Database Modeling & Design

Third Edition

TOBY J. TEOREY

Morgan Kaufmann Publishers, Inc.
San Francisco, California

design problems we might encounter in industry and small businesses. We will also see that the simple form is easily translatable into SQL data definitions, and thus it has an immediate use as an aid for database implementation.

The complex level of ER model definition includes concepts that go far beyond the original model. It includes concepts from the semantic models of artificial intelligence and from competing conceptual data models such as the binary relationship model and NIAM methodology (see Chapter 2 for an overview of some NIAM concepts). ER modeling at this level helps the database designer capture more semantics without having to resort to narrative explanations. It is also useful to the database application programmer because certain integrity constraints defined in the ER model relate directly to code—code that checks range limits on data values and null values, for example. However, such detail in very large ER diagrams actually detracts from end-user understanding. Therefore, the simple level is recommended as a communication tool for database design verification.

1.3 The 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. In succeeding chapters we will focus on the database design process from the modeling of requirements through distributed data allocation (steps I through IV below). We illustrate the result of each step of the life cycle with a series of diagrams in Figures 1.2 through 1.4. 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 actual database implementation. These forms are discussed in much more detail in succeeding chapters.

I. ***Requirements analysis.*** The database requirements are determined by interviewing both the producers and users of data and producing 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

