

# SQL

## FUNDAMENTALS

### THIRD EDITION



- ▶ The hands-on SQL guide for every business and IT professional... no SQL experience needed
- ▶ Master SQL for the world's #1 enterprise and desktop databases: Oracle® and Microsoft® Access®
- ▶ Write accurate, efficient queries that are easy to verify, modify, and extend
- ▶ Create sorts, summaries, joins, cross-tabs, tables, and much more
- ▶ On the Web: extensive library of sample databases and code

**JOHN J. PATRICK**

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales  
(800) 382-3419  
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales  
international@pearsoned.com

Visit us on the Web: [informit.com/ph](http://informit.com/ph)

*Library of Congress Cataloging-in-Publication Data*

Patrick, John J.

SQL fundamentals / John J. Patrick. — 3rd ed.  
p. cm.

Includes indexes.

ISBN 978-0-13-712602-6 (pbk. : alk. paper) 1. SQL (Computer program language) 2. Oracle. 3. Microsoft Access.

I. Title.

QA76.73.S67P38 2008

005.75'65—dc22

2008024745

Copyright © 2009 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.  
Rights and Contracts Department  
501 Boylston Street, Suite 900  
Boston, MA 02116  
Fax: (617) 671-3447

ISBN-13: 978-0-13-712602-6

ISBN-10: 0-13-712602-6

Text printed in the United States on recycled paper at Courier in Stoughton, Massachusetts.

First printing, August 2008



## PREFACE

SQL is one of the most important computer languages. It is the language of databases. Whenever you search for the information you need in a large library of information, the code that performs the search is likely to be using SQL. Many applications in which you share information to coordinate with other people also use SQL.

It is used in more than 100 software products, and new ones are being added all the time. This book shows you how to get the most out of the databases you use. It explains how to use SQL to solve practical problems, using the most widely used SQL products, Oracle and Microsoft Access. Oracle and Access are both widely used, easily available, and run on personal computers. By learning these two products in detail, you will have all the basic skills to use any of the many products based on SQL.

## How the Topics Are Presented

---

This book uses an informal conversational style to take you on a tour of SQL topics. Oracle and Access are placed side by side doing the same tasks, so you can see their similarities and differences. Most topics are illustrated with an example of SQL code. I have intentionally kept the tables small in these examples, which makes them easy to check and understand.

Each example of SQL code begins by setting a task. Then the SQL code is given that performs that task. Whenever possible, I wrote the SQL code so that it works in both Oracle and Access. However, sometimes I could not do that, so I wrote one version of SQL code for Oracle and a different version for Access.

To make this book easier to read, each example of SQL shows the beginning and ending data table(s). This allows you to check that you understand what the SQL is doing. I have tried to make these examples small so they are easy to check.

Each example is often followed by notes to explain any subtle points about the SQL code or the data tables.

Finally, I give you a problem to solve to check your understanding of the topic. You can decide if you want to do these problems or not. Usually they are fairly easy and require only a small modification of the SQL code in the example. If you decide to do a problem, the Web site will allow you to determine if your solution is correct.

Each example of SQL code in this book is designed to be independent and stand on its own, without needing any changes performed in previous sections. This allows you to skip around in the book and read the sections in any order you want. Some people may want to read the book from beginning to end, but it is not necessary to do this.

Be sure to look at the appendices for practical tips on how to run Oracle and Access. The database files and the code for all the examples are available from the Web site. In several places throughout this book, I have expressed opinions about computer technology, something that many other technical books avoid doing. These opinions are my own and I take full responsibility for them. I also reserve the right to change my mind. If I do so, I will put my revised opinion, and the reasons that have caused me to change my thinking, on the Web site for this book.

## The Companion Web Site

---

The Web site for this book is Box.com, a file download service. The Web address is:

**`http://www.box.com/shared/ylbckg2fn0`**

Download zip file:

**`SQLFUN_3ed_All_Files.zip`**

The zip file contains:

- Oracle SQL code to build all the data tables used in this book.
- Access databases with all the data tables used in this book. Databases are available for several versions of Access.
- Ways to check your answers to problems in the book.
- A list of corrections, if there are any.

## Acknowledgments

---

Many people contributed greatly to this book. I would like to thank them for all the support they have given me during the time I was writing it. Their ideas and feedback have improved the quality of the material and the way I present it. In particular, I want to thank the following people for their suggestions and help with this third edition:

- Dejang Liu
- Alma Lynn Bane

People who helped with the previous editions include:

- Anila Manning, for much help in writing the second edition.
- Paul Reavis, who taught this course with me at UC Berkeley Extension.

- Todd Matson, who reviewed the Access material.
- Faysal Shaarani and Bill Allaway, who reviewed the Oracle material.
- Spencer Brucker and the UC Berkeley Extension, who have supported me in teaching the SQL Fundamentals course and developing the material in this book.
- All the folks at Prentice Hall, especially Bernard Goodwin, editor; Vanessa Moore, Moore Media, Inc., production editor; Michael Meehan and Jeffery Pepper, the original editors for this book; and the many other people with whom I never worked directly.
- Thanks especially to my mom, Jean Praninskas, and to my son, Richard Watts, who also reviewed this book.

Thanks also to Brian Akitoye, Mehran Ansari, Asa Ashraf, Anne Bester, Sandra Bush, Connie Chong, Patricia Cleveland, Robert D'Antony, Gan Davnarrain, Bruce Douglass, James Drummond, Ron Duckworth, Dean Evans, Steve Fajardo, Earl Gardner, Wolday Gebremichael, Neelam Hasni, Reda Ismail, Marques Junior, John Karsnak, Allyson Kinney, Gladys Lattier, Brian Lester, Mahen Luximan, Alex McDougall, E. Muljadi, Satyendra Narayan, Bade Oyebamiji, Stefan Pantazi, Todd Perchert, Oxana Rakova, Jacob Relles, Ricardo Ribeiro, Cindy Roberts, John Rusk, Ty Seward, Gary Shapiro, David Smith, Kenneth Smith, Joan Spasyk, Patricia Warfel, and William White.

# chapter 1

## STORING INFORMATION IN TABLES

In relational databases, all the data is stored in tables and all the results are expressed in tables. In this chapter, we examine tables in detail.

<b>Introduction</b> .....	<b>3</b>
<b>1-1</b> What is SQL? .....	3
<b>1-2</b> What is a relational database and why would you use one? .....	4
<b>1-3</b> Why learn SQL? .....	6
<b>1-4</b> What is in this book? .....	8
<b>The Parts of a Table</b> .....	<b>9</b>
<b>1-5</b> Data is stored in tables .....	10
<b>1-6</b> A row represents an object and the information about it .....	11
<b>1-7</b> A column represents one type of information. ....	12
<b>1-8</b> A cell is the smallest part of a table. ....	14
<b>1-9</b> Each cell should express just one thing .....	15
<b>1-10</b> Primary key columns identify each row .....	16
<b>1-11</b> Most tables are tall and thin. ....	18
<b>Examples of Tables</b> .....	<b>19</b>
<b>1-12</b> An example of a table in Oracle and Access .....	19
<b>1-13</b> Some design decisions in the l_employees table .....	22
<b>1-14</b> The Lunches database .....	23
<b>Key Points</b> .....	<b>30</b>



## 1-1 What is SQL?

The name SQL stands for Structured Query Language. It is pronounced “S-Q-L” and can also be pronounced “sequel.”

SQL is a computer language designed to get information from data that is stored in a relational database. In a moment, I discuss what a relational database is. For now, you can think of it as one method of organizing a large amount of data on a computer. SQL allows you to find the information you want from a vast collection of data. The purpose of this book is to show you how to get the information you want from a database.

SQL is different from most other computer languages. With SQL, you describe the type of information you want. The computer then determines the best procedure to use to obtain it and runs that procedure. This is called a **declarative** computer language because the focus is on the result: You specify what the result should look like. The computer is allowed to use any method of processing as long as it obtains the correct result.

Most other computer languages are **procedural**. These are languages like C, Cobol, Java, Assembler, Fortran, Visual Basic, and others. In these languages, you describe the procedure that will be applied to the data; you do not describe the result. The result is whatever emerges from applying the procedure to the data.

Let me use an analogy to compare these two approaches. Suppose I go to a coffee shop in the morning. With the declarative approach, used by SQL, I can say **what** I want: “I would like a cup of coffee and a donut.” With the procedural approach, I cannot say that. I have to say **how** the result can be obtained and give a specific procedure for it. That is, I have to say how to make a cup of coffee and how to make a donut. So, for the coffee, I have to say, “Grind up some roasted coffee beans, add boiling water to them, allow the coffee to brew, pour it into a cup, and give it to me.” For the donut, I will have to read from a cookbook. Clearly, the declarative approach is much closer to the way we usually speak and it is much easier for most people to use.

The fact that SQL is easy to use, relative to most other computer languages, is the main reason it is so popular and important. The claim is often made that anyone can learn SQL in a day or two. I think that claim is more a wish than a reality. After all, SQL is a computer language, and computers are not as easy to use as telephones — at least not yet.

Nonetheless, SQL is easy to use. With one day of training, most people can learn to obtain much useful information. That includes people who are not programmers. People throughout an organization, from secretaries to vice presidents, can use SQL to obtain the information they need to make business decisions. That is the hope and, to a large extent, it has been proven true.

Information is not powerful by itself. It only becomes powerful when it is available to people throughout an organization when they need to use it. SQL is a tool for delivering that information.

### Notes about SQL

- SQL is the designated language for getting information from a relational database.
- SQL says *what* information to get, rather than *how* to get it.
- Basic SQL is easy to learn.
- SQL empowers people by giving them control over information.
- SQL allows people to handle information in new ways.
- SQL makes information powerful by bringing it to people when they need it.

## 1-2 What is a relational database and why would you use one?

A *relational database* is one way to organize data in a computer. There are other ways to organize it, but in this book, we do not discuss these other ways, except to say that each method has some strengths and some drawbacks. For now, we look only at the advantages a relational database has to offer.

SQL is one of the main reasons to organize data into a relational database. Using SQL, information can be obtained from the data fairly easily by people throughout the organization. That is very important.

Another reason is that data in a relational database can be used by many people at the same time. Sometimes hundreds or thousands of people can all share the data in a database. All the people can see the data and change the data (if they have the authority to do so). From a business perspective, this provides a way to coordinate all the employees and have everybody working from the same body of information.

A third reason is that a relational database is designed with the expectation that your information requirements may change over time. You might need to reorganize the information you have or add new pieces of information to it. Relational databases are designed to make this type of change easy. Most other computer systems are difficult to change. They assume that you know what all the requirements will be before you start to construct them. My experience is that people are not very good at predicting the future, even when they say they can, but here I am showing my own bias toward relational databases.

From the perspective of a computer programmer, the flexibility of a relational database and the availability of SQL make it possible to develop new computer applications much more rapidly than with traditional techniques. Some organizations take advantage of this; others do not.

The idea of a relational database was first developed in the early 1970s to handle very large amounts of data — millions of records. At first, the relational database was thought of as a back-end processor that would provide information to a computer application written in a procedural language such as C or Cobol. Even now, relational databases bear some of the traits of that heritage.

Today, however, the ideas have been so successful that entire information systems are often constructed as relational databases, without much need for procedural code (except to support input forms). That is, the ideas that were originally developed to play a supporting role for procedural code have now taken center stage. Much of the procedural code is no longer needed.

