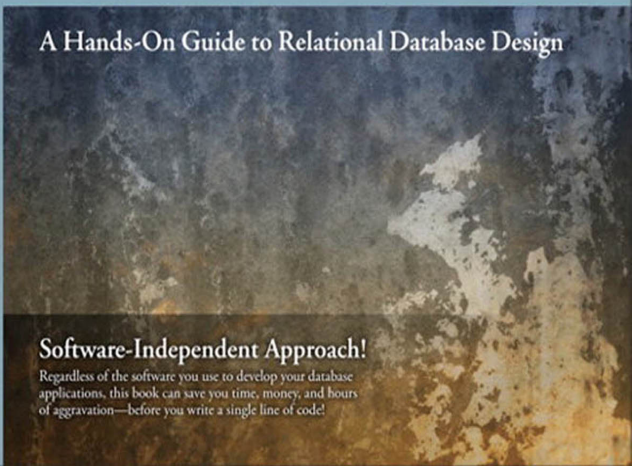
Michael J. Hernandez
Foreword by Ken Getz

DATABASE DESIGN FOR MERE MORTALS®

THIRD EDITION



Database Design for Mere Mortals®

Third Edition

This page intentionally left blank

Database Design for Mere Mortals®

A Hands-on Guide to Relational Database Design

Third Edition

Michael J. Hernandez

Addison-Wesley

Upper Saddle River, NJ ▪ Boston ▪ Indianapolis ▪ San Francisco
New York ▪ Toronto ▪ Montreal ▪ London ▪ Munich ▪ Paris ▪ Madrid
Capetown ▪ Sydney ▪ Tokyo ▪ Singapore ▪ Mexico City

Cataloging-in-Publication Data is on file with the Library of Congress.

ISBN-13: 978-0-321-88449-7
ISBN-10: 0-321-88449-3
Text printed in the United States on recycled paper at Edwards Brothers Malloy in Ann Arbor, Michigan.
Third printing, October 2014

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.

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

This page intentionally left blank

About the Author

Michael J. Hernandez has been an independent relational database consultant specializing in relational database design. He has more than twenty years of experience in the technology industry, developing database applications for a broad range of clients. He's been a contributing author to a wide variety of magazine columns, white papers, books, and periodicals, and is coauthor of the best-selling *SQL Queries for Mere Mortals*® (Addison-Wesley, 2007). Mike has been a top-rated and noted technical trainer for the government, the military, the private sector, and companies throughout the United States. He has spoken at numerous national and international conferences, and has consistently been a top-rated speaker and presenter.

Aside from his technical background, Mike has a diverse set of skills and interests that he also pursues, ranging from the artistic to the metaphysical. His greatest interest is still the guitar, as he's been a practicing guitarist for more than forty years and played professionally for fifteen years. He is a great cook, loves to teach (writing, public speaking, music), has a gift for bad puns, and even reads tarot cards.

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.

This page intentionally left blank

Contents

This page intentionally left blank

Foreword

To the *Third Edition*

Here it is, ten years later, and Mike and I cross paths even less than we used to. For those who were unaware, we share the same birthday (although he's much older than me, at least one full year), and we meet up at least once each year and congratulate ourselves for making it another year. It's also funny how Microsoft "reboots" its technology every ten years or so, and now, revisiting the foreword I wrote ten years ago, nothing much has changed—I'm still hip-deep in a new Microsoft technology, but this time it's all about WinRT and Windows 8, rather than .NET. One thing that hasn't changed, however, is the need for carefully planned and executed database design. Nothing Mike wrote in his original volume has changed very much, and although this new edition modifies some details, the basics of good database design haven't changed in the ensuing ten years. I must confess a little jealousy that Mike has written a book with such enduring shelf life, but, if he's going to have a book that succeeds for this many years, at least it's a good one. Whether this is your first visit to Mike's detailed explanation of database design, or your second or third, be assured that you'll find a carefully considered, helpful path through the vagaries of database design here. But let's get past the intro, and get to work!

—Ken Getz, November 14, 2012

From the Second Edition . . .

I don't see Mike Hernandez as much as I used to. Both our professional lives have changed a great deal since I first wrote the foreword to his original edition. If nothing else, we travel less, and our paths cross less often than they did. If you'll indulge me, I might try to add that the entire world has changed since that first edition. On the most

mundane level, my whole development life has changed, since I've bought into this Microsoft .NET thing whole-heartedly and full-time. One thing that hasn't changed, however, is the constant need for data, and well-designed data. Slapping together sophisticated applications with poorly designed data will hurt you just as much now as when Mike wrote his first edition—perhaps even more. Whether you're just getting started developing with data, or are a seasoned pro; whether you've read Mike's previous book, or this is your first time; whether you're happier letting someone else design your data, or you love doing it yourself—this is the book for you. Mike's ability to explain these concepts in a way that's not only clear, but fun, continues to amaze me.

—Ken Getz, October 10, 2002

From the First Edition . . .

Perhaps you're wondering why the world needs another book on database design. When Mike Hernandez first discussed this book with me, *I* wondered. But the fact is—as you may have discovered from leafing through pages before landing here in the foreword—the world *does* need a book like this one. You can certainly find many books detailing the theories and concepts behind the science of database design, but you won't find many (if any) written from Mike's particular perspective. He has made it his goal to provide a book that is clearly based on the sturdy principles of mathematical study, but has geared it toward practical use instead of theoretical possibilities. No matter what specific database package you're using, the concepts in this book will make sense and will apply to your database-design projects.

I knew this was the book for me when I turned to the beginning of Chapter 6 and saw this suggestion:

> Do not adopt the current database structure as the basis for the new database structure.

If I'd had someone tell me this when I was starting out on this database developer path years ago I could have saved a *ton* of time! And that's my point here: Mike has spent many years designing databases for clients; he has spent lots of time thinking, reading, and studying about the *right* way to create database applications; and he has put it all here, on paper, for the rest of us.

This book is full of the right stuff, illustrated with easy-to-understand examples. That's not to say that it doesn't contain the hardcore information you need to do databases right—it does, of course. But it's geared toward real developers, not theoreticians.

I've spent some time talking with Mike about database design. Over coffee, in meetings, writing courseware, it's always the same: Mike is passionate about this material. Just as the operating system designer seeks the perfect, elegant algorithm, Mike spends his time looking for just the right way to solve a design puzzle and—as you will read in this book—how best to explain it to others. I've learned much of what I know about database design from Mike over the years and feel sure that I have a lot more to learn from this book. After reading through this concise, detailed presentation of the information you need to know in order to create professional databases, I'm sure you'll feel the same way.

