SQL
IN A NUTSHELL

Third Edition

Kevin E. Kline
with Daniel Kline and Brand Hunt

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

SQL in a Nutshell, Third Edition
by Kevin E. Kline with Daniel Kline and Brand Hunt

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editors: John Steele and Mary Treseler
Production Editor: Rachel Monaghan
Copyeditor: Rachel Head
Indexer: Angela Howard
Production Services: Octal Publishing, Inc.
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Jessamyn Read

Printing History:

January 2001:	First Edition.
September 2004:	Second Edition.
November 2008:	Third Edition.

ISBN: 978-0-596-51884-4
[M]

Table of Contents

Preface

Since its first incarnation in the 1970s, the Structured Query Language (SQL) has been developed hand in hand with the information boom, and as a result, it is the most widely used database manipulation language in business and industry. A number of different software companies and program developers, including those in the open source movement (*http://www.opensource.org*), have concurrently developed their own SQL dialects in response to specific professional needs. All the while, standards bodies have developed a growing list of common features.

SQL in a Nutshell, Third Edition, describes the latest ANSI standard, SQL2003 (SQL3) version of each SQL command, and then documents each platform's implementation of that command. In this book, you will find a concise explanation of the relational database management system (RDBMS) model, a clear-cut explanation of foundational RDBMS concepts, and thorough coverage of SQL syntax and commands.

Most importantly, at least if you're a programmer or developer, *SQL in a Nutshell*, Third Edition, provides a concise guide both to the most popular commercial database packages on the market (Microsoft SQL Server and Oracle). It is also the guide for two of the best-known open source database products (MySQL and PostgreSQL). The attention this book pays to open source SQL platforms recognizes the growing importance of the open source movement within the computing community.

The SQL syntax covered in this book includes:

- ANSI SQL2003 (also known as SQL3) standard syntax
- MySQL version 5.1
- Oracle Database 11*g*
- PostgreSQL version 8.2.1
- Microsoft SQL Server 2008

Why This Book?

The primary source of information for relational databases is the documentation and help files provided by the vendors themselves. While each vendor's documentation is an indispensable resource that most database programmers and database administrators turn to first, this documentation has a number of limitations:

- It describes the vendor's implementation of SQL without giving you any context as to how well that implementation meets the ANSI standard for SQL.
- It covers only a single, specific vendor's product. There is no coverage of translation, migration, or integration issues.
- It typically describes programming methods in a multitude of small, disconnected documents or help files.
- It covers individual commands, often in confusing detail, obscuring the simple and direct uses of commands that programmers and administrators use every day.

In other words, the documentation included with a vendor's database provides an exhaustive explanation of every aspect of that particular vendor's platform. This is only natural; after all, help texts are geared toward delivering the main facts about a product. They'll tell you a command's specific syntax (and all its obscure variants) and, in general terms, how to implement it. However, if you move between RDBMSs and you need to be productive very quickly, you will rarely use those obscure command variations; instead, you'll utilize the capabilities most common in real-life situations.

This book begins where the vendor documentation ends by distilling the experiences of professional database administrators and developers who have used these SQL variants day in and day out to support complex enterprise applications. It offers you the benefit of their experience in a compact and easily usable format. Whether SQL is new to you or you have been using SQL since its earliest days, there are always new tips and techniques to learn. And when you're moving between different implementations, it's always important to find out about the issues that can bite you if you're not careful and informed.

Who Should Read This Book?

SQL in a Nutshell, Third Edition, benefits several groups of users. It will be useful for programmers who require a concise and handy SQL reference tool; for developers who need to migrate from one SQL dialect to another; and for database administrators (DBAs) who need to both execute a myriad of SQL statements to keep their enterprise databases up and running, and create and manage objects such as tables, indexes, and views.

This book is a reference work, not a tutorial. The writing is not expository. For example, we won't explain the concept of an elementary loop. Experienced developers already know such things—you want the meat. So we will explain, for example, the detailed workings of an ANSI standard cursor, how it works on each of the database platforms we cover, the special capabilities of cursors on each database platform, and the various pitfalls of cursors and how to get around them.

While we don't intend for *SQL in a Nutshell*, Third Edition, to be a tutorial on SQL or a handbook for database design, we do provide some brief coverage of introductory topics, and we hope you'll find that helpful. Chapter 1 and Chapter 2 provide a concise introduction to SQL, covering the general origins, essential structure, and basic operation of the language. If you're new to SQL, these chapters will help you get started.

How This Book Is Organized

SQL in a Nutshell, Third Edition, is divided into four chapters and one appendix:

Chapter 1, *SQL History and Implementations*
: Discusses the relational database model, describes the current and previous SQL standards, and introduces the SQL implementations covered in this book.

Chapter 2, *Foundational Concepts*
: Describes the fundamental concepts necessary for understanding relational databases and SQL commands.

Chapter 3, *SQL Statement Command Reference*
: Provides an alphabetical command reference to SQL statements. This chapter details the latest ANSI standard (SQL3) for each command, as well as the implementation of each command by MySQL, Oracle, PostgreSQL, and SQL Server.

