1

INTRODUCTION

CHAPTER OUTLINE

Database technology has evolved rapidly in the past three decades since the rise and eventual dominance of relational database systems. While many specialized database systems (object-oriented, spatial, multimedia, etc.) have found substantial user communities in the sciences and engineering, relational systems remain the dominant database technology for business enterprises.

Relational database design has evolved from an art to a science that has been partially implementable as a set of software design aids. Many of these design aids have appeared as the database component of computer-aided software engineering (CASE) tools, and many of them offer interactive modeling capability using a simplified data modeling approach. Logical design—that is, the structure of basic data relationships and their definition in a particular database system—is largely the domain of application designers. The work of these designers can be effectively done with tools such as the ERwin Data Modeler or Rational Rose with Unified Modeling Language (UML), as well as with a purely manual approach. Physical design—the creation of efficient data storage and retrieval mechanisms on the computing platform you are using—is typically the domain of the

database administrator (DBA). Today's DBAs have a variety of vendor-supplied tools available to help design the most efficient databases. This book is devoted to the logical design methodologies and tools most popular for relational databases today. Physical design methodologies and tools are covered in a separate book.

In this chapter, we review the basic concepts of database management and introduce the role of data modeling and database design in the database life cycle.

Data and Database Management

The basic component of a file in a file system is a *data item*, which is the smallest named unit of data that has meaning in the real world—for example, last name, first name, street address, ID number, and political party. A group of related data items treated as a unit by an application is called a *record*. Examples of types of records are order, salesperson, customer, product, and department. A *file* is a collection of records of a single type. Database systems have built upon and expanded these definitions: In a relational database, a data item is called a *column* or *attribute*, a record is called a *row* or *tuple*, and a file is called a *table*.

A *database* is a more complex object; it is a collection of interrelated stored data that serves the needs of multiple users within one or more organizations—that is, an interrelated collection of many different types of tables. The motivation for using databases rather than files has been greater availability to a diverse set of users, integration of data for easier access and update for complex transactions, and less redundancy of data.

A *database management system* (DBMS) is a generalized software system for manipulating databases. A DBMS supports a logical view (schema, subschema); physical view (access methods, data clustering); data definition language; data manipulation language; and important utilities such as transaction management and concurrency control, data integrity, crash recovery, and security. Relational database systems, the dominant type of systems for well-formatted business databases, also provide a greater degree of data independence than the earlier hierarchical and

network (CODASYL) database management systems. *Data independence* is the ability to make changes in either the logical or physical structure of the database without requiring reprogramming of application programs. It also makes database conversion and reorganization much easier. Relational DBMSs provide a much higher degree of data independence than previous systems; they are the focus of our discussion on data modeling.

Database Life Cycle

The database life cycle incorporates the basic steps involved in designing a global schema of the logical database, allocating data across a computer network, and defining local DBMS-specific schemas. Once the design is completed, the life cycle continues with database implementation and maintenance. This chapter contains an overview of the database life cycle, as shown in Figure 1.1. In succeeding chapters we will focus on the database design process from the modeling of requirements through logical design (Steps I and II below). We illustrate the result of each step of the life cycle with a series of diagrams in Figure 1.2. Each diagram shows a possible form of the output of each step so the reader can see the progression of the design process from an idea to an actual database implementation. These forms are discussed in much more detail in Chapters 2–6.

I. ***Requirements analysis.*** The database requirements are determined by interviewing both the producers and users of data and using the information to produce a formal requirements specification. That specification includes the data required for processing, the natural data relationships, and the software platform for the database implementation. As an example, Figure 1.2 (Step I) shows the concepts of products, customers, salespersons, and orders being formulated in the mind of the end user during the interview process.

II. ***Logical design.*** The *global schema*, a conceptual data model diagram that shows all the data and their relationships, is developed using techniques such as entity-relationship (ER) or UML. The data model constructs must be ultimately transformed into tables.

