# 2. Design Objectives

*Everything factual is, in a sense, theory. The blue of the sky exhibits the basic laws of chromatics. There is no sense in looking for something behind phenomena; they are theory.*
—GOETHE

## TOPICS COVERED IN THIS CHAPTER



## WHY SHOULD YOU BE CONCERNED WITH DATABASE DESIGN?

Some of you who work with relational database management system (RDBMS) application programs may wonder why you should be concerned with database design. After all, most programs come with sample databases that you can copy and modify to suit your own needs, and you can even borrow tables from the sample databases and use them in other databases that you've created. Some programs also provide tools that will guide you through the process of defining and creating tables. However, these tools don't actually help you *design* a database—they merely help you create the physical tables that you will include in the database.

What you must understand is that it's better for you to use these tools *after you've created the logical database structure.* RDBMS programs provide the design tools and the sample databases to help minimize the time it takes you to *implement* the database structure physically. Theoretically, reducing implementation time gives you more time to focus on creating and building end-user applications.

Yet the primary reason you should be concerned with database design is that it is crucial to the consistency, integrity, and accuracy of the data in a database. If you design a database improperly, it will be difficult for you to retrieve certain types of information, and you'll run the risk that your searches will produce inaccurate information. *Inaccurate information is probably the most detrimental result of improper database design—it can adversely affect your organization's bottom line.* In fact, if your database affects the manner in which your business performs its daily

operations or if it's going to influence the future direction of your business, you *must* be concerned with database design.

Let's look at this from a different perspective for a moment: Think about how you would go about having a custom home built. What's the first thing you're going to do? Certainly you're not going to hire a contractor immediately and let him build your home however he wishes. *Surely* you will first engage an architect to design your new home *and then* hire a contractor to build it. The architect will explore your needs and express them as a set of blueprints, recording decisions about size and shape and requirements for various systems (structural, mechanical, electrical). Next, the contractor will procure the labor and materials, including the listed systems, and then assemble them according to the drawings and specifications.

Now let's return to our database perspective and think of the logical database design as the architectural blueprints and the physical database implementation as the completed home. The logical database design describes the size, shape, and necessary systems for your database and it addresses the informational and operational needs of your business. You then build the physical implementation of the logical database design using your RDBMS program. Once you've created your tables, set up table relationships, and established the appropriate levels of data integrity, your database is complete. Now you're ready to design and create applications that allow you and your users to interact easily with the data stored in the database, and you can be confident that these applications will provide you with timely and, above all, accurate information.

Although you can implement a poor design in an RDBMS, implementing a good design is far more to your advantage because it will yield accurate information, store data more efficiently and effectively, and be easier for you to manage and maintain.

## THE IMPORTANCE OF THEORY

Note

In this chapter, I use the term *theory* to represent "general propositions used as principles" and not "conjectures or proposals."

A number of major disciplines (and their associated design methodologies) have some type of theoretical basis. Structural engineers design an unlimited variety of structures using the theories of physics. Composers create beautiful symphonies and orchestral pieces using the concepts found in music theory. The automobile industry uses aerodynamics theories to design more fuel-efficient vehicles. The aerospace industry uses the same theories to design airplane wings that reduce wind drag.

These examples demonstrate that theory is relevant and very important. The chief advantage of theory is that it helps you predict outcomes; it allows you to predict what will happen if you perform a certain action or series of actions. You know if you drop a stone, it will fall to the ground. If you are agile, you can get your toes out of the way of Newton's theory of gravity. The point is that it works *every time*. If you chisel a stone flat and place it on another flat stone, you can predict that it will stay where you put it. This theory allows you to design pyramids and cathedrals and brick outhouses. Now

consider a database example. Let's assume you have a pair of tables that are related to each other. You know that you can draw data from both tables simultaneously simply because of the way relational database theory works. The data you draw from both tables is based on matching values of a shared field between the tables themselves. Again, your actions have a predictable result.

The relational database is based on two branches of mathematics known as *set theory* and *first-order predicate logic*. This very fact is what allows the relational database to guarantee accurate information. These branches of mathematics also provide the basis for formulating good design methodologies and the building blocks necessary to create good relational database structures.

