Second Edition

Beginning Database Design Solutions

Understanding and Implementing Database Design Concepts for the Cloud and Beyond

Rod Stephens

WILEY

BEGINNING DATABASE DESIGN SOLUTIONS

BEGINNING
Database Design Solutions

BEGINNING

Database Design Solutions

UNDERSTANDING AND IMPLEMENTING DATABASE DESIGN CONCEPTS FOR THE CLOUD AND BEYOND

Second Edition

Rod Stephens

WILEY

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada and the United Kingdom.

ISBN: 978-1-394-15572-9
ISBN: 978-1-394-15583-5 (ebk.)
ISBN: 978-1-394-15584-2 (ebk.)

Library of Congress Control Number: 2022946646

Cover image: © mfto/Getty Images
Cover design: Wiley

For those who read the back cover, dedication, introductory chapter, and glossary. The five of us need to stick together!

Also to Amy, Ken, and Elissa for sanity-preserving doses of silliness.

ABOUT THE AUTHOR

Rod Stephens started out as a mathematician, but while studying at MIT, he discovered how much fun programming is and he's been programming professionally ever since. He's a longtime developer, instructor, and author who has written more than 250 magazine articles and 37 books that have been translated into languages around the world.

During his career, Rod has worked on an eclectic assortment of applications in such fields as telephone switching, billing, repair dispatching, tax processing, wastewater treatment, concert ticket sales, cartography, optometry, and training for professional football teams. (That's U.S. football, not one of the kinds with the round ball. Or the kind with three downs. Or the kind with an oval field. Or the indoor kind. Let's just say NFL and leave it at that.)

Rod's popular C# Helper website (`www.csharphelper.com`) receives millions of hits per year and contains thousands of tips, tricks, and example programs for C# programmers. His VB Helper website (`www.vb-helper.com`) contains similar material for Visual Basic programmers.

ABOUT THE TECHNICAL EDITOR

John Mueller is a freelance author and technical editor. He has writing in his blood, having produced 123 books and more than 600 articles to date. The topics range from networking to artificial intelligence and from database management to heads-down programming. Some of his current books include discussions of data science, machine learning, and algorithms. He also writes about computer languages such as C++, C#, and Python. His technical editing skills have helped more than 70 authors refine the content of their manuscripts. John has provided technical editing services to a variety of magazines, performed various kinds of consulting, and he writes certification exams. Be sure to read John's blog at `http://blog.johnmuellerbooks.com`. You can reach John on the Internet at `John@JohnMuellerBooks.com`. John also has a website at `www.johnmuellerbooks.com`.

ACKNOWLEDGMENTS

Thanks to Ken Brown, Jan Lynn Neal, Archana Pragash, Melissa Burlock, and all the others who worked so hard to make this book possible.

Thanks also to technical editor and longtime friend John Mueller for giving me the benefit of his valuable experience. You can see what John's up to at `www.johnmuellerbooks.com`.

CONTENTS

INTRODUCTION

It has been estimated that more than 80 percent of all computer programming is database-related. This is certainly easy to believe. After all, a database can be a powerful tool for doing exactly what computer programs do best: store, manipulate, and display data.

Even many programs that seem at first glance to have little to do with traditional business-oriented data use databases to make processing easier. In fact, looking back on 40 some years of software development experience, I'm hard-pressed to think of a single nontrivial application that I've worked on that didn't use some kind of database.

Not only do databases play a role in many applications, but they often play a critical role. If the data is not properly stored, it may become corrupted, and the program will be unable to use it meaningfully. If the data is not properly organized, the program may be unable to find what it needs in a reasonable amount of time.

Unless the database stores its data safely and effectively, the application will be useless no matter how well-designed the rest of the system may be. The database is like the foundation of a building; without a strong foundation, even the best crafted building will fail, sometimes spectacularly (the Leaning Tower of Pisa notwithstanding).

With such a large majority of applications relying so heavily on databases, you would expect everyone involved with application development to have a solid, formal foundation in database design and construction. Everyone, including database designers, application architects, programmers, database administrators, and project managers, should ideally understand what makes a good database design. Even an application's key customers and users could benefit from understanding how databases work.

Sadly, that is usually not the case. Many IT professionals have learned what they know about databases through rumor, trial-and-error, tarot cards, and painful experience. Over the years, some develop an intuitive feel for what makes a good database design, but they may still not understand the reasons a design is good or bad, and they may leave behind a trail of rickety, poorly constructed programs built on shaky database foundations.

This book provides the tools you need to design a database. It explains how to determine what should go in a database and how a database should be organized to ensure data integrity and a reasonable level of performance. It explains techniques for designing a database that is strong enough to store data safely and consistently, flexible enough to allow the application to retrieve the data it needs quickly and reliably, and adaptable enough to accommodate a reasonable amount of change.

