Chapter 11

Distributed Databases

Chapter Outline

What You Will Learn in This Chapter

- Why do you need a distributed database?
- What are distributed databases?
- How is data distributed with client/server systems?
- Can a Web approach solve the data distribution issues?
- How much data can you send to a client form?
- What benefits are provided by cloud computing and data storage?
- How will Sally's employees access the database?

A Developer's View

Ariel: How is the new job going, Miranda?

Miranda: Great! The other developers are really fun to work with.

Ariel: So you're not bored with the job yet?

Miranda: No. I don't think that will ever happen—everything keeps changing. Now they want me to set up a Web site for the sales application. They want a site where customers can check on their order status and maybe even enter new orders.

Ariel: That sounds hard. I know a little about HTML, but I don't have any idea of how you access a database over the Web.

Miranda: Well, there are some nice tools out there now. With SQL and a little programming, it should not be too hard.

Ariel: That sounds like a great opportunity. If you learn how to build Web sites that access databases, you can write your ticket to a job anywhere.

Getting Started

Databases and applications need to be used in multiple locations. You need to decide where to physically place the DBMS and databases and which data to transfer to users. Four primary methods are used to distribute data: (1) Linked databases, (2) Replicated databases, (3) Web applications, and (3) Cloud computing. You need to understand the strengths and weaknesses of each to choose the best method for any application.

Introduction

Why do you need a distributed database? Today even small businesses have more than one computer. At a minimum they have several personal computers. More realistically, most organizations take advantage of networks of computers by installing portions of their database and applications on more than one computer. As companies open offices in new locations, they need to share data across a larger distance. Increasingly, companies are finding it useful and necessary to share data with people around the world. Manufacturing companies need to connect to suppliers, distributors, and customers. Service companies need to share data among employees or partners. All of these situations are examples of distributed databases. Many applications can take advantage of the network capabilities of the Internet and the presentation standards of the World Wide Web. Even applications that seem simple need to consider some distributed issues. For instance, a basic transaction-processing system, such as Sally's Pet Store might need to access the data from sales terminals as well as several management computers. If the company expands to multiple stores, you would need to decide how to handle the data for the individual stores, yet still combine the information for use by management.

Building applications that function over networks and managing distributed databases can be complicated tasks. The goal is to provide location transparency to the users. Users should never have to know where data is stored. This feature requires a good DBMS, a solid network, and considerable database, network, and security management skills. However, a well-designed distributed database application also makes it easier for a company to expand its operations. On the other hand, databases that run on more than one computer significantly complicate transaction processing.

Increasingly, people are carrying tablets and smart phones to run Web applications. Although business applications are behind social networks and other commercial Web sites, it is likely that eventually most applications will need at least some Web-based capabilities. Certainly, most new applications will be built to run in browsers. With Web-based applications, database content and most application processing is handled on central servers, while user interaction is handled on the portable clients using HTML and JavaScript. In a sense, the database becomes centralized while the user applications are decentralized.

The Internet offers other alternatives for distributed databases. Cloud computing and data storage can be purchased from third-party providers such as Amazon and Microsoft. These services offer the ability to store databases online with fast data transfers and network and computer support provided by experts. But, it can be expensive, so you need to understand the tradeoffs.

Higher speeds in network connections are simplifying the issues of distributed databases. When tablets and cell phones have 40 mbps or higher reliable data connections, it becomes easier to store data and applications centrally.

Two-Minute Chapter

Databases and applications become more complex when users need to access the data from different locations. Locations can be relatively close or they can be thousands of miles apart. Networks are used to connect users to the database and application servers. Local networks can be very fast but still create some design issues for large applications. Wide area networks that require paying for data transfers across relatively slow public networks cause bigger problems with sharing data. The primary design decision is where to locate the data. Today, the two main options are (1) store replicated copies locally and synchronize the copies, or (2) keep the data centralized and use Web servers and applications to connect to users.