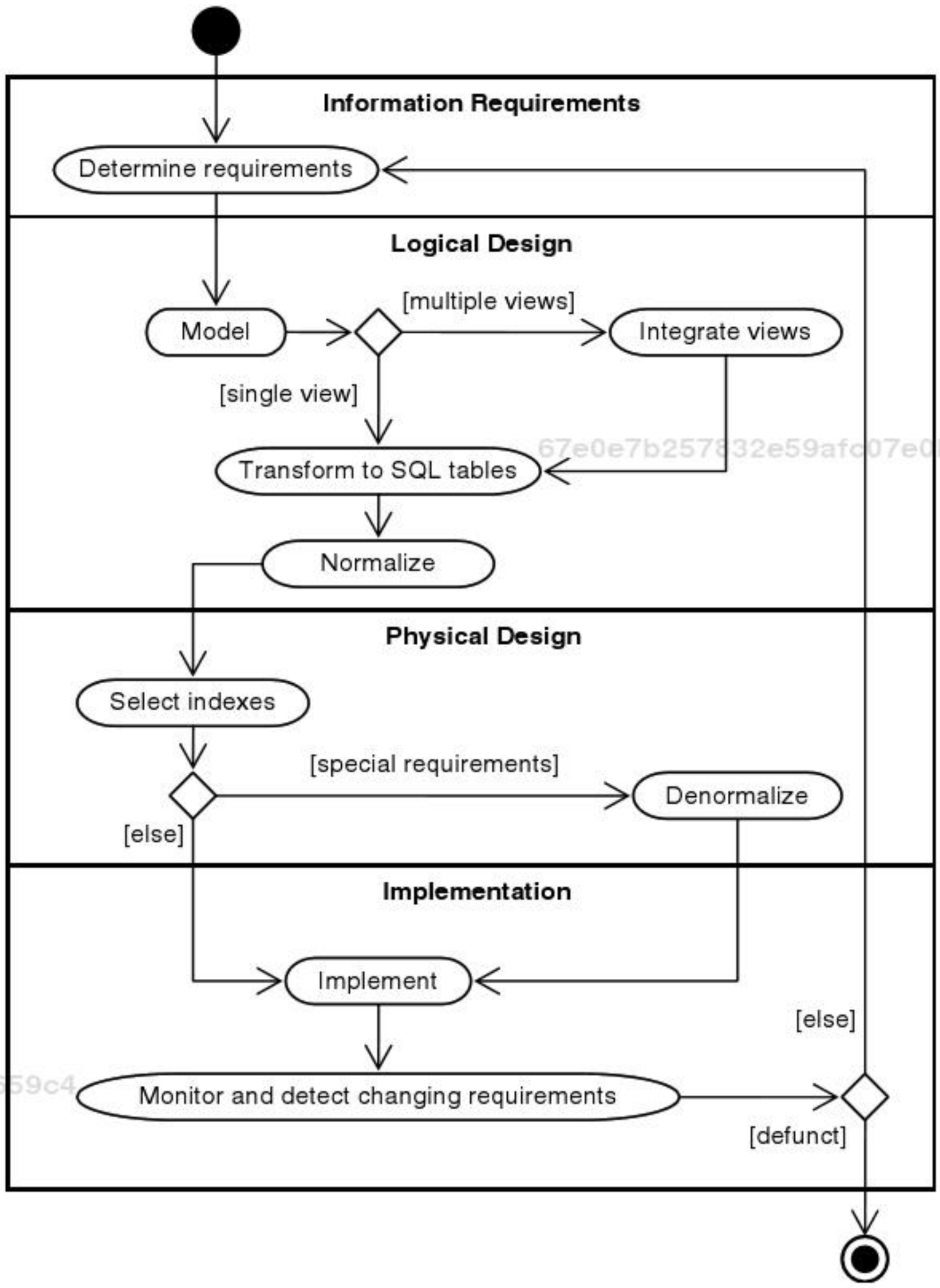
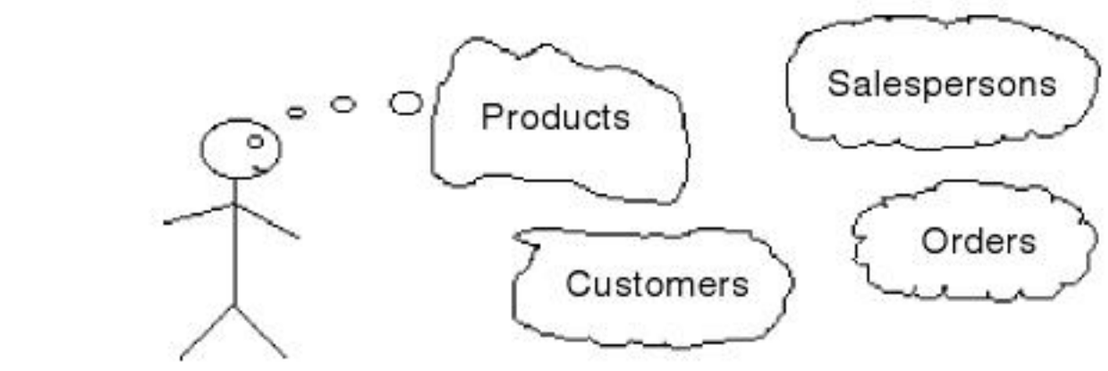