With the ideas and techniques described in this book, you will be able to build a strong foundation for database applications.

WHO THIS BOOK IS FOR

This book is intended for IT professionals and students who want to learn how to design, analyze, and understand databases. The material will benefit those who want a better high-level understanding of databases such as proposal managers, architects, project managers, and even customers. The material will also benefit those who will actually design, build, and work with databases such as database designers, database administrators, and programmers. In many projects, these roles overlap so the same person may be responsible for working on the proposal, managing part of the project, and designing and creating the database.

This book is aimed at readers of all experience levels. It does not assume that you have any previous experience with databases or programs that use them. It doesn't even assume that you have experience with computers. All you really need is a willingness and desire to learn.

WHAT THIS BOOK COVERS

This book explains database design. It tells how to plan a database's structure so the database will be robust, resistant to errors, and flexible enough to accommodate a reasonable amount of future change. It explains how to discover database requirements, build data models to study data needs, and refine those models to improve the database's effectiveness.

The book solidifies these concepts by working through a detailed example that designs a (sort of) realistic database. Later chapters explain how to actually build databases using a few different database products. The book finishes by describing topics you need to understand to keep a database running effectively such as database maintenance and security.

WHAT YOU NEED TO USE THIS BOOK

This book explains database design. It tells how to determine what should go in a database and how the database should be structured to give the best results.

This book does not focus on actually *creating* the database. The details of database construction are different for different database tools, so to remain as generally useful as possible, this book doesn't concentrate on any particular database system. You can apply most of the techniques described here equally to whatever database tool you use, whether it's MariaDB, PostgreSQL, SQL Server, or some other database product.

> ***NOTE*** *Most database products include free editions that you can use for smaller projects. For example, SQL Server Express Edition, Oracle Express Edition, and MariaDB Community Server are all free.*

To remain database-neutral, most of the book does not assume you are using a particular database, so you don't need any particular software or hardware. To work through the exercises, all you need is a pencil and some paper. You are welcome to type solutions into your computer if you like, but you may actually find working with pencil and paper easier than using a graphical design tool to draw pictures, at least until you are comfortable with database design and are ready to pick a computerized design tool.

Chapters 16 through 25 build example databases using particular database offerings, so their material is tied to the databases that they demonstrate. Chapter 15, "Example Overview," introduces those chapters and lists the databases that they use.

To experiment with the SQL database language described in Chapter 26, "Introduction to SQL," and Chapter 27, "Building Databases with SQL Scripts," you need any database product that supports SQL (that includes pretty much all relational databases) running on any operating system.

HOW THIS BOOK IS STRUCTURED

The chapters in this book are divided into five parts plus appendixes. The chapters in each part are described here. If you have previous experience with databases, you can use these descriptions to decide which chapters to skim and which to read in detail.

Part I: Introduction to Databases and Database Design

The chapters in this part of the book provide background that is necessary to understand the chapters that follow. You can skim some of this material if it is familiar to you, but don't take it too lightly. If you understand the fundamental concepts underlying database design, it will be easier to understand the point behind important design concepts presented later.

Chapter 1, "Database Design Goals," explains the reasons people and organizations use databases. It explains a database's purpose and conditions that it must satisfy to be useful. This chapter also describes the basic ACID (Atomicity, Consistency, Isolation, Durability) and CRUD (Create, Read, Update, Delete) features that any good database should have. It explains in high-level general terms what makes a good database and what makes a bad database.

Chapter 2, "Relational Overview," explains basic relational database concepts such as tables, rows, and columns. It explains the common usage of relational database terms in addition to the more technical terms that are sometimes used by database theorists. It describes different kinds of constraints that databases use to guarantee that the data is stored safely and consistently.

Chapter 3, "NoSQL Overview," explains the basics of NoSQL databases, which are growing quickly in popularity. Those databases include document, key-value, column-oriented, and graph databases. Both relational and NoSQL databases can run either locally or in the cloud, but many NoSQL databases are more cloud-oriented, largely because they are newer technology so they're cloud-native.

Part II: Database Design Process and Techniques

The chapters in this part of the book discuss the main pieces of relational database design. They explain how to understand what should be in the database, develop an initial design, separate important pieces of the database to improve flexibility, and refine and tune the design to provide the most stable and useful design possible.

Chapter 4, "Understanding User Needs," explains how to learn about the users' needs and gather user requirements. It tells how to study the users' current operations, existing databases (if any), and desired improvements. It describes common questions that you can ask to learn about users' operations, desires, and needs, and how to build the results into requirements documents and specifications. This chapter explains what use cases are and shows how to use them and the requirements to guide database design and to measure success.

