Michael J. Hernandez

Foreword by Michelle Poolet
Mount Vernon Data Systems LLC

Database Design

for Mere Mortals®

Database Design for Mere Mortals®

This page intentionally left blank

Database Design for Mere Mortals®

25th Anniversary Edition

A Hands-On Guide to Relational Database Design

Fourth Edition

Michael J. Hernandez

Addison-Wesley

Boston ▪ Columbus ▪ New York ▪ San Francisco ▪ Amsterdam ▪ Cape Town
Dubai ▪ London ▪ Madrid ▪ Milan ▪ Munich ▪ Paris ▪ Montreal ▪ Toronto ▪ Delhi
Mexico City ▪ São Paulo ▪ Sydney ▪ Hong Kong ▪ Seoul ▪ Singapore ▪ Taipei ▪ Tokyo

Library of Congress Control Number: 2020945073

Published by Pearson Education, Inc.

Cover image: CARACOLLA / Shutterstock

ISBN-13: 978-0-13-678804-1
ISBN-10: 0-13-678804-1

ScoutAutomatedPrintCode

Editor-in-Chief: Mark L. Taub
Acquisitions Editor: Malobika Chakraborty
Development Editor: Chris Zahn
Managing Editor: Sandra Schroeder
Senior Project Editor: Lori Lyons
Production Manager: Vaishnavi Venkatesan/codeMantra
Copy Editor: Paula Lowell
Indexer: Erika Millen
Proofreader: Betty Pessagno
Cover Designer: Chuti Prasertsith
Compositor: codeMantra

For my wife, who has always believed in me and continues to do so.

To those who have helped me along my journey—teachers, mentors, friends, and colleagues.

To anyone who has unsuccessfully attempted to design a relational database.

To the hope that a better, more equitable, and more unified world will evolve from the 2020 COVID-19 pandemic.

This page intentionally left blank

Contents at a Glance

Contents

Figure Credits

Figure	Attribution
Cover	© CARACOLLA / Shutterstock
Figure 6.2	Screenshot of Visio © Microsoft 2020
Figure 6.3	Screenshot of Visio © Microsoft 2020
Figure 6.6	Screenshot of Visio © Microsoft 2020
Figure 6.7	Screenshot of Visio © Microsoft 2020
Figure 6.14	Screenshot of Visio © Microsoft 2020
Figure 6.15	Screenshot of Visio © Microsoft 2020
Figure 6.16	Screenshot of Visio © Microsoft 2020
Figure 6.18	Screenshot of Visio © Microsoft 2020

Foreword

Twenty-five years ago, Information Technology looked a lot different from the way it looks today. Twenty-five years from now, it's going to look a lot different again. One thing is certain; shared repositories where we park our data, that is, databases, are still going to be around. They've stood the test of time. And as long as we have databases, we're going to need to design them, so that we can find whatever it was we stored in them.

Mike Hernandez and I came to database design from different directions. Mike spent long years designing and developing databases, learning on the job as he was developing his technique. I started with a theoretical background and dived into database design armed with normalization tools and entity-relationship diagrams. At some point over these 25 years, we've managed to meet in the middle, and now here we are.

Mike's methodology is unique. He has based it on solid theory, but has been able to describe it in totally practical terms, detached from any single database product, wisely steering away from overburdening the reader with theoretical concepts and instead providing methods and tools that any person charged with designing a database on any platform can use to successfully complete his or her project.

Over the years, Mike and I have traded tips and quips and even argued over certain issues. I used his second edition when I was teaching classes in database fundamentals before the turn of the century (if that doesn't make you feel old. . .). At some point between the second and third editions of the book, he finally gave up and added the appendix on normalization—thank you, Mike. Even though data normalization can cause brain damage if you study it too hard or too

long, at least you, dear reader, will have an opportunity to get some exposure.

At this point I'm comfortable with the content. Just so you know, I've done a total review of this book before it was published, and if Mike has followed my suggestions, it'll be a good and terribly useful read. This book is not meant to be a formal treatise on database design theory. It is a book for everyman and everywoman. It is a book that lays out the fundamentals of how to create a database that will do what it's meant to do, as long as you follow the steps in Mike's methodology. As he says more than once, skip a step and live to regret!

