THE EXPERT'S VOICE® IN SQL SERVER

Pro SQL Server 2012 Relational Database Design and Implementation

COMBINE RELATIONAL DATABASE BEST PRACTICES WITH THE LATEST IN SQL SERVER IMPLEMENTATION FEATURES

Louis Davidson with Jessica M. Moss

Apress®

Pro SQL Server 2012 Relational Database Design and Implementation

Louis Davidson
with Jessica M. Moss

Apress®

Pro SQL Server 2012 Relational Database Design and Implementation

ISBN 978-1-4302-3695-5

ISBN 978-1-4302-3696-2 (eBook)

President and Publisher: Paul Manning
Lead Editor: Richard Carey
Technical Reviewer: Rodney Landrum
Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Morgan Ertel, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Tom Welsh
Coordinating Editors: Jessica Belanger and Stephen Moles
Copy Editor: Heather Lang
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media, LLC., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

This book is dedicated my mom.

–Louis

Contents at a Glance

Contents

Foreword

When Louis asked me to write the foreword to this book, I thought he was joking. Why would anyone want a developer to write the foreword for a database book? A quick reminder from Louis made it all clear—I mention the predecessor to this book in my consulting engagements. Who better to demonstrate how effectively he communicates the concepts than someone who lives and breathes databases everyday? Well, I am here to tell you that if you are looking for a sound technical resource for working with SQL Server, look no further.

Once again, Louis has done a remarkable job turning the critical details of SQL Server into an easy-to-read style that is more of a conversation than a technical manual. Louis relates what you really need to know based on his considerable experience and insight. I'm proud to have the opportunity to get to know Louis through his community endeavors and to leverage his knowledge to save time in delivering some of my solutions.

—John Kellar
Chairman, Devlink Technical Conference, and Microsoft MVP

About the Author

Louis has been in the IT industry (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 eight years and this is the fifth edition of this database design book. Louis has been active speaking about database design and implementation at many conferences over the past ten years, including SQL PASS, SQL Rally, SQL Saturday events, CA World, and the Devlink developer conference. Louis has worked for the Christian Broadcasting Network as a developer, DBA, and data architect, supporting offices in Virginia Beach, Virginia, and in Nashville, Tennessee, for over 14 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 drsql.org.

Jessica M. Moss is a well-known practitioner, author, and speaker of 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, websites, 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 Reviewer

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.

Acknowledgments

"I awoke this morning with devout thanksgiving for my friends, the old and the new."

—Ralph Waldo Emerson

I am not a genius, nor am I some form of pioneer in the database design world. I acknowledge that the following "people" have been extremely helpful in making this book happen along the way. Some help 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 a fifth time.

Gary Cornell, for 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. All of my coworkers at CBN that provide 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, for giving me access to learn more about the internals of SQL Server over the years.

The fantastic editing staff I've had, including Jonathan Gennick who (figuratively) busted my lip a few times over my poor use of the English language 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 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; 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 that were so instrumental in getting this book where it is from all of the many changes and lessons over the years. I built upon the help you all provided over the past 12+ years.

Louis Davidson

Introduction

I often ask myself, "Why do I do this? Why am I writing another edition of this book? Is it worth it? Isn't there anything else that I could be doing that would be more beneficial to me, my family, or the human race? Well, of course there is. The fact is, however, I generally love relational databases, I love to write, and I want to help other people get better at what they do.

When I was first getting started designing databases, I learned from a few great mentors, but as I wanted to progress, I started looking for material on database design, and there wasn't much around. The best book I found was an edition of Chris Date's *An Introduction to Database Systems* (Addison Wesley, 2003), and I read as much as I could comprehend. The problem, however, was that I quickly got lost and started getting frustrated that I couldn't readily translate the theory of it all into a design process that really seems quite simple once you get down to it. I really didn't get it until I had spent years designing databases, failing over and over until I finally saw the simplicity of it all. In Chris's book, as well as other textbooks I had used, it was clear that a lot of theory, and even more math, went into creating the relational model.

If you want a deep understanding or relational theory, Chris's book is essential reading, along with lots of other books (Database Debunkings, www.dbdebunk.com/books.html, is a good place to start looking for more titles). The problem is that most 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. My book's goal is simply to fill that void and bridge the gap between academic textbooks and the purely implementation-oriented books that are commonly written on SQL Server. My intention is not to knock those books, not at all—I have numerous versions of those types of books on my shelf. This book is more of a technique-oriented book than a how-to book teaching you the features of SQL Server. I will cover many 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.

If you have previous editions of this book, you might question why you need this next edition, and I ask myself that every time I sit down to work on the next edition. You might guess that the best reason is that I cover the new SQL Server 2012 features. Clearly that is a part of it, but the base features in the relational engine that you need to know to design and implement most databases is not changing tremendously over time. Under the covers, the engine has taken more leaps, and hardware has continued up and up as the years progress. The biggest changes to SQL Server 2012 for the relational programmer lie in some of the T-SQL language features, like windowing functions that come heavily into play for the programmer that will interact with your freshly designed and loaded databases.

No, the best reason to buy the latest version of the book is that I continue to work hard to come up with new content to make your job easier. I've reworked the chapter on normalization to be easier to understand, added quite a few more patterns of development to Chapter 7, included a walkthrough of the development process (including testing) in Chapter 6, some discussion about the different add-ins you can use to enhance your databases, and generally attempted to improve the entire book throughout to be more concise (without losing the folksy charm, naturally). Finally, I added a chapter about data warehousing, written by a great friend and fellow MVP Jessica Moss.

