



PART I

Database Basics



CHAPTER 1

Introduction to Databases



Welcome to the wonderful world of databases. “Huh?” you might say to yourself. “What’s so wonderful about databases?” The answer lies not with databases themselves, but with what they contain: information. Information that can make your life easier, transform a mountain of chaos into a manageable chunk of order, and help you discover things you would never have the time to find out otherwise. When you learn how to use databases knowledgeably, you learn how to control the way you receive information. More and more, this fundamental skill is becoming the difference between getting the answers you need and not.

What Exactly Is a Database?

Stripped down to its most basic form, a *database* is a list of information. Or a set of lists that work together. A *database program* is a fancy list manager.

Databases are a regular part of just about everyone’s life. For example, a telephone book is a paper representation of a database. It provides you with specific pieces of information about people, and it sorts that information into an order designed to help you find what you want quickly. If the telephone book contains business listings—often called the “yellow pages”—the information there will be sorted by business type, and within each business type, it will be sorted by name.

You probably have an address book—it’s a database, too. So is your checkbook register. If your local television provider has a channel that shows what’s playing on each channel, that information is coming from a database.

You have probably used databases on the Internet, too. If you have looked for a book or CD using a web site, the information that came back to you was pulled from a database. (I recently designed just such a database for the world’s largest music-publishing company.) Online auction sites are large databases containing information about buyers, sellers, items, bids, and feedback. Internet search engines such as Google and Yahoo! are enormous databases containing key information about millions of web pages.

Tables

Databases are always designed to store a particular type of information. For instance, in the case of a telephone book, the information is about people (in the white pages) and about businesses (in the yellow pages). A database would generally store such information by having one *table* containing all the information about people, and another table containing the information about businesses. Each of these tables would look a lot like a spreadsheet, with individual columns for each type of information being stored (name, address, number, and so on) and a row for each person or business. For instance, Table 1-1 demonstrates a simple employee table.

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | SALARY | HIRE_DATE |
|-------------|------------|-----------|--------|-----------|
| 1024 | Devin | Campbell | 63000 | 15-JAN-04 |
| 2048 | Alexa | Hammond | 67000 | 17-FEB-05 |
| 3072 | Blake | Anthony | 68000 | 27-JAN-06 |
| 4096 | Brianna | Berlin | 69000 | 29-MAY-07 |

TABLE 1-1. *A simple employee table*

The most important thing to remember about a table is this: *It stores information about **one** type of thing.* A table about people will not store information about lawnmowers! A table about school classes will not store information about the people who take those classes. If a database needs to store information about more than one type of thing—and it almost always will—it does so by using more than one table. For example, to properly track school classes, a database would have (at the very least) a table for faculty, another one for classes, a third one for classrooms, and a fourth one for students. Keeping the different types of information separate allows a database to store information very efficiently, and in a highly organized (and therefore easy-to-use) manner.

Rows/Records

The simple employee table shown earlier contains information about four people. Each person's information is on a line of its own. Each line is called a *row*, and the data it contains is called a *record*. Each row will contain the information for one—and only one—of the items defined by the table's name. For instance, in an employee table, each row contains information for only one employee. Similarly, each employee's information is stored on just one row. You design the table so that it only takes one row to hold all of the information specific to each of whatever the table's name says it holds. (You'll be doing this yourself very soon.)

Columns/Fields

Each row contains several pieces of information. In the employee example, those pieces included employee ID, first name, last name, and so on. In a table, these pieces of information are stored in *columns*. The junction point of a row and a column—for instance, a particular person's first name—is called a *field*. A field contains a single piece of information about something—for instance, a telephone number for one person.

6 Oracle Database 10g PL/SQL 101

Let's think about a concrete example. Imagine that you want to put information for five of your friends onto 3" × 5" index cards. Each friend will get his or her own index card, so you'll use a total of five cards. By doing this, you're creating a small database. It's a physical database, as opposed to one on a computer, but it's a database nonetheless, and the concepts of tables, records, and fields still apply. Figure 1-1 shows the relationships between the index cards and these terms.