Chapter 4, *SQL Functions*
: Provides an alphabetical reference of the ANSI SQL3 functions, describing vendor implementations of all SQL3 functions. In addition, this chapter includes coverage of all platform-specific functions that are unique to each implementation.

Appendix, *Shared and Platform-Specific Keywords*
: Provides a table of keywords declared in SQL3 and by the different database platforms. You can use this table to look for words that you should not use for object or variable names.

How to Use This Book

SQL in a Nutshell, Third Edition, is primarily a command reference. As a consequence, you'll probably use it to look up a variety of SQL commands and functions. However, with documentation for the ANSI standard itself plus four database platforms, the description for each command has the potential to get very large.

In order to reduce the verbiage describing each command, we compare each platform's implementation to the SQL3 standard. If the platform supports a clause described in the SQL3 discussion, we won't repeat that clause again.

Generic and transportable examples are provided within the body of each SQL3 command description. Since the SQL3 standard is ahead of most database platforms, examples aren't provided for elements of the SQL3 commands that are not

supported by any platform discussed in this book. In addition, more examples are provided for each database platform that highlight unique extensions and enhancements, of which there are many.

We recognize that our approach may necessitate jumping from a description of a platform's implementation of a command back to the corresponding SQL3 command description. However, we felt that this was better than packing the book with hundreds of pages of redundant content.

Resources

The following websites provide additional information about the various platforms covered in this book:

MySQL
: The corporate resource for MySQL is *http://www.mysql.com*, and another good site is *http://dev.mysql.com/doc/refman/5.1/en/*. A great developer resource with lots of useful tips is Devshed.com. See *http://www.devshed.com/c/b/MySQL/* for MySQL-specific information.

PostgreSQL
: The home for this open source database is located at *http://www.postgresql.org*. In addition to making a great deal of useful information available for download, this site also maintains mailing lists for PostgreSQL users. Another PostgreSQL site worth investigating is *http://www.pgsql.com*, which offers support for commercial customers.

Oracle
: Oracle's cyberspace home is *http://www.oracle.com*. A great resource for hardcore Oracle users is *http://www.oracle.com/technology/*. You can also find all Oracle documentation at *http://www.oracle.com/technology/documentation/index.html*. For useful independent information about Oracle, be sure to check out the Independent Oracle User Group at *http://www.ioug.org*.

SQL Server
: The official Microsoft SQL Server website is *http://www.microsoft.com/sql/*. Another good resource is found at the home of the Professional Association for SQL Server (PASS) at *http://www.sqlpass.org*.

Changes in the Third Edition

One of the biggest reasons to release a new edition of a technology book is because the technology has progressed. Since the second edition of this book was published, the ANSI standard has released one new version and all of the database platforms it covers have delivered at least one major release. Consequently, our readers want fresh content on the latest versions of SQL in the marketplace today.

Here are more details about changes in this third edition:

Reduced footprint
: The readership of *SQL in a Nutshell*, Second Edition, loved its expansive coverage of all major database platforms. However, maintaining such a huge amount of content proved to be too difficult for the return on the investment. Therefore, based upon the results of a large readership survey, we decided to remove the two least popular database platforms from this edition: Sybase Adaptive Server and IBM's DB2 UDB.

Improved organization
: The second edition did a great job of presenting everything readers could want to know about all of the commands and functions available in SQL and the major database platforms, but that didn't mean the content was always easy to find or navigate. We've added better indexes, tables of content, and page headers and footers so you can navigate much more quickly and effectively.

More examples
: It's impossible to have too many examples. We've added to our already large set of basic examples, including more sample code that highlights the unique and powerful capabilities of the SQL standard and the extensions offered by each database platform.

Conventions Used in This Book

This book uses the following typographical conventions:

`Constant width`
: Used to indicate programming syntax, code fragments, and examples.

`Constant width italic`
: Used to indicate variables in code that should be replaced with user-supplied values.

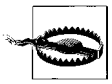
`Constant width bold`
: Used in code sections to highlight portions of the code.

Italic
: Used to introduce new terms, for emphasis, to indicate commands or user-specified file and directory names, and to indicate variables within text.

Bold
: Used to display the names of database objects, such as tables, columns, and stored procedures.

UPPERCASE ITALIC
: Used to indicate SQL keywords when they appear in the text.

Indicates a tip, suggestion, or general note.



Indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from this book *does* require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation *does* require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*SQL in a Nutshell*, Third Edition, by Kevin E. Kline with Daniel Kline and Brand Hunt. Copyright 2009 O'Reilly Media, Inc., 978-0-596-51884-4."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

How to Contact Us

We have tested and verified the information in this book to the best of our ability, but you may find that features have changed (or even that we have made mistakes!). We want to hear from you, especially with information that will make this book better. Please let us know about any errors you find, as well as your suggestions for future editions, by writing to:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the U.S. or Canada)
707-829-0515 (international/local)
707-829-0104 (fax)

We have a website for this book, where we'll list any examples, errata, or plans for future editions. You can access this page at:

http://www.oreilly.com/catalog/9780596518844/

Please help us out by pointing out any typos or syntactical errors that you encounter. (You can imagine how hard it is to proofread a book covering the ANSI standard and four separate products.) You may also ask technical questions or comment on the book by sending an email to:

bookquestions@oreilly.com

For more information about our books, conferences, software, Resource Centers, and the O'Reilly Network, see the O'Reilly website:

http://www.oreilly.com

Safari® Books Online

When you see a Safari® Books Online icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at *http://safari.oreilly.com*.