In relational databases, all the data is kept in tables, which are two-dimensional structures with columns and rows. I describe tables in detail later in this chapter. After you work with them for a while, you will find that tables provide a very useful structure for handling data. They adapt easily to changes, they share data with all users at the same time, and SQL can be run on the data in a table. Many people start thinking of their data in terms of tables. Tables have become the metaphor of choice when working with data.

Today, people use small personal databases to keep their address books, catalog their music, organize their libraries, or track their finances. Business applications are also built as relational databases. Many people prefer to have their data in a database, even if it has only a few records in it.

## The beginning of relational databases

- Relational databases were originally developed in the 1970s to organize large amounts of information in a consistent and coherent manner.
- They allowed thousands of people to work with the same information at the same time.
- They kept the information current and consistent at all times.
- They made information easily available to people at all levels of an organization, from secretaries to vice presidents. They used SQL, forms, standardized reports, and ad-hoc reports to deliver information to people in a timely manner.
- They were designed to work as an information server back end. This means that most people would not work directly with the database; instead, they would work with another layer of software. This other software would get the information from the database and then adapt it to the needs of each person.
- They empowered people by making current information available to them when they needed to use it.

## Today — How relational databases have changed

- In addition to the large databases described already, now there are also many smaller databases that handle much smaller amounts of information. These databases can be used by a single person or shared by a few people.
- Databases have been so successful and are so easy to use that they are now employed for a wider range of applications than they were originally designed for.
- Many people now work directly with a database instead of through another layer of software.
- Many people prefer to keep their data in databases. They feel that relational databases provide a useful and efficient framework for handling all types of data.

### 1-3 Why learn SQL?

SQL is used in more than 100 software products. Once you learn SQL, you will be able to use all of these products. Of course, each one will require a little study of its special features, but you will soon feel at home with it and know how to use it. You can use this one set of skills over and over again.

<b>Major SQL Products</b>	<b>Other SQL Products (and Products Based on SQL)</b>
<b>Oracle</b>	4th Dimension
<b>Microsoft SQL Server</b>	SQLBase
<b>Microsoft Access</b>	CSQL
<b>MySQL</b>	FileMaker PRO
<b>DB2 (IBM Data Server)</b>	Helix Database
<b>Informix</b>	ODBC
<b>PostgreSQL</b>	Ingres
<b>Sybase</b>	MonetDB
<b>Microsoft Visual FoxPro</b>	H2
<b>NonStop SQL</b>	MaxDB
<b>Dataphor</b>	VMDS
<b>Teradata</b>	TimesTen
	Openbase
	eXtremeDB
	Interbase
	OpenEdge ABL
	SmallSQL
	Lintor SQL DMBS
	Derby
	Adabas D
	Greenplum Database
	HSQldb
	Alpha_Five
	One\$DB
	ScimoreDB
	Pervasive PSQL
	Gladius DB
	Daffodil database
	solidDB
	(and many more)

There are reasons SQL is used so much. One reason is that it is easy to learn, relative to many other computer languages. Another reason is that it opens the door to relational databases and the many advantages they offer. Some people say that SQL is the best feature of relational databases and it is what makes them successful. Other people say that relational databases make SQL successful. Most people agree that together they are a winning team.

SQL is the most successful declarative computer language — a language with which you say what you want rather than how to get it. There are some other declarative languages and report-generation tools, but most of them are much more limited in what they can do. SQL is more powerful and can be applied in more situations.

SQL can help you get information from a database that may not be available to people who do not know SQL. It can help you learn and understand the many products that are based on it.

Finally (don't tell your boss), learning SQL can be enjoyable and fun. It can stretch your mind and give you new tools with which to think. You might start to view some things from a new perspective.

## **1-4 What is in this book?**

### **The subject of this book**

This book shows you how to use SQL to get information from a relational database. It begins with simple queries that retrieve selected data from a single table. It progresses step by step to advanced queries that summarize the data, combine it with data from other tables, or display the data in specialized ways. It goes beyond the basics and shows you how to get the information you need from the databases you have.

### **Who should read this book?**

Anyone with an interest in getting information from a database can read this book. It can be a first book about databases for people who are new to the subject. You do not need to be a computer programmer. The discussion begins at the beginning and it does not assume any prior knowledge about databases. The only thing you need is the persistence to work through the examples and a little prior experience working with your own computer.

Professional programmers can also use this book. The techniques shown here can help them find solutions to many problems. Whether you are a novice or a professional, an end user or a manager, the SQL skills you learn will be useful to you over and over again.

## Organization of this book

This book discusses the practical realities of getting information from a database. A series of specific tasks are accomplished and discussed. Each concept is presented with an example.

The tasks are designed and arranged to show the most important aspects of the subject. Each topic is discussed thoroughly and in an organized manner. All the major features and surprising aspects of each topic are shown.

## Why compare two different implementations of SQL — Oracle and Access?

If a book discusses only the theory of SQL, and no particular product that implements it, the reader will be left with no practical skills. He or she will be able to think about the concepts, but might have difficulty writing code that works.

If a book discusses only one implementation of SQL, it is easy to get distracted by the quirks and special features it has. You also lose sight of the fact that SQL is used in many products, although in slightly different ways.

This book compares Oracle and Access because they are two of the most widely used SQL products and because they both run on a PC. They are somewhat different. You will see them side by side. Oracle is used mostly for larger business applications. Access is used mostly for personal database applications and smaller business applications.

## The Parts of a Table

---

SQL always deals with data that is in tables. You probably understand tables already on an informal level. The tables used in a relational database have a few unusual features. Because computers need precise definitions, the description of a table must be formalized. In this section, I define what a table is and what its parts are.

## 1-5 Data is stored in tables

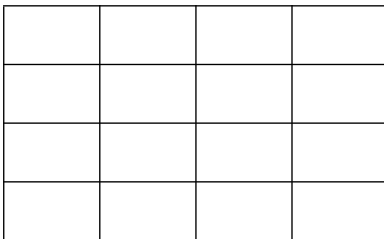
In a relational database, all the data is stored in tables. A table is a two-dimensional structure that has **columns** and **rows**. Using more traditional computer terminology, the columns are called **fields** and the rows are called **records**. You can use either terminology.

Most people are familiar with seeing information in tables. Bus schedules are usually presented in tables. Newspapers use tables to list stock values. We all know how to use these tables. They are a good way to present a lot of information in a very condensed format. The tables in a relational database are very similar to these tables, which we all understand and use every day.

All the information in a relational database is kept in tables. There is no other type of container to keep it in — there are no other data structures. Even the most complex information is stored in tables. Someone once said that there are three types of data structures in a relational database: tables, tables, and tables. In a relational database, we have nothing but tables; there are no numbers, no words, no letters, and no dates unless they are stored in a table.

You might think that this restricts what a relational database can do and the data it can represent. Is it a limitation? The answer is no. All data is capable of being represented in this format. Sometimes you have to do some work to put it in this format. It doesn't always just fall into this format by itself. But you can always succeed at putting data into tables, no matter how complex the data is. This has been proven in mathematics. The proof is long and complex and I do not show it to you here, but you can have confidence that tables are versatile enough to handle all types of data.

The following two depictions show a basic table structure and how a table might store information.




**A conceptual diagram of a table.**



First Name	Last Name	Age	Gender	Favorite Game
Nancy	Jones	1	F	Peek-a-boo
Paula	Jacobs	5	F	Acting
Deborah	Kahn	4	F	Dolls
Howard	Green	7	M	Baseball
Jack	Lee	5	M	Trucks
Cathy	Rider	6	F	Monsters

**An example of a table that stores information about children.**

Each row contains information about one child. Each column contains one type of information for all the children. As always, this table contains only a limited amount of information about each child. It does not say, for instance, how much each child weighs.

## Notes

- In a relational database, all the data is stored in tables.
- A table has two dimensions called columns and rows.
- Tables can hold even the most complex information.
- All operations begin with tables and end with tables. All the data is represented in tables.

### **1-6 A row represents an object and the information about it**

Each row of a table represents one object, event, or relationship. I call them all objects for now, so I do not have to keep repeating the phrase “object, event, or relationship.”

All the rows within a table represent the same type of object. If you have 100 doctors in a hospital, you might keep all the information about them in a single table. If you also want to keep information about 1,000 patients who are in the hospital, you would use a separate table for that information.

The tables in a relational database may contain hundreds or thousands of rows. Some tables even contain many millions of rows. In theory, there is no limit to the number of rows a table can have. In practice, your computer will limit the number of rows you can have. Today, business databases running on large computers sometimes reach billions of rows.

There are also some tables with only one row of data. You can even have an empty table with no rows of data in it. This is something like an empty box. Usually, a table is only empty when you first build it. After it is created, you start to put rows of data into it.

In a relational database, the rows of a table are considered to be in no particular order so they are an unordered set. This is different from the tables most people are familiar with. In a bus schedule, the rows are in a definite and logical order. They are not scrambled in a random order.

Database administrators (DBAs) are allowed to change the order of the rows in a table to make the computer more efficient. In some products, such as Access, this can be done automatically by the computer. As a result, you, the end user seeking information, cannot count on the rows being in a particular order.



**A conceptual diagram of a row.**

## Notes

- A row contains data for one object, event, or relationship.
- All the rows in a table contain data for similar objects, events, or relationships.
- A table may contain hundreds or thousands of rows.
- The rows of a table are not in a predictable order.

### **1-7 A column represents one type of information**

A column contains one particular type of information that is kept about all the rows in the table. A column cannot, or should not, contain one type of information for one row and another type for another row. Each column usually contains a separate type of information.

Each column has a name, for instance “favorite game,” and a datatype. We discuss datatypes in chapter 6, but for now let’s keep it simple. There are three main datatypes: text, numbers, and dates. This means that there are three types of columns: columns containing text, columns containing numbers, and columns containing dates.

Some columns allow nulls, which are unknown values. Other columns do not allow them. If a column does not allow nulls, then data is required in the column for every row of the table. This means it is a required field. When a column does allow nulls, the field is optional.

Most tables contain 5 to 40 columns. A table can contain more columns, 250 or more, depending on the relational database product you are using, but this is unusual.

Each column has a position within the table. That is, the columns are an ordered set. This contrasts with the rows, which have no fixed order.

Information about the columns — their names, datatypes, positions, and whether they accept nulls — is all considered to be part of the definition of the table itself. In contrast, information about the rows is considered to be part of the data and not part of the definition of the table.



**A conceptual diagram of a column.**

## Notes

- A column contains one type of data about each row of the table.
- Each column has a name.
- Each column has a datatype. The most important datatypes are:
  - Text
  - Numbers
  - Dates with times

- Some columns accept nulls, and others do not. A null is an unknown value.
- Each column has a position within the table. In contrast to rows, the columns of a table form an ordered set. There is a first column and a last column.
- Most tables have 40 columns or fewer.

## 1-8 A cell is the smallest part of a table

A *cell* occurs where one row meets with one column. It is the smallest part of a table and it cannot be broken down into smaller parts.

A cell contains one single piece of data, a single unit of information. At least that is the way it is in theory, and this is how you should begin to think about it. In practice, sometimes a cell can contain several pieces of information. In some applications a cell can contain an entire sentence, a paragraph, or an entire document with hundreds of pages. For now we will consider that a cell can contain one of the following:

- One word
- One letter
- One number
- One date, which includes the time
- A null, which indicates that there is no data in the cell

For the first few chapters of this book, we consider the information in a cell to be *atomic*, which means that it is a single indivisible unit of information. We gather and arrange information from a table by manipulating its cells. We either use all the information within a cell or we do not use that cell at all. Later, when we discuss row functions, you will see how to use only part of the data from a cell.