—Ken Getz, MCW Technologies (KenG@mcwtech.com)

This page intentionally left blank

Preface

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

Paths may change and
the course may need adjustment,
but the journey continues . . .
—Michael J. Hernandez
Database Design for Mere Mortals®, Second Edtion

To say that the technology field, and database management in particular, has changed significantly in the nine years since the second edition of this book was published would be an understatement, to be sure. Small, handheld devices containing storage capacity and processing power that once would have required several room-sized mainframe computers are now so ubiquitous that many people take them for granted, especially the more recent generations. (My young nephew would likely never understand the excitement I experienced when I purchased my first 40MB storage expansion card for my IBM PC. But that's another story.) Database management systems can now handle *terabytes* of data, and there's recently been a considerable amount of emphasis on storing, managing, and accessing data "in the cloud."

Is there still a need, then, for a book such as the one you hold in your hands? Absolutely! 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 in order 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 music and cooking.

My journey has taken me along new and different paths in recent years, and I'm really enjoying what I do. I've been doing a lot more writing lately, which is why I thought it was time to do this new edition. I thought I'd share some new nuggets of information I've learned along the way and perhaps clarify my perspectives on this subject a little more. Now that I've completed this work, I can't wait to see where my journey takes me next.

An important note to readers:

Visit Informit.com/titles/0321884493 to access additional content referenced in the book.

Acknowledgments

Writing is truly a cooperative effort, despite what you may have heard about it. I'm so grateful for the editors, colleagues, friends, and family 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.

First and foremost, I want to thank my wonderful editor, Joan Murray, for the opportunity to write yet another edition of my book. We had been talking about this project for a couple of years, and it was her perseverance, patience, kindness, and leadership that helped me decide to take on this work and bring it to successful completion. I also want to thank production editor Caroline Senay for guiding the author review process with such a deft hand and copy editor Audrey Doyle for her precise and detailed review of the content. And a special thanks to John Fuller and his production staff—they did great work, as they always do! I've always had a wonderful relationship with the Addison-Wesley team, and I just can't imagine why I'd ever want to write technical books for anyone else.

Next, I'd like to acknowledge my distinguished technical review team: Tracy Thornton, Tony Wiggins, and Theodor Richardson. These folks graciously and generously gave their time, effort, and expertise to provide me with a wealth of valuable feedback and suggestions. This book definitely benefitted from their contributions. My thanks once again to each of you for your time and input and for helping to make this edition even better than I first envisioned.

I want to extend a very special thanks to Ken Getz for once again providing the foreword for my book. Ken is a well-respected expert, a colleague, and a good friend. I'm so pleased to have his thoughts and comments at the beginning of the book.

A special thanks also goes to all of those readers who took the time to send me their thoughts and comments. I am humbled by their praise and support and particularly appreciative of the good, constructive criticism that eventually helped me to improve the material in this edition. I also wish to thank all the academic institutions, government agencies, and commercial organizations that have adopted my book and made it "standard reading" for those just beginning their database careers. I am honored by their support of my work.

Finally, I want to thank my wife for her unending patience while I was enmeshed in my writing. Her help and support have been invaluable, and once 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.

Introduction

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

In the past, the process of designing a database has been a task performed by people in information technology (IT) departments and professional database developers. These people usually had mathematical, computer science, or systems design backgrounds and typically worked with large mainframe databases. Many of them were experienced programmers and had coded a number of database application programs consisting of thousands of lines of code. (And these people were usually very overworked due to the nature and importance of their work!)

People designing database systems at that time needed to have a solid educational background because most of the systems they created were meant to be used companywide. Even when creating databases for single departments within a company or for small businesses, database designers still required extensive formal training because of the complexity of the programming languages and database application programs they were using. As technology advanced, however, those educational requirements evolved.

Database software programs have evolved quite a bit since the 1980s, too. Many vendors developed software that ran on desktop computers and could be more easily programmed to collect, store, and manage data than their mainframe counterparts. As computing power and demand for complexity grew, vendors produced software that allowed groups of people to access and share centralized data within a variety

of environments, such as client/server architectures on computers connected within local area networks (LANs) and wide area networks (WANs). People within a company or organization were no longer strictly dependent on mainframe databases or on having their information needs met by centralized IT departments.

The emergence and wide use of the laptop computer and the evolution and greater acceptance of the Internet have also played a part in database software development. Laptops have become quite powerful, with gigabytes of memory and storage, and extremely fast processing power. They've become so ubiquitous that they've all but replaced the desktop computer in many environments. They've also allowed people to be connected to the Internet even in such mundane places as coffee shops, restaurants, and airports. (And I won't even mention the plethora of other devices that now allow the same type of access—that's for another book and another discussion.) As such, there's been a greater push by both software vendors and businesses to run database software and manage databases from the Internet, thus allowing people to access their applications and data from anywhere at any time. It will be interesting to see how this whole idea progresses over the next several years.

Vendors continue to add new features and enhance the tool sets in their database software, 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. Then you 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* 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, or knows too little about the software he's using to implement the database. 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 Third 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.
- The discussion on data types has been updated.
- The Recommended Reading section includes the latest editions of the books and now includes each book's ISBN.
- A new appendix on Normalization very briefly explains the concept and then explains in detail how it is incorporated into the design process presented in this book.

Visit Informit.com/titles/0321884493 to access additional content referenced in the book.

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. It's better that you learn how to create a database properly from the beginning than that you learn 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.

It would be a good idea for you to read this book 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 there were parts of the design process that you just did not completely understand. Those points with which you had difficulty will finally become clear once 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 here, you'll probably find that there are 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 here from a different viewpoint. At the very least, this book can serve as a great refresher course in database design.

The Purpose of This Book

In general terms, there are three phases to the overall database development process.

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.

There are many database design books out on the market that 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 it can be difficult for a reader to obtain 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 are important topics 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

Here's 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 in order 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 in order 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 it's a good idea to review the final design of the database structure 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 it may be necessary to stray from the techniques and concepts of the design process. This chapter tells you when you should consider bending the rules, as well as how it should be done.

Part IV: Appendixes

These appendices 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 of 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 there are a wide variety of examples in this book. 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, there are many ways that you can design an order-tracking database. 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.*

A New Approach to Learning

Here's an 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 once 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. Once 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	Michael J. Hernandez	First Name, Middle Initial, 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 of 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

8

Keys

A fact in itself is nothing. It is valuable only for the idea attached to it, or for the proof which it furnishes.
—Claude Bernard

Topics Covered in This Chapter

By now you've identified all the subjects that the database will track and defined the table structures that will represent those subjects. Furthermore, you've put the structures through a screening process to control their makeup and quality. In this next stage of the database design process, you'll begin the task of assigning *keys* to each table. You'll soon learn that there are different types of keys, and each plays a particular role within the database structure. All but one key is assigned during this stage; you'll assign the remaining key later (in Chapter 10, "Table Relationships") as you establish relationships between tables.