Best wishes, dear reader, as you start your journey on the path to database design awareness. You're in good hands with Mike.

Mount Vernon Data Systems LLC

Preface

Life, as the most ancient of all metaphors insists,
is a journey. . .
—Jonathan Raban, For Love and Money

The Past is behind you; you must let it go
The Future is in front of you; you must allow it to unfold
The Present is where you stand;
it was the Future that will become your Past
Therefore, be full in the moment—for it is all you really have
—Michael J. Hernandez, July 2020

It's now 25 years since the first edition of this book was published. That's a long time. I never would have imagined that my work would have lasted this long. More on this in a moment. . . .

So, the question after all this time is this: Is there *still* a need for a book such as the one you hold in your hands? And the answer (surprisingly, yet undoubtedly) is still "Yes!" Regardless of how complex or complicated database management becomes, there will always be a need for a book on the basics of database design. You must learn the fundamentals to know how and why things work the way they do. This is true of many other areas of expertise, whether they are technical disciplines such as architectural design and engineering or artistic disciplines such as acting and music.

My friends and colleagues have always known that this work would last a long time. They said the software-agnostic approach was a brilliant move on my part because I would not be endlessly tied to revising the book based on the latest software editions. The manner in which I wrote the book, they said, also contributed to its timelessness—no dependence on programming languages, SQL, RDBMS feature sets, or technical jargon that constantly changes anyway. All I did was just talk about defining the logical design of a database. The fact of the

matter is that what was true 25 years ago about the basics of designing a database is still true now. And I'm most certainly glad and proud that my work is still out there helping people to design databases.

My journey has taken me along new and different paths in recent years, and I'm really enjoying what I do. I don't work on databases as much as I used to, anymore, but I still find the work fascinating and rewarding. Other types of work are keeping me busier than ever, and I can't wait to see on what other paths my journey takes me now.

Register Your Book

Register your copy of *Database Design for Mere Mortals*® at informit.com for convenient access to downloads, updates, and corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN 9780136788041 and click Submit. Once the process is complete, you will find any available bonus content under "Registered Products."

Acknowledgments

Writing has always been and will always be a truly cooperative and collaborative effort, despite what you may have heard about it. Editors, production staff, technical reviewers, colleagues, family, and friends all help you realize your dream project and get it off the ground. I've always been very lucky and grateful throughout the years to have such people around me, who continue to be ready and willing to lend their help. These are the people who provided encouragement and kept me focused on the task at hand, and it is to them that I extend my most heartfelt appreciation. For this 25th anniversary edition of *Database Design for Mere Mortals*®, I think it is an appropriate time as any to thank the key people who have helped nurture this book since Day One.

First and foremost, I want to thank my absolutely wonderful editor, Kim Spenceley, who competently and lovingly guided this edition from a passing thought in my mind to the book you have in your hands now. She was so encouraging and enthusiastic about me writing a 25th anniversary edition, and it was her patience, kindness, and leadership that helped me take on this work and bring it to successful completion. A special thanks to Chris Zahn and his production staff—they've done top-notch work! I must say that Chris is meticulous, and you can tell he loves what he does. I want to thank my production editor Lori Lyons for guiding the copyedit and final review process and Vaishnavi Venkatesan for guiding composition. It went so smoothly, and I think the book looks just great! I also thank my copy editor Paula Lowell for the absolutely marvelous job she did on the copyedit. She really sharpened up the material, and any final mistakes you see in this book are very likely mine.

I would like to acknowledge my distinguished technical reviewers Michelle Poolet, co-founder and President of Mount Vernon Data Systems LLC; Richard Banks, a Senior Consultant and Trainer at SoftResult.com; and Dr. Jared Wuerzburge, an Assistant Professor at Indiana State University. All graciously and generously gave their time, effort, and vast expertise to lend a hand in refreshing and refining the material. Michelle served as the chief technical reviewer and provided me with a wealth of valuable feedback and suggestions. Her contributions undoubtedly helped to update and fine-tune the overall content of the book. To top it off, she graciously agreed to write the Foreword for this edition, and I'm so happy and honored that she did. Richard has an eagle eye for details, and he caught subtle and nuanced points about the material that other reviewers would have missed. He definitely helped to smooth out some final rough edges. Professor Wuerzburge brought a few things to my attention from a different perspective. It was quite refreshing and added to the final polish of the material. Many thanks once again to all of you for your time and input and for helping to make this edition the best yet!