Acknowledgments

We'd like to take a moment to thank a few special individuals at O'Reilly Media. First, we owe a huge debt of gratitude to Julie Steele, the editor of this third edition. Julie helped keep our noses to the grindstone and ensured that we finished what we started. With her helpful and relaxed work style, Julie was always a pleasure to work with. Thank you for all you've done for us!

We also owe a debt to our fine technical reviewers. To Peter Gulutzan (SQL Standard, from the second edition), Thomas Lockhart (PostgreSQL), Ronald Bradford (Oracle/MySQL), and Richard Sonnen (Oracle): we owe you a hearty thanks! Your contributions have greatly improved the accuracy, readability, and value of this book. Without you, our sections on each of the language extensions would have been on shaky ground. In addition, we'd like to tip our hat to Peter Gulutzan and Trudy Pelzer for their book *SQL-99 Complete, Really!* (R&D), which helped us understand the ANSI SQL3 standards.

Kevin E. Kline's Acknowledgments

Many people helped deliver the big, thick book you hold in your hands. This note expresses our appreciation to those who helped make this book a reality.

First of all, a big thanks to my two awesome coauthors, Dan and Brand. You guys are amazing and a pleasure to work with. Next, Julie Steele, our editor at O'Reilly Media, gets a big hug for all of her help. You helped keep us on task and on track. Thank you!

To all of my colleagues at Quest Software go very big thanks for your support and encouragement. Christian Hasker, Andy Grant, Heather Eichman, David Gugick, Billy Bosworth, Douglas Chrystall, David Swanson, Jason Hall, Ariel Weil, and my many other friends at Quest Software: thank you for making these last six years with Quest Software such a blast.

Here's a dedication to my loved ones, Dylan, Emily, Anna, and Katie. You were my hope and breath and light when it seemed that no hope or breath or light remained anywhere in the world. Thank you for loving me so completely and so selflessly. And finally, to Rachel, more precious than jewels and more valuable than rubies, your love has restored my heart and my faith.

Daniel Kline's Acknowledgments

I'd like to thank my brother, Kevin, for his continued willingness to work with me; my colleagues at the University of Alaska Anchorage for their suggestions; and the users of the first two editions of *SQL in a Nutshell* for their honest feedback and useful critiques. We've also received some terrific feedback from the different translators of the first two editions, and I'd like to thank them for their help as well.

Brand Hunt's Acknowledgments

To my wife, Michelle: without your continued support and love, I wouldn't be a part of this project. I'm appreciative of every moment we've shared and of your forgiveness for my keeping you awake at night with the "tappity-tap-taps" emanating from the computer.

Thanks also to my parents, Rex and Jackie, the two biggest influences in everything I've ever done correctly—especially those things that frequently take multiple attempts (like writing!).

A huge thanks to my teammates, Kevin, Daniel, and Jonathan for letting me participate in this project and exercising so much patience in tutoring a first-time O'Reilly author. Your professionalism, work ethic, and ability to make the most tedious tasks fun is so admirable I plan to steal it and adopt it as my own!

Mad props to my friends and colleagues at Rogue Wave Software, ProWorks, NewCode Technology, and Systems Research and Development, for providing the ultimate sandbox for refining skills in SQL, databases, business, software development, writing, and friendship: Gus Waters, Greg Koerper, Marc Manley, Wendi Minne, Erin Foley, Elaine Cull, Randall Robinson, Dave Ritter, Edin Zulic, David Noor, Jim Shur, Chris Mosbrucker, Dan Robin, Mike Faux, Jason Prothero, Tim Romanowski, Andy Mosbrucker, Jeff Jonas, Jeff Butcher, Charlie Barbour, Steve Dunham, Brian Macy, and Ze'ev Mehler.

1

SQL History and Implementations

In the early 1970s, the seminal work of IBM research fellow Dr. E. F. Codd led to the development of a relational data model product called SEQUEL, or *Structured English Query Language*. SEQUEL ultimately became SQL, or *Structured Query Language*.

IBM, along with other relational database vendors, wanted a standardized method for accessing and manipulating data in a relational database. Although IBM was the first to develop relational database theory, Oracle was first to market the technology. Over time, SQL proved popular enough in the marketplace to attract the attention of the American National Standards Institute (ANSI), which released standards for SQL in 1986, 1989, 1992, 1999, 2003, and 2006. This text covers the ANSI 2003 standard because the 2006 standard deals with elements of SQL outside the scope of the commands described in this book. (In essence, the SQL2006 standard describes how XML would be used in SQL.)

Since 1986, various competing languages have allowed programmers and developers to access and manipulate relational data. However, few were as easy to learn or as universally accepted as SQL. Programmers and administrators now have the benefit of being able to learn a single language that, with minor adjustments, is applicable to a wide variety of database platforms, applications, and products.