Now let's say you've put the information for the same five friends into a spreadsheet. Figure 1-2 shows how the terms you've just learned would apply to that situation.

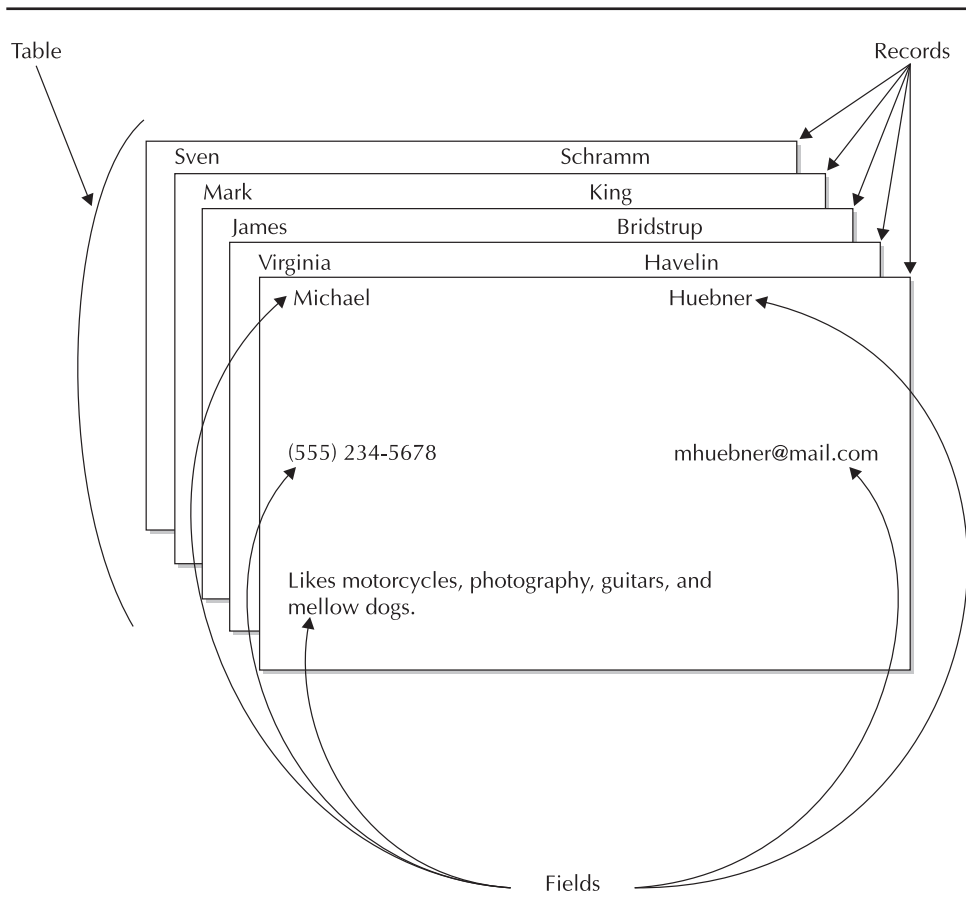
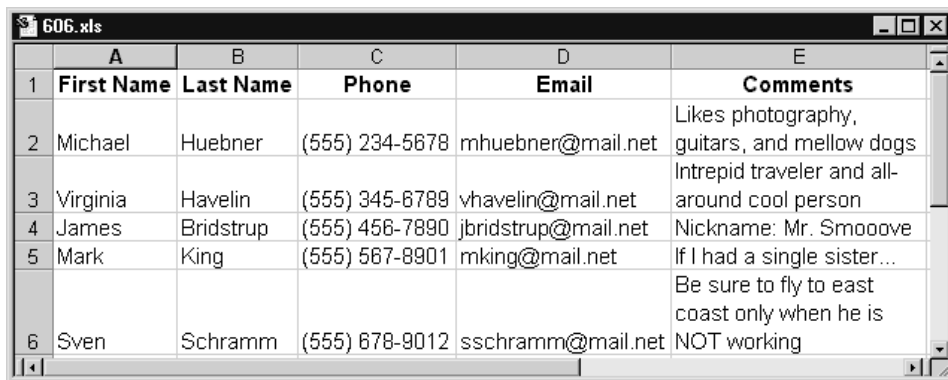