A column is a collection of cells. These cells have the same datatype and represent the same type of information. A row is a collection of cells. Together, they represent information about the same object, event, or relationship.



**A conceptual diagram of a cell.**

## Notes

- A cell contains a single piece of data, a single unit of information.
- Usually a cell contains one of the following types of data:
  - Text, sometimes one word, or sometimes a one-letter code, such as M for male or F for female
  - A number
  - A date and time
  - A null, which is an unknown value (some people call this an empty cell, or missing data)
- All the cells in a column contain the same type of information.
- All the cells in a row contain data about the same object, event, or relationship.

### **1-9 Each cell should express just one thing**

Each cell expresses just one thing — one piece of information. That is the intent of the theory of relational databases. In practice, it is not always clear what this means. The problem, partly, is that English and other spoken languages do not always express information clearly. Another part of the problem is that information does not always come in separate units.

Let's examine one case in detail. A person in America usually has two names — a first name and a last name. Now that is a bit of a problem to me when I want to put information in the computer. There is one person, but there are two names. How should I identify the person? Should I put both names together in one cell? Should I put the names into two separate cells? The answer is not clear.

Both methods are valid. The designers of the database usually decide questions like this. If the database designers think that both names will always be used together, they will usually put both names in a single cell. But if they think that the names will be used separately, they will put each name in a separate cell.

The problem with this is that the way a database is used may change over time, so even if a decision is correct when it is made, it might become incorrect later on.

<b>Full Name</b>	<b>First Name      Last Name</b>		
Susan Riley	<table style="border-collapse: collapse; width: 100%;"> <tr> <td style="border: 1px solid black; padding: 5px;">Susan</td> <td style="border: 1px solid black; padding: 5px;">Riley</td> </tr> </table>	Susan	Riley
Susan	Riley		
(A)	(B)		

**Two ways to show the name of a person in a table. (A) One column for the name. Both the first and last names are put in a single cell. (B) Two separate columns: one for the first name and another for the last name. Each cell contains a single word.**

## Notes

- Both methods are equally valid.
- The first method emphasizes that Susan Riley is one person, even though the English language uses two separate words to express her name. It implies that we will usually call her “Susan Riley,” using both her names together as a single unit.
- The second method emphasizes the English words. It implies that we will want to use several different variations of her name, calling her “Susan” or “Susan Riley” or “Miss Riley.” The words “Susan” or “Riley” can come from the table in the database. Any other words must be supplied by some other means.
- The database design intends each cell to be used in whole or not used at all. In theory, you should not need to subdivide the data in a cell. However, in practice that is sometimes required.

## 1-10 Primary key columns identify each row

Most tables contain a **primary key** that identifies each row in the table and gives it a name. Each row must have its own identity, so no two rows are allowed to have the same primary key.

The primary key consists of several columns of the table. By convention, these are usually the first few columns. The primary key may be one column or more than one. We say that there is only one primary key, even when it consists of several columns, so it is the collection of these columns, taken as a single unit, that is the primary key and serves to identify each row.

The primary key is like a noun because it names the object of each row. The other columns are like adjectives because they give additional information about the object.

A table can only contain a single primary key, even if it consists of several columns. This makes sense because there is no point in identifying a row twice — those identities could conflict with each other. Suppose, for example, that we have a table of employees. Each employee can be identified by an employee number or a Social Security number. The database designers would need to choose which column to make the primary key of the table. They could choose either one to be the primary key of the table, or they could choose to use both together to make a primary key. However, they are not allowed to say that each column by itself is a primary key.

The name of a column is considered to be part of the definition of the table. In contrast, the name of a row, which is the primary key of the row, is considered to be part of the data in the table.

There are two rules that regulate the columns of the primary key of a table:

1. None of the columns of the primary key can contain a null. This makes sense because a null is an unknown value. Therefore, a null in any part of the primary key would mean we do not know the identity of the object or the row. In databases, we do not want to enter information about unidentified rows.
2. Each row must have an identity that is different from every other row in the table. That is, no two rows can have the same identity — the same values in all the columns of the primary key. For any two rows of the table, there must be at least one column of the primary key where the values are different.

### Primary Key

Primary Key			
A			
B			
C			
D			

**The first column is usually the primary key of the table.**

### Primary Key

Primary Key			
A	1		
A	2		
B	1		
B	2		

Sometimes the primary key is the first several columns of the table.

### Notes

- Most tables have primary keys.
- Usually, the primary key consists of the first column or the first several columns of the table.
- The primary key names the object, event, or relationship the row represents. In grammatical terms, it is a noun because it is the subject of all the information in the row.
- The other columns of the table make statements about the primary key. In grammatical terms, they are adjectives or adverbs that describe the object named by the primary key and give additional information about it.

## 1-11 Most tables are tall and thin

Many books on SQL give the impression that tables are usually square — that they have about the same number of rows as they have columns. This false impression is left because the tables in most SQL books are approximately square. In any book, the tables must be kept small. In a book, when you run SQL code you must be able to examine the results in full detail.

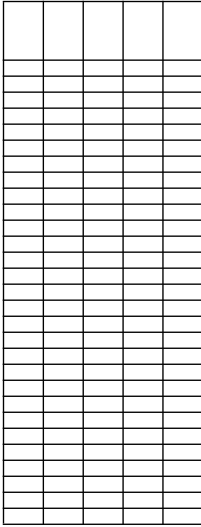
However, the tables that are used in real production systems usually have a different shape. They are tall and thin. They may have 30 columns, but 1,000,000 rows.

Not all tables have this shape, but most do. Some tables have only one row.

I tell you this because I like to visualize the data and the tables I am working with. If you like to visualize them, too, then at least I have provided you



with the correct picture. If you are not inclined to visualize these things, do not worry about it. Just go on to the next page.




**Most tables have many more rows than columns.**

## Examples of Tables

---

Up to now, we have discussed the theory of tables, but you have not seen any real ones. In the following sections you will see some actual tables. We look at a table to see how it looks in both Oracle and Access. We discuss some of the design decisions that are used in constructing many tables. We also examine the tables of the `Lunches` database, which is used in many of the examples throughout this book.

### **1-12 An example of a table in Oracle and Access**

This section shows the same table in both Oracle and Access. This is our first opportunity to examine how Oracle and Access compare.

You will have to decide for yourself how similar they are and how different they are. To me, this example shows that they are about 90 percent similar and about 10 percent different. Of course, this is just one example. You might ask yourself which percentages you would use to describe this.

Oracle tables can be shown in two formats that are very similar, but have a few slight differences. To keep things simple here, I am only showing you one of those formats. The following Oracle table was obtained using the “SQL Command Line” environment. The other Oracle format occurs in the “Database Home Page” environment. I will discuss it briefly in the notes at the end of this section.

### 1\_employees table: Oracle format

EMPLOYEE ID	FIRST_NAME	LAST_NAME	DEPT CODE	HIRE_DATE	CREDIT LIMIT	PHONE NUMBER	MANAGER ID
201	SUSAN	BROWN	EXE	01-JUN-1998	\$30.00	3484	(null)
202	JIM	KERN	SAL	16-AUG-1999	\$25.00	8722	201
203	MARTHA	WOODS	SHP	02-FEB-2009	\$25.00	7591	201
204	ELLEN	OWENS	SAL	01-JUL-2008	\$15.00	6830	202
205	HENRY	PERKINS	SAL	01-MAR-2006	\$25.00	5286	202
206	CAROL	ROSE	ACT	(null)	(null)	(null)	(null)
207	DAN	SMITH	SHP	01-DEC-2008	\$25.00	2259	203
208	FRED	CAMPBELL	SHP	01-APR-2008	\$25.00	1752	203
209	PAULA	JACOBS	MKT	17-MAR-1999	\$15.00	3357	201
210	NANCY	HOFFMAN	SAL	16-FEB-2007	\$25.00	2974	203

### 1\_employees table: Access format

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPT_CODE	HIRE_DATE	CREDIT_LIMIT	PHONE_NUMBER	MANAGER_ID	Add New Field
201	Susan	Brown	Exe	01-Jun-1998	\$30.00	3484		
202	Jim	Kern	Sal	16-Aug-1999	\$25.00	8722	201	
203	Martha	Woods	Shp	02-Feb-2009	\$25.00	7591	201	
204	Ellen	Owens	Sal	01-Jul-2008	\$15.00	6830	202	
205	Henry	Perkins	Sal	01-Mar-2006	\$25.00	5286	202	
206	Carol	Rose	Act					
207	Dan	Smith	Shp	01-Dec-2008	\$25.00	2259	203	
208	Fred	Campbell	Shp	01-Apr-2008	\$25.00	1752	203	
209	Paula	Jacobs	Mkt	17-Mar-1999	\$15.00	3357	201	
210	Nancy	Hoffman	Sal	16-Feb-2007	\$25.00	2974	203	

### Similarities between Oracle and Access

- Column names are printed at the top of the column. The column names are part of the structure of the table, not part of the data in the table.
- Sometimes the column names shown in the column headings are truncated. This is a slight problem. You are given tools to deal with it.
- Columns containing text data are justified to the left.
- Columns containing numbers are justified to the right.

- Columns containing dates often display only the date. The format for displaying the date is not part of the data. The value of the date is stored in the table, but the format of the date is specified separately. The date actually contains both a date and a time, but the time is often not displayed.
- Columns displaying currency amounts are actually stored as numbers, and use a format to put in the dollar signs and decimal points.

## Differences between Oracle and Access

- **Display framework:** Oracle displays lines of character data. Access uses graphical techniques to display the data in a grid and color the borders of the grid.
- **Case:** The Oracle table is shown all in uppercase. The Access table uses uppercase only for the first letter. It is a common convention to set the databases up this way. Mixed-case data can be put into an Oracle table, but this makes the data more difficult to handle, so Oracle data is usually either all uppercase or all lowercase. Access data is handled as if it were all uppercase, although it is displayed in mixed case. This makes it look nicer, but sometimes it can also be deceiving. In Access, the data appears to be mixed case, but the data behaves as if it were in uppercase. For instance, `JOhn` and `jOhn` appear different in Access, but they are handled as if they are the same.
- **Column headings:** Oracle can use several lines for a column heading. Access displays the heading on a single line.
- **Date formats:** The dates above show Oracle and Access using the same date format. I made that happen here because I wanted Oracle and Access to look similar. However, on your computer the dates will probably use different formats.

Oracle and Access can both display dates in a variety of formats. Each has a default format to use for dates when no other format is specified. However, Oracle uses one method to specify this default format for dates and Access uses a different method.

- **Date alignment:** Oracle aligns dates to the left, whereas Access aligns them to the right.
- **Nulls:** In this book, I have set up Oracle to always display nulls as (`null`) in all the columns of every table. This cannot easily be done in Access.

- **Position pointer:** The Access table contains a record selector and a pointer to a particular field within that record, which allows you to modify the data. The Oracle table does not contain these.
- **Ability to add data:** In Access, a blank row at the bottom of a table indicates that new rows of data can be entered into the table. Also an extra column is displayed called “Add New Field”. This is not done in Oracle.

## Notes

The other Oracle format is used in the “Database Home Page” environment. It has several technical differences, but none that will challenge your understanding of what is going on. Here are a few of these differences:

- Tables are displayed on pages in your Web browser.
- Column headings are never truncated.
- All fields are justified to the left.
- Nulls are shown with dashes
- Dollar amounts are not automatically formatted.