Figure 1.1 The database life cycle.

a. *Conceptual data modeling.* The data requirements are analyzed and modeled by using an ER or UML diagram that includes many features we will study in Chapters 2 and 3, for example, semantics for optional relationships, ternary relationships, supertypes, and subtypes (categories). Processing requirements are typically specified using natural language expressions or SQL commands along with the frequency of occurrence. Figure 1.2 (Step II.a) shows a possible ER



Integration of retail salesperson's and customer's views

Figure 1.2 Life cycle results, step by step (continued on following page).

model representation of the product/customer database in the mind of the end user.

b. *View integration.* Usually, when the design is large and more than one person is involved in requirements analysis, multiple views of data and relationships occur, resulting in inconsistencies due to variance in taxonomy, context, or perception. To eliminate redundancy and inconsistency from the model, these views must

Step II.c Transformation of the conceptual data model to SQL tables

Customer

cust-no	cust-name	.....

Product

prod-no	prod-name	qty-in-stock

Salesperson

sales-name	addr	dept	job-level	vacation-days

Order

order-no	sales-name	cust-no

Order-product

order-no	prod-no

```
create table customer
    (cust_no integer,
    cust_name char(15),
    cust_addr char(30),
    sales_name char(15),
    prod_no integer,
    primary key (cust_no),
    foreign key (sales_name)
        references salesperson,
    foreign key (prod_no)
        references product):
```

Step II.d Normalization of SQL tables

Decomposition of tables and removal of update anomalies.

Salesperson

sales-name	addr	dept	job-level

SalesVacations

job-level	vacation-days

Step III Physical Design

Indexing
Clustering
Partitioning
Materialized views
Denormalization

Figure 1.2, cont'd
Further life cycle results, step by step.

be "rationalized" and consolidated into a single global view. View integration requires the use of ER semantic tools such as identification of synonyms, aggregation, and generalization. In Figure 1.2 (Step II.b), two possible views of the product/customer database are merged into a single global view based on common data for customer and order. View integration is also important when applications have to be integrated, and each may be written with its own view of the database.

c. *Transformation of the conceptual data model to SQL tables.* Based on a categorization of data modeling constructs and a set of mapping rules, each relationship and its associated entities are transformed into a set of DBMS-specific candidate relational tables. We will show these transformations in standard SQL in Chapter 5. Redundant tables are eliminated as part of this process. In our example, the tables in Step II.c of Figure 1.2 are the result of transformation of the integrated ER model in Step II.b.

d. *Normalization of tables.* Given a table (R), a set of attributes (B) is functionally dependent on another set of attributes (A) if, at each instant of time, each A value is associated with exactly one B value. Functional dependencies (FDs) are derived from the conceptual data model diagram and the semantics of data relationships in the requirements analysis. They represent the dependencies among data elements that are unique identifiers (keys) of entities. Additional FDs, which represent the dependencies between key and nonkey attributes within entities, can be derived from the requirements specification. Candidate relational tables associated with all derived FDs are normalized (i.e., modified by decomposing or splitting tables into smaller tables) using standard normalization techniques. Finally, redundancies in the data that occur in normalized candidate tables are analyzed further for possible elimination, with the constraint that data integrity must be preserved. An example of normalization of the Salesperson table into the new Salesperson and SalesVacations tables is shown in Figure 1.2 from Step II.c to Step II.d.

We note here that database tool vendors tend to use the term *logical model* to refer to the conceptual data model, and they use the term *physical model* to refer to the DBMS-specific implementation model (e.g., SQL tables). We also note that many conceptual data models are obtained not from scratch, but from the process of *reverse engineering* from an existing DBMS-specific schema (Silberschatz et al., 2010).

III. ***Physical design.*** The physical design step involves the selection of indexes (access methods), partitioning, and clustering of data. The logical design methodology in Step II simplifies the approach to designing large relational databases by reducing the number of data dependencies that need to be analyzed. This is accomplished by inserting the conceptual data modeling and integration steps (Steps II.a and II.b of Figure 1.2) into the traditional relational design approach. The objective of these steps is an accurate representation of reality. Data integrity is preserved through normalization of the candidate tables created when the conceptual data model is transformed into a relational model. The purpose of physical design is to then optimize performance. As part of the physical design, the global schema can sometimes be refined in limited ways to reflect processing (query and transaction) requirements if there are obvious large gains to be made in efficiency. This is called *denormalization*. It consists of selecting dominant processes on the basis of high frequency, high volume, or explicit priority; defining simple extensions to tables that will improve query performance; evaluating total cost for query, update, and storage; and considering the side effects, such as possible loss of integrity. This is particularly important for online analytical processing (OLAP) applications.

