CHAPTER 1

Basic Concepts

In this chapter, I'm going to introduce you to the concepts behind databases and why databases are important to your application program. I'll also cover different database architectures and follow that up with a brief overview of the database systems to be covered in this book.

In This Chapter

Why use a database?

Database architecture

Common databases

Why Use a Database?

A *database* is simply a general-purpose tool that allows you to store any information you want that can be read and updated by one or more concurrent application programs in a secure and reliable fashion. Note that while this definition describes conventional database systems such as SQL Server and Oracle 8*i*, it can also be used to describe a collection of normal disk files. I want to explore these concepts using conventional disk files to help you better understand why your application needs a database system.

Storing information

There are many ways to store information on a computer, but eventually all of them boil down to placing information in a file somewhere on a disk drive. There are two basic types of files, which are characterized by how they are accessed. *Sequential access* files are accessed as a single unit, while *random access* files allow you to directly access the file in small pieces.

Using conventional file access

In order to process the data in a sequential access file, you must read or write the entire file. This type of access is useful for a word processor where you want to make all of the information in the file available to the user at the same time.

Generally a program will load the entire file into memory, make whatever changes are necessary, and then save the entire file when the user is ready to exit the program.

When dealing with sequential access files, you need not load the entire file into memory. Indeed, many programs merely load enough data from the file to perform a given task and load more data when the task is complete. The main idea is that your program reads the data from the beginning of the file to the end in a linear fashion. Hence the term *sequential*.

In most application programs, you only want to access a subset of the data at a given point in time, so rather than processing the entire file just to get a small piece of data, a random access file allows you to divide the file into chunks called *records*. (See "Organizing data in records," below for details.) Each record has a unique record number, which is determined by its relative position in the file. You can retrieve or update any record in the file by specifying the appropriate record number.

The key to making random access files work is that each record in the file is always the same size. This allows you to determine the location of the record by computing the starting byte of the record, just by knowing the size of the record and the relative record number in the file.

Even though a random access file may be organized differently than a sequential access file, you can also process a random access file sequentially by reading the records in order of their record numbers. This implies that you don't lose any functionality of a sequential access file if you choose to create your file as random access.

Organizing data in records

A record represents a chunk of data contained in a file and is organized into one or more *fields*. Each field has a specific name and *data type* associated with it. The name is used to uniquely identify the field in the record, and the data type describes what kind of information the field can hold.

See "Columns and Data Types" in Chapter 2 for more information on this topic.

Because each record in the file always holds the same information, you can think of a file full of records as a table or grid, where each record corresponds to a row and each field corresponds to a column (see Figure 1-1). The table concept also conveys the concept of how the information is actually stored in the file, since the relative position of each field is fixed in relation to the other fields and the relative record number uniquely identifies the record's place in the table.

Name	Address	City	State	Zip Code
Christopher James	1234 Main Street	College Park	MD	20742
Samantha Ashley	2345 Central Avenue	College Park	MD	20742
Terry Katz	3456 Pennsylvania Ave	College Park	MD	20742
Robert Weiler	4567 Redwood Way	College Park	MD	20742
Wayne Freeze	5678 Baltimore Street	College Park	MD	20742
Bonnie Jean	6789 Oak Street	College Park	MD	20742
Jill Heyer	7890 Washington Drive	College Park	MD	20742
Raymond Bucky	8901 Souix Circle	College Park	MD	20742

Figure 1-1: A file with records looks like a grid or a table.

Concurrency

One of the most critical issues to consider when dealing with an application is ensuring that two or more concurrently-running programs don't interfere with each other while processing data. If the proper steps aren't taken, it would be easy for one program to destroy information that another program changed in the file. For instance, assume that you have a random access file that contains customers' checking account balances. Then assume that you have one program that debits the account for checks that the customer wrote, while another program credits customer deposits. While this scenario is unlikely, it is possible that the two programs may attempt to update the same customer record at the same time.

Now, consider the following sequence of events. The debit program reads the customer's record. Before the debit program has a chance to update the information, the credit program reads the customer's record and adds the deposit to the account's total balance. After the credit program updates the customer's information, the debit program completes its process by writing its updated account balance back to the file. In this scenario, the credit to the account was lost, since the update done by the debit program used an out-of-date account balance.