## 1-13 Some design decisions in the 1\_employees table

The 1\_employees table contains some design decisions that I want to point out to you because they reflect some common practices within relational databases. Like all design decisions, they could have been made in other ways. This is not the only way to design the table. It might not even be the best way. But you may often encounter these design decisions and you need to be aware of them.

### 1\_employees table

EMPLOYEE			DEPT			CREDIT	PHONE	MANAGER
ID	FIRST_NAME	LAST_NAME	CODE	HIRE_DATE	LIMIT	NUMBER	ID	
201	SUSAN	BROWN	EXE	01-JUN-1998	\$30.00	3484	(null)	
202	JIM	KERN	SAL	16-AUG-1999	\$25.00	8722	201	
203	MARTHA	WOODS	SHP	02-FEB-2009	\$25.00	7591	201	
204	ELLEN	OWENS	SAL	01-JUL-2008	\$15.00	6830	202	
205	HENRY	PERKINS	SAL	01-MAR-2006	\$25.00	5286	202	
206	CAROL	ROSE	ACT	(null)	(null)	(null)	(null)	
207	DAN	SMITH	SHP	01-DEC-2008	\$25.00	2259	203	
208	FRED	CAMPBELL	SHP	01-APR-2008	\$25.00	1752	203	
209	PAULA	JACOBS	MKT	17-MAR-1999	\$15.00	3357	201	
210	NANCY	HOFFMAN	SAL	16-FEB-2007	\$25.00	2974	203	

## Design decisions to be aware of

- The `phone_number` column contains text data, not numbers. Although the data look like numbers, and the column name says number, it actually has a text datatype. You can tell this by its alignment, which is to the left. The reason the table is set up this way is that the phone number data will never be used for arithmetic. You never add two phone numbers together or multiply them. You only use them the way they are, as a text field. So this table stores them as text.
- The `employee_id` column contains numbers. You can tell this by its alignment, which is to the right. Now, we do not do arithmetic with employee IDs, we never add them together, so why isn't this a text field, too? The answer is that numbers are often used for primary key columns even when no arithmetic will be performed on them. This can allow the computer to handle the table more quickly.
- The `manager_id` column contains numbers, but it is not a primary key column. So why doesn't it contain text? This column is intended to match with the `employee_id` column, so it has been given the same datatype as that column. This improves the speed of matching the two columns.
- The name of the table, `l_employees`, might seem strange. The `l` indicates that this table is part of a group of tables. The names of all the tables in the group start with the same letter(s). In this case it shows that the table is part of the `Lunches` database. (Here I use the term *database* to mean a collection of related tables.)
- The people who design databases put a considerable amount of work into the consistent naming of objects, using standard prefixes, suffixes, abbreviations, and column names. This makes the whole model easier to understand and more usable for the code that is developed for each database.

## 1-14 The `Lunches` database

Most of the examples of SQL code in this book are based on the `Lunches` database. You can get a complete listing of this database from the Web site. To read this book, you will need to understand the story and the data, so here is the basic story.

There is a small company with ten employees. This company will serve lunch to its employees on three occasions. Each employee can attend as many of these lunches as his or her schedule permits. When employees register to attend a lunch, they get to pick what they want to eat. They may choose from among the ten foods available to them. They can decide to have a single portion or a double portion of any of these foods. The Lunches database keeps track of all this information.

That is the story. Now let's look at the data. When I call this a database, I mean that it is a collection of related tables. The set of tables, taken together, tell the story. There are seven tables in this database:

- Employees (1\_employees)
- Departments (1\_departments)
- Constants (1\_constants)
- Lunches (1\_lunches)
- Foods (1\_foods)
- Suppliers (1\_suppliers)
- Lunch Items (1\_lunch\_items)

To show that these tables are all related to each other and to distinguish them from other tables we may use, the names of these tables are all prefixed with the letter 1. When there are multiple words, such as lunch\_items, the spaces are replaced with underscore characters. This helps the computer understand that the two words together are a single name.

### 1\_employees table

EMPLOYEE ID	FIRST_NAME	LAST_NAME	DEPT CODE	HIRE_DATE	CREDIT LIMIT	PHONE NUMBER	MANAGER ID
201	SUSAN	BROWN	EXE	01-JUN-1998	\$30.00	3484	(null)
202	JIM	KERN	SAL	16-AUG-1999	\$25.00	8722	201
203	MARTHA	WOODS	SHP	02-FEB-2009	\$25.00	7591	201
204	ELLEN	OWENS	SAL	01-JUL-2008	\$15.00	6830	202
205	HENRY	PERKINS	SAL	01-MAR-2006	\$25.00	5286	202
206	CAROL	ROSE	ACT	(null)	(null)	(null)	(null)
207	DAN	SMITH	SHP	01-DEC-2008	\$25.00	2259	203
208	FRED	CAMPBELL	SHP	01-APR-2008	\$25.00	1752	203
209	PAULA	JACOBS	MKT	17-MAR-1999	\$15.00	3357	201
210	NANCY	HOFFMAN	SAL	16-FEB-2007	\$25.00	2974	203

The `l_employees` table lists all the employees. Each employee can be identified by an employee ID, which is a number assigned to him or her. This allows the company to hire two people with the same name. The primary key is the `employee_id` column.

Each employee has a manager, who is also an employee of the company. The manager is identified by his or her employee ID. For instance, the `manager_id` column shows that Jim Kern is managed by employee 201. Employee 201 is Susan Brown.

Susan Brown and Carol Rose are the only employees without a manager. You can tell this because there is a null in the `manager_id` columns. However, these nulls mean different things.

Susan Brown is the head of the company. The null in this case does not mean that we do not know who her manager is. Rather, it means that she does not have a manager.

Carol Rose is a new hire. The null in her `manager_id` column could mean that she has not yet been assigned to a manager or it could mean that the information has not yet been entered into the database.

### **l\_departments table**

DEPT CODE	DEPARTMENT_NAME
----	-----
ACT	ACCOUNTING
EXE	EXECUTIVE
MKT	MARKETING
PER	PERSONNEL
SAL	SALES
SHP	SHIPPING

Each employee works for one department. The department code is shown in the `l_employees` table. The full name of each department is shown in the `l_departments` table. The primary key of this table is `dept_code`.

These tables can be linked together by matching the `dept_code` columns. For example, the `l_employees` table shows us that employee 202, Jim Kern, has a department code of `SAL`. The `l_departments` table says that the sales department uses the department code `SAL`. This tells us that Jim Kern works in the sales department.

**1\_constants table**

BUSINESS_NAME	BUSINESS START_DATE	LUNCH_BUDGET	OWNER_NAME
CITYWIDE UNIFORMS	01-JUN-1998	\$200.00	SUSAN BROWN

The `1_constants` table contains some constant values and has only one row. We use these values with the other tables of the database. These values are expected to change infrequently, if at all. Storing them in a separate table keeps the SQL code flexible by providing an alternative to hard-coding these values into SQL. Because the table of constants has only one row, it does not need a primary key.

**1\_lunches table**

LUNCH_ID	LUNCH_DATE	EMPLOYEE_ID	DATE_ENTERE
1	16-NOV-2011	201	13-OCT-2011
2	16-NOV-2011	207	13-OCT-2011
3	16-NOV-2011	203	13-OCT-2011
4	16-NOV-2011	204	13-OCT-2011
6	16-NOV-2011	202	13-OCT-2011
7	16-NOV-2011	210	13-OCT-2011
8	25-NOV-2011	201	14-OCT-2011
9	25-NOV-2011	208	14-OCT-2011
12	25-NOV-2011	204	14-OCT-2011
13	25-NOV-2011	207	18-OCT-2011
15	25-NOV-2011	205	21-OCT-2011
16	05-DEC-2011	201	21-OCT-2011
17	05-DEC-2011	210	21-OCT-2011
20	05-DEC-2011	205	24-OCT-2011
21	05-DEC-2011	203	24-OCT-2011
22	05-DEC-2011	208	24-OCT-2011

The `1_lunches` table registers an employee to attend a lunch. It assigns a lunch ID to each lunch that will be served. For example, employee 207, Dan Smith, will attend a lunch on November 16, 2011. His lunch is identified as `lunch_id = 2`.

The `lunch_id` column is the primary key of this table. This is an example of a **surrogate key**, which is also called a **meaningless primary key**. Each row is assigned a unique number, but there is no intrinsic meaning to that number. It is just a convenient name to use for the row, or the object that the row represents — in this case, a lunch.



The `1_lunches` table shows the most common way to use a surrogate key. Usually a single column is the primary key. That column has a different value in every row.

Some database designers like to use surrogate keys because they can improve the efficiency of queries within the database. Surrogate keys are used especially to replace a primary key that would have many columns, and when a table is often joined to many other tables.

Other designers do not like surrogate keys because they prefer to have each column contain meaningful data. This is an area of debate among database designers, with many pros and cons on each side. People who use databases need only be aware that these columns are meaningless numbers used to join one table to another.

**1\_foods table**

SUPPLIER ID	PRODUCT CODE	MENU ITEM	DESCRIPTION	PRICE	INCREASE
ASP	FS	1	FRESH SALAD	\$2.00	\$0.25
ASP	SP	2	SOUP OF THE DAY	\$1.50	(null)
ASP	SW	3	SANDWICH	\$3.50	\$0.40
CBC	GS	4	GRILLED STEAK	\$6.00	\$0.70
CBC	SW	5	HAMBURGER	\$2.50	\$0.30
FRV	BR	6	BROCCOLI	\$1.00	\$0.05
FRV	FF	7	FRENCH FRIES	\$1.50	(null)
JBR	AS	8	SODA	\$1.25	\$0.25
JBR	VR	9	COFFEE	\$0.85	\$0.15
VSB	AS	10	DESSERT	\$3.00	\$0.50

The `1_foods` table lists the foods an employee can choose for his or her lunch. Each food is identified by a supplier ID and a product code. Together, these two columns form the primary key. The product codes belong to the suppliers. It is possible for two suppliers to use the same product code for different foods. In fact, the product code `AS` has two different meanings. Supplier `JBR` uses this product code for soda, but supplier `VSB` uses it for dessert.

The price increases are proposed, but are not yet in effect. The nulls in the `price_increase` column mean that there will not be a price increase for those food items.

**1\_suppliers table**

SUPPLIER	
ID	SUPPLIER_NAME
-----	
ARR	ALICE & RAY'S RESTAURANT
ASP	A SOUP PLACE
CBC	CERTIFIED BEEF COMPANY
FRV	FRANK REED'S VEGETABLES
FSN	FRANK & SONS
JBR	JUST BEVERAGES
JPS	JIM PARKER'S SHOP
VSB	VIRGINIA STREET BAKERY

The `1_suppliers` table shows the full names for the suppliers of the foods. For example, the `1_foods` table shows that french fries will be obtained from supplier ID `FRV`. The `1_suppliers` table shows that Frank Reed's Vegetables is the full name of this supplier. The primary key of these tables is the supplier ID.

**1\_lunch\_items table**

		SUPPLIER	PRODUCT		
LUNCH_ID	ITEM_NUMBER	ID	CODE	QUANTITY	
-----					
	1	1 ASP	FS	1	
	1	2 ASP	SW	2	
	1	3 JBR	VR	2	
	2	1 ASP	SW	2	
	2	2 FRV	FF	1	
	2	3 JBR	VR	2	
	2	4 VSB	AS	1	
	3	1 ASP	FS	1	
	3	2 CBC	GS	1	
	3	3 FRV	FF	1	
	3	4 JBR	VR	1	
	3	5 JBR	AS	1	

(and many more rows)

When you look at the `1_lunch_items` table you need to be aware that the data in the `item_number` column is aligned to the right because it is a column of numbers. The data in the `supplier_id` column is aligned to the left because it is a column of text. So when you look at the first row, `1 ASP` is not a single piece of data. Instead, the `item_number` value is 1 and the `supplier_id` value is ASP.

