Beginning SQL

Paul Wilton and John W. Colby

WILEY

Wiley Publishing, Inc.

Beginning SQL

Beginning SQL

Paul Wilton and John W. Colby

WILEY

Wiley Publishing, Inc.

Beginning SQL

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

ISBN: 0-7645-7732-8

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

1MA/RW/QS/QV/IN

Library of Congress Cataloging-in-Publication Data

Wilton, Paul, 1969-
Beginning sql / Paul Wilton and John W. Colby.
p. cm.
Includes bibliographical references and index.
ISBN 0-7645-7732-8 (paper/website : alk. paper)
1. SQL (Computer program language) I. Colby, John W., 1954- II. Title.
QA76.73.S67W57 2005
005.75'65--dc22

2004031057

About the Authors

Paul Wilton

After an initial start as a Visual Basic applications programmer at the Ministry of Defense in the U.K., Paul found himself pulled into the Net. Having joined an Internet development company, he spent the last three years helping create Internet solutions and is currently working on an e-commerce Web site for a major British bank.

Paul's main skills are in developing Web front ends using DHTML, JavaScript, VBScript, and Visual Basic and back-end solutions with ASP, Visual Basic, and SQL Server. Currently, Paul is working on a new Web-based application that will hopefully make him millions. . . well, thousands at least!

Paul Wilton contributed Chapters 1–9 and Appendixes A, B and C to this book.

John W. Colby

John Colby is an independent consultant who has specialized in Access development since 1994. He has designed databases for companies in the U.S., Mexico, Canada, and Ireland. John is past president and current board member of Database Advisors, Inc. (`www.databaseAdvisors.com`), a not-for-profit organization dedicated to providing fellow developers with a place to discuss Access, SQL Server, Visual Basic, and other topics relative to modern database applications development. Database Advisors also allows developers to showcase their talents by sharing databases, wizards, and various code packages.

John lives in northwestern Connecticut with his wife and two small children. He enjoys music, travel, and all things computers, and he dreams of working from his laptop while enjoying travel with his family.

John W. Colby contributed Chapters 10–13 to this book.

Credits

Senior Acquisitions Editor
Jim Minatel

Development Editor
Brian Herrmann

Production Editor
Felicia Robinson

Technical Editor
Wiley-Dreamtech India Pvt Ltd

Copy Editor
Publication Services

Editorial Manager
Mary Beth Wakefield

Vice President & Executive Group Publisher
Richard Swadley

Vice President and Publisher
Joseph B. Wikert

Project Coordinator
April Farling

Graphics and Production Specialists
Lauren Goddard
Jennifer Heleine
Amanda Spagnuolo

Quality Control Technician
John Greenough
Leeann Harney
Jessica Kramer
Brian H. Walls

Proofreading and Indexing
TECHBOOKS Production Services

Paul Wilton: *With lots of love to my darling Beci, who, now that the book's finished, will get to see me for more than ten minutes a week.*

John W. Colby: *Dedicated to my son Robbie and my daughter Allie, who give me so much inspiration, and to my wife Mary, a wonderful soul mate and mother.*

Contents

Acknowledgments

Paul Wilton

Many thanks to Catherine who for many years supported me and ensured that my sanity chip remained plugged in. I'd also like to thank Brian Herrmann, who has been a great editor to work with and has done amazing work on the book and kept his professionalism and sense of humor even when faced with another of my "just a few more days and I'll get the chapter to you" emails! Thanks also to Jim Minatel for allowing me to subject the world to yet another of my books. Finally, pats and treats to my German shepherd Katie, who does an excellent job in warding off disturbances from door-to-door salespeople.

Introduction

Data, data, data! Data is where it's at as far as computers go, whether processing millions of calculations or keeping a record of your Aunt Maude's birthday. When it comes to storing data, the database is the king. In almost eight years of professional programming, every single project I've worked on has involved databases somewhere along the line — that's how essential they are to most business applications and projects. Admittedly, some areas, such as computer games, don't make the same use of databases. My guess is that "Mega Doom 99: The Final Bloody Massacre" isn't running an Oracle database in the background!

However, I have a confession! Around 10 years ago, when I first started learning about databases, I initially found them very confusing. I'd been programming in my spare time for a few years and was used to using text files to store information. I decided to leap right in and start creating databases and writing SQL, and I got very confused and odd results. Databases, their design, and their underlying concepts are very different from storing data in simple files, and the Structured Query Language (SQL) used to access and manipulate data in databases is very different from any procedural language. One of my first aims with this book is to soften the blow of new concepts and ways of doing things. To that end, I explain all the underlying concepts and theory you'll need to get started with databases and in programming with SQL. How to get the answers you want from a database and all the results you get will be fully explained, as SQL can throw up some surprises if you're not forewarned.