Chapter 5, "Translating User Needs into Data Models," introduces data modeling. It explains how to translate the user's conceptual model and the requirements into other, more precise models that define the database design rigorously. This chapter describes several database modeling techniques, including user-interface models, semantic object models, entity-relationship diagrams, and relational models.

Chapter 6, "Extracting Business Rules," explains how a database can handle business rules. It explains what business rules are, how they differ from database structure requirements, and how you can identify business rules. This chapter explains the benefits of separating business rules from the database structure and tells how to achieve that separation.

Chapter 7, "Normalizing Data," explains one of the most important tools in relational database design: normalization. Normalization techniques allow you to restructure a database to increase its flexibility and make it more robust. This chapter explains various forms of normalization, emphasizing the stages that are most common and important: first, second, and third normal forms (1NF, 2NF, and 3NF). It explains how each of these kinds of normalization helps prevent errors and tells why it is sometimes better to leave a database slightly less normalized to improve performance.

Chapter 8, "Designing Databases to Support Software," explains how databases fit into the larger context of application design and the development life cycle. This chapter explains how later development depends on the underlying database design. It discusses multi-tier architectures that can help decouple the application and database so there can be at least some changes to either without requiring changes to both.

Chapter 9, "Using Common Design Patterns," explains some common patterns that are useful in many applications. Some of these techniques include implementing various kinds of relationships among objects, storing hierarchical and network data, recording temporal data, and logging and locking.

Chapter 10, "Avoiding Common Design Pitfalls," explains some common design mistakes that occur in database development. It describes problems that can arise from insufficient planning, incorrect normalization, and obsession with ID fields and performance.

Part III: A Detailed Case Study

If you follow all of the examples and exercises in the earlier chapters, by this point you will have seen all of the major steps for producing a good database design. However, it's often useful to see all the steps in a complicated process put together in a continuous sequence. The chapters in this part of the book walk through a detailed case study following all the phases of database design for the fictitious Pampered Pet database.

Chapter 11, "Defining User Needs and Requirements," walks through the steps required to analyze the users' problem, define requirements, and create use cases. It describes interviews with fictitious customers that are used to identify the application's needs and translate them into database requirements.

Chapter 12, "Building a Data Model," translates the requirements gathered in the previous chapter into a series of data models that precisely define the database's structure. This chapter builds user interface models, entity-relationship diagrams, semantic object models, and relational models to refine the database's initial design. The final relational models match the structure of a relational database fairly closely, so they are easy to implement.

Chapter 13, "Extracting Business Rules," identifies the business rules embedded in the relational model constructed in the previous chapter. It shows how to extract those rules in order to separate them logically from the database's structure. This makes the database more robust in the face of future changes to the business rules.

Chapter 14, "Normalizing and Refining," refines the relational model developed in the previous chapter by normalizing it. It walks through several versions of the database that are in different normal forms. It then selects the degree of normalization that provides a reasonable trade-off between robust design and acceptable performance.

Part IV: Example Programs

Though this book focuses on abstract database concepts that do not depend on a particular database product, it's also worth spending at least some time on more concrete implementation issues. The chapters in this part of the book describe some of those issues and explain how to build simple example programs that demonstrate a few different database products.

Chapter 15, "Example Overview," provides a roadmap for the chapters that follow. It tells which chapters use which databases and how to get the most out of those chapters. Chapters 16 through 25 come in pairs, with the first describing an example in Python and the second describing a similar (although not always identical) program in C#.

Chapters 16 and 17 describe examples that use the popular MariaDB column-oriented relational database running on the local machine.

Chapters 18 and 19 demonstrate the (also popular) PostgreSQL database, also running on the local machine.

Chapters 20 and 21 show how to use the Neo4j AuraDB graph database running in the cloud.

Chapters 22 and 23 describe examples that use the MongoDB Atlas document database, also running in the cloud.

Chapters 24 and 25 demonstrate the Apache Ignite key-value database running locally.

These examples are just intended to get you started. They are relatively simple examples and they do not show all of the possible combinations. For example, you can run an Apache Ignite database in the cloud if you like; there were just too many combinations to cover them all in this book.

Part V: Advanced Topics

Although this book does not assume you have previous database experience, that doesn't mean it cannot cover some more advanced subjects. The chapters in this part of the book explain some more sophisticated topics that are important but not central to database design.

Chapter 26, "Introduction to SQL," provides an introduction to SQL (Structured Query Language). It explains how to use SQL commands to add, insert, update, and delete data. By using SQL, you can help insulate a program from the idiosyncrasies of the particular database product that it uses to store data.