The `lunch_items` table shows which foods each employee has chosen for his or her lunch. It also shows whether they want a single or a double portion. For example, look at `lunch_id` 2, which we already know to be Dan Smith's lunch on November 16. It consists of four items. The first item is identified as `ASP-SW`. Here I am putting the `supplier_id` and the `product_code` column data together separated by a hyphen. Looking in the `lunches` table, we find this is a sandwich. The `lunch_items` table says he wants two of them, which is shown in the `quantity` column. See if you can figure out all the foods he wants for his lunch.

The correct answer is:

2 sandwiches

1 order of french fries

2 cups of coffee

1 dessert

The primary key of this table consists of the first two columns of the table, `lunch_id` and `item_number`. The `item_number` column is a ***tie-breaker column***, which is another type of meaningless primary key. In this design, I wanted to use the lunch ID to identify each food within a lunch. However, most lunches have several foods. So I cannot use the lunch ID by itself as a primary key, because that would create several rows in the table with the same value in the primary key, which is not allowed. I needed a way for each row to have a different value in the primary key. That is what a tie-breaker column does. The `item_number` column numbers the items within each lunch. Therefore, the combination of lunch ID and item number provides a unique identity for each row of the table and can serve as the primary key. A primary key of this sort, containing more than one column, is sometimes called a ***composite key***.

## Challenging features of the Lunches database

Most SQL books have you work with a database that is tame and contains no challenges. This book is different. I have intentionally put some features in the Lunches database that could cause you to get the wrong result if you do not handle them properly. I show you how to become aware of these situations and how to deal with them. Many real business databases contain similar challenges. Here are a few of them:

- Two employees are not attending any of the lunches — employee 209, Paula Jacobs, and employee 206, Carol Rose.

- One food has not been ordered in any of the lunches — broccoli.
- One of the departments is not yet staffed with any employees — the personnel department.

## Key Points

---

- In this book we assume that the database has already been built and you just need to learn how to use it. By analogy, this book shows you how to drive a car without trying to show you how to build one.
- Databases are used in many businesses and SQL is used in many software products, so the skills you learn will help you in many different situations.
- Tables are the main construct of a database. All data is kept in tables. Also any data that is given to you will be given in the form of a table. Tables have columns and rows. Usually there are many more rows than columns.
- Most tables have a primary key. This gives a name to each row of the table and prevents the table from having any two rows that are identical.
- There are a few differences between Oracle and Access, but there are many more similarities.
- Oracle is mostly used in businesses with large databases. Hundreds of people may be using the database at the same time. The database can help coordinate all the people in a business and keep them working together.
- Access is mostly used by individuals with small personal databases. Usually only one person is using the database at any given time. Access is also used in some business situations.

# INDEX

## Symbols and Numbers

- ' (single quote):
  - apostrophe, 105, 107, 127
  - single quote, 52, 66, 103, 107
- (dash):
  - subtract a date from another date, 351
  - subtract a number from a date, 351
  - subtract numbers, 334
- (double dash), comment line, 109
- ! (exclamation mark), 109
- != (not equal), 60, 111
- " (double quote), 40, 103, 106
- # (pound sign):
  - date indicator, 55, 103, 108, 247, 252
  - wildcard character, 66, 112
- \$ (dollar sign), 36, 57, 110
- % (percent), wildcard character, 66, 112
- & (ampersand):
  - concatenate, 110, 340, 344, 348
  - variable in SQL\*Plus, 109, 689, 690, 693
- && (variable in SQL\*Plus), 693
- @ (at sign), 725
- \* (asterisk):
  - count all rows, 403, 411
  - multiply, 111, 334
  - select all columns, 37, 111
  - wildcard character, 66, 112
- , (comma), 36, 57, 102, 106, 110
- . (period):
  - decimal point, 36, 57, 110
  - in table names, 109, 724, 725, 727
- / (forward slash):
  - divide numbers, 111, 335
  - statement end in Oracle, 111
- /\* \*/ (multiline comment), 111
- \ (backslash), integer divide, 335
- : (colon), prefix creating a variable, 110, 689
- ; (semicolon), end of SQL statement, 104, 110
- ? (question mark), wildcard character, 66, 112
- [ ] (square brackets):
  - handling spaces in names, 40, 106, 111
  - variable in Access, 689
  - in wildcard, 66, 112
- ^ (caret, Shift + 6, exponent), 335
- ^= (not equal), 60, 111
- \_ (underscore):
  - in names, 36, 40, 106
  - wildcard character, 66, 12
- || (two double bars or two pipe symbols), concatenate, 110, 340, 344, 349
- + (plus):
  - add a number to a date, 351
  - add numbers, 334
- < (less than), 51, 56, 499
- <= (less than or equal to), 51, 56, 499
- <> (not equal), 51, 60, 111
- = (equal):
  - do not use with null, 69
  - equal condition, 51, 52
  - = null (okay in update), 296
- > (greater than), 51, 56, 499
- >= (greater than or equal to), 51, 499
- 0-parameter function, 365
- 3-valued logic, 60, 120

**A**

Abs function, absolute value of a number, 335

Accept command, get value of a variable, 695

Access:

- autocommit, 174
- case sensitivity, 118
- commit, 173
- count distinct not supported, 416
- create a table with a union, 569
- database versions, 772
- Data Dictionary not supported, 197
- datasheet, 183
- datatypes for numbers, many, 213
- default date format, 247
- documentation, 374
- enter an SQL query, 774
- error messages, 776
- error with addition, 369
- full outer join not supported directly, 526
- indexed with no duplicates property, 286
- input format of dates, 252
- parameter query, 699, 700
- print, 776, 777
- quick start, 771
- read-only file, 773
- read-write file, 773
- relationships, 300
- run a query, 775
- security warning, 774, 778
- select distinct not supported, 45
- set difference not supported, 592
- set intersect not supported, 591
- start database, 772
- time period is represented by numbers, 222
- Trust Center, 778
- validation rule property, 284
- validation text property, 284
- with check option not supported, 189

Access GUI:

- add column, 229
- add row (insert), 165
- autonumber datatype (sequence), 271
- cascade delete in RI, 306
- cascade updates in RI, 308
- change data (update), 166
- column field properties, 269
- column names, 203
- create a saved query (view), 139
- create a sequence, 260
- delete a row, 166

delete a saved query (view), 145

delete a table, 144

Expression Builder, 375

find definition of saved query (view), 201

find indexes, 274

find name of saved query, 198

find table names, 198

format property, 255

primary key, 205, 208

referential integrity, 300

required field, 287

unique constraint, 286

Add (+):

numbers, 334

number to a date, 351

Add a row to a table, 151

Addition:

column function, 403, 420

problem and solution, 422

row function, 328, 329

Add\_months function, 353

Ad-hoc reporting, 670

Administrator account (in Windows), 766

ADO, 742

Age in days, 388, 389

Aggregate function, 401, 402

Algebraic equation, 393

Alter table command:

add a new column, 229

check constraint, 283

check not-null, 287

commit not needed, 227

drop column, 233

drop constraint, 228

drop primary key, 228

foreign key, 291

modify datatype, 231

not-null constraint, 287

primary key, 227, 288

RI (referential integrity), 291

unique constraint, 285

Ampersand (&):

concatenate, 110, 340, 344, 348

variable in SQL\*Plus, 109, 689, 690, 693

Analyze Table command, 226

AND, 51, 85, 90

Apostrophe ('), 105, 107, 127

Application development, rapid, 5

Approximate numbers, 222

## AS:

- column alias, 39
- create table, 135
- create view, 139

ASC, ascending sort order, 72, 77

Associative rule, left and right outer joins do not obey it, 549

## Asterisk (\*):

- count all rows, 403, 411
- multiply, 111, 334
- select all columns, 41, 111
- wildcard character, 66, 112

Atomic data, 14

At sign (@), 725

Autocommit, 174

Automatic datatype conversion:

- in a row function, 369
- in a union, 570

AutoNumber datatype, 216, 260, 391

Avg column function, average, 403, 420

**B**

Backslash (\), integer divide, 335

Base table, 142

Between condition, 51, 53, 64, 65, 124

Bfile datatype, 216

Binary datatype, 215

Binary pattern, 369

Bit datatype, 216

Blob datatype, binary large object datatype, 215

Boolean condition, standard form, 90

Boolean connectors and expression, 85

Byte datatype, 215

**C**

C#, 742

Calendar, 376

Capitalization, *see* case sensitivity

Caret (^; Shift + 6), exponent, 335

Cartesian product, 599

Cascade rule, 303, 306, 308

Case function, 676–689

Case sensitivity, 21, 113–120

CDate function, convert to a date, 364

CDbl function, convert to a Double number, 364

Ceil function, round a number upward, 335

Cell, 14, 15

Change data through a view:

- conceptual diagram, 181, 182
- examples, 183–188

Character function, 340

Character string, 107

Char datatype, fixed length text string, 214

Check constraint, 281, 283

Child table, 290

CInt function, convert to an integer, 364, 371

Clob datatype, character large object datatype, 215

COBOL, 742

Colon (:), prefix creating a variable, 110, 689

Column:

- add a new column, 229
- column alias, 38, 39
- created by a row function, 325
- Data Dictionary, 277
- datatype, 13, 213, 266
- date column, 13
- defined, 10–13
- drop a column (delete), 233
- find names of columns, 266
- format of a column, 38
- heading of a column, 20, 39
- modify datatype of a column, 231
- name, 13,
- number column, 13, 38
- position (sequence), 13
- rename with a column alias, 38, 39
- sequence, 13
- text column, 13, 38
- truncated heading, 39
- width, 39

Column alias:

- order by clause, 74, 564
- union, 561

Column constraint, 316

Column function:

- avg, average, 420
- count, 411
- defined, 399, 401
- grouped, 435
- max, maximum, 403, 404
- min, minimum, 403, 404
- non-grouped, 401
- null group, 444
- restrictions, 467
- stddev, standard deviation, 403
- stdev, standard deviation, 403
- sum, 420
- used in a subquery, 706
- variance, 403

