THE EXPERT'S VOICE® IN SQL

Pro SQL Server Relational Database Design and Implementation

Database design delivering business results

—

Fifth Edition

—

Louis Davidson
with Jessica Moss
Foreword by Kalen Delaney

Apress®

Pro SQL Server Relational Database Design and Implementation

Fifth Edition

Louis Davidson

with Jessica Moss

Foreword by Kalen Delaney

Apress®

Pro SQL Server Relational Database Design and Implementation

Louis Davidson
Antioch, Tennessee
USA

Jessica Moss
Mechanicsville, Virginia
USA

ISBN-13 (pbk): 978-1-4842-1972-0
ISBN-13 (electronic): 978-1-4842-1973-7
DOI 10.1007/978-1-4842-1973-7

Library of Congress Control Number: 2016963113

Managing Director: Welmoed Spahr
Lead Editor: Jonathan Gennick
Development Editor: Laura Berendson
Technical Reviewers: Rodney Landrum, Arun Pande, Andy Yun, Alexzander Nepomnjashiy
Editorial Board: Steve Anglin, Pramila Balan, Laura Berendson, Aaron Black, Louise Corrigan, Jonathan Gennick, Todd Green, Robert Hutchinson, Celestin Suresh John, Nikhil Karkal, James Markham, Susan McDermott, Matthew Moodie, Natalie Pao, Gwenan Spearing
Coordinating Editor: Jill Balzano
Copy Editor: Bill McManus
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

This book is dedicated to my mom, who passed away just after Thanksgiving in 2013. She had no idea what SQL was really, but was always encouraging and very proud of me as I was of her.

—Louis

Contents at a Glance

Contents

Foreword

Database design is a hugely important topic, but one that is frequently overlooked. Because good database design is not dependent on a specific database product, vendors don't usually support education in design. As a long-time certified trainer, I was always well aware that issues pertaining to design were not covered in the standard courseware, because they are so vendor independent. For many years, and many versions, Microsoft had a course called "Implementing a Database Design", and people frequently shortened the name to "The DB Design Course", but it was about implementation, not design. It was about how to create the database after the design was done.

Proper design is not only important for data correctness, but it can also help you to troubleshoot effectively and isolate performance issues. Sometimes, little design changes can make a huge performance difference, but to understand what changes those might be, you need to understand not only design issues in general, but also the details of your system's actual design. And of course, the more you know, the better. But the ideal way to get the best performance from your database design is to start with a good design from the very beginning of a project.

So if no database vendor offers training courses in relational database design, how is one supposed to learn about this topic? Obviously, from books like the one you have here. Louis Davidson shows you not just how to implement a database design but how to first design your database from the stated requirements. In the first few chapters, Louis gives you a solid foundation in relational database concepts and data modeling, the difference between the logical and physical model, and the details of normalization. He goes on in the second part to show you how to take that design and create a database from it. Along the way he gives you the basics of several other database related concepts that can make an enormous difference in the performance (and thus the ultimate success) of your new database application: indexes and concurrency.

Modern relational database products have been built to allow you to create a database in minimal time and start loading data and writing queries very quickly. But if you want those queries to perform well and continue to give good performance as your database grows to gigabytes and perhaps even terabytes, you need a solid foundation of a good design. Time spent on database design is an investment with a huge return over the life of an application. You deserve to start out well, and Louis can help make that possible.

Kalen Delaney

www.SQLServerInternals.com

Poulsbo, WA – November 2016

About the Authors

Louis Davidson has been working with databases (for what is starting to seem like a really long time) as a corporate database developer and architect. He has been a Microsoft MVP for 13 years and this is the fifth edition of this database design book with Apress. Louis has been active speaking about database design and implementation at many conferences over the past 14 years, including SQL PASS, SQL Rally, SQL Saturday events, CA World, and the devLink Technical Conference. Louis has worked for the Christian Broadcasting Network (CBN) as a developer, DBA, and data architect, supporting offices in Virginia Beach, Virginia, and in Nashville, Tennessee, for over 18 years. Louis has a bachelor's degree from the University of Tennessee at Chattanooga in computer science.

For more information please visit his web site at www.drsql.org.