FIGURE 1-1. Friend information laid out on index cards



| | A | B | C | D | E |
|---|-------------------|------------------|----------------|---------------------|--|
| 1 | First Name | Last Name | Phone | Email | Comments |
| 2 | Michael | Huebner | (555) 234-5678 | mhuebner@mail.net | Likes photography, guitars, and mellow dogs |
| 3 | Virginia | Havelin | (555) 345-6789 | vhavelin@mail.net | Intrepid traveler and all-around cool person |
| 4 | James | Bridstrup | (555) 456-7890 | jbridstrup@mail.net | Nickname: Mr. Smooove |
| 5 | Mark | King | (555) 567-8901 | mking@mail.net | If I had a single sister... |
| 6 | Sven | Schramm | (555) 678-9012 | sschramm@mail.net | Be sure to fly to east coast only when he is NOT working |

FIGURE 1-2. *Friend information laid out in a spreadsheet*

How Is a Database Different from a Spreadsheet?

Based on the “list of friends” example, it would be easy to think “Well, why not just keep the list in a spreadsheet?” For a list like the one shown, you could. A database becomes appropriate when the lists get more complicated, or when they need to be used in a more sophisticated environment. What follows are several distinctive features that databases offer.

Many Rows

Since spreadsheets are designed primarily for financial calculations—essentially, they function as ledger sheets with formulas where the answers go—they aren’t designed to accommodate the number of rows that a business database will need. It’s common for a spreadsheet to have a limit of 65,536 on the number of rows it can contain. While that is a lot of rows for a spreadsheet, it isn’t really that many for a database. (For example, I just used a web site to find out how many Internet newsgroup messages contain the word “Oracle.” There were approximately 2,330,000 matching messages, and every one of those messages is stored in the web site’s database. That web site’s database contains over two million records about Oracle messages alone; imagine how many records it contains in total!) It’s not uncommon for a business database to contain millions of rows and for large businesses to have databases containing billions of rows. No spreadsheet is going to handle that!

Many Users Simultaneously

Because databases are at the core of many businesses, it's essential that they allow lots of people access to the same data simultaneously. To understand why, imagine a retail store chain that has a hundred computerized cash registers distributed among its stores. On a busy sale day, many of those registers are going to be doing transactions at the same time. If you had to wait for all of the other transactions to be completed before yours could go through, you would probably get frustrated and leave, and so would a lot of other people—and sales would suffer. On a bigger scale, airline reservation systems can deal with thousands of requests every second—if each of those had to wait for all the others, the reservation system would be very slow and annoying. The ability to accommodate large numbers of simultaneous users is one of the key characteristics of a database. A well-designed database can answer requests from thousands—or even millions—of users simultaneously and still provide excellent performance.

Security

Databases contain some of the most sensitive information in a business: salaries, customer information, and project schedules, for instance. If this information were to be deleted, changed, or revealed to coworkers or competitors, it could cause problems ranging from embarrassment to failure of the business itself. Because of this, databases have extremely robust security systems. You won't find any passwords stored in easily snooped text files in a database system; everything is encrypted, including the information sent between the database and a user's computer when he or she logs in.

Even valid users of a database don't necessarily get access to everything the database contains. Users can be given privileges to specific tables, and not to others. It's even possible to make some columns within a table visible to all users, and other columns visible to only a select group. In addition, a database can be instructed to filter a table's rows so that some users see only certain rows, while other users see all rows.