The proper way to prevent the concurrency problem is by controlling access to the data. Typically, this is done by *locking* the file to prevent other users from accessing any of the data in the file you want to use. The sequence of events in the above scenario would now look like this: The debit program would lock the file. Then it would read the customer record. The credit program would attempt to lock the file. Since the debit program already locked the file, the client program would be suspended. The debit program would complete its update and then release the lock on the file. The credit program would be resumed and the lock would be granted. It would then perform its update and release the lock, thus ensuring that the correct balance would be in the file.

Securing your data

In most computer systems, you can identify the users who are able to read or update a particular file, and deny access to all others using normal operating system security. However, it is often desirable to allow someone to see only part of a record — in a case, for instance, where you want to keep an employee's salary information in the same record with the employee's address and telephone number.

To prevent people from seeing salary information, you have to create another file that contains information about each person who can access your file and which fields they are permitted to access. Then your application programs must properly use this information to prevent unwanted access to data.

Of course this method is only as reliable as your application programming staff. One small mistake in a program could allow a user to read and/or change the value that they shouldn't have access to.

Performing backups and using transaction logs

No matter what you do to your data, you need to ensure that once a change has been made to it, it won't be lost. After all, it is possible that your hard disk could crash and destroy your files or your computer could suffer a power failure while the files were being updated.

There are a number of different ways to prevent your data from being lost. The most common practice is to back up your data on a regular basis. Of course you can't permit any users to update data while you're doing this, or your data files may be in an inconsistent state because someone changed a value in one file before the backup process copied the one file and after the backup process copied another file. Alternate ways to prevent you from losing your data include using redundant disk drives and/or database servers so that your data will still exist even if you lose a disk drive or server. However, I strongly recommend that you back up your data even if you use redundant equipment, just in case you lose both your primary copy and your redundant copy.

Most applications back up their data once a day, which limits the amount of data you might lose to a maximum of a single day. But losing even half a day's worth of information can be a big problem in most situations. For instance, in the case of a mail-order processing application, you always have the original order documents that you can reenter. However, in the case of a telephone order processing application, you may not even know which customers placed an order if you lose your files, and even if you did, do you really want to call all of your customers back to find out what they ordered?

There are a couple of solutions to this problem. First you can print every change you make to the files on a printer. This way, you will always have a paper record of the changes, and can reenter the information if you need to.

A better way would be to write all of your changes to a special file known as a *transaction log.* This file includes only the changes that were made to the file. Thus, if you have to restore the file from the previous day's backup, you could run a program that would read the values from the transaction log and reapply them to the main file.

Both of these approaches aren't 100% reliable, since the application programmer must explicitly send the information to the printer or write the changes to the transaction log. If the programmer forgets to include the appropriate calls in their program, the information is pretty much useless.

A database is the answer

While you can use files to hold your organization's data, you can see that there are a number of problems associated with this practice. A database system solves these problems and others that you may not have thought of.

Tables, rows, and columns

Recall that a database holds information in tables that correspond to random access files. Each record in the file corresponds to a row in the table. Each field in the file corresponds to a column in the table. Unlike a record, you can choose which columns you wish to retrieve from a table. By retrieving only the columns you need, you help to isolate your program from changes in the table. This means you can add and delete columns from a table, and as long as your program doesn't reference any of these columns, your program will continue to work without change.

The concept of a relative record number isn't available to locate a row in a table. It has been replaced with a more powerful concept known as the *primary key.* The primary key is a set of one or more fields whose values will uniquely identify a row in a table. Where you may have used an employee number or part number as the relative record number in a random access file, you can now use an arbitrary set of values to locate the row. This gives you a lot more flexibility when designing a table.

Locking

Just as you use locking to prevent two programs from accessing the same data in a file-oriented application, the database management system uses locking to prevent the same thing from happening. The database system automatically determines how to use locks to prevent two or more programs from accessing the same data at the same time. Unlike file-oriented applications, where the entire file is locked, the database system is smart enough to allow programs to update data in the same table at the same time as long as all of the programs are all accessing different rows.

You can get a big performance boost in your application because a database permits multiple programs to access data in the same table at the same time. With file-based applications, one program had to complete the read, process, and update sequence before the next program could start. This translates into longer and longer delays as more and more users try to access the files at the same time. Since the database system permits multiple, non-conflicting read, process, and update sequences to be active at the same time, the wait time is no longer a factor.

Security

Database systems are designed from the ground up to be secure. You can specify which users can access which data in which fashion. Since the security is moved outside the application, it is much easier to verify that only properly authorized users can access a piece of information.