Jessica Moss is a well-known practitioner, author, and speaker on Microsoft SQL Server business intelligence. She has created numerous data warehouse and business intelligence solutions for companies in different industries and has delivered training courses on Integration Services, Reporting Services, and Analysis Services. While working for a major clothing retailer, Jessica participated in the SQL Server 2005 TAP program, where she developed best implementation practices for Integration Services. Jessica has authored technical content for multiple magazines, web sites, and books, and has spoken internationally at conferences such as the PASS Community Summit, SharePoint Connections, and the SQLTeach International Conference. As a strong proponent of developing user-to-user community relations, Jessica actively participates in local user groups and code camps in central Virginia. In addition, Jessica volunteers her time to help educate people through the PASS organization.

About the Technical Reviewers

Rodney Landrum has been architecting solutions for SQL Server for over 12 years. He has worked with and written about many SQL Server technologies, including DTS, integration services, analysis services, and reporting services. He has authored three books on reporting services. He is a regular contributor to *SQL Server* Magazine, SQLServerCentral.com, and Simple-Talk.com. Rodney is also an SQL Server MVP.

Alexzander Nepomnjashiy is CIO at Kivach Clinic, www.kivach.ru.

Andy Yun has been a database developer and administrator for SQL Server for almost 15 years. Leveraging knowledge of SQL Server Internals and extensive experience in highly transactional environments, he strives to make T-SQL leaner and meaner. Andy is extremely passionate about passing knowledge on to others, regularly speaking at User Groups, SQL Saturdays, and PASS Summit. Andy is a co-founder of the Chicago SQL Association, co-Chapter Leader of the Chicago Suburban SQL Server User Group, and part of the Chicago SQL Saturday Organizing Committee.

Acknowledgments

You are never too old to set another goal or to dream a new dream.

—C. S. Lewis

I am not a genius, nor am I some form of pioneer in the database design world. I am just a person who 15 or so years ago asked the question of a publisher: "Do you have any books on database design?" The reply: "No, why don't you write one?" So I did, and I haven't stopped writing since then, with this book now in its fifth edition. I acknowledge that the following "people" have been extremely helpful in making this book happen and evolve along the way. Some have helped me directly, while others probably don't even know that this book exists. Either way, they have all been an important part of the process.

> Far above anyone else, Jesus Christ, without whom I wouldn't have had the strength to complete the task of writing this book. I know I am not ever worthy of the Love that You give me.
>
> My wife, Valerie Davidson, for putting up with this craziness for yet another time, all while working on her doctorate in education.
>
> Gary Cornell, for a long time ago giving me a chance to write the book that I wanted to write.
>
> My current managers, Mark Carpenter, Andy Curley, and Keith Griffith, for giving me time to go to several conferences that really helped me to produce as good of a book as I did. And to all of my coworkers at CBN, who have provided me with many examples for this book and my other writing projects.
>
> The PASS conferences (particularly SQL Saturday events), where I was able to hone my material and meet thousands of people over the past three years and find out what they wanted to know.
>
> Jessica Moss, for teaching me a lot about data warehousing, and taking the time to write the last chapter of this book for you.
>
> Paul Nielsen, for challenging me to progress and think harder about the relational model and its strengths and weaknesses.
>
> The MVP Program, and perhaps even more importantly, the MVPs and Microsoft folks I have come into contact with over the years. I have learned so much in the newsgroups, e-mail lists, and in particular, the MVP Summit that I could not have done half as well without them (check the first edition of this book for evidence that things got better after I became an MVP!).

The fantastic editing staff I've had, including Jonathan Gennick and Jill Balzano, who (figuratively) busted my lip a few times over my poor structure, use of the English language, etc., and without whom the writing would sometimes appear to come from an illiterate chimpanzee. Most of these people are included on the copyright page, but I want to say a specific thanks to them and to Tony Davis (who had a big hand in the 2005 version of the book) for making this book great, despite my frequently rambling writing style.

To the academics out there who have permeated my mind with database theory, such as E. F. Codd, Chris Date, Fabian Pascal, Joe Celko, my professors at the University of Tennessee at Chattanooga, and many others. I wouldn't know half as much without you. And thanks to Mr. Date for reviewing Chapter 1 in the previous edition; you probably did more for the next version of this book than the current one.

All of the people I have acknowledged in previous editions who were so instrumental in getting this book to where it is from all of the many changes and lessons over the years. I've built upon the help you've all provided over the past 12+ years.

