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, almost 40 years later, 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) for a fuller discussion.

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

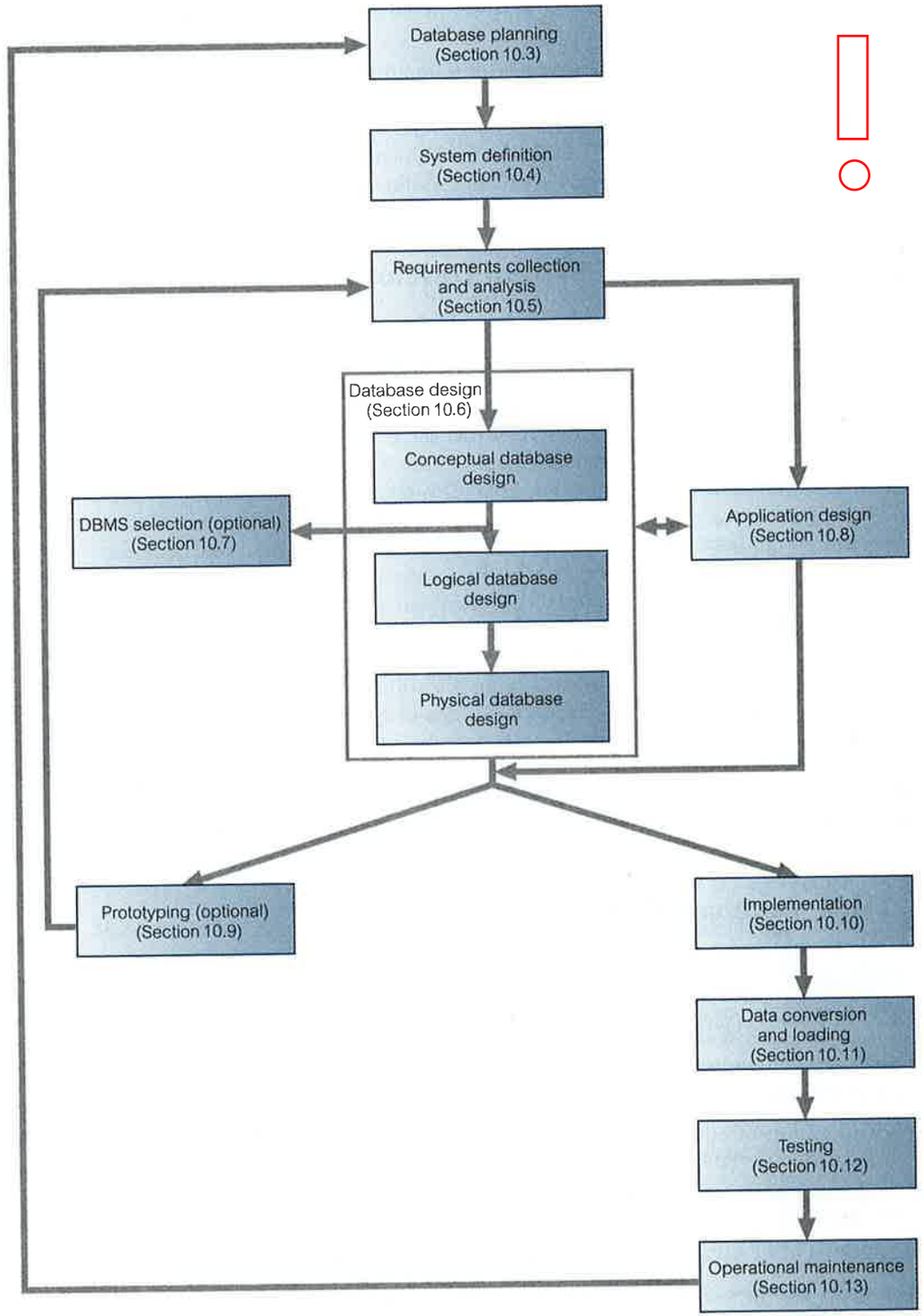

Figure 10.1 The stages of the database system development lifecycle.

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 (optional)	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.

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.