Next, I want to acknowledge and thank the folks who helped me with the third edition: my fabulous editor, Joan Murray; production editor, John Fuller, and his staff; production editor, Caroline Senay; copy editor, Audrey Doyle; and my technical review team Tracy Thornton, Tony Wiggins, and Theodor Richardson.

Let me now acknowledge and thank the folks who helped me with the second edition: my outstanding editor, Mary O'Brien, and her assistants Alicia Carey, Stacie Parillo, and Brenda Mulligan; production editor, John Fuller, and his staff; production editor, Tyrrell Albaugh; copy editor, Jennifer Kelland; and my technical review team, Sandy Barker, Michael Blaha, Matt Greer, and Michelle Poolet (this was our first time working together).

Finally, I want to acknowledge and thank the folks who helped me with the first edition. First, I thank my awesome editor Kathleen Tibbets, who approached me about writing this book in the first place. Here's the story on how this book came to be.

We were both attending a PC Database Conference in Bellevue, Washington, and she was making the rounds, asking the typical rogues' gallery of authors (most of whom were my friends) if they wanted to write a book. She didn't have much luck, as many of them were busy at the time with other projects. She then comes to speak with me.

I remember what happened next as if it happened yesterday.

I didn't know her at the time. She sees me in the hallway, introduces herself and says, "I hear you have a book to write." That seemed like an odd thing to say, so I asked her what she meant by that statement. "Well, I've talked to John and Roger and Armen and Michael and Ken and Paul and Chris and Matt, and they're all too busy to write anything new this year. However, to a person, they all made one point: *Go talk to Mike Hernandez! He keeps complaining that there are no simple, straightforward books on database design. Well, as most of us know, he's become quite an expert at it. So, tell him to stop complaining and put up or shut up—tell him to write the damn book himself!*"

Kathleen and I then meet for lunch the next day to discuss my ideas for such a book. By the time our *three-hour* lunch meeting was over, we were making plans to get the contracts set up so that I could start on the book right away. That was the origin of *Database Design for Mere Mortals*®.

I also acknowledge and thank John Fuller and his staff (again) for the awesome work they did on that edition; my review team, Jim Booth and Christopher Webber; and the people who had an early, positive influence on my database career: Karen Watterson, Mike Johnson, Karl Fischer, Paul Litwin, John Viescas, Ken Getz, and Gregory Piercy.

I want to extend a very special thanks to my longtime friend, colleague, and well-respected expert, Ken Getz, who graciously agreed to write the Foreword for the first three editions of the book. From what I understand, Ken is now pursuing a life of peace, happiness, and serenity somewhere in California. He's happily playing music now, which is a passion we both share.

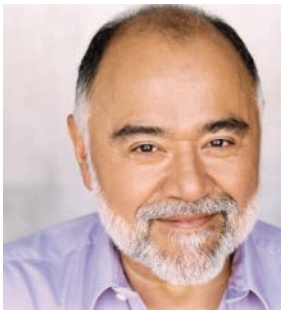
A special thanks also goes to all those readers who took the time over the years to send me emails from time to time and post their reviews and comments on Amazon. I am humbled by their praise and support and particularly appreciative of the good, constructive criticism that eventually helped me to improve the material throughout its evolution. I also want to thank all the academic institutions, government agencies, and commercial organizations that have adopted my book and made it "standard reading" over the past 25 years. I am honored by their support of my work.

I want to take a moment to honor the memory of my very dear friend and mentor, Alastair Black, who passed away a number of years ago. He was the one who taught me the *craft* of writing, of what it means to be a good writer. I am forever indebted to him for what he taught me, and I hope everything I've written since our time together has been a credit to his memory.

Finally, I want to thank my wife for her unending patience while I was enmeshed in writing each edition of the book. Her unending help and support have always been invaluable, and yet again, I owe her a great debt. I would tell you exactly how I feel about her, but she abhors any sort of PDA (public display of affection).

Instead, I'll just extend her a laurel, and hardy handshake.