Using centralized Web server applications is tempting in many situations because it simplifies the management and control of data. But, despite improvements in networks some applications probably need to remain on local systems. For instance, a checkout systems for retail stores would probably be too slow to run as Web applications. And checkout speed is a critical factor for many retail stores. But network and browser technologies continue to improve so developers have to continue to examine the tradeoffs.

Distributed databases consist of database files stored in different locations, under the control of different copies of a DBMS. Developers need to decide if data should be shared instantaneously across all sites or if replicas should be used that periodically synchronize the data changes. The choice depends on whether all sites need to see exactly the same data. Distributed databases also make it harder to deal with key generation, concurrent data access, and distributed transactions.

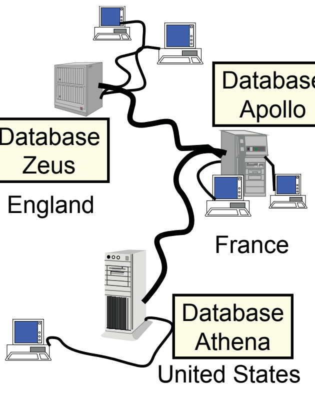


Figure 11.1

Distributed database. Each office has its own hardware and databases. For international projects, workers in different offices can easily share data. The workers do not need to know that the data is stored in different locations.

High-end DBMSs contain mechanisms for handling many of these elements automatically, but they tend to be expensive.

Older client-server systems split applications into the front-end of forms and reports, connected to a back-end database server. Sometimes middleware applications are used to provide business logic and data connectivity between the two layers.

Increasingly, applications are moving to centralized Web servers—perhaps using cloud computing. The data for these sites is centralized but the applications are distributed and available on computers, tablets, and mobile devices through just a Web browser. This approach simplifies the data issues but requires relatively high-speed, and highly-reliable Internet connections to each user.

Distributed Databases

What are distributed databases? ? A **distributed database** system consists of multiple independent databases that operate on two or more computers that are connected and share data over a network. The databases are usually in different physical locations. Each database is controlled by an independent DBMS, which is responsible for maintaining the integrity of its own databases. In extreme situations, the databases might be installed on different hardware, use different operating systems, and could even use DBMS software from different vendors. That last contingency is the hardest one to handle. Most current distributed databases function better if all of the environments are running DBMS software from the same vendor.

In the example shown in Figure 11.1, a company could have offices in three different nations. Each office would have its own computer and database. Much of the data would stay within the individual offices. For example, workers in the United States would rarely need to see the daily schedules of workers in France. On the other hand, workers in France and England could be working on a large international project. The network and distributed database enable them to share data and treat the project as if all the information were in one place.

Distributed databases do not have to be international. They might even exist within the same building. The key part of the definition is that the database runs on different computers—those computers could simply be in different rooms. However, distributed systems where the machines are physically close to each other are much easier to configure. The primary issue in distributed databases is the speed of the network connections. When the machines are physically close, you can easily install high-speed networks at a relatively low cost. When you can transfer huge amounts of data between computers for almost no cost, the distributed issues are easier to solve. On the other hand, you still have to understand the options and plan for potential problems.

The main issue with a distributed database is identifying the data that needs to be shared. Data that is used exclusively within one location, on a single database, is easy to handle—just keep it on the local computer. Data that needs to be accessed from multiple locations is more complex. You have to choose how the data will be shared. For example, you could keep all of the data on one central computer. Or, you could replicate copies to each location. The challenge is to balance the benefits and costs to achieve the performance needed for each application.

Goals and Rules

It is difficult to create a DBMS that can adequately handle distributed databases. (The major issues will be addressed in later sections.) In fact, early systems faced various problems. Consequently, a few writers have created a set of goals or rules that constitute the useful features of a distributed DBMS. C. J. Date, who worked with E. F. Codd to define the relational database approach, lists several rules that he feels are important. This section summarizes Date's rules.

In anyone's definition of a distributed database, the most important rule is that the user should not know or care that the database is distributed. For example, the user should be able to create and run a simple query just as if the database were on one computer. Behind the scenes the DBMS might connect to three different computers, collect data, and format the results. But the user does not know about these steps.