IV. ***Database implementation, monitoring, and modification.*** Once the design is completed, the database can be created through implementation of the formal schema using the data definition language (DDL) of a DBMS. Then the data manipulation language (DML) can be used to query and update the database, as well as to set up indexes and establish constraints, such as referential integrity. The language SQL contains both DDL and DML constructs; for example, the *create table* command represents DDL, and the *select* command represents DML.

As the database begins operation, monitoring indicates whether performance requirements are being met. If they are not being satisfied, modifications should be made to improve performance. Other modifications may be necessary when requirements change or end

user expectations increase with good performance. Thus, the life cycle continues with monitoring, redesign, and modifications. In the next two chapters we look first at the basic data modeling concepts; then, starting in Chapter 4, we apply these concepts to the database design process.

Conceptual Data Modeling

Conceptual data modeling is the driving component of logical database design. Let us take a look of how this important component came about and why it is important. Schema diagrams were formalized in the 1960s by Charles Bachman. He used rectangles to denote record types and directed arrows from one record type to another to denote a one-to-many relationship among instances of records of the two types. The *entity-relationship* (ER) approach for conceptual data modeling, one of the two approaches emphasized in this book, and described in detail in Chapter 2, was first presented in 1976 by Peter Chen. The Chen form of ER models uses rectangles to specify entities, which are somewhat analogous to records. It also uses diamond-shaped objects to represent the various types of relationships, which are differentiated by numbers or letters placed on the lines connecting the diamonds to the rectangles.

The Unified Modeling Language (UML) was introduced in 1997 by Grady Booch and James Rumbaugh and has become a standard graphical language for specifying and documenting large-scale software systems. The data modeling component of UML (now UML-2) has a great deal of similarity with the ER model, and will be presented in detail in Chapter 3. We will use both the ER model and UML to illustrate the data modeling and logical database design examples throughout this book.

In conceptual data modeling, the overriding emphasis is on simplicity and readability. The goal of conceptual schema design, where the ER and UML approaches are most useful, is to capture real-world data requirements in a simple and meaningful way that is understandable by both the database designer and the end user. The end user is the person responsible for accessing the database and

executing queries and updates through the use of DBMS software, and therefore has a vested interest in the database design process.

Summary

Knowledge of data modeling and database design techniques is important for database practitioners and application developers. The database life cycle shows what steps are needed in a methodical approach to designing a database, from logical design, which is independent of the system environment, to physical design, which is based on the details of the database management system chosen to implement the database. Among the variety of data modeling approaches, the ER and UML data models are arguably the most popular in use today because of their simplicity and readability.

Tips and Insights for Database Professionals

Tip 1. Work methodically through the steps of the life cycle. Each step is clearly defined and has produced a result that can serve as a valid input to the next step.

Tip 2. Correct design errors as soon as possible by going back to the previous step and trying new alternatives. The later you wait, the more costly the errors and the longer the fixes.

Tip 3. Separate the logical and physical design completely because you are trying to satisfy completely different objectives.

Logical design. The objective is to obtain a feasible solution to satisfy all known and potential queries and updates. There are many possible designs; it is not necessary to find a "best" logical design, just a feasible one. Save the effort for optimization for physical design.

Physical design. The objective is to optimize performance for known and projected queries and updates.

Literature Summary

Much of the early data modeling work was done by Bachman (1969, 1972), Chen (1976), Senko et al. (1973), and others. Database design textbooks that adhere to a significant portion of the relational database life cycle described in this chapter are Teorey and Fry (1982), Muller (1999), Stephens and Plew (2000), Silverston (2001), Harrington (2002), Bagui (2003), Hernandez and Getz (2003), Simsion and Witt (2004), Powell (2005), Ambler and Sadalage (2006), Scamell and Umanath (2007), Halpin and Morgan (2008), Mannino (2008), Stephens (2008), Churcher (2009), and Hoberman (2009).

Temporal (time-varying) databases are defined and discussed in Jenson and Snodgrass (1996) and Snodgrass (2000). Other well-used approaches for conceptual data modeling include IDEF1X (Bruce, 1992; IDEF1X, 2005) and the data modeling component of the Zachmann Framework (Zachmann, 1987; Zachmann Institute for Framework Advancement, 2005). Schema evolution during development, a frequently occurring problem, is addressed in Harriman, Hodgetts, and Leo (2004).

This page intentionally left blank