly once.
Shareability	Not specific to any particular application or technology and thereby usable by many.
Extensibility	Ability to evolve to support new requirements with minimal effect on existing users.
Integrity	Consistency with the way the enterprise uses and manages information.
Diagrammatic representation	Ability to represent a model using an easily understood diagrammatic notation.

The first phase of database design is called **conceptual database design** and involves the creation of a conceptual data model of the part of the enterprise that we are interested in modeling. The data model is built using the information documented in the users' requirements specification. Conceptual database design is entirely independent of implementation details such as the target DBMS software, application programs, programming languages, hardware platform, or any other physical considerations. In Chapter 16, we present a practical step-by-step guide on how to perform conceptual database design.

Throughout the process of developing a conceptual data model, the model is tested and validated against the users' requirements. The conceptual data model of the enterprise is a source of information for the next phase, namely logical database design.

Logical database design

Logical database design	The process of constructing a model of the data used in an enterprise based on a specific data model, but independent of a particular DBMS and other physical considerations.

The second phase of database design is called **logical database design**, which results in the creation of a logical data model of the part of the enterprise that we interested in modeling. The conceptual data model created in the previous phase is refined and mapped onto a logical data model. The logical data model is based on the target data model for the database (for example, the relational data model).

Whereas a conceptual data model is independent of all physical considerations, a logical model is derived knowing the underlying data model of the target DBMS. In other words, we know that the DBMS is, for example, relational, network, hierarchical, or object-oriented. However, we ignore any other aspects of the chosen DBMS and, in particular, any physical details, such as storage structures or indexes.

Throughout the process of developing a logical data model, the model is tested and validated against the users' requirements. The technique of **normalization** is used to test the correctness of a logical data model. Normalization ensures that the relations derived from the data model do not display data redundancy, which can cause update anomalies when implemented. In Chapter 14 we illustrate the problems associated with data redundancy and describe the process of normalization in detail. The logical data model should also be examined to ensure that it supports the transactions specified by the users.

The logical data model is a source of information for the next phase, namely physical database design, providing the physical database designer with a vehicle for making trade-offs that are very important to efficient database design. The logical model also serves an important role during the operational maintenance stage of the database system development lifecycle. Properly maintained and kept up to date, the data model allows future changes to application programs or data to be accurately and efficiently represented by the database.

In Chapter 17 we present a practical step-by-step guide for logical database design.

Physical database design

Physical database design

The process of producing a description of the implementation of the database on secondary storage; it describes the base relations, file organizations, and indexes used to achieve efficient access to the data, and any associated integrity constraints and security measures.

Physical database design is the third and final phase of the database design process, during which the designer decides how the database is to be implemented. The previous phase of database design involved the development of a logical structure for the database, which describes relations and enterprise constraints. Although this structure is DBMS-independent, it is developed in accordance with a particular data model, such as the relational, network, or hierarchic. However, in developing the physical database design, we must first identify the target DBMS. Therefore, physical design is tailored to a specific DBMS system. There is feedback between physical and logical design, because decisions are taken during physical design for improving performance that may affect the structure of the logical data model.

In general, the main aim of physical database design is to describe how we intend to physically implement the logical database design. For the relational model, this involves:

- creating a set of relational tables and the constraints on these tables from the information presented in the logical data model;
- identifying the specific storage structures and access methods for the data to achieve an optimum performance for the database system;
- designing security protection for the system.

Ideally, conceptual and logical database design for larger systems should be separated from physical design for three main reasons:

- it deals with a different subject matter—the *what,* not the *how;*
- it is performed at a different time—the *what* must be understood before the *how* can be determined;
- it requires different skills, which are often found in different people.

Database design is an iterative process that has a starting point and an almost endless procession of refinements. They should be viewed as learning processes. As the designers come to understand the workings of the enterprise and the meanings of its data, and express that understanding in the selected data models, the information gained may well necessitate changes to other parts of the design. In particular, conceptual and logical database designs are critical to the overall success of the system. If the designs are not a true representation of the enterprise, it will be difficult, if not impossible, to define all the required user views or to maintain database integrity. It may even prove difficult to define the physical implementation or to maintain acceptable system performance. On the other hand, the ability to adjust to change is one hallmark of good database design. Therefore, it is worthwhile spending the time and energy necessary to produce the best possible design.

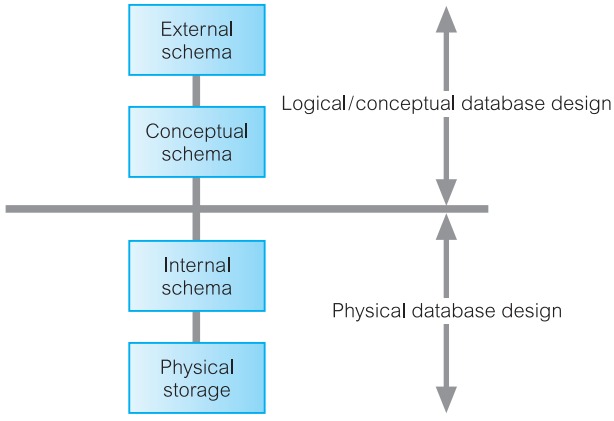


Figure 10.5
Data modeling and the ANSI-SPARC architecture.

In Chapter 2, we discussed the three-level ANSI-SPARC architecture for a database system, consisting of external, conceptual, and internal schemas. Figure 10.5 illustrates the correspondence between this architecture and conceptual, logical, and physical database design. In Chapters 18 and 19 we present a step-by-step methodology for the physical database design phase.

10.7 DBMS Selection

DBMS selection The selection of an appropriate DBMS to support the database system.

If no DBMS exists, an appropriate part of the lifecycle in which to make a selection is between the conceptual and logical database design phases (see Figure 10.1). However, selection can be done at any time prior to logical design provided sufficient information is available regarding system requirements such as performance, ease of restructuring, security, and integrity constraints.

Although DBMS selection may be infrequent, as enterprise needs expand or existing systems are replaced, it may become necessary at times to evaluate new DBMS products. In such cases, the aim is to select a system that meets the current and future requirements of the enterprise, balanced against costs that include the purchase of the DBMS product, any additional software/hardware required to support the database system, and the costs associated with changeover and staff training.

A simple approach to selection is to check off DBMS features against requirements. In selecting a new DBMS product, there is an opportunity to ensure that the selection process is well planned, and the system delivers real benefits to the enterprise. In the following section we describe a typical approach to selecting the "best" DBMS.

10.7.1 Selecting the DBMS

The main steps to selecting a DBMS are listed in Table 10.3.

TABLE 10.3 Main steps to selecting a DBMS.

Define Terms of Reference of study
Shortlist two or three products
Evaluate products
Recommend selection and produce report

Define Terms of Reference of study

The Terms of Reference for the DBMS selection is established, stating the objectives and scope of the study and the tasks that need to be undertaken. This document may also include a description of the criteria (based on the users' requirements specification) to be used to evaluate the DBMS products, a preliminary list of possible products, and all necessary constraints and timescales for the study.

Shortlist two or three products

Criteria considered to be "critical" to a successful implementation can be used to produce a preliminary list of DBMS products for evaluation. For example, the decision to include a DBMS product may depend on the budget available, level of vendor support, compatibility with other software, and whether the product runs on particular hardware. Additional useful information on a product can be gathered by contacting existing users, who may provide specific details on how good the vendor support actually is, on how the product supports particular applications, and whether certain hardware platforms are more problematic than others. There may also be benchmarks available that compare the performance of DBMS products. Following an initial study of the functionality and features of DBMS products, a shortlist of two or three products is identified.

The World Wide Web is an excellent source of information and can be used to identify potential candidate DBMSs. For example, InfoWorld's online technology test center (available at www.infoworld/test-center.com) provides a comprehensive review of DBMS products. Vendors' Web sites can also provide valuable information on DBMS products.

Evaluate products

There are various features that can be used to evaluate a DBMS product. For the purposes of the evaluation, these features can be assessed as groups (for example, data definition) or individually (for example, data types available). Table 10.4 lists possible features for DBMS product evaluation grouped by data definition, physical definition, accessibility, transaction handling, utilities, development, and other features.

If features are checked off simply with an indication of how good or bad each is, it may be difficult to make comparisons between DBMS products. A more useful approach is to weight features and/or groups of features with respect to their importance to the organization, and to obtain an overall weighted value that can

TABLE 10.4 Features for DBMS evaluation.

DATA DEFINITION	**PHYSICAL DEFINITION**
Primary key enforcement	File structures available
Foreign key specification	File structure maintenance
Data types available	Ease of reorganization
Data type extensibility	Indexing
Domain specification	Variable length fields/records
Ease of restructuring	Data compression
Integrity controls	Encryption routines
View mechanism	Memory requirements
Data dictionary	Storage requirements
Data independence	
Underlying data model	
Schema evolution	
ACCESSIBILITY	**TRANSACTION HANDLING**
Query language: SQL2/SQL:2011/ODMG compliant	Backup and recovery routines Checkpointing facility
Interfacing to 3GLs	Logging facility
Multi-user	Granularity of concurrency
Security	Deadlock resolution strategy
• Access controls	Advanced transaction models
• Authorization mechanism	Parallel query processing
UTILITIES	**DEVELOPMENT**
Performance measuring	4GL/5GL tools
Tuning	CASE tools
Load/unload facilities	Windows capabilities
User usage monitoring	Stored procedures, triggers, and rules
Database administration support	Web development tools
OTHER FEATURES	
Upgradability	Interoperability with other DBMSs and other systems
Vendor stability	Web integration
User base	Replication utilities
Training and user support	Distributed capabilities

(Continued)

TABLE 10.4 *(Continued)*

OTHER FEATURES	
Documentation	Portability
Operating system required	Hardware required
Cost	Network support
Online help	Object-oriented capabilities
Standards used	Architecture (2- or 3-tier client/server)
Version management	Performance
Extensibile query optimization	Transaction throughput
Scalability	Maximum number of concurrent users
Support for reporting and analytical tools	XML and Web services support

be used to compare products. Table 10.5 illustrates this type of analysis for the "Physical definition" group for a sample DBMS product. Each selected feature is given a rating out of 10, a weighting out of 1 to indicate its importance relative to other features in the group, and a calculated score based on the rating times the weighting. For example, in Table 10.5 the feature "Ease of reorganization" is given a rating of 4, and a weighting of 0.25, producing a score of 1.0. This feature is given the highest weighting in this table, indicating its importance in this part of the evaluation. Additionally, the "Ease of reorganization" feature is weighted, for example, five times higher than the feature "Data compression" with the lowest weighting of 0.05, whereas the two features "Memory requirements" and "Storage

TABLE 10.5 Analysis of features for DBMS product evaluation.

DBMS: Sample Product		Vendor: Sample Vendor		
Physical Definition Group				
FEATURES	**COMMENTS**	**RATING**	**WEIGHTING**	**SCORE**
File structures available	Choice of 4	8	0.15	1.2
File structure maintenance	*Not* self-regulating	6	0.2	1.2
Ease of reorganization		4	0.25	1.0
Indexing		6	0.15	0.9
Variable length fields/records		6	0.15	0.9
Data compression	Specify with file structure	7	0.05	0.35
Encryption routines	Choice of 2	4	0.05	0.2
Memory requirements		0	0.00	0
Storage requirements		0	0.00	0
Totals		41	1.0	**5.75**
Physical definition group		5.75	0.25	**1.44**

requirements" are given a weighting of 0.00 and are therefore not included in this evaluation.

We next sum all the scores for each evaluated feature to produce a total score for the group. The score for the group is then itself subject to a weighting, to indicate its importance relative to other groups of features included in the evaluation. For example, in Table 10.5, the total score for the "Physical definition" group is 5.75; however, this score has a weighting of 0.25.

Finally, all the weighted scores for each assessed group of features are summed to produce a single score for the DBMS product, which is compared with the scores for the other products. The product with the highest score is the "winner".

In addition to this type of analysis, we can also evaluate products by allowing vendors to demonstrate their product or by testing the products in-house. In-house evaluation involves creating a pilot testbed using the candidate products. Each product is tested against its ability to meet the users' requirements for the database system. Benchmarking reports published by the Transaction Processing Council can be found at www.tpc.org.

Recommend selection and produce report

The final step of the DBMS selection is to document the process and to provide a statement of the findings and recommendations for a particular DBMS product.

10.8 Application Design

Application design — The design of the user interface and the application programs that use and process the database.

In Figure 10.1, observe that database and application design are parallel activities of the database system development lifecycle. In most cases, it is not possible to complete the application design until the design of the database itself has taken place. On the other hand, the database exists to support the applications, and so there must be a flow of information between application design and database design.

We must ensure that all the functionality stated in the users' requirements specification is present in the application design for the database system. This involves designing the application programs that access the database and designing the transactions, (that is, the database access methods). In addition to designing how the required functionality is to be achieved, we have to design an appropriate user interface to the database system. This interface should present the required information in a user-friendly way. The importance of user interface design is sometimes ignored or left until late in the design stages. However, it should be recognized that the interface may be one of the most important components of the system. If it is easy to learn, simple to use, straightforward and forgiving, the users will be inclined to make good use of what information is presented. On the other hand, if the interface has none of these characteristics, the system will undoubtedly cause problems.

In the following sections, we briefly examine two aspects of application design: transaction design and user interface design.

10.8.1 Transaction Design

Before discussing transaction design, we first describe what a transaction represents.

Transaction An action, or series of actions, carried out by a single user or application program, that accesses or changes the content of the database.

Transactions represent "real-world" events such as the registering of a property for rent, the addition of a new member of staff, the registration of a new client, and the renting out of a property. These transactions have to be applied to the database to ensure that data held by the database remains current with the "real-world" situation and to support the information needs of the users.

A transaction may be composed of several operations, such as the transfer of money from one account to another. However, from the user's perspective, these operations still accomplish a single task. From the DBMS's perspective, a transaction transfers the database from one consistent state to another. The DBMS ensures the consistency of the database even in the presence of a failure. The DBMS also ensures that once a transaction has completed, the changes made are permanently stored in the database and cannot be lost or undone (without running another transaction to compensate for the effect of the first transaction). If the transaction cannot complete for any reason, the DBMS should ensure that the changes made by that transaction are undone. In the example of the bank transfer, if money is debited from one account and the transaction fails before crediting the other account, the DBMS should undo the debit. If we were to define the debit and credit operations as separate transactions, then once we had debited the first account and completed the transaction, we are not allowed to undo that change (without running another transaction to credit the debited account with the required amount).

The purpose of transaction design is to define and document the high-level characteristics of the transactions required on the database, including:

- data to be used by the transaction;
- functional characteristics of the transaction;
- output of the transaction;
- importance to the users;
- expected rate of usage.

This activity should be carried out early in the design process to ensure that the implemented database is capable of supporting all the required transactions. There are three main types of transactions: retrieval transactions, update transactions, and mixed transactions:

- **Retrieval transactions** are used to retrieve data for display on the screen or in the production of a report. For example, the operation to search for and display

the details of a property (given the property number) is an example of a retrieval transaction.

- **Update transactions** are used to insert new records, delete old records, or modify existing records in the database. For example, the operation to insert the details of a new property into the database is an example of an update transaction.
- **Mixed transactions** involve both the retrieval and updating of data. For example, the operation to search for and display the details of a property (given the property number) and then update the value of the monthly rent is an example of a mixed transaction.

10.8.2 User Interface Design Guidelines

Before implementing a form or report, it is essential that we first design the layout. Useful guidelines to follow when designing forms or reports are listed in Table 10.6 (Shneiderman et al., 2009).

Meaningful title

The information conveyed by the title should clearly and unambiguously identify the purpose of the form/report.

Comprehensible instructions

Familiar terminology should be used to convey instructions to the user. The instructions should be brief, and, when more information is required, help screens should

TABLE 10.6 Guidelines for form/report design.

Meaningful title
Comprehensible instructions
Logical grouping and sequencing of fields
Visually appealing layout of the form/report
Familiar field labels
Consistent terminology and abbreviations
Consistent use of color
Visible space and boundaries for data entry fields
Convenient cursor movement
Error correction for individual characters and entire fields
Error messages for unacceptable values
Optional fields marked clearly
Explanatory messages for fields
Completion signal

be made available. Instructions should be written in a consistent grammatical style, using a standard format.

Logical grouping and sequencing of fields

Related fields should be positioned together on the form/report. The sequencing of fields should be logical and consistent.

Visually appealing layout of the form/report

The form/report should present an attractive interface to the user. The form/report should appear balanced with fields or groups of fields evenly positioned throughout the form/report. There should not be areas of the form/report that have too few or too many fields. Fields or groups of fields should be separated by a regular amount of space. Where appropriate, fields should be vertically or horizontally aligned. In cases where a form on screen has a hardcopy equivalent, the appearance of both should be consistent.

Familiar field labels

Field labels should be familiar. For example, if Sex were replaced by Gender, it is possible that some users would be confused.

Consistent terminology and abbreviations

An agreed list of familiar terms and abbreviations should be used consistently.

Consistent use of color

Color should be used to improve the appearance of a form/report and to highlight important fields or important messages. To achieve this, color should be used in a consistent and meaningful way. For example, fields on a form with a white background may indicate data entry fields and those with a blue background may indicate display-only fields.

Visible space and boundaries for data-entry fields

A user should be visually aware of the total amount of space available for each field. This allows a user to consider the appropriate format for the data before entering the values into a field.

Convenient cursor movement

A user should easily identify the operation required to move a cursor throughout the form/report. Simple mechanisms such as using the Tab key, arrows, or the mouse pointer should be used.

Error correction for individual characters and entire fields

A user should easily identify the operation required to make alterations to field values. Simple mechanisms should be available, such as using the Backspace key or overtyping.

Error messages for unacceptable values

If a user attempts to enter incorrect data into a field, an error message should be displayed. The message should inform the user of the error and indicate permissible values.

Optional fields marked clearly

Optional fields should be clearly identified for the user. This can be achieved using an appropriate field label or by displaying the field using a color that indicates the type of the field. Optional fields should be placed after required fields.

Explanatory messages for fields

When a user places a cursor on a field, information about the field should appear in a regular position on the screen, such as a window status bar.

Completion signal

It should be clear to a user when the process of filling in fields on a form is complete. However, the option to complete the process should not be automatic, as the user may wish to review the data entered.

10.9 Prototyping

At various points throughout the design process, we have the option to either fully implement the database system or build a prototype.

Prototyping | Building a working model of a database system.

A prototype is a working model that does not normally have all the required features or provide all the functionality of the final system. The main purpose of developing a prototype database system is to allow users to use the prototype to identify the features of the system that work well or are inadequate, and—if possible—to suggest improvements or even new features to the database system. In this way, we can greatly clarify the users' requirements for both the users and developers of the system and evaluate the feasibility of a particular system design. Prototypes should have the major advantage of being relatively inexpensive and quick to build.

There are two prototyping strategies in common use today: requirements prototyping and evolutionary prototyping. **Requirements prototyping** uses a prototype to determine the requirements of a proposed database system, and once the requirements are complete, the prototype is discarded. Although **evolutionary prototyping** is used for the same purposes, the important difference is that the prototype is not discarded, but with further development becomes the working database system.

10.10 Implementation

Implementation | The physical realization of the database and application designs.

On completion of the design stages (which may or may not have involved prototyping), we are now in a position to implement the database and the application programs. The database implementation is achieved using the DDL of the selected DBMS or a GUI, which provides the same functionality while hiding the low-level DDL statements. The DDL statements are used to create the database structures and empty database files. Any specified user views are also implemented at this stage.

The application programs are implemented using the preferred third- or fourth-generation language (3GL or 4GL). Parts of these application programs are the database transactions, which are implemented using the DML of the target DBMS, possibly embedded within a host programming language, such as Visual Basic (VB), VB.net, Python, Delphi, C, C++, C#, Java, COBOL, Fortran, Ada, or Pascal. We also implement the other components of the application design such as menu screens, data entry forms, and reports. Again, the target DBMS may have its own fourth-generation tools that allow rapid development of applications through the provision of nonprocedural query languages, reports generators, forms generators, and application generators.

Security and integrity controls for the system are also implemented. Some of these controls are implemented using the DDL, but others may need to be defined outside the DDL, using, for example, the supplied DBMS utilities or operating system controls. Note that SQL is both a DML and a DDL, as described in Chapters 6, 7, and 8.

10.11 Data Conversion and Loading

Data conversion and loading — Transferring any existing data into the new database and converting any existing applications to run on the new database.

This stage is required only when a new database system is replacing an old system. Nowadays, it is common for a DBMS to have a utility that loads existing files into the new database. The utility usually requires the specification of the source file and the target database, and then automatically converts the data to the required format of the new database files. Where applicable, it may be possible for the developer to convert and use application programs from the old system for use by the new system. Whenever conversion and loading are required, the process should be properly planned to ensure a smooth transition to full operation.

10.12 Testing

Testing — The process of running the database system with the intent of finding errors.

Before going live, the newly developed database system should be thoroughly tested. This is achieved using carefully planned test strategies and realistic data, so that the entire testing process is methodically and rigorously carried out. Note that in our definition of testing we have not used the commonly held view that testing is the process of demonstrating that faults are not present. In fact, testing cannot

show the absence of faults; it can show only that software faults are present. If testing is conducted successfully, it will uncover errors with the application programs and possibly the database structure. As a secondary benefit, testing demonstrates that the database and the application programs *appear* to be working according to their specification and that performance requirements *appear* to be satisfied. In addition, metrics collected from the testing stage provide a measure of software reliability and software quality.

As with database design, the users of the new system should be involved in the testing process. The ideal situation for system testing is to have a test database on a separate hardware system, but often this is not possible. If real data is to be used, it is essential to have backups made in case of error.

Testing should also cover usability of the database system. Ideally, an evaluation should be conducted against a usability specification. Examples of criteria that can be used to conduct the evaluation include the following (Sommerville, 2010):

- Learnability: How long does it take a new user to become productive with the system?
- Performance: How well does the system response match the user's work practice?
- Robustness: How tolerant is the system of user error?
- Recoverability: How good is the system at recovering from user errors?
- Adapatability: How closely is the system tied to a single model of work?

Some of these criteria may be evaluated in other stages of the lifecycle. After testing is complete, the database system is ready to be "signed off" and handed over to the users.

10.13 Operational Maintenance

Operational maintenance The process of monitoring and maintaining the database system following installation.

In the previous stages, the database system has been fully implemented and tested. The system now moves into a maintenance stage, which involves the following activities:

- Monitoring the performance of the system. If the performance falls below an acceptable level, tuning or reorganization of the database may be required.
- Maintaining and upgrading the database system (when required). New requirements are incorporated into the database system through the preceding stages of the lifecycle.

Once the database system is fully operational, close monitoring takes place to ensure that performance remains within acceptable levels. A DBMS normally provides various utilities to aid database administration, including utilities to load data into a database and to monitor the system. The utilities that allow system monitoring give information on, for example, database usage, locking efficiency (including number of deadlocks that have occurred, and so on), and query execution strategy. The DBA can use this information to tune the system to give better

performance; for example, by creating additional indexes to speed up queries, by altering storage structures, or by combining or splitting tables.

The monitoring process continues throughout the life of a database system and in time may lead to reorganization of the database to satisfy the changing requirements. These changes in turn provide information on the likely evolution of the system and the future resources that may be needed. This, together with knowledge of proposed new applications, enables the DBA to engage in capacity planning and to notify or alert senior staff to adjust plans accordingly. If the DBMS lacks certain utilities, the DBA can either develop the required utilities in-house or purchase additional vendor tools, if available. We discuss database administration in more detail in Chapter 20.

When a new database system is brought online, the users should operate it in parallel with the old system for a period of time. This approach safeguards current operations in case of unanticipated problems with the new system. Periodic checks on data consistency between the two systems need to be made, and only when both systems appear to be producing the same results consistently should the old system be dropped. If the changeover is too hasty, the end-result could be disastrous. Despite the foregoing assumption that the old system may be dropped, there may be situations in which both systems are maintained.

10.14 CASE Tools

The first stage of the database system development lifecycle—database planning—may also involve the selection of suitable Computer-Aided Software Engineering (CASE) tools. In its widest sense, CASE can be applied to any tool that supports software development. Appropriate productivity tools are needed by data administration and database administration staff to permit the database development activities to be carried out as efficiently and effectively as possible. CASE support may include:

- a data dictionary to store information about the database system's data;
- design tools to support data analysis;
- tools to permit development of the corporate data model, and the conceptual and logical data models;
- tools to enable the prototyping of applications.

CASE tools may be divided into three categories: upper-CASE, lower-CASE, and integrated-CASE, as illustrated in Figure 10.6. **Upper-CASE** tools support the initial stages of the database system development lifecycle, from planning through to database design. **Lower-CASE** tools support the later stages of the lifecycle, from implementation through testing, to operational maintenance. **Integrated-CASE** tools support all stages of the lifecycle and thus provide the functionality of both upper- and lower-CASE in one tool.

Benefits of CASE

The use of appropriate CASE tools should improve the productivity of developing a database system. We use the term "productivity" to relate both to the efficiency of the development process and to the effectiveness of the developed system.