About the Author



Michael J. Hernandez is a man with many talents and has done quite a bit in his life, and he continues to do many things to keep him busy. You could say he's a modern-day Renaissance Man.

For instance, he has been an independent database consultant with more than 25 years of experience in the technology industry, and has been a contributing author to a wide variety of magazine articles, white papers, books, and periodicals. Mike has also been a top-rated and noted technical instructor for two national training organizations, the government, the military, the private sector, and companies throughout the United States, and he has been a top-rated speaker and presenter at numerous national and international conferences. He also worked full-time at Microsoft for seven years as a program manager, product manager and marketing manager in the Visual Studio Group.

Mike is mainly pursuing his artistic side at this point in his life.

He is currently a professional actor (with more than seven years of experience as of this writing), performing in commercials for the likes of U.S. Bank, Allstate Insurance and Subaru; in TV for shows such as "Veep," "American Crime," and "Unbelievable"; and feature films such as *Ernesto's Manifesto*, *The Honor List*, and *Almost Home*.

Mike is also a talented and adept Tarot reader with more than 11 years of experience. He's done private and public readings at a variety of venues, such as metaphysical stores, psychic fairs, conferences, business events, and a variety of parties.

He also continues to pursue his ultimate artistic love: playing the guitar. Mike has studied and played the guitar for more than 50 years. He has played in a variety of professional bands (actually playing professionally for 15 years), was a member of the United States Navy Band in Seattle, Washington, for 5 years, and even led a couple of employee-based bands when he worked at Microsoft!

He says he's never going to retire, per se, but rather just change whatever it is he's doing whenever he finally gets tired of it and move on to something else that interests him.

Introduction

Plain cooking cannot be entrusted to plain cooks.
—Countess Morphy

It never ceases to amaze me how much database software programs have evolved in the past 25 years. We've gone from databases stored and run from personal computers to databases shared and centralized company-wide on client/server architectures connected within local area networks and wide area networks, to apps accessed on personal devices with databases stored in the cloud.

Database management systems have evolved, too. They are much easier to use and maintain, they now handle very high volumes of data, and they're as ubiquitous as hamburgers. You use applications built with them all the time on your computer and devices whenever you're working, shopping, or looking for information on the Internet.

Vendors of database software continue to add new features and enhance the tool sets in their products, enabling database developers to create more powerful and flexible database applications. They're also constantly improving the ease with which the software can be used, enabling many people to create their own database applications. Today's database software greatly simplifies the process of creating efficient database structures and intuitive user interfaces.

Most programs provide sample database structures that you can copy and alter to suit your specific needs. Although you might initially think that it would be quite advantageous for you to use these sample structures as the basis for a new database, you should stop and

reconsider that move for a moment. Why? Because you could easily and unwittingly create an improper, inefficient, and incomplete design. You then would eventually encounter problems in what you believed to be a dependable database design. This, of course, raises the question, "What types of problems would I encounter?"

Most problems that surface in a database fall into two categories: *application utilization* problems and *data* problems. Application problems include such things as problematic data entry/edit forms, confusing menus and toolbars, confusing dialog boxes, and tedious task sequences. These problems typically arise when the database developer is inexperienced, is unfamiliar with a good application design methodology, knows too little about the software he's using to implement the database, or has an insufficient understanding of the data with which he is working. Problems of this nature are common and important to address, but they are beyond the scope of this work.

❖ **Note** One good way to solve many of your application problems is to purchase and study third-party "developer" books that cover the software you're using. Such books discuss application design issues, advanced programming techniques, and various tips and tricks that you can use to improve and enhance an application. Armed with these new skills, you can revamp and fine-tune the database application so that it works correctly, smoothly, and efficiently.

Data problems, on the other hand, include such things as missing data, incorrect data, mismatched data, and inaccurate information. Poor database design is typically the root cause of these types of problems. A database will not fulfill an organization's information requirements if it is not structured properly. Although poor design is typically generated by a database developer who lacks knowledge of good database design principles, it shouldn't necessarily reflect negatively on

the developer. Many people, including experienced programmers and database developers, have had little or no instruction in any form of database design methodology. Many are unaware that design methodologies even exist. Data problems and poor design are the issues that this work will address.

What's New in the Fourth Edition

I revised this edition to improve readability, update or extend existing topics, add new content, and enhance its educational value. Here is a list of the changes you'll find in this edition.