You might harbor an understandable reluctance to study complicated mathematical concepts simply to carry out what seems to be a rather limited task. You're still sure to hear claims that the mathematical theories on which the relational database and its associated design methodologies are based don't have any relevance to the real world, or that they are somehow impractical. This is not true: Math is centralto the relational model and is what guarantees the model's viability. But cheer up—it isn't really necessary for you to know anything about set theory or first-order predicate logic in order to use a relational database! You certainly don't have to know all the details of aerodynamics just to drive an automobile. Aerodynamics theories may help you understand and appreciate how an automobile can get better gas mileage, but they won't help you learn how to parallel park.

Mathematical theory provides the foundation for the relational database model, and thus makes the model predictable, reliable, and sound. Theory describes the basic building blocks used to create a relational database and provides guidelines for how it should be arranged. Arranging building blocks to achieve a desired result is defined as "design."

## THE ADVANTAGE OF LEARNING A GOOD DESIGN METHODOLOGY

You could learn how to design a database properly by trial and error, but it would take you a very long time and you would probably have to repair many mistakes along the way. The best approach is to learn a good database design methodology, such as the one in this book, and then embark on designing your database.

You'll gain several advantages from learning and using a good design methodology.

• *It gives you the skills you need to design a sound database structure.* A large number of data processing problems can be attributed to the presence of redundant data, duplicate data, and invalid data, or the absence of required data. All of these problems produce erroneous information and make certain queries and reports difficult to run. You can avoid almost all of these problems by employing a good design methodology.

• *It provides you with an organized set of techniques that will guide you step-by-step through the design process.* The organization of the techniques enables you to make informed decisions on every aspect of your design.

• *It helps you keep your missteps and design reiterations to a minimum.* Of course, you will naturally make *some* mistakes when you're designing a database, but a good

methodology helps you recognize errors in your design and gives you the tools to correct them. Additionally, the organization of the techniques within the methodology keeps you from unnecessarily repeating a given design process.

• *It makes the design process easier and reduces the amount of time you spend designing the database.* You will inevitably waste valuable time taking an arbitrary trial-and-error approach to design because it lacks the logic and organization that a good methodology provides.

• *It will help you understand and use your RDBMS application program more fully and effectively*.As your knowledge of proper design expands and grows, you'll actually begin to understand *why* a given RDBMS provides certain tools and *how* you can use them to implement the structure within the RDBMS program.

Regardless of whether you use the design methodology presented in this book or some other established methodology, you should choose a design methodology, learn it as well as you can, and use it faithfully to design your databases.

## OBJECTIVES OF GOOD DESIGN

There are distinct objectives you must achieve in order to design a good, sound database structure. You can avoid many of the problems mentioned in the previous section if you keep these objectives in mind and constantly focus on them while you're designing your database.

• *The database supports both required and ad hoc information retrieval.* The database must store the data necessary to support information requirements defined during the design process and any possible ad hoc queries that may be posed by a user.

• *The tables are constructed properly and efficiently.* Each table in the database represents a single subject, is composed of relatively distinct fields, keeps redundant data to an absolute minimum, and is identified throughout the database by a field with unique values.

• *Data integrity is imposed at the field, table, and relationship levels.* These levels of integrity help guarantee that the data structures and their values will be valid and accurate at all times.

• *The database supports business rules relevant to the organization.* The data must provide valid and accurate information that is always meaningful to the business.

• *The database lends itself to future growth.* The database structure should be easy to modify or expand as the information requirements of the business change and grow.

You might find it difficult at times to fulfill these objectives, but you'll certainly be pleased with your final database structure once you've met them.

## BENEFITS OF GOOD DESIGN

The time you invest in designing a sound database structure is time well spent. Good design *saves* you time in the long run because you do not constantly have to revamp a quickly and poorly designedstructure. You gain the following benefits when you apply good design techniques.

• *The database structure is easy to modify and maintain.* Modifications you make to a field or table will not adversely affect other fields or tables in the database.