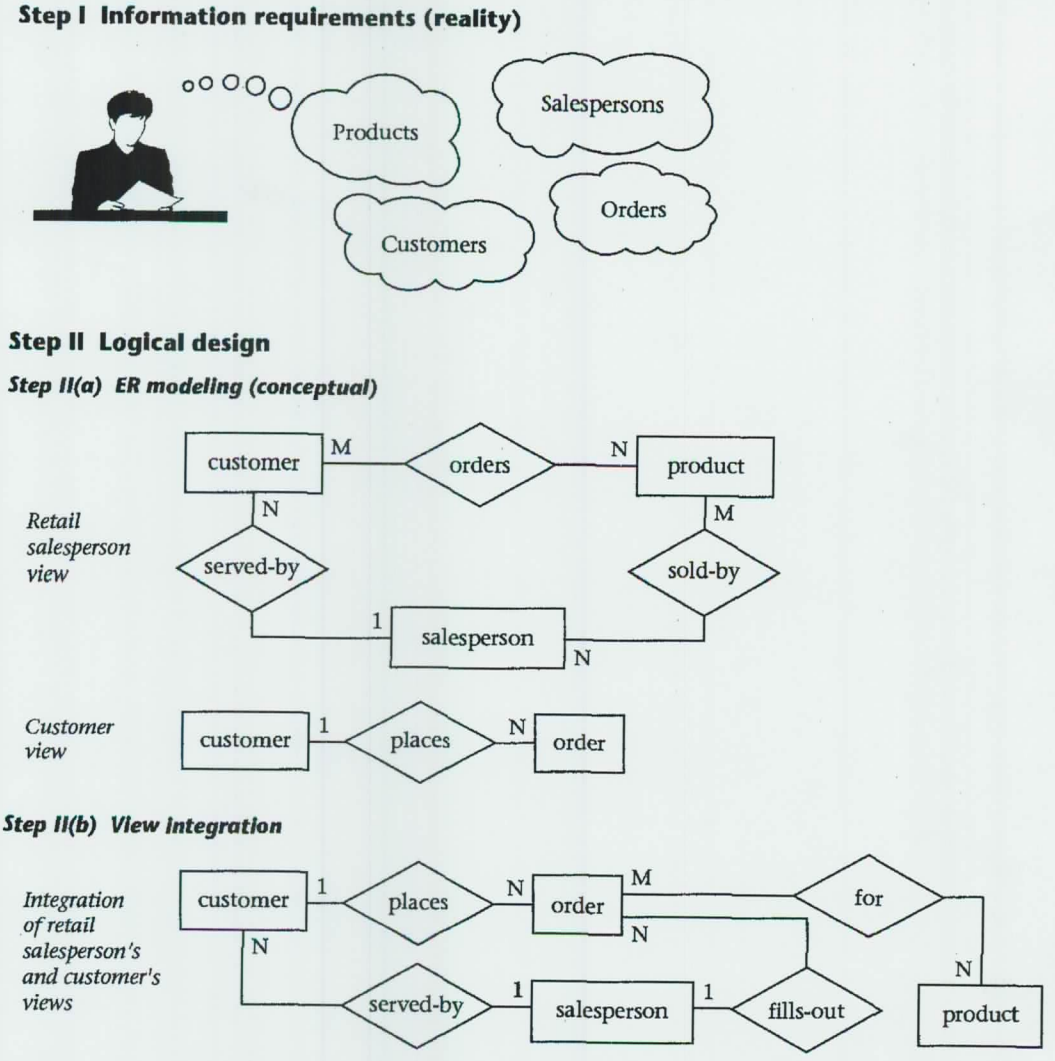

Figure 1.2 Database life cycle: requirements and logical design

orders being formulated in the mind of the end user during the interview process.

II. *Logical design.* The *global schema*, which shows all the data and their relationships, is developed using conceptual data modeling techniques such as ER. The data model constructs must ultimately be transformed into normalized (global) relations, or tables. The global schema development methodology is the same for either a distributed or centralized database.

a. *ER modeling.* The data requirements are analyzed and modeled by using an ER diagram that includes, 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 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 result. To eliminate redundancy and inconsistency from the model, these views must eventually be 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.

c. *Transformation of the ER model to SQL tables*. Based on a categorization of ER constructs and a set of mapping rules, each relationship and its associated entities are transformed into a set of DBMS-specific candidate relational tables, called the *external user schema*. We will show these transformations in standard SQL in Chapter 4. Redundant tables are eliminated as part of this process. In our example, the tables in Figure 1.3 are the result of transformation of the integrated ER model in Figure 1.2.

d. *Normalization of tables*. Functional dependencies (FDs) are derived from the ER diagram and the semantics of data relationships in the requirements analysis. They represent the dependencies among data elements that are keys of entities. Additional FDs and multivalued dependencies (MVDs), which represent the dependencies among key and nonkey attributes within entities, can be derived from the requirements specification. Candidate relational tables associated with all derived FDs and MVDs are normalized (i.e., modified by decomposing or splitting tables into smaller tables) to the highest degree desired 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 Sales-vacations tables from step II(c) to step II(d) is shown in Figure 1.3.

III. ***Physical design.*** The last step in the design phase for centralized databases is to produce a physical structure for the database. The physical design step involves the selection of indexes (access methods) and clustering of data. The logical design methodology in step II simplifies the approach to designing large relational databases by reducing the number

Step II(c) Transformation of the ER diagram 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 (3NF, BCNF, 4NF, 5NF)

Decomposition of tables and removal of update anomalies

Salesperson

sales-name	addr	dept	job-level

Sales-vacations

job-level	vacation-days

Step III Physical design (including denormalization)

Customer

cust-no	cust-name

Customer / refined

cust-no	cust-name	sales-name

Order

order-no	sales-name	cust-no

Physical design parameters: indexing, access methods, clustering

Figure 1.3 Database life cycle: ER-to-SQL, normalization, and physical design

of data dependencies that need to be analyzed. This is accomplished by inserting ER modeling and integration steps (steps II(a) and II(b)) 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 ER model is transformed into a relational model. The purpose of physical design is to then optimize performance as closely as possible.

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. Denormalization is illustrated in Figure 1.3 (step III), with the extension of the Customer table to include "sales-name" for greater efficiency with database queries such as "Who is the salesperson for a customer named Pamela Wilson?"

IV. ***Data distribution.*** Data fragmentation and allocation are also forms of physical design because they must take into account the physical environment, that is, the network configuration. This applies only to distributed databases.

A *fragmentation schema* describes the one-to-many mapping used to partition each global table into fragments. Fragments are logical portions of global tables that are physically located at one or several sites of the network. A data *allocation schema* designates where each copy of each fragment is to be stored. A one-to-one mapping in the allocation schema results in nonredundancy; a one-to-many mapping defines a replicated distributed database. An example of the allocation of a fragmented and partially replicated database is shown in Figure 1.4, where various subsets of the whole database are allocated to different sites in a computer network.