- Column function (*continued*):
  - var, variance, 403
  - where clause processed first, 404, 407
  - workarounds, 467
- Column name, in order by clause of a union, 564
- Column position number, in order by clause of a union, 564
- Combinations, create with cross join, 601, 608
- Comma (,), 36, 57, 102, 106, 110
- Comment:
  - line comment (--), 109, 226
  - multiline comment (*/\* \*/*), 111
- Commit command:
  - alter table command, 227
  - autocommit, 174, 194
  - commit command, 173
  - transaction, 175
- Common mistake, 92
- Compare:
  - full outer joins, 546
  - one pattern with another, 542
  - tables, 509, 544, 578, 592
- Compatibility of datatypes, 571
- Complexity, control of, 146
- Composite key, 29
- Compound condition, 51, 85, 90, 92
- Computer language:
  - different from English, 85
  - punctuation, 102
- Compute Statistics, 226
- Concat function, concatenation, 340, 348
- Conceptual diagram:
  - atomic data, 16
  - beginning and ending tables for a query, 33
  - cell, 14
  - changing data through a view, 181, 182
  - column, 13
  - column function, 401
  - constant value used in a query, 96
  - cross join, 613
  - effect of a query, 33
  - eliminate unneeded raw data early, 613
  - grouped summarization, 437
  - inner join as a series of steps, 478
  - inner join of several tables, 476
  - lunches database, 784
  - primary key, 17, 18
  - problem with addition, 424, 425
  - query without row functions, 323
  - referential integrity (RI), 290
  - row, 12
  - row function:
    - effect on one row, 324
    - effect on the whole table, 325
  - self join, 616
  - shape of a table, 19
  - table, 10
  - table of constants used in a query, 99
  - two-dimensional layout, 609, 611, 612
  - union, 530
  - union contrasted with join, 557
  - whole process of a query, 79,
- Configuration:
  - distributed configuration, 726
  - multiuser (client/server) configuration, 724
  - single-user configuration, 723
  - using Internet, 726
- Consistency:
  - in computer code, 98
  - of data, 281
  - implmented by a constraint, 283
  - implmented by a transaction, 176
- Constant value, *see* literal value
  - hard-coded into queries, 96
  - table of constants, 98
- Constraint:
  - allows only some changes to the data, 283
  - check constraint, 281
  - coded in create table statement, 316
  - foreign key constraint, 290
  - not-null constraint, 281
  - primary key constraint, 288
  - referential integrity, 289
  - uniqueness constraint, 285
- Coordinate:
  - parts of an application, 146
  - people, 721
- Correlated subquery, 714
- Cost of a query, 663
- Count column function:
  - data in a column excluding nulls, 411
  - distinct, 417
  - distinct dates, 431
  - rows, 411
  - to zero, 413
- Count distinct column function, 417
- Counter datatype, 216
  - autonumber, 216
- Create index, 263
  - create unique index, 286



Create or replace view command, 150  
 Create synonym command, 729  
 Create table command:  
   with constraints, 316  
   by defining columns, 211  
   with select statement, 135  
   with a union, 567  
   use a new name, 136  
 Create unique index command, 286  
 Create view command, 139  
   with a union, 567  
 Cross join,  
   avoid using with large tables, 612  
   create a list of all combinations, 608  
   defined, 599  
   properties of inner join, 604  
   used to define inner join, 601  
   used to detect errors, 601, 605  
   uses of, 601  
 Cross product, *see* cross join  
 CStr function, convert to a text string, 364  
 currency:  
   currency datatype, 215  
   stored as a number, 21  
 CurrentUser () function, 366  
 Currval, current value of a sequence, 258

## D

Database:  
   configuration, 723  
   distributed, 725  
   evolution, 6  
   joins are part of the design, 666  
   link, 725  
   objects, 274  
 Datab warehouse, 748  
 Data Dictionary:  
   all database objects, 274  
   all data is in tables, 195  
   ALL\_ prefix, 736  
   all\_tables, 196  
   all\_views, 196  
   column names, 201  
   columns of Data Dictionary, 277  
   datatype of a column, 266  
   DBA\_ prefix, 736  
   definition of a view, 200  
   dict\_columns, 237, 277  
   dictionary table, 237, 276  
   index, 271  
   index of columns, 737  
   index of Data Dictionary, 276  
   overview, 195  
   primary key, 512  
   sequence, 269  
   table names, 197  
   user\_cons\_columns, 204  
   user\_constraints, 196  
   user\_ind\_columns, 271  
   user\_indexes, 271  
   USER\_ prefix, 196, 736  
   user\_sequences, 266  
   user\_tab\_columns, 196  
   user\_tables, 196, 197  
   user\_views, 196, 199  
   view, find statement that defines it, 200  
   view names, 199  
 Datasheet, 183  
 Data table, 142, 290  
 Datatype:  
   automatic datatype conversion, 369, 570  
   autonumber, 216  
   bfile, 216  
   binary, 215  
   bit, 216  
   blob, 215  
   byte, 215  
   char, 214  
   clob, 215  
   counter, 216  
   currency, 215  
   date, 214  
   datetime, 214  
   definition, 212  
   difference in details between products, 213  
   external name, 213  
   find in Data Dictionary, 266  
   float, 215  
   functions to convert, 364, 369  
   image, 215  
   integer, 215  
   internal name, 213  
   interval, 216  
   memo, 215  
   money, 215  
   number, 214, 215  
   numeric, 222  
   OLE object, 215, 216  
   raw, 215  
   rowid, 216

- Datatype (*continued*):
  - rownum, 216
  - smallint, 215
  - storage, 213
  - synonym, 213
  - text, 214, 217
  - timestamp, 216
  - union, 574
  - validation of data, 289
  - varchar, 214
  - varchar2, 214
  - yesno, 216
- Date:
  - alignment, 21
  - between, 129
  - count distinct, 431
  - data entry format, 252
  - datatype, 214
  - default date format, 247
  - display format, 249
  - distinct dates, 129
  - enclosed in pound signs in Access, 55, 63
  - enclosed in single quotes in Oracle, 55, 63
  - format, 247
  - includes time, 222, 350
  - internal format is compressed, 247
  - pattern, 381, 383
  - remove time, 356
  - row function, 130, 350
  - stored with a time, 247
  - time not displayed, 21
  - year, 129
- DateAdd function, 351, 353
- Date datatype, 214
- DateDiff function, 351, 353
- Date() function, 363, 365
- Date indicator (#), 55, 103, 108, 247, 252
- Datetime datatype, 214
- DateValue function, 352
- Day function, 351
- DBA (database administrator), 179, 194, 227, 262, 723, 730, 732
- DBMS (database management system), 195, 264, 690,
- DDL (data definition language), 36
- Decimal point, 36, 57, 110
- Declarative language, 3
- Decode function:
  - apply two functions to a column, 687
  - attach messages to rows, 683
  - defined, 674
  - divide a column into two columns, 685
  - syntax, 675
- Default date format, 247
- Delete, *see* Drop command
  - restricted by a constraint, 167
  - row, 159
- Delete command:
  - autocommit, 194
  - commit, 173
  - constraint, 283
  - limited by security, 169
  - referential integrity (RI), 289, 290
  - rollback, 173
  - view, 179–181
- Delete option: 304
- Desc, descending sort order, 72, 77
- Describe command, 202
- Design decision:
  - consistent names, 23
  - match the datatypes of columns, 23
  - numeric datatype, 23
  - prefix of table names, 23
  - referential integrity (RI), 290
  - table of constants, 95
  - text datatype, 23
  - tie-breaker column, 29
- Design objective of SQL:
  - coordinate people to work together, 743
  - focus on information, 742
  - handle a large amount of information, 741
  - keep it simple, 742
- Dict\_columns table, 237, 277
- Dictionary table, 276, 737
- Distinct:
  - combination of columns, 417
  - count distinct, 415
  - select distinct, 44
  - uniqueness constraint, 285
- Distributed configuration, 725, 729
- Dividing one column into two, 585
- DML (data modification language), 227
- Documentation:
  - Access, 374
  - Access row functions, 375
  - Oracle, 373
- Dollar sign (\$), 57, 110
- Double bar (pipe symbol, Shift + \):
  - concatenate (||), 110, 340, 344, 348
  - heading separator (|), 110

Double dash (--), comment line, 109

Double quote ("), 40, 103, 106

Drill down, 441

Drop command:

column, 233

constraint, 228

primary key, 228

sequence, 257

table, 144

view, 145

Dual table, 336

Duplicate rows:

can all be the same object, 238

can be different objects, 237

distinguish them, 240

eliminate, 49, 237, 239

problem with, 237

when to allow, 237

Duplicate values, 34

## E

Efficiency, 262, 264, 661, 663–665

English:

different than computer languages, 85

misuse of AND and OR, 86

misuse of NOT, 86

punctuation, 102

Equal (=), 51, 52

Error detection:

cross join, 601

punctuation, 102

too many rows in the result, 605

truncation, 154

where clause, 605

Error message:

constraint, 167, 316

details are often wrong, 122, 448

location of the error is usually right, 122

one error message is best, 123

plan your error messages, 316

Errors in a pattern, how to find them, 542

Event, 11

Exclamation mark (!), 109

Exists condition in a subquery, 716

Exp function, exponent, 335

Explain option, 194

Exponent (^), 335

Expression, 601

Expression Builder, 375

External name, 213

Extract function, 351, 352

## F

Field, 10

Fixed length string, 217

Flexibility in computer code, 98

Float datatype, 215

Floor function, round a number downward, 335

Foreign key, 290

Foreign key clause, 292

Form, 744

Format:

date, 247

defined, 247

used to display a date, 249

used to enter a date, 252

Format function, 250, 254, 356, 365

Forward slash (/), divide, 111, 335

Free format, 108

From clause:

full outer join, 526

inner join, 502

joining several tables at once, 658

left outer join, 522

new syntax for inner join, 502

overview, 34

right outer join, 524

Full outer join:

comparing two full outer joins, 546

defined, 522

easy to handle, 661

saves all information, 660

of several tables, 661

sorted, 539

symmetric, 661

syntax, 526

uses more computer resources, 661

Functions:

column functions, 401, 403

dates, 350

numbers, 334

text, 341

## G

Grand total, 455, 744

Grant command, 728, 733

used with a role, 734

Greater than condition (>), 51, 56, 499

Greater than or equal to condition ( $\geq$ ), 51, 499

Greatest function, pick greatest number, 364

Group by clause:

- add more columns, 451
- cannot mix detail with summary, 452
- eliminating some groups, 459
- groups formed on one column, 438
- groups formed on several columns, 441
- how to mix detail with summary, 455
- null group, 444
- restrictions on a grouped query, 467

GUI (graphical user interface), advantages and limitations, 161

## H

Hard-coded value, 101

Having clause:

- contrast with where clause, 462
- defined, 460
- redundant, 463
- replaced by where clause, 463, 466

Heading separator (|), 110

Help, 374

History option, 192

HR userID, 754

HTML data, 222

## I

Identify user, 732

If-then-else logic, 673

- case function, 676
- decode function, 675
- IIF function, 680

Image datatype, 215

Immediate if (IIF) function:

- apply two functions to column, 685, 687
- attach messages to rows, 683
- divide one column into two, 685
- syntax, 680

Inactive view, 147

In condition,

- contrast with between condition, 64
- contrast with equal to condition, 62
- used to handle dates, 65
- used with a subquery, 703

Index:

- create an index, 263
- created by DBA, 262
- find indexes in Data Dictionary, 271

how an index works, 262, 264

optimizer, 264

primary key, 263

used for efficiency, 664

Information:

- can always be put in a table, 10
- focus on, 741
- handle a large amount of information, 5, 741
- handle a small amount of information, 5
- information level, 741
- systems, 5
- a table contains one type of information, 11

Information loss:

- full outer join keeps all information, 660
- inner join, 660
- left and right outer join, 660

Initcap function, capitalize first letter of each word, 342

Inner join, 473–515

- with between in join condition, 497
- defined from a cross join, 601
- drops unmatched rows, 489, 491, 604
- examples, 504
- with greater than in join condition, 499
- joining many tables, 513
- joining two tables,
  - lookup table, 505
  - loses information, 489, 491, 660
  - many-to-many relationship, 487
  - many-to-one relationship, 483
  - matching on a range of values, 497
  - new syntax (from clause), 502
  - nulls in columns of join condition, 491
  - one-to-many relationship, 485
  - one-to-one relationship, 479
  - properties of an inner join, 604
  - related to outer join, 519
  - with a row function in join condition, 501
  - self join, 613
  - with several matching columns, 495
  - standardized join conditions, 666
  - with summarization, 510
  - symmetry of an inner join, 660
  - variations of syntax, 493, 502
  - writing SQL in a series of steps, 477, 507, 655