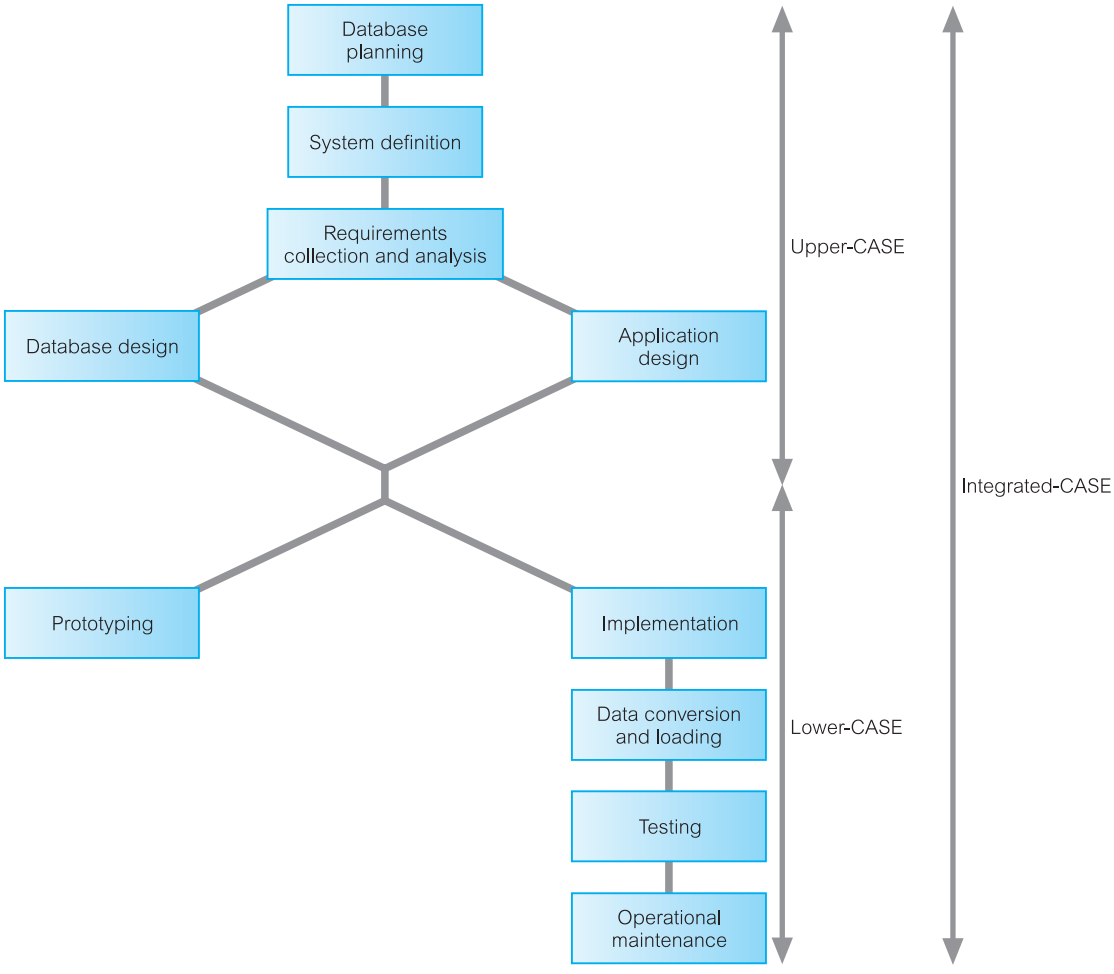


Figure 10.6 Application of CASE tools.

Efficiency refers to the cost, in terms of time and money, of realizing the database system. CASE tools aim to support and automate the development tasks and thus improve efficiency. *Effectiveness* refers to the extent to which the system satisfies the information needs of its users. In the pursuit of greater productivity, raising the effectiveness of the development process may be even more important than increasing its efficiency. For example, it would not be sensible to develop a database system extremely efficiently when the end-product is not what the users want. In this way, effectiveness is related to the quality of the final product. Because computers are better than humans at certain tasks—for example, consistency checking—CASE tools can be used to increase the effectiveness of some tasks in the development process.

CASE tools provide the following benefits that improve productivity:

- *Standards:* CASE tools help to enforce standards on a software project or across the organization. They encourage the production of standard test

components that can be reused, thus simplifying maintenance and increasing productivity.

- *Integration:* CASE tools store all the information generated in a repository, or data dictionary. Thus, it should be possible to store the data gathered during all stages of the database system development lifecycle. The data then can be linked together to ensure that all parts of the system are integrated. In this way, an organization's information system no longer has to consist of independent, unconnected components.
- *Support for standard methods:* Structured techniques make significant use of diagrams, which are difficult to draw and maintain manually. CASE tools simplify this process, resulting in documentation that is correct and more current.
- *Consistency:* Because all the information in the data dictionary is interrelated, CASE tools can check its consistency.
- *Automation:* Some CASE tools can automatically transform parts of a design specification into executable code. This feature reduces the work required to produce the implemented system, and may eliminate errors that arise during the coding process.

Chapter Summary

- An **information system** is the resources that enable the collection, management, control, and dissemination of information throughout an organization.
- A computer-based information system includes the following components: database, database software, application software, computer hardware (including storage media), and personnel using and developing the system.
- The database is a fundamental component of an information system, and its development and usage should be viewed from the perspective of the wider requirements of the organization. Therefore, the lifecycle of an organizational information system is inherently linked to the lifecycle of the database that supports it.
- The main stages of the **database system development lifecycle** include: database planning, system definition, requirements collection and analysis, database design, DBMS selection (optional), application design, prototyping (optional), implementation, data conversion and loading, testing, and operational maintenance.
- **Database planning** involves the management activities that allow the stages of the database system development lifecycle to be realized as efficiently and effectively as possible.
- **System definition** involves identifying the scope and boundaries of the database system and user views. A **user view** defines what is required of a database system from the perspective of a particular job role (such as Manager or Supervisor) or enterprise application (such as marketing, personnel, or stock control).
- **Requirements collection and analysis** is the process of collecting and analyzing information about the part of the organization that is to be supported by the database system, and using this information to identify the requirements for the new system. There are three main approaches to managing the requirements for a database system that has multiple user views: the **centralized** approach, the **view integration** approach, and a combination of both approaches.
- The **centralized** approach involves merging the requirements for each user view into a single set of requirements for the new database system. A data model representing all user views is created during the database design stage. In the **view integration** approach, requirements for each user view remain as separate lists. Data models representing each user view are created then merged later during the database design stage.

- **Database design** is the process of creating a design that will support the enterprise's mission statement and mission objectives for the required database system. There are three phases of database design: conceptual, logical, and physical database design.
- **Conceptual database design** is the process of constructing a model of the data used in an enterprise, independent of *all* physical considerations.
- **Logical database design** is the process of constructing a model of the data used in an enterprise based on a specific data model, but independent of a particular DBMS and other physical considerations.
- **Physical database design** is the process of producing a description of the implementation of the database on secondary storage; it describes the base relations, file organizations, and indexes used to achieve efficient access to the data, and any associated integrity constraints and security measures.
- **DBMS selection** involves selecting a suitable DBMS for the database system.
- **Application design** involves user interface design and transaction design, which describes the application programs that use and process the database. A database **transaction** is an action or series of actions carried out by a single user or application program, which accesses or changes the content of the database.
- **Prototyping** involves building a working model of the database system, which allows the designers or users to visualize and evaluate the system.
- **Implementation** is the physical realization of the database and application designs.
- **Data conversion and loading** involves transferring any existing data into the new database and converting any existing applications to run on the new database.
- **Testing** is the process of running the database system with the intent of finding errors.
- **Operational maintenance** is the process of monitoring and maintaining the system following installation.
- **Computer-Aided Software Engineering** (CASE) applies to any tool that supports software development and permits the database system development activities to be carried out as efficiently and effectively as possible. CASE tools may be divided into three categories: upper-CASE, lower-CASE, and integrated-CASE.

Review Questions

10.1 Discuss the interdependence that exists between DSDLC stages.

10.2 What do you understand by the term "system mission"? Why is it important during system development?

10.3 Describe the main purpose(s) and activities associated with each stage of the database system development lifecycle.

10.4 Discuss what a user view represents in the context of a database system.

10.5 Discuss the main approaches to database design. Discuss the contexts where each is appropriate.

10.6 Compare and contrast the three phases of database design.

10.7 What are the main purposes of data modeling and identify the criteria for an optimal data model?

10.8 Identify the stage(s) in which it is appropriate to select a DBMS and describe an approach to selecting the "best" DBMS.

10.9 Application design involves transaction design and user interface design. Describe the purpose and main activities associated with each.

10.10 Discuss why testing cannot show the absence of faults, only that software faults are present.

10.11 What is the difference between the prototyping approach and the database systems development lifecycle?

Exercises

10.12 Assume you have been contracted to develop a database system for a university library. You are required to use a systems development lifecycle approach. Discuss how you are going to approach the project. Describe user groups that will be involved during the requirement analysis. What are the key issues that need to be answered during fact finding?

10.13 Describe the process of evaluating and selecting a DBMS product for each of the case studies described in Appendix B.

10.14 Assume that you are an employee of a consultancy company that specializes in the analysis, design, and implementation of database systems. A client has recently approached your company with a view to implementing a database system but they are not familiar with the development process. You have been assigned the task to present an overview of the Database System Development Lifecycle (DSDL) to them, identifying the main stages of this lifecycle. With this task in mind, create a slide presentation and/or short report for the client. (The client for this exercise can be any one of the fictitious case studies given in Appendix B or some real company identified by you or your professor).

10.15 This exercise requires you to first gain permission to interview one or more people responsible for the development and/or administration of a real database system. During the interview(s), find out the following information:

(a) The approach taken to develop the database system.

(b) How the approach taken differs or is similar to the DSDL approach described in this chapter.

(c) How the requirements for different users (user views) of the database systems were managed.

(d) Whether a CASE tool was used to support the development of the database system.

(e) How the DBMS product was evaluated and then selected.

(f) How the database system is monitored and maintained.

CHAPTER

Database Analysis and the *DreamHome* Case Study

Chapter Objectives

In this chapter you will learn:

- When fact-finding techniques are used in the database system development lifecycle.
- The types of facts collected in each stage of the database system development lifecycle.
- The types of documentation produced in each stage of the database system development lifecycle.
- The most commonly used fact-finding techniques.
- How to use each fact-finding technique and the advantages and disadvantages of each.
- About a property rental company called *DreamHome*.
- How to apply fact-finding techniques to the early stages of the database system development lifecycle.

In Chapter 10 we introduced the stages of the database system development lifecycle. There are many occasions during these when it is critical that the database developer captures the necessary facts to build the required database system. The necessary facts include, for example, the terminology used within the enterprise, problems encountered using the current system, opportunities sought from the new system, necessary constraints on the data and users of the new system, and a prioritized set of requirements for the new system. These facts are captured using fact-finding techniques.

Fact-finding The formal process of using techniques such as interviews and questionnaires to collect facts about systems, requirements, and preferences.

In this chapter we discuss when a database developer might use fact-finding techniques and what types of facts should be captured. We present an overview of how these facts are used to generate the main types of documentation used throughout the database system development lifecycle. We describe the most commonly used fact-finding techniques and identify the advantages and disadvantages

of each. We finally demonstrate how some of these techniques may be used during the earlier stages of the database system development lifecycle using a property management company called *DreamHome*. The *DreamHome* case study is used throughout this book.

Structure of this Chapter In Section 11.1 we discuss when a database developer might use fact-finding techniques. (Throughout this book we use the term "database developer" to refer to a person or group of people responsible for the analysis, design, and implementation of a database system.) In Section 11.2 we illustrate the types of facts that should be collected and the documentation that should be produced at each stage of the database system development lifecycle. In Section 11.3 we describe the five most commonly used fact-finding techniques and identify the advantages and disadvantages of each. In Section 11.4 we demonstrate how fact-finding techniques can be used to develop a database system for a case study called *DreamHome*, a property management company. We begin this section by providing an overview of the *DreamHome* case study. We then examine the first three stages of the database system development lifecycle, namely database planning, system definition, and requirements collection and analysis. For each stage we demonstrate the process of collecting data using fact-finding techniques and describe the documentation produced.

11.1 When Are Fact-Finding Techniques Used?

There are many occasions for fact-finding during the database system development life cycle. However, fact-finding is particularly crucial to the early stages of the lifecycle, including the database planning, system definition, and requirements collection and analysis stages. It is during these early stages that the database developer captures the essential facts necessary to build the required database. Fact-finding is also used during database design and the later stages of the lifecycle, but to a lesser extent. For example, during physical database design, fact-finding becomes technical as the database developer attempts to learn more about the DBMS selected for the database system. Also, during the final stage, operational maintenance, fact-finding is used to determine whether a system requires tuning to improve performance or further development to include new requirements.

Note that it is important to have a rough estimate of how much time and effort is to be spent on fact-finding for a database project. As we mentioned in Chapter 10, too much study too soon leads to *paralysis by analysis*. However, too little thought can result in an unnecessary waste of both time and money, due to working on the wrong solution to the wrong problem.

11.2 What Facts Are Collected?

Throughout the database system development lifecycle, the database developer needs to capture facts about the current and/or future system. Table 11.1 provides examples of the sorts of data captured and the documentation produced for each stage of the lifecycle. As we mentioned in Chapter 10, the stages of the database system development lifecycle are not strictly sequential, but involve some amount of repetition of previous stages through feedback loops. This is also true for the data captured and the documentation produced at each stage. For example, problems encountered during database design may necessitate additional data capture on the requirements for the new system.

TABLE 11.1 Examples of the data captured and the documentation produced for each stage of the database system development lifecycle.

STAGE OF DATABASE SYSTEM DEVELOPMENT LIFECYCLE	EXAMPLES OF DATA CAPTURED	EXAMPLES OF DOCUMENTATION PRODUCED
Database planning	Aims and objectives of database project	Mission statement and objectives of database system
System definition	Description of major user views (includes job roles or business application areas)	Definition of scope and boundary of database system; definition of user views to be supported
Requirements collection and analysis	Requirements for user views; systems specifications, including performance and security requirements	Users' and system requirements specifications
Database design	Users' responses to checking the conceptual/logical database design; functionality provided by target DBMS	Conceptual/logical database design (includes ER model(s), data dictionary, and relational schema); physical database design
Application design	Users' responses to checking interface design	Application design (includes description of programs and user interface)
DBMS selection	Functionality provided by target DBMS	DBMS evaluation and recommendations
Prototyping	Users' responses to prototype	Modified users' requirements and systems specifications
Implementation	Functionality provided by target DBMS	
Data conversion and loading	Format of current data; data import capabilities of target DBMS	
Testing	Test results	Testing strategies used; analysis of test results
Operational maintenance	Performance testing results; new or changing user and system requirements	User manual; analysis of performance results; modified users' requirements and systems specifications

11.3 Fact-Finding Techniques

A database developer normally uses several fact-finding techniques during a single database project. There are five commonly used fact-finding techniques:

- examining documentation;
- interviewing;
- observing the enterprise in operation;
- research;
- questionnaires.

In the following sections we describe these fact-finding techniques and identify the advantages and disadvantages of each.

11.3.1 Examining Documentation

Examining documentation can be useful when we are trying to gain some insight as to how the need for a database arose. We may also find that documentation can help to provide information on the part of the enterprise associated with the problem. If the problem relates to the current system, there should be documentation associated with that system. By examining documents, forms, reports, and files associated with the current system, we can quickly gain some understanding of the system. Examples of the types of documentation that should be examined are listed in Table 11.2.

11.3.2 Interviewing

Interviewing is the most commonly used and normally the most useful fact-finding technique. We can interview to collect information from individuals face-to-face. There can be several objectives to using interviewing, such as finding out

TABLE 11.2 Examples of types of documentation that should be examined.

PURPOSE OF DOCUMENTATION	EXAMPLES OF USEFUL SOURCES
Describes problem and need for database	Internal memos, emails, and minutes of meetings Employee complaints and documents that describe the problem Social media such as blogs and tweets Performance reviews/reports
Describes the part of the enterprise affected by problem	Organizational chart, mission statement, and strategic plan of the enterprise Objectives for the part of the enterprise being studied Task/job descriptions Samples of completed manual forms and reports Samples of completed computerized forms and reports
Describes current system	Various types of flowcharts and diagrams Data dictionary Database system design Program documentation User/training manuals

TABLE 11.3 Advantages and disadvantages of using interviewing as a fact-finding technique.

ADVANTAGES	DISADVANTAGES
Allows interviewee to respond freely and openly to questions	Very time-consuming and costly, and therefore may be impractical
Allows interviewee to feel part of project	Success is dependent on communication skills of interviewer
Allows interviewer to follow up on interesting comments made by interviewee	Success can be dependent on willingness of interviewees to participate in interviews
Allows interviewer to adapt or reword questions during interview	
Allows interviewer to observe interviewee's body language	

facts, verifying facts, clarifying facts, generating enthusiasm, getting the end-user involved, identifying requirements, and gathering ideas and opinions. However, using the interviewing technique requires good communication skills for dealing effectively with people who have different values, priorities, opinions, motivations, and personalities. As with other fact-finding techniques, interviewing is not always the best method for all situations. The advantages and disadvantages of using interviewing as a fact-finding technique are listed in Table 11.3.

There are two types of interview: unstructured and structured. **Unstructured interviews** are conducted with only a general objective in mind and with few, if any, specific questions. The interviewer counts on the interviewee to provide a framework and direction to the interview. This type of interview frequently loses focus and, for this reason, it often does not work well for database analysis and design.

In **structured interviews**, the interviewer has a specific set of questions to ask the interviewee. Depending on the interviewee's responses, the interviewer will direct additional questions to obtain clarification or expansion. **Open-ended questions** allow the interviewee to respond in any way that seems appropriate. An example of an open-ended question is: "Why are you dissatisfied with the report on client registration?" **Closed-ended questions** restrict answers to either specific choices or short, direct responses. An example of such a question might be: "Are you receiving the report on client registration on time?" or "Does the report on client registration contain accurate information?" Both questions require only a "Yes" or "No" response.

To ensure a successful interview includes selecting appropriate individuals to interview, preparing extensively for the interview, and conducting the interview in an efficient and effective manner.

11.3.3 Observing the Enterprise in Operation

Observation is one of the most effective fact-finding techniques for understanding a system. With this technique, it is possible to either participate in or watch a person perform activities to learn about the system. This technique is particularly useful when the validity of data collected through other methods is in question or when the complexity of certain aspects of the system prevents a clear explanation by the end-users.

TABLE 11.4 Advantages and disadvantages of using observation as a fact-finding technique.

ADVANTAGES	DISADVANTAGES
Allows the validity of facts and data to be checked	People may knowingly or unknowingly perform differently when being observed
Observer can see exactly what is being done	May miss observing tasks involving different levels of difficulty or volume normally experienced during that time period
Observer can also obtain data describing the physical environment of the task	Some tasks may not always be performed in the manner in which they are observed
Relatively inexpensive	May be impractical
Observer can do work measurements	

As with the other fact-finding techniques, successful observation requires preparation. To ensure that the observation is successful, it is important to know as much about the individuals and the activity to be observed as possible. For example, "When are the low, normal, and peak periods for the activity being observed?" and "Will the individuals be upset by having someone watch and record their actions?" The advantages and disadvantages of using observation as a fact-finding technique are listed in Table 11.4.

11.3.4 Research

A useful fact-finding technique is to research the application and problem. Computer trade journals, reference books, and the Internet (including user groups and bulletin boards) are good sources of information. They can provide information on how others have solved similar problems, plus on whether software packages exist to solve or even partially solve the problem. The advantages and disadvantages of using research as a fact-finding technique are listed in Table 11.5.

11.3.5 Questionnaires

Another fact-finding technique is to conduct surveys through questionnaires. Questionnaires are special-purpose documents that allow facts to be gathered from a large number of people while maintaining some control over their responses.

TABLE 11.5 Advantages and disadvantages of using research as a fact-finding technique.

ADVANTAGES	DISADVANTAGES
Can save time if solution already exists	Requires access to appropriate sources of information
Researcher can see how others have solved similar problems or met similar requirements	May ultimately not help in solving problem because problem is not documented elsewhere
Keeps researcher up to date with current developments	

TABLE 11.6 Advantages and disadvantages of using questionnaires as a fact-finding technique.

ADVANTAGES	DISADVANTAGES
People can complete and return questionnaires at their convenience	Number of respondents can be low, possibly only 5% to 10%
Relatively inexpensive way to gather data from a large number of people	Questionnaires may be returned incomplete
People more likely to provide the real facts as responses can be kept confidential	May not provide an opportunity to adapt or reword questions that have been misinterpreted
Responses can be tabulated and analyzed quickly	Cannot observe and analyze the respondent's body language

When dealing with a large audience, no other fact-finding technique can tabulate the same facts as efficiently. The advantages and disadvantages of using questionnaires as a fact-finding technique are listed in Table 11.6.

There are two types of questions that can be asked in a questionnaire: free-format and fixed-format. **Free-format questions** offer the respondent greater freedom in providing answers. A question is asked and the respondent records the answer in the space provided after the question. Examples of free-format questions are: "What reports do you currently receive and how are they used?" and "Are there any problems with these reports? If so, please explain." The problems with free-format questions are that the respondent's answers may prove difficult to tabulate, and in some cases, may not match the questions asked.

Fixed-format questions require specific responses from individuals. Given any question, the respondent must choose from the available answers. This makes the results much easier to tabulate. On the other hand, the respondent cannot provide additional information that might prove valuable. An example of a fixed-format question is: "The current format of the report on property rentals is ideal and should not be changed." The respondent may be given the option to answer "Yes" or "No" to this question, or be given the option to answer from a range of responses including "Strongly agree," "Agree," "No opinion," "Disagree," and "Strongly disagree."

11.4 Using Fact-Finding Techniques: A Worked Example

In this section we first present an overview of the *DreamHome* case study and then use this case study to illustrate how to establish a database project. In particular, we illustrate how fact-finding techniques can be used and the documentation produced in the early stages of the database system development lifecycle—namely, the database planning, system definition, and requirements collection and analysis stages.

11.4.1 The *DreamHome* Case Study—An Overview of the Current System

The first branch office of *DreamHome* was opened in 1992 in Glasgow in the UK. Since then, the Company has grown steadily and now has several offices in most of the main cities of the UK. However, the Company is now so large that more and more administrative staff are being employed to cope with the ever-increasing amount of paperwork. Furthermore, the communication and sharing of information between offices, even in the same city, is poor. The Director of the Company, Sally Mellweadows, feels that too many mistakes are being made and that the success of the Company will be short-lived if she does not do something to remedy the situation. She knows that a database could help in part to solve the problem and has requested that a database system be developed to support the running of *DreamHome*. The Director has provided the following brief description of how *DreamHome* currently operates.

DreamHome specializes in property management, taking an intermediate role between owners who wish to rent out their furnished property and clients of *DreamHome* who require to rent furnished property for a fixed period. *DreamHome* currently has about 2000 staff working in 100 branches. When a member of staff joins the Company, the *DreamHome* staff registration form is used. The staff registration form for Susan Brand is shown in Figure 11.1.

Each branch has an appropriate number and type of staff including a Manager, Supervisors, and Assistants. The Manager is responsible for the day-to-day running of a branch and each Supervisor is responsible for supervising a group of staff called Assistants. An example of the first page of a report listing the details of staff working at a branch office in Glasgow is shown in Figure 11.2.

DreamHome
Staff Registration Form

Staff Number SG5
Full Name Susan Brand
Sex F DOB 3-Jun-70

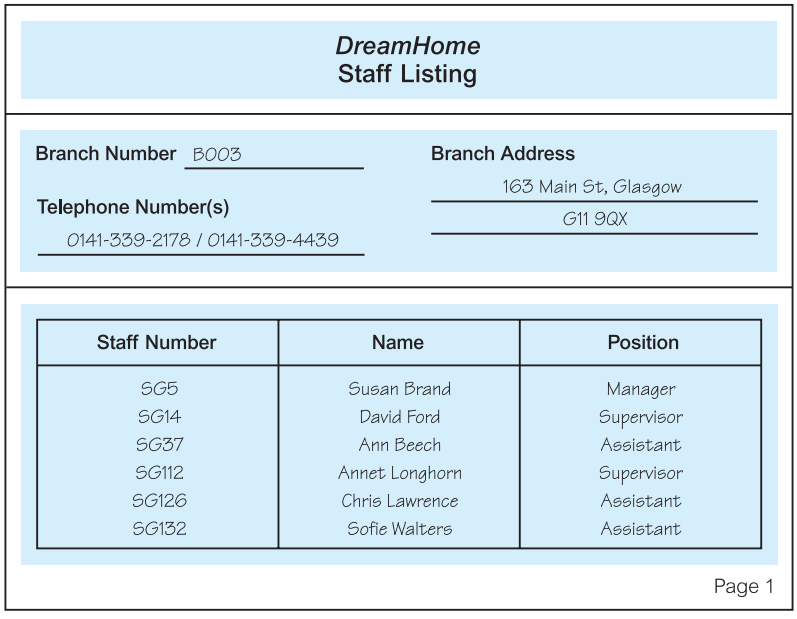
Branch Number B003
Branch Address
163 Main St, Glasgow
Telephone Number(s)
0141-339-2178 / 0141-339-4439

Position Manager
Salary 24000

Enter details where applicable
Supervisor Name

Manager Start Date 01-Jun-99
Manager Bonus 2350

Figure 11.1 The *DreamHome* staff registration form for Susan Brand.



DreamHome
Staff Listing

Branch Number B003

Branch Address
163 Main St, Glasgow
G11 9QX

Telephone Number(s)
0141-339-2178 / 0141-339-4439

Staff Number	Name	Position
SG5	Susan Brand	Manager
SG14	David Ford	Supervisor
SG37	Ann Beech	Assistant
SG112	Annet Longhorn	Supervisor
SG126	Chris Lawrence	Assistant
SG132	Sofie Walters	Assistant

Page 1

Figure 11.2 Example of the first page of a report listing the details of staff working at a *DreamHome* branch office in Glasgow.

Each branch office offers a range of properties for rent. To offer property through *DreamHome*, a property owner normally contacts the *DreamHome* branch office nearest to the property for rent. The owner provides the details of the property and agrees an appropriate rent for the property with the branch Manager. The registration form for a property in Glasgow is shown in Figure 11.3.

Once a property is registered, *DreamHome* provides services to ensure that the property is rented out for maximum return for both the property owner and, of course, *DreamHome*. These services include interviewing prospective renters (called clients), organizing viewings of the property by clients, advertising the property in local or national newspapers (when necessary), and negotiating the lease. Once rented, *DreamHome* assumes responsibility for the property including the collection of rent.

Members of the public interested in renting out property must first contact their nearest *DreamHome* branch office to register as clients of *DreamHome*. However, before registration is accepted, a prospective client is normally interviewed to record personal details and preferences of the client in terms of property requirements. The registration form for a client called Mike Ritchie is shown in Figure 11.4.

Once registration is complete, clients are provided with weekly reports that list properties currently available for rent. An example of the first page of a report listing the properties available for rent at a branch office in Glasgow is shown in Figure 11.5.

Clients may request to view one or more properties from the list and after viewing will normally provide a comment on the suitability of the property. The first page of a report describing the comments made by clients on a property in Glasgow is shown in Figure 11.6. Properties that prove difficult to rent out are normally advertised in local and national newspapers.