Another of my aims in writing this book is to get you quickly and effectively to the point where you're able to go off on your own and design your own databases and write your own SQL code in a practical environment. Personally, I dislike books that waffle on about every small detail and eventuality so that it takes months to be able to stand on your own feet and create your own work. I stick with the stuff that you'll find is used in most database applications, keeping the fine details and more advanced stuff for the later chapters. The first few chapters' aim is to get you up and running in SQL quickly, but they do not skimp on essential concepts, code, and techniques, which are all thoroughly discussed and backed up with lots of practical examples.

Finally, I'm a hands-on, practical person, and those are the sort of computer books I like to read, rather than books that contain lots of theory. This book reflects my "put it into action" nature and is full of examples and places where you can get hands-on experience. I do introduce and explain theory where it's necessary to build a foundation of understanding, but I do this only with the eventual aim of putting theory into practice. I use databases and SQL most days in my programming, and I hope to bring that real-world experience to this book.

Who This Book Is For

This book starts right from the basics with databases and SQL. Prior database or SQL knowledge is not necessary, as this book covers everything from database design to creating your first database and understanding how the SQL language is used with databases.

If you have some previous experience with databases and SQL, then you'll have a head start and you may want to just skim Chapter 1. You'll need to follow its instructions for creating the book's example database, as this is used for all the examples throughout the book.

What This Book Covers

This book will look at Structured Query Language, or SQL as it's usually abbreviated. SQL works with a database to create the database and to insert and extract data. Therefore, it's essential to understand the theory and concepts behind database systems. Hence, this book also covers database theory and database design, so that you're equipped to create an effective database.

The SQL code in this book reflects the modern SQL standards set by organizations such as the American National Standards Institute (ANSI) and the International Standards Organization (ISO). However, while standards are great, what's available for practical use is what really counts. This book, then, concentrates on the sort of SQL supported by most modern database systems. You should that find most of the code runs with little or no modification on most database systems released within the last six or seven years.

How This Book Is Structured

This book has been split into two main parts. The first part, which consists of Chapters 1–3, provides the foundations for understanding databases and SQL. The aim in this first part is to get you up to speed on all the essential details. These chapters take you through the following:

- ❑ The essentials of database theory
- ❑ Writing SQL code
- ❑ Good database design
- ❑ Creating a database
- ❑ Entering, updating, and deleting data using SQL
- ❑ Extracting data using SQL — more specifically, how to answer the sort of questions often posed of databases in real-life situations

By the time you've completed Chapter 3, you'll be ready to go out and create your own databases and write your own SQL code to a sufficient standard for many real-life programming situations. You may want to go and create a few databases of your own before returning to the second part of the book.

The second half of the book, Chapters 4 onward, goes into more detail and looks at more advanced topics. Its aim is to provide a fairly wide and thorough grounding in many aspects of SQL programming. The sort of topics covered include the following:

- Advanced database design, taking a look at the theory and practical application of normalization, and how to improve a database's efficiency and reliability
- Using and manipulating data with SQL's built-in data manipulation and calculation functions
- Selecting data from lots of different tables
- Database security
- Database optimization

The book also includes three appendixes. Appendix A contains the answers to the exercise questions in each chapter, so no peeking until you've given the questions a go. Appendix B covers how to download, install, and use each of the five supported database systems used by this book. Appendix C includes the initial data for the example database, which is available to download from `www.wrox.com` if you want to avoid aching fingers!

What You Need to Use This Book

To really make use of this book and run the examples, you need to have a database system on which to practice. This book's code has been thoroughly tested on the following five commonly available database systems:

- MySQL
- Microsoft SQL Server
- IBM DB2
- Microsoft Access
- Oracle 10g

The good news is that almost all of those can be downloaded free off the Internet as full or trial versions. In Appendix B, you'll learn where to download them, how to install them, and how to use them.

It's not a problem if you're using a different database system, because as often as possible, I've avoided database system-specific code and have kept to standard SQL supported by most database systems. Where it's impossible to have the same code for all the database systems, I've listed the ways around as well as alternative syntax. You'll likely find that one of the variations of syntax will work on your system, possibly with a little modification.

Conventions

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

Try It Out

The *Try It Out* is an exercise that you should work through, following the text in the book.

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

How It Works

After each Try It Out, the code you've typed will be explained in detail.