- Portions of the text have been rewritten to improve clarity and reader comprehension.
- Figures have been updated for improved relevance as appropriate.
- There is now a new "Example" section at the end of Chapters 5–13.
- Chapter 1 has been revised, removing content that is no longer important and adding a "What's Next?" section regarding what I see as the current status and future of the relational database.
- Chapter 5 has a new note concerning COVID-19 and it's impact on doing interviews for the design process.
- The discussion in Chapter 5 on interviews and the interviewer guidelines have both been slightly revised to account for the rising popularity and acceptance of online meetings and work from home (WFH) scenarios.
- The sample interview dialog in Chapter 7 has been updated as a result of the interviewer guidelines changes in Chapter 5.

- The Field Specifications form has been slightly streamlined, and the Business Rules form has been adjusted as appropriate to synchronize it with the changes in the Field Specifications form.
- The Subject Identification Technique and Characteristic Identification Technique processes have also been adjusted to account for on-line meeting scenarios.
- The end-of-chapter "Review Questions" have been updated to reflect new questions, and changes have been made to existing questions.
- The "Recommended Reading" section includes the latest editions of the books.

Who Should Read This Book

No previous background in database design is necessary to read this book. The reason you have this book in your hands is to learn how to design a database properly. If you're just getting into database management and you're thinking about developing your own databases, this book will be very valuable to you. Learning how to create a database properly from the beginning is better than learning by trial and error. Believe me—the latter method takes much longer.

If you fall into the category of those people who have been working with database programs for a while and are ready to begin developing new databases for your company or business, you should read this book. You probably have a good feel for what a good database structure should look like but aren't quite sure how database developers arrive at an effective design. Maybe you're a programmer who has created a number of databases following a few basic guidelines, but you have always ended up writing a lot of code to get the database to work properly. If this is the case, this book is also for you.

Reading this book is also a good idea even if you already have some background in database design. Perhaps you learned a design methodology back in college or attended a database class that discussed design, but your memory is vague about some details, or you just did not completely understand some parts of the design process. Those points with which you had difficulty will finally become clear after you learn and understand the design process presented in this book.

This book is also appropriate for those of you who are experienced database developers and programmers. Although you may already know many of the aspects of the design process presented in this book, you'll probably find some elements that you've never before encountered or considered. You may even come up with fresh ideas about how to design your databases by reviewing the material in this book because many of the design processes familiar to you are presented from a different viewpoint. But remember, if you *are* this type of person, *for you,* this will be just a basic review of the overall concepts of good database design. No normalization. No technical jargon. No programming language or code references. Just a good review of what sound data structures look like and the reasons behind making good, sound structural decisions.

The Purpose of This Book

In general terms, the overall database development process has three phases.

1. *Logical design:* The first phase involves determining and defining tables and their fields, establishing primary and foreign keys, establishing table relationships, and determining and establishing the various levels of data integrity.

2. *Physical implementation:* The second phase entails creating the tables, establishing key fields and table relationships, and using the proper tools to implement the various levels of data integrity.
3. *Application development:* The third phase involves creating an application that allows a single user or group of users to interact with the data stored in the database. The application development phase itself can be divided into separate processes, such as determining end-user tasks and their appropriate sequences, determining information requirements for report output, and creating a menu system for navigating the application.

You should always go through the logical design first and execute it as completely as possible. After you've created a sound structure, you can then implement it within any database software you choose. As you begin the implementation phase, you may find that you need to modify the database structure based on the pros and cons or strengths and weaknesses of the database software you've chosen. You may even decide to make structural modifications to enhance data processing performance. Performing the logical design first ensures that you make conscious, methodical, clear, and informed decisions concerning the structure of your database. As a result, you help minimize the potential number of further structural modifications you might need to make during the physical implementation and application development phases.

This book deals with *only the logical design phase* of the overall development process, and the book's main purpose is to explain the process of relational database design without using the advanced, orthodox methodologies found in an overwhelming majority of database design books. I've taken care to avoid the complexities of these methodologies by presenting a relatively straightforward, commonsense approach to the design process. I also use a simple and straightforward data

modeling method as a supplement to this approach and present the entire process as clearly as possible and with a minimum of technical jargon.