Insert command:

- autocommit, 194
- commit, 173
- constraint, 283
- limited by security, 169

- null, 151, 152
  - referential integrity (RI), 289, 290
  - restricted by a constraint, 167
  - rollback, 173
  - with a select statement, 154
  - several rows at once, 154
  - specify which columns, 152
  - text fields may be truncated, 154
  - transaction, 175
  - with a view, 181
- InStr function, test if one text string contains another, 343, 346
- Integer datatype, 215
- Internal name of a datatype, 213
- Intersect, intersection of tables, 590
- Interval datatype, 222
- Int function, round to an integer, 335
- Into clause, 135
- I/O (input/output), 264
- Is not null condition, 51, 88
- Is null condition, 51, 69

**J**

- Java, 742
- JET engine (in Access), 742
- Join:
  - ad-hoc join, 670
  - combining several tables, 473, 474, 655
  - contrast with a union, 557
  - cross join, 597, 599
  - database design, 666
  - from clause, 658
  - inner join, 473
  - outer join, 475, 519
  - primary key, 558
  - self join, 597
  - in a series of steps, 447, 655
  - standardized join conditions, 666
  - where clause, 658
- Justification of data, 20

**L**

- Last\_day function, 353
- Last row in an Access datasheet, 21
- Lcase function, change text to lowercase, 342
- Least function, pick smallest number, 364
- Left outer join:
  - defined, 521
  - difficult on three tables, 660

- loses information, 660
  - non-associative, 549
  - not symmetric, 660
  - problem with it, 549
  - syntax, 522
- Len function, length of a text string, 343
- Length function, length of a text string, 343
- Less than condition (<), 51, 56, 499
- Less than or equal to condition (<=), 51, 56, 499
- Like condition, 51, 66
- Linked tables, 725
- Literal value, 38, 41, 95–98
- Local server, 725
- Logical expression, 601
- Long string, 217
- Lookup table, 504
- Lower function, change text to lowercase, 342
- Lpad function, put spaces on the left of text, 341
- Ltrim function, remove spaces from the left of text, 342
- Lunches database:
  - crosstab query, 638, 641
  - data validation rules, 785
  - diagram, 783
  - introduction, 24
  - join conditions, 784

**M**

- Many only means more than one, 483
- Many-to-many relationship, 487
- Many-to-one relationship, 483
- Master index, 373
- Materialized view, 730
- Max column function, maximum value, 403
- Memo datatype, 215
- Mid function, get text from middle of a string, 341, 346
- Midnight, 252
- Min column function, minimum value, 403, 409
- Minus, subtract one table from another, 592
- Missing data, *see* Null
- Mod function, remainder after division, 335, 379
- Money datatype, 215, 663
- Month function, extract the month from a date, 352
- Months\_between function, number of months between two dates, 353
- Multiuser configuration, 721, 727

**N**

Names, spaces in, 40, 102, 106

Next\_day function, date of the next day, 353

Nextval, next value in a sequence, 258

Non-associative:

left outer join, 549

right outer join, 549

Non-procedural language, *see* Declarative language

Not:

boolean not, 51, 85, 88

is not null condition, 51, 88

not between condition, 51, 88

not equal condition, 51, 88

not in condition, 51, 88, 708

not like condition, 51, 88

Not equal condition (<>, !=, ^=), 60, 111

Not-null constraint, 281, 287

Now() function, current date and time, 252, 363, 365

Null:

cannot be in a primary key, 17

change to another value, 363, 366

defined, 13

displayed in Oracle,

effect on inner join, 605

has no datatype, 71

in column functions, 402, 404, 411, 413, 422

in matching columns of an inner join, 489, 491

insert statement, 151, 152

is null condition, 69

missing or unknown data, 13, 70

not in condition, 708

null group, 438, 444

number in a column, 430

problems within subqueries, 708

row function, 327

select distinct, 44, 48

sort order of nulls, 46, 72

value substituted for a null, 422, 428

where clause conditions, 51, 52

why use them, 70

Null group, 48, 438, 444

Number datatype, 214, 213

Numbers:

approximate, 222

floating-point, 222

generating numbers, 621

number functions, 334

numbering lines of a report, 390

precise, 222

NVL function, changes nulls to other values, 363, 366

NVL2 function, 364

NZ function, changes nulls to other values, 363, 366

**O**

Object, 11

OLE object datatype, 215, 216

ON clause:

contains join condition, 502

delete rule in RI, 304

update rule in RI, 306

One-to-many relationship, 485

One-to-one relationship (two meanings), 481

Online help, 374

Optimizer, 264, 466, 655, 701

OR, 51, 85, 90

Oracle:

autocommit, 174

automatic start, 763

commit, 173

create a new userID, 755

crosstab query, 645, 674

cube command, 455

current information, 752

Data Dictionary, 736

default date format, 247

documentation, 373

download file, 753

download Oracle, 753

dual table, 260

enter a query, 769

entering formatted dates, 252

error messages, 763

explain option, 194

free, 751

full outer join, 526

get started after install, 754

go to database home page, 766

home page, 768

how to get it, 751

how to stop it, 736

install Oracle, 754

intersect command, 590

login to administrator account, 766

manual startup, 764

minus command, 592

- numbers all have one datatype, 213
  - page not found error, 767
  - print result tables, 770
  - quick start, 765
  - regular expression, 112
  - remove automatic start, 763
  - rollback, 173
  - rollup command, 455
  - run a query, 769
  - sequence, 257
  - setup for this book, 754
  - SQL Commands page, 768
  - sqlfun\_build\_oracle\_tables.txt file, 758
  - sqlfun\_delete\_oracle\_tables.txt file, 763
  - start an Oracle session, 766
  - start database, 764, 767
  - stop database, 763
  - system requirements, 753
  - transaction, 175
  - Web browser interface, 766
  - which version to get, 752
  - Oracle GUI:
    - add a new row (insert), 163
    - change a row (update), 163
    - create a new user, 755
    - delete a row, 164
    - running a script file, 758
    - running one SQL command, 192
    - SQL Commands page, 192
  - Order by clause:
    - column alias, 74
    - column name, 71
    - column position number, 72
    - in an insert statement, 155
    - more than one column, 75
    - overview, 34, 71
    - sorting a full outer join, 539
    - in a table, 135, 390
    - union, 561, 563
    - in a view, 140, 390
  - OTN (Oracle Technology Network), 753
  - Outer join:
    - applications, 534
    - full, 522, 526
    - left, 521, 522
    - related to inner join, 519
    - right, 521, 524
    - self join, 613
    - separating into two steps, 536
    - sorted full outer join, 539
    - standardizing joins, 666
    - written as a subquery, 717
  - Owner name, 725
  - Owner of a table, 724, 727
- P**
- Parameter query:
    - in Access, 699
    - in Oracle, 690
  - Parentheses and precedence, 92
  - Parent table, 290
  - Parts of a table, 9
  - Password, 732
  - Pattern, 66
  - Patterns of dates, 381, 383
  - Patterns of numbers, 376
  - Percent (%), wildcard character, 66, 112
  - Performance, *see* Efficiency
  - Period (.):
    - decimal point, 57, 110
    - in table names, 109, 724–727
  - Pipe (|), 110
  - Pivot clause, 624
  - Pivot query, 624
  - Position number of a column, 22
  - Pound sign (#):
    - date indicator, 55, 63, 103, 108, 252
    - wildcard character, 66, 112
  - Power function, exponent, 335
  - Precise numbers, 222
  - Presentation level, 741
  - Preventative delete, 149
  - Primary key:
    - can have many columns, 17
    - cannot contain a null, 17
    - change primary key, 228
    - composit key, 229
    - create primary key, 226
    - Data Dictionary, 512
    - defined, 16
    - index, 263
    - join, 558
    - meaningless, 26
    - numeric, 23
    - only one allowed, 227
    - part of the data, 17
    - prevents duplicate rows, 237
    - referential integrity, 290
    - subject of the row, 17
    - surrogate key, 26

Primary key (*continued*):

- two meanings, 311
  - union, 558
  - usually the first columns, 17
- Prime numbers, 379
- Private synonym, 728
- Privilege, 732
- Procedural language, 3
- Production table, 653
- Prompt command, display a message, 695
- Public synonym, 728
- Punctuation, 102, 110, 224

**Q**

## Query:

- ad-hoc query, 670
  - crosstab query, 597
  - frequent running query, 665
  - parameter query, 673, 690
  - query result listing, 36
  - saved query in Access, 139
  - select statement, 34
  - subquery, 673, 700, 701
- Question mark (?), wildcard in Access, 66, 112
- Quick start:
- with Access, 771
  - with Oracle, 765

**R**

- Rapid application development, 5
- Raw datatype, 215
- Real datatype, 215
- Record defined, 10
- Reference clause, 292
- Reference table, 290
- Referential integrity (RI) constraint:
- cascaded deletes, 303, 306
  - cascaded updates, 308
  - change one of the valid values, 298
  - defined, 290
  - delete rules, 303
  - deletes allowed by RI, 294
  - deletes prevented by RI, 295
  - inserts allowed by RI, 294
  - inserts prevented by RI, 293
  - primary key, 311
  - relationship between tables, 299
  - RI on a single table, 315
  - RI on several columns, 313
  - set null rule for deletes, 303, 304
  - turned off temporarily, 289
  - update rules, 303, 308
  - updates allowed by RI, 294
  - updates prevented by RI, 393, 395
- Regular expression, 112
- Relational database, 3–5
- Relationship, 11, 300
- Remote server, 725
- Replace function, substitute one string for another, 341
- Replica, 725, 730
- Replication, 725
- Report, can mix detail with summary data, 455, 744
- Required field, 287
- Reserved word, 104
- Restrict rule, 303
- Result table, 36
- Revoke command, 733, 734
- Right outer join:
- defined, 521
  - difficult on three tables, 660
  - information loss, 660
  - non-associative, 549
  - non-symmetric, 660
  - problem with it, 549
  - syntax, 524
- Role, 734
- Rollback command:
- autocommit, 194
  - cancel changes to data, 173
  - used in a transaction, 175
  - used to change data temporarily, 427
- Rollup function, 455
- Round function:
- round dates, 353
  - round numbers, 334, 335
- Row:
- defined, 10
  - delete a row, 159
  - distinct rows, 49
  - insert (add) a row, 151
  - with maximum or minimum value, 409
  - number in output of query, 663
  - unmatched, 489, 491, 519, 604
  - unordered, 12
  - update (change) a row, 157