Boxes like this one hold important, not-to-be-forgotten information that is directly relevant to the surrounding text.

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

As for styles in the text:

- New terms and important words are *italicized* as they are introduced.
- Keyboard strokes are shown like this: Ctrl+A.
- Filenames, URLs, and code within the text are shown like so: `persistence.properties`.
- Code is presented in two different ways:

```
In code examples, new and important code is highlighted with a gray background.
```

```
The gray highlighting is not used for code that's less important in the present
context or that has been shown before.
```

Source Code

As you work through the examples in this book, you may choose either to type in all the code manually or to use the source code files that accompany the book. All of the source code used in this book is available for download at `http://www.wrox.com`. Once at the site, simply locate the book's title (either by using the Search box or by using one of the title lists) and click the Download Code link on the book's detail page to obtain all the source code for the book.

Because many books have similar titles, you may find it easiest to search by ISBN; for this book, the ISBN is 0-7645-7732-8.

Once you download the code, just decompress it with your favorite compression tool. Alternatively, you can go to the main Wrox code download page at `http://www.wrox.com/dynamic/books/download.aspx` to see the code available for this book and all other Wrox books.

Errata

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, like a spelling mistake or faulty piece of code, we would be very grateful for your feedback. By sending in errata, you may save another reader hours of frustration, and at the same time you will be helping us provide even higher-quality information.

To find the errata page for this book, go to `http://www.wrox.com` and locate the title using the Search box or one of the title lists. Then, on the Book Details page, click the Book Errata link. On this page, you can view all errata that has been submitted for this book and posted by Wrox editors. A complete book list including links to each book's errata is also available at `www.wrox.com/misc-pages/booklist.shtml`.

If you don't spot "your" error on the Book Errata page, go to `www.wrox.com/contact/techsupport.shtml` and complete the form there to send us the error you have found. We'll check the information and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.

p2p.wrox.com

For author and peer discussion, join the P2P forums at `p2p.wrox.com`. The forums are a Web-based system for you to post messages relating to Wrox books and related technologies and interact with other readers and technology users. The forums offer a subscription feature to email you topics of interest of your choosing when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

At `http://p2p.wrox.com` you will find a number of different forums that will help you not only as you read this book but also as you develop your own applications. To join the forums, just follow these steps:

1. Go to `p2p.wrox.com` and click the Register link.
2. Read the terms of use and click Agree.
3. Complete the required information to join as well as any optional information you wish to provide and click Submit.
4. You will receive an email with information describing how to verify your account and complete the joining process.

You can read messages in the forums without joining P2P, but in order to post your own messages, you must join.

Once you join, you can post new messages and respond to messages that other users post. You can read messages at any time on the Web. If you would like to have new messages from a particular forum emailed to you, click the Subscribe to this Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

1

Introduction to SQL

A nice, gentle introductory chapter, this chapter begins by looking at databases in terms of what they are and why and when you want to use them. Then the chapter turns to SQL and discovers how it links in with databases and how it can be useful. After tackling the basics of SQL and how it works in theory, you examine how to use it to create a database. This chapter also walks you through creating the structure of the example database used throughout the book.

By the end of the chapter, you should understand how a database enables you to efficiently organize and retrieve the information you want, as well as how to create a fully functional database, all ready and waiting to accept add data. But before diving headlong into writing lines of SQL code, it's helpful to know a little bit of background about databases.

A Brief History of Databases

Modern databases emerged in the 1960s thanks to research at IBM, among other companies. The research mainly centered around office automation, in particular automating data storage and indexing tasks that previously required a great deal of manual labor. Computing power and storage had become much cheaper, making the use of computers for data indexing and storage a viable solution. A pioneer in the database field was Charles W. Bachman, who received the Turing Award in 1973 for pioneering work in database technology. In 1970, an IBM researcher named Ted Codd published the first article on relational databases.

Although IBM was a leader in database research, Honeywell Information Systems, Inc., released a commercial product in 1976 based on the same principles as the IBM information system, but it was designed and implemented separately from IBM's work.

In the early 1980s, the first database systems built upon the SQL standard appeared from companies such as Oracle, with Oracle Version 2, and later SQL/DS from IBM, as well as a host of other systems from other companies.

Now that you have a brief idea of where databases came from, you can turn to the more practical task of what databases are and why and when to use them.

Identifying Databases

What is a database, you ask?

The Free On-Line Dictionary of Computing (`http://foldoc.doc.ic.ac.uk`) defines a database as "one or more large structured sets of persistent data, usually associated with software to update and query the data. A simple database might be a single file containing many records, each of which contains the same set of fields where each field is a certain fixed width."