Why Keys Are Important

Keys are crucial to a table structure for the following reasons.

- *They ensure that each record in a table is precisely identified.* As you already know, a table represents a singular collection of similar objects or events. (For example, a CLASSES table represents a *collection* of classes, not just a single class.) The complete set of records within the table constitutes the collection, and each record represents a unique instance of the table's subject within that collection. You must have some means of accurately identifying each instance, and a key is the device that allows you to do so.
- *They help establish and enforce various types of integrity.* Keys are a major component of table-level integrity and relationship-level integrity. For instance, they enable you to ensure that a table has unique records and that the fields you use to establish a relationship between a pair of tables always contain matching values.
- *They serve to establish table relationships.* As you'll learn in Chapter 10, you'll use keys to establish a relationship between a pair of tables.

Always make certain that you define the appropriate keys for each table. Doing so will help you guarantee that the table structures are sound, that redundant data within each table is minimal, and that the relationships between tables are solid.

Establishing Keys for Each Table

Your next task is to establish keys for each table in the database. There are four main types of keys: *candidate, primary, foreign,* and *non-keys.* A key's type determines its function within the table.

Candidate Keys

The first type of key you establish for a table is the *candidate* key, which is a field or set of fields that uniquely identifies a single instance of the table's subject. Each table must have *at least one* candidate key. You'll eventually examine the table's pool of available candidate keys and designate one of them as the official primary key for the table.

Before you can designate a field as a candidate key, you must make certain it complies with *all* of the Elements of a Candidate Key. These elements constitute a set of guidelines you can use to determine whether the field is fit to serve as a candidate key. You cannot designate a field as a candidate key if it fails to conform to *any* of these elements.

Elements of a Candidate Key

- *It cannot be a multipart field.* You've seen the problems with multipart fields, so you know that using one as an identifier is a bad idea.
- *It must contain unique values.* This element helps you guard against duplicating a given record within the table. Duplicate records are just as bad as duplicate fields, and you must avoid them at all costs.
- *It cannot contain null values.* As you already know, a null value represents the absence of a value. There's absolutely no way a candidate key field can identify a given record if its value is null.
- *Its value cannot cause a breach of the organization's security or privacy rules.* Values such as passwords and Social Security numbers are not suitable for use as a candidate key.
- *Its value is not optional in whole or in part.* A value that is optional implies that it may be null at some point. You can infer, then, that an optional value automatically violates the previous element and is, therefore, unacceptable. (This caveat is especially

applicable when you want to use two or more fields as a candidate key.)

- *It comprises a minimum number of fields necessary to define uniqueness.* You can use a combination of fields (treated as a single unit) to serve as a candidate key, so long as each field contributes to defining a unique value. Try to use as few fields as possible, however, because overly complex candidate keys can ultimately prove to be difficult to work with and difficult to understand.
- *Its values must uniquely and exclusively identify each record in the table.* This element helps you guard against duplicate records and ensures that you can accurately reference any of the table's records from other tables in the database.
- *Its value must exclusively identify the value of each field within a given record.* This element ensures that the table's candidate keys provide the only means of identifying each field value within the record. (You'll learn more about this particular element in the section on primary keys.)
- *Its value can be modified only in rare or extreme cases.* You should never change the value of a candidate key unless you have an absolute and compelling reason to do so. A field is likely to have difficulty conforming to the previous elements if you can change its value arbitrarily.

Establishing a candidate key for a table is quite simple: Look for a field or set of fields that conforms to all of the Elements of a Candidate Key. You'll probably be able to define more than one candidate key for a given table. Loading a table with sample data will give you the means to identify potential candidate keys accurately. (You used this same technique in the previous chapter.)

See if you can identify any candidate keys for the table in Figure 8.1.

Employees

Employee ID	Social Security Number	EmpFirst Name	EmpLast Name	EmpStreet Address	EmpCity	EmpState	EmpZipcode	EmpHome Phone
1000	987-65-9938	Kira	Bently	1204 Bryant Road	Seattle	WA	98157	363-9948
1001	987-65-6531	Katherine	Erlich	101 C Street, Apt. 32	Bellevue	WA	98046	322-6992
1002	987-65-0039	Timothy	Ennis	7402 Kingman Drive	Redmond	WA	98115	527-4992
1003	987-65-1299	Shannon	Black	4141 Lake City Way	Seattle	WA	98136	336-5992
1004	987-65-6529	Susan	Black	2100 Mineola Avenue	Seattle	WA	98115	572-9948
1005	987-65-5583	Estela	Rosales	101 C Street, Apt. 32	Bellevue	WA	98046	322-6992
1006	987-65-1734	Timothy	Sherman	66 NE 120th	Bothell	WA	98216	522-3232

Figure 8.1 *Are there any candidate keys in this table?*

You probably identified EMPLOYEE ID, SOCIAL SECURITY NUMBER, EMPLAST NAME, EMPFIRST NAME *and* EMPLAST NAME, EMPZIPCODE, and EMPHOME PHONE as potential candidate keys. But you'll need to examine these fields more closely to determine which ones are truly eligible to become candidate keys. Remember that you must automatically disregard any field(s) failing to conform to even *one* of the Elements of a Candidate Key.

Upon close examination, you can draw the following conclusions.

- *EMPLOYEE ID is eligible.* This field conforms to every element of a candidate key.
- *SOCIAL SECURITY NUMBER is ineligible because it could contain null values and will most likely compromise the organization's privacy rules.* Contrary to what the sample data shows, this field could contain a null value. For example, there are many people working in the United States who do not have Social Security numbers because they are citizens of other countries.

❖ **Note** Despite its widespread use in many types of databases, I strongly recommend that you refrain from using SOCIAL SECURITY NUMBER as a candidate key (or as a primary key, for that matter)

continues

in any of your database structures. In many instances, it doesn't conform to the Elements of a Candidate Key.

The Philadelphia Region section of the Social Security Online web site provides some very interesting facts about Social Security numbers and identify theft, which is yet another good reason why you should avoid using SSNs as candidate/primary keys. You can access their site here: www.ssa.gov/phila/ProtectingSSNs.htm.