Chapter 27, "Building Databases with SQL Scripts," explains how to use SQL scripts to build a database. It explains the advantages of this technique, such as the ability to create scripts to initialize a database before performing tests. It also explains some of the restrictions on this method, such as the fact that the user may need to create and delete tables in a specific order to satisfy table relationships.

Chapter 28, "Database Maintenance," explains some of the database maintenance issues that are part of any database application. Though performing and restoring backups, compressing tables, rebuilding indexes, and populating data warehouses are not strictly database design tasks, they are essential to any working application.

Chapter 29, "Database Security," explains database security issues. It explains the kinds of security that some database products provide. It also explains some additional techniques that can enhance database security such as using database views to appropriately restrict the users' access to data.

Appendixes

The book's appendixes provide additional reference material to supplement the earlier chapters.

Appendix A, "Exercise Solutions," gives solutions to the exercises at the end of most of the book's chapters so that you can check your progress as you work through the book.

Appendix B, "Sample Relational Designs," shows some sample designs for a variety of common database situations. These designs store information about such topics as books, movies, documents, customer orders, employee timekeeping, rentals, students, teams, and vehicle fleets.

The Glossary provides definitions for useful database and software development terms. The Glossary includes terms defined and used in this book in addition to a few other useful terms that you may encounter while reading other database material.

HOW TO USE THIS BOOK

Because this book is aimed at readers of all experience levels, you may find some of the material familiar if you have previous experience with databases. In that case, you may want to skim chapters covering material that you already thoroughly understand.

If you are familiar with relational databases, you may want to skim Chapter 1, "Database Design Goals," and Chapter 2, "Relational Overview." Similarly if you have experience with NoSQL databases, you may want to skip Chapter 3, "NoSQL Overview."

If you have previously helped write project proposals, you may understand some of the questions you need to ask users to properly understand their needs. In that case, you may want to skim Chapter 4, "Understanding User Needs."

If you have built databases before, you may understand at least some of the data normalization concepts explained in Chapter 7, "Normalizing Data." This is a complex topic, however, so I recommend that you not skip this chapter unless you really know what you're doing.

If you have extensive experience with SQL, you may want to skim Chapter 26, "Introduction to SQL." (Many developers who have used but not designed databases fall into this category.)

In any case, I strongly recommend that you at least skim the material in every chapter to see if there are any new concepts you can pick up along the way. At least look at the Exercises at the end of each chapter before you decide that you can safely skip to the next. If you don't know how to outline the solutions to the Exercises, then you should consider looking at the chapter more closely.

Different people learn best in different ways. Some learn best by listening to lecturers, others by reading, and others by doing. Everyone learns better by combining learning styles. You will get the most from this book if you read the material and then work through the Exercises. It's easy to think to yourself, "Yeah, that makes sense" and believe you understand the material, but working through some of the Exercises will help solidify the material in your mind. Doing so may also help you see new ways that you can apply the concepts covered in the chapter.

NOTE *Normally, when I read a new technical book, I work through every example, modifying the problems to see what happens if I try different things not covered by the author. I work through as many questions and exercises as I can until I reach the point where more examples don't teach me anything new (or I'm tired of breaking my system and having to reinstall things). Then I move on. It's one thing to read about a concept in the chapter; it's another to try to apply it to data that is meaningful to you.*

After you have learned the ideas in the book, you can use it for a reference. For example, when you start a new project, you may want to refer to Chapter 4, "Understanding User Needs," to refresh your memory about the kinds of questions you should ask users to discover their true needs.

Visit the book's website to look for updates and addendums. If readers find typographical errors or places where a little additional explanation may help, I'll post updates on the website.

Finally, if you get stuck on a really tricky concept and need a little help, email me at `RodStephens@csharphelper.com` and I'll try to help you out.

NOTE TO INSTRUCTORS

Database programming is boring. Maybe not to you and me, who have discovered the ecstatic joy of database design, the thrill of normalization, and the somewhat risqué elation brought by slightly denormalizing a database to achieve optimum performance. But let's face it, to a beginner, database design and development can be a bit dull.

There's little you can do to make the basic concepts more exciting, but you can do practically anything with the data. At some point it's useful to explain how to design a simple inventory system, but that doesn't mean you can't use other examples designed to catch students' attention. Data that relates to the students' personal experiences or that is just plain outrageous keeps them awake and alert (and most of us know that it's easier to teach students who are awake).

The examples in this book are intended to demonstrate the topic at hand but not all of them are strictly business-oriented. I've tried to make them cover a wide variety of topics from serious to silly. To keep your students interested and alert, you should add new examples from your personal experiences and from your students' interests.

I've had great success in my classroom using examples that involve sports teams (particularly local rivalries), music (combining classics such as Bach, Beethoven, and Tone Loc), the students in the class (but be sure not to put anyone on the spot), television shows and stars, comedians, and anything else that interests the students.