A database's security features go even further. In addition to controlling who can see what information, a database allows you to specify who can insert new information, update existing information, or delete information. This helps ensure that people who have no business reason to change or delete data, for instance, cannot do so accidentally (or not so accidentally).

In a large database system—say, one with 1,000 users or more—managing all of these different kinds of privileges would quickly become impossible if they had to be set on a user-by-user basis. Fortunately, databases like Oracle allow you to gather a specific set of privileges into something called a *role*. That way, whenever users are added to the database, they are assigned one or more roles, and those roles carry the privileges the users can exercise. This works well because businesses generally have specific job descriptions, and the privileges each user will need relate directly to his

or her job description. An accounting clerk will need the ability to enter data from bills, but perhaps only the accounting managers will have the ability to change data once it's been entered. Similarly, perhaps only accounting executives can do anything at all with the company's salary data. Each of these three job descriptions would be a good candidate for a database role. For instance, an "accounting clerk" role would be assigned to all of the accounting clerks. Security roles help ensure that everyone has exactly the privileges they need. Roles also make it very easy to assign a new privilege to a group: You just add the privilege to that group's role, and the job is done.

Relational Abilities

Since a database employs separate tables to store different types of data—remember the example of a school's database having individual tables for faculty, classes, classrooms, and students—there must be a way to connect records in one table with relevant records in the other tables. Databases accommodate this by letting you define *relationships* between the tables.

For example, let's consider an order-entry system. The core of this type of system is the business' inventory, so there will always be a table containing information about products. The PRODUCT table will store each piece of information pertaining to an inventory item, including description, manufacturer, price, and quantity currently in stock. The PRODUCT table will also store a unique identifier for each product, as a way of unquestionably identifying one product as opposed to another. Let's say for the sake of discussion that in the PRODUCT table, the unique identifier is the Stock Keeping Unit (SKU).

Now that we have a table in which to store products, we need another table in which to store orders for those products. Each row in the ORDER table will store the date, time, location, and total order value. The ORDER table must also identify what product the order is for, and it can do this simply by storing the product's SKU as part of the order record. Here's where the relationship comes in: *The only product information an order contains is the product's SKU*. The product's description, price, and other information are not stored in the ORDER table. Why? It would waste space, among other reasons, because each product's description, price, and so on are already available in the PRODUCT table. The only requirement to making this work is that the database must be told how to relate an order's SKU to a unique record in the PRODUCT table. Once it knows that, the database can join information from both tables, and present the combined information on a single line, as if it came from one table.

A database that employs this technique of relating records in separate tables is called a *relational database*. It's not uncommon for business databases to contain tables that have relationships to dozens of other tables. There are many reasons for doing this, and they will be discussed in depth in Chapter 6. The opposite of this approach is a single large table that repeats information every time it is needed. This type of table is called a *flat file*, indicating that it is two-dimensional—just rows and columns, no related tables.

Constraints to Ensure Data Quality

Sometimes the data stored in a database comes directly from other machines: automated sensors, timers, or counters. Most of the data in a database, though, is entered by people. And people make mistakes. (Not you, of course, but I'm sure you know others who do.) When designing a database, it's easy to define *constraints* identifying conditions that data in a particular field must meet before the database accepts the record. The constraint defines what must be true about the data in order for it to be accepted. These constraints can be very simple—like ensuring that a price is a positive number—or more involved, like ensuring that a SKU entered into an order actually exists in the PRODUCT table, or requiring that certain fields in a record be entered if other fields contain specific values. By automating these types of quality-control functions, the database helps guarantee that the data it contains is clean.

Review Questions

1. What is the most significant characteristic of a table?
2. Define the following terms: *row*, *record*, *column*, *field*, *table*, *database*, *constraint*.
3. What are the key features that make a database suitable for storing large amounts of data?

