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

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

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.

Introduction

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.

Major SQL Products	Other SQL Products (and Products Based on SQL)
Oracle	4th Dimension
Microsoft SQL Server	SQLBase
Microsoft Access	CSQL
MySQL	FileMaker PRO
DB2 (IBM Data Server)	Helix Database
Informix	ODBC
PostgreSQL	Ingres
Sybase	MonetDB
Microsoft Visual FoxPro	H2
NonStop SQL	MaxDB
Dataphor	VMDS
Teradata	TimesTen
	Openbase
	eXtremeDB
	Interbase
	OpenEdge ABL
	SmallSQL
	Linter 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

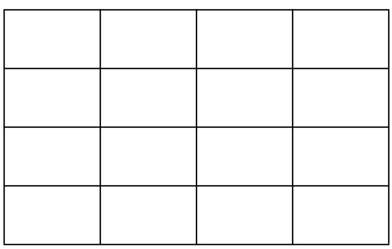
In a relational database, all the data is stored in tables. A table is a two-dimensional structure that has ***columns*** and ***rows***. Using more traditional computer terminology, the columns are called ***fields*** and the rows are called ***records***. You can use either terminology.

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."

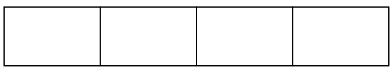
All the rows within a table represent the same type of object. If you have 100 doctors in a hospital, you might keep all the information about them in a single table. If you also want to keep information about 1,000 patients who are in the hospital, you would use a separate table for that information.

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.

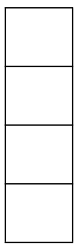
Each column has a name, for instance “favorite game,” and a datatype. We discuss datatypes in chapter 6, but for now let’s keep it simple. There are three main datatypes: text, numbers, and dates. This means that there are three types of columns: columns containing text, columns containing numbers, and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

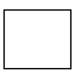
A ***cell*** occurs where one row meets with one column. It is the smallest part of a table and it cannot be broken down into smaller parts.

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.

Full Name
Susan Riley

(A)

First Name	Last Name
Susan	Riley

(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

A			
B			
C			
D			

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

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.

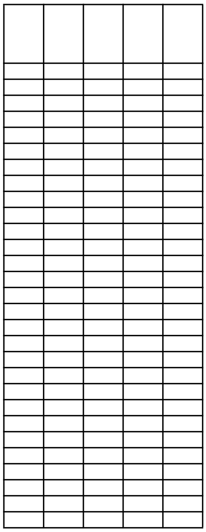
1-11 Most tables are tall and thin

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.

`l_employees` table: Oracle format

```
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
```

`l_employees` table: Access format

L_EMPLOYEES

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 `l_employees` table

The `l_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.

`l_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 (`l_employees`)
- Departments (`l_departments`)
- Constants (`l_constants`)
- Lunches (`l_lunches`)
- Foods (`l_foods`)
- Suppliers (`l_suppliers`)
- Lunch Items (`l_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 `l`. 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.

`l_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.

l_constants table

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

The `l_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.

l_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 `l_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 `l_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.

l_foods table

SUPPLIER ID	PRODUCT CODE	MENU ITEM	DESCRIPTION	PRICE	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 `l_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.

l_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 l_suppliers table shows the full names for the suppliers of the foods. For example, the l_foods table shows that french fries will be obtained from supplier ID FRV. The l_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.

l_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 l_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 l_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 l_foods table, we find this is a sandwich. The l_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

A

N

O