Many database design books on the market include chapters on implementing the database within a specific database product, and some books even seem to meld the design and implementation phases together. (I've never particularly agreed with the idea of combining these phases, and I've always maintained that a database developer should perform the logical design and implementation phases separately to ensure maximum focus, effectiveness, and efficiency.) The main drawback that I've encountered with these types of books is that a reader can have difficulty obtaining any useful or relevant information from the implementation chapters if he or she doesn't work with the particular database software or programming language that the book incorporates. It is for this reason that I decided to write a book that focuses strictly on the logical design of the database.

This book should be easier to read than other books you may have encountered on the subject. Many of the database design books on the market are highly technical and can be difficult to assimilate. I think most of these books can be confusing and overwhelming if you are not a computer science major, database theorist, or experienced database developer. The design principles you'll learn within these pages are easy to understand and remember, and the examples are common and generic enough to be relevant to a wide variety of situations.

Most people I've met in my travels around the country have told me that they just want to learn how to create a sound database structure without having to learn about normal forms or advanced mathematical theories. Many people are not as worried about implementing a structure within a specific database software program as they are about learning how to optimize their data structures and how to impose data integrity. In this book, you'll learn how to create efficient database

structures, how to impose *several* levels of data integrity, as well as how to relate tables together to obtain information in an almost infinite number of ways. Don't worry; this isn't as difficult a task as you might think. You'll be able to accomplish all of this by understanding a few key terms and by learning and using a specific set of commonsense techniques and concepts.

You'll also learn how to analyze and leverage an existing database, determine information requirements, and determine and implement business rules. These topics are important because many of you will probably inherit old databases that you'll need to revamp using what you'll learn by reading this book. They'll also be just as important when you create a new database from scratch.

When you finish reading this book, you'll have the knowledge and tools necessary to create a good relational database structure. I'm confident that this entire approach will work for a majority of developers and the databases they need to create.

How to Read This Book

I strongly recommend that you read this book in sequence from beginning to end, regardless of whether you are a novice or a professional. You'll keep everything in context this way and avoid the confusion that generally comes from being unable to see the "big picture" first. It's also a good idea to learn the process as a whole before you begin to focus on any one part.

If you are reading this book to refresh your design skills, you could read just those sections that are of interest to you. As much as possible, I've tried to write each chapter so that it can stand on its own; nonetheless, I still recommend that you glance through each chapter to make sure you're not missing any new ideas or points on design that you may not have considered up to now.

How This Book Is Organized

The following is a brief overview of what you'll find in each part and each chapter.

Part I: Relational Database Design

This section provides an introduction to databases, the idea of database design, and some of the terminology you'll need to be familiar with to learn and understand the design process presented in this book.

Chapter 1, "The Relational Database," provides a brief discussion of the types of databases you'll encounter, common database models, and a brief history of the relational database.

Chapter 2, "Design Objectives," explores why you should be concerned with design, points out the objectives and advantages of good design, and provides a brief introduction to normalization and normal forms.

Chapter 3, "Terminology," covers the terms you need to know to learn and understand the design methodology presented in this book.

Part II: The Design Process

Each aspect of the database design process is discussed in detail in Part II, including establishing table structures, assigning primary keys, setting field specifications, establishing table relationships, setting up views, and establishing various levels of data integrity.

Chapter 4, "Conceptual Overview," provides an overview of the design process, showing you how the different components of the process fit together.

Chapter 5, "Starting the Process," covers how to define a mission statement and mission objectives for the database, both of which provide you with an initial focus for creating your database.

Chapter 6, "Analyzing the Current Database," covers issues concerning the existing database. We look at reasons for analyzing the current database, how to look at current methods of collecting and presenting data, why and how to conduct interviews with users and management, and how to compile initial field lists.

Chapter 7, "Establishing Table Structures," covers topics such as determining and defining what subjects the database should track, associating fields with tables, and refining table structures.

Chapter 8, "Keys," covers the concept of keys and their importance to the design process, as well as how to define candidate and primary keys for each table.

Chapter 9, "Field Specifications," covers a topic that a number of database developers tend to minimize. Besides indicating how each field is created, field specifications determine the very nature of the values a field contains. Topics in this chapter include the importance of field specifications, types of specification characteristics, and how to define specifications for each field in the database.