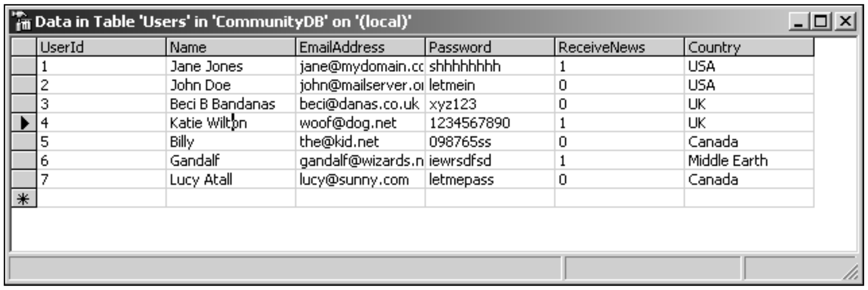
Breaking this definition down into something more manageable, first it says that a database consists of structured sets of data, which means that a database contains collections of data. For example, the database might contain the details of Uncle Bob's golf scores or data about all the books in a library. You probably wouldn't want to mix these two collections of data, or else when you want to find data about a book you'd have to look through irrelevant data on golf scores. In short, databases help you organize your data. A database stores its collections of data in *tables*, a concept explored further in Chapter 2.

The definition goes on to say that databases are usually associated with software that allows the data to be updated and queried. Real-life examples of database software include Microsoft's Access, Oracle's 10g, IBM's DB2, MySQL AB's MySQL, and Microsoft's SQL Server 2000. Often these programs are referred to as databases, but strictly speaking, they are database management systems (DBMS). A database is the *sets* (collections of related data) grouped into one entity. You could, for example, create an Access database, call it MyDatabase, include various data collections inside that one database, and manage the whole thing with the MS Access software.

Finally, the definition states that, as with the Access database example, a simple database might be just one file with many records with each record broken down into *fields*. But what are records and fields? A field is a single item of data about a specific thing. A thing could be a person, and a single item of data about a person could be their date of birth. Or the thing might be the address of a house and the specific item of data might be its street. Using the book example, the year a book was published is a specific piece of data that can be stored in a field. Another field might be the book's title; yet another could be the author's name. For this book, the fields would contain 2005 for the Year Published field, *Beginning SQL* for the Title field, and Paul Wilton and John Colby for the Author field. All these fields refer to one specific thing, a book called *Beginning SQL*. Collectively these fields are known as a *record*. Each book has its own record, and all the records are stored collectively in a database in something called a *table*. A single database can contain one or more tables. If all this information is a bit too much to absorb at once, don't worry: I'll be revisiting the concepts of fields and records later in this chapter.

By now, hopefully you get the idea that a database helps you store, organize, and retrieve data. One last thing to mention is the term *relational database,* which is a database containing data organized and linked (related) to each other. All records in a database are organized into tables. Related data, such as details of sales persons, are grouped in one table. You could put the details of cars they have sold in another table and then specify a relationship between which salesperson sold which cars — for example, salesperson X sold car Y on date Z. Figure 1-1 shows a table from the example database. On first glance, you may notice its resemblance to a spreadsheet with rows being your records and columns containing the fields for the records. In Chapter 3 you discover that you really need to think in terms of sets of data.

Most database management systems these days are relational, termed relational database management system (RDBMS). These systems make storing data and returning results easier and more efficient. They allow different questions to be posed of the database — even questions the original designer of the database didn't expect to be asked.



Data in Table 'Users' in 'CommunityDB' on '(local)'

UserId	Name	EmailAddress	Password	ReceiveNews	Country
1	Jane Jones	jane@mydomain.cc	shhhhhhhh	1	USA
2	John Doe	john@mailserver.oı	letmein	0	USA
3	Beci B Bandanas	beci@danas.co.uk	xyz123	0	UK
4	Katie Wilton	woof@dog.net	1234567890	1	UK
5	Billy	the@kid.net	098765ss	0	Canada
6	Gandalf	gandalf@wizards.n	iewrsdfsd	1	Middle Earth
7	Lucy Atall	lucy@sunny.com	letmepass	0	Canada

Figure 1-1

Why and When to Use a Database

When there are a huge number of alternative ways to store data, why should you trouble yourself creating a database? What advantages does a database hold?