Three important objectives of data distribution are

- the separation of data fragmentation and allocation,
- control of redundancy, and
- independence from local database management systems.

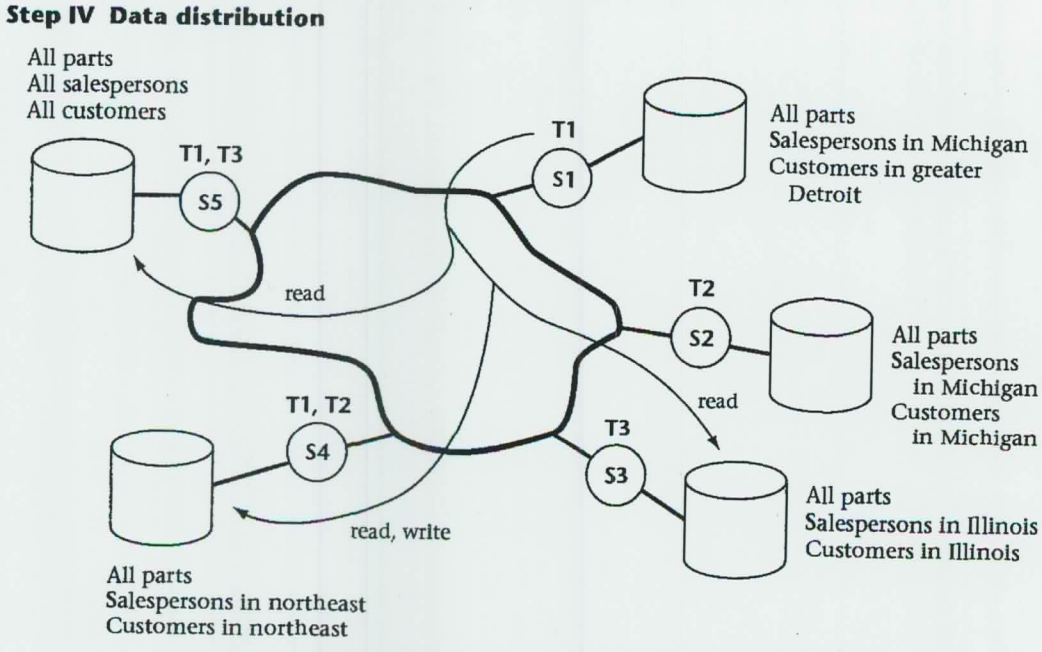

Figure 1.4 Database life cycle: data distribution

The distinction between designing the fragmentation and allocation schema is important: The first one is a logical, the second a physical mapping. In general, it is not possible to determine the optimal fragmentation and allocation by solving the two problems independently since they are interrelated. However, near-optimal solutions can be obtained with separable design steps, and this is the only practical solution available today.

V. ***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.

Most CASE tools available today focus on steps II, III, and V of the database life cycle. Within step II, the ER modeling, view integration, transformation to SQL, and normalization substeps are all commonly supported by many (but not all) tools. Within step III, transaction modeling has become commonplace, and automatic application generation is provided by some tools. An excellent survey of both CASE tools that offer database design and purely database design tools can be found in [BCN92]. In the next chapter we look first at the basic data modeling concepts, then—starting in Chapter 3—we apply these concepts to the database design process. The detailed steps in the database life cycle are illustrated using examples taken from real-life databases.

1.4 Summary

Knowledge of data modeling and database design techniques is important for database practitioners. Among the variety of data modeling approaches, the ER model is arguably the most popular in use today because of its simplicity and readability. A simple form of the ER model is used in most CASE tools and is easy to learn and apply to a variety of industrial and business applications. It is also a very useful tool for communicating with the end user about the conceptual model and for verifying the assumptions made in the modeling process. A more complex form, a superset of the simple form, is useful for the more experienced designer who wants to capture greater semantic detail in diagram form and avoid having to write long and tedious narrative to explain certain requirements and constraints over and over again.

The database life cycle shows what steps are needed in a methodical approach to database design from logical design, which is independent of the system environment, to local physical design, which is based on the details of the database management system chosen to implement the database, and to data distribution in a computer network.