As part of this rule, the data should be stored independently of location. For example, if the business changes, it should be straightforward to move data from one machine and put it in a different office. This move should not crash the entire application, and the applications should run with a few simple changes. The system should not rely on a central computer to coordinate all the others. Instead, each computer should contact the others as needed. This separation improves system performance and enables the other offices to continue operations even if one computer or part of the network goes down.

Some additional goals are more idealistic. The DBMS should be hardware and operating system independent so that when a newer, faster computer is needed, the company could simply transfer the software and data to the new machine and have everything run as it did before. Similarly, it is beneficial if the system runs independently of its network. Most large networks are built from components and software from a variety of companies. A good distributed DBMS should be able to function across different networks. Finally, it is preferable if the distributed application does not rely on using DBMS software from only one vendor. For example, if two companies were to merge, it would be great if they could just install a network connection and have all the applications continue to function—even if the companies have different networks, different hardware, and database software from different vendors. This idealistic scenario does not yet exist.

These features are desirable because they would make it easier for a company to expand or alter its databases and applications without discarding the existing work. By providing for a mix of hardware, software, and network components, these objectives also enable an organization to choose the individual components that best support its needs.

Advantages and Applications

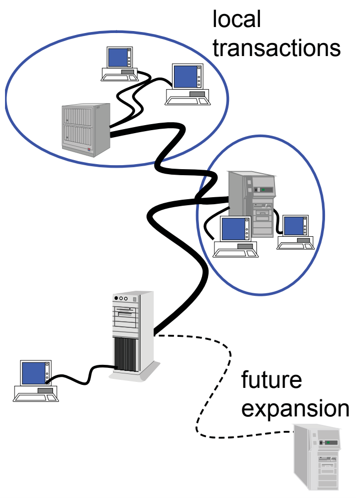
The main strength of the distributed database approach is that it matches the way organizations function. Business operations are often distributed across different locations. For example, work and data are segmented by departments. Workers within each department share most data and communications with other workers within that department. Yet some data needs to be shared with the rest of the company as well. Similarly, larger companies often have offices in different geographical regions. Again, much of the data collected within a region is used within that region; however, some of the data needs to be shared by workers in different regions.

Three basic configurations exist for sharing data: (1) one central computer that collects and processes all data, (2) independent computer systems in each office that do not share data with the others, and (3) a distributed database system.

The first option—a single computer—was the earliest approach to the problem. Interestingly, it is regaining popularity in recent years. Originally, simple terminals were connected to a single, expensive, computer. Today, data can be stored on central servers and accessed via Web browsers from anywhere in the world. Keeping all of the data in one location greatly simplifies coordination and security. Using relatively inexpensive computers as browsers makes it easy to replace them if something breaks.

Figure 11.2

Distributed database strengths. Most data is collected and stored locally. Only data that needs to be shared is transmitted across the network. The system is flexible because it can be expanded in sections as the organization grows.



The second option is a possibility—as long as the offices rarely need to share data. It is still a common approach in many situations. Data that needs to be shared is transmitted via paper reports, e-mail messages, or perhaps text files. Of course, these are ineffective methods for sharing data on a regular basis.

Figure 11.2 illustrates the third option of using a distributed database approach. The main advantage is that distributed systems provide a significant performance advantage through better alignment with the needs of the organization. Most updates and queries are performed locally. Each office retains local control and responsibility for the data in that office. Yet the system enables anyone with the proper authority to retrieve and integrate data from any portion of the company as it is needed.

A second advantage to distributed databases is that, compared to centralized systems, they are easier to expand. Think about what happens if the company is using one large, centralized computer. If the company expands into a new region, requiring more processing capacity, the entire computer might have to be replaced. With a distributed database approach, expanding into a new area would be supported by adding another computer with a database to support the new operations. All existing hardware and applications remain the same. By using smaller computer systems, it is easier and cheaper to match the changing needs of the organization.