—Louis Davidson

Introduction

It was a dark and stormy night…

—Snoopy

These words start many a work of fiction, and usually not the most believable works of fiction either. While this book is clearly, at its core, nonfiction, I felt the need to warn you that nearly every example in this book is fiction, carefully tailored to demonstrate some principle of database design. Why fictitious examples? My good friend, Jeremiah Peschka, once explained it perfectly when he tweeted: "@peschkaj: I'm going to demo code on a properly configured server with best practices code. Then you're all going to [complain] that it takes too long." The most egregious work of fiction will be the chapter on requirements, as in most cases the document to describe where to find the actual requirements documents will be longer than the chapter where I describe the process and include several examples.

So don't expect that if you can understand all of the examples in the book that you can easily be an expert in a week. The fact is, the real-world problems you will encounter will be far more complex, and you will need lots of practice to get things close to right. The thing you will get out of my book is knowledge of how to get there, and ideals to follow. If you are lucky, you will have a mentor or two who already know a few things about database design to assist you with your first designs. I know that when I was first getting started, I learned from a few great mentors, and even today I do my best to bounce ideas off of others before I create my first table. (Note that you don't need an expert to help you validate designs. A bad design, like spoiled milk, smells fonky, which is 3.53453 times worse than funky.)

However, what is definitely not fiction is my reason for writing (and rewriting) this book: that great design is still necessary. There is a principle among many programmers that as technology like CPU and disk improves, code needn't be written as well to get the job done fast enough. While there is a modicum of truth to that principle, consider just how wasteful this is. If it takes 100 milliseconds to do something poorly, but just 30 milliseconds to do it right, which is better? If you have to do the operation once, then either is just as good. But we don't generally write software to do something once. Each execution of that poorly written task is wasting 70 milliseconds of resources to get the job done. Now consider how many databases and how much code exist out there and guess what the impact to your slice of the world, and to the entire world, would be. "How good is good enough?" is a question one must ask, but if you aim for sleeping on the sidewalk, you are pretty much guaranteed not to end up with a mansion in Beverly Hills (swimming pools and movie stars!).

I cannot promise you the deepest coverage of the theory that goes into the database design process, nor do I want to. If you want to go to the next level, the latest edition of Chris Date's *An Introduction to Database Systems* (Addison Wesley) is essential reading, and you'll find hundreds of other database design books listed if you search for "database design" on a book seller's web site. The problem is that a lot of these books have far more theory than the average practitioner wants (or will take the time to read), and they don't really get into the actual implementation on an actual database system. Other books that are implementation oriented don't give you enough theory and focus solely on the code and tuning aspects that one needs **after** the database is a mess. So many years ago, I set out to write the book you have in your hands, and this is the fifth edition under the Apress banner (with one earlier edition through a publisher to remain nameless). The technology has changed greatly, with the versions of SQL Server from 2012 and beyond ratcheting up the complexity tremendously.

This book's goal is simply to be a technique-oriented book that starts out with "why" to design like the founders suggested, and then addresses "how" to make that happen using the features of SQL Server. I will cover many of the most typical features of the relational engine, giving you techniques to work with. I can't, however, promise that this will be the only book you need on your shelf on the subject of database design, and particularly on SQL Server.

Oscar Wilde, the poet and playwright, once said, "I am not young enough to know everything." It is with some chagrin that I must look back at the past and realize that I thought I knew everything just before I wrote my first book, *Professional SQL Server 2000 Database Design* (Wrox Press, 2001). It was ignorant, unbridled, unbounded enthusiasm that gave me the guts to write the first book. In the end, I did write that first edition, and it was a decent enough book, largely due to the beating I took from my technical editing staff. And if I hadn't possessed such enthusiasm initially, I would not likely be writing this edition today. However, if you had a few weeks to burn and you went back and compared each edition of this book, chapter by chapter, section by section, to the current edition, you would notice a progression of material and a definite maturing of the writer.