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 be likely to be writing this fifth edition of the book. 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. Both of them were very tough on my writing style and did wonders on the structure of the book. Another reason is simply experience, as over eight 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 (drsql.org). You may also find an addendum there that covers any material 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 to 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 databases. 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 comprised of the following chapters, with the first five chapters being an introduction to the fundamental topics and process that one needs to go through/know before designing a database. Chapters 6 is an exercise in learning how a database is put together using scripts, and the rest of the book is taking topics of design and implementation and providing 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: Initial 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 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: Data Protection with CHECK Constraints and Triggers. Beyond the way data is arranged in tables and columns, other business rules may 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 even using SQL Server Profiler.

Chapter 10: Table Structures and Indexing. 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: Coding for Concurrency. As part of the code that's written, some consideration needs to be taken when you have to share resources. 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 your databases you implement to provide a standard problem solving interface for all of your systems while minimizing inter-database dependencies

Chapter 13: Considering Data Access Strategies. In this chapter, the concepts and concerns of writing code that accesses SQL Server are covered. I cover 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), as well as discuss 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 appendix, including example tests of how they work and tips and pointers for writing effective triggers.

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 an experience programming using any language.

Downloading the Code

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

Contacting the Authors

Don't hesitate to give me feedback on the book, anytime, at my web site (drsql.org) or my e-mail (louis@drsql.org). I'll try to improve any sections that people find lacking and publish them to my blog (http://sqlblog.com/blogs/louis_davidson) with the tag DesignBook, as well as to my web site (http://drsql.org/ProSQLServerDatabaseDesign.aspx). 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

A successful man is one who can lay a firm foundation with the bricks others have thrown at him.

—David Brinkley

Face it, education in fundamentals is rarely something that anyone considers exactly fun, at least unless you already have a love for the topic in 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 all skill levels. Administrative assistants build databases using Excel; newbie programmers do so with Access and even SQL Server over and over, and they rarely are 100% wrong. The problem is that in almost every case the design produced is fundamentally flawed, and these flaws are multiplied during the course of implementation; they may actually end up requiring the user to do far more work than necessary and cause future developers even more pain. When you are finished with this book, you should be able to design databases that reduce the effects of 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: designing databases. 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 SQL PASS Summit [www.sqlpass.org]. For this chapter, we will start out looking at the basic background topics that are so 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 concise 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. In particular, misunderstanding the vital role of keys in the database can lead to severe data integrity issues and to the mistaken belief that such keys and constraints can be effectively implemented outside the database. (Here is a subtle hint: they can't.)
- *Relationships between entities*: We will briefly survey the different types of relationships that can exist between relational 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 functional 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. Some of these terms are misunderstood and misused by a large number (if not a majority) of people. If we are not in agreement on their meaning from the beginning, eventually you might end up wondering what the heck we're talking about. Some might say that 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.

Among the terms that need introduction is modeling, specifically data modeling. Modeling is the process of capturing the essence of a system in a known language that is common to the user. A data model is a specific type of model that focuses entirely on the storage and management of the data storage medium, reflecting all of the parts of a database. It is a tool that we will use throughout the process from documentation to the end of the process where users have a database. The term "modeling" is often used as a generic term for the overall process of creating a database. As you can see from this example, we need to get on the same page when it comes to the concepts and basic theories that are fundamental to proper database design.

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 were 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 different computer subsystem, 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 the 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 be considered relational.

After introducing Codd's rules, I will touch very briefly on the different standards as they have evolved over the years.

Introducing Codd's Rules for an RDBMS

I feel it is useful to start with Codd's rules, because while these rules are now 27 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 was not only 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 rows and columns of data that users worked with.

Many different data structures were in use back then 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 work just like everyone else's? Very few, and new innovations are coming every day.

While innovation is rarely a bad thing, innovation in relational databases is ideally limited to the layer that is encapsulated from the user's view. The same database code that worked 20 years ago could easily work today with the simple difference that it now runs a great deal faster. There have been advances in the language we use (SQL), but it hasn't changed tremendously because it just plain works.

Rule 2: Guaranteed Access

> *Each and every datum (atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name.*

This rule is an extension of the first rule's definition of how data is accessed. While all of the terms in this rule will be defined in greater detail later in this chapter, suffice it to say that columns are used to store individual points of data in a row of data, and a primary key is a way of uniquely identifying a row using one or more columns of data. This rule defines that, at a minimum, there will be a non-implementation-specific way to access data in the database. The user can simply ask for data based on known data that uniquely identifies the requested data. "Atomic" is a term that we will use frequently; it simply means a value that cannot be broken down any further without losing its fundamental value. It will be covered several more times in this chapter and again in Chapter 5 when we cover normalization.

Together with the first rule, rule two establishes a kind of addressing system for data as well. The table name locates the correct table; the primary key value finds the row containing an individual data item of interest, and the column is used to address an individual piece of data.

Rule 3: Systematic Treatment of NULL Values

> *NULL values (distinct from empty character string or a string of blank characters and distinct from zero or any other number) are supported in the fully relational RDBMS for representing missing information in a systematic way, independent of data type.*

Good grief, if there is one topic I would have happily avoided in this book, it is missing values and how they are implemented with NULLs. NULLs are the most loaded topic of all because they are so incredibly different to use than all other types of data values you will encounter, and they are so often interpreted and used wrong. However, if we are going to broach the subject sometime, we might as well do so now.

The NULL rule requires that the RDBMS support a method of representing "missing" data the same way for every implemented datatype. This is really important because it allows you to indicate that you have no value for every column consistently, without resorting to tricks. For example, assume you are making a list of how many computer mice you have, and you think you still have an Arc mouse, but you aren't sure. You list Arc mouse to