• *The data is easy to modify.* Changes you make to the value of a given field in a table will not adversely affect the values of other fields within the table. Furthermore, a well-designed database keeps duplicate fields to an absolute minimum, so you typically modify a particular data value in one field only.

• *Information is easy to retrieve.* You'll be able to create queries easily because the tables are well constructed and the relationships between them are properly established.

• *End-user applications are easy to develop and build.* You can spend more time on programming and addressing the data manipulation tasks at hand, instead of working around the inevitable problems that arise when you work with a poorly designed database.

## DATABASE DESIGN METHODS

### Traditional Design Methods

In general, traditional methods of database design incorporate three phases: *requirements analysis, data modeling,* and *Normalization*.

The requirements analysis phase involves an examination of the business being modeled, interviews with users and management to assess the current system and to analyze future needs, and an assessment of information requirements for the business as a whole. This process is relatively straightforward, and, indeed, the design process presented in this book follows the same line of thinking.

The data modeling phase involves modeling the database structure using a data modeling method, such as entity-relationship (ER) diagramming, semantic-object modeling, object-role modeling, or UML modeling. Each of these modeling methods provides a means of visually representing various aspects of the database structure, such as the tables, table relationships, and relationship characteristics. In fact, the modeling method used in this book is a basic version of ER diagramming. Figure 2.1 shows an example of a basic ER diagram.

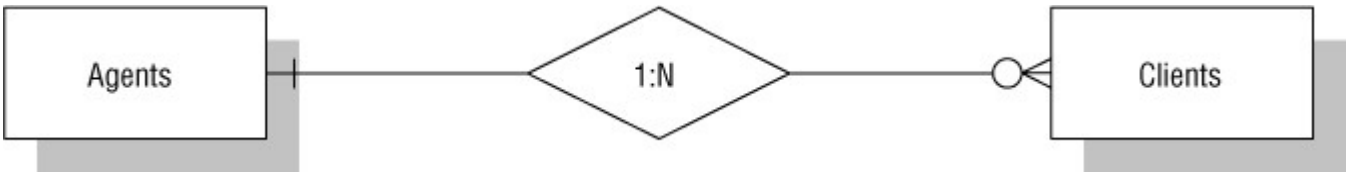


**Figure 2.1. An example of a basic ER diagram**

Note

I've incorporated the data modeling method I use in this book into the design process itself rather than treating it separately. I'll introduce and explain each modeling technique as appropriate throughout the process.

Each data modeling method incorporates a set of diagramming symbols used to represent a database's structure and characteristics. For example, the diagram in Figure 2.1 provides information on several aspects of the database.

• The rectangles represent two tables called AGENTS and CLIENTS.

• The diamond represents a relationship between these two tables, and the "1:N" within the diamond indicates that it is a one-to-many relationship.

• The vertical line next to the AGENTS table indicates that a client must be associated with an agent, and the circle next to the CLIENTS table indicates that an agent doesn't necessarily have to be associated with a client.

Fields are also defined and associated with the appropriate tables during the data modeling phase. Each table is assigned a *primary key,* various levels of data integrity are identified and implemented, and relationships are established via *foreign keys*. Once the initial table structures are complete and the relationships have been established according to the data model, the database is ready to go through the Normalization phase.

Normalization is the process of decomposing large tables into smaller ones in order to eliminate redundant data and duplicate data, and to avoid problems with inserting, updating, or deleting data. During the Normalization process, table structures are tested against *normal forms* and then modified if any of the aforementioned problems are found. A normal form is a specific set of rules that can be used to test a table structure to ensure that it is sound and free of problems. There are a number of normal forms, and each one is used to test for a particular set of problems. The normal forms currently in use are First Normal Form, Second Normal Form, Third Normal Form, Fourth Normal Form, Fifth Normal Form, Sixth Normal Form, Boyce-Codd Normal Form, and Domain/Key Normal Form.