The main advantage is fast and efficient data retrieval. A database helps you to organize your data in a logical manner. Database management systems are fine-tuned to rapidly retrieve the data you want in the way you want it. Databases also enable you to break data into specific parts. Retrieving data from a database is called *querying*. You'll often see the term *SQL query*, which briefly means any SQL code that extracts data from the database. This topic is covered in more depth later in this chapter.

Relational databases have the further advantage of allowing you to specify how different data relates to each other, as you saw in the car sales database example. If you store sales details and salesperson data in related databases, the question "How many cars has salesperson X sold in January?" becomes very easy to answer. If you just shoved all the information into a large text file, you'd find it one enormous task to question, or query, the data and find out specific answers.

Databases also allow you to set up rules that ensure that data remains consistent when you add, update, or delete data. Imagine that your imaginary car sales company has two salespeople named Julie Smith. You can set up a database to ensure that each salesperson has a unique ID, called a unique identifier (so that the Julies don't get mixed up); otherwise, telling who sold which cars would prove impossible. Other data storage systems, such as text files or spreadsheets, don't have these sorts of checks and quite happily allow you to store erroneous data. In later chapters you learn how to set up other rules to limit the risk of data becoming corrupted. For example, you might specify that an employee's social security number must be unique in the database. Or if a car is sold and it's listed as being sold by the employee with an ID of 123, you might add a check to see that full details of employee 123 are held in one of the database tables.

A properly set-up database minimizes data redundancy. Again using the car sales example, you can store all the details of a salesperson just once in the database and then use a unique ID to identify each salesperson. When you have other data that relates to a particular salesperson (for example, which cars they've sold), you can use the unique ID to search for the data. The unique ID is often a number that takes up less storage space than the person's full name.

Databases store raw data — just the facts, so to speak, and no intelligence. A car sales database might contain the make, model, and price of each car, but you wouldn't normally store the average number of cars sold in a month, because you can calculate that from the car sales information, the raw data.

A spreadsheet, however, may contain processed data, such as averages and statistical analysis. A database simply stores the data and generally leaves data processing to a *front-end* program, or the interface the user sees. Examples of front-end programs include a Web page that draws its data from a database or a program that hooks into the database's data and allows the user to view it.

Sharing data is also much easier using a database. You can share data among a number of users on the same computer or among users on different computers linked via a network or the Internet. If the example car sales company has branches in New York, Washington, and Boston, you could set up a computer containing a database in one location that is accessible by all of the offices via a network. This is not only possible but also safe because databases have a clearly defined structure and also enforce rules that protect the data contained. They also allow more than one person to access the database at the same time and change the data stored; the database management system handles simultaneous changes. Imagine the potential chaos if you used an Excel spreadsheet, and two salespeople change data simultaneously. You want to keep both sets of changes, but whoever saves the spreadsheet last is the person whose changes are stored, overwriting any earlier changes.

Databases also make sharing data between different systems much easier than using proprietary data formats — that is, a format specific to a particular program, manufacturer, or operating system. An Excel spreadsheet, for example, is easily read on a Windows machine with MS Office, but it is more of a challenge to read on a UNIX, Macintosh, or Linux machine because those computers handle data in a different way. Even on a Windows machine, you need to have MS Office installed. You can house a database on a central computer, put the database management system on there, and then enable access via a local network or the Internet.

As an alternative to databases, text files and spreadsheets have one big advantage, which is also their weakness: flexibility. Text files have no real rules. You can insert whatever text data you like wherever you like. To a large extent, spreadsheets are the same. You can ask users to add data in a predefined structure, but you have no real way to enforce such a request. Using databases limits user access to just the data and does not allow users to change the structure.

One final significant advantage of databases is security. Most database management systems allow you to create users in order to specify various levels of security. Before someone accesses the database, he or she must log on as a specific user. Each user has various rights and limits. Someone who maintains the database has full ability to edit data, change the database's structure, add and delete users, and so on. Other users may only have the ability to view data but not change it, or you may even want to limit what data they can view. Many database management systems provide a granular level of security, that is, they are very specific as to what a user can do. They are not just an all-or-nothing approach in which the user either has access or has no access.

Databases are used pretty much everywhere. Data processing played a big part in the development of computers, and even today it is one of their main roles. Nearly every walk of life or business requires a database somewhere along the way. Databases are commonly used on personal computers to store data used locally, and on company networks databases store and share company-wide information. The Internet has seen a big rise in databases used to share information; most online shops of a reasonable size use databases. When you visit online stores of any significant size, a database usually provides all the information on the goods being sold. Rather than every page being created by hand, large merchants use a template for book or CD details, and SQL retrieves the book information from the database. Imagine how much work it'd be if Amazon created every single page by hand!