SQL in a Nutshell, Third Edition, provides the syntax for five common implementations of SQL2003 (SQL3):

- The ANSI SQL standard
- MySQL version 5.1
- Oracle Database 11*g*
- PostgreSQL version 8.3
- Microsoft's SQL Server 2008

The Relational Model and ANSI SQL

Relational database management systems (RDBMSs) such as those covered in this book are the primary engines of information systems worldwide, and particularly of web applications and distributed client/server computing systems. They enable a multitude of users to quickly and simultaneously access, create, edit, and manipulate data without impacting other users. They also allow developers to write useful applications to access their resources and provide administrators with the capabilities they need to maintain, secure, and optimize organizational data resources.

An RDBMS is defined as a system whose users view data as a collection of tables related to each other through common data values. Data is stored in *tables*, which are composed of *rows* and *columns*. Tables of independent data can be linked (or *related*) to one another if they each have unique, identifying columns of data (called *keys*) that represent data values held in common. E. F. Codd first described relational database theory in his landmark paper "A Relational Model of Data for Large Shared Data Banks," published in the *Communications of the ACM* (Association for Computing Machinery) in June, 1970. Under Codd's new relational data model, data was *structured* (into tables of rows and columns); *manageable* using operations such as selections, projections, and joins; and *consistent* as the result of integrity rules such as keys and referential integrity. Codd also articulated rules that governed how a relational database should be designed. The process for applying these rules is now known as *normalization*.

Codd's Rules for Relational Database Systems

Codd applied rigorous mathematical theories (primarily set theory) to the management of data, and he compiled a list of criteria a database must meet to be considered relational. At its core, the relational database concept centers around storing data in tables. This concept is now so common as to seem trivial; however, not long ago the goal of designing a system capable of sustaining the relational model was considered a long shot with limited usefulness.

Following are Codd's *Twelve Principles of Relational Databases*:

1. Information is represented logically in tables.
2. Data must be logically accessible by table, primary key, and column.
3. Null values must be uniformly treated as "missing information," not as empty strings, blanks, or zeros.
4. Metadata (data about the database) must be stored in the database just as regular data is.
5. A single language must be able to define data, views, integrity constraints, authorization, transactions, and data manipulation.
6. Views must show the updates of their base tables and vice versa.
7. A single operation must be available to do each of the following operations: retrieve data, insert data, update data, or delete data.
8. Batch and end-user operations are logically separate from physical storage and access methods.

9. Batch and end-user operations can change the database schema without having to recreate it or the applications built upon it.
10. Integrity constraints must be available and stored in the metadata, not in an application program.
11. The data manipulation language of the relational system should not care where or how the physical data is distributed and should not require alteration if the physical data is centralized or distributed.
12. Any row processing done in the system must obey the same integrity rules and constraints that set-processing operations do.

These principles continue to be the litmus test used to validate the "relational" characteristics of a database platform: a database that does not meet all of these rules is not fully relational. While these rules do not apply to applications development, they do determine whether the database engine itself can be considered truly "relational." Currently, most commercial RDBMS products pass Codd's test. Among the platforms discussed in *SQL in a Nutshell*, Third Edition, only MySQL failed to support all of these requirements, and only then in releases prior to the one covered in this book.

Understanding Codd's principles assists programmers and developers in the proper development and design of relational databases (RDBs). The following sections detail how some of these requirements are met within SQL using RDBs.

Data structures (rules 1, 2, and 8)

Codd's rules 1 and 2 state that "information is represented logically in tables" and that "data must be logically accessible by table, primary key, and column." So, the process of defining a table for a relational database does not require that programs instruct the database how to interact with the underlying physical data structures. Furthermore, SQL logically isolates the processes of accessing data and physically maintaining that data, as required by rule 8: "batch and end-user operations are logically separate from physical storage and access methods."

In the relational model, data is shown logically as a two-dimensional *table* that describes a single entity (for example, business expenses). Academics refer to tables as *entities* and to columns as *attributes*. Tables are composed of *rows*, or *records* (academics call them *tuples*), and *columns* (called *attributes*, since each column of a table describes a specific attribute of the entity). The intersection of a record and a column provides a single *value*. The column or columns whose values uniquely identify each record can act as a *primary key*. These days this representation seems elementary, but it was actually quite innovative when it was first proposed.

SQL3 defines a whole data structure hierarchy beyond simple tables, though tables are the core data structure. Relational design handles data on a table-by-table basis, not on a record-by-record basis. This table-centric orientation is the heart of set programming. Consequently, almost all SQL commands operate much more efficiently against sets of data within or across tables than against individual records. Said another way, effective SQL programming requires that you think in terms of sets of data, rather than of individual rows.

Figure 1-1 is a description of the SQL3 terminology used to describe the hierarchical data structures used by a relational database: *clusters* contain sets of *catalogs*; *catalogs* contain sets of *schemas*; *schemas* contain sets of *objects*, such as *tables* and *views*; and *tables* are composed of sets of *columns* and *records*.