For exercises, encourage students to design databases that they will find personally useful. I've had students build databases that track statistics for the players on their favorite football teams, inventory their DVD or CD collections, file and search recipe collections, store data on "Magic: The Gathering" trading cards, track role-playing game characters, record information about classic cars, and schedule athletic tournaments. (The tournament scheduler didn't work out too well—the scheduling algorithms were too tricky.) One student even built a small but complete inventory application for his mother's business that she actually found useful. I think he was as shocked as anyone to discover he'd learned something practical.

When students find an assignment interesting and relevant, they become emotionally invested and will apply the same level of concentration and intensity to building a database that they normally reserve for console gaming, *Star Wars*, and World of Warcraft. They may spend hours crafting a database to track WoW alliances just to fulfill a 5-minute assignment. They may not catch every nuance of domain/key normal form, but they'll probably learn a lot about building a functional database.

NOTE TO STUDENTS

If you're a student and you peeked at the previous section, "Note to Instructors," shame on you! If you didn't peek, do so now.

Building a useful database can be a lot of work, but there's no reason it can't be interesting and useful to you when you're finished. Early in your reading, pick some sort of database that you would find useful (see the previous section for a few ideas) and think about it as you read through the text. When the book talks about creating an initial design, sketch out a design for your database. When the book explains how to normalize a database, normalize yours. As you work through the exercises, think about how they would apply to your dream database.

Don't be afraid to ask your instructor if you can use your database instead of one suggested by the book for a particular assignment (unless you have one of those instructors who hand out extra work to anyone who crosses their path; in that case, keep your head down). Usually an instructor's thought process is quite simple: "I don't care what database you use as long as you learn the material." Your instructor may want your database to contain several related tables so that you can create the complexity needed for a particular exercise, but it's usually not too hard to make a database complicated enough to be interesting.

When you're finished, you will hopefully know a lot more about database design than you do now, and if you're persistent, you might just have a database that's actually good for something. Hopefully you'll also know how to design other useful databases in the future. (And when you're finished, email me at `RodStephens@csharphelper.com` and let me know what you built!)

CONVENTIONS

To help you get the most from the text and keep track of what's happening, we've used a number of conventions throughout the book.

NOTE *Tips, hints, tricks, and asides to the current discussion are offset and placed in italics* like this.

Activities are exercises that you should work through, following the text in the book.

1. They usually consist of a set of steps.
2. Each step has a number.
3. Follow the steps with your copy of the database.

continues

(continued)

How It Works

After most activity instruction sections, the process you've stepped through is explained in detail.

As for styles in the text:

- We *highlight* new terms and important words when we introduce them.
- We show keyboard strokes like this: Ctrl+A.
- We show filenames, URLs, and code within the text like so: `SELECT * FROM Students`.
- We present blocks of code like this:

```
We use a monofont type with no highlighting for code examples.
```

SOURCE CODE

As you work through the examples in this book, you may choose either to type in all the code manually or to use the source code files that accompany the book. All of the source code used in this book is available for download at `www.wiley.com/go/beginningdbdesign2e`.

CONTACTING THE AUTHOR

If you have questions, suggestions, comments, want to swap cookie recipes, or just want to say "Hi," email me at `RodStephens@csharphelper.com`. I can't promise that I'll be able to help you with every problem, but I do promise to try.

DISCLAIMER

Many of the examples in this book were chosen for interest or humorous effect. They are not intended to disparage anyone. I mean no disrespect to police officers (or anyone else who regularly carries a gun), plumbers, politicians, jewelry store owners, street luge racers (or anyone else who wears helmets and Kevlar body armor to work), or college administrators. Or anyone else for that matter.

Well, maybe politicians.

PART 1
Introduction to Databases and Database Design

The chapters in this part of the book provide background that is useful when studying database design.

Chapter 1 explains the reasons why database design is important. It discusses the goals that you should keep in mind while designing databases. If you keep those goals in mind, then you can stay focused on the end result and not become bogged down in the minutiae of technical details. If you understand those goals, then you will also know when it might be useful to bend the rules a bit.

Chapter 2 provides background on relational databases. It explains common relational database terms and concepts that you need to understand for the chapters that follow. You won't get as much out of the rest of the book if you don't understand the terminology.

Chapter 3 describes NoSQL databases. While this book (and most other database books) focuses on relational databases, there are other kinds of databases that are better suited to some tasks. NoSQL databases provide some alternatives that may work better for you under certain circumstances. (I once worked on a 40-developer project that failed largely because it used the wrong kind of database. Don't let that happen to you!)