Restoring lost data

Unlike a collection of random-access files, a database system has a comprehensive mechanism to ensure that once data has been written to the database, it isn't lost. A database system includes tools that will back up the database in a way that will prevent the problem where some of the changes are captured, while others aren't. This means that you can always restore your database to a consistent state.

Also, a database system includes an integrated transaction log that will automatically capture all of the changes to the database. When you have to restore a database, the restore process can automatically process the transaction log. All of the transactions written to the log will be recovered, which means that the data will be correct as of the moment before your disk crashed.

Database Architecture

Nearly all database systems available on the market today are implemented using a *client/server* architecture. This architecture defines two types of programs and how they interact with each other.

Servers and clients

A *client* is a program that generates requests that are sent to another program, called a *server*, for execution. When the server has finished the request, the results are returned to the client. In today's computing environment, many applications are implemented using client/server technology. For instance, your Web browser is a client program that talks to a Web server. A file server contains files that are made available to you over a network. Likewise, a print server allows a network manager to share a single printer with many different users.

Servers, clients, hardware, software, and confusion: The terms *client* and *server* are often used to describe both the software applications and the hardware they run on. This can lead to some confusion. It is quite possible that you may use a Web browser (client) application to access a Web server on the same computer. For example, I frequently test new Web pages on a test Web server running on the same computer as the browser before uploading them to a production Web server. In general, when I use the terms client and server I'm referring to the software applications and not the physical machines they run on.

Database servers and database clients

A *database server* is a program that receives *database requests* from a *database client* and processes the requests on the database client's behalf (see Figure 1-2). A database client is a program that generates database requests by interacting with a user, processing a data file or in response to a particular event in your computer. A database request is a specific operation that is to be performed by the database server, such as returning data from a table, updating one or more rows in a table, or performing some other database management task.

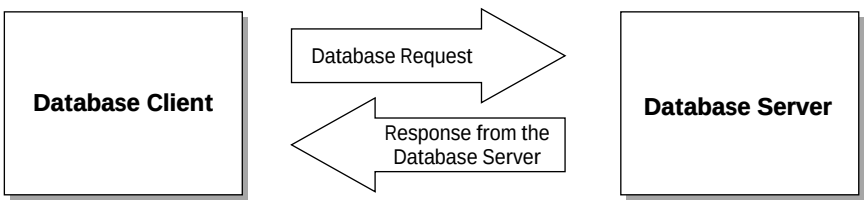

Figure 1-2: Database clients send database requests to a database server for processing.

Database servers

A database server typically runs on its own computer system and receives database requests across a network. If the request generates a response, then the response is returned to the database client computer over the network. This arrangement isolates the database server from each of the database clients, which improves the performance and reliability of the database system.

A special software package known as the *Database Management System* (DBMS) is run on the database server. It is this software that receives database requests, processes them, and returns the resulting information back to the database client.

The database server is typically run on an operating system designed to support servers, such as Windows 2000/NT Server or any of the various flavors of Unix. Desktop operating systems such as Windows 98/95 don't provide the reliability or stability necessary to run a database server.

In small environments, you can run several different servers on your server computer — such as a file server, Web server, mail server and transaction server — in addition to your database server. However, as your workload increases, it will become desirable to dedicate computers to each of these functions.

This type of arrangement would allow you to scale the size of each computer system to accommodate your workload. You can add more memory, faster CPUs and more disk space to keep pace with increases in your workload. Also, by dedicating a computer to running the database server, you run less of a risk of a system crash because of a software problem in another server such as the transaction server.

Database clients

The database client is simply an application running on the same network as the database server that requests information from the database. The application program generates database requests using an *Applications Programming Interface* (API), which is nothing more than a set of subroutine calls or set of objects that your program uses to send requests to the database and receive information from the database.

The API allows the program to communicate with more than one database system through the use of a special piece of code known as a *database driver*. A database driver represents a special program that is installed on the database client computer. It translates the standardized database requests made using the API into the special language used by the specific database server.

By using database drivers, the same application program can communicate with different database systems without changing the application itself. This level of independence is important since it allows an organization to replace one database system with another with only a minimal impact to the applications themselves.

Simple database systems

While most database systems fall into the client/server architecture I described above, a few simple database systems work differently. These are typically low-end database systems where performance and scalability issues are not a problem. These systems still communicate with your application via a standard API, but instead of passing requests onto a database server on a different computer, they pass the requests to code residing in the client computer.