**The Design Method Presented in This Book**

The design method that I use in this book is one that I've developed over the years. It incorporates a requirements analysis and a simple ER diagramming method to diagram the database structure. However, it *does not* incorporate the traditional Normalization process or involve the use of normal forms. The reason is simple: Normal forms can be confusing to anyone who has not taken the time to study formal relational database theory. For example, examine the following definition of Third Normal Form:

A relvar is in 3NF if and only if it is in 2NF and every non-key attribute is nontransitively dependent on the primary key.[1]

1. C. J. Date, *An Introduction to Database Systems,* 7th ed. (Boston: Addison-Wesley, 2000), 362; emphasis added.

This description is relatively meaningless to a reader who is unfamiliar with the terms *relvar, 3NF, 2NF, non-key attribute, transitively dependent,* and *primary key*.

The process of designing a database is not and should not be hard to understand. As long as the process is presented in a straightforward manner and each concept or technique is clearly explained, anyone should be able to design a database properly. For

example, the following definition is derived from the*results* of using Third Normal Form against a table structure, and I believe most people will find it clear and easy to understand:

A table should have a field that uniquely identifies each of its records, and each field in the table should describe the subject that the table represents.

The process I used to formulate this definition is the same one I used to develop my entire design methodology.

## NORMALIZATION

Back in the late 1980s, it occurred to me that the relational model had been in existence for almost 20 years and that people had been designing databases using the same basic methodology for about twelve years. (And I'm still surprised we're using it some 20+ years later.) I was using the traditional design methodology at that time, but I occasionally found it difficult to employ. The two things that bothered me the most about it were the Normalization process (as a whole) and the seemingly endless iterations it took to arrive at a proper design. Of course, these seemed to be sore points with most of the other database developers that I knew, so I certainly wasn't alone in my frustrations. I thought about these problems for quite some time, and then I came up with a solution.

I already knew that the purpose of Normalization is to take an improperly or poorly designed table and transform it into a table with a sound structure. I also understood the process: Take a given table and test it against the normal forms to determine whether it is properly designed. If it isn't designed properly, make the appropriate modifications, retest it, and repeat the entire process until the table structure is sound. Figure 2.2 shows how I visualized the process at this point.

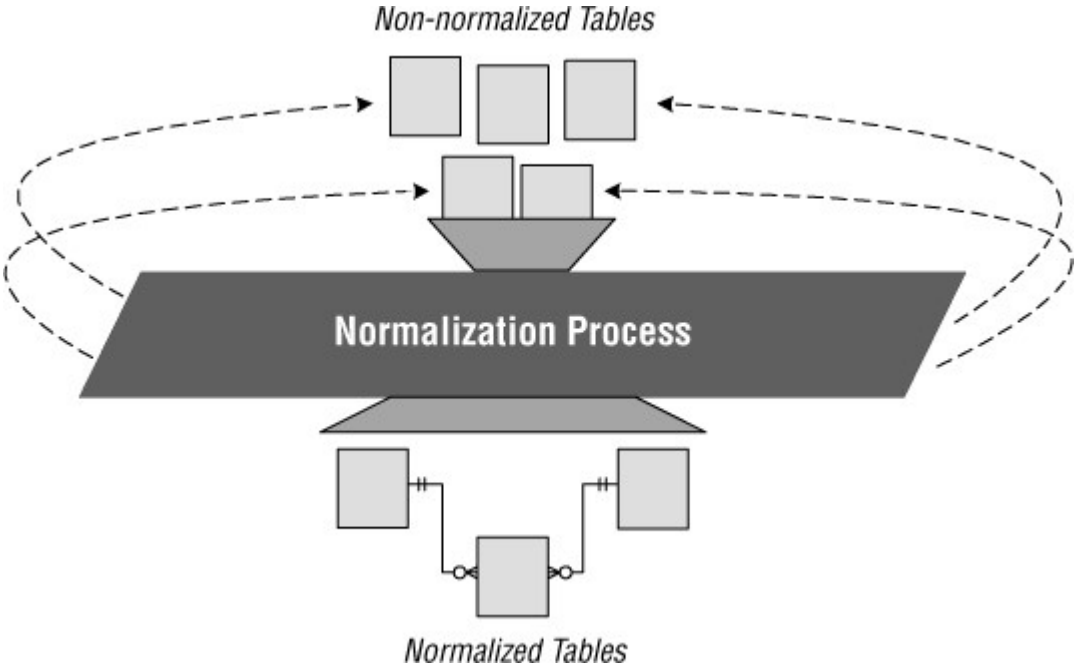


**Figure 2.2. How I viewed the general Normalization process**

I kept these facts in mind and then posed the following questions.