There are a few reasons for this progression and maturity. One reason is the editorial staff I have had over the past three versions: first Tony Davis and now Jonathan Gennick for the third time. Both of them were very tough on my writing style and did wonders on the structure of the book (which is why this edition has no major structural changes). Another reason is simply experience, as over 15 years have passed since I started the first edition. But most of the reason that the material has progressed is that it's been put to the test. While I have had my share of nice comments, I have gotten plenty of feedback on how to improve things (some of those were not-nice comments!). And I listened very intently, keeping a set of notes that start on the release date. I am always happy to get any feedback that I can use (particularly if it doesn't involve any anatomical terms for where the book might fit). I will continue to keep my e-mail address available (`louis@drsql.org`), and you can leave anonymous feedback on my web site if you want (`www.drsql.org`). You may also find an addendum there that covers any material that I didn't have space for or that I may uncover that I wish I had known at the time of this writing.

Purpose of Database Design

What is the purpose of database design? Why the heck should you care? The main reason is that a properly designed database is straightforward to work with, because everything is in its logical place, much like a well-organized cupboard. When you need paprika, it's easier to go to the paprika slot in the spice rack than it is to have to look for it everywhere until you find it, but many systems are organized just this way. Even if every item has an assigned place, of what value is that item if it's too hard to find? Imagine if a phone book wasn't sorted at all. What if the dictionary was organized by placing a word where it would fit in the text? With proper organization, it will be almost instinctive where to go to get the data you need, even if you have to write a join or two. I mean, isn't that fun after all?

You might also be surprised to find out that database design is quite a straightforward task and not as difficult as it may sound. Doing it right is going to take more up-front time at the beginning of a project than just slapping a database as you go along, but it pays off throughout the full life cycle of a project. Of course, because there's nothing visual to excite the client, database design is one of the phases of a project that often gets squeezed to make things seem to go faster. Even the least challenging or uninteresting user interface is still miles more interesting to the average customer than the most beautiful data model. Programming the user interface takes center stage, even though the data is generally why a system gets funded and finally created. It's not that your colleagues won't notice the difference between a cruddy data model and one that's a thing of beauty. They certainly will, but the amount of time required to decide the right way to store data correctly can be overlooked when programmers need to code. I wish I had an answer for that problem, because I could sell a million books with just that. This book will assist you with some techniques and processes that will help you through the process of designing databases, in a way that's clear enough for novices and helpful to even the most seasoned professional.

This process of designing and architecting the storage of data belongs to a different role than those of database setup and administration. For example, in the role of data architect, I seldom create users, perform backups, or set up replication or clustering. Little is mentioned of these tasks, which are considered administration and the role of the DBA. It isn't uncommon to wear both a developer hat and a DBA hat (in fact, when you work in a smaller organization, you may find that you wear so many hats your neck tends to hurt), but your designs will generally be far better thought out if you can divorce your mind from the more implementation-bound roles that make you wonder how hard it will be to use the data. For the most part, database design looks harder than it is.

Who This Book Is For

This book is written for professional programmers who have the need to design a relational database using any of the Microsoft SQL Server family of technology. It is intended to be useful for the beginner to advanced programmer, either strictly database programmers or a programmer that has never used a relational database product before to learn why relational databases are designed in the way they are, and get some practical examples and advice for creating databases. Topics covered cater to the uninitiated to the experienced architect to learn techniques for concurrency, data protection, performance tuning, dimensional design, and more.

How This Book Is Structured

This book is composed of the following chapters, with the first five chapters being an introduction to the fundamental topics and processes that one needs to go through/know before designing a database. Chapter 6 is an exercise in learning how a database is put together using scripts, and the rest of the book is takes topics of design and implementation and provides instruction and lots of examples to help you get started building databases.

Chapter 1: The Fundamentals. This chapter provides a basic overview of essential terms and concepts necessary to get started with the process of designing a great relational database.

Chapter 2: Introduction to Requirements. This chapter provides an introduction to how to gather and interpret requirements from a client. Even if it isn't your job to do this task directly from a client, you will need to extract some manner or requirements for the database you will be building from the documentation that an analyst will provide to you.

Chapter 3: The Language of Data Modeling. This chapter serves as the introduction to the main tool of the data architect—the model. In this chapter, I introduce one modeling language (IDEF1X) in detail, as it's the modeling language that's used throughout this book to present database designs. I also introduce a few other common modeling languages for those of you who need to use these types of models for preference or corporate requirements.