This database code in the local client computer is designed to cooperate with other copies of the database code running in other computers to access a common file or set of files located somewhere on the network. The database files could be located on the same computer as one of the database clients, or located on a central file server. A low-end database system is also a good choice if you are designing a stand-alone application for a single user.

Administration of these database systems is fairly easy. Backing up the database is accomplished by backing up the file containing the database, and recovery processes are limited to restoring a backed-up version of the database.

Of course this approach can have a big impact on performance, because locking must now be done at the file level rather than based on the values selected from the database. While this will obviously hurt an application with many users, the overhead is low enough that for a handful of users, the performance may actually be better than the client/server architecture.

Types of Databases

There are four basic types of database systems: hierarchical, networked, indexed and relational. While each of the four types has many similar concepts, their differences result from how they store their data.

Hierarchical databases

The *hierarchical database* is the oldest form of database in existence. Data is arranged in a series of tree structures, which often reflect the natural relationship between data. IBM's IMS database, which is still available today, is the classic example of a hierarchical database.

A hierarchy models a data relationship known as *1 to many*. A 1 to many relationship means that one data value is related to one or more other data values. The classic example for this type of relationship is students and classes. When students attend school, they take many classes, a situation which can be described with a natural hierarchy (see Figure 1-3).

Networked databases

The *networked database* was developed as an alternative to the hierarchical database. While a lot of data can be organized using a hierarchical relationship, it is difficult to model other data relationships. Consider the case of students and their teachers. A student will take courses from many different teachers, whereas a teacher will teach more than one student (see Figure 1-4). This relationship is called *many to many*.

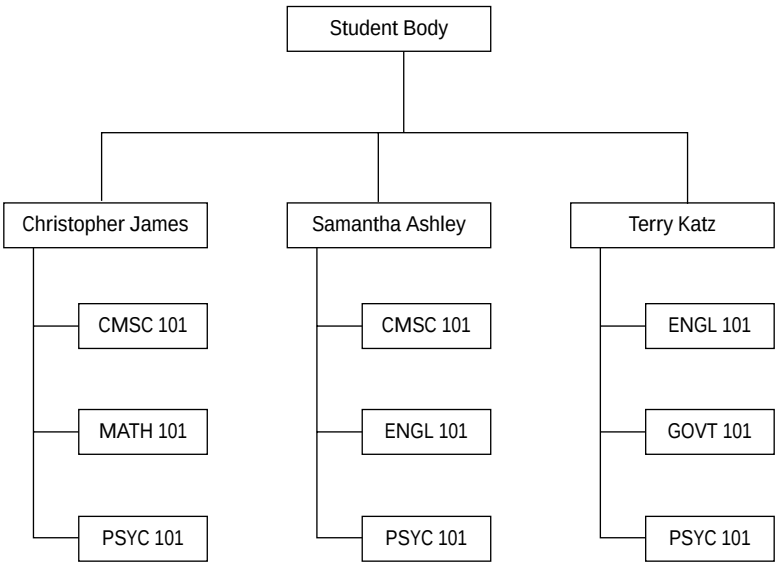

Figure 1-3: A hierarchical database reflects the natural relationship betw een data.