Even if you're somewhat familiar with databases, give these chapters at least a quick glance to ensure that you don't miss anything important. Pay particular attention to the terms described in Chapter 2, because you'll need to know them later.

1

Database Design Goals

Using modern database tools, just about anyone can build a database. The question is, will the resulting database be useful?

A database won't do you much good if you can't get data out of it quickly, reliably, and consistently. It won't be useful if it's full of incorrect or contradictory data, nor will it be useful if it is stolen, lost, or corrupted by data that was only half written when the system crashed.

You can address all of these potential problems by using modern database tools, a good database design, and a pinch of common sense, but only if you understand what those problems are so you can avoid them.

The first step in the quest for a useful database is understanding database goals. What should a database do? What makes a database useful and what problems can it solve? Working with a powerful database tool without goals is like flying a plane through clouds without a compass—you have the tools you need but no sense of direction.

This chapter describes the goals of database design. By studying information containers, such as files that can play the role of a database, the text defines properties that good databases have and problems that they should avoid.

In this chapter, you will learn about the following:

- ➤ Why a good database design is important
- ➤ The strengths and weaknesses of various kinds of information containers that can act as databases
- ➤ How computerized databases can benefit from those strengths and avoid those weaknesses
- ➤ How good database design helps achieve database goals
- ➤ What CRUD, ACID, and BASE are, and why they are relevant to database design

THE IMPORTANCE OF DESIGN

Forget for a moment that this book is about designing databases and consider software design in general. Software design plays a critical role in software development. The design lays out the general structure and direction that future development will take. It determines which parts of the system will interact with other parts. It decides which subsystems will provide support for other pieces of the application.

If an application's underlying design is flawed, the system as a whole is at risk. Bad assumptions in the design creep into the code at the application's lowest levels, resulting in flawed subsystems. Higher-level systems built on those subsystems inherit those design flaws, and soon their code is corrupted, too.

Sometimes, a sort of decay pervades the entire system and nobody notices until relatively late in the project. The longer the project continues, the more entrenched the incorrect assumptions become, and the more reluctant developers are to scrap the whole design and start over. The longer problems remain in the system, the harder they are to remove. At some point, it might be easier to throw everything away and start over from scratch, a decision that few managers will want to present to upper management.

SPACE SPAT

An engineer friend of mine was working on a really huge satellite project. After a while, the engineers all realized that the project just wasn't feasible given the current state of technology and the design. Eventually, the project manager was forced to admit this to upper management and he was fired. The new project manager stuck it out for a while and then he, too, was forced to confess to upper management that the project was unfeasible. He, too, was fired.

For a while, this process continued—with a new manager taking over, realizing the hopelessness of the design, and being fired. That is, until eventually even upper management had to admit the project wasn't going to work out and the whole thing collapsed.

They could have saved time, money, and several careers if they had spent more time up-front on the design and either fixed the problems or realized right away that the project wasn't going to work and scrapped it at the start.

Building an application is often compared to building a house or skyscraper. You probably wouldn't start building a multibillion-dollar skyscraper without a comprehensive design that is based on well-established architectural principles. Unfortunately, software developers often rush off to start coding as soon as they possibly can because coding is more fun and interesting than design is. Coding also lets developers tell management and customers how many lines of code they have written, so it seems like they are making progress even if the lines of code are corrupted by false assumptions. Only later do they realize that the underlying design is flawed, the code they wrote is worthless, and the project is in serious trouble.

Now, let's get back to database design. Few parts of an application's design are as critical as the database's design. The database is the repository of the information that the rest of the application manages and displays to the users. If the database doesn't store the right data, doesn't keep the data safe, or doesn't let the application find the data it needs, then the application has little chance for success. Here, the *garbage-in, garbage-out* (*GIGO*) principle is in full effect. If the underlying data is unsound, it doesn't matter what the application does with it; the results will be suspect at best.

For example, imagine that you've built an order-tracking system that can quickly fetch information about a customer's past orders. Unfortunately, every time you ask the program to fetch a certain customer's records, it returns a slightly different result. Although the program can find data quickly, the results are not trustworthy enough to be usable.

For another example, imagine that you have built an amazing program that can track the thousands of tasks that make up a single complex job, such as building a cruise liner or passenger jet. It can track each task's state of completion, determine when you need to order new parts for them to be ready for future construction phases, and can even determine the present value of future purchases so you can decide whether it is better to buy parts now or wait until they are needed. Unfortunately, the program takes hours to recalculate the complex task schedule and pricing details. Although the calculations are correct, they are so slow that users cannot reasonably make any changes. Changing the color of the fabric of a plane's seats or the tile used in a cruise liner's hallways could delay the whole project. (I once worked on a project with a similar issue. It worked, but it was so slow that it became a serious problem.)