Chapter 4: Conceptual and Logical Data Model Production. In the early part of creating a data model, the goal is to discuss the process of taking a customer's set of requirements and to put the tables, columns, relationships, and business rules into a data model format where possible. Implementability is less of a goal than is to faithfully represent the desires of the eventual users.

Chapter 5: Normalization. The goal of normalization is to make your usage of the data structures that get designed in a manner that maps to the relational model that the SQL Server engine was created for. To do this, we will take the set of tables, columns, relationships, and business rules and format them in such a way that every value is stored in one place and every table represents a single entity. Normalization can feel unnatural the first few times you do it, because instead of worrying about how you'll use the data, you must think of the data and how the structure will affect that data's quality. However, once you've mastered normalization, not to store data in a normalized manner will feel wrong.

Chapter 6: Physical Model Implementation Case Study. In this chapter, we will walk through the entire process of taking a normalized model and translating it into a working database. This is the first point in the database design process in which we fire up SQL Server and start building scripts to build database objects. In this chapter, I cover building tables—including choosing the datatype for columns—as well as relationships.

Chapter 7: Expanding Data Protection with Check Constraints and Triggers. Beyond the way data is arranged in tables and columns, other business rules need to be enforced. The front line of defense for enforcing data integrity conditions in SQL Server is formed by `CHECK` constraints and triggers, as users cannot innocently avoid them.

Chapter 8: Patterns and Anti-Patterns. Beyond the basic set of techniques for table design, there are several techniques that I use to apply a common data/query interface for my future convenience in queries and usage. This chapter will cover several of the common useful patterns as well as take a look at some patterns that some people will use to make things easier to implement the interface that can be very bad for your query needs.

Chapter 9: Database Security and Security Patterns. Security is high in most every programmer's mind these days, or it should be. In this chapter, I cover the basics of SQL Server security and show how to employ strategies to use to implement data security in your system, such as employing views, triggers, encryption, and using other tools that are a part of the SQL Server toolset.

Chapter 10: Index Structures and Application. In this chapter, I show the basics of how data is structured in SQL Server, as well as some strategies for indexing data for better performance.

Chapter 11: Matters of Concurrency. As part of the code that's written, some consideration needs to be given to sharing resources, if applicable. In this chapter, I describe several strategies for how to implement concurrency in your data access and modification code.

Chapter 12: Reusable Standard Database Components. In this chapter, I discuss the different types of reusable objects that can be extremely useful to add to many (if not all) of the databases you implement to provide a standard problem-solving interface for all of your systems while minimizing inter-database dependencies.

Chapter 13: Architecting Your System. This chapter covers the concepts and concerns of choosing the storage engine and writing code that accesses SQL Server. I cover on-disk or in-memory, ad hoc SQL versus stored procedures (including all the perils and challenges of both, such as plan parameterization, performance, effort, optional parameters, SQL injection, and so on), and whether T-SQL or CLR objects are best.

Chapter 14: Reporting Design. Written by Jessica Moss, this chapter presents an overview of how designing for reporting needs differs from OLTP/relational design, including an introduction to dimensional modeling used for data warehouse design.

Appendix A: Scalar Datatype Reference. In this appendix, I present all of the types that can be legitimately considered scalar types, along with why to use them, their implementation information, and other details.

Appendix B: DML Trigger Basics and Templates. Throughout the book, triggers are used in several examples, all based on a set of templates that I provide in this downloadable appendix, including example tests of how they work and tips and pointers for writing effective triggers. (Appendix B is available as a download along with the code from `www.apress.com` or my web site.)

Prerequisites

The book assumes that the reader has some experience with SQL Server, particularly writing queries using existing databases. Beyond that, most concepts that are covered will be explained and code should be accessible to anyone with experience programming using any language.

Downloading the Code

A download will be available as individual files from the Apress download site. Files will also be available from my web site, `http://www.drsql.org/Pages/ProSQLServerDatabaseDesign.aspx`, as well as links to additional material I may make available between now and any future editions of the book.

Contacting the Author

Don't hesitate to give me feedback on the book, anytime, at my web site (`www.drsql.org`) or my e-mail (`louis@drsql.org`). I'll try to improve any sections that people find lacking in one of my blogs, or as articles, with links from my web site, currently at (`www.drsql.org/Pages/ProSQLServerDatabaseDesign.aspx`), but if that direct link changes, this book will feature prominently on my web site one way or another. I'll be putting more information there, as it becomes available, pertaining to new ideas, goof-ups I find, or additional materials that I choose to publish because I think of them once this book is no longer a jumble of bits and bytes and is an actual instance of ink on paper.