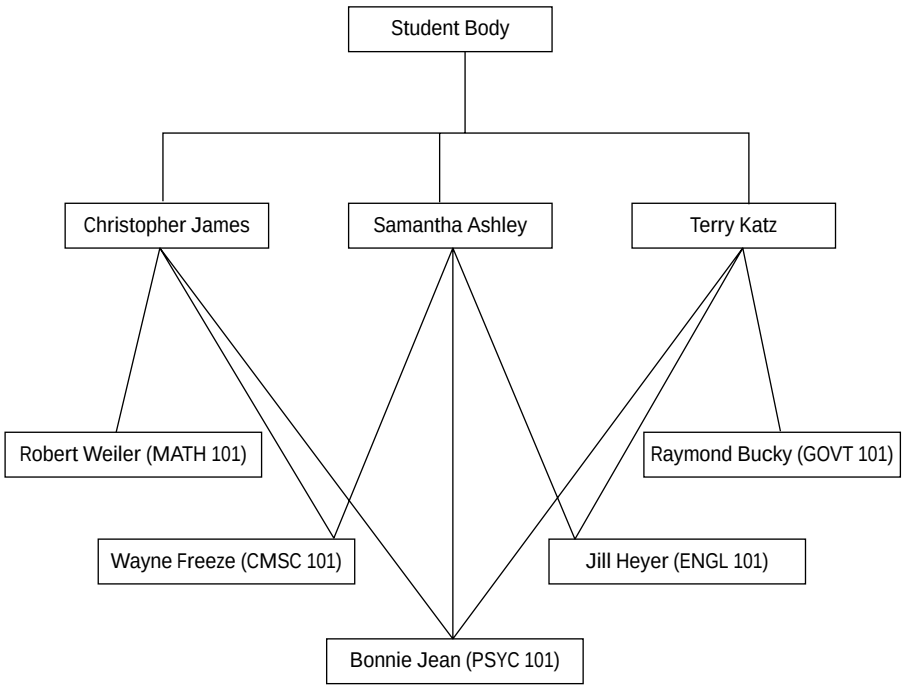

Figure 1-4: A networked database can handle situations that a hierarchical database cannot.

A networked database stores information in datasets, which are similar to files and tables. A record on one dataset that is related to a set of records in another dataset will have a set of physical pointers, also known as a link. Thus the record in the first dataset is linked to the records in the second dataset. The primary drawback to networked databases is that it can be quite complicated to maintain all of the links. A single broken link can lead to enormous problems in the database.

Indexed databases

While similar in concept to a networked database, an *indexed database* replaces links with an *inverted list.* An inverted list is another file that contains a list of unique values found in a particular field in a dataset, along with pointers to each record in which the value appears. This value is known as a *key value.* If you know the key value, you can quickly list all of the records in the dataset that contain that value. In practice, an indexed database is the most efficient database system in the marketplace today.

Relational databases

The concept of the *relational database management system* (RDBMS) goes back to the early 1970's, when IBM researchers were looking for a better way to manage data. One of the problems with hierarchical, networked, and indexed databases is that they all require pointers to connect data together. The researchers built a mathematical model, which discarded the pointers and used common values to link multiple records together. Thus, you weren't limited to hierarchical structures as in the hierarchical database model, and you didn't have to set links as in the networked database model. (The indexed model evolved after the original work was done on the relational database model.)

The primary drawback to the relational database model is that a relational database is very slow when compared to the other database models. Without pointers to help you quickly locate a particular record, you may have to read every record in a table to find one particular record. To help combat this problem, the concept of an *index* was introduced. Even with indexes, relational databases are generally less efficient than databases implemented using the other database models. However, the relational database gained popularity as computing began to shift from traditional mainframe computers to smaller mini-computers and personal computers. The initial applications of relational databases were far less demanding than the database systems running on mainframes, so the performance issues were not as much of a factor as they could have been. As time went on, the speed of mini-computers and personal computers increased to the point where they rivaled mainframe computers.

See Chapter 2 for a more in-depth discussion of relational databases.

Common Databases

In today's marketplace there are a number of relational database systems you can use. In this book, however, I'm going to focus on only three: Microsoft SQL Server 7.0, Microsoft Jet 3.5/4.0 and Oracle 8*i*. These three database systems make up the majority of the database systems in the world today, at least from the Visual Basic programmer's perspective. Although there are other database systems that run in the mainframe world, such as CA's IDMS and IBM's IMS, they aren't typically used by Visual Basic programmers.

SQL Server 7.0

One of Microsoft's major design goals with SQL Server 7.0 was to create a scalable product that ran the same on a Windows 98 system with limited memory and disk space, up to a large Windows 2000/NT Server system with multiple processors, gigabytes of main memory, and lots of disk space. To accomplish this, Microsoft created three versions of SQL Server. The Desktop Edition runs on Windows 98 and Windows 2000/NT Workstation. It's designed to handle smaller databases and is ideal for helping developers test their programs away from the production database server. The Standard Edition is the most common edition and provides the features that you really need for most applications. The Enterprise Edition expands the Standard Edition by adding features that help SQL Server handle applications with a lot of data and a large number of transactions.

Besides focusing on scalability, Microsoft also worked hard to reduce the total cost of ownership for a database system. Modern relational databases are often served by a group of specialists known as *database administrators*. These people are expensive to hire and difficult to keep, due to the increasing demand for their skills. SQL Server 7.0 addresses this problem by including a number of wizards that make it easy to perform routine tasks. Also the SQL Server Agent can be used to run various activities when your machine is left unattended.

A good primer: See *Microsoft SQL Server 7 for Dummies* by Anthony T. Mann for more detailed information about how to administer an SQL Server 7 database server.