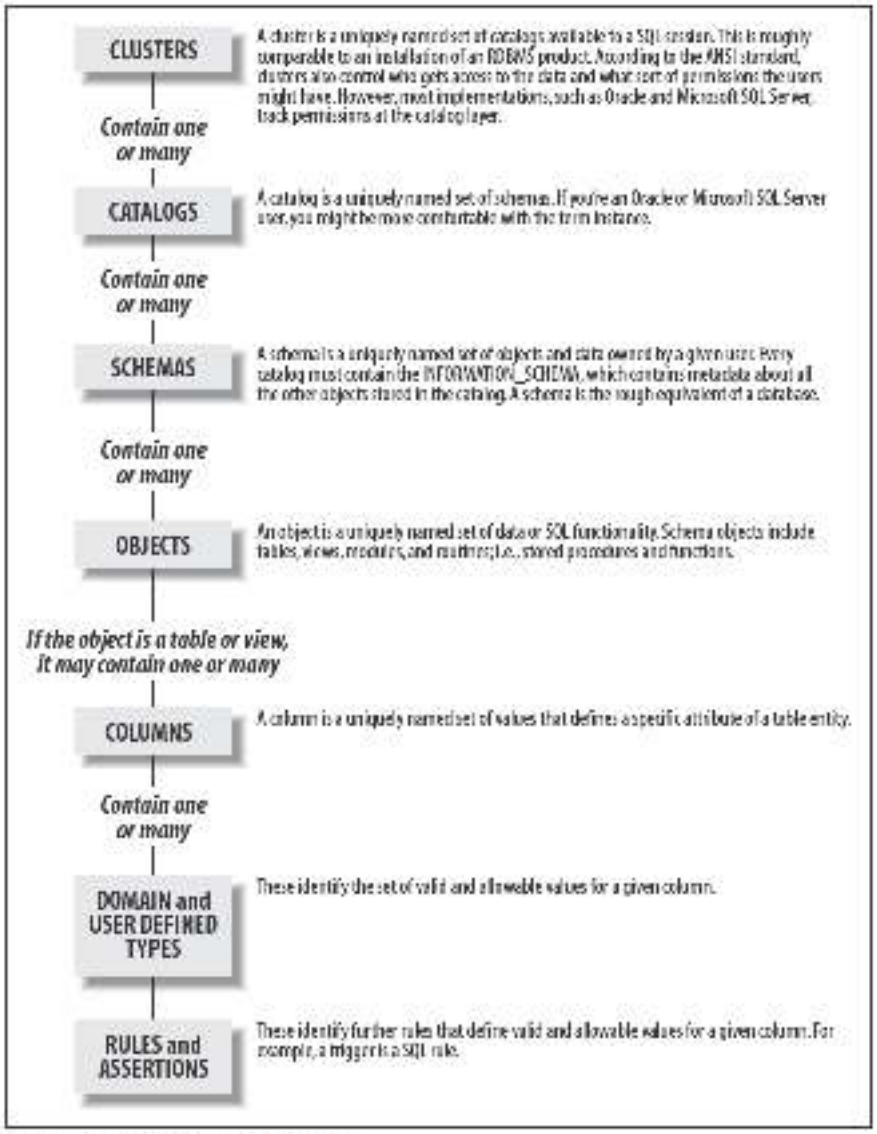


Figure 1-1. SQL3 dataset hierarchy

For example, in a **Business_Expense** table, a column called **Expense_Date** might show when an expense was incurred. Each record in the table describes a specific entity; in this case, everything that makes up a business expense (when it happened, how much it cost, who incurred the expense, what it was for, and so on).

Each attribute of an expense—in other words, each column—is supposed to be *atomic*; that is, each column is supposed to contain one, and only one, value. If a table is constructed in which the intersection of a row and column can contain more than one distinct value, one of SQL's primary design guidelines has been violated. (Some of the database platforms discussed in this book do allow you to place more than one value into a column, via the *VARRAY* or *TABLE* datatypes.)

Rules of behavior are specified for column values. Foremost is that column values must share a common *domain*, better known as a *datatype*. For example, if the **Expense_Date** field is defined as having a *DATE* datatype, the value *ELMER* should not be placed into that field because it is a string, not a date, and the **Expense_Date** field can contain only dates. In addition, SQL3 allows further control of column values through the application of *constraints* (discussed in detail in Chapter 2) and *assertions*. A SQL constraint might, for instance, limit **Expense_Date** to expenses less than a year old. Additionally, data access for all individuals and computer processes is controlled at the schema level by an *AuthorizationID* or *user*. Permissions to access or modify specific sets of data may be granted or restricted on a per-user basis.

SQL databases also employ *character sets* and *collations*. Character sets are the "symbols" or "alphabets" used by the "language" of the data. For example, the American English character set does not contain the special character for *ñ* in the Spanish character set. Collations are sets of sorting rules that operate on a character set. A collation defines how a given data manipulation operation sorts data. For example, an American English character set might be sorted either by *character-order, case-insensitive*, or by *character-order, case-sensitive*.

The ANSI standard does not say how sorts should be done, only that platforms must provide common collations found in a particular language.

It is important to know what collation you are using when writing SQL code against a database platform, as it can have a direct impact on how queries behave, and particularly on the behavior of the *WHERE* and *ORDER BY* clauses of *SELECT* statements. For example, a query that sorts data using a binary collation will return data in a very different order than one that sorts data using, say, an American English collation.