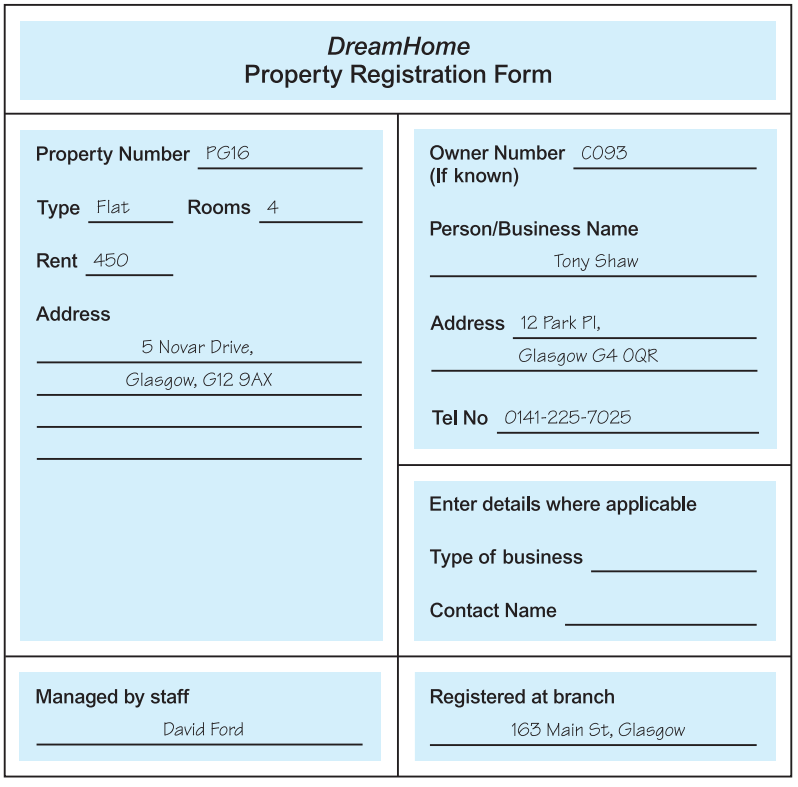


Figure 11.3 The *DreamHome* property registration form for a property in Glasgow.

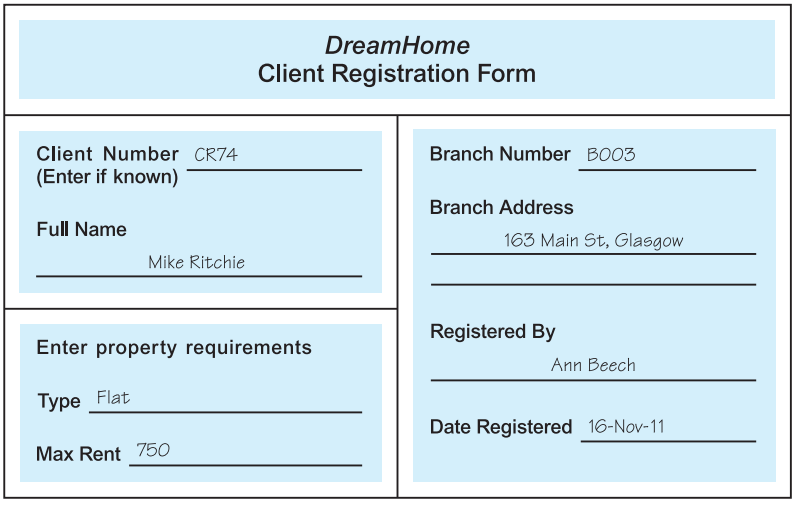
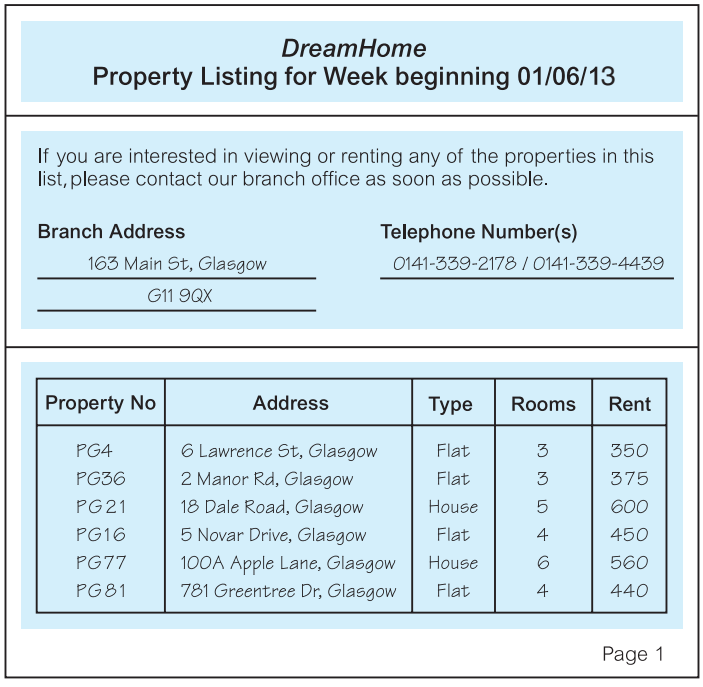


Figure 11.4 The *DreamHome* client registration form for Mike Ritchie.



DreamHome
Property Listing for Week beginning 01/06/13

If you are interested in viewing or renting any of the properties in this list, please contact our branch office as soon as possible.

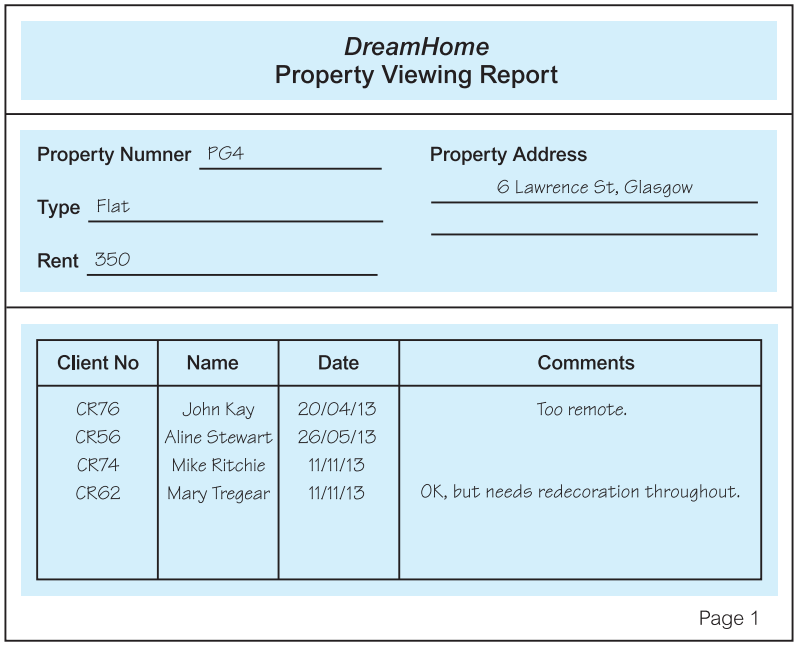
Branch Address: 163 Main St, Glasgow, G11 9QX

Telephone Number(s): 0141-339-2178 / 0141-339-4439

Property No	Address	Type	Rooms	Rent
PG4	6 Lawrence St, Glasgow	Flat	3	350
PG36	2 Manor Rd, Glasgow	Flat	3	375
PG21	18 Dale Road, Glasgow	House	5	600
PG16	5 Novar Drive, Glasgow	Flat	4	450
PG77	100A Apple Lane, Glasgow	House	6	560
PG81	781 Greentree Dr, Glasgow	Flat	4	440

Page 1

Figure 11.5 The first page of the *DreamHome* property for rent report listing property available at a branch in Glasgow.



DreamHome
Property Viewing Report

Property Numner: PG4

Property Address: 6 Lawrence St, Glasgow

Type: Flat

Rent: 350

Client No	Name	Date	Comments
CR76	John Kay	20/04/13	Too remote.
CR56	Aline Stewart	26/05/13	
CR74	Mike Ritchie	11/11/13	
CR62	Mary Tregear	11/11/13	OK, but needs redecoration throughout.

Page 1

Figure 11.6 The first page of the *DreamHome* property viewing report for *a* property in Glasgow.

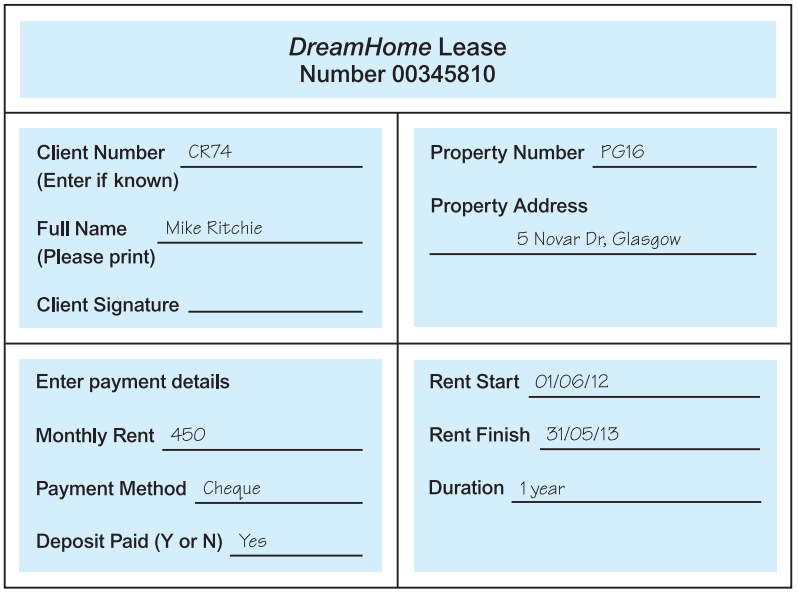


Figure 11.7 The *DreamHome* lease form for a client called Mike Ritchie renting a property in Glasgow.

Once a client has identified a suitable property, a member of staff draws up a lease. The lease between a client called Mike Ritchie and a property in Glasgow is shown in Figure 11.7.

At the end of a rental period a client may request that the rental be continued; however, this requires that a new lease be drawn up. Alternatively, a client may request to view alternative properties for the purposes of renting.

11.4.2 The *DreamHome* Case Study—Database Planning

The first step in developing a database system is to clearly define the **mission statement** for the database project, which defines the major aims of the database system. Once the mission statement is defined, the next activity involves identifying the **mission objectives**, which should identify the particular tasks that the database must support (see Section 10.3).

Creating the mission statement for the *DreamHome* database system

We begin the process of creating a mission statement for the *DreamHome* database system by conducting interviews with the Director and any other appropriate staff, as indicated by the Director. Open-ended questions are normally the most useful at this stage of the process. Examples of typical questions we might ask include:

"What is the purpose of your company?"
"Why do you feel that you need a database?"
"How do you know that a database will solve your problem?"

For example, the database developer may start the interview by asking the Director of *DreamHome* the following questions:

Database Developer: *What is the purpose of your company?*

Director: We offer a wide range of high-quality properties for rent to clients registered at our branches throughout the UK. Our ability to offer quality properties, of course, depends upon the services we provide to property owners. We provide a highly professional service to property owners to ensure that properties are rented out for maximum return.

Database Developer: *Why do you feel that you need a database?*

Director: To be honest, we can't cope with our own success. Over the past few years we've opened several branches in most of the main cities of the UK, and at each branch we now offer a larger selection of properties to a growing number of clients. However, this success has been accompanied with increasing data management problems, which means that the level of service we provide is falling. Also, there's a lack of cooperation and sharing of information between branches, which is a very worrying development.

Database Developer: *How do you know that a database will solve your problem?*

Director: All I know is that we are drowning in paperwork. We need something that will speed up the way we work by automating a lot of the day-to-day tasks that seem to take forever these days. Also, I want the branches to start working together. Databases will help to achieve this, won't they?

Responses to these types of questions should help formulate the mission statement. An example mission statement for the *DreamHome* database system is shown in Figure 11.8. When we have a clear and unambiguous mission statement that the staff of *DreamHome* agree with, we move on to define the mission objectives.

Creating the mission objectives for the *DreamHome* database system

The process of creating mission objectives involves conducting interviews with appropriate members of staff. Again, open-ended questions are normally the most useful at this stage of the process. To obtain the complete range of mission

> "The purpose of the *DreamHome* database system is to maintain the data that is used and generated to support the property rentals business for our clients and property owners and to facilitate the cooperation and sharing of information between branches."

Figure 11.8 Mission statement for the *DreamHome* database system.

objectives, we interview various members of staff with different roles in *DreamHome*. Examples of typical questions that we might ask include:

"What is your job description?"
"What kinds of tasks do you perform in a typical day?"
"What kinds of data do you work with?"
"What types of reports do you use?"
"What types of things do you need to keep track of?"
"What service does your company provide to your customers?"

These questions (or similar) are put to the Director of *DreamHome* and members of staff in the role of Manager, Supervisor, and Assistant. It may be necessary to adapt the questions as required, depending on whom is being interviewed.

Director

Database Developer: *What role do you play for the company?*

Director: I oversee the running of the company to ensure that we continue to provide the best possible property rental service to our clients and property owners.

Database Developer: *What kinds of tasks do you perform in a typical day?*

Director: I monitor the running of each branch by our Managers. I try to ensure that the branches work well together and share important information about properties and clients. I normally try to keep a high profile with my branch Managers by calling into each branch at least once or twice a month.

Database Developer: *What kinds of data do you work with?*

Director: I need to see everything, well at least a summary of the data used or generated by *DreamHome*. That includes data about staff at all branches, all properties and their owners, all clients, and all leases. I also like to keep an eye on the extent to which branches advertise properties in newspapers.

Database Developer: *What types of reports do you use?*

Director: I need to know what's going on at all the branches and there are lots of them. I spend a lot of my working day going over long reports on all aspects of *DreamHome*. I need reports that are easy to access and that let me get a good overview of what's happening at a given branch and across all branches.

Database Developer: *What types of things do you need to keep track of?*

Director: As I said before, I need to have an overview of everything; I need to see the whole picture.

Database Developer: *What service does your company provide to your customers?*

Director: We aim to provide the best property rental service in the UK. I believe that this will be achieved with the support of the new database system, which will allow

my staff to deal more efficiently with our customers and clients and better marketing of our properties through the development of a new *DreamHome* Web site. This site will allow our current and new renting clients to view our properties on the Web.

Manager

Database Developer: *What is your job description?*

Manager: My job title is Manager. I oversee the day-to-day running of my branch to provide the best property rental service to our clients and property owners.

Database Developer: *What kinds of tasks do you perform in a typical day?*

Manager: I ensure that the branch has the appropriate number and type of staff on duty at all times. I monitor the registering of new properties and new clients, and the renting activity of our currently active clients. It's my responsibility to ensure that we have the right number and type of properties available to offer our clients. I sometimes get involved in negotiating leases for our top-of-the-range properties, although due to my workload, I often have to delegate this task to Supervisors.

Database Developer: *What kinds of data do you work with?*

Manager: I mostly work with data on the properties offered at my branch and the owners, clients, and leases. I also need to know when properties are proving difficult to rent out so that I can arrange for them to be advertised in newspapers. I need to keep an eye on this aspect of the business, because advertising can get costly. I also need access to data about staff working at my branch and staff at other local branches. This is because I sometimes need to contact other branches to arrange management meetings or to borrow staff from other branches on a temporary basis to cover staff shortages due to sickness or during holiday periods. This borrowing of staff between local branches is informal and thankfully doesn't happen very often. Besides data on staff, it would be helpful to see other types of data at the other branches such as data on property, property owners, clients, and leases, you know, to compare notes. Actually, I think the Director hopes that this database project is going to help promote cooperation and sharing of information between branches. However, some of the Managers I know are not going to be too keen on this, because they think we're in competition with each other. Part of the problem is that a percentage of a Manager's salary is made up of a bonus, which is related to the number of properties we rent out.

Database Developer: *What types of reports do you use?*

Manager: I need various reports on staff, property, owners, clients, and leases. I need to know at a glance which properties we need to lease out and what clients are looking for.

Database Developer: *What types of things do you need to keep track of?*

Manager: I need to keep track of staff salaries. I need to know how well the properties on our books are being rented out and when leases are coming up for renewal. I also need to keep eye on our expenditure on advertising in newspapers.

Database Developer: *What service does your company provide to your customers?*

Manager: Remember that we have two types of customers; that is, clients wanting to rent property and property owners. We need to make sure that our clients find the property they're looking for quickly without too much legwork and at a reasonable rent, and, of course, that our property owners see good returns from renting out their properties with minimal hassle. As you may already know from speaking to our Director, as well as from developing a new database system, we also intend to develop a new *DreamHome* Web site. This Web site will help our clients view our properties at home before coming into our branches to arrange a viewing. I need to ensure that no matter how clients contacts us—either by email through using our Web site, by phone, or in person—that they receive the same efficient service to help them find the properties that they seek.

Supervisor

Database Developer: *What is your job description?*

Supervisor: My job title is Supervisor. I spend most of my time in the office dealing directly with our customers; that is, clients wanting to rent property and property owners. I'm also responsible for a small group of staff called Assistants and making sure that they are kept busy, but that's not a problem, as there's always plenty to do—it's never-ending actually.

Database Developer: *What kinds of tasks do you perform in a typical day?*

Supervisor: I normally start the day by allocating staff to particular duties, such as dealing with clients or property owners, organizing for clients to view properties, and filing paperwork. When a client finds a suitable property, I process the drawing up of a lease, although the Manager must see the documentation before any signatures are requested. I keep client details up to date and register new clients when they want to join the Company. When a new property is registered, the

Manager allocates responsibility for managing that property to me or one of the other Supervisors or Assistants.

Database Developer: *What kinds of data do you work with?*

Supervisor: I work with data about staff at my branch, property, property owners, clients, property viewings, and leases.

Database Developer: *What types of reports do you use?*

Supervisor: Reports on staff and properties for rent.

Database Developer: *What types of things do you need to keep track of?*

Supervisor: I need to know what properties are available for rent and when currently active leases are due to expire. I also need to know what clients are looking for. I need to keep our Manager up to date with any properties that are proving difficult to rent out. I need to ensure that clients who contact us by email requesting to view properties are given a quick response from us inviting them to call into their nearest *DreamHome* branch office. As part of the service we provide to our property owners, we need to interview all clients first before they are allowed to view our properties. There is nothing unusual about this, as we have always interviewed our clients on their first visit to a *DreamHome* branch, and it's during this time that we note their details and their property requirements.

Assistant

Database Developer: *What is your job description?*

Assistant: My job title is Assistant. I deal directly with our clients.

Database Developer: *What kinds of tasks do you perform in a typical day?*

Assistant: I answer general queries from clients about properties for rent. You know what I mean: "Do you have such and such type of property in a particular area of Glasgow?" I also register new clients and arrange for clients to view properties. When we're not too busy, I file paperwork, but I hate this part of the job—it's so boring.

Database Developer: *What kinds of data do you work with?*

Assistant: I work with data on property and property viewings by clients and sometimes leases.

Database Developer: *What types of reports do you use?*

Assistant: Lists of properties available for rent. These lists are updated every week.

Database Developer: *What types of things do you need to keep track of?*

Assistant: Whether certain properties are available for renting out and which clients are still actively looking for property.

To maintain (enter, update, and delete) data on branches.
To maintain (enter, update, and delete) data on staff.
To maintain (enter, update, and delete) data on properties for rent.
To maintain (enter, update, and delete) data on property owners.
To maintain (enter, update, and delete) data on clients.
To maintain (enter, update, and delete) data on property viewings.
To maintain (enter, update, and delete) data on leases.
To maintain (enter, update, and delete) data on newspaper adverts.

To perform searches on branches.
To perform searches on staff.
To perform searches on properties for rent.
To perform searches on property owners.
To perform searches on clients.
To perform searches on property viewings.
To perform searches on leases.
To perform searches on newspaper adverts.

To track the status of property for rent.
To track the status of clients wishing to rent.
To track the status of leases.

To report on branches.
To report on staff.
To report on properties for rent.
To report on property owners.
To report on clients.
To report on property viewings.
To report on leases.
To report on newspaper adverts.

Figure 11.9 Mission objectives for the *DreamHome* database system.

Database Developer: *What service does your company provide to your customers?*

Assistant: We try to answer questions about properties available for rent such as: "Do you have a two-bedroom flat in Hyndland, Glasgow?" and "What should I expect to pay for a one-bedroom flat in the city center?"

Responses to these types of questions should help to formulate the mission objectives. An example of the mission objectives for the *DreamHome* database system is shown in Figure 11.9.

11.4.3 The *DreamHome* Case Study—System Definition

The purpose of the system definition stage is to define the scope and boundary of the database system and its major user views. In Section 10.4.1 we described how a user view represents the requirements that should be supported by a database system as defined by a particular job role (such as Director or Supervisor) or business application area (such as property rentals or property sales).

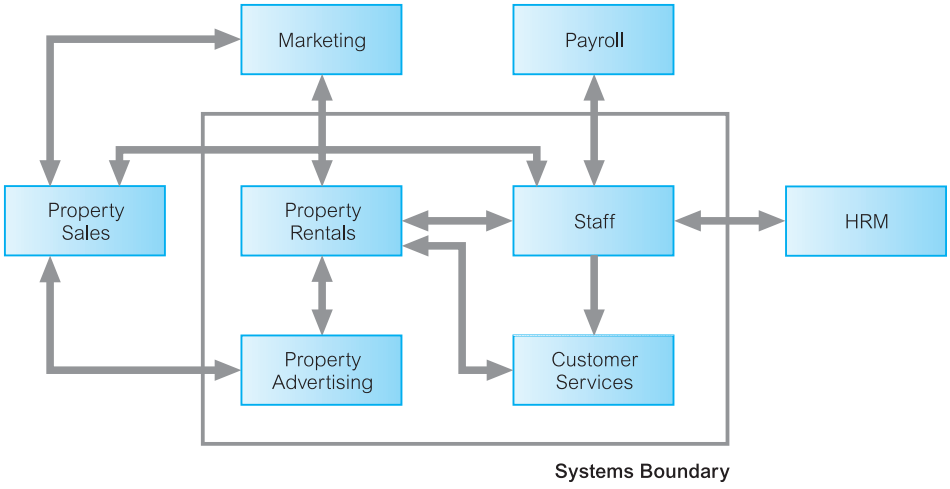


Figure 11.10 Systems boundary for the *DreamHome* database system.

Defining the systems boundary for the *DreamHome* database system

During this stage of the database system development lifecycle, further interviews with users can be used to clarify or expand on data captured in the previous stage. However, additional fact-finding techniques can also be used, including examining the sample documentation shown in Section 11.4.1. The data collected so far is analyzed to define the boundary of the database system. The systems boundary for the *DreamHome* database system is shown in Figure 11.10.

Identifying the major user views for the *DreamHome* database system

We now analyze the data collected so far to define the main user views of the database system. The majority of data about the user views was collected during interviews with the Director and members of staff in the role of Manager, Supervisor, and Assistant. The main user views for the *DreamHome* database system are shown in Figure 11.11.

11.4.4 The *DreamHome* Case Study—Requirements Collection and Analysis

During this stage, we continue to gather more details on the user views identified in the previous stage, to create a **users' requirements specification** that describes in detail the data to be held in the database and how the data is to be used. While gathering more information on the user views, we also collect any general requirements for the system. The purpose of gathering this information is to create a **systems specification**, which describes any features to be included in the new database system, such as networking and shared access requirements, performance requirements, and the levels of security required.

As we collect and analyze the requirements for the new system, we also learn about the most useful and most troublesome features of the current system. When

Data	Access Type	Director	Manager	Supervisor	Assistant	Client
All Branches	Maintain					
	Query	X	X			
	Report	X	X			
Single Branch	Maintain		X			
	Query		X			
	Report		X			
All Staff	Maintain					
	Query	X	X			
	Report	X	X			
Branch Staff	Maintain		X			
	Query		X	X		
	Report		X	X		
All Property	Maintain					
	Query	X				X
	Report	X	X			X
Branch Property	Maintain		X	X		
	Query		X	X	X	
	Report		X	X	X	
All Owners	Maintain					
	Query	X				
	Report	X	X			
Branch Owners	Maintain		X	X		
	Query		X	X	X	
	Report		X			
All Clients	Maintain					X
	Query	X				X
	Report	X	X			
Branch Clients	Maintain		X	X		
	Query		X	X	X	
	Report		X			
All Viewings	Maintain					
	Query					
	Report					
Branch Viewings	Maintain			X	X	
	Query			X	X	
	Report			X	X	
All Leases	Maintain					
	Query	X				
	Report	X	X			
Branch Leases	Maintain		X	X		
	Query		X	X	X	
	Report		X	X		
All Newspapers	Maintain					
	Query	X				
	Report	X	X			
Branch Newspapers	Maintain		X			
	Query		X			
	Report		X			

Figure 11.11 Major user views for the *DreamHome* database system.

building a new database system, it is sensible to try to retain the good things about the old system while introducing the benefits that will be part of using the new system.

An important activity associated with this stage is deciding how to deal with situations in which there are more than one user view. As we discussed in Section 10.6, there are three major approaches to dealing with multiple user views: the **centralized** approach, the **view integration** approach, and a combination of both approaches. We discuss how these approaches can be used shortly.

Gathering more information on the user views of the *DreamHome* database system

To find out more about the requirements for each user view, we may again use a selection of fact-finding techniques, including interviews and observing the business in operation. Examples of the types of questions that we may ask about the data (represented as X) required by a user view include:

"What type of data do you need to hold on X?"
"What sorts of things do you do with the data on X?"

For example, we might ask a Manager the following questions:

Database Developer:	*What type of data do you need to hold on staff?*
Manager:	The types of data held on a member of staff is his or her full name, position, gender, date of birth, and salary.
Database Developer:	*What sorts of things do you do with the data on staff?*
Manager:	I need to be able to enter the details of new members of staff and delete their details when they leave. I need to keep the details of staff up to date and print reports that list the full name, position, and salary of each member of staff at my branch. I need to be able to allocate staff to Supervisors. Sometimes when I need to communicate with other branches, I need to find out the names and telephone numbers of Managers at other branches.

We need to ask similar questions about all the important data to be stored in the database. Responses to these questions will help identify the necessary details for the users' requirements specification.

Gathering information on the system requirements of the *DreamHome* database system

While conducting interviews about user views, we should also collect more general information on the system requirements. Examples of the types of questions that we might ask about the system include:

"What transactions run frequently on the database?"
"What transactions are critical to the operation of the organization?"
"When do the critical transactions run?"