- *EMPLAST NAME is ineligible because it can contain duplicate values.* As you've learned, the values of a candidate key must be unique. In this case there can be more than one occurrence of a particular last name.
- *EMPFIRST NAME and EMPLAST NAME are eligible.* The combined values of both fields will supply a unique identifier for a given record. Although multiple occurrences of a particular first name or last name will occur, the combination of a given first name and last name will always be unique. (Some of you are probably saying, "This is not necessarily always true." You're absolutely right. Don't worry; we'll address this issue shortly.)
- *EMPZIPCODE is ineligible because it can contain duplicate values.* Many people live in the same zip code area, so the values in EMPZIPCODE cannot possibly be unique.
- *EMPHOME PHONE is ineligible because it can contain duplicate values and is subject to change.* This field will contain duplicate values for either of these two reasons.

  1. One or more family members work for the organization.
  2. One or more people share a residence that contains a single phone line.

You can confidently state that the EMPLOYEES table has two candidate keys: EMPLOYEE ID and the combination of EMPFIRST NAME and EMPLAST NAME.

Mark candidate keys in your table structures by writing the letters "CK" next to the name of each field you designate as a candidate key. A candidate key composed of two or more fields is known as a *composite candidate key,* and you'll write "CCK" next to the names of the fields that make up the key. When you have two or more composite candidate keys, use a number within the mark to distinguish one from another. If you had two composite candidate keys, for example, you would mark one as "CCK1" and the other as "CCK2."

Apply this technique to the candidate keys for the EMPLOYEES table in Figure 8.1. Figure 8.2 shows how your structure should look when you've completed this task.

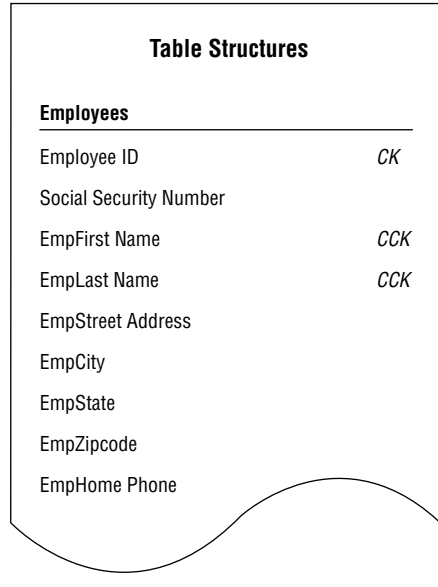


Figure 8.2 *Marking candidate keys in the EMPLOYEES table structure*

Parts

Part Name	Model Number	Manufacturer Name	Retail Price
Shimka XT Cranks	XT-113	Shimka Incorporated	199.95
Faust Brake Levers	BL / 45	Faust USA	53.79
MiniMite Pump		MiniMite	35.00
Hobo Fanny Pack		Hobo Bike Company	59.00
Diablo Bike Pedals	Mtn-A26	Diablo Sports	129.50
Shimka Truing Stand	SP-100		37.95
Faust Brake Levers	BL / 60	Faust USA	79.95

Figure 8.3 *Can you identify any candidate keys in the PARTS table?*

Now try to identify as many candidate keys as you can for the PARTS table in Figure 8.3.

At first glance, you may believe that PART NAME, MODEL NUMBER, the combination of PART NAME and MODEL NUMBER, and the combination of MANUFACTURER NAME and PART NAME are potential candidate keys. After investigating this theory, however, you come up with the following results.

- *PART NAME is ineligible because it can contain duplicate values.* A given part name will be duplicated when the part is manufactured in several models. For example, this is the case with Faust Brake Levers.
- *MODEL NUMBER is ineligible because it can contain null values.* A candidate key value must exist for each record in the table. As you can see, some parts do not have a model number.
- *PART NAME and MODEL NUMBER are ineligible because either field can contain null values.* The simple fact that MODEL NUMBER can contain null values instantly disqualifies this combination of fields.
- *MANUFACTURER NAME and PART NAME are ineligible because the values for these fields seem to be optional.* Recall that a candidate key

value cannot be optional in whole or in part. In this instance, you can infer that entering the manufacturer name is optional when it appears as a component of the part name; therefore, you cannot designate this combination of fields as a candidate key.

It's evident that you don't have a single field or set of fields that qualifies as a candidate key for the PARTS table. This is a problem because each table must have at least *one* candidate key. Fortunately, there is a solution.

Artificial Candidate Keys

When you determine that a table does not contain a candidate key, you can create and use an *artificial* (or *surrogate*) candidate key. (It's artificial in the sense that it didn't occur "naturally" in the table; you have to manufacture it.) You establish an artificial candidate key by creating a new field that conforms to all of the Elements of a Candidate Key and then adding it to the table; this field becomes the official candidate key.

You can now solve the problem in the PARTS table. Create an artificial candidate key called PART NUMBER and assign it to the table. (The new field will automatically conform to the Elements of a Candidate Key because you're creating it from scratch.) Figure 8.4 shows the revised structure of the PARTS table.

When you've established an artificial candidate key for a table, mark the field name with a "CK" in the table structure, just as you did for the EMPLOYEES table in the previous example.

You may also choose to create an artificial candidate key when it would be a stronger (and thus, more appropriate) candidate key than any of the existing candidate keys. Assume you're working on an EMPLOYEES table and you determine that the only available candidate key is the combination of the EMPFIRST NAME and EMPLAST NAME

Parts

Part Number	Part Name	Model Number	Manufacturer Name	Retail Price
41000	Shimka XT Cranks	XT-113	Shimka Incorporated	199.95
41001	Faust Brake Levers	BL / 45	Faust USA	53.79
41002	MiniMite Pump		MiniMite	35.00
41003	Hobo Fanny Pack		Hobo Bike Company	59.00
41004	Diablo Bike Pedals	Mtn-A26	Diablo Sports	129.50
41005	Shimka Truing Stand	SP-100		37.95
41006	Faust Brake Levers	BL / 60	Faust USA	79.95

Figure 8.4 *The PARTS table with the artificial candidate key* PART NUMBER

fields. Although this may be a valid candidate key, using a single-field candidate key might prove more efficient and may identify the subject of the table more easily. Let's say that everyone in the organization is accustomed to using a unique identification number rather than a name as a means of identifying an employee. In this instance, you can choose to create a new field named EMPLOYEE ID and use it as an artificial candidate key. This is an absolutely acceptable practice—do this without hesitation or reservation if you believe it's appropriate.

❖ **Note** I commonly create an ID field (such as EMPLOYEE ID, VENDOR ID, DEPARTMENT ID, CATEGORY ID, and so on) and use it as an artificial candidate key. It always conforms to the Elements of a Candidate Key, makes a great primary key (eventually), and, as you'll see in Chapter 10, makes the process of establishing table relationships much easier.