**1.** If we assume that a thoroughly normalized table is properly and efficiently designed, couldn't we identify the specific characteristics of such a table and state these to be the attributes of an ideal table structure?

**2.** Couldn't we then use that ideal table as a model for all tables we create for the database throughout the design process?

The answer to both questions, of course, is yes, so I began in earnest to develop the basis for my "new" design methodology. I first compiled distinct sets of guidelines for creating sound structures by identifying the final characteristics of a well-defined database that *successfully* passed the tests of *each* normal form. I then conducted a few tests, using the new guidelines to create table structures for a new database and to correct flaws in the table structures of an existing database. These tests went very well, so I decided to apply this technique to the entire traditional design methodology. I formulated guidelines to address other issues associated with the traditional design method, such as domains, subtypes, relationships, data integrity, and referential integrity. After I completed the new guidelines, I performed more tests and found that my methodology worked quite well.

The main advantage of my design methodology is that it removes many aspects of the traditional design methodology that new database developers find intimidating. For example, Normalization, in the traditional sense, is now transparent to the developer because it is incorporated (via the new guidelines) throughout the design process. Another major advantage is that the methodology is clear and easy to implement. I believe much of this is due to the fact that I've written all the guidelines in plain English, making them easy for most anyone to understand.

It's important for you to understand that this design methodology will yield a fully normalized database structure *only if you follow it as faithfully as you would any other design methodology*. You cannot shortcut, circumvent, de-emphasize, or omit any part of this methodology (or any design methodology, for that matter) and expect to develop a sound structure. You must go through the process diligently, methodically, and completely in order to reap the expected rewards.

Note

I've provided a more detailed explanation of how I incorporated Normalization into my design methodology in Appendix G, "On Normalization."

There are a few basic terms you'll have to learn before you delve into the design process, and we'll cover them in the next chapter.

## SUMMARY

At the beginning of this chapter we looked at the importance of being concerned with database design. You now understand that database design is crucial to the integrity and consistency of the data contained in a database. We have seen that the chief problem resulting from improper or poor design is *inaccurate information*. Proper design is of paramount concern because bad design can adversely affect the information used by an organization.

Next, we entered into a discussion of the importance of theory, as well as its relevance to the relational database model. You learned that the model's foundation in mathematical theory makes it a very sound and reliable structure.

Following this discussion, we looked at the advantages gained by learning a design methodology. Among other things, using a good methodology yields an efficient and reliable database structure, reduces the time it takes to design a database, and allows you to avoid the typical problems caused by poor design.

Next, we listed the objectives of good design. Meeting these objectives is crucial to the success of the database design process because they help you ensure that the database structure is sound. We then enumerated the advantages of good design, and you learned that the time you invest in designing a sound database structure is time well spent.

We closed this chapter with a short discussion of traditional database design methods, an explanation of the premise behind the design method presented in this book, and Normalization. By now, you understand that traditional design methods are complex and can take some time to learn and comprehend. On the other hand, the design method used in this book is presented in a clear and straightforward manner, is easy to implement, and will yield the same results as the traditional design methodology.

## REVIEW QUESTIONS

**1.** When is the best time to use an RDBMS program's design tools?

**2.** True or False: Design is crucial to the consistency, integrity, and accuracy of data.

**3.** What is the most detrimental result of improper database design?

**4.** What fact makes the relational database structurally sound and able to guarantee accurate information?

**5.** State two advantages of learning a design methodology.

**6.** True or False: You will use your RDBMS program more effectively if you understand database design.

**7.** State two objectives of good design.

**8.** What helps to guarantee that data structures and their values are valid and accurate at all times?

**9.** State two benefits of applying good design techniques.

**10.** True or False: You can take shortcuts through some of the design processes and still arrive at a good, sound design.