For a final example, suppose you have built an efficient subscription application that lets customers subscribe to your company's quarterly newsletters, data services, and sarcastic demotivational quote of the day. It lets you quickly find and update any customer's subscriptions, and it always consistently shows the same values for a particular customer. Unfortunately, when you change the price of one of your publications, you find that not all of the customers' records show the updated price. Some customers' subscriptions are at the new rate, some are at the old rate, and some seem to be at a rate you've never seen before. (This example isn't as far-fetched as it may seem. Some systems allow you to offer sale prices or special incentives to groups of customers, or they allow sales reps to offer special prices to particular customers. That kind of system requires careful design if you want to be able to do things like change standard prices without messing up customized pricing.)

Poor database design can lead to these and other annoying and potentially expensive scenarios. A good design creates a solid foundation on which you can build the rest of the application.

Experienced developers know that the longer a bug remains in a system, the harder it is to find and fix. From that it logically follows that it is extremely important to get the design right before you start building on it.

Database design is no exception. A flawed database design can doom a project to failure before it has begun as surely as ill-conceived software architecture, poor implementation, or incompetent programming can.

INFORMATION CONTAINERS

What is a database? This may seem like a trivial question, but if you take it seriously the result can be pretty enlightening. By studying the strengths and weaknesses of some physical objects that meet the definition of a database, you can learn about the features that you might like a computerized database to have.

> ***DEFINITION*** *A database is a tool that stores data and lets you create, read, update, and delete the data in some manner.*

This is a pretty broad definition and includes a lot of physical objects that most people don't think of as modern databases. For example, Figure 1.1 shows a box full of business cards, a notebook, a filing cabinet full of customer records, and your brain, all of which fit this definition. Each of these physical databases has advantages and disadvantages that can give insight into the features that you might like in a computer database.

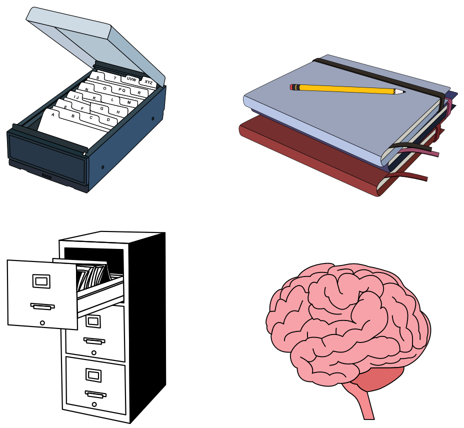

FIGURE 1.1

A box of business cards is useful as long as it doesn't contain too many cards. You can find a particular piece of data (for example, the phone number for your favorite Canadian restaurant) by looking through all the cards. You can easily expand the database by shoving more cards into the box, at least up to a point. If you have more than a dozen or so business cards, finding a particular card can be time consuming. You can even rearrange the cards a bit to improve performance for cards you use often. Each time you use a card, you can move it to the front of the box. Over time, those that are used most often will migrate to the front.

A notebook (the cardboard and paper kind, not the small laptop kind) is small, easy to use, easy to carry, doesn't require electricity, and doesn't need to boot before you can use it. A notebook database is also easily extensible because you can buy another notebook to add to your collection when the first one is full. However, a notebook's contents are arranged sequentially. If you want to find information about a particular topic, you'll have to look through the pages one at a time until you find what you want. The more data you have, the harder this kind of search becomes.

A filing cabinet can store a lot more information than a notebook, and you can easily expand the database by adding more files or cabinets. Finding a particular piece of information in the filing cabinet can be easier than finding it in a notebook, as long as you are searching for the type of data used to arrange the records. If the filing cabinet is full of customer information sorted by customer name, and you want to find a particular customer's data, then you're in luck. If you want to find all of the customers who live in a certain city, you'll have to dig through the files one at a time.

Your brain is the most sophisticated database ever created. It can store an incredible amount of data and allows you to retrieve a particular piece of data in several different ways. For example, right now you could probably easily answer the following questions about the restaurants that you visit frequently:

- Which is closest to your current location?
- Which has the best desserts?
- Which has the best service?
- Which is least expensive?
- Which is the best for a business lunch?
- Which is your overall favorite?
- Why don't we go there tonight?

Your brain provides many different ways you can access the same restaurant information. You can search based on a variety of keys (such as location, quality of dessert, expense, and so forth). To answer these questions with a box of business cards, a notebook, or a filing cabinet would require a long and grueling search.

Still your brain has some drawbacks, at least as a database. Most notably it forgets. You may be able to remember an incredible number of things, but some become less reliable or disappear completely over time. Do you remember the names of all of your elementary school teachers? I don't. (I don't remember my own teachers' names, much less yours!)