- Row function:
  - apply two functions to a column, 587
  - arithmetic on numbers, 334
  - change datatype, 369
  - change nulls to other values, 366
  - creates a new column in a table, 325
  - creates a new value, 324
  - defined, 232, 324
  - defined in the first step, 331
  - documentation, 372
  - functions on dates, 350
  - functions on numbers, 334
  - functions on text, 341
  - identify the date and the user, 363
  - null input gives null output, 227
  - numeric test of a row function, 336, 337
  - other row functions, 363
  - pick one value, 364
  - test a row function, 336, 337
  - used in a join condition, 501
  - used in all clauses of a select statement, 327, 329, 332
  - used in a series of steps, 331
- Rowid datatype, 216
- Rownum datatype, 216
- Rownum function, 390
- Rpad function, add spaces to the right of text, 341
- Rtrim function, remove spaces from right of text, 342
- Run button, 136
- S**
- Saved query, in Access, 139
- Security, 169, 732
- Select clause:
  - column alias, 39
  - overview, 34, 37
  - in a union, 561
- Select data early in the process, 664
- Select distinct:
  - Access does not support this, 45
  - distinct values in one column, 44
  - distinct values in several columns, 48
  - null, how it is handled, 44
  - overview, 37
  - used to eliminate duplicate rows, 239
- Select statement:
  - defined, 34
  - overview, 34
  - processes one row at a time, 614
  - query, 35
  - whole process, 79, 463
  - within a union, 561
- Self join:
  - defined, 613
  - example, 616, 618, 621
  - used with a sequence, 618
- Semicolon (;), end of SQL statement, 104, 110
- Sequence:
  - create a sequence in Oracle, 257
  - Data Dictionary, 269
  - defined, 257
  - sequences in Access, 260
  - sequences in Oracle, 258
  - value in row does not change, 257
- Series of steps, SQL written in:
  - inner join, 507
  - inner or outer join, 655
  - outer join, 536
  - row function, 331
  - technique shown, 146
  - views created in layers, 146
- Server:
  - local, 725
  - remote, 725
- Set difference, 590, 592
- Set intersection, 590
- Set null rule, 303, 304
- Sgn function, sign of a number, 335
- Shared application, 724
- Shared database, 663
- Sharing the database with other people:
  - autocommit, 194
  - snapshot, 730
  - synonym, 728
  - table, 173, 727, 736
  - transaction, 175
- Sign function, sign of a number, 335
- Simple logical condition, contrast with complex logical condition, 90, 92
- Simultaneous users, 4
- Single quote ('), 52, 63, 66, 103, 107
- Single-user configuration, 723
- Slash (/):
  - backslash (\) used for integer divide, 335
  - divide numbers (/), 335
  - multiline comment (/\* \*/), 111
  - statement end in Oracle (/), 111
- Slowest operation in the CPU is I/O, 264

- Smallint datatype, 215
- Small table, 71, 608, 664
- Snapshot, 725, 727, 730
- Sort, 73, *see* Order by clause
- Sort order:
  - position of a null, 46
  - primary sort order, 75
  - secondary sort order, 75
- Soundex function, find words that sound alike, 341
- Space function, creates a string of blank characters, 342
- Spaces in names, avoid them, 39, 40, 102
- Spacial data, 222
- SQL:
  - defined, 3, 4
  - design objectives, 739, 741
  - newer interfaces, 743
  - products, 7
  - typical applications, 748
- SQL Commands page, 192
- Sqlfun:
  - sqlfun\_build\_oracle\_tables.txt file, 224, 758
  - sqlfun\_delete\_oracle\_tables.txt file, 763
- SQL\*Plus, 254, 673, 742
- SQL script, how to run, 758
- Sqr function, square root, 335
- Sqrt function, square root, 335
- Square brackets [ ]:
  - handling spaces in names, 40, 106, 111
  - variable in Access, 689
  - in wildcard, 66, 112
- Standard form:
  - of a boolean expression, 90
  - of a where clause, 90
- Stddev function (with two Ds), standard deviation, 403
- Stdev function (with one D), standard deviation, 403
- Storage data, 222
- StrConv function, convert a text string to uppercase, lowercase, or with the first letter of each word capitalized, 342
- String (of alphanumeric characters):
  - fixed-length string, 217
  - long string, 217
  - variable-length string, 217
- String function, create a string that repeats a character a specified number of times, 341
- Subquery:
  - correlated, 714
  - defined, 701
  - example, 710
  - with an exists condition, 716
  - with an in condition, 708
  - limitations, 719
  - with a list of values, 703
  - nested, 718
  - with a not in condition, 708
  - older features, 714
  - replace with a join, 719
  - to select most current data, 714
  - with a single value, 706
  - used in an update, 710
  - used to compare tables, 712
  - to write an outer join, 717
- Substr function, substring of text, 341, 346
- Subtotal, 455, 744
- Subtract (-):
  - a date from another date, 351
  - a number from a date, 351
  - a number from another number, 334
- Sum:
  - problem with addition, 422
  - sum column function, 403, 420, 422
  - sum row function, 328, 329, 334, 422
  - using NVL or NZ in a sum, 422
- Summarizing data:
  - cannot mix detail with summary, 447
  - column functions, 403
  - eliminating groups of summarized data, 459
  - group by clause, 438
  - grouped summarization defined, 435, 437
  - how to mix detail with summary, 455
  - null group, 444
  - summarizing and an inner join, 510
  - summarizing an entire column, 401
  - table of summarized data, 665
  - workaround restrictions, 467
- Surrogate key, 26
- Symmetry in joins:
  - cross join, 605
  - full outer join, 661
  - inner join, 605
- Synonym:
  - alternate name of a datatype, 213
  - alternate name of a table, 728
- Sysdate function:
  - contrast with date function, 365

current time and date, 252, 363, 365  
 System Catalog, 195, *see* Data Dictionary  
 System userID, 754, 755  
 System variable, 365  
 Systimestamp function, current date and time  
     (very precise), 363

**T**

## Table:

add a column, 229  
 add a primary key, 225, 226  
 add rows, 223  
 alter table command, 226–234, 283–316  
 base table, 142  
 can handle all information, 10  
 can have only one primary key, 17  
 child table, 290  
 column names are part of the table, 201  
 combining, 473, 475, 557, 599  
 combining with cross join, 599  
 compare two tables, 544, 578, 590, 592, 712  
 conceptual diagram, 10  
 constants table, 95  
 contains a single type of information, 11  
 create a table, 135, 211, 212, 224, 316  
 data table, 142, 290  
 definition, 211  
 delete a column, 233  
 delete rows, 159  
 derived table, 36  
 dictionary table, 737  
 difference from a view, 142  
 drop (delete) a table, 144  
 dual table, 336, 365  
 with duplicate rows, 236, 237  
 empty, 212  
 insert (add) rows, 151, 154  
 large, 12, 85, 601, 612  
 lookup table, 290, 504  
 make unusual changes, 234  
 modify the datatype of a column, 231  
 most tables have a primary key, 17  
 name of a table, 197  
 with one row, 12  
 with an order by clause, 390  
 owned by another person, 727  
 owner, 724  
 parent table, 290  
 parts of a table, 9  
 preventative delete of a table, 149

primary key, 16  
 production tables, 653  
 qualification of table name, 724  
 reference table, 290  
 result table, 36  
 RI as a relationship between tables, 299  
 shape of a table, 18  
 similar to a view, 139, 142  
 simultaneous use, 736  
 small table, 34, 71, 599, 608, 664  
 static data, 142  
 stored on disk, 142  
 summarized data kept in a table, 665  
 synonym, 728  
 table of constants, 96  
 update (change) rows, 157  
 usually has more rows than columns, 18  
 virtual table, 36

## Table alias, 481

contrast with synonym, 728  
 used in a self join, 614

## Table of constants, 98

## Text column, 13, 38

## Text datatype, 214

## Text functions, 340

## Text string, 107

## Three-valued logic, 60, 120

## Time:

date datatype, 214  
 datetime datatype, 214  
 entered with a date, 252  
 included with a date, 350  
 period of time, 216  
 point of time, 216  
 remove the time from the date, 356  
 set the time to midnight for a date, 252  
 time is always stored with a date, 247

## Time() function, get the current time, 363, 365

## Timestamp datatype, 216, 222

## To\_char function, convert to text, 249, 356, 364, 365

## To\_date function, convert to a date, 252, 352, 364

## To\_number function, convert to a number, 364

## Transaction, 175, 194

## Transform clause, 624

## Trim function, trim spaces from both ends of text, 342

## Trunc function:

truncate time from dates, 352  
 truncate numbers, 335

Tune a database, 262

Two-dimensional layout, 609, 611, 624

## U

Ucase function, change text to uppercase, 342

Underscore (\_):

- in names, 36, 40, 106

- wildcard character, 66, 12

Union:

- applications, 576

- apply two functions to a column, 587

- attach messages to rows, 583

- automatic datatype conversion, 570

- basics, 557

- contrast with a join, 557

- create a table with a union, 567

- create a view with a union, 567

- divide one column into two columns, 585

- identify source of data, 581

- introduction to unions, 529

- match columns with different datatypes, 570, 574

- matching datatypes, 557

- order by clause in a union, 563

- primary key, 558

- same number of columns, 557

- select statements in a union, 561

- syntax, 530

- of tables with different numbers of columns, 576

- unconventional, 573

- union all, 559

- union of several tables, 588

- used to create a full outer join, 526

Union all:

- contrast with union, 559

- syntax, 560

- when to use it, 560

- why to use it, 559

Unique index, 263, 285

Uniqueness constraint, 281, 285

Unknown, truth value of a null, 120

Unmatched rows:

- dropped by an inner join, 489, 491, 604

- restored by an outer join, 519

Updateable views, 180

Update command:

- autocommit, 194

- change data, 157

- commit, 173

- constraints restricting changes, 283, 393, 395

- rollback, 173

- set clause, 157

- subquery, used in, 710

- view, used for updates, 179–181

- where clause, 157

Upper function, change text to uppercase, 342

User\_cons\_columns table, 196, 204

User\_constraints table, 196, 204

User function, userID using the database, 363, 732

UserID, 732

User\_ind\_columns table, 271

User\_indexes table, 271

User\_objects table, 275

User\_sequences table, 269

User\_tab\_columns table, 196, 266

User\_tables table, 196, 197

User\_views table, 196

## V

Validation, of data:

- check constraints, 283

- datatype as data validation, 289

- not-null constraint, 287

- primary key constraint, 288

- referential integrity constraint, 289

- uniqueness constraint, 285

Validation rules, 283

Validation text, 284

Valid values, 290

Varchar datatype, 214

Varchar2 datatype, 214, 215

Var column function, variance, 403

Variable:

- & (ampersand) in variable name, 690, 693

- && (double ampersand) in variable name, 693

- accept command, 693, 695

- define a variable in SQL\*Plus, 695

- used in a parameter query, 689, 690

- variables in Access, 698

Variance column function, variance, 403

View:

- change data through a view, 179

- circular definitions not allowed, 146

- column names, 201

- create a view, 139

- create or replace a view, 151

- difference from a table, 142

- drop (delete) a view, 145

- dynamic data, 142
- find column names, 199
- find select statement that defines a view, 200
- layers of views, 146
- materialized views, 725, 730
- might be dropped automatically, 146
- with an order by clause, 390
- preventative delete of a view, 149
- restoring a view, 147
- select statement stored on disk, 142
- similar to a table, 139
- used to standardize joins, 666
- updateable view, 180
- with check option, 189

Visual Basic, 742

**W**

- Web browser, 742
- Web page data, 222
- Web site for this book, 752
- Web tools, 745

- Weekday function, day of the week, 352
- Where clause:
  - compound conditions in where clause, 85
  - contrast with having clause, 462
  - not used in cross join, 600
  - overview, 43, 50
  - preferred to having clause, 463
  - standard form, 90
  - used to eliminate raw data, 466
  - used to join several tables at once, 658
  - why it is complex, 50
- Wildcard characters, 66, 112
- With check option, 189

**Y**

- Year function, extract the year from a date, 352
- Yesno datatype, 216

**Z**

- Zero, counting to, 413, 455, 457, 534
- Zero-length string, 70