Review the candidate keys you've selected and make absolutely certain that they thoroughly comply with the Elements of a Candidate Key. Don't be surprised if you discover that one of them is not a candidate key after all—incorrectly identifying a field as a candidate key happens

occasionally. When this does occur, just remove the "CK" designator from the field name in the table structure. Deleting a candidate key won't pose a problem so long as the table has more than one candidate key. If you discover, however, that the only candidate key you identified for the table is *not* a candidate key, you *must* establish an artificial candidate key for the table. After you've defined the new candidate key, remember to mark its name with a "CK" in the table structure.

Primary Keys

By now, you've established all the candidate keys that seem appropriate for every table. Your next task is to establish a *primary* key for each table, which is the most important key of all.

- A *primary key field* exclusively identifies the table throughout the database structure and helps establish relationships with other tables. (You'll learn more about this in Chapter 10.)
- A *primary key value* uniquely identifies a given record within a table and exclusively represents that record throughout the entire database. It also helps to guard against duplicate records.

A primary key must conform to the exact same elements as a candidate key. This requirement is easy to fulfill because you select a primary key from a table's pool of available candidate keys. The process of selecting a primary key is somewhat similar to that of a presidential election. Every four years, several people run for the office of President of the United States. These individuals are known as "candidates" and they have all of the qualifications required to become president. A national election is held, and a single individual from the pool of available presidential candidates is elected to serve as the country's official president. Similarly, you identify each qualified candidate key in the table, run your own election, and select one of them to become the official primary key of the table. You've already identified the candidates, so now it's election time!

Assuming that there is no other marginal preference, here are a couple of guidelines you can use to select an appropriate primary key.

1. *If you have a simple (single-field) candidate key and a composite candidate key, choose the simple candidate key.* It's always best to use a candidate key that contains the least number of fields.
2. *Choose a candidate key that incorporates part of the table name within its own name.* For example, a candidate key with a name such as SALES INVOICE NUMBER is a good choice for the SALES INVOICES table.

Examine the candidate keys and choose one to serve as the primary key for the table. The choice is largely arbitrary—you can choose the one that you believe most accurately identifies the table's subject or the one that is the most meaningful to everyone in the organization. For example, consider the EMPLOYEES table again in Figure 8.5.

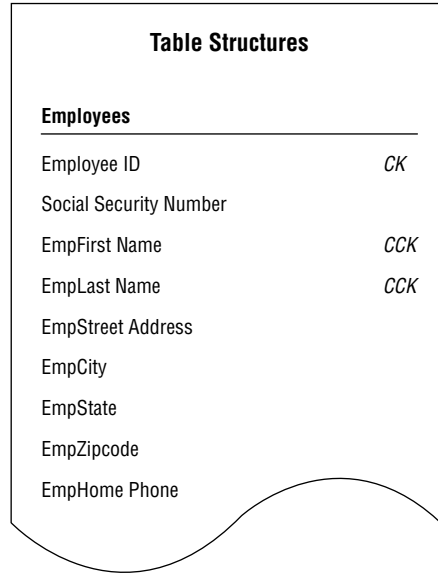


Figure 8.5 *Which candidate key should become the primary key of the EMPLOYEES table?*

Either of the candidate keys you identified within the table could serve as the primary key. You might decide to choose Employee ID if everyone in the organization is accustomed to using this number as a means of identifying employees in items such as tax forms and employee benefits programs. The candidate key you ultimately choose becomes the primary key of the table and is governed by the Elements of a Primary Key. These elements are exactly the same as those for the candidate key, and you should enforce them to the letter. For the sake of clarity, here are the Elements of a Primary Key:

Elements of a Primary Key

- It cannot be a multipart field.
- It must contain unique values.
- It cannot contain null values.
- Its value cannot cause a breach of the organization's security or privacy rules.
- Its value is not optional in whole or in part.
- It comprises a minimum number of fields necessary to define uniqueness.
- Its values must uniquely and exclusively identify each record in the table.
- Its value must exclusively identify the value of each field within a given record.
- Its value can be modified only in rare or extreme cases.

Before you finalize your selection of a primary key, it is imperative that you make absolutely certain that the primary key fully complies with this particular element:

> Its value must exclusively identify the value of each field within a given record.

Each field value in a given record should be unique throughout the entire database (unless it is participating in establishing a relationship between a pair of tables) and should have *only one* exclusive means of identification—the specific primary key value for that record.

You can determine whether a primary key fully complies with this element by following these steps.

1. Load the table with sample data.
2. Select a record for test purposes and note the current primary key value.
3. Examine the value of the first field (the one immediately after the primary key) and ask yourself this question:

   Does this primary key value *exclusively* identify the current value of *<fieldname>?*

   a. If the answer is yes, move to the next field and repeat the question.

   b. If the answer is no, *remove the field from the table,* move to the next field, and repeat the question.
4. Continue this procedure until you've examined every field value in the record.

A field value that the primary key *does not* exclusively identify indicates that the field itself is *unnecessary* to the table's structure; therefore, you should remove the field and reconfirm that the table complies with the Elements of the Ideal Table. You can then add the field you just removed to another table structure, if appropriate, or you can discard it completely because it is truly unnecessary.

Here's an example of how you might apply this technique to the partial table structure in Figure 8.6. (Note that INVOICE NUMBER is the primary key of the table.)

Sales Invoices

Invoice Number	Invoice Date	CustFirst Name	CustLast Name	EmpFirst Name	EmpLast Name	EmpHome Phone
13000	06/15/02	Frank	DeSoto	Estela	Rosales	363-9948
13001	06/15/02	Gregory	Mattson	Katherine	Erlich	322-6992
13002	06/15/02	Carmen	Aguilar	Kira	Bently	527-4992
13003	06/16/02	David	Cunningham	Kira	Bently	336-5992
13004	06/16/02	Carmen	Aguilar	Shannon	Black	572-9948
13005	06/17/02	Frank	DeSoto	Estela	Rosales	322-6992

Figure 8.6 *Does the primary key exclusively identify the value of each field in this table?*

First, you load the table with sample data. You then select a record for test purposes—we'll use the third record for this example—and note the value of the primary key (13002). Now, pose the following question for each field value in the record.

> Does this primary key value *exclusively* identify the current value of . . .
>
> INVOICE DATE? Yes, it does. This invoice number will always identify the specific date that the invoice was created.
>
> CUSTFIRST NAME? Yes, it does. This invoice number will always identify the specific first name of the particular customer who made this purchase.
>
> CUSTLAST NAME? Yes, it does. This invoice number will always identify the specific last name of the particular customer who made this purchase.
>
> EMPFIRST NAME? Yes, it does. This invoice number will always identify the specific first name of the particular employee who served the customer for this sale.