NULLs (rule 3)

Most databases allow any of their supported datatypes to store NULL values. Inexperienced SQL programmers and developers tend to think of NULL as zero or blank. In fact, NULL is neither of these. In SQL3, NULL literally means that the value is unknown or indeterminate. (This question alone—whether NULL should be considered unknown or indeterminate—is the subject of much academic debate.) This differentiation enables a database designer to distinguish between those entries that represent a deliberately placed zero, for example, and those where either the data is not recorded in the system or a NULL has been explicitly entered. As an illustration of this semantic difference, consider a system that tracks payments. If a product has a NULL price, that does not mean the product

is free; instead, a NULL price indicates that the amount is not known or perhaps has not yet been determined.

There is a good deal of differentiation between the database platforms in terms of how they handle NULL values. This leads to some major porting issues between those platforms relating to NULLs. For example, an empty string (i.e., a NULL string) is inserted as a NULL value on Oracle. All the other databases covered in this book permit the insertion of an empty string into *VARCHAR* and *CHAR* columns.

One side effect of the indeterminate nature of a NULL value is that it cannot be used in a calculation or a comparison. Here are a few brief but very important rules, from the ANSI standard, to remember about the behavior of NULL values when dealing with NULLs in SQL statements:

- A NULL value cannot be inserted into a column defined as *NOT NULL*.
- NULL values are not equal to each other. It is a frequent mistake to compare two columns that contain NULL and expect the NULL values to match. (The proper way to identify a NULL value in a *WHERE* clause or in a Boolean expression is to use phrases such as "value IS NULL" and "value IS NOT NULL".)
- A column containing a NULL value is ignored in the calculation of aggregate values such as *AVG*, *SUM*, or *MAX COUNT*.
- When columns that contain NULL values are listed in the *GROUP BY* clause of a query, the query output contains a single row for NULL values. In essence, the ANSI standard considers all NULLs found to be in a single group.
- *DISTINCT* and *ORDER BY* clauses, like *GROUP BY*, also see NULL values as indistinguishable from each other. With the *ORDER BY* clause, the vendor is free to choose whether NULL values sort high (first in the result set) or sort low (last in the result set) by default.

Metadata (rules 4 and 10)

Codd's fourth rule for relational databases states that data about the database must be stored in standard tables, just as all other data is. Data that describes the database itself is called *metadata*. For example, every time you create a new table or view in a database, records are created and stored that describe the new table. Additional records are needed to store any columns, keys, or constraints on the table. This technique is implemented in most commercial and open source SQL database products. For example, SQL Server uses what it calls "system tables" to track all the information about the databases, tables, and database objects in any given database. It also has "system databases" that keep track of information about the server on which the database is installed and configured.

The language (rules 5 and 11)

Codd's rules do *not* require SQL to be used with a relational database. His rules, particularly rules 5 and 11, only specify how the language should behave when coupled with a relational database. At one time SQL competed with other languages (such as Digital's RDO and Fox/PRO) that might have fit the relational bill, but SQL won out, for three reasons. First, SQL is a relatively simple, intuitive, English-like language that handles most aspects of data manipulation. Second, SQL is satisfyingly high-level. A programmer or database administrator (DBA) does not have to spend time ensuring that data is stored in the proper memory registers or that data is cached to disk: the database management system (DBMS) handles that task automatically. Finally, because no single vendor owns SQL, it was adopted across a number of platforms.

Views (rule 6)

A *view* is a virtual table that does not exist as a physical repository of data, but is instead constructed on the fly from a *SELECT* statement whenever that view is queried. Views enable you to construct different representations of the same source data for a variety of audiences without having to alter the way in which the data is stored.

Some vendors support database objects called *materialized views*. Don't let the similarity of terms confuse you; materialized views are not governed by the same rules as ANSI standard views.

Set operations (rules 7 and 12)

Other database manipulation languages, such as the venerable Xbase, perform their data operations quite differently from SQL. These languages require you to tell the program exactly how to treat the data, one record at a time. Since the program cycles down through a list of records, performing its logic on one record after another, this style of programming is frequently called *row processing* or *procedural programming*.

In contrast, SQL programs operate on logical *sets* of data. Set theory is applied in almost all SQL statements, including *SELECT*, *INSERT*, *UPDATE*, and *DELETE* statements. In effect, data is selected from a set called a "table." Unlike the row processing style, set *processing* allows a programmer to tell the database simply *what* is required, not *how* each individual piece of data should be handled. Sometimes set processing is referred to as *declarative processing*, since a programmer declares only what data is wanted (as in, "Give me all employees in the southern region who earn more than $70,000 per year") rather than describing the exact procedure to be used to retrieve or manipulate the data.

Set theory was the brainchild of mathematician Georg Cantor, who developed it at the end of the nineteenth century. At the time, set theory (and Cantor's theory of the infinite) was quite controversial. Today, set theory is such a common part of life that it is learned in elementary school. Things like card catalogs, the Dewey Decimal System, and alphabetized phone books are all simple and common examples of applied set theory.

Examples of set theory in conjunction with relational databases are detailed in the following section.