Hands-On Project

1. Gather four blank sheets of paper. The sheets can be any size—index cards will work as well as 8.5" × 11" sheets.
2. On each sheet, write the first name, last name, phone number, and e-mail address of a friend or associate (you will do this for a total of four people—one per sheet).
3. After you have written the information onto the sheets, lay the sheets on a table—separately, so that they do not touch each other.
4. Move the sheets right next to each other, so they form a grid that is two sheets wide and two sheets high. In this arrangement, each sheet will touch two other sheets.
5. Tape the sheets together in this arrangement.
6. Take a pen that is a different color and on the top-left sheet, circle each item that would be stored in a database field. Write the word "Fields" at the bottom of the sheet, and draw arrows to each of the circled fields.

7. Next, put a rectangle around each item in the four-sheet grid that would relate to a row in a database. Write the word “Rows” in the center of the four-sheet grid, and draw arrows from the word to each square.
8. Finally, write “e-mail column” wherever on the four-sheet grid you have some space. Then draw one long line that connects that phrase to every item in the grid that would be stored in a table’s e-mail column.

How Will Knowing This Help You?

Everybody is busy these days, and if you are reading a book about PL/SQL, you are probably busier than most. It’s reasonable that you will want to know how something is going to help you before investing the time to learn it. What follows is a list of the ways that learning PL/SQL can help you in different situations.

When Doing Database Administration

It’s impossible to be an Oracle database administrator (DBA) without knowing the standard database language called Structured Query Language (SQL), and very difficult without knowing Oracle’s SQL superset named PL/SQL. This is because many of the tasks that are generally done by a DBA are accomplished using SQL, and quite a few require the programming capabilities of PL/SQL, as well. While Oracle does provide a number of software programs enabling administrators to perform tasks using a nice graphic user interface (GUI), these tools have their fair share of bugs. In addition, some tasks can just be done faster when using SQL directly, and in some database installations, connections to databases are accomplished using text-based terminals that cannot run the pretty GUI tools. The importance of SQL is reflected in the fact that Oracle’s own DBA certification program, which consists of five separate exams, devotes the entire first exam to SQL and PL/SQL.

When Developing Software

When writing programs of your own, the chances are good that at some point you will need to write some SQL code to interact with a database directly. Many developers stumble through this, making mistakes that cost them time and performance. Investing the time to learn SQL properly will pay for itself many times over. The documentation supplied with Oracle includes sections dedicated to interacting with Oracle via Java, C, C++, and COBOL.

When Doing Business Analysis

Being able to slice and dice huge mounds of data into the information you need is an essential part of a business analyst’s job. Lots of people do this by getting an extract of their company’s database, putting it in a spreadsheet, and manually creating the analyses they need. This approach is flexible, but it can be time-consuming. Some

companies also provide their analysts with software tools designed to make data analysis quick and easy. But even these tools don't provide every imaginable way of looking at the information; they provide the subsets that their designers think are the most likely to be used. The chances are good that you will want to look at the data in some way that the tool doesn't support. Often a single SQL query can provide the information you need.

If You Just Want to Know How to Use Databases Better

Recently I visited Silicon Valley and, when I had some free time, decided to go to a movie. I purchased a local newspaper and turned to the movie listings. I was struck by what I saw: *two* sets of listings, one sorted by geographic area, and the other sorted by movie title. So if you wanted to find out what was showing near you, you would use one listing; if you wanted to find out where a specific movie was playing, you would use the other. This simple, powerful idea made the listings a real pleasure to use. I'm positive that it was thought up by someone with database experience, someone who was familiar with the idea that the content of the data and how it is displayed are two different things.

These days, practically everything is built around a database. If you understand how databases work, you understand how a lot of businesses function. This can be extremely useful. For instance, if you call a company's customer service department but don't have your customer number with you, you might think to ask, "What else can you search on to find my record?" When you use a web search site to locate information, you will get the results you want much more quickly if you understand how databases interpret search terms. (My friends are regularly amazed at how quickly I can find relevant information using search sites. My only trick: educated decisions about what to enter as search criteria.) Understanding databases is becoming a lot like being able to do basic math quickly in your head: It isn't essential, but it sure comes in handy a lot.