EMPLAST NAME? Yes, it does. This invoice number will always identify the specific last name of the particular employee who served the customer for this sale.

EMPHOME PHONE? *No,* it doesn't! The invoice number *indirectly* identifies the employee's home phone number via the employee's name. In fact, it is the *current value* of both EMPFIRST NAME and EMPLAST NAME that exclusively identifies the value of EMPHOME PHONE—change the employee's name and you *must* change the phone number as well. You should now remove EMPHOME PHONE from the table for two reasons: The primary key does not exclusively identify its current value and (as you've probably already ascertained) it is an unnecessary field. As it turns out, you can discard this field completely because it is already part of the EMPLOYEES table structure.

After you've removed the unnecessary fields you identified during this test, examine the revised table structure and make sure it complies with the Elements of the Ideal Table.

The primary key should now exclusively identify the values of the remaining fields in the table. This means that the primary key is truly sound and you can designate it as the official primary key for the table. Remove the "CK" next to the field name in the table structure and replace it with a "PK." (A primary key composed of two or more fields is known as a *composite primary key,* and you mark it with the letters "CPK.") Figure 8.7 shows the revised structure of the SALES INVOICES table with INVOICE NUMBER as its primary key.

As you create a primary key for each table in the database, keep these two rules in mind:

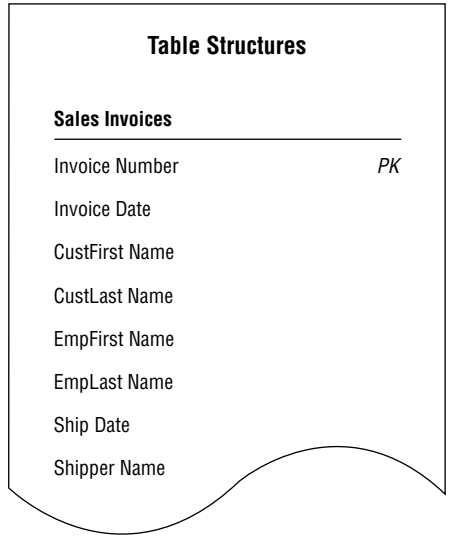


Figure 8.7 *The revised SALES INVOICES table with its new primary key*

Rules for Establishing a Primary Key

1. *Each table must have one—and only one—primary key.* Because the primary key *must* conform to each of the elements that govern it, only one primary key is necessary for a particular table.
2. *Each primary key within the database must be unique—no two tables should have the same primary key unless one of them is a subset table.* You learned at the beginning of this section that the primary key exclusively identifies a table throughout the database structure; therefore, each table must have its own *unique* primary key in order to avoid any possible confusion or ambiguity concerning the table's identity. A subset table is excluded from this rule because it represents a more specific version of a particular data table's subject—both tables *must* share the same primary key.

Later in the database design process, you'll learn how to use the primary key to help establish a relationship between a pair of tables.

Alternate Keys

Now that you've selected a candidate key to serve as the primary key for a particular table, you'll designate the remaining candidate keys as *alternate* keys. These keys can be useful to you in an RDBMS program because they provide an alternative means of uniquely identifying a particular record within the table. If you choose to use an alternate key in this manner, mark its name with "AK" or "CAK" (composite alternate key) in the table structure; otherwise, remove its designation as an alternate key and simply return it to the status of a normal field. You won't be concerned with alternate keys for the remainder of the database design process, but you will work with them once again as you implement the database in an RDBMS program. (Implementing and using alternate keys in RDBMS programs is beyond the scope of this work—our only objective here is to designate them as appropriate. This is in line with the focus of the book, which is the logical design of a database.)

Figure 8.8 shows the final structure for the EMPLOYEES table with the proper designation for both the primary key and the alternate keys.

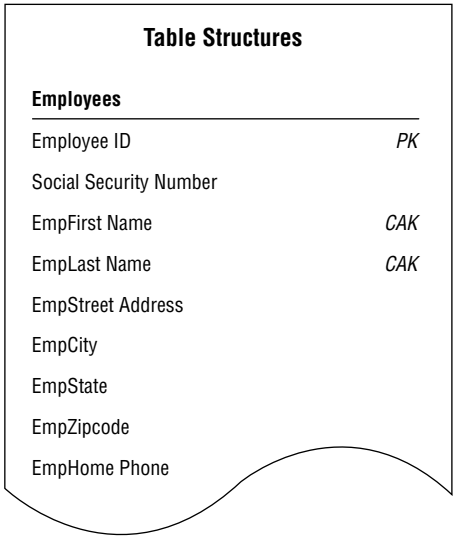


Figure 8.8 *The EMPLOYEES table with designated primary and alternate keys*

Non-keys

A *non-key* is a field that does not serve as a *candidate, primary, alternate,* or *foreign* key. Its sole purpose is to represent a characteristic of the table's subject, and its value is determined by the primary key. There is no particular designation for a non-key, so you don't need to mark it in the table structure.

Table-Level Integrity

This type of integrity is a major component of overall data integrity, and it ensures the following.

- There are no duplicate records in a table.
- The primary key exclusively identifies each record in a table.
- Every primary key value is unique.
- Primary key values are not null.

You began establishing table-level integrity when you defined a primary key for each table and ensured its enforcement by making absolutely certain that each primary key fully complied with the Elements of a Primary Key. In the next chapter, you'll enhance the table's integrity further as you establish *field specifications* for each field within the table.

Reviewing the Initial Table Structures

Now that the fundamental table definitions are complete, you need to conduct interviews with users and management to review the work you've done so far. This set of interviews is fairly straightforward and should be relatively easy to conduct.

During these interviews, you will accomplish these tasks.

- *Ensure that the appropriate subjects are represented in the database.* Although it's highly unlikely that an important subject is missing at this stage of the database design process, it can happen. When it does happen, identify the subject, use the proper techniques to transform it into a table, and develop it to the same degree as the other tables in the database.
- *Make certain that the table names and table descriptions are suitable and meaningful to everyone.* When a name or description appears to be confusing or ambiguous to several people in the organization, work with them to clarify the item as much as possible. It's common for some table names and descriptions to improve during the interview process.
- *Make certain that the field names are suitable and meaningful to everyone.* Selecting field names typically generates a great deal of discussion, especially when there is an existing database in place. You'll commonly find people who customarily refer to a particular field by a certain name because "that's what it's called on my screen." When you change a field name—you have good reasons for doing so—you must diplomatically explain to these folks that you renamed the field so that it conforms to the standards imposed by the new database. You can also tell them that the field can appear with the more familiar name once the database is implemented in an RDBMS program. What you've said is true; many RDBMSs allow you to use one name for the field's physical definition and another name for display purposes. This feature, however, does not change, reduce, or negate the need for you to follow the guidelines for creating field names that you learned in Chapter 7, "Establishing Table Structures."
- *Verify that all the appropriate fields are assigned to each table.* This is your best opportunity to make certain that all of the necessary characteristics pertaining to the subject of the table are

in place. You'll commonly discover that you accidentally overlooked one or two characteristics earlier in the design process. When this happens, identify the characteristics, use the appropriate techniques to transform them into fields, and follow all the necessary steps to add them to the table.