Microsoft also addressed data warehousing applications by including two key features in SQL Server 7. The first is the *Data Transformation Services* (DTS). This feature makes it easy to move data from one place to another. This is particularly important, since the data in a data warehouse is usually taken from tactical applications like accounting and inventory systems, even if they exist on the corporate mainframe.

The second key feature is *OLAP Services* (OLAP stands for online analytical processing). These services bridge the gap between the data warehouse and the analysis tools running on the user's workstation. The data warehouse data is preprocessed in the OLAP server before the results are presented to the analysis tool. Microsoft has made the *application programming interface* (API) to OLAP Services available, so that other companies can design client tools to analyze OLAP data.

Microsoft Jet 3.5/4.0

Microsoft Access is a simple desktop database application development tool that is targeted at small- to medium-sized organizations. At the heart of Access is a true relational database engine known as Microsoft Jet. Like SQL Server, Jet is based on the SQL standard. Unlike SQL Server, the database code runs in the application itself rather than in a database server. Also, because Jet was developed independently of SQL Server, it isn't totally compatible with SQL Server.

Even though Jet is developed as part of Microsoft Access, Jet is included with Visual Basic. You can develop applications using Jet and freely distribute the Jet runtime code with your application. You can't do this with SQL Server applications.

Jet 3.5 is the version of Jet that shipped with Access 97 and Visual Basic 6, while Jet 4.0 is the version shipped with Access 2000. While Jet 4.0 offers some improvements over Jet 3.5, you should probably stick with Jet 3.5 unless you need to share your database with an Access 2000 application, since Access 2000 works only with Jet 4.0.

Oracle 8*i*

Oracle 8*i* is a high performance database system that runs on many different operating systems. While SQL Server is available only for Windows-based systems, Oracle 8*i* runs on everything from small Linux systems to large Sun Solaris-based systems and IBM mainframes running MVS/ESA or OS/390. Of course it also runs on Windows 2000/NT. Since the same code base is used for all of the different platforms, Oracle 8*i* applications need not worry about the computer that is hosting the database server.

Like SQL Server, Oracle 8*i* is available in several editions. The Standard Edition represents the most common form of Oracle 8*i*, and is suitable for most database applications. Oracle 8*i* Enterprise Edition is designed to support high-volume *online transaction processing* (OLTP) applications and query-intensive data warehouses. Oracle 8*i* Personal Edition is targeted at single user development and deployment applications.

Thoughts on Database Systems

Although the architecture of a database system may permit you to write an application program that will work with several different database systems, reality is a lot different than theory.

Most database systems offer assorted extensions that can improve the performance and abilities of most applications. However, if you take advantage of these extensions, your application becomes tied to a single database system. While this isn't necessarily bad, it is something to consider if you think you may change database systems in the future. The key to addressing this problem is to design your database and your application so that they only contain features that are common to all of the database systems you may use.

Isolating the database system from the application is even more important if you are developing applications for resale. Most likely your customers will already have an investment in a database system and would prefer that your application use that database system rather than having to invest in a totally new database system.

Isolating your application from your database system is more difficult than you might believe. This is especially true if you use only one type of database system. You'll often find yourself in situations that are easier to solve if you use a database system-specific feature than if you solve it in a more general-purpose way. This isn't necessarily bad, and often will allow you to build a better application in the long run.

If you plan to use only one database system, selecting the proper one is very important. Although many people pick a database by finding the one with the best set of features, you really should look for a database that will be around in five or ten years. Over time, a feature that is found in only one database management system will be included in the other systems in their next release. After a few releases, those features that you found very important when you picked your database vendor aren't all that important anymore.

Evaluate a database vendor in terms of their commitment to their database system. A vendor that is constantly improving their database is much better than a database vendor that doesn't. A lack of enhancements over a long period of time means that people will tend to select other databases to fit their needs. Eventually, if the enhancements stop, the vendor will stop supporting the database system and you may find yourself in a situation where all of your applications are so dependent on one database system that you can't move them to another database system.

Summary

In this chapter you learned that:

- ✦ Database systems have many advantages over using conventional files to hold your data, including better concurrency controls, better security, and better backup and recovery mechanisms.
- ✦ Database systems are usually implemented using client/server architectures.
- ✦ The different types of databases include the hierarchical, networked, indexed, and relational models.
- ✦ SQL Server 7.0, *Oracle8i* and Jet are three of the most common database systems that a Visual Basic programmer is likely to encounter.

✦ ✦ ✦