History of SQL

A little bit of history is useful to give perspective, and the history of SQL parallels the history of relational databases. In 1969, Dr. Edgar F. Codd published an IBM Research Report with the catchy title *Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks*. This paper described an approach to structuring databases with related tables, which was quite different than the flat-file approach used by databases at the time. The paper was stamped with IBM's Limited Distribution Notice, so it wasn't widely read. Codd revised the concepts and published them in a 1970 article titled "A Relational Model of Data for Large Shared Data Banks" in the journal of the Association of Computer Machinery. The relational model described by Codd was used in a prototype relational database management system (RDBMS) called System R in 1974. Describing the system's query language in a November 1976 article in

the *IBM Journal of R&D*, IBM used the name Structured English QUery Language (SEQUEL). The language and its name evolved, becoming Structured Query Language (SQL, pronounced either “sequel” or “S-Q-L”). The first commercially available version of SQL was released in 1979 by Oracle Corporation (although the company’s name was Relational Software, Inc. at the time).

In 1986, the American National Standards Institute (ANSI) stepped in and published a formal SQL standard, which it identified as ANSI X3.135-1986. The International Standards Organization (ISO) picked up this standard the following year and published it as ISO 9075-1987. The specification was expanded in 1992, and then again in 1999. The current specification is in five parts, named ANSI/ISO/IEC 9051-1-1999 through 9051-5-1999.

SQL has become the de facto standard language for querying databases. Each database company modifies it a bit to suit its needs, but the core SQL functions remain essentially unchanged. This is good news for database users and developers, because the time invested in learning SQL will reap benefits for years to come, across software revision after software revision, and even to other products.

The bottom line is: SQL is a versatile and essential tool for anyone who works with databases regularly.

SQL Command Categories

SQL commands fall into functional groups that help make them easy to remember. These groups include:

- Data Definition
- Data Manipulation
- Data Control
- Data Retrieval
- Transaction Control

Your work with these commands will start in Chapter 2, and will continue throughout the book. Here’s an overview of the command categories you will learn to use.

Data Definition

Oracle and all major database programs are database platforms—meaning they provide an environment that supports working with tables very well, but they don’t provide any tables already created for your use. You get to define what data will be stored and in what configuration. SQL provides a collection of commands for these purposes: CREATE, ALTER, DROP, RENAME, and TRUNCATE.

These commands fall into a group called Data Definition Language, which is routinely referred to as DDL.

Data Manipulation

Okay, you learn how to create some tables. What's the next step? Putting data into them. SQL provides an INSERT command enabling you to add data to tables. After the data has been inserted, you can change it using the UPDATE command, or remove it using the DELETE command.

This category of commands is called SQL's Data Manipulation Language, more often referred to as DML.

Data Control

Remember the security features discussed earlier in this chapter? (I'm sure you do, but if anyone reading over your shoulder doesn't remember it, it's in the section titled "How Is a Database Different from a Spreadsheet?") The ability to let some users utilize particular tables while other users cannot is enforced by assigning users *privileges* for specific tables or activities. An *object privilege* allows a user to perform specified actions on a table (or other database object). An example of an object privilege would be the ability to insert records into an EMPLOYEE table. In contrast, a *system privilege* enables a user to perform a particular type of action anywhere in the database. An example of a system privilege would be the ability to insert records into *any* table in the database, or the ability to create a new table.

Database privileges are assigned and removed using the SQL commands GRANT and REVOKE. These commands fall into the category of Data Control Language (DCL).

Data Retrieval

The whole point of putting information into a database is getting it out again in a controlled fashion. There is just one command in this category—SELECT—but it has a wealth of parameters that provide a *lot* of flexibility. The SELECT command is the command you're likely to use more than any other, especially if you plan to use SQL from another programming language.

Transaction Control

Oracle's SQL provides an undo capability enabling you to cancel any recent DML commands before they are applied to the database. (Quick quiz: What commands are DML commands? If you need a reminder, take another look at the section titled "SQL Command Categories" earlier in this chapter.) After performing one or more DML commands, you can either issue a COMMIT command to save your changes to the database, or issue a ROLLBACK command to undo them.