When you've completed the interviews, you'll move to the next phase of the database design process and establish *field specifications* for every field in the database.

CASE STUDY

It's now time to establish keys for each table in the Mike's Bikes database. As you know, your first order of business is to establish candidate keys for each table. Let's say you decide to start with the CUSTOMERS table in Figure 8.9.

As you review each field, you try to determine whether it conforms to the Elements of a Candidate Key. You determine that STATUS, CUSTHOME

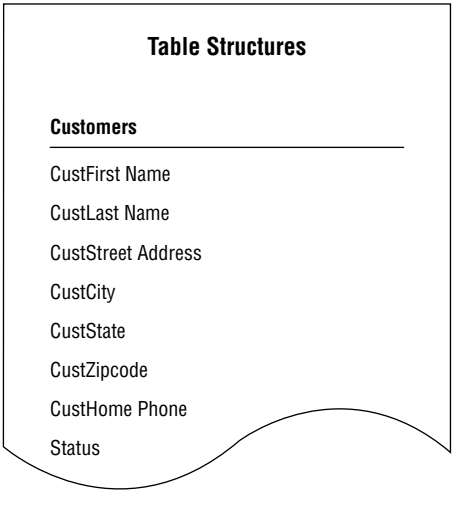


Figure 8.9 *The CUSTOMERS table structure in the Mike's Bikes database*

PHONE, and the combination of CUSTFIRST NAME and CUSTLAST NAME are potential candidate keys, but you're not quite certain whether any of them will completely conform to all of the elements. So you decide to test the keys by loading the table with sample data as shown in Figure 8.10.

Customers

CustFirst Name	CustLast Name	CustStreet Address	CustCity	CustState	CustZipcode	CustHome Phone	Status
Bridget	Berlin	2121 NE 35th	Bellevue	WA	98004	422-4982	Valued
Phillip	Bradley	101 9th Avenue	Kent	WA	98126	322-1178	
Kel	Brigan	7525 Taxco Lane	Redmond	WA	98225	363-9360	Valued
Barbara	Carmichael	7525 Taxco Lane	Redmond	WA	98225	363-9360	Preferred
Daniel	Chavez	750 Pike Street	Bothell	WA	98001	441-3987	Valued
Daniel	Chavez	301 N Main	Seattle	WA	98115	365-7199	
Sandi	Cooper	115 Pine Place	Seattle	WA	98026	332-0499	Preferred

Figure 8.10 *Testing candidate keys in the CUSTOMERS table*

Always remember that a field must comply with *all* of the Elements of a Candidate Key in order to qualify as a candidate key. You must immediately disqualify the field if it does not fulfill this requirement.

As you examine the table, you draw these conclusions.

- *STATUS is ineligible because it will probably contain duplicate values.* As business grows, Mike is going to have many "Valued" customers.
- *CUSTHOME PHONE is ineligible because it will probably contain duplicate values.* The sample data reveals that two customers can live in the same residence and have the same phone number.
- *CUSTFIRST NAME and CUSTLAST NAME are ineligible because they will probably contain duplicate values.* The sample data reveals that the combination of first name and last name can represent more than one distinct customer.

These findings convince you to establish an artificial candidate key for this table. You then create a field called CUSTOMER ID, confirm that it complies with the requirements for a candidate key, and add the new field to the table structure with the appropriate designation.

Figure 8.11 shows the revised structure of the CUSTOMERS table.

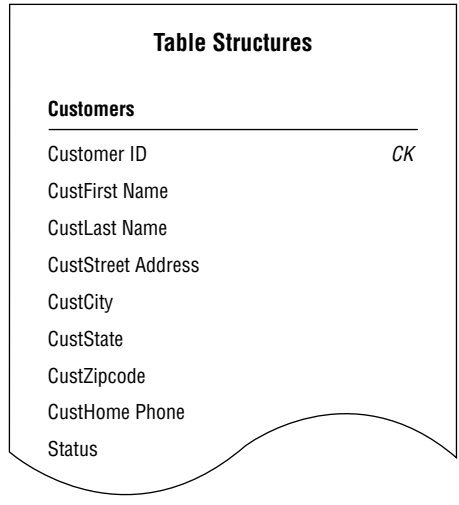


Figure 8.11 *The CUSTOMERS table with the new artificial candidate key, CUSTOMER ID*

Now you'll repeat this procedure for each table in the database. Remember to make certain that every table has at least *one* candidate key.

The next order of business is to establish a primary key for each table. As you know, you select the primary key for a particular table from the table's pool of available candidate keys. Here are a few points to keep in mind when you're choosing a primary key for a table with more than one candidate key.

- Choose a simple (single-field) candidate key over a composite candidate key.

- If possible, pick a candidate key that has the table name incorporated into its own name.
- Select the candidate key that best identifies the subject of the table or is most meaningful to everyone in the organization.

You begin by working with the EMPLOYEES table in Figure 8.12. As you review the candidate keys, you decide that EMPLOYEE NUMBER is a much better choice for a primary key than the combination of EMPFIRST NAME and EMPLAST NAME because Mike's employees are already accustomed to identifying themselves by their assigned numbers. Using EMPLOYEE NUMBER makes perfect sense, so you select it as the primary key for the table.

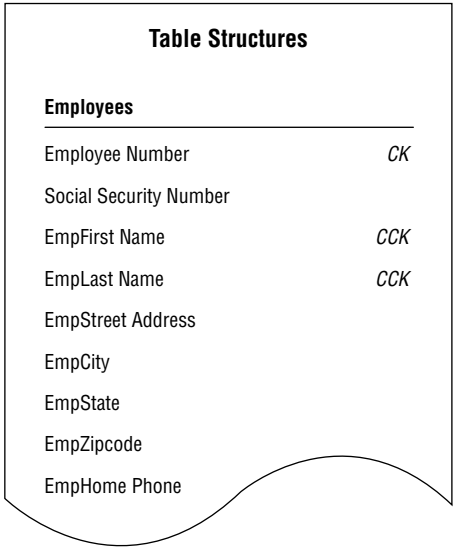


Figure 8.12 *The EMPLOYEES table structure in the Mike's Bikes database*

Now you perform one final task before you designate EMPLOYEE NUMBER as the official primary key of the table: You make absolutely certain that it exclusively identifies the value of each field within a given record. So you test EMPLOYEE NUMBER by following these steps.