Because the distributed database approach can be tailored to match the layout of any company, it has many applications. In a transaction processing system, each region would be responsible for collecting the detailed transaction data that it uses on a daily basis. For instance, a manufacturing plant would have a database to collect and store data on purchases, human relations, and production. Most of this data would be used by the individual plant to manage its operations. Yet as part of the corporate network, summary data could be collected from each plant and sent to headquarters for analysis. As another example, consider a consulting firm with offices in several countries. The workers can store their notes and comments in a local database. If a client in one country needs specialized assistance or encounters a unique problem, the local partners can use the database to search for similar problems and solutions at other offices around the world. The distributed database enables workers within the company to share their knowledge and experiences.

Creating a Distributed Database System

The basic steps to building a distributed database are similar to those for creating any database application. Once you identify the user needs, the developers organize the data through normalization, create queries using SQL, define the user interface, and build the application. However, as shown in Figure 11.3, a distributed database requires some additional steps. In particular, a network must connect the computers from all the locations. Even if the network already exists, it might have to be modified or extended to support the chosen hardware and DBMS software.

Another crucial step is to determine where to store the data. The next section examines some of the issues you will encounter with processing queries on a distributed database. For now, remember that the goal is to store the data as close as possible to the location where it will be used the most. It is also possible to replicate heavily used data so that it can be stored on more than one computer. Of course, then you need to choose and implement a strategy to make sure that each copy is kept up-to-date.

Design administration plan.
Choose hardware, DBMS vendor, and network.
Set up network and DBMS connections.
Choose locations for data.
Choose replication strategy.
Create backup plan and strategy.
Create local views and synonyms.
Perform stress test: loads and failures.

Figure 11.3

Additional steps to creating a distributed database. After the individual systems and network are installed, you must choose where to store the data. Data can also be replicated and stored in more than one location. Local views and synonyms are used to provide transparency and security. Be sure to stress test the applications under heavy loads and to ensure that they handle failures in the network and in remote computers.

Backup and recovery plans are even more critical with a distributed database. Remember that several computers will be operating in different locations. Each system will probably have a different DBA. Yet the entire database must be protected from failures, so every system must have consistent backup and security plans. Developing these plans will probably require negotiation among the administrators—particularly when the systems cross national boundaries and multiple time zones. For example, it would be virtually impossible to back up data everywhere at the same time.

Once the individual systems are installed and operational, each location must create local views, synonyms, and stored procedures that will connect the databases, grant access to the appropriate users, and connect the applications running on each system. Each individual link must be tested, and the final applications must be tested both for connections and for stress under heavy loads. It should also be tested for proper behavior when a network link is broken or a remote computer fails.

Operating and managing a distributed database system is considerably more difficult than a handling single database. Identifying the cause of problems is much more difficult. Basic tasks like backup and recovery require coordination of all DBAs. Some tools exist to make these jobs easier, but they can be improved. Do you remember the rule that a distributed database should be transparent to the user? That same rule does not yet apply to DBAs or to application developers. Coordination among administrators and developers is crucial to making applications more accessible to users.

Network Speeds

The challenge with distributed databases comes down to physics and economics. As illustrated in Figure 11.4, data that is stored on a local disk drive can be transferred to the CPU at transfer rates of 60 to 400 megabytes per second (higher speeds with SSD and RAID drives). Data that is stored on a server attached to a **local area network (LAN)** can be transferred at rates from 10 to 100 megabytes per second (100 to 1,000 megabits per second). Using public transmission lines to connect across a **wide area network (WAN)** provides transfer rates from 0.2 to 300 megabytes per second. To get that 300 megabytes per second (on an OC-

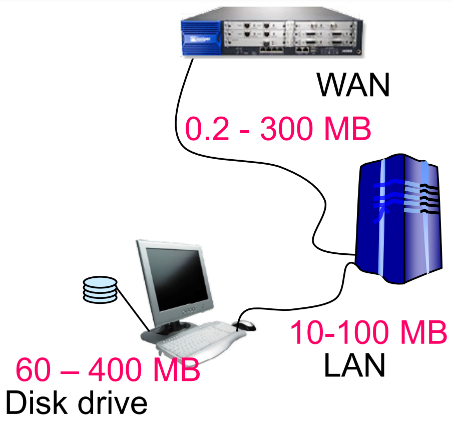


Figure 11.4

Network transfer rates. Network performance is shown here in megabytes per second. Local disks and networks are considerably faster than wide area networks and WAN costs are higher.