Your brain also gets tired, and when it is tired it is less accurate.

Although your brain is good at certain tasks, such as recognizing faces or picking restaurants, it is not so good at other tasks like providing an accurate list of every item a particular customer purchased in the past year. Those items have less emotional significance than, for example, your spouse's name, so they're harder to remember.

All of these information containers (business cards, notebooks, filing cabinets, and your brain) can become contaminated with misleading, incorrect, and contradictory information. If you write different versions of the same information in a notebook, the data won't be consistent. Later when you try to look up the data, you may find either version first and may not even remember that there's another version. (Your brain can become especially cluttered with inconsistent and contradictory information, particularly if you listen to politicians during an election year.)

The following section summarizes some of the strengths and weaknesses of these information containers.

STRENGTHS AND WEAKNESSES OF INFORMATION CONTAINERS

By understanding the strengths and weaknesses of information containers like those described in the previous section, you can learn about features that would be useful in a computerized database. So, what are some of those strengths and weaknesses?

The following list summarizes the advantages of some information containers:

- None of these databases require electricity so they are safe from power failures (although your brain requires food; as the dormouse said, feed your head).
- These databases keep data fairly safe and permanent (barring fires and memory loss). The data doesn't just disappear.
- These databases (excluding your brain) are inexpensive and easy to buy.
- These databases have simple user interfaces so that almost anyone can use them.
- Using these databases, it's fairly easy to add, edit, and remove data.
- The filing cabinet lets you quickly locate data if you search for it in the same way it is arranged (for example, by customer name).
- Your brain lets you find data by using different keys (for example, by location, cost, or quality of service).
- All of these databases allow you to find every piece of information that they contain, although it may take a while to dig through it all.
- All of these databases (except possibly your brain) provide consistent results as long as the facts they store are consistent. For example, two people using the same notebook will find the same data. Similarly if you look at the same notebook at a later time, it will show the same data you saw before (if it hasn't been modified).
- All of these databases except the filing cabinet are portable.
- Your brain can perform complex calculations, at least of a limited type and number.
- All of these databases provide atomic transactions.

The final advantage is a bit more abstract than the others so it deserves some additional explanation. An *atomic transaction* is a possibly complex series of actions that is considered to be a single operation by those who are not involved directly in performing the transaction.

The classic example is transferring money from one bank account to another. Suppose Alice writes Bob a check for $100 and you need to transfer the money between their accounts. You pick up the account book, subtract $100 from Alice's record, add $100 to Bob's record, and then put the notebook down. Someone else who uses the notebook might see it before the transaction (when Alice has the $100) or after the transaction (when Bob has the $100), but they won't see it during the transaction where the $100 has been subtracted from Alice but not yet given to Bob. The office bullies aren't allowed to grab the notebook from your hands when you're halfway through and play keep-away. It's an all-or-nothing transaction.

In addition to their advantages, information containers like notebooks and filing cabinets have some disadvantages. It's worth studying these disadvantages so that you can try to avoid them when you build computerized databases.

The following list summarizes some of the disadvantages that these information containers have:

- All of these databases can hold incomplete, incorrect, or contradictory data.
- Some of them are easy to lose or steal. Someone could grab your notebook while you're eating lunch or read over your shoulder on the bus. You could even forget your notebook at the security counter as you dash to catch your flight.
- In all of these databases, correcting large errors in the data can be difficult. For example, it's easy to use a pen to change one person's address in an address notebook. It's much harder to update hundreds of addresses if a new city is created in your area. (This recently happened near where I live.) Such a circumstance might require a tedious search through a set of business cards, a notebook, or a filing cabinet. It may be years before your brain makes the switch completely.
- These databases are relatively slow at creating, retrieving, updating, and deleting data. Your brain is much faster than the others at some tasks but is not good at manipulating a lot of information all at once. For example, how quickly can you list your 20 closest friends in alphabetical order? Even picking your closest friends can be difficult at times. (And don't think you can cheat by using Facebook because you probably have hundreds of friends there and we only want your top 20.)
- Your brain can give different results at different times depending on uncontrollable factors such as your mood, how tired you are, and even whether you're hungry.
- Each of these databases is located in a single place so it cannot be easily shared. Each also cannot be easily backed up, so if the original is lost or destroyed, you lose your data.

The following section considers how you can translate these strengths and weaknesses into features to prefer or avoid in a computerized database.

DESIRABLE DATABASE FEATURES

By looking at the advantages and disadvantages of physical databases, you can create a list of features that a computerized database should have. Some are fundamental characteristics that any database must have. ("You should be able to get data from it." How obvious is that?)

Most of these features, however, depend at least in part on good database design. If you don't craft a good design, you'll miss out on some or all of the benefits of these features. For example, any decent