1. Load the EMPLOYEES table with sample data.
2. Select a record for test purposes and note the current value of EMPLOYEE NUMBER.
3. Examine the value of the first field (the one immediately after EMPLOYEE NUMBER) and ask yourself this question:

   Does this primary key value *exclusively* identify the current value of *<fieldname>?*

   a. If the answer is yes, move to the next field and repeat the question.

   b. If the answer is no, *remove the field from the table,* move to the next field, and repeat the question. (Be sure to determine whether you can add the field you just removed to another table structure, if appropriate, or discard it completely because it is truly unnecessary.)
4. Continue this procedure until you've examined every field value in the record.

You know that you'll have to remove any field containing a value that EMPLOYEE NUMBER *does not* exclusively identify. EMPLOYEE NUMBER does exclusively identify the value of each field in the test record, however, so you use it as the official primary key for the EMPLOYEES table and mark its name with the letters "PK" in the table structure. You then repeat this process with the rest of the tables in Mike's new database until every table has a primary key.

Remember to keep these rules in mind as you establish primary keys for each table.

- Each table must have one—and only one—primary key.
- Each primary key within the database should be unique—no two tables should have the same primary key (unless one of them is a subset table).

As you work through the tables in Mike's database, you remember that the SERVICES table is a subset table. You created it during the previous stage of the design process (in Chapter 7), and it represents a more specific version of the subject represented by the PRODUCTS table. The PRODUCT NAME field is what currently relates the PRODUCTS table to the SERVICES subset table. You now know, however, that a subset table *must* have the same primary key as the table to which it is related, so you'll use PRODUCT NUMBER (the primary key of the PRODUCTS table) as the primary key of the SERVICES table. Figure 8.13 shows the PRODUCTS and SERVICES tables with their primary keys.

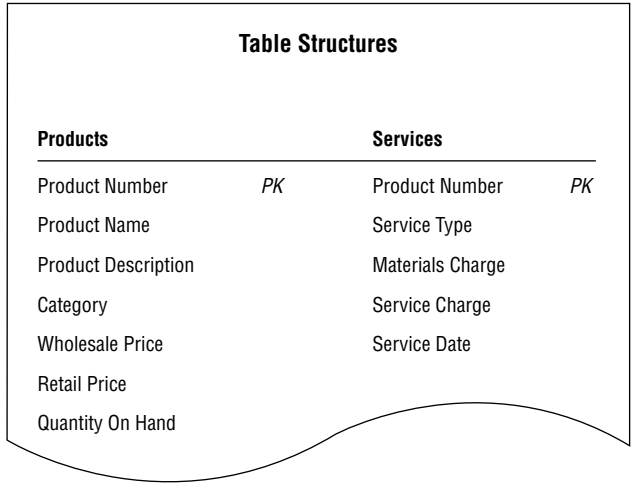


Figure 8.13 *Establishing the primary key for the SERVICES subset table*

The last order of business is to conduct interviews with Mike and his staff and review all the work you've performed on the tables in the database. As you conduct these interviews, make certain you check the following.

- The appropriate subjects are represented in the database.
- The table names and descriptions are suitable and meaningful to everyone.

- The field names are suitable and meaningful to everyone.
- All the appropriate fields are assigned to each table.

By the end of the interview, everyone agrees that the tables are in good form and that all the subjects with which they are concerned are represented in the database. Only one minor point came up during the discussions: Mike wants to add a CALL PRIORITY field to the VENDORS table. There are instances in which more than one vendor supplies a particular product, and Mike wants to create a way to indicate which vendor he should call first if that product is unexpectedly out of stock. So you add the new field to the VENDORS table and bring the interview to a close.

Summary

The chapter opened with a discussion of the importance of *keys.* You learned that there are different types of keys, and each type plays a different role within the database. Each key performs a particular function, such as uniquely identifying records, establishing various types of integrity, and establishing relationships between tables. You now know that you can guarantee sound table structure by making certain that the appropriate keys are established for each table.

We then discussed the process of establishing keys for each table. We began by identifying the four main types of keys: *candidate, primary, foreign,* and *non-keys.* First, we looked at the process of establishing candidate keys for each table. You learned about the Elements of a Candidate Key and how to make certain that a field (or set of fields) complies with these elements. Then you learned that you can create and use an artificial candidate key when none of the fields in a table can serve as a candidate key or when a new field would make a stronger candidate key than any of the existing candidate key fields.

The chapter continued with a discussion of *primary keys.* You learned that you select a primary key from a table's pool of candidate keys and that the primary key is governed by a set of specific elements. We then covered a set of guidelines that help you determine which candidate key to use as a primary key. Next, you learned how to ensure that the chosen primary key exclusively identifies a given record and its set of field values. When the primary key does not exclusively identify a particular field value, you know that you must remove the field from the table in order to ensure the table's structural integrity. You also know that each table must have a single, unique primary key.

You then learned that you designate any remaining candidate keys as *alternate keys.* These keys will be most useful to you when you implement the database in an RDBMS program because they provide an alternate means of identifying a given record. We then discussed the *non-key* field, which is any field not designated as a candidate, primary, alternate, or foreign key. You now know that a non-key field represents a characteristic of the table's subject and that the primary key exclusively identifies its value.

Table-level integrity was the next subject of discussion, and you learned that it is established through the use of primary keys and enforced by the Elements of a Primary Key.

The chapter closed with some guidance on conducting further *interviews* with users and management. You now know that these interviews provide you with a means of reviewing the work you have performed on the tables and help you to verify and validate the current database structure.

Review Questions

1. State the three reasons why keys are important.
2. What are the four main types of *keys?*

3. What is the purpose of a *candidate key?*
4. State four items of the Elements of a Candidate Key.
5. True or False: A candidate key can be composed of more than one field.
6. Can a table have more than one candidate key?
7. What is an *artificial candidate key?*
8. What is the most important key you assign to a table?
9. Why is this key important?
10. How do you establish a *primary key?*
11. State four items of the *Elements of a Primary Key.*
12. What must you do before you finalize your selection of a primary key?
13. What is an *alternate key?*
14. What do you ensure by establishing table-level integrity?
15. Why should you review the initial table structures?

This page intentionally left blank

Index

C

D

G

H

I

K

L

P

Q

R

S

T

U

V

W

Z