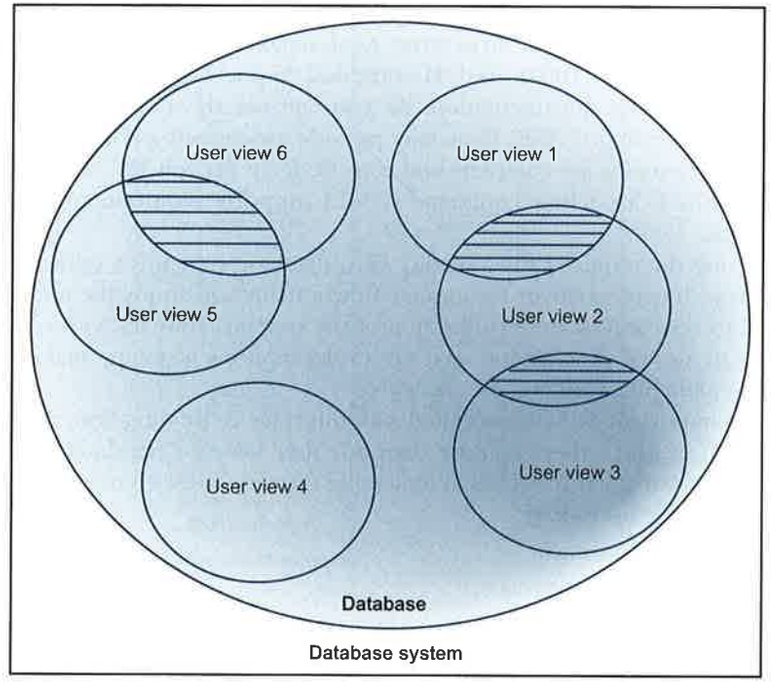

Figure 10.2 Representation of a database system with multiple user views: user views (1, 2, and 3) and (5 and 6) have overlapping requirements (shown as hatched areas), whereas user view 4 has distinct requirements.

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 25.7 we will discuss how the Unified Modeling Language (UML) supports requirements collection and analysis.

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

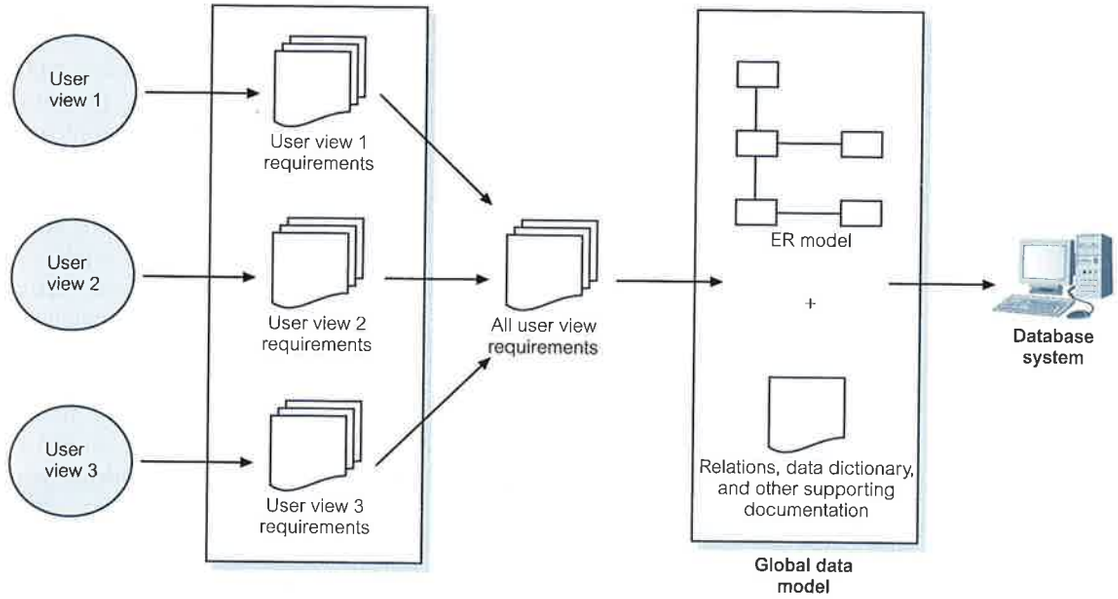

Figure 10.3 The centralized approach to managing multiple user views 1 to 3.

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.