48 line at 2488 mbps), your company would probably have to pay over $50,000 a month in network costs. As technology changes these numbers are continually improving. One of the biggest changes in recent years is the performance gains of local area networks. With gigabit performance, it is relatively easy to move data away from the processor and place it on a network attached storage device on a storage area network (SAN). Separating the data from the processor makes it easier to upgrade processors and provide backup facilities—both in terms of backing up the data and replacing servers. Although a SAN offers several benefits to running servers, it does not solve the distributed database problem because the distance is still limited.

Note that most DBMS vendors also sell enterprise versions of the software that can take advantage of a computer cluster. A **cluster** consists of multiple computers that effectively work as a single machine. Because the system uses a single copy of the DBMS, it is not a distributed system. However, it does take advantage of fast LAN speeds and attached storage. The major strengths of clusters are **fault tolerance** and **scalability**. The system automatically balances the load across the servers. If one of the processors fails, the system simply ignores it. The DBA can shut down the failed server and replace it with a new one. Likewise, as the database grows, and you need more processing power, you simply add another machine to the cluster. The system detects the new capabilities and redirects processing needs to the new server.

The real issues of distributed databases arise when you need to connect machines using a wide area network (WAN). Although high-speed WANs are becoming more common, they are still relatively expensive. The goal of distributed processing is to minimize the transfer of data on slower networks and to reduce the costs of network transfers. Part of this goal can be accomplished through design—developers must carefully choose where data should be located. Data should be stored as close as possible to where it will be used the most. However, trade-offs always arise when data is used in several locations.

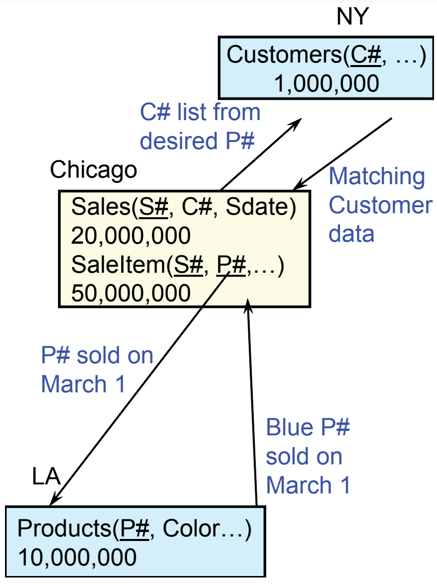


Figure 11.5

Distributed database query example. List customers who bought blue products on March 1. A bad idea is to transfer all data to Chicago. The goal is to restrict each set and transfer the least amount of data.

Query Processing and Data Transfer

Data transfer rates are a key issue in distributed processing. To understand their importance, consider the issue of responding to queries if data is stored in separate locations. If a query needs to retrieve data from several different computers, the time to transfer the data and process the query depends heavily on how much data must be transferred and the speed of the transmission lines. Consequently, the result depends on how the DBMS joins the data from the multiple tables. In some cases the difference can be extreme. One method could produce a result in a few seconds. A different approach to the same query might take several days to process! Ideally, the DBMS should evaluate the query, the databases used, and the transmission times to determine the most efficient way to answer the query.

Figure 11.5 illustrates the basic problem. Consider tables on three different databases: (1) a Customer table in New York with 1 million rows, (2) a Production table in Los Angeles with 10 million rows, and (3) a Sales table in Chicago with 20 million rows. A manager in Chicago wants to run the following query: List customers who bought blue products on March 1. This query could be processed in several ways. First, consider a bad idea. Transfer all of the rows to Chicago; then join the tables and select the rows that match the query. This method results in 11 million rows of data being transferred to Chicago. Even with a relatively fast WAN, anything less than 30 minutes for this query would be fast.

A better idea would be to tell the database in Los Angeles to find all of the blue products and send the resulting rows to Chicago. Assuming only some of the products are blue, this method could significantly cut the number of rows that need to be transmitted. The performance gain will depend on what percentage of rows consists of blue products.