The undo capability provides multiple levels, too: You can reverse just the last DML transaction, or the last several, or whatever level you need. Taking advantage of this multiple-level redo takes a little more forethought than it does in your favorite word processor, however. If you want to be able to undo to intermediate points, you have to mark those points by issuing a SAVEPOINT command at whatever point you want to be able to roll back to.

Chapter Summary

Stripped down to its most basic form, a database is a list of information. Or a set of lists that work together. A database program is a fancy list manager. Familiar databases include telephone books, checkbook registers, and web sites providing online auctions, ordering, and searching.

Databases are always designed to store a particular type of information. For instance, in the case of a telephone book, the information is about people (in the white pages), and about businesses (in the yellow pages). A database would generally store such information by having one table containing all the information about people, and another table containing the information about businesses. Each of these tables would look a lot like a spreadsheet, with individual columns for each type of information being stored (name, address, number, and so on) and a row for each person or business.

The most important thing to remember about a table is this: *It stores information about **one** type of thing.* If a database needs to store information about more than one type of thing—and it almost always will—it does so by using more than one table. Keeping the different types of information separate allows a database to store information very efficiently, and in a highly organized (and therefore easy-to-use) manner.

Each line in a table contains information about one instance of whatever the table is designed to store. Each line is called a *row*, and the data it contains is called a *record*. You design the table so that it only takes one row to hold all of the information that is specific to each item the table contains.

Each row contains several pieces of information. In a table, these pieces of information are stored in *columns*. The junction point of a row and a column—for instance, a particular person's first name—is called a *field*. A field contains a single piece of information about something—for example, the last name for one person.

While a table in a database stores data in rows and columns like a spreadsheet, there are many characteristics that make a database more appropriate when the data gets more complicated, or when it needs to be used in a more sophisticated environment. These include the ability to handle billions of rows, accommodate thousands of simultaneous users, provide object-specific security, relate many tables together, and constrain the content of incoming data to ensure quality information.

Understanding how to use SQL can benefit you in a number of different areas. It's an essential skill if you're planning to be an Oracle DBA, because many DBA tasks are executed using SQL commands. When developing software, it's likely you will need to write SQL commands to insert, select, update, and delete data within programs you write. When doing business analysis, knowing SQL enables you to interact directly with the database, slicing and dicing its information the way you want, without being limited by pre-designed queries created by someone else.

And if you just want to know how to use databases better, understanding SQL will help you understand how to use a variety of products and services used in daily life.

SQL is based on concepts pioneered by Dr. Edgar F. Codd and first published in 1969. It has become the de facto standard language for interacting with all major database programs. Its commands fall into functional categories, which include data definition, data manipulation, data control, data retrieval, and transaction control. The Data Definition Language (DDL) commands are used to define how you want your data to be stored; these commands include CREATE, ALTER, DROP, RENAME, and TRUNCATE. The Data Manipulation Language (DML) commands enable you to work with your data; they include INSERT, UPDATE, and DELETE. The Data Control Language (DCL) commands control who can do what within the database, with object privileges controlling access to individual database objects and system privileges providing global privileges across the entire database. The DCL commands include GRANT and REVOKE. For data retrieval, SELECT is the sole command, but its many variations are likely to make it the most-used command in your arsenal. To control transactions, SQL provides the COMMIT command to save recent DML changes to the database; the ROLLBACK command to undo recent DML changes; and the SAVEPOINT command to let you undo some, but not all, of a string of DML commands.

Chapter Questions

- 1. Which of the following are examples of databases that you're likely to encounter in daily life?**
 - A. Front page news
 - B. Telephone books
 - C. Checkbook registers
 - D. Web sites providing online auctions, ordering, and searching
 - E. Ads with sale prices
 - F. Movie listings
- 2. What is the most significant characteristic of a table?**
 - A. It has rows and columns.
 - B. It can be related to other tables.
 - C. It stores information about one type of thing.
 - D. It contains records and fields.