Codd's Rules in Action: Simple SELECT Examples

Up to this point, this chapter has focused on the individual aspects of a relational database platform as defined by Codd and implemented under ANSI SQL. This section presents a high-level overview of the most important SQL statement, *SELECT*, and some of its most salient points—namely, the relational operations known as *projections*, *selections*, and *joins*:

Projection
: Retrieves specific columns of data

Selection
: Retrieves specific rows of data

Join
: Returns columns and rows from two or more tables in a single result set

Although at first glance it might appear as though the *SELECT* statement deals only with the relational selection operation, in actuality, *SELECT* deals with all three operations.

The following statement embodies the projection operation by selecting the first and last names of an author, plus his home state, from the **authors** table:

```
SELECT au_fname, au_lname, state
FROM   authors
```

The results from any such *SELECT* statement are presented as another table of data:

```
au_fname            au_lname                          state
------------------- --------------------------------- -----
Johnson             White                             CA
Marjorie            Green                             CA
Cheryl              Carson                            CA
Michael             O'Leary                           CA
Meander             Smith                             KS
Morningstar         Greene                            TN
Reginald            Blotchet-Halls                    OR
Innes               del Castillo                      MI
```

The resulting data is sometimes called a *result set*, *work table*, or *derived table*, differentiating it from the *base table* in the database that is the target of the *SELECT* statement.

It is important to note that the relational operation of projection, not selection, is specified using the *SELECT* clause (that is, the keyword *SELECT* followed by a list of expressions to be retrieved) of a *SELECT* statement. Selection—the operation of retrieving specific rows of data—is specified using the *WHERE* clause in a *SELECT* statement. *WHERE* filters out unwanted rows of data and retrieves only the requested rows. Continuing with the previous example, the following statement selects authors from states other than California:

```
SELECT au_fname, au_lname, state
FROM   authors
WHERE  state <> 'CA'
```

Whereas the first query retrieved all authors, the result of this second query is a much smaller subset of records:

```
au_fname             au_lname                           state
-------------------- ---------------------------------- -----
Meander              Smith                              KS
MorningStar          Greene                             TN
Reginald             Blotchet-Halls                     OR
Innes                del Castillo                       MI
```

By combining the capabilities of projection and selection in a single query, you can use SQL to retrieve only the columns and records that you need at any given time.

Joins are the next, and last, relational operation we're going to talk about in this section. A join relates one table to another in order to return a result set consisting of related data from both tables.

Different vendors allow you to join varying numbers of tables in a single join operation. For example, Oracle places no limit on the number of tables in a join, while Microsoft SQL Server allows up to 256 tables in a join operation.

The ANSI standard method of performing joins is to use the *JOIN* clause in a *SELECT* statement. An older method, know as a *theta join*, performs the join analysis in the *WHERE* clause. The following example shows both approaches. Each statement retrieves employee information from the employee base table as well as job descriptions from the jobs base table. The first *SELECT* uses the newer, ANSI *JOIN* clause, while the second *SELECT* uses a theta join:

```
-- ANSI style
SELECT a.au_fname, a.au_lname, t.title_id
FROM   authors AS a
JOIN   titleauthor AS t ON a.au_id = t.au_id
WHERE  a.state <> 'CA'

-- Theta style
SELECT a.au_fname, a.au_lname, t.title_id
FROM   authors AS a,
       titleauthor AS t
WHERE  a.au_id = t.au_id
  AND a.state <> 'CA'
```

For more information about joins, see the "JOIN Subclause" section in Chapter 3.

History of the SQL Standard

In response to the proliferation of SQL dialects, ANSI published its first SQL standard in 1986 to bring about greater conformity among vendors. This was followed by a second, widely adopted standard in 1989. The International Standards Organization (ISO) also approved the SQL standard. ANSI released one update in 1992, known as SQL92 or SQL2, and another in 1999, termed SQL99 or SQL3. The next update, made in 2003, is also referred to as SQL3 (or SQL2003). When we use that term in this book, we are referring to the 2003 revision of the standard.

Each time it revises the SQL standard, ANSI adds new features and incorporates new commands and capabilities into the language. For example, the SQL99 standard added a group of capabilities that handled object-oriented datatype extensions.

What's New in SQL2006

The ANSI standards body that regulates SQL issued a new standard in 2006, in which the important major improvements of SQL3 were retained and augmented. The ANSI SQL2006 release was evolutionary over the SQL3 release, but it did not include any significant changes to the SQL3 commands and functions that were described in the second edition of this book. Instead, SQL2006 described an entirely new functional area of behavior for the SQL standard. Briefly, SQL2006 describes how SQL and XML (the eXtensible Markup Language) interact. For example, the SQL2006 standard describes how to import and store XML data in a SQL database, manipulate that data, and then publish the data both in native XML form and as conventional SQL data wrapped in XML form. The SQL2006 standard provides a means of integrating SQL application code with XQuery, the XML Query Language standardized by the World Wide Web Consortium (W3C). Because XML and XQuery are disciplines in their own right, they are considered beyond the scope of this book and are not covered here.

What's New in SQL2003 (SQL3)

SQL99 had two main parts, *Foundation:1999* and *Bindings:1999*. The SQL3 Foundation section includes all of the Foundation and Bindings standards from SQL99, as well as a new section called *Schemata*.