Chapter 10, "Table Relationships," explains the importance of table relationships, types of relationships, setting up relationships, and establishing relationship characteristics.

Chapter 11, "Business Rules," covers types of business rules, determining and establishing business rules, and using validation tables. Business rules are very important in any database because they provide a distinct level of data integrity.

Chapter 12, "Views," looks into the concept of views and why they are important, types of views, and how to determine and set up views.

Chapter 13, "Reviewing Data Integrity," reviews each level of integrity that has been defined and discussed in previous chapters. Here you learn that reviewing the final design of the database structure is a good idea to ensure that you've imposed data integrity as completely as you can.

Part III: Other Database Design Issues

This section deals with topics such as avoiding bad design and bending the rules set forth in the design process.

Chapter 14, "Bad Design—What Not to Do," covers the types of designs you should avoid, such as a flat-file design and a spreadsheet design.

Chapter 15, "Bending or Breaking the Rules," discusses those rare instances in which straying from the techniques and concepts of the design process may be necessary. This chapter tells you when you should consider bending the rules, as well as how you should do it.

Part IV: Appendixes

These appendixes provide information that I thought would be valuable to you as you're learning about the database design process and when you're working on developing your database.

Appendix A, "Answers to Review Questions," contains the answers to all the review questions in Chapters 1 through 12.

Appendix B, "Diagram of the Database Design Process," provides a diagram that maps the entire database design process.

Appendix C, "Design Guidelines," provides an easy reference to the various sets of design guidelines that appear throughout the book.

Appendix D, "Documentation Forms," provides blank copies of the Field Specifications, Business Rule Specifications, and View

Specifications sheets, which you can copy and use on your database projects.

Appendix E, "Database Design Diagram Symbols," contains a quick and easy reference to the diagram symbols used throughout the book.

Appendix F, "Sample Designs," contains sample database designs that can serve as the basis for ideas for databases you may want or need to create.

Appendix G, "On Normalization," provides a discussion on how I incorporated normalization into my design methodology.

Appendix H, "Recommended Reading," provides a list of books that you should read if you are interested in pursuing an in-depth study of database technology.

Glossary contains concise definitions of various words and phrases used throughout the book.

IMPORTANT: READ THIS SECTION!

A Word about the Examples and Techniques in This Book

You'll notice that this book contains a wide variety of examples. I've made sure that they are as generic and relevant as possible. However, you may notice that several of the examples are rather simplified, incomplete, or occasionally even incorrect. Believe it or not, I created them that way on purpose.

I've created some examples with errors so that I could illustrate specific concepts and techniques. Without these examples, you wouldn't see how the concepts or techniques are put to use, as well as the

results you should expect from using them. Other examples are simple because, once again, the focus is on the technique or concept and not on the example itself. For instance, you can design an order-tracking database in many ways. However, the structure of the sample order-tracking database I use in this book is simple because the focus is specifically on the *design process,* not on creating an elaborate order-tracking database system.

So what I'm really trying to emphasize here is this:

> Focus on the concept or technique and its intended results, *not on the example used to illustrate it.*

My Approach to Learning

Here's my approach to learning the design process (or pretty much anything else, for that matter) that I've found very useful in my database design classes.

Think of all the techniques used in the design process as a set of tools; each tool (or technique) is used for a specific purpose. The idea here is that after you learn how a tool is used generically, you can then use that tool in any number of situations. The reason you can do this is *because you use the tool the same way in each situation.*

Take a Crescent wrench, for example. Generically speaking, you use a Crescent wrench to fasten and unfasten a nut to a bolt. You open or close the jaw of the wrench to fit a given bolt by using the adjusting screw located on the head of the wrench. Now that you're clear about its use, try using it on a few bolts. Try it on the legs of an outdoor chair, or the fan belt cover on an engine, or the side panel of an outdoor cooling unit, or the hinge plates of an iron gate. Do you notice that regardless of where you encounter a nut and bolt, you can always fasten and unfasten the nut by using the Crescent wrench in the same manner?