3. Relate each term on the left with the appropriate description on the right.

| Term | Description |
|-------------|---|
| Row | Stores all information about one type of thing (for instance, people or products) |
| Record | Defines what must be true about the data in order for it to be accepted by the database |
| Column | One line in a table |
| Field | Collection of one type of information stored in a table (for instance, all of the phone numbers or all of the last names) |
| Table | Data contained in a table row |
| Database | Contains a single piece of information about something |
| Constraint | Collection of related tables |

4. Which of the following are reasons why a database is the best choice for handling large quantities of business data?

- A. Can handle billions of rows
- B. Runs only on PCs
- C. Can accommodate thousands of simultaneous users
- D. Provides object-specific security
- E. Can relate many tables together
- F. Allows you to define constraints defining conditions that data must satisfy before it is accepted into the database

5. Whose work pioneered relational database theory?

- A. E. F. Skinner
- B. Edgar Winter
- C. Edgar F. Codd
- D. Edgar Piece

6. Relate each SQL command category on the left with the appropriate commands on the right.

| SQL Command Category | Commands |
|----------------------------------|---|
| Data Definition Language (DDL) | GRANT and REVOKE |
| Data Manipulation Language (DML) | SELECT |
| Data Control Language (DCL) | CREATE, ALTER, DROP, RENAME, and TRUNCATE |
| Data Retrieval | COMMIT, ROLLBACK, and SAVEPOINT |
| Transaction Control | INSERT, UPDATE, and DELETE |

Answers to Chapter Questions

1. B, C, D, F. Telephone books; checkbook registers; web sites providing online auctions, ordering, and searching; and movie listings.

Explanation The essence of a database is a list that can be presented in a chosen order, and filtered to display only selected records. Front page news does not fit this description; it cannot be sorted or filtered. The same is true for ads. However, the other categories do fit this description.

2. C. It stores information about one type of thing.

Explanation Choices A and D are true, but they are also true of spreadsheets, so they cannot be a table's most significant characteristic. B is also true, but its importance does not compare with choice C. The single most important characteristic of a table is that it stores information about one type of thing.

| 3. Term | Description |
|----------------|---|
| Row | One line in a table |
| Record | Data contained in a table row |
| Column | Collection of one type of information stored in a table (for instance, all of the phone numbers or all of the last names) |
| Field | Contains a single piece of information about something |
| Table | Stores all information about one type of thing (for instance, people or products) |

| Term | Description |
|------------|---|
| Database | Collection of related tables |
| Constraint | Defines what must be true about the data in order for it to be accepted by the database |

4. A, C, D, E, F. Can handle billions of rows, can accommodate thousands of simultaneous users, provides object-specific security, can relate many tables together, allows you to define constraints defining conditions that data must satisfy before it is accepted into the database.

Explanation Each of the answers provided is a powerful reason for using a relational database to handle large quantities of information, with the exception of B. Large databases do not run only on PCs—for instance, Oracle is available for computers running Unix, Linux, Windows NT, and a variety of other operating systems. Even if it was available only on PCs, that would be a liability, not an asset, because the other operating systems are designed for industrial-strength use and tend to be more stable than PC-based systems.

5. C. Edgar F. Codd

Explanation Dr. Edgar F. Codd published ground-breaking papers about relational database theory in 1969. He is widely regarded as the father of relational database design.

| 6. SQL Command Category | Commands |
|----------------------------------|---|
| Data Definition Language (DDL) | CREATE, ALTER, DROP, RENAME, and TRUNCATE |
| Data Manipulation Language (DML) | INSERT, UPDATE, and DELETE |
| Data Control Language (DCL) | GRANT and REVOKE |
| Data Retrieval | SELECT |
| Transaction Control | COMMIT, ROLLBACK, and SAVEPOINT |