The Core requirements of SQL3 did not change from Core SQL99, so the database platforms that conformed to Core SQL99 automatically conform to SQL3. Although the Core of SQL3 has no additions (except for a few new reserved words), a number of individual statements and behaviors have been updated or modified. Because these updates are reflected in the individual syntax descriptions of each statement in Chapter 3, we won't spend time on them here.

A few elements of the Core in SQL99 were deleted in SQL3, including:

- The *BIT* and *BIT VARYING* datatypes
- The *UNION JOIN* clause
- The *UPDATE…SET ROW* statement

A number of other features, most of which were or are rather obscure, have also been added, deleted, or renamed. Many of the new features of the SQL3 standard are currently interesting mostly from an academic standpoint, because none of the database platforms support them yet. However, a few new features hold more than passing interest:

Elementary OLAP functions
: SQL3 adds an Online Analytical Processing (OLAP) amendment, including a number of windowing functions to support widely used calculations such as moving averages and cumulative sums. Windowing functions are aggregates computed over a window of data: *ROW_NUMBER*, *RANK*, *DENSE_RANK*, *PERCENT_RANK*, and *CUME_DIST*. OLAP functions are fully described in T611 of the standard. Some database platforms are starting to support the OLAP functions. Refer to Chapter 1 for details.

Sampling
: SQL3 adds the *TABLESAMPLE* clause to the *FROM* clause. This is useful for statistical queries on large databases, such as a data warehouse.

Enhanced numeric functions
: SQL3 adds a large number of numeric functions. In this case, the standard was mostly catching up with the trend in the industry, since one or more database platforms already supported the new functions. Refer to Chapter 1 for details.

Levels of Conformance

SQL99 is built upon SQL92's *levels of conformance*. SQL92 first introduced levels of conformance by defining three categories: *Entry*, *Intermediate*, and *Full*. Vendors had to achieve at least Entry-level conformance to claim ANSI SQL compliance. The U.S. National Institute of Standards and Technology (NIST) later added the *Transitional* level between the Entry and Intermediate levels, so NIST's levels of conformance were Entry, Transitional, Intermediate, and Full, while ANSI's were only Entry, Intermediate, and Full. Each higher level of the standard was a superset of the subordinate level, meaning that each higher level included all the features of the lower levels of conformance.

Later, SQL99 altered the base levels of conformance, doing away with the Entry, Intermediate, and Full levels. With SQL99, vendors must implement all the features of the lowest level of conformance, Core SQL99, in order to claim (and publish) that they are SQL99 ready. Core SQL99 includes the old Entry SQL92 feature set, features from other SQL92 levels, and some brand new features. A vendor may also choose to implement additional feature packages described in the SQL99 standard.

Supplemental Features Packages in the SQL3 Standard

The SQL3 standard represents the ideal, but very few vendors currently meet or exceed the Core SQL3 requirements. The Core standard is like the interstate speed limit: some drivers go above it and others go below it, but few go exactly the speed limit. Similarly, vendor implementations can vary greatly.

Two committees—one within ANSI, the other within ISO, and both composed of representatives from virtually every RDBMS vendor—drafted the supplemental feature definitions described in this section. In this collaborative and somewhat political environment, vendors compromised on exactly which proposed features and implementations would be incorporated into the new standard.

New features in the ANSI standard often are derived from an existing product or are the outgrowth of new research and development in the academic community. Consequently, vendor adoption of specific ANSI standards can be spotty. A relatively new addition to the SQL3 standard is SQL/XML (greatly expanded in SQL2006.) The other parts of the SQL99 standard remain in SQL3, though their names may have changed and they may have been slightly rearranged.

The nine supplemental features packages, representing different subsets of commands, are platform-optional. Some features might show up in multiple packages, while others do not appear in any of the packages. These packages and their features are described in the following list:

Part 1, SQL/Framework
: Includes common definitions and concepts used throughout the standard. Defines the way the standard is structured and how the various parts relate to one another, and describes the conformance requirements set out by the standards committee.

Part 2, SQL/Foundation
: Includes the Core, an augmentation of the SQL99 Core. This is the largest and most important part of the standard.

Part 3, SQL/CLI (Call-Level Interface)
: Defines the call-level interface for dynamically invoking SQL statements from external application programs. Also includes over 60 routine specifications to facilitate the development of truly portable shrink-wrapped software.

Part 4, SQL/PSM (Persistent Stored Modules)
: Standardizes procedural language constructs similar to those found in database platform-specific SQL dialects such as PL/SQL and Transact-SQL.

Part 9, SQL/MED (Management of External Data)
: Defines the management of data located outside of the database platform using datalinks and a wrapper interface.

Part 10, SQL/OBJ (Object Language Binding)
: Describes how to embed SQL statements in Java programs. It is closely related to JDBC, but offers a few advantages. It is also very different from the traditional host language binding possible in early versions of the standard.

Part 11, SQL/Schemata
: Defines over 85 views (three more than in SQL99) used to describe the metadata of each database and stored in a special schema called *INFORMATION_SCHEMA*. Updates a number of views that existed in SQL99.