"When are the low, normal, and high workload periods for the critical transactions?"
"What type of security do you want for the database system?"
"Is there any highly sensitive data that should be accessed only by certain members of staff?"
"What historical data do you want to hold?"
"What are the networking and shared access requirements for the database system?"
"What type of protection from failures or data loss do you want for the database system?"

For example, we might ask a Manager the following questions:

Database Developer: *What transactions run frequently on the database?*

Manager: We frequently get requests either by phone or by clients who call into our branch to search for a particular type of property in a particular area of the city and for a rent no higher than a particular amount. We hope that clients using the new *DreamHome* Web site will be able to view our properties at any time of the day or night. We also need up-to-date information on properties and clients so that reports can be run off that show properties currently available for rent and clients currently seeking property.

Database Developer: *What transactions are critical to the operation of the business?*

Manager: Again, critical transactions include being able to search for particular properties and to print out reports with up-to-date lists of properties available for rent. Our clients would go elsewhere if we couldn't provide this basic service.

Database Developer: *When do the critical transactions run?*

Manager: Every day.

Database Developer: *When are the low, normal, and high workload periods for the critical transactions?*

Manager: We're open six days a week. In general, we tend to be quiet in the mornings and get busier as the day progresses. However, the busiest time-slots each day for dealing with customers are between 12 and 2pm and 5 and 7pm. We hope that clients using the new *DreamHome* Web site will be able to search through our properties on their own PCs; this should cut down on the number of property queries that staff have to deal with.

We might ask the Director the following questions:

Database Developer: *What type of security do you want for the database system?*

Director: I don't suppose a database holding information for a property rental company holds very sensitive data, but I wouldn't want any of our competitors to see the data

on properties, owners, clients, and leases. Staff should see only the data necessary to do their job in a form that suits what they're doing. For example, although it's necessary for Supervisors and Assistants to see client details, client records should be displayed only one at a time and not as a report. As far as clients using the new *DreamHome* Web site are concerned, we want them to have access to our properties and their own details—but nothing else.

Database Developer: *Is there any highly sensitive data that should be accessed only by certain members of staff?*

Director: As I said before, staff should see only the data necessary to do their jobs. For example, although Supervisors need to see data on staff, salary details should not be included.

Database Developer: *What historical data do you want to hold?*

Director: I want to keep the details of clients and owners for a couple of years after their last dealings with us, so that we can mail them our latest offers, and generally try to attract them back. I also want to be able to keep lease information for a couple of years, so that we can analyze it to find out which types of properties and areas of each city are the most popular for the property rental market, and so on.

Database Developer: *What are the networking and shared access requirements for the database system?*

Director: I want all the branches networked to our main branch office here in Glasgow so that staff can access the system from wherever and whenever they need to. At most branches, I would expect about two or three staff to be accessing the system at any one time, but remember that we have about 100 branches. Most of the time the staff should be just accessing local branch data. However, I don't really want there to be any restrictions about how often or when the system can be accessed, unless it's got real financial implications. As I said earlier, clients using the new *DreamHome* Web site should have access to our properties and their own details, but nothing else.

Database Developer: *What type of protection from failures or data loss do you want for the database system?*

Director: The best, of course. All our business is going to be conducted using the database, so if it goes down, we're sunk. To be serious for a minute, I think we probably have to back up our data every evening when the branch closes. What do you think?

We need to ask similar questions about all the important aspects of the system. Responses to these questions should help identify the necessary details for the system requirements specification.

Managing the user views of the *DreamHome* database system

How do we decide whether to use the centralized or view integration approach, or a combination of both to manage multiple user views? One way to help make a decision is to examine the overlap in the data used between the user views identified during the system definition stage. Table 11.7 cross-references the Director, Manager, Supervisor, Assistant, and Client user views with the main types of data used by each user view.

We see from Table 11.7 that there is overlap in the data used by all user views. However, the Director and Manager user views and the Supervisor and Assistant user views show more similarities in terms of data requirements. For example, only the Director and Manager user views require data on branches and newspapers, whereas only the Supervisor and Assistant user views require data on property viewings. The Client user view requires access to the least amount of data, and that is only the property and client data. Based on this analysis, we use the *centralized* approach to first merge the requirements for the Director and Manager user views (given the collective name of **Branch** user views) and the requirements for the Supervisor, Assistant, and Client user views (given the collective name of **StaffClient** user views). We then develop data models representing the Branch and StaffClient user views and then use the *view integration* approach to merge the two data models.

Of course, for a simple case study like *DreamHome*, we could easily use the centralized approach for all user views, but we will stay with our decision to create two collective user views so that we can describe and demonstrate how the view integration approach works in practice in Chapter 17.

It is difficult to give precise rules as to when it is appropriate to use the centralized or view integration approaches. The decision should be based on an assessment of the complexity of the database system and the degree of overlap between the various user views. However, whether we use the centralized or view integration approach or a mixture of both to build the underlying database, ultimately we need to re-establish

TABLE 11.7 Cross-reference of user views with the main types of data used by each.

	DIRECTOR	MANAGER	SUPERVISOR	ASSISTANT	CLIENT
branch	X	X			
staff	X	X	X		
property for rent	X	X	X	X	X
owner	X	X	X	X	
client	X	X	X	X	X
property viewing			X	X	
lease	X	X	X	X	
newspaper	X	X			

the original user views (namely Director, Manager, Supervisor, Assistant, and Client) for the working database system. We describe and demonstrate the establishment of the user views for the database system in Chapter 18.

All of the information gathered so far on each user view of the database system is described in a document called a **users' requirements specification**. The users' requirements specification describes the data requirements for each user view and examples of how the data is used by the user view. For ease of reference, the users' requirements specifications for the Branch and StaffClient user views of the *DreamHome* database system are given in Appendix A. In the remainder of this chapter, we present the general systems requirements for the *DreamHome* database system.

The systems specification for the *DreamHome* database system

The systems specification should list all the important features for the *DreamHome* database system. The types of features that should be described in the systems specification include:

- initial database size;
- database rate of growth;
- the types and average number of record searches;
- networking and shared access requirements;
- performance;
- security;
- backup and recovery;
- legal issues.

Systems Requirements for *DreamHome* Database System

Initial database size

(1) There are approximately 2000 members of staff working at over 100 branches. There is an average of 20 and a maximum of 40 members of staff at each branch.

(2) There are approximately 100,000 properties available at all branches. There is an average of 1000 and a maximum of 3000 properties at each branch.

(3) There are approximately 60,000 property owners. There is an average of 600 and a maximum of 1000 property owners at each branch.

(4) There are approximately 100,000 clients registered across all branches. There is an average of 1000 and a maximum of 1500 clients registered at each branch.

(5) There are approximately 4,000,000 viewings across all branches. There is an average of 40,000 and a maximum of 100,000 viewings at each branch.

(6) There are approximately 400,000 leases across all branches. There are an average of 4000 and a maximum of 10,000 leases at each branch.

(7) There are approximately 50,000 newspaper ads in 100 newspapers across all branches.

Database rate of growth

(1) Approximately 500 new properties and 200 new property owners will be added to the database each month.

(2) Once a property is no longer available for rent, the corresponding record will be deleted from the database. Approximately 100 records of properties will be deleted each month.

(3) If a property owner does not provide properties for rent at any time within a period of two years, his or her record will be deleted. Approximately 100 property owner records will be deleted each month.

(4) Approximately 20 members of staff join and leave the company each month. The records of staff who have left the company will be deleted after one year. Approximately 20 staff records will be deleted each month.

(5) Approximately 1000 new clients register at branches each month. If a client does not view or rent out a property at any time within a period of two years, his or her record will be deleted. Approximately 100 client records will be deleted each month.

(6) Approximately 5000 new viewings are recorded across all branches each day. The details of property viewings will be deleted one year after the creation of the record.

(7) Approximately 1000 new leases will be recorded across all branches each month. The details of property leases will be deleted two years after the creation of the record.

(8) Approximately 1000 newspaper adverts are placed each week. The details of newspaper adverts will be deleted one year after the creation of the record.

The types and average number of record searches

(1) Searching for the details of a branch—approximately 10 per day.

(2) Searching for the details of a member of staff at a branch—approximately 20 per day.

(3) Searching for the details of a given property—approximately 5000 per day (Monday to Thursday), and approximately 10,000 per day (Friday and Saturday). Peak workloads are 12.00–14.00 and 17.00–19.00 daily. (The workloads for property searches should be reassessed after the *DreamHome* Web site is launched.)

(4) Searching for the details of a property owner—approximately 100 per day.

(5) Searching for the details of a client—approximately 1000 per day (Monday to Thursday), and approximately 2000 per day (Friday and Saturday). Peak workloads are 12.00–14.00 and 17.00–19.00 daily.

(6) Searching for the details of a property viewing—approximately 2000 per day (Monday to Thursday), and approximately 5000 per day (Friday and Saturday). Peak workloads are 12.00–14.00 and 17.00–19.00 daily.

(7) Searching for the details of a lease—approximately 1000 per day (Monday to Thursday), and approximately 2000 per day (Friday and Saturday). Peak workloads are 12.00–14.00 and 17.00–19.00 daily.

Networking and shared access requirements

All branches should be securely networked to a centralized database located at *DreamHome's* main office in Glasgow. The system should allow for at least two to

three people concurrently accessing the system from each branch. Consideration needs to be given to the licensing requirements for this number of concurrent accesses.

Performance

(1) During opening hours, but not during peak periods, expect less than a 1-second response for all single record searches. During peak periods, expect less than a 5-second response for each search.

(2) During opening hours, but not during peak periods, expect less than a 5-second response for each multiple record search. During peak periods, expect less than a 10-second response for each multiple record search.

(3) During opening hours, but not during peak periods, expect less than a 1-second response for each update/save. During peak periods, expect less than a 5-second response for each update/save.

Security

(1) The database should be password-protected.

(2) Each member of staff should be assigned database access privileges appropriate to a particular user view, namely Director, Manager, Supervisor, or Assistant.

(3) A member of staff should see only the data necessary to do his or her job in a form that suits what he or she is doing.

(4) A client should see only property data and their own personal details using the *DreamHome* Web site.

Backup and Recovery

The database should be backed up daily at 12 midnight.

Legal Issues

Each country has laws that govern the way that the computerized storage of personal data is handled. As the *DreamHome* database holds data on staff, clients, and property owners, any legal issues that must be complied with should be investigated and implemented. The professional, legal, and ethical issues associated with data management are discussed in Chapter 21.

11.4.5 The *DreamHome* Case Study—Database Design

In this chapter we demonstrated the creation of the users' requirements specification for the Branch and Staff user views and the systems specification for the *DreamHome* database system. These documents are the sources of information for the next stage of the lifecycle called **database design**. In Chapters 16 to 19 we provide a step-by-step methodology for database design and use the *DreamHome* case study and the documents created for the *DreamHome* database system in this chapter to demonstrate the methodology in practice.

Chapter Summary

- **Fact-finding** is the formal process of using techniques such as interviews and questionnaires to collect facts about systems, requirements, and preferences.
- Fact-finding is particularly crucial to the early stages of the database system development lifecycle, including the database planning, system definition, and requirements collection and analysis stages.
- The five most common fact-finding techniques are examining documentation, interviewing, observing the enterprise in operation, conducting research, and using questionnaires.
- There are two main documents created during the requirements collection and analysis stage: the **users' requirements specification** and the **systems specification**.
- The **users' requirements specification** describes in detail the data to be held in the database and how the data is to be used.
- The **systems specification** describes any features to be included in the database system, such as the performance and security requirements.

Review Questions

11.1 Briefly discuss the objectives of interviews in a database development project.

11.2 Describe how fact-finding is used throughout the stages of the database system development lifecycle.

11.3 For each stage of the database system development lifecycle identify examples of the facts captured and the documentation produced.

11.4 A database developer normally uses several fact-finding techniques during a single database project. The five most commonly used techniques are examining documentation, interviewing, observing the business in operation, conducting research, and using questionnaires. Describe each fact-finding technique and identify the advantages and disadvantages of each.

11.5 What are the dangers of not defining mission objectives and a mission statement for a database system?

11.6 What is the purpose of identifying the systems boundary for a database system?

11.7 How do the contents of a users' requirements specification differ from a systems specification?

11.8 Database development and fact finding are inseparable. One factor for the success of the development processes if user involvement. What is the role played by users in the database development process?

Exercises

11.9 Assume that your friend is currently employed by a multinational consultancy company that deals with database analysis and development in Tanzania. His first assignment is to carry out a fact-finding mission for a client that intends to develop a database system for their company to control their daily business transactions. Your friend has decided to ask you for advice.

Task: Prepare a brief note that would help your friend successfully choose the best fact-finding technique. For each technique, outline the factors crucial for realizing quality facts. The note should also detail issues to be avoided or taken care of for each tool to succeed.

The client for this exercise and those that follow can be any one of the fictitious case studies given in Appendix B or some real company identified by you or your professor.

11.10 Assume that you are an employee of a consultancy company that specializes in the analysis, design, and implementation of database systems. A client has recently approached your company with a view to implementing a database system.

Task: You are required to establish the database project through the early stages of the project. With this task in mind, create a mission statement and mission objectives and high-level systems diagram for the client's database system.

11.11 Assume that you are an employee of a consultancy company that specializes in the analysis, design, and implementation of database systems. A client has recently approached your company with a view to implementing a database system. It has already been established that the client's database system will support many different groups of users (user views).

Task: You are required to identify how to best manage the requirements for these user views. With this task in mind, create a report that identifies high-level requirements for each user view and shows the relationship between the user views. Conclude the report by identifying and justifying the best approach to managing the multi-user view requirements.

CHAPTER

12 Entity–Relationship Modeling

Chapter Objectives

In this chapter you will learn:

- How to use Entity–Relationship (ER) modeling in database design.
- The basic concepts associated with the ER model: entities, relationships, and attributes.
- A diagrammatic technique for displaying an ER model using the Unified Modeling Language (UML).
- How to identify and resolve problems with ER models called connection traps.

In Chapter 11 we described the main techniques for gathering and capturing information about what the users require of a database system. Once the requirements collection and analysis stage of the database system development lifecycle is complete and we have documented the requirements for the database system, we are ready to begin the database design stage.

One of the most difficult aspects of database design is the fact that designers, programmers, and end-users tend to view data and its use in different ways. Unfortunately, unless we gain a common understanding that reflects how the enterprise operates, the design we produce will fail to meet the users' requirements. To ensure that we get a precise understanding of the nature of the data and how it is used by the enterprise, we need a model for communication that is nontechnical and free of ambiguities. The Entity–Relationship (ER) model is one such example. ER modeling is a top-down approach to database design that begins by identifying the important data called *entities* and *relationships* between the data that must be represented in the model. We then add more details, such as the information we want to hold about the entities and relationships called *attributes* and any *constraints* on the entities, relationships, and attributes. ER modeling is an important technique for any database designer to master and forms the basis of the methodology presented in this book.

In this chapter we introduce the basic concepts of the ER model. Although there is general agreement about what each concept means, there are a number of different notations that can be used to represent each concept diagrammatically. We have chosen a diagrammatic notation that uses an increasingly popular object-oriented modeling language called the **Unified Modeling Language** (**UML**) (Booch *et al.*,

1999). UML is the successor to a number of object-oriented analysis and design methods introduced in the 1980s and 1990s. The Object Management Group (OMG) is responsible for the creation and management of UML (available at www.uml.org). UML is currently recognized as the de facto industry standard modeling language for object-oriented software engineering projects. Although we use the UML notation for drawing ER models, we continue to describe the concepts of ER models using traditional database terminology. In Section 27.8 we will provide a fuller discussion on UML. We also include a summary of two alternative diagrammatic notations for ER models in Appendix C.

In the next chapter we discuss the inherent problems associated with representing complex database applications using the basic concepts of the ER model. To overcome these problems, additional "semantic" concepts were added to the original ER model, resulting in the development of the Enhanced Entity–Relationship (EER) model. In Chapter 13 we describe the main concepts associated with the EER model, called specialization/generalization, aggregation, and composition. We also demonstrate how to convert the ER model shown in Figure 12.1 into the EER model shown in Figure 13.8.

Structure of this Chapter In Sections 12.1, 12.2, and 12.3 we introduce the basic concepts of the Entity–Relationship model: entities, relationships, and attributes. In each section we illustrate how the basic ER concepts are represented pictorially in an ER diagram using UML. In Section 12.4 we differentiate between weak and strong entities and in Section 12.5 we discuss how attributes normally associated with entities can be assigned to relationships. In Section 12.6 we describe the structural constraints associated with relationships. Finally, in Section 12.7 we identify potential problems associated with the design of an ER model called connection traps and demonstrate how these problems can be resolved.

The ER diagram shown in Figure 12.1 is an example of one of the possible end-products of ER modeling. This model represents the relationships between data described in the requirements specification for the Branch view of the *DreamHome* case study given in Appendix A. This figure is presented at the start of this chapter to show the reader an example of the type of model that we can build using ER modeling. At this stage, the reader should not be concerned about fully understanding this diagram, as the concepts and notation used in this figure are discussed in detail throughout this chapter.

12.1 Entity Types

Entity type A group of objects with the same properties, which are identified by the enterprise as having an independent existence.

The basic concept of the ER model is the **entity type**, which represents a group of "objects" in the "real world" with the same properties. An entity type has an

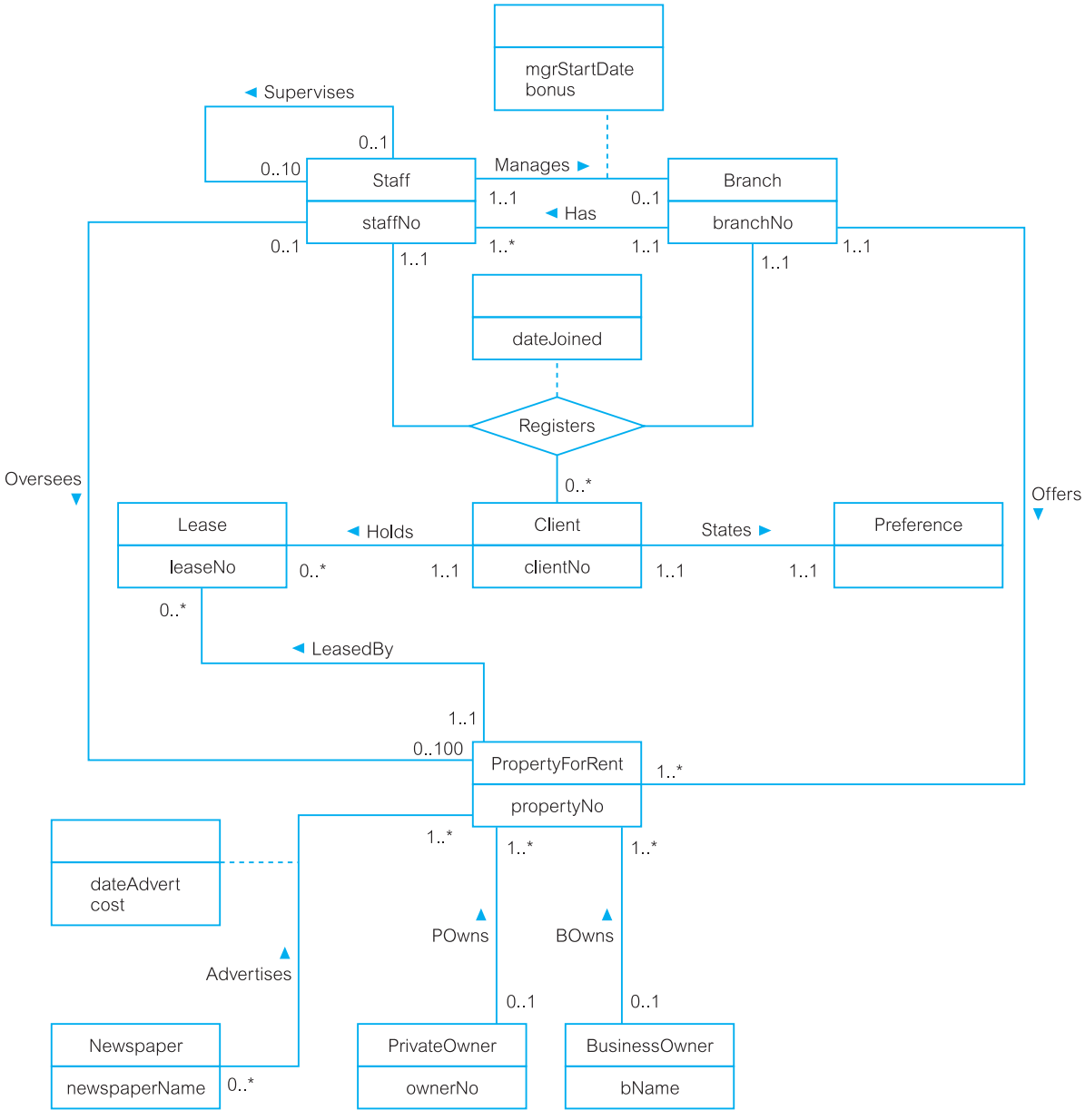


Figure 12.1 An Entity–Relationship (ER) diagram of the Branch view of *DreamHome*.

independent existence and can be objects with a physical (or "real") existence or objects with a conceptual (or "abstract") existence, as listed in Figure 12.2. Note that we are able to give only a working definition of an entity type, as no strict formal definition exists. This means that different designers may identify different entities.

Entity occurrence A uniquely identifiable object of an entity type.

Figure 12.2 Example of entities with a physical or conceptual existence.

Physical existence	
Staff	Part
Property	Supplier
Customer	Product
Conceptual existence	
Viewing	Sale
Inspection	Work experience

Each uniquely identifiable object of an entity type is referred to simply as an **entity occurrence**. Throughout this book, we use the terms "entity type" or "entity occurrence"; however, we use the more general term "entity" where the meaning is obvious.

We identify each entity type by a name and a list of properties. A database normally contains many different entity types. Examples of entity types shown in Figure 12.1 include: Staff, Branch, PropertyForRent, and PrivateOwner.

Diagrammatic representation of entity types

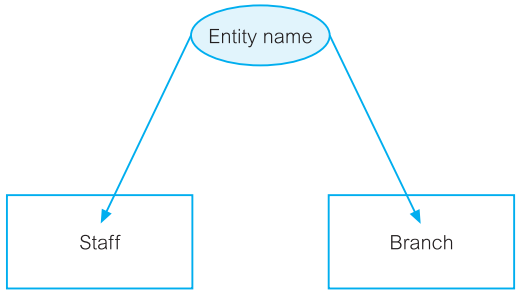
Each entity type is shown as a rectangle, labeled with the name of the entity, which is normally a singular noun. In UML, the first letter of each word in the entity name is uppercase (for example, Staff and PropertyForRent). Figure 12.3 illustrates the diagrammatic representation of the Staff and Branch entity types.

12.2 Relationship Types

Relationship type A set of meaningful associations among entity types.

A **relationship type** is a set of associations between one or more participating entity types. Each relationship type is given a name that describes its function. An example of a relationship type shown in Figure 12.1 is the relationship called *POwns,* which associates the PrivateOwner and PropertyForRent entities.

Figure 12.3 Diagrammatic representation of the Staff and Branch entity types.



As with entity types and entities, it is necessary to distinguish between the terms "relationship type" and "relationship occurrence."

Relationship occurrence	A uniquely identifiable association that includes one occurrence from each participating entity type.

A relationship occurrence indicates the particular entity occurrences that are related. Throughout this book, we use the terms "relationship type" or "relationship occurrence." However, as with the term "entity," we use the more general term "relationship" when the meaning is obvious.

Consider a relationship type called *Has*, which represents an association between Branch and Staff entities, that is Branch *Has* Staff. Each occurrence of the *Has* relationship associates one Branch entity occurrence with one Staff entity occurrence. We can examine examples of individual occurrences of the *Has* relationship using a *semantic net*. A semantic net is an object-level model, which uses the symbol • to represent entities and the symbol ◈ to represent relationships. The semantic net in Figure 12.4 shows three examples of the *Has* relationships (denoted rl, r2, and r3). Each relationship describes an association of a single Branch entity occurrence with a single Staff entity occurrence. Relationships are represented by lines that join each participating Branch entity with the associated Staff entity. For example, relationship rl represents the association between Branch entity B003 and Staff entity SG37.

Note that we represent each Branch and Staff entity occurrences using values for the primary key attributes, namely branchNo and staffNo. Primary key attributes uniquely identify each entity occurrence and are discussed in detail in the following section.

If we represented an enterprise using semantic nets, it would be difficult to understand, due to the level of detail. We can more easily represent the relationships between entities in an enterprise using the concepts of the ER model. The ER model uses a higher level of abstraction than the semantic net by combining sets of entity occurrences into entity types and sets of relationship occurrences into relationship types.

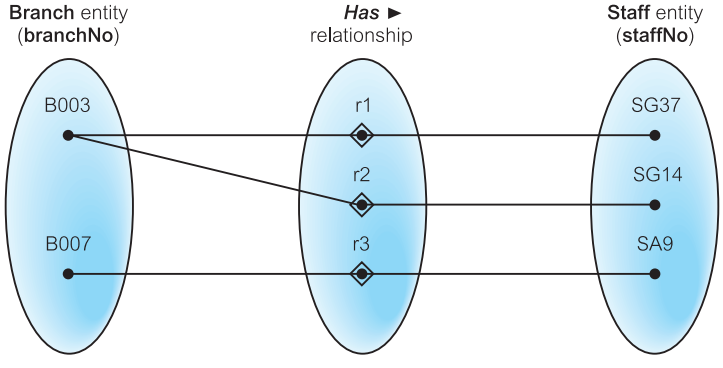


Figure 12.4 A semantic net showing individual occurrences of the *Has* relationship type.

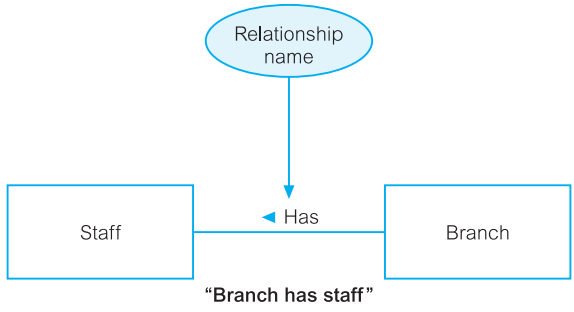


Figure 12.5
A diagrammatic representation of Branch *Has* Staff relationship type.