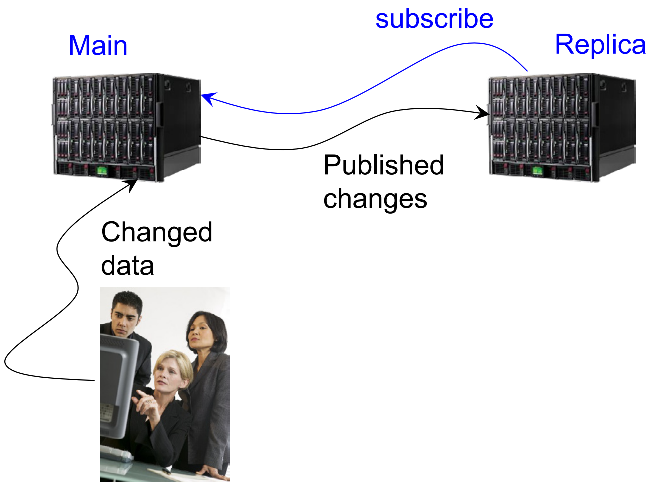


Figure 11.6

Replication with subscribe/publish. The databases are linked by subscribing a replica to the main database. When changes are made on the main, they are published to all of the subscribers.

An even better idea is to get the list of items sold on March 1 from the Chicago table, which requires no transmission cost. Send this list to Los Angeles and have that database determine which of the products are blue. Send the matching CustomerID to the New York database, which returns the corresponding Customer data.

Notice that to optimize the query the DBMS needs to know a little about the data in each table. For example, if there are many blue products in the Los Angeles database and not very many sales on March 1, then the database should send the Sales data from Chicago to Los Angeles. On the other hand, if there are few blue products, it will be more efficient to send the product data from Los Angeles to Chicago. In some cases, the network also needs to know the transfer speed of the network links. A good DBMS contains a query optimizer that checks the database contents and network transfer speeds to choose the best method to answer the query. You still might have to optimize some queries yourself. The basic rule is to transfer the least amount of data possible.

Data Replication

Sometimes there is no good way to optimize a query; or there might be many queries and each requires conflicting optimization methods. When large data sets are needed in several different places, it can be more efficient to **replicate** the tables and store copies in each location. The problem is that the databases involved have to know about each of the copies. If a user updates data in one location, the changes have to be replicated to all the other copies. Two common methods are used to hanlde synchronization: (a) replication mangement, and (2) subscribe/publish connections.

Developers and database administrators can tune the performance by specifying how the database should be replicated. You can control how often the changes

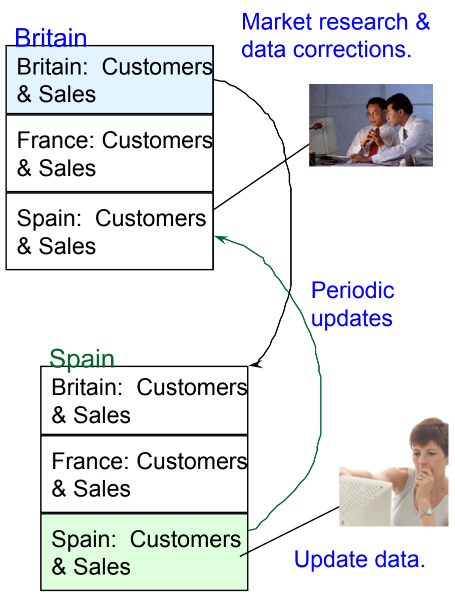

Britain
Britain: Customers & Sales
France: Customers & Sales
Spain: Customers & Sales
Market research & data corrections.
Periodic updates
Spain
Britain: Customers & Sales
France: Customers & Sales
Spain: Customers & Sales
Update data.

Figure 11.7

Replicated databases. If managers do not need immediate data from other nations, the tables can be replicated and updates can be transferred at night when costs are lower.

are distributed and whether they are sent in pieces or as a bulk transfer of the