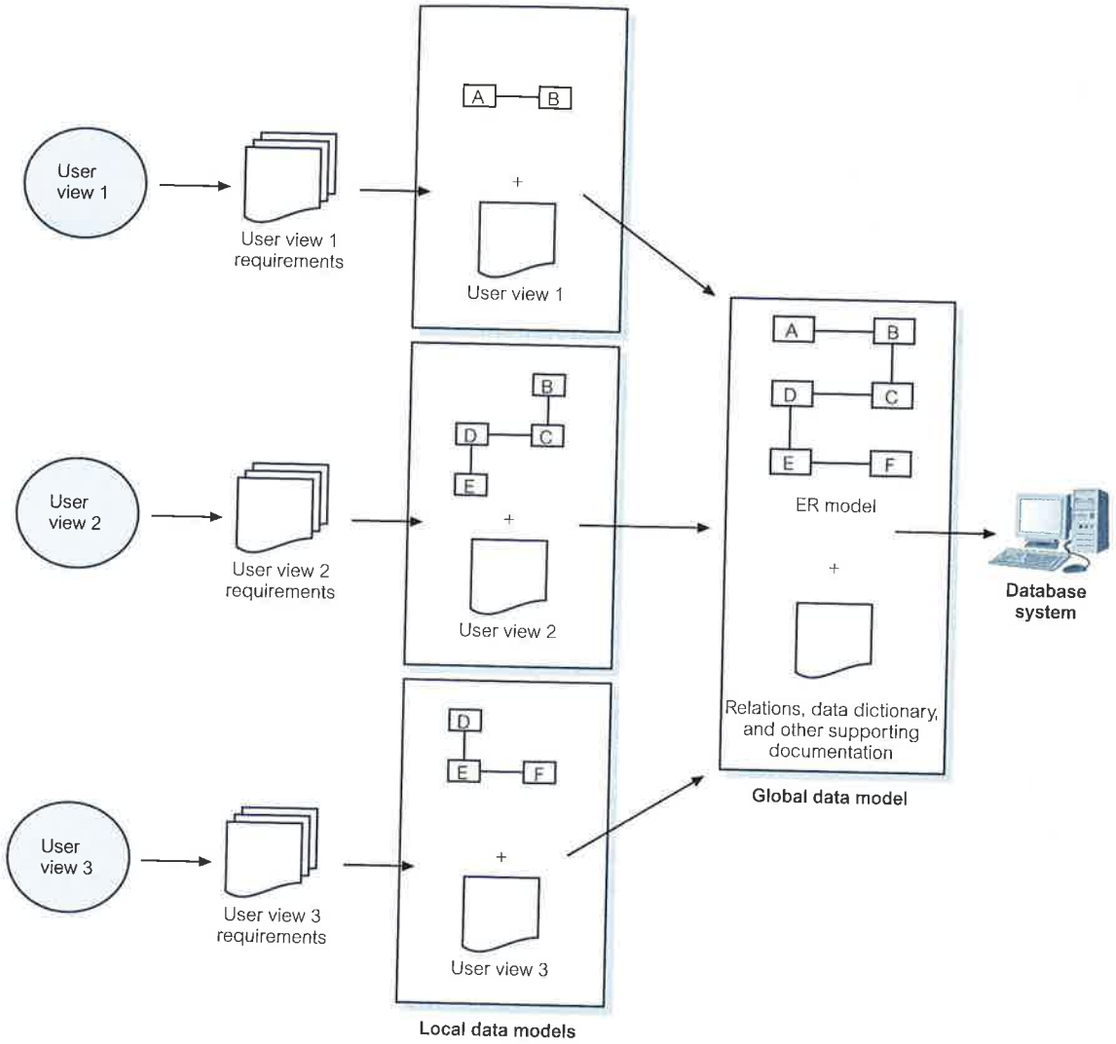

Figure 10.4 The view integration approach to managing multiple user views 1 to 3.

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.

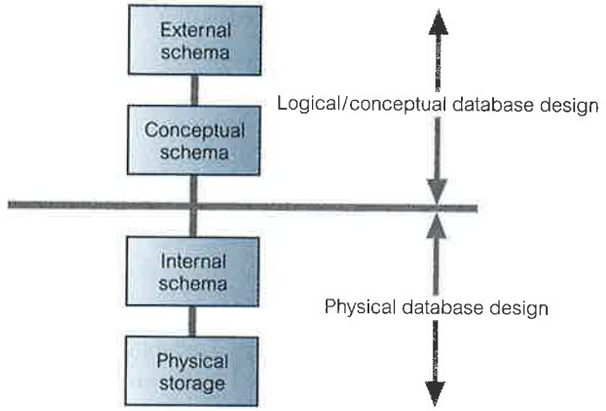

Figure 10.5 Data modeling and the ANSI-SPARC architecture.