CHAPTER 1

The Fundamentals

Success is neither magical nor mysterious. Success is the natural consequence of consistently applying the basic fundamentals.

—Jim Rohn, American entrepreneur and motivational speaker

I have a love–hate relationship with fundamentals. The easier the task seems to be, the less enjoyable I seem to find it, at least unless I already have a love for the topic at some level. In elementary school, there were fun classes, like recess and lunch for example. But when handwriting class came around, very few kids really liked it, and most of those who did just loved the taste of the pencil lead. But handwriting class was an important part of childhood educational development. Without it, you wouldn't be able to write on a white board, and without that skill, could you actually stay employed as a programmer? I know I personally am addicted to the smell of whiteboard marker, which might explain more than my vocation.

Much like handwriting was an essential skill for life, database design has its own set of skills that you need to get under your belt. While database design is not a hard skill to learn, it is not exactly a completely obvious one either. In many ways, the fact that it isn't a hard skill makes it difficult to master. Databases are being designed all of the time by people of limited understanding of what makes one "good." Administrative assistants build databases using Excel, kids make inventories of their video games on a sheet of paper, and newbie programmers build databases with all sorts of database management tools, and rarely are any of the designs and implementations 100% wrong. The problem is that in almost every case the design produced is fundamentally flawed, causing future modifications to be very painful. When you are finished with this book, you should be able to design databases that reduce the effects of many of the common fundamental blunders. If a journey of a million miles starts with a single step, the first step in the process of designing quality databases is understanding why databases are designed the way they are, and this requires us to cover the fundamentals.

I know this topic may bore you, but would you drive on a bridge designed by an engineer who did not understand physics? Or would you get on a plane designed by someone who didn't understand the fundamentals of flight? Sounds quite absurd, right? So, would you want to store your important data in a database designed by someone who didn't understand the basics of database design?

The first five chapters of this book are devoted to the fundamental tasks of relational database design and preparing your mind for the task at hand: implementing a relational database. The topics won't be particularly difficult in nature, and I will do my best to keep the discussion at the layman's level, and not delve so deeply that you punch me if you pass me in the hall at the PASS Summit (`www.sqlpass.org`).

Electronic supplementary material The online version of this chapter (doi:10.1007/978-1-4842-1973-7_1) contains supplementary material, which is available to authorized users.

L. Davidson and J. Moss, *Pro SQL Server Relational Database Design and Implementation*,
DOI 10.1007/978-1-4842-1973-7_1

For this chapter, we will start out looking at basic background topics that are very useful.

- *History*: Where did all of this relational database stuff come from? In this section I will present some history, largely based on Codd's 12 Rules as an explanation for why the RDBMS (Relational Database Management System) is what it is.
- *Relational data structures*: This section will provide introductions of some of the fundamental database objects, including the database itself, as well as tables, columns, and keys. These objects are likely familiar to you, but there are some common misunderstandings in their usage that can make the difference between a mediocre design and a high-class, professional one.
- *Relationships between entities*: We will briefly survey the different types of relationships that can exist between the relational data structures introduced in the relational data structures section.
- *Dependencies*: The concept of dependencies between values and how they shape the process of designing databases later in the book will be discussed.
- *Relational programming*: This section will cover the differences between procedural programming using C# or VB (Visual Basic) and relational programming using SQL (Structured Query Language).
- *Database design phases*: This section provides an overview of the major phases of relational database design: conceptual/logical, physical, and storage. For time and budgetary reasons, you might be tempted to skip the first database design phase and move straight to the physical implementation phase. However, skipping any or all of these phases can lead to an incomplete or incorrect design, as well as one that does not support high-performance querying and reporting.