The tools used to design a database work in *exactly* the same way. After you understand how a tool is used generically, it will work the same way regardless of the circumstances under which it is used. For instance, consider the tool (or technique) for decomposing a field value. Say you have a single Address field in a CUSTOMERS table that contains the street address, city, state, and ZIP code for a given customer. You'll find it difficult to use this field in your database because it contains more than one item of data; you'll certainly have a hard time retrieving information for a particular city or sorting the information by a specific ZIP code.

The solution to this apparent dilemma is to decompose the Address field into smaller fields. You do this by identifying the distinct items that make up the value of the field, and then treating each item as its own separate field. That's all there is to it! This process constitutes a "tool" that you can now use on *any* field containing a value composed of two or more distinct data items, such as these sample fields. The following table shows the results of the decomposition process.

Current Field Name	Sample Value	New Field Names
Address	7402 Kingman Dr., Seattle, WA 98012	Street Address, City, State, ZIP Code
Phone	(206) 555-5555	Area Code, Phone Number
Name	Mike Hernandez	First Name, Last Name
EmployeeCode	ITDEV0516	Department, Category, ID Number

❖ **Note** You'll learn more about decomposing field values in Chapter 7, "Establishing Table Structures."

You can use all the techniques ("tools") that are part of the design process presented in this book in the same manner. You'll be able to

design a sound database structure using these techniques regardless of the type of database you need to create. Just be sure to remember this:

> Focus on the concept or technique being presented and its intended results, *not on the example used to illustrate it.*

This page intentionally left blank

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?

The Importance of Theory

The Advantage of Learning a Good Design Methodology

Objectives of Good Design

Benefits of Good Design

Database-Design Methods

Normalization

Summary

Review Questions

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 wants. *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. After 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 upon which you can make sound business decisions.

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 airplanes and spacecraft.

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 likely happen if you perform a certain action or series of actions. You know if you drop a stone, it will fall to the ground. Sir Isaac Newton posited a theory of gravity in the late 17th century, which we now consider a law. If you are agile, you can get your toes out of the way of Newton's Law of Gravity before they get smashed by the falling stone. The point is that once theory becomes law, 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 stylized residential buildings. 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. To this day, 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 central to the relational model and is what guarantees the model's viability. But cheer up—you don't really need to know anything about

set theory or first-order predicate logic to use a relational database! You certainly don't have to know all the details of aerodynamics just to drive an automobile or to fly an airplane. Aerodynamic 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 understand or render them relatively meaningless. 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 step.
- *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

You must achieve distinct objectives 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 fulfilling these objectives to be difficult at times, but you'll certainly be pleased with your final database structure after 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 designed

structure. 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, table, or relationship need not adversely affect other fields, tables, or relationships 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. The inter-table relationships are fairly obvious in a well-designed database, even when they're not being enforced.
- *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.

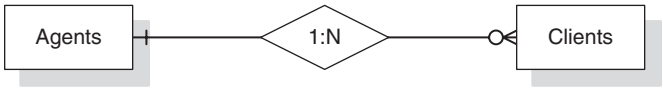


Figure 2.1 *An example of a basic ER diagram.*

❖ **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 only one agent, and the circle and

"crow's foot" next to the CLIENTS table indicates that an agent doesn't necessarily have to be associated with a client, but can be associated with one or more.

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*. After 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 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 relation is in 3NF if and only if it is in 2NF and every non-key attribute is non-transitively dependent on the primary key.[1]

This description is relatively meaningless to a reader who is unfamiliar with the terms *relation, 3NF, 2NF, non-key attribute, non-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 12 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

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

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 whom 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.

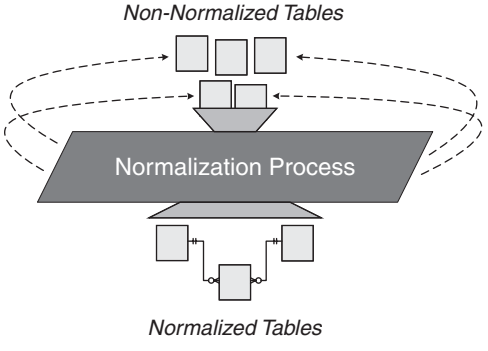


Figure 2.2 *How I viewed the general normalization process.*

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 designer 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."

You'll have to learn a few basic terms 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 needed 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.

Index

D

E

F

G

S

T

W-X-Y-Z