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, the DBMS magazine's Web site (available at www.intelligententerprise.com) provides a comprehensive index 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:2008/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 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 and Plaisant, 2004).

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

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

should 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, 2006):

- 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.

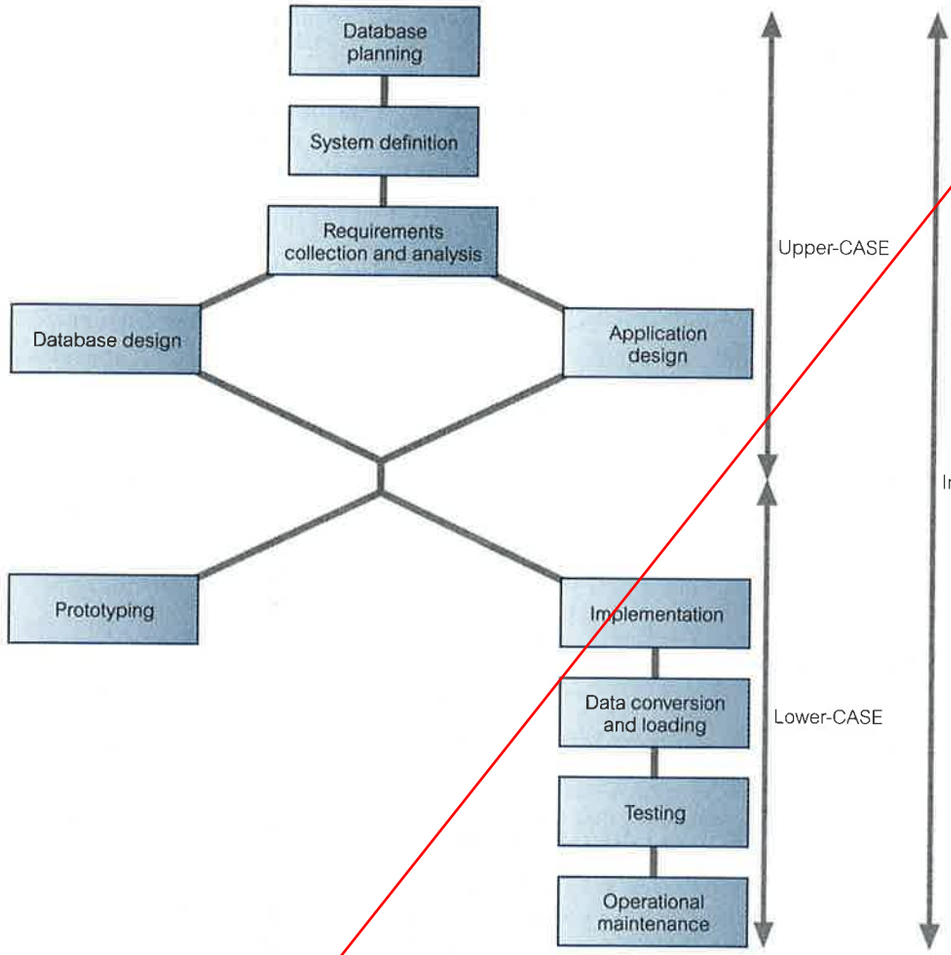

Figure 10.6 Application of CASE tools.

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, as discussed in Section 2.7. 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 Describe the major components of an information system.

10.2 Discuss the relationship between the information systems lifecycle and the database system development lifecycle.

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 for managing the design of a database system that has multiple user views.

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 Describe the main advantages of using the prototyping approach when building a database system.

Exercises

10.12 Assume that you are responsible for selecting a new DBMS product for a group of users in your organization. To undertake this exercise, you must first establish a set of requirements for the group and then identify a set of features that a DBMS product must provide to fulfill the requirements. Describe the process of evaluating and selecting the best DBMS product.

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 11

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 stages 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