At a minimum, this chapter on fundamentals should get us to a place where we have a set of common terms and concepts to use throughout this book when discussing and describing relational databases. Throughout my years of reading and research, I've noticed that lack of agreed-upon terminology is one of the biggest issues in the database community. Academics have one (well, ten) set(s) of terms to mean the same thing as we people who actually develop code. Sometimes multiple different words are used to mean one concept, but worst case, one word means multiple things. Tradespeople (like myself, and probably you the reader) have their own terminology, and it is usually used very sloppily. I am not immune to sloppy terminology myself when chatting about databases, but in this book I do my best to stick to a single set of terms. Some might say that this is all semantics, and semantics aren't worth arguing about, but honestly, they are the *only* thing worth arguing about. Agreeing to disagree is fine if two parties understand one another, but the true problems in life tend to arise when people are in complete agreement about an idea but disagree on the terms used to describe it.

Taking a Brief Jaunt Through History

No matter what country you hail from, there is, no doubt, a point in history when your nation began. In the United States, that beginning came with the Declaration of Independence, followed by the Constitution of the United States (and the ten amendments known as the Bill of Rights). These documents are deeply ingrained in the experience of any good citizen of the United States. Similarly, we have three documents that are largely considered the start of relational databases.

In 1979, Edgar F. Codd, who worked for the IBM Research Laboratory at the time, wrote a paper entitled "A Relational Model of Data for Large Shared Data Banks," which was printed in *Communications of the ACM* ("ACM" is the Association for Computing Machinery [www.acm.org]). In this 11-page paper, Codd introduces a revolutionary idea for how to break the physical barriers of the types of databases in use at that time.

Then, most database systems were very structure oriented, requiring a lot of knowledge of how the data was organized in the storage. For example, to use indexes in the database, specific choices would be made, like only indexing one key, or if multiple indexes existed, the user was required to know the name of the index to use it in a query.

As most any programmer knows, one of the fundamental tenets of good programming is to attempt low coupling of computer subsystems, and needing to know about the internal structure of the data storage was obviously counterproductive. If you wanted to change or drop an index, the software and queries that used the database would also need to be changed. The first half of Codd's relational model paper introduced a set of constructs that would be the basis of what we know as a relational database. Concepts such as tables, columns, keys (primary and candidate), indexes, and even an early form of normalization are included. The second half of the paper introduced set-based logic, including joins. This paper was pretty much the database declaration of storage independence.

Moving six years in the future, after companies began to implement supposed relational database systems, Codd wrote a two-part article published by *Computerworld* magazine entitled "Is Your DBMS Really Relational?" and "Does Your DBMS Run By the Rules?" on October 14 and October 21, 1985. Though it is nearly impossible to get a copy of these original articles, many web sites outline these rules, and I will too. These rules go beyond relational theory and define specific criteria that need to be met in an RDBMS, if it's to be truly considered relational.

Introducing Codd's Rules for an RDBMS

I feel it is useful to start with Codd's rules, because while these rules are over 30 years old, they do probably the best job of setting up not only the criteria that can be used to measure how relational a database is but also the reasons why relational databases are implemented as they are. The neat thing about these rules is that they are seemingly just a formalized statement of the KISS manifesto for database users—keep it simple stupid, or keep it standard, either one. By establishing a formal set of rules and principles for database vendors, users could access data that not only was simplified from earlier data platforms but worked pretty much the same on any product that claimed to be relational. Of course, things are definitely not perfect in the world, and these are not the final principles to attempt to get everyone on the same page. Every database vendor has a different version of a relational engine, and while the basics are the same, there are wild variations in how they are structured and used. The basics are the same, and for the most part the SQL language implementations are very similar (I will discuss very briefly the standards for SQL in the next section). The primary reason that these rules are so important for the person just getting started with design is that they elucidate why SQL Server and other relational engine–based database systems work the way they do.

Rule 1: The Information Principle

> *All information in the relational database is represented in exactly one and only one way—by values in tables.*

While this rule might seem obvious after just a little bit of experience with relational databases, it really isn't. Designers of database systems could have used global variables to hold data or file locations or come up with any sort of data structure that they wanted. Codd's first rule set the goal that users didn't have to think about where to go to get data. One data structure—the table—followed a common pattern of rows and columns of data that users worked with.

Many different data structures were in use in the early days that required a lot of internal knowledge of data. Think about all of the different data structures and tools you have used. Data could be stored in files, a hierarchy (like the file system), or any method that someone dreamed of. Even worse, think of all of the computer programs you have used; how many of them followed a common enough standard that they worked just like everyone else's? Very few, and new innovations are coming every day.