Diagrammatic representation of relationship types

Each relationship type is shown as a line connecting the associated entity types and labeled with the name of the relationship. Normally, a relationship is named using a verb (for example, *Supervises* or *Manages*) or a short phrase including a verb (for example, *LeasedBy*). Again, the first letter of each word in the relationship name is shown in uppercase. Whenever possible, a relationship name should be unique for a given ER model.

A relationship is only labeled in one direction, which normally means that the name of the relationship only makes sense in one direction (for example, Branch *Has* Staff makes more sense than Staff *Has* Branch). So once the relationship name is chosen, an arrow symbol is placed beside the name indicating the correct direction for a reader to interpret the relationship name (for example, Branch Has ▶ Staff) as shown in Figure 12.5.

12.2.1 Degree of Relationship Type

Degree of a relationship type — The number of participating entity types in a relationship.

The entities involved in a particular relationship type are referred to as **participants** in that relationship. The number of participants in a relationship type is called the **degree** of that relationship. Therefore, the degree of a relationship indicates the number of entity types involved in a relationship. A relationship of degree two is called **binary**. An example of a binary relationship is the *Has* relationship shown in Figure 12.5 with two participating entity types; namely, Staff and Branch. A second example of a binary relationship is the *POwns* relationship shown in Figure 12.6 with two participating entity types; namely, PrivateOwner and PropertyForRent. The *Has* and *POwns* relationships are also shown in Figure 12.1 as well as other examples of

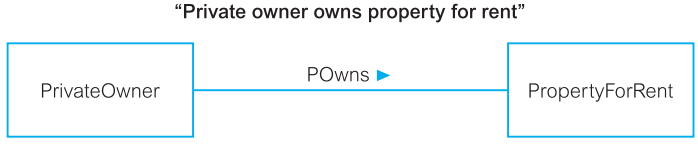


Figure 12.6
An example of a binary relationship called *POwns*.

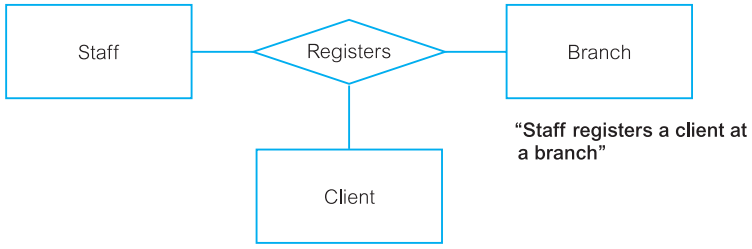


Figure 12.7 An example of a ternary relationship called *Registers*.

binary relationships. In fact, the most common degree for a relationship is binary, as demonstrated in this figure.

A relationship of degree three is called **ternary**. An example of a ternary relationship is *Registers* with three participating entity types: Staff, Branch, and Client. This relationship represents the registration of a client by a member of staff at a branch. The term "complex relationship" is used to describe relationships with degrees higher than binary.

Diagrammatic representation of complex relationships

The UML notation uses a diamond to represent relationships with degrees higher than binary. The name of the relationship is displayed inside the diamond, and in this case, the directional arrow normally associated with the name is omitted. For example, the ternary relationship called *Registers* is shown in Figure 12.7. This relationship is also shown in Figure 12.1.

A relationship of degree four is called **quaternary**. As we do not have an example of such a relationship in Figure 12.1, we describe a quaternary relationship called *Arranges* with four participating entity types—namely, Buyer, Solicitor, FinancialInstitution, and Bid—in Figure 12.8. This relationship represents the situation where a buyer, advised by a solicitor and supported by a financial institution, places a bid.

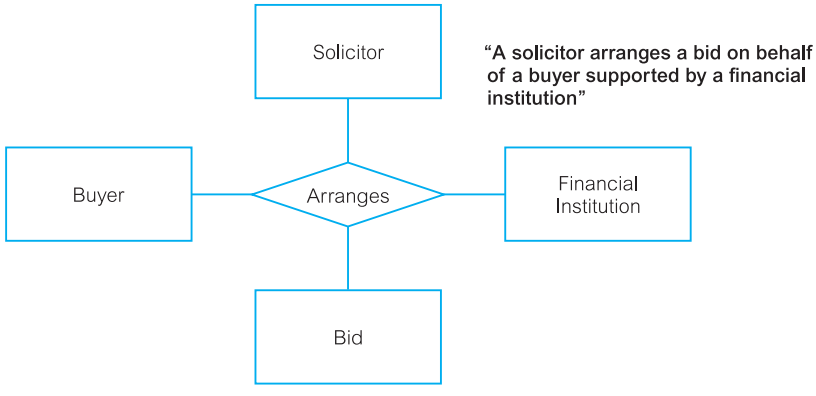


Figure 12.8 An example of a quaternary relationship called *Arranges*.

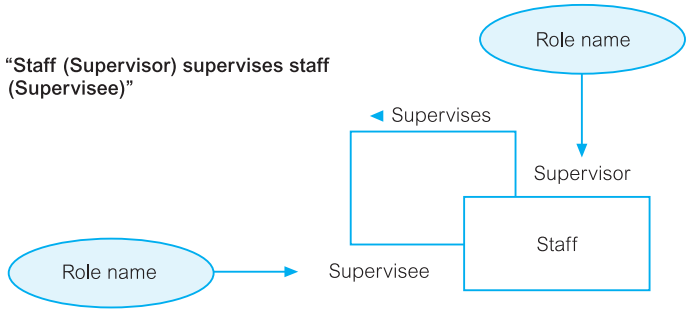


Figure 12.9
An example of a recursive relationship called *Supervises* with role names Supervisor and Supervisee.

12.2.2 Recursive Relationship

Recursive relationship — A relationship type in which the *same* entity type participates more than once in *different roles*.

Consider a recursive relationship called *Supervises*, which represents an association of staff with a Supervisor where the Supervisor is also a member of staff. In other words, the Staff entity type participates twice in the *Supervises* relationship; the first participation as a Supervisor, and the second participation as a member of staff who is supervised (Supervisee). Recursive relationships are sometimes called *unary* relationships.

Relationships may be given **role names** to indicate the purpose that each participating entity type plays in a relationship. Role names can be important for recursive relationships to determine the function of each participant. The use of role names to describe the *Supervises* recursive relationship is shown in Figure 12.9. The first participation of the Staff entity type in the *Supervises* relationship is given the role name "Supervisor" and the second participation is given the role name "Supervisee."

Role names may also be used when two entities are associated through more than one relationship. For example, the Staff and Branch entity types are associated through two distinct relationships called *Manages* and *Has*. As shown in Figure 12.10,

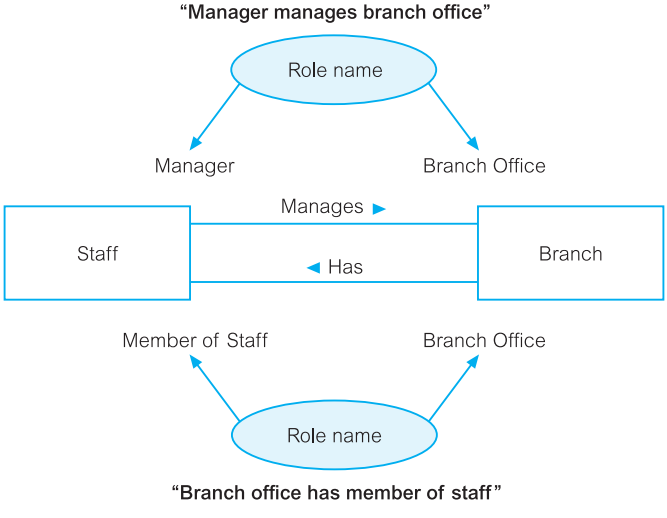


Figure 12.10
An example of entities associated through two distinct relationships called *Manages* and *Has* with role names.

the use of role names clarifies the purpose of each relationship. For example, in the case of Staff *Manages* Branch, a member of staff (Staff entity) given the role name "Manager" manages a branch (Branch entity) given the role name "Branch Office." Similarly, for Branch *Has* Staff, a branch, given the role name "Branch Office" has staff given the role name "Member of Staff."

Role names are usually not required if the function of the participating entities in a relationship is unambiguous.

12.3 Attributes

Attribute A property of an entity or a relationship type.

The particular properties of entity types are called attributes. For example, a Staff entity type may be described by the staffNo, name, position, and salary attributes. The attributes hold values that describe each entity occurrence and represent the main part of the data stored in the database.

A relationship type that associates entities can also have attributes similar to those of an entity type, but we defer discussion of this until Section 12.5. In this section, we concentrate on the general characteristics of attributes.

Attribute domain The set of allowable values for one or more attributes.

Each attribute is associated with a set of values called a **domain**. The domain defines the potential values that an attribute may hold and is similar to the domain concept in the relational model (see Section 4.2). For example, the number of rooms associated with a property is between 1 and 15 for each entity occurrence. We therefore define the set of values for the number of rooms (rooms) attribute of the PropertyForRent entity type as the set of integers between 1 and 15.

Attributes may share a domain. For example, the address attributes of the Branch, PrivateOwner, and BusinessOwner entity types share the same domain of all possible addresses. Domains can also be composed of domains. For example, the domain for the address attribute of the Branch entity is made up of subdomains: street, city, and postcode.

The domain of the name attribute is more difficult to define, as it consists of all possible names. It is certainly a character string, but it might consist not only of letters but also hyphens or other special characters. A fully developed data model includes the domains of each attribute in the ER model.

As we now explain, attributes can be classified as being: *simple* or *composite, single-valued* or *multi-valued,* or *derived.*

12.3.1 Simple and Composite Attributes

Simple attribute An attribute composed of a single component with an independent existence.

Simple attributes cannot be further subdivided into smaller components. Examples of simple attributes include position and salary of the Staff entity. Simple attributes are sometimes called *atomic* attributes.

Composite attribute An attribute composed of multiple components, each with an independent existence.

Some attributes can be further divided to yield smaller components with an independent existence of their own. For example, the address attribute of the Branch entity with the value (163 Main St, Glasgow, G11 9QX) can be subdivided into street (163 Main St), city (Glasgow), and postcode (G11 9QX) attributes.

The decision to model the address attribute as a simple attribute or to subdivide the attribute into street, city, and postcode is dependent on whether the user view of the data refers to the address attribute as a single unit or as individual components.

12.3.2 Single-valued and Multi-valued Attributes

Single-valued attribute An attribute that holds a single value for each occurrence of an entity type.

The majority of attributes are single-valued. For example, each occurrence of the Branch entity type has a single value for the branch number (branchNo) attribute (for example, B003), and therefore the branchNo attribute is referred to as being single-valued.

Multi-valued attribute An attribute that holds multiple values for each occurrence of an entity type.

Some attributes have multiple values for each entity occurrence. For example, each occurrence of the Branch entity type can have multiple values for the telNo attribute (for example, branch number B003 has telephone numbers 0141-339-2178 and 0141-339-4439) and therefore the telNo attribute in this case is multi-valued. A multi-valued attribute may have a set of numbers with upper and lower limits. For example, the telNo attribute of the Branch entity type has between one and three values. In other words, a branch may have a minimum of a single telephone number to a maximum of three telephone numbers.

12.3.3 Derived Attributes

Derived attribute An attribute that represents a value that is derivable from the value of a related attribute or set of attributes, not necessarily in the same entity type.

The values held by some attributes may be derived. For example, the value for the duration attribute of the Lease entity is calculated from the rentStart and rentFinish attributes, also of the Lease entity type. We refer to the duration attribute as a derived attribute, the value of which is derived from the rentStart and rentFinish attributes.

In some cases, the value of an attribute is derived from the entity occurrences in the same entity type. For example, the total number of staff (totalStaff) attribute of the Staff entity type can be calculated by counting the total number of Staff entity occurrences.

Derived attributes may also involve the association of attributes of different entity types. For example, consider an attribute called deposit of the Lease entity type. The value of the deposit attribute is calculated as twice the monthly rent for a property. Therefore, the value of the deposit attribute of the Lease entity type is derived from the rent attribute of the PropertyForRent entity type.

12.3.4 Keys

Candidate key — The minimal set of attributes that uniquely identifies each occurrence of an entity type.

A candidate key is the minimal number of attributes, whose value(s) uniquely identify each entity occurrence. For example, the branch number (branchNo) attribute is the candidate key for the Branch entity type, and has a distinct value for each branch entity occurrence. The candidate key must hold values that are unique for every occurrence of an entity type. This implies that a candidate key cannot contain a null (see Section 4.2). For example, each branch has a unique branch number (for example, B003), and there will never be more than one branch with the same branch number.

Primary key — The candidate key that is selected to uniquely identify each occurrence of an entity type.

An entity type may have more than one candidate key. For the purposes of discussion, consider that a member of staff has a unique company-defined staff number (staffNo) and also a unique National Insurance Number (NIN) that is used by the government. We therefore have two candidate keys for the Staff entity, one of which must be selected as the primary key.

The choice of primary key for an entity is based on considerations of attribute length, the minimal number of attributes required, and the future certainty of uniqueness. For example, the company-defined staff number contains a maximum of five characters (for example, SG14), and the NIN contains a maximum of nine characters (for example, WL220658D). Therefore, we select staffNo as the primary key of the Staff entity type and NIN is then referred to as the **alternate key**.

Composite key A candidate key that consists of two or more attributes.

In some cases, the key of an entity type is composed of several attributes whose values together are unique for each entity occurrence but not separately. For example, consider an entity called Advert with propertyNo (property number), newspaperName, dateAdvert, and cost attributes. Many properties are advertised in many newspapers on a given date. To uniquely identify each occurrence of the Advert entity type requires values for the propertyNo, newspaperName, and dateAdvert attributes. Thus, the Advert entity type has a composite primary key made up of the propertyNo, newspaperName, and dateAdvert attributes.

Diagrammatic representation of attributes

If an entity type is to be displayed with its attributes, we divide the rectangle representing the entity in two. The upper part of the rectangle displays the name of the entity and the lower part lists the names of the attributes. For example, Figure 12.11 shows the ER diagram for the Staff and Branch entity types and their associated attributes.

The first attribute(s) to be listed is the primary key for the entity type, if known. The name(s) of the primary key attribute(s) can be labeled with the tag {PK}. In UML, the name of an attribute is displayed with the first letter in lowercase and, if the name has more than one word, with the first letter of each subsequent word in uppercase (for example, address and telNo). Additional tags that can be used include partial primary key {PPK} when an attribute forms part of a composite primary key, and alternate key {AK}. As shown in Figure 12.11, the primary key of the Staff entity type is the staffNo attribute and the primary key of the Branch entity type is the branchNo attribute.

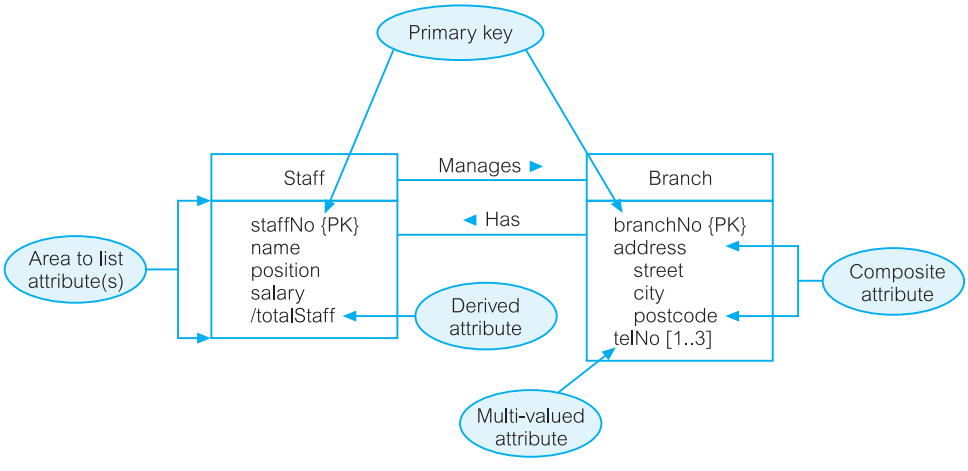


Figure 12.11 Diagrammatic representation of Staff and Branch entities and their attributes.

For some simpler database systems, it is possible to show all the attributes for each entity type in the ER diagram. However, for more complex database systems, we just display the attribute, or attributes, that form the primary key of each entity type. When only the primary key attributes are shown in the ER diagram, we can omit the {PK} tag.

For simple, single-valued attributes, there is no need to use tags, so we simply display the attribute names in a list below the entity name. For composite attributes, we list the name of the composite attribute followed below and indented to the right by the names of its simple component attributes. For example, in Figure 12.11 the composite attribute address of the Branch entity is shown, followed below by the names of its component attributes, street, city, and postcode. For multivalued attributes, we label the attribute name with an indication of the range of values available for the attribute. For example, if we label the telNo attribute with the range [1..*], this means that the values for the telNo attribute is one or more. If we know the precise maximum number of values, we can label the attribute with an exact range. For example, if the telNo attribute holds one to a maximum of three values, we can label the attribute with [1..3].

For derived attributes, we prefix the attribute name with a "/". For example, the derived attribute of the Staff entity type is shown in Figure 12.11 as /totalStaff.

12.4 Strong and Weak Entity Types

We can classify entity types as being strong or weak.

Strong entity type An entity type that is *not* existence-dependent on some other entity type.

An entity type is referred to as being strong if its existence does not depend upon the existence of another entity type. Examples of strong entities are shown in Figure 12.1 and include the Staff, Branch, PropertyForRent, and Client entities. A characteristic of a strong entity type is that each entity occurrence is uniquely identifiable using the primary key attribute(s) of that entity type. For example, we can uniquely identify each member of staff using the staffNo attribute, which is the primary key for the Staff entity type.

Weak entity type An entity type that is existence-dependent on some other entity type.

A weak entity type is dependent on the existence of another entity type. An example of a weak entity type called Preference is shown in Figure 12.12. A characteristic of a weak entity is that each entity occurrence cannot be uniquely identified using only the attributes associated with that entity type. For example, note that there is no primary key for the Preference entity. This means that we cannot identify each occurrence of the Preference entity type using only the attributes of this

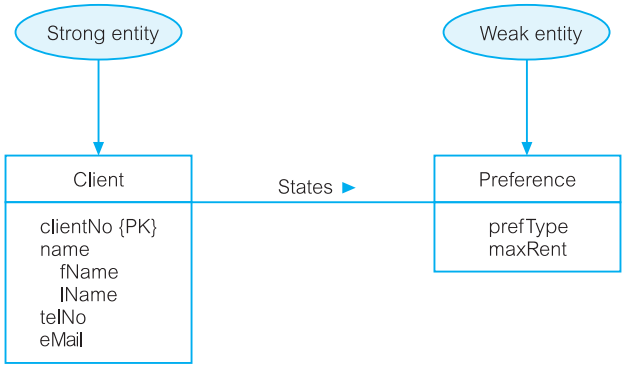


Figure 12.12 A strong entity type called Client and a weak entity type called Preference.

entity. We can uniquely identify each preference only through the relationship that a preference has with a client who is uniquely identifiable using the primary key for the Client entity type, namely clientNo. In this example, the Preference entity is described as having existence dependency for the Client entity, which is referred to as being the owner entity.

Weak entity types are sometimes referred to as *child, dependent,* or *subordinate* entities and strong entity types as *parent, owner,* or *dominant* entities.

12.5 Attributes on Relationships

As we mentioned in Section 12.3, attributes can also be assigned to relationships. For example, consider the relationship *Advertises*, which associates the Newspaper and PropertyForRent entity types as shown in Figure 12.1. To record the date the property was advertised and the cost, we associate this information with the *Advertises* relationship as attributes called dateAdvert and cost, rather than with the Newspaper or the PropertyForRent entities.

Diagrammatic representation of attributes on relationships

We represent attributes associated with a relationship type using the same symbol as an entity type. However, to distinguish between a relationship with an attribute and an entity, the rectangle representing the attribute(s) is associated with the relationship using a dashed line. For example, Figure 12.13 shows the *Advertises* relationship with the attributes dateAdvert and cost. A second example shown in Figure 12.1 is the *Manages* relationship with the mgrStartDate and bonus attributes.

The presence of one or more attributes assigned to a relationship may indicate that the relationship conceals an unidentified entity type. For example, the presence of the dateAdvert and cost attributes on the *Advertises* relationship indicates the presence of an entity called Advert.

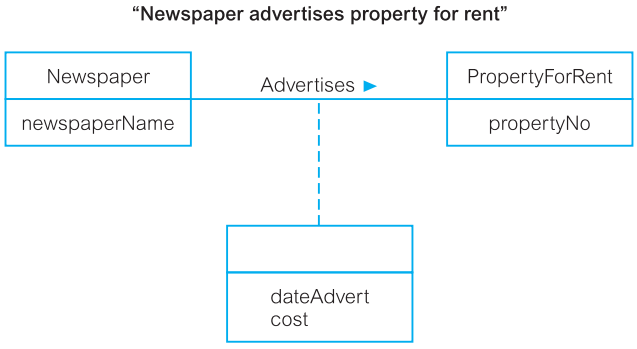


Figure 12.13 An example of a relationship called *Advertises* with attributes dateAdvert and cost.

12.6 Structural Constraints

We now examine the constraints that may be placed on entity types that participate in a relationship. The constraints should reflect the restrictions on the relationships as perceived in the "real world." Examples of such constraints include the requirements that a property for rent must have an owner and each branch must have staff. The main type of constraint on relationships is called **multiplicity**.

Multiplicity The number (or range) of possible occurrences of an entity type that may relate to a single occurrence of an associated entity type through a particular relationship.

Multiplicity constrains the way that entities are related. It is a representation of the policies (or business rules) established by the user or enterprise. Ensuring that all appropriate **constraints** are identified and represented is an important part of modeling an enterprise.

As we mentioned earlier, the most common degree for relationships is binary. Binary relationships are generally referred to as being one-to-one (1:1), one-to-many (1:*), or many-to-many (*:*). We examine these three types of relationships using the following integrity constraints:

- a member of staff manages a branch (1:1);
- a member of staff oversees properties for rent (1:*);
- newspapers advertise properties for rent (*:*).

In Sections 12.6.1, 12.6.2, and 12.6.3 we demonstrate how to determine the multiplicity for each of these constraints and show how to represent each in an ER diagram. In Section 12.6.4 we examine multiplicity for relationships of degrees higher than binary.

It is important to note that not all integrity constraints can be easily represented in an ER model. For example, the requirement that a member of staff receives an additional day's holiday for every year of employment with the enterprise may be difficult to represent in an ER model.

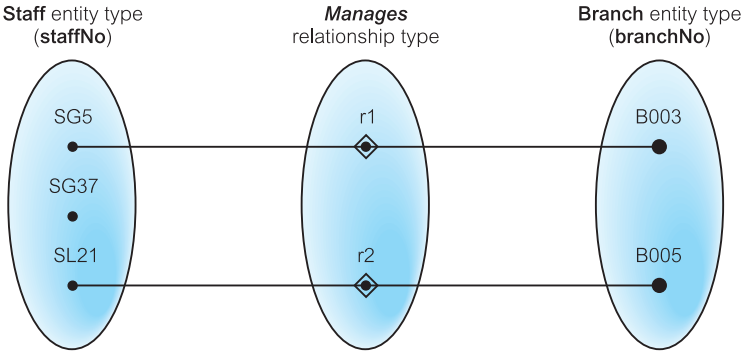


Figure 12.14(a) Semantic net showing two occurrences of the Staff *Manages* Branch relationship type.

12.6.1 One-to-One (1:1) Relationships

Consider the relationship *Manages*, which relates the Staff and Branch entity types. Figure 12.14(a) displays two occurrences of the *Manages* relationship type (denoted rl and r2) using a semantic net. Each relationship (r*n*) represents the association between a single Staff entity occurrence and a single Branch entity occurrence. We represent each entity occurrence using the values for the primary key attributes of the Staff and Branch entities, namely staffNo and branchNo.

Determining the multiplicity

Determining the multiplicity normally requires examining the precise relationships between the data given in a enterprise constraint using sample data. The sample data may be obtained by examining filled-in forms or reports and, if possible, from discussion with users. However, it is important to stress that to reach the right conclusions about a constraint requires that the sample data examined or discussed is a true representation of all the data being modeled.

In Figure 12.14(a) we see that staffNo SG5 manages branchNo B003 and staffNo SL21 manages branchNo BOO5, but staffNo SG37 does not manage any branch. In other words, a member of staff can manage *zero or one* branch and each branch is managed by *one* member of staff. As there is a maximum of one branch for each member of staff involved in this relationship and a maximum of one member of staff for each branch, we refer to this type of relationship as *one-to-one*, which we usually abbreviate as (1:1).

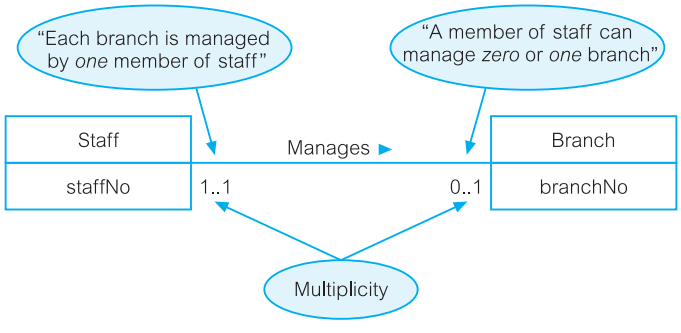


Figure 12.14(b) The multiplicity of the Staff *Manages* Branch one-to-one (1:1) relationship.

Diagrammatic representation of 1:1 relationships

An ER diagram of the Staff *Manages* Branch relationship is shown in Figure 12.14(b). To represent that a member of staff can manage *zero or one* branch, we place a "0..1" beside the Branch entity. To represent that a branch always has *one* manager, we place a "1..1" beside the Staff entity. (Note that for a 1:1 relationship, we may choose a relationship name that makes sense in either direction.)

12.6.2 One-to-Many (1:*) Relationships

Consider the relationship *Oversees*, which relates the Staff and PropertyForRent entity types. Figure 12.15(a) displays three occurrences of the Staff *Oversees* PropertyForRent relationship type (denoted rl, r2, and r3) using a semantic net. Each relationship (r*n*) represents the association between a single Staff entity occurrence and a single PropertyForRent entity occurrence. We represent each entity occurrence using the values for the primary key attributes of the Staff and PropertyForRent entities, namely staffNo and propertyNo.

Determining the multiplicity

In Figure 12.15(a) we see that staffNo SG37 oversees propertyNos PG21 and PG36, and staffNo SA9 oversees propertyNo PA14 but staffNo SG5 does not oversee any properties for rent and propertyNo PG4 is not overseen by any member of staff. In summary, a member of staff can oversee *zero or more* properties for rent and a property for rent is overseen by *zero or one* member of staff. Therefore, for members of staff participating in this relationship there are *many* properties for rent, and for properties participating in this relationship there is a maximum of *one* member of staff. We refer to this type of relationship as *one-to-many*, which we usually abbreviate as (1:*).

Diagrammatic representation of 1:* relationships An ER diagram of the Staff *Oversees* PropertyForRent relationship is shown in Figure 12.15(b). To represent that a member of staff can oversee *zero or more* properties for rent, we place a "0..*" beside the PropertyForRent entity. To represent that each property for rent is overseen by *zero or one* member of staff, we place a "0..1" beside the Staff entity. (Note that with 1:* relationships, we choose a relationship name that makes sense in the 1:* direction.)

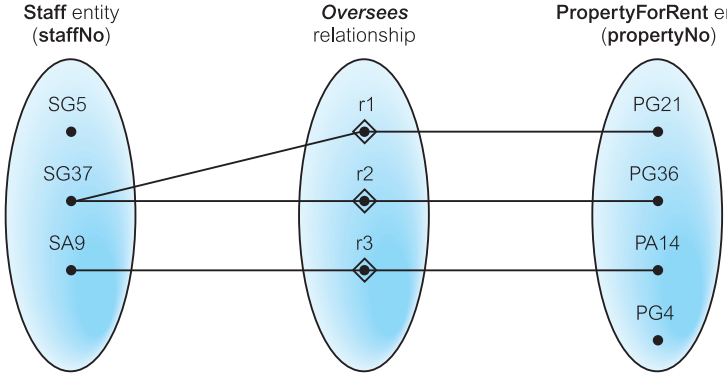


Figure 12.15(a) Semantic net showing three occurrences of the Staff *Oversees* PropertyForRent relationship type.

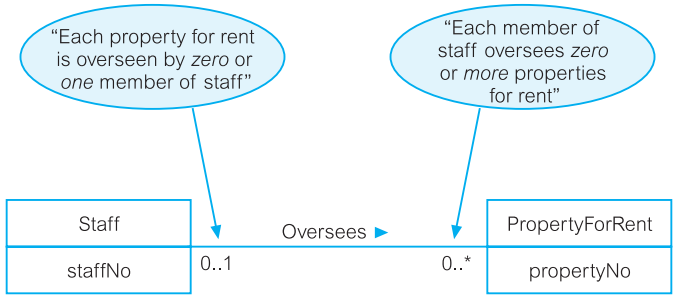


Figure 12.15(b) The multiplicity of the Staff *Oversees* PropertyForRent one-to-many (1:*) relationship type.

If we know the actual minimum and maximum values for the multiplicity, we can display these instead. For example, if a member of staff oversees a minimum of zero and a maximum of 100 properties for rent, we can replace the "0..*" with "0..100."

12.6.3 Many-to-Many (*:*) Relationships

Consider the relationship *Advertises*, which relates the Newspaper and PropertyForRent entity types. Figure 12.16(a) displays four occurrences of the *Advertises* relationship (denoted rl, r2, r3, and r4) using a semantic net. Each relationship (r*n*) represents the association between a single Newspaper entity occurrence and a single PropertyForRent entity occurrence. We represent each entity occurrence using the values for the primary key attributes of the Newspaper and PropertyForRent entity types, namely newspaperName and propertyNo.

Determining the multiplicity

In Figure 12.16(a) we see that the *Glasgow Daily* advertises propertyNos PG21 and PG36, *The West News* also advertises propertyNo PG36 and the *Aberdeen Express* advertises propertyNo PA14. However, propertyNo PG4 is not advertised in any newspaper. In other words, *one* newspaper advertises *one or more* properties for rent and *one* property for rent is advertised in *zero or more* newspapers. Therefore, for

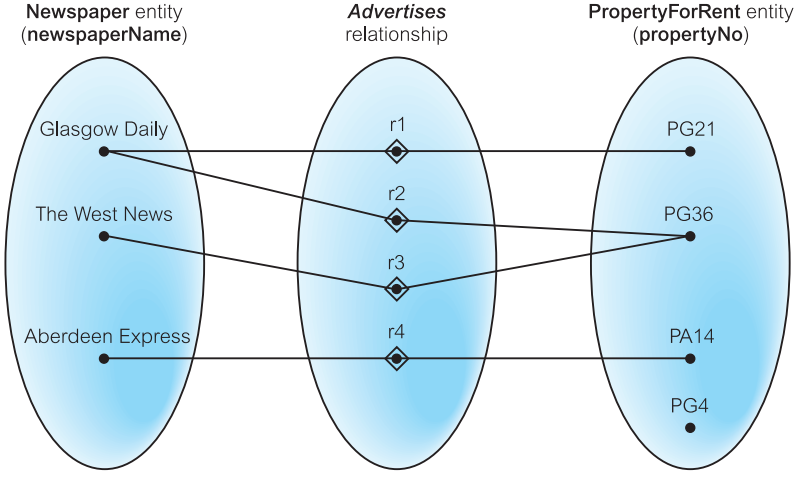


Figure 12.16(a) Semantic net showing four occurrences of the Newspaper *Advertises* PropertyForRent relationship type.

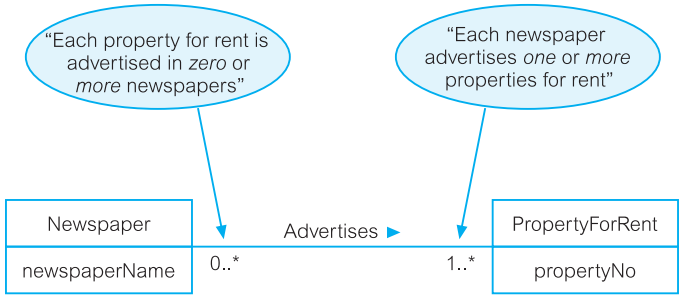


Figure 12.16(b) The multiplicity of the Newspaper *Advertises* PropertyForRent many-to-many (*:*) relationship.

newspapers there are *many* properties for rent, and for each property for rent participating in this relationship there are *many* newspapers. We refer to this type of relationship as many-to-many, which we usually abbreviate as (*:*).

Diagrammatic representation of *:* relationships

An ER diagram of the Newspaper *Advertises* PropertyForRent relationship is shown in Figure 12.16(b). To represent that each newspaper can advertise *one or more* properties for rent, we place a "1..*" beside the PropertyForRent entity type. To represent that each property for rent can be advertised by *zero or more* newspapers, we place a "0..*" beside the Newspaper entity. (Note that for a *:* relationship, we may choose a relationship name that makes sense in either direction.)

12.6.4 Multiplicity for Complex Relationships

Multiplicity for complex relationships—that is, those higher than binary—is slightly more complex.

Multiplicity (complex relationship) The number (or range) of possible occurrences of an entity type in an n-ary relationship when the other $(n–1)$ values are fixed.

In general, the multiplicity for n-ary relationships represents the potential number of entity occurrences in the relationship when $(n–1)$ values are fixed for the other participating entity types. For example, the multiplicity for a ternary relationship represents the potential range of entity occurrences of a particular entity in the relationship when the other two values representing the other two entities are fixed. Consider the ternary *Registers* relationship between Staff, Branch, and Client shown in Figure 12.7. Figure 12.17(a) displays five occurrences of the *Registers* relationship (denoted rl to r5) using a semantic net. Each relationship (rn) represents the association of a single Staff entity occurrence, a single Branch entity occurrence, and a single Client entity occurrence. We represent each entity occurrence using the values for the primary key attributes of the Staff, Branch, and Client entities, namely staffNo, branchNo, and clientNo. In Figure 12.17(a) we examine the *Registers* relationship when the values for the Staff and Branch entities are fixed.

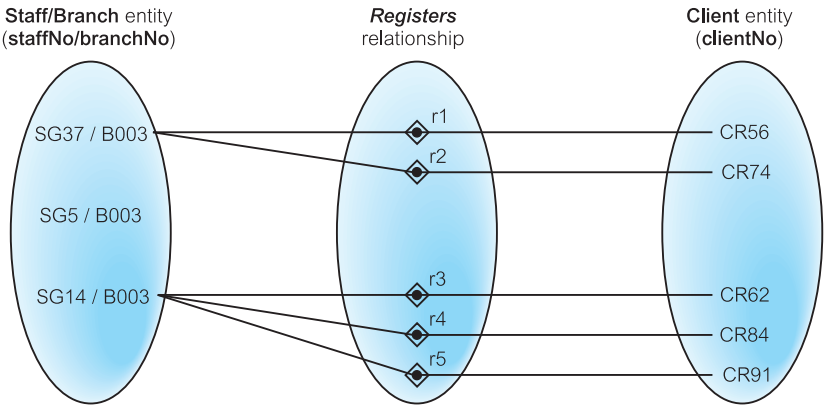


Figure 12.17(a) Semantic net showing five occurrences of the ternary *Registers* relationship with values for Staff and Branch entity types fixed.

Determining the multiplicity

In Figure 12.17(a) with the staffNo/branchNo values fixed there are *zero or more* clientNo values. For example, staffNo SG37 at branchNo B003 registers clientNo CR56 and CR74, and staffNo SG14 at branchNo B003 registers clientNo CR62, CR84, and CR91. However, SG5 at branchNo B003 registers no clients. In other words, when the staffNo and branchNo values are fixed the corresponding clientNo values are *zero or more*. Therefore, the multiplicity of the *Registers* relationship from the perspective of the Staff and Branch entities is 0..*, which is represented in the ER diagram by placing the 0..* beside the Client entity.

If we repeat this test we find that the multiplicity when Staff/Client values are fixed is 1..1, which is placed beside the Branch entity, and the Client/Branch values are fixed is 1..1, which is placed beside the Staff entity. An ER diagram of the ternary *Registers* relationship showing multiplicity is in Figure 12.17(b).

A summary of the possible ways that multiplicity constraints can be represented along with a description of the meaning is shown in Table 12.1.

12.6.5 Cardinality and Participation Constraints

Multiplicity actually consists of two separate constraints known as cardinality and participation.

Cardinality Describes the maximum number of possible relationship occurrences for an entity participating in a given relationship type.

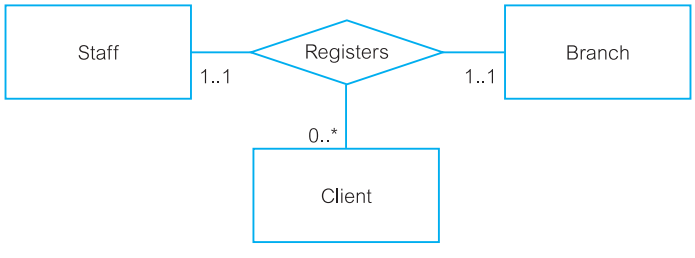


Figure 12.17(b) The multiplicity of the ternary *Registers* relationship.

TABLE 12.1 A summary of ways to represent multiplicity constraints.

ALTERNATIVE WAYS TO REPRESENT MULTIPLICITY CONSTRAINTS	MEANING
0..1	Zero or one entity occurrence
1..1 (or just 1)	Exactly one entity occurrence
0..* (or just *)	Zero or many entity occurrences
1..*	One or many entity occurrences
5..10	Minimum of 5 up to a maximum of 10 entity occurrences
0, 3, 6–8	Zero or three or six, seven, or eight entity occurrences

The cardinality of a binary relationship is what we previously referred to as a one-to-one (1:1), one-to-many (1:*), and many-to-many (*:*). The cardinality of a relationship appears as the *maximum* values for the multiplicity ranges on either side of the relationship. For example, the *Manages* relationship shown in Figure 12.18 has a one-to-one (1:1) cardinality, and this is represented by multiplicity ranges with a maximum value of 1 on both sides of the relationship.

Participation Determines whether all or only some entity occurrences participate in a relationship.

The participation constraint represents whether all entity occurrences are involved in a particular relationship (referred to as **mandatory** participation) or only some (referred to as **optional** participation). The participation of entities in a

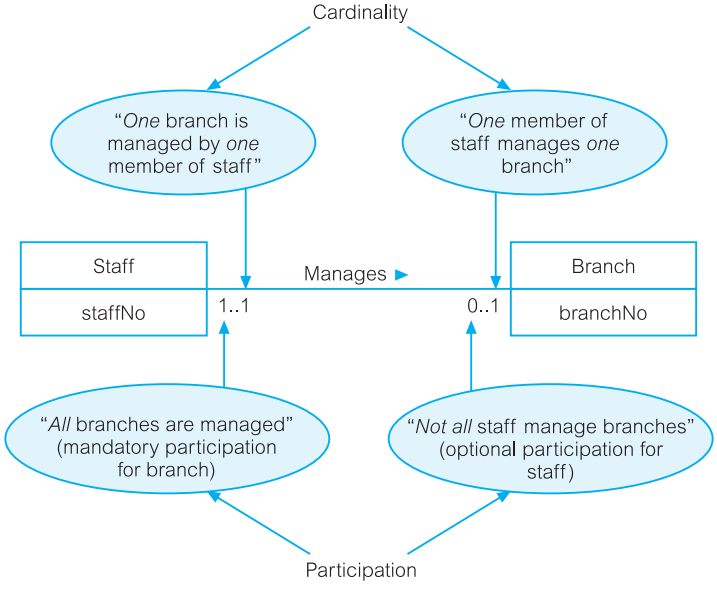


Figure 12.18 Multiplicity described as cardinality and participation constraints for the Staff *Manages* Branch (1:1) relationship.

relationship appears as the *minimum* values for the multiplicity ranges on either side of the relationship. Optional participation is represented as a minimum value of 0, and mandatory participation is shown as a minimum value of 1. It is important to note that the participation for a given entity in a relationship is represented by the minimum value on the *opposite* side of the relationship; that is, the minimum value for the multiplicity beside the related entity. For example, in Figure 12.18, the optional participation for the Staff entity in the *Manages* relationship is shown as a minimum value of 0 for the multiplicity beside the Branch entity and the mandatory participation for the Branch entity in the *Manages* relationship is shown as a minimum value of 1 for the multiplicity beside the Staff entity.

A summary of the conventions introduced in this section to represent the basic concepts of the ER model is shown on the inside front cover of this book.

12.7 Problems with ER Models

In this section we examine problems that may arise when creating an ER model. These problems are referred to as **connection traps**, and normally occur due to a misinterpretation of the meaning of certain relationships (Howe, 1989). We examine two main types of connection traps, called **fan traps** and **chasm traps**, and illustrate how to identify and resolve such problems in ER models.

In general, to identify connection traps we must ensure that the meaning of a relationship is fully understood and clearly defined. If we do not understand the relationships we may create a model that is not a true representation of the "real world."

12.7.1 Fan Traps

Fan trap Where a model represents a relationship between entity types, but the pathway between certain entity occurrences is ambiguous.

A fan trap may exist where two or more 1:* relationships fan out from the same entity. A potential fan trap is illustrated in Figure 12.19(a), which shows two 1:* relationships (*Has* and *Operates*) emanating from the same entity called Division.

This model represents the facts that a single division operates *one or more* branches and has *one or more* staff. However, a problem arises when we want to know which members of staff work at a particular branch. To appreciate the problem, we examine some occurrences of the *Has* and *Operates* relationships using values for the primary key attributes of the Staff, Division, and Branch entity types, as shown in Figure 12.19(b).

If we attempt to answer the question: "At which branch does staff number SG37 work?" we are unable to give a specific answer based on the current structure. We can determine only that staff number SG37 works at Branch B003 *or* B007. The

Figure 12.19(a) An example of a fan trap.

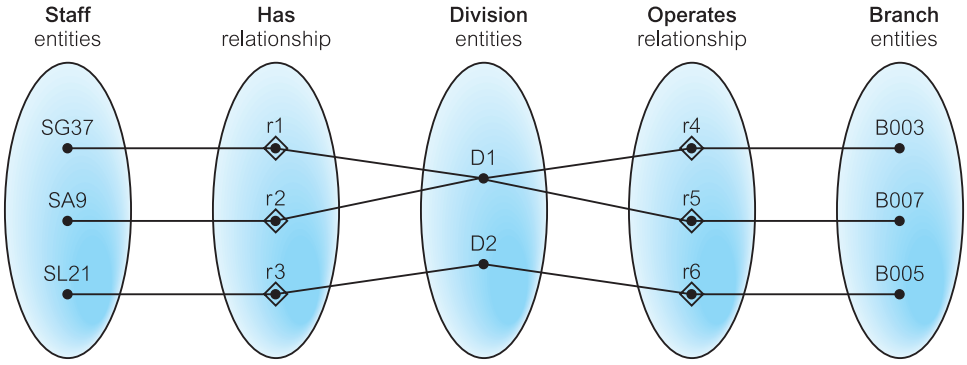


Figure 12.19(b) The semantic net of the ER model shown in Figure 12.19(a).

inability to answer this question specifically is the result of a fan trap associated with the misrepresentation of the correct relationships between the Staff, Division, and Branch entities. We resolve this fan trap by restructuring the original ER model to represent the correct association between these entities, as shown in Figure 12.20(a).

If we now examine occurrences of the *Operates* and *Has* relationships, as shown in Figure 12.20(b), we are now in a position to answer the type of question posed earlier. From this semantic net model, we can determine that staff number SG37 works at branch number B003, which is part of division Dl.

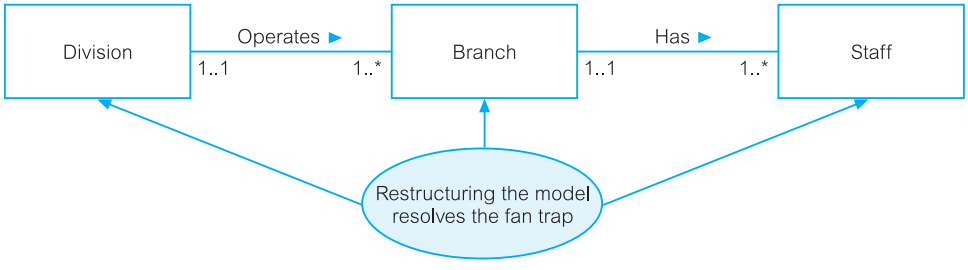


Figure 12.20(a) The ER model shown in Figure 12.19(a) restructured to remove the fan trap.

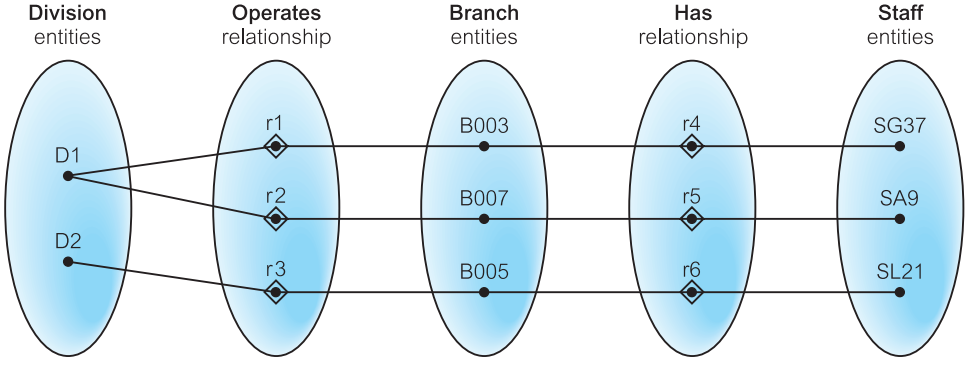


Figure 12.20(b) The semantic net of the ER model shown in Figure 12.20(a).

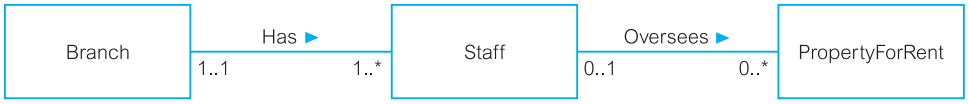


Figure 12.21(a) An example of a chasm trap.

12.7.2 Chasm Traps

Chasm trap Where a model suggests the existence of a relationship between entity types, but the pathway does not exist between certain entity occurrences.

A chasm trap may occur where there are one or more relationships with a minimum multiplicity of zero (that is, optional participation) forming part of the pathway between related entities. A potential chasm trap is illustrated in Figure 12.21(a), which shows relationships between the Branch, Staff, and PropertyForRent entities.

This model represents the facts that a single branch has *one or more* staff who oversee *zero or more* properties for rent. We also note that not all staff oversee property, and not all properties are overseen by a member of staff. A problem arises when we want to know which properties are available at each branch. To appreciate the problem, we examine some occurrences of the *Has* and *Oversees* relationships using values for the primary key attributes of the Branch, Staff, and PropertyForRent entity types, as shown in Figure 12.21(b).

If we attempt to answer the question: "At which branch is property number PA14 available?" we are unable to answer this question, as this property is not yet allocated to a member of staff working at a branch. The inability to answer this question is considered to be a loss of information (as we know a property must be available at a branch), and is the result of a chasm trap. The multiplicity of both the Staff and PropertyForRent entities in the *Oversees* relationship has a minimum value of zero, which means that some properties cannot be associated with a branch through a member of staff. Therefore, to solve this problem, we need to identify the missing relationship, which in this case is the *Offers* relationship

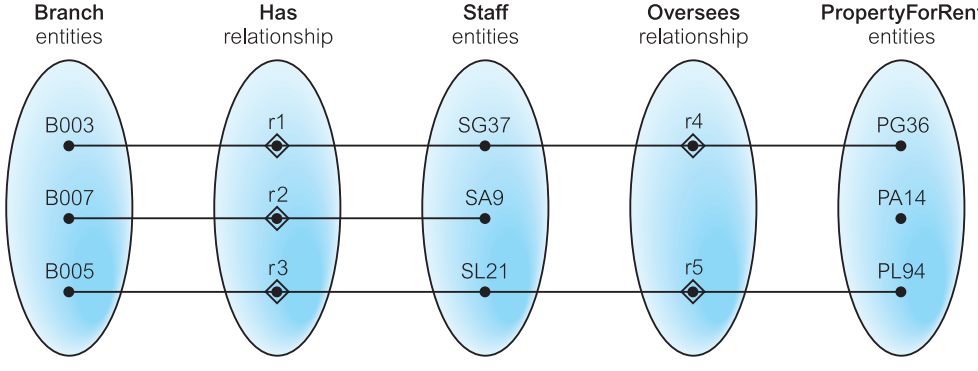


Figure 12.21(b) The semantic net of the ER model shown in Figure 12.21(a).

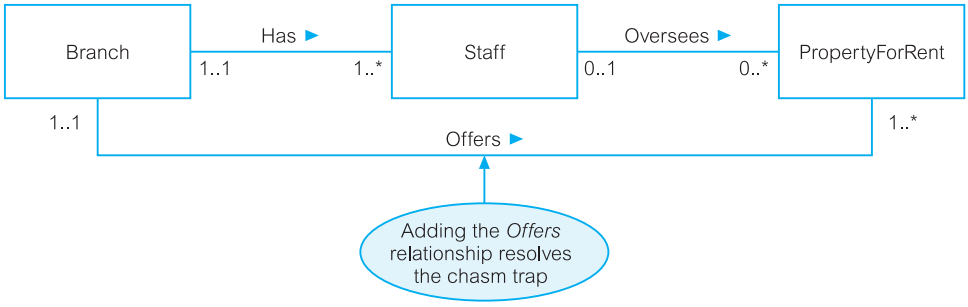


Figure 12.22(a) The ER model shown in Figure 12.21(a) restructured to remove the chasm trap.

between the Branch and PropertyForRent entities. The ER model shown in Figure 12.22(a) represents the true association between these entities. This model ensures that at all times, the properties associated with each branch are known, including properties that are not yet allocated to a member of staff.

If we now examine occurrences of the *Has*, *Oversees*, and *Offers* relationship types, as shown in Figure 12.22(b), we are now able to determine that property number PA14 is available at branch number B007.

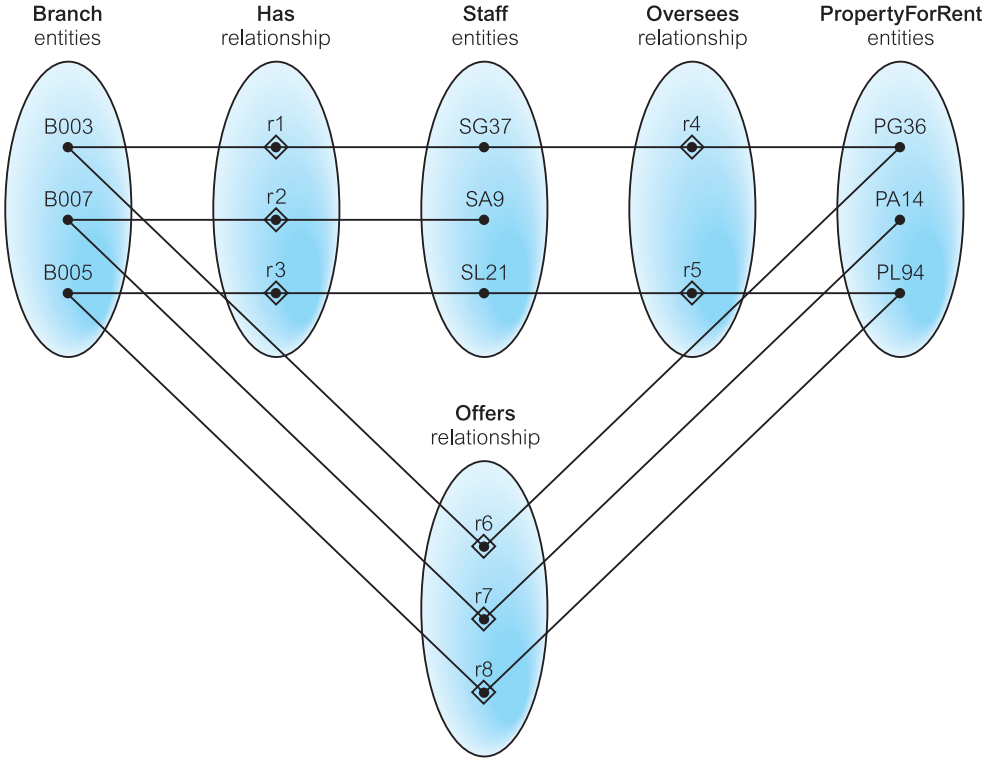


Figure 12.22(b) The semantic net of the ER model shown in Figure 12.22(a).

Chapter Summary

- An **entity type** is a group of objects with the same properties, which are identified by the enterprise as having an independent existence. An **entity occurrence** is a uniquely identifiable object of an entity type.
- A **relationship type** is a set of meaningful associations among entity types. A **relationship occurrence** is a uniquely identifiable association, which includes one occurrence from each participating entity type.
- The **degree of a relationship type** is the number of participating entity types in a relationship.
- A **recursive relationship** is a relationship type where the *same* entity type participates more than once in *different* roles.
- An **attribute** is a property of an entity or a relationship type.
- An **attribute domain** is the set of allowable values for one or more attributes.
- A **simple attribute** is composed of a single component with an independent existence.
- A **composite attribute** is composed of multiple components each with an independent existence.
- A **single-valued attribute** holds a single value for each occurrence of an entity type.
- A **multi-valued attribute** holds multiple values for each occurrence of an entity type.
- A **derived attribute** represents a value that is derivable from the value of a related attribute or set of attributes, not necessarily in the same entity.
- A **candidate key** is the minimal set of attributes that uniquely identifies each occurrence of an entity type.
- A **primary key** is the candidate key that is selected to uniquely identify each occurrence of an entity type.
- A **composite key** is a candidate key that consists of two or more attributes.
- A **strong entity type** is *not* existence-dependent on some other entity type. A **weak entity type** is existence-dependent on some other entity type.
- **Multiplicity** is the number (or range) of possible occurrences of an entity type that may relate to a single occurrence of an associated entity type through a particular relationship.
- **Multiplicity for a complex relationship** is the number (or range) of possible occurrences of an entity type in an *n*-ary relationship when the other (*n*–l) values are fixed.
- **Cardinality** describes the maximum number of possible relationship occurrences for an entity participating in a given relationship type.
- **Participation** determines whether all or only some entity occurrences participate in a given relationship.
- A **fan trap** exists where a model represents a relationship between entity types, but the pathway between certain entity occurrences is ambiguous.
- A **chasm trap** exists where a model suggests the existence of a relationship between entity types, but the pathway does not exist between certain entity occurrences.

Review Questions

12.1 Why is the ER model considered a top-down approach? Describe the four basic components of the ER model.

12.2 Describe what relationship types represent in an ER model and provide examples of unary, binary, ternary, and quaternary relationships.

12.3 The ER model uses a number of notations and tags to represent different concepts. Outline how the basic ER components are represented in an ER diagram.

12.4 Describe what the multiplicity constraint represents for a relationship type.

12.5 What are integrity constraints and how does multiplicity model these constraints?

12.6 How does multiplicity represent both the cardinality and the participation constraints on a relationship type?

12.7 Provide an example of a relationship type with attributes.

12.8 Distinguish between the Entity—Relationship model and the Entity—Relationship diagram.

12.9 Describe how fan and chasm traps can occur in an ER model and how they can be resolved.

Exercises

12.10 Create an ER model for each of the following descriptions:
- **(a)** Each company operates four departments, and each department belongs to one company.
- **(b)** Each department in part (a) employs one or more employees, and each employee works for one department.
- **(c)** Each of the employees in part (b) may or may not have one or more dependents, and each dependent belongs to one employee.
- **(d)** Each employee in part (c) may or may not have an employment history.
- **(e)** Represent all the ER models described in (a), (b), (c), and (d) as a single ER model.

12.11 Assume you have been contracted by a university to develop a database system to keep track of student registration and accommodation records. The university courses are offered by faculties. Depending on the student's IQ, there are no limitations to how many courses a student can enroll in. The faculties are not responsible for student accommodation. The university owns a number of hostels and each student is given a shared room key after enrollment. Each room has furniture attached to it.
- **(a)** Identify the main entity types for the project.
- **(b)** Identify the main relationship types and specify the multiplicity for each relationship. State any assumptions that you make about the data.
- **(c)** Using your answers for (a) and (b), draw a single ER diagram to represent the data requirements for the project.

12.12 Read the following case study, which describes the data requirements for a DVD rental company. The DVD rental company has several branches throughout the United States. The data held on each branch is the branch address made up of street, city, state, and zip code, and the telephone number. Each branch is given a branch number, which is unique throughout the company. Each branch is allocated staff, which includes a Manager. The Manager is responsible for the day-to-day running of a given branch. The data held on a member of staff is his or her name, position, and salary. Each member of staff is given a staff number, which is unique throughout the company. Each branch has a stock of DVDs. The data held on a DVD is the catalog number, DVD number, title, category, daily rental, cost, status, and the names of the main actors and the director. The catalog number uniquely identifies each DVD. However, in most cases, there are several copies of each DVD at a branch, and the individual copies are identified using the DVD number. A DVD is given a category such as Action, Adult, Children, Drama, Horror, or Sci-Fi. The status indicates whether a specific copy of a DVD is available for rent. Before borrowing a DVD from the company, a customer must first register as a member of a local branch. The data held on a member is the first and last name, address, and the date that the member registered at a branch. Each member is given a member number, which is unique throughout all branches of the company. Once registered, a member is free to rent DVDs, up to a maximum of ten at any one time. The data held on each DVD rented is the rental number, the full name and number of the member, the DVD number, title, and daily rental, and the dates the DVD is rented out and returned. The DVD number is unique throughout the company.
- **(a)** Identify the main entity types of the DVD rental company.
- **(b)** Identify the main relationship types between the entity types described in part (a) and represent each relationship as an ER diagram.

(c) Determine the multiplicity constraints for each relationships described in part (b). Represent the multiplicity for each relationship in the ER diagrams created in part (b).
(d) Identify attributes and associate them with entity or relationship types. Represent each attribute in the ER diagrams created in (c).
(e) Determine candidate and primary key attributes for each (strong) entity type.
(f) Using your answers to parts (a) to (e), attempt to represent the data requirements of the DVD rental company as a single ER diagram. State any assumptions necessary to support your design.

12.13 Create an ER model for each of the following descriptions:
(a) A large organization has several parking lots, which are used by staff.
(b) Each parking lot has a unique name, location, capacity, and number of floors (where appropriate).
(c) Each parking lot has parking spaces, which are uniquely identified using a space number.
(d) Members of staff can request the sole use of a single parking space. Each member of staff has a unique number, name, telephone extension number, and vehicle license number.
(e) Represent all the ER models described in parts (a), (b), (c), and (d) as a single ER model. Provide any assumptions necessary to support your model.

The final answer to this exercise is shown as Figure 13.11.

12.14 Create an ER model to represent the data used by the library.

The library provides books to borrowers. Each book is described by title, edition, and year of publication, and is uniquely identified using the ISBN. Each borrower is described by his or her name and address and is uniquely identified using a borrower number. The library provides one or more copies of each book and each copy is uniquely identified using a copy number, status indicating if the book is available for loan, and the allowable loan period for a given copy. A borrower may loan one or many books, and the date each book is loaned out and is returned is recorded. Loan number uniquely identifies each book loan.

The answer to this exercise is shown as Figure 13.12.

CHAPTER 13

Enhanced Entity–Relationship Modeling

Chapter Objectives

In this chapter you will learn:

- The limitations of the basic concepts of the Entity–Relationship (ER) model and the requirements to represent more complex applications using additional data modeling concepts.
- The most useful additional data modeling concepts of the Enhanced Entity–Relationship (EER) model called specialization/generalization, aggregation, and composition.
- A diagrammatic technique for displaying specialization/generalization, aggregation, and composition in an EER diagram using the Unified Modeling Language (UML).

In Chapter 12 we discussed the basic concepts of the ER model. These basic concepts are normally adequate for building data models of traditional, administrative-based database systems such as stock control, product ordering, and customer invoicing. However, since the 1980s there has been a rapid increase in the development of many new database systems that have more demanding database requirements than those of the traditional applications. Examples of such database applications include Computer-Aided Design (CAD), Computer-Aided Manufacturing (CAM), Computer-Aided Software Engineering (CASE) tools, Office Information Systems (OIS) and Multimedia Systems, Digital Publishing, and Geographical Information Systems (GIS). The main features of these applications are described in Chapter 27. As the basic concepts of ER modeling are often insufficient to represent the requirements of the newer, more complex applications, this stimulated the need to develop additional "semantic" modeling concepts. Many different semantic data models have been proposed and some of the most important semantic concepts have been successfully incorporated into the original ER model. The ER model supported with additional semantic concepts is called the Enhanced Entity–Relationship (EER) model. In this chapter we describe three of the most important and useful additional concepts of the EER model, namely specialization/generalization, aggregation, and composition. We also illustrate how specialization/generalization, aggregation, and composition are represented in an EER diagram using UML (Booch *et al.*, 1998). In Chapter 12 we introduced UML and demonstrated how UML could be used to diagrammatically represent the basic concepts of the ER model.

Structure of this Chapter In Section 13.1 we discuss the main concepts associated with specialization/generalization and illustrate how these concepts are represented in an EER diagram using UML. We conclude this section with a worked example that demonstrates how to introduce specialization/generalization into an ER model using UML. In Section 13.2 we describe the concept of aggregation and in Section 13.3 the related concept of composition. We provide examples of aggregation and composition and show how these concepts can be represented in an EER diagram using UML.

13.1 Specialization/Generalization

The concept of specialization/generalization is associated with special types of entities known as **superclasses** and **subclasses**, and the process of **attribute inheritance**. We begin this section by defining superclasses and subclasses and by examining superclass/subclass relationships. We describe the process of attribute inheritance and contrast the process of specialization with the process of generalization. We then describe the two main types of constraints on superclass/subclass relationships called participation and disjoint constraints. We show how to represent specialization/generalization in an EER diagram using UML. We conclude this section with a worked example of how specialization/generalization may be introduced into the ER model of the Branch user views of the *DreamHome* case study described in Appendix A and shown in Figure 12.1.

13.1.1 Superclasses and Subclasses

As we discussed in Chapter 12, an entity type represents a set of entities of the same type such as Staff, Branch, and PropertyForRent. We can also form entity types into a hierarchy containing superclasses and subclasses.

Superclass An entity type that includes one or more distinct subgroupings of its occurrences, which must be represented in a data model.

Subclass A distinct subgrouping of occurrences of an entity type, which must be represented in a data model.

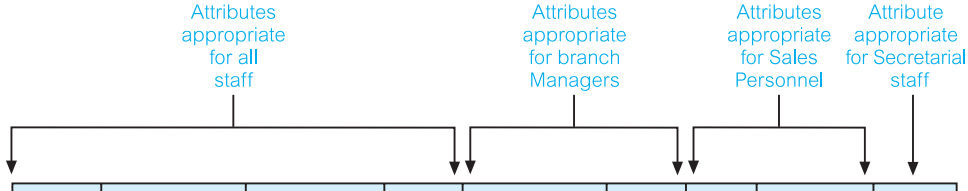
Entity types that have distinct subclasses are called superclasses. For example, the entities that are members of the Staff entity type may be classified as Manager, SalesPersonnel, and Secretary. In other words, the Staff entity is referred to as the **superclass** of the Manager, SalesPersonnel, and Secretary **subclasses**. The relationship between a superclass and any one of its subclasses is called a superclass/subclass relationship. For example, Staff/Manager has a superclass/subclass relationship.

13.1.2 Superclass/Subclass Relationships

Each member of a subclass is also a member of the superclass. In other words, the entity in the subclass is the same entity in the superclass, but has a distinct role. The relationship between a superclass and a subclass is one-to-one (1:1) and is called a superclass/subclass relationship (see Section 12.6.1). Some superclasses may contain overlapping subclasses, as illustrated by a member of staff who is both a Manager and a member of Sales Personnel. In this example, Manager and SalesPersonnel are overlapping subclasses of the Staff superclass. On the other hand, not every member of a superclass is necessarily a member of a subclass; for example, members of staff without a distinct job role such as a Manager or a member of Sales Personnel.

We can use superclasses and subclasses to avoid describing different types of staff with possibly different attributes within a single entity. For example, Sales Personnel may have special attributes such as salesArea and carAllowance. If all staff attributes and those specific to particular jobs are described by a single Staff entity, this may result in a lot of nulls for the job-specific attributes. Clearly, Sales Personnel have common attributes with other staff, such as staffNo, name, position, and salary. However, it is the unshared attributes that cause problems when we try to represent all members of staff within a single entity. We can also show relationships that are associated with only particular types of staff (subclasses) and not with staff, in general. For example, Sales Personnel may have distinct relationships that are not appropriate for all staff, such as SalesPersonnel *Uses* Car.

To illustrate these points, consider the relation called AllStaff shown in Figure 13.1. This relation holds the details of all members of staff, no matter what position they hold. A consequence of holding all staff details in one relation is that while the attributes appropriate to all staff are filled (namely, staffNo, name, position, and salary), those that are only applicable to particular job roles are only partially filled. For example, the attributes associated with the Manager (mgrStartDate and bonus),



staffNo	name	position	salary	mgrStartDate	bonus	sales Area	car Allowance	typing Speed
SL21	John White	Manager	30000	01/02/95	2000			
SG37	Ann Beech	Assistant	12000					
SG66	Mary Martinez	Sales Manager	27000			SA1A	5000	
SA9	Mary Howe	Assistant	9000					
SL89	Stuart Stern	Secretary	8500					100
SL31	Robert Chin	Snr Sales Asst	17000			SA2B	3700	
SG5	Susan Brand	Manager	24000	01/06/91	2350			

Figure 13.1 The AllStaff relation holding details of all staff.

SalesPersonnel (salesArea and carAllowance), and Secretary (typingSpeed) subclasses have values for those members in these subclasses. In other words, the attributes associated with the Manager, SalesPersonnel, and Secretary subclasses are empty for those members of staff not in these subclasses.

There are two important reasons for introducing the concepts of superclasses and subclasses into an ER model. First, it avoids describing similar concepts more than once, thereby saving time for the designer and making the ER diagram more readable. Second, it adds more semantic information to the design in a form that is familiar to many people. For example, the assertions that "Manager IS-A member of staff" and "flat IS-A type of property," communicates significant semantic content in a concise form.

13.1.3 Attribute Inheritance

As mentioned earlier, an entity in a subclass represents the same "real world" object as in the superclass, and may possess subclass-specific attributes, as well as those associated with the superclass. For example, a member of the SalesPersonnel subclass *inherits* all the attributes of the Staff superclass, such as staffNo, name, position, and salary together with those specifically associated with the SalesPersonnel subclass, such as salesArea and carAllowance.

A subclass is an entity in its own right and so it may also have one or more subclasses. An entity and its subclasses and their subclasses, and so on, is called a **type hierarchy**. Type hierarchies are known by a variety of names, including **specialization hierarchy** (for example, Manager is a specialization of Staff), **generalization hierarchy** (for example, Staff is a generalization of Manager), and **IS-A hierarchy** (for example, Manager IS-A (member of) Staff). We describe the process of specialization and generalization in the following sections.

A subclass with more than one superclass is called a **shared subclass**. In other words, a member of a shared subclass must be a member of the associated superclasses. As a consequence, the attributes of the superclasses are inherited by the shared subclass, which may also have its own additional attributes. This process is referred to as **multiple inheritance**.

13.1.4 Specialization Process

Specialization The process of maximizing the differences between members of an entity by identifying their distinguishing characteristics.

Specialization is a top-down approach to defining a set of superclasses and their related subclasses. The set of subclasses is defined on the basis of some distinguishing characteristics of the entities in the superclass. When we identify a set of subclasses of an entity type, we then associate attributes specific to each subclass (where necessary), and also identify any relationships between each subclass and other entity types or subclasses (where necessary). For example, consider a model where all members of staff are represented as an entity called Staff. If we apply the process of specialization on the Staff entity, we attempt to identify differences between members of this entity, such as members with distinctive attributes

and/or relationships. As described earlier, staff with the job roles of Manager, Sales Personnel, and Secretary have distinctive attributes and therefore we identify Manager, SalesPersonnel, and Secretary as subclasses of a specialized Staff superclass.

13.1.5 Generalization Process

Generalization The process of minimizing the differences between entities by identifying their common characteristics.

The process of generalization is a bottom-up approach, that results in the identification of a generalized superclass from the original entity types. For example, consider a model where Manager, SalesPersonnel, and Secretary are represented as distinct entity types. If we apply the process of generalization on these entities, we attempt to identify similarities between them, such as common attributes and relationships. As stated earlier, these entities share attributes common to all staff, and therefore we identify Manager, SalesPersonnel, and Secretary as subclasses of a generalized Staff superclass.

As the process of generalization can be viewed as the reverse of the specialization process, we refer to this modeling concept as "specialization/generalization."

Diagrammatic representation of specialization/generalization

UML has a special notation for representing specialization/generalization. For example, consider the specialization/generalization of the Staff entity into subclasses that represent job roles. The Staff superclass and the Manager, SalesPersonnel, and Secretary subclasses can be represented in an EER diagram, as illustrated in Figure 13.2. Note that the Staff superclass and the subclasses, being entities, are represented as rectangles. The subclasses are attached by lines to a triangle that points toward the superclass. The label below the specialization/generalization triangle, shown as {Optional, And}, describes the constraints on the relationship between the superclass and its subclasses. These constraints are discussed in more detail in Section 13.1.6.

Attributes that are specific to a given subclass are listed in the lower section of the rectangle representing that subclass. For example, salesArea and carAllowance attributes are associated only with the SalesPersonnel subclass, and are not applicable to the Manager or Secretary subclasses. Similarly, we show attributes that are specific to the Manager (mgrStartDate and bonus) and Secretary (typingSpeed) subclasses.

Attributes that are common to all subclasses are listed in the lower section of the rectangle representing the superclass. For example, staffNo, name, position, and salary attributes are common to all members of staff and are associated with the Staff superclass. Note that we can also show relationships that are only applicable to specific subclasses. For example, in Figure 13.2, the Manager subclass is related to the Branch entity through the *Manages* relationship, whereas the Staff superclass is related to the Branch entity through the *Has* relationship.

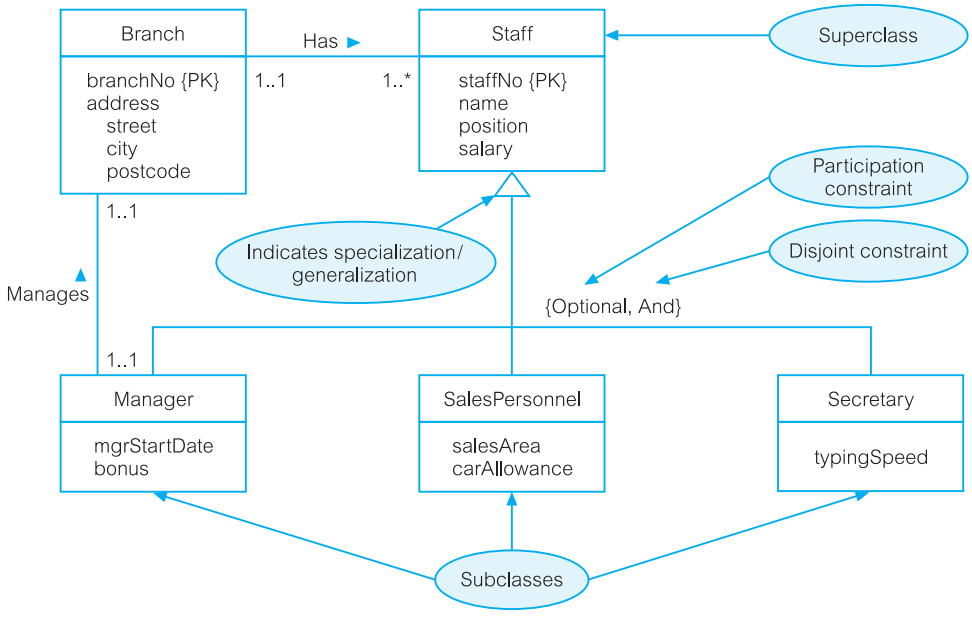


Figure 13.2 Specialization/generalization of the Staff entity into subclasses representing job roles.

We may have several specializations of the same entity based on different distinguishing characteristics. For example, another specialization of the Staff entity may produce the subclasses FullTimePermanent and PartTimeTemporary, which distinguishes between the types of employment contract for members of staff. The specialization of the Staff entity type into job role and contract of employment subclasses is shown in Figure 13.3. In this figure, we show attributes that are specific to the FullTimePermanent (salaryScale and holidayAllowance) and PartTimeTemporary (hourlyRate) subclasses.

As described earlier, a superclass and its subclasses and their subclasses, and so on, is called a type hierarchy. An example of a type hierarchy is shown in Figure 13.4, where the job roles specialization/generalization shown in Figure 13.2 are expanded to show a shared subclass called SalesManager and the subclass called Secretary with its own subclass called AssistantSecretary. In other words, a member of the SalesManager shared subclass must be a member of the SalesPersonnel and Manager subclasses as well as the Staff superclass. As a consequence, the attributes of the Staff superclass (staffNo, name, position, and salary), and the attributes of the subclasses SalesPersonnel (salesArea and carAllowance) and Manager (mgrStartDate and bonus) are inherited by the SalesManager subclass, which also has its own additional attribute called salesTarget.

AssistantSecretary is a subclass of Secretary, which is a subclass of Staff. This means that a member of the AssistantSecretary subclass must be a member of the Secretary subclass and the Staff superclass. As a consequence, the attributes of the Staff superclass (staffNo, name, position, and salary) and the attribute of the Secretary subclass (typingSpeed) are inherited by the AssistantSecretary subclass, which also has its own additional attribute called startDate.

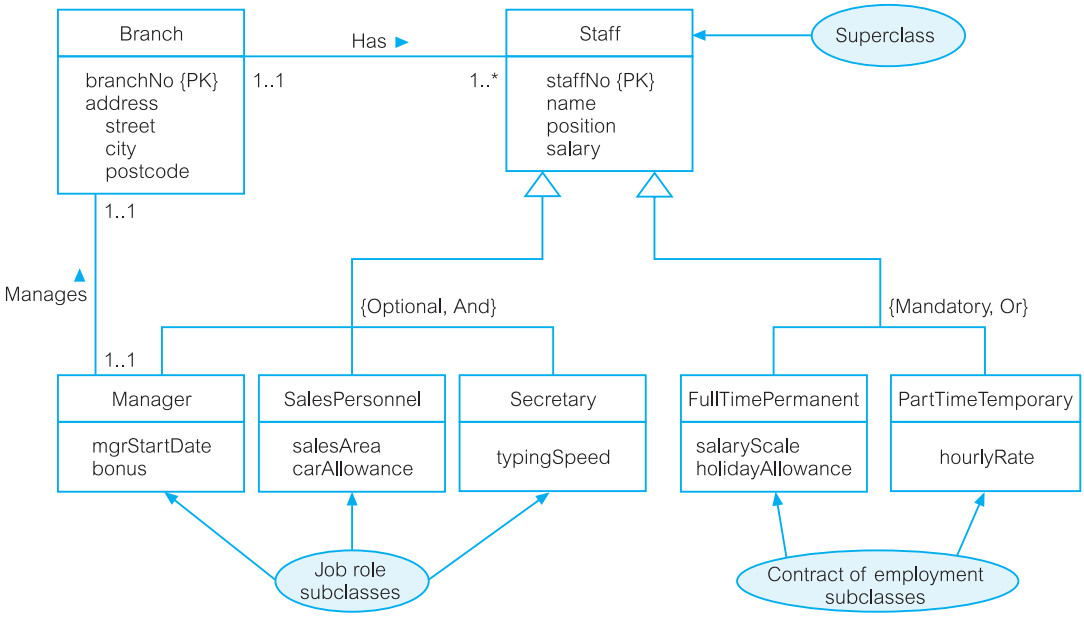


Figure 13.3 Specialization/generalization of the Staff entity into subclasses representing job roles and contracts of employment.

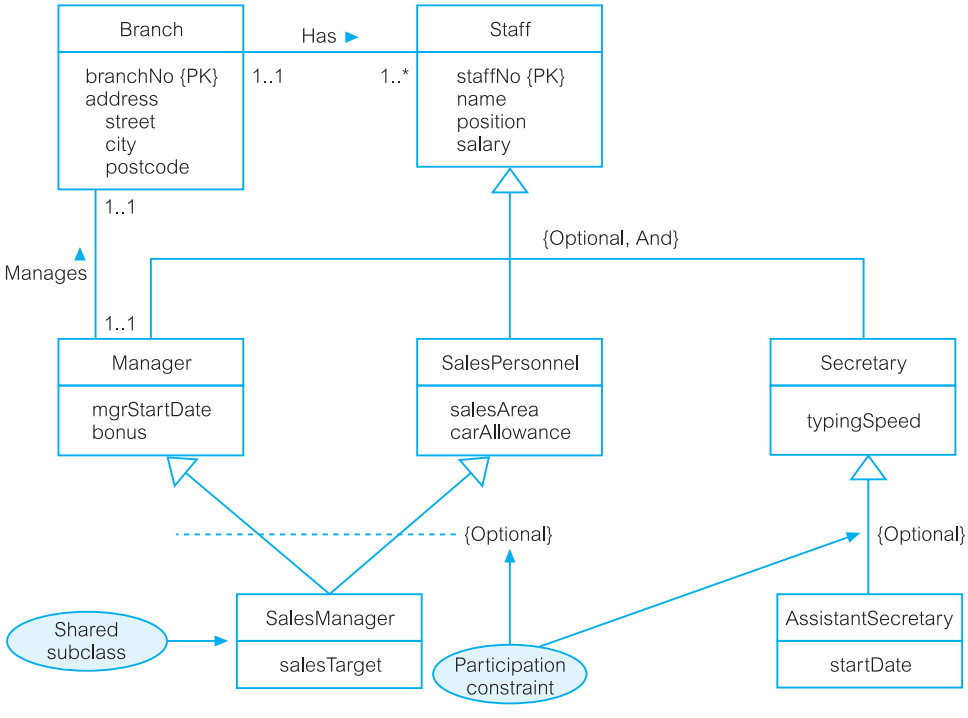


Figure 13.4 Specialization/generalization of the Staff entity into job roles including a shared subclass called SalesManager and a subclass called Secretary with its own subclass called AssistantSecretary.

13.1.6 Constraints on Specialization/Generalization

There are two constraints that may apply to a specialization/generalization called **participation constraints** and **disjoint constraints**.

Participation constraints

Participation constraint — Determines whether every member in the superclass must participate as a member of a subclass.

A participation constraint may be **mandatory** or **optional**. A superclass/subclass relationship with mandatory participation specifies that every member in the superclass must also be a member of a subclass. To represent mandatory participation, "Mandatory" is placed in curly brackets below the triangle that points towards the superclass. For example, in Figure 13.3 the contract of employment specialization/generalization is mandatory participation, which means that every member of staff must have a contract of employment.

A superclass/subclass relationship with optional participation specifies that a member of a superclass need not belong to any of its subclasses. To represent optional participation, "Optional" is placed in curly brackets below the triangle that points towards the superclass. For example, in Figure 13.3 the job role specialization/generalization has optional participation, which means that a member of staff need not have an additional job role such as a Manager, Sales Personnel, or Secretary.

Disjoint constraints

Disjoint constraint — Describes the relationship between members of the subclasses and indicates whether it is possible for a member of a superclass to be a member of one, or more than one, subclass.

The disjoint constraint only applies when a superclass has more than one subclass. If the subclasses are **disjoint**, then an entity occurrence can be a member of only one of the subclasses. To represent a disjoint superclass/subclass relationship, "Or" is placed next to the participation constraint within the curly brackets. For example, in Figure 12.3 the subclasses of the contract of employment specialization/generalization is disjoint, which means that a member of staff must have a full-time permanent *or* a part-time temporary contract, but not both.

If subclasses of a specialization/generalization are not disjoint (called **nondisjoint**), then an entity occurrence may be a member of more than one subclass. To represent a nondisjoint superclass/subclass relationship, "And" is placed next to the participation constraint within the curly brackets. For example, in Figure 13.3 the job role specialization/generalization is nondisjoint, which means that an entity occurrence can be a member of both the Manager, SalesPersonnel, and Secretary subclasses. This is confirmed by the presence of the shared subclass called SalesManager shown in Figure 13.4. Note that it is not necessary to include the disjoint constraint for hierarchies that have a single subclass at a given level and for this reason only the participation constraint is shown for the SalesManager and AssistantSecretary subclasses of Figure 13.4.

The disjoint and participation constraints of specialization and generalization are distinct, giving rise to four categories: "mandatory and disjoint," "optional and disjoint," "mandatory and nondisjoint," and "optional and nondisjoint."

13.1.7 Worked Example of using Specialization/Generalization to Model the Branch View of the *DreamHome* Case Study

The database design methodology described in this book includes the use of specialization/generalization as an optional step (Step 1.6) in building an EER model. The choice to use this step is dependent on the complexity of the enterprise (or part of the enterprise) being modeled and whether using the additional concepts of the EER model will help the process of database design.

In Chapter 12 we described the basic concepts necessary to build an ER model to represent the Branch user views of the *DreamHome* case study. This model was shown as an ER diagram in Figure 12.1. In this section, we show how specialization/generalization may be used to convert the ER model of the Branch user views into an EER model.

As a starting point, we first consider the entities shown in Figure 12.1. We examine the attributes and relationships associated with each entity to identify any similarities or differences between the entities. In the Branch user views' requirements specification there are several instances where there is the potential to use specialization/generalization as discussed shortly.

(a) For example, consider the Staff entity in Figure 12.1, which represents all members of staff. However, in the data requirements specification for the Branch user views of the *DreamHome* case study given in Appendix A, there are two key job roles mentioned, namely Manager and Supervisor. We have three options as to how we may best model members of staff. The first option is to represent all members of staff as a generalized Staff entity (as in Figure 12.1), the second option is to create three distinct entities Staff, Manager, and Supervisor, and the third option is to represent the Manager and Supervisor entities as subclasses of a Staff superclass. The option we select is based on the commonality of attributes and relationships associated with each entity. For example, all attributes of the Staff entity are represented in the Manager and Supervisor entities, including the same primary key, namely staffNo. Furthermore, the Supervisor entity does not have any additional attributes representing this job role. On the other hand, the Manager entity has two additional attributes: mgrStartDate and bonus. In addition, both the Manager and Supervisor entities are associated with distinct relationships, namely Manager *Manages* Branch and Supervisor *Supervises* Staff. Based on this information, we select the third option and create Manager and Supervisor subclasses of the Staff superclass, as shown in Figure 13.5. Note that in this EER diagram, the subclasses are shown above the superclass. The relative positioning of the subclasses and superclass is not significant, however; what is important is that the specialization/generalization triangle points toward the superclass.

The specialization/generalization of the Staff entity is optional and disjoint (shown as {Optional, Or}), as not all members of staff are Managers or Supervisors, and in addition a single member of staff cannot be both a Manager

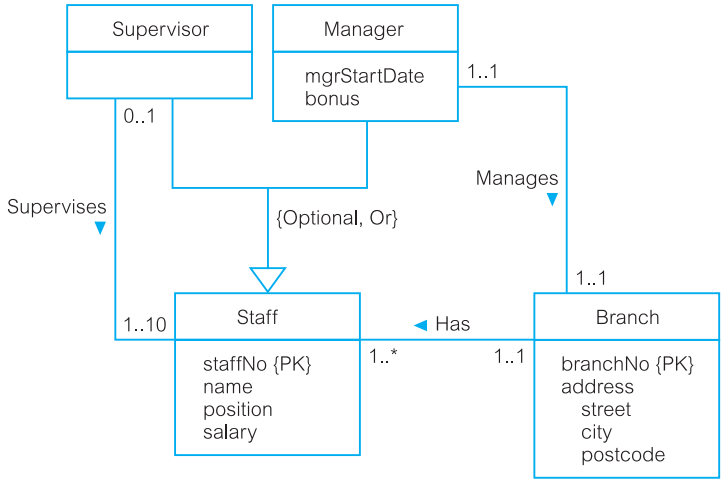


Figure 13.5
Staff superclass with Supervisor and Manager subclasses.

and a Supervisor. This representation is particularly useful for displaying the shared attributes associated with these subclasses and the Staff superclass and also the distinct relationships associated with each subclass, namely Manager *Manages* Branch and Supervisor *Supervises* Staff.

(b) Consider for specialization/generalization the relationship between owners of property. The data requirements specification for the Branch user views describes two types of owner, namely PrivateOwner and BusinessOwner as shown in Figure 12.1. Again, we have three options as to how we may best model owners of property. The first option is to leave PrivateOwner and BusinessOwner as two distinct entities (as shown in Figure 12.1), the second option is to represent both types of owner as a generalized Owner entity, and the third option is to represent the PrivateOwner and BusinessOwner entities as subclasses of an Owner superclass. Before we are able to reach a decision we first examine the attributes and relationships associated with these entities. PrivateOwner and BusinessOwner entities share common attributes, namely address and telNo and have a similar relationship with property for rent (namely PrivateOwner *POwns* PropertyForRent and BusinessOwner *BOwns* PropertyForRent). However, both types of owner also have different attributes; for example, PrivateOwner has distinct attributes ownerNo and name, and BusinessOwner has distinct attributes bName, bType, and contactName. In this case, we create a superclass called Owner, with PrivateOwner and BusinessOwner as subclasses, as shown in Figure 13.6.

The specialization/generalization of the Owner entity is mandatory and disjoint (shown as {Mandatory, Or}), as an owner must be either a private owner *or* a business owner, but cannot be both. Note that we choose to relate the Owner superclass to the PropertyForRent entity using the relationship called Owns.

The examples of specialization/generalization described previously are relatively straightforward. However, the specialization/generalization process can be taken further as illustrated in the following example.

(c) There are several persons with common characteristics described in the data requirements specification for the Branch user views of the *DreamHome* case

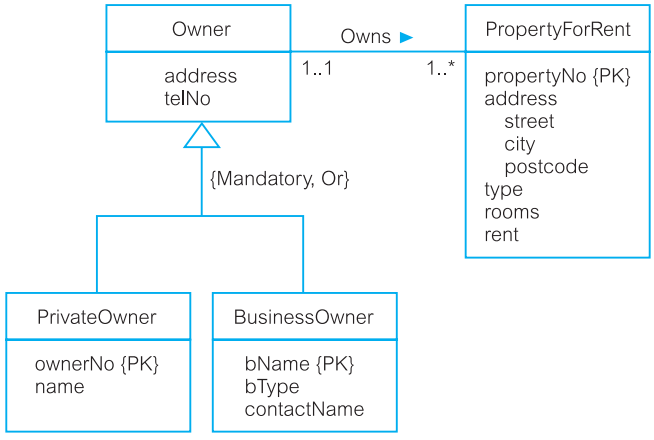


Figure 13.6
Owner superclass with PrivateOwner and BusinessOwner subclasses.

study. For example, members of staff, private property owners, and clients all have number and name attributes. We could create a Person superclass with Staff (including Manager and Supervisor subclasses), PrivateOwner, and Client as subclasses, as shown in Figure 13.7.

We now consider to what extent we wish to use specialization/generalization to represent the Branch user views of the *DreamHome* case study. We decide to use the specialization/generalization examples described in (a) and (b) above but not (c), as shown in Figure 13.8. To simplify the EER diagram only attributes

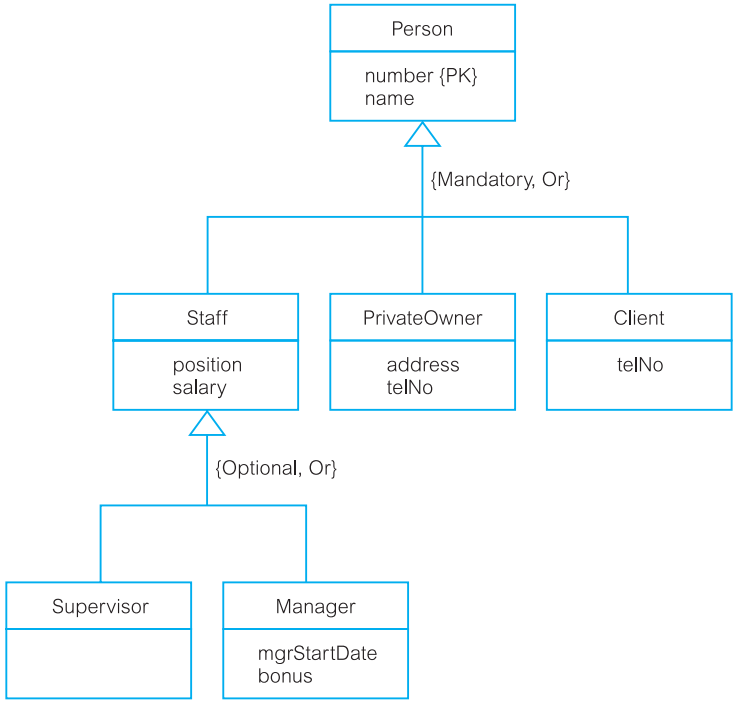


Figure 13.7
Person superclass with Staff (including Supervisor and Manager subclasses), PrivateOwner, and Client subclasses.

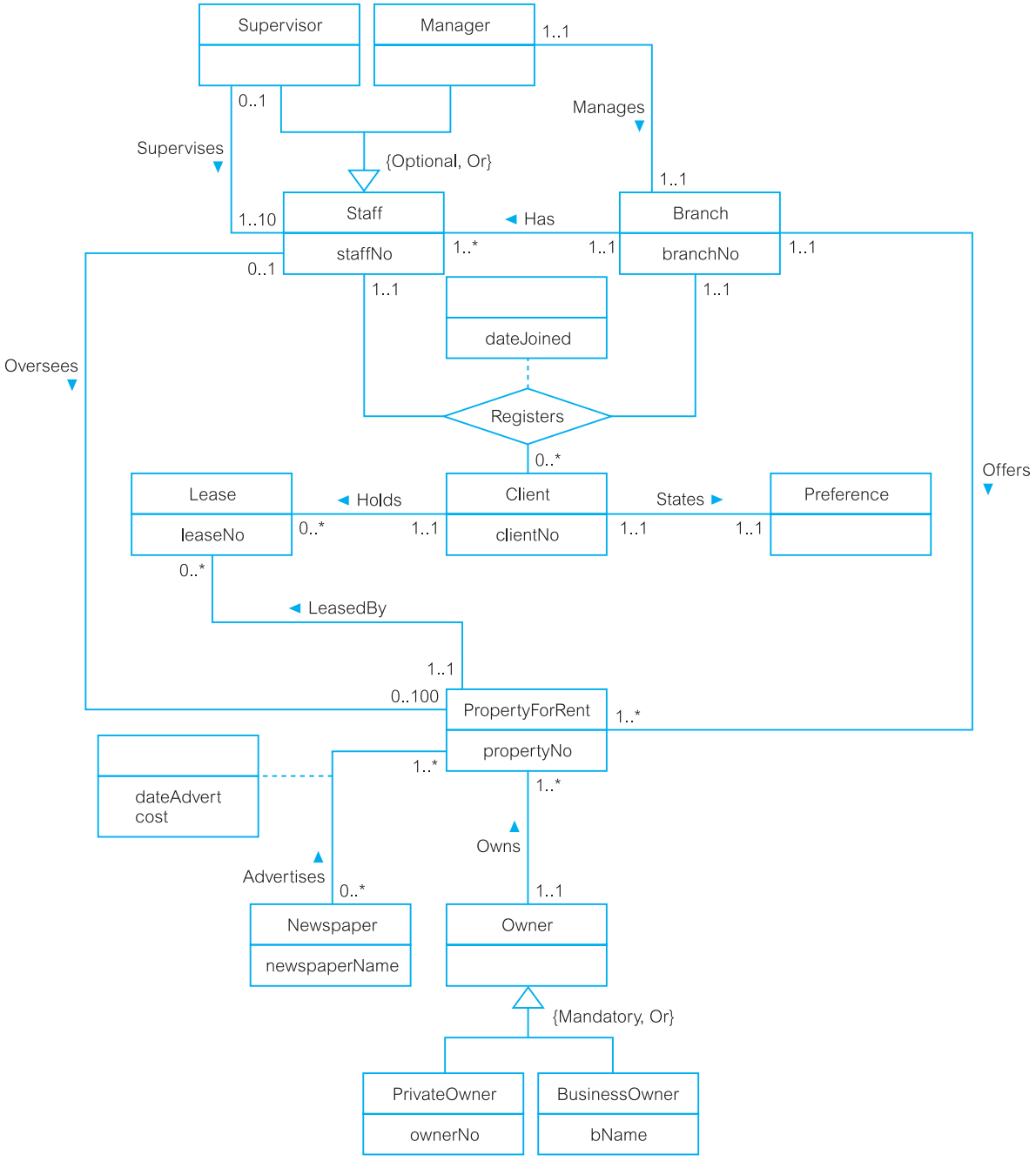


Figure 13.8 An EER model of the Branch user views of *DreamHome* with specialization/generalization.

associated with primary keys or relationships are shown. We leave out the representation shown in Figure 13.7 from the final EER model, because the use of specialization/generalization in this case places too much emphasis on the relationship between entities that are persons rather than emphasizing the relationship between these entities and some of the core entities such as Branch and PropertyForRent.

The option to use specialization/generalization, and to what extent, is a subjective decision. In fact, the use of specialization/generalization is presented as an optional step in our methodology for conceptual database design discussed in Chapter 16, Step 1.6.

As described in Section 2.3, the purpose of a data model is to provide the concepts and notations that allow database designers and end-users to unambiguously and accurately communicate their understanding of the enterprise data. Therefore, if we keep these goals in mind, we should use the additional concepts of specialization/generalization only when the enterprise data is too complex to easily represent using only the basic concepts of the ER model.

At this stage, we may consider whether the introduction of specialization/generalization to represent the Branch user views of *DreamHome* is a good idea. In other words, is the requirement specification for the Branch user views better represented as the ER model shown in Figure 12.1 or as the EER model shown in Figure 13.8? We leave this for the reader to consider.

13.2 Aggregation

Aggregation Represents a "has-a" or "is-part-of" relationship between entity types, where one represents the "whole" and the other the "part."

A relationship represents an association between two entity types that are conceptually at the same level. Sometimes we want to model a "has-a" or "is-part-of" relationship, in which one entity represents a larger entity (the "whole"), consisting of smaller entities (the "parts"). This special kind of relationship is called an **aggregation** (Booch *et al.*, 1998). Aggregation does not change the meaning of navigation across the relationship between the whole and its parts, or link the lifetimes of the whole and its parts. An example of an aggregation is the *Has* relationship, which relates the Branch entity (the "whole") to the Staff entity (the "part").

Diagrammatic representation of aggregation

UML represents aggregation by placing an open diamond shape at one end of the relationship line, next to the entity that represents the "whole." In Figure 13.9, we redraw part of the EER diagram shown in Figure 13.8 to demonstrate aggregation. This EER diagram displays two examples of aggregation, namely Branch *Has* Staff and Branch *Offers* PropertyForRent. In both relationships, the Branch entity represents the "whole" and therefore the open diamond shape is placed beside this entity.

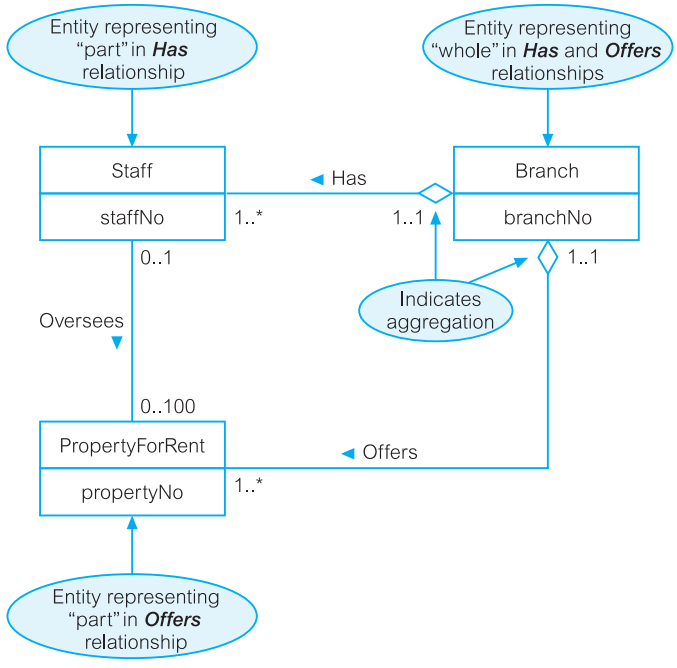


Figure 13.9
Examples of aggregation: Branch *Has* Staff and Branch *Offers* PropertyForRent.

13.3 Composition

Composition A specific form of aggregation that represents an association between entities, where there is a strong ownership and coincidental lifetime between the "whole" and the "part."

Aggregation is entirely conceptual and does nothing more than distinguish a "whole" from a "part." However, there is a variation of aggregation called **composition** that represents a strong ownership and coincidental lifetime between the "whole" and the "part" (Booch *et al.*, 1998). In a composite, the "whole" is responsible for the disposition of the "parts," which means that the composition must manage the creation and destruction of its "parts." In other words, an object may be part of only one composite at a time. There are no examples of composition in Figure 13.8. For the purposes of discussion, consider an example of a composition, namely the *Displays* relationship, which relates the Newspaper entity to the Advert entity. As a composition, this emphasizes the fact that an Advert entity (the "part") belongs to exactly one Newspaper entity (the "whole"). This is in contrast to aggregation, in which a part may be shared by many wholes. For example, a Staff entity may be "a part of" one or more Branches entities.

Diagrammatic representation of composition

UML represents composition by placing a filled-in diamond shape at one end of the relationship line next to the entity that represents the "whole" in the relationship. For example, to represent the Newspaper *Displays* Advert composition, the filled-in diamond shape is placed next to the Newspaper entity, which is the "whole" in this relationship, as shown in Figure 13.10.

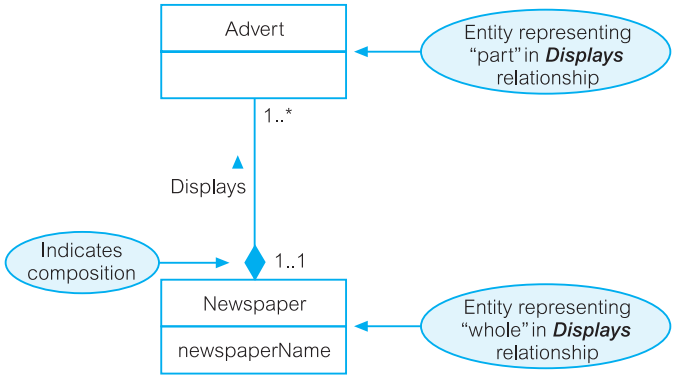


Figure 13.10
An example of composition: Newspaper *Displays* Advert.

As discussed with specialization/generalization, the options to use aggregation and composition, and to what extent, are again subjective decisions. Aggregation and composition should be used only when there is a requirement to emphasize special relationships between entity types such as "has-a" or "is-part-of," which has implications on the creation, update, and deletion of these closely related entities. We discuss how to represent such constraints between entity types in our methodology for logical database design in Chapter 17, Step 2.4.

If we remember that the major aim of a data model is to unambiguously and accurately communicate an understanding of the enterprise data. We should only use the additional concepts of aggregation and composition when the enterprise data is too complex to easily represent using only the basic concepts of the ER model.

Chapter Summary

- A **superclass** is an entity type that includes one or more distinct subgroupings of its occurrences, which require to be represented in a data model. A **subclass** is a distinct subgrouping of occurrences of an entity type, which require to be represented in a data model.
- **Specialization** is the process of maximizing the differences between members of an entity by identifying their distinguishing features.
- **Generalization** is the process of minimizing the differences between entities by identifying their common features.
- There are two constraints that may apply to a specialization/generalization called **participation constraints** and **disjoint constraints**.
- A **participation constraint** determines whether every member in the superclass must participate as a member of a subclass.
- A **disjoint constraint** describes the relationship between members of the subclasses and indicates whether it is possible for a member of a superclass to be a member of one, or more than one, subclass.
- **Aggregation** represents a "has-a" or "is-part-of" relationship between entity types, where one represents the "whole" and the other the "part."

- **Composition** is a specific form of aggregation that represents an association between entities, where there is a strong ownership and coincidental lifetime between the "whole" and the "part."

Review Questions

13.1 What are the key differences between the ER and EER models?

13.2 Describe situations that would call for an enhanced entity—relationship in data modeling.

13.3 Describe and illustrate using an example the process of attribute inheritance.

13.4 What are the main reasons for introducing the concepts of superclasses and subclasses into an ER model?

13.5 Describe what a shared subclass represents and how this concept relates to multiple inheritance.

13.6 Describe and contrast the process of specialization with the process of generalization.

13.7 Describe the UML notation used to represent superclass/subclass relationships.

13.8 Describe and contrast the concepts of aggregation and composition and provide an example of each.

Exercises

13.9 Consider whether it is appropriate to introduce the enhanced concepts of specialization/generalization, aggregation, and/or composition for the case studies described in Appendix B.

13.10 The features of EER and ER models can co-exist in the same diagram, What situations lead to this co-existence? Analyze the case presented in question 12.11 and redraw the ER diagram as an EER diagram with the additional enhanced concepts.

13.11 Introduce specialization/generalization concepts into the ER model shown in Figure 13.11 and described in Exercise 12.13 to show the following:

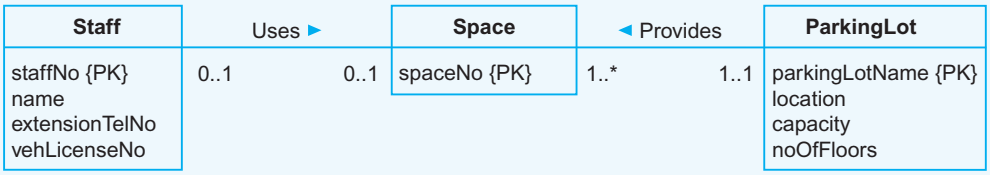


Figure 13.11 Parking lot ER model was described in Exercise 12.13.

(a) The majority of parking spaces are under cover and each can be allocated for use by a member of staff for a monthly rate.

(b) Parking spaces that are not under cover are free to use and each can be allocated for use by a member of staff.

(c) Up to twenty covered parking spaces are available for use by visitors to the company. However, only members of staff are able to book out a space for the day of the visit. There is no charge for this type of booking, but the member of staff must provide the visitor's vehicle license number.

The final answer to this exercise is shown as Figure 17.11.

13.12 The library case study described in Exercise 12.14 is extended to include the fact that the library has a significant stock of books that are no longer suitable for loaning out. These books can be sold for a fraction of the original cost. However, not all library books are eventually sold as many are considered too damaged to sell on, or are simply lost or stolen. Each book copy that is suitable for selling has a price and the date that the book is no longer to be loaned out. Introduce enhanced concepts into the ER model shown in Figure 13.12 and described in Exercise 12.14 to accommodate this extension to the original case study.

The answer to this exercise is shown as Figure 17.12.

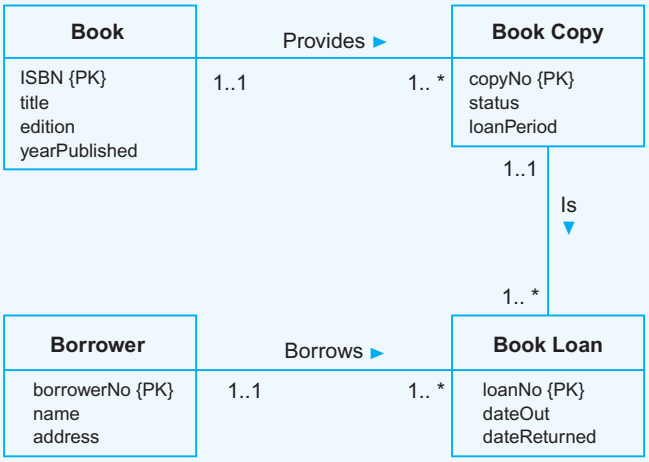


Figure 13.12 Library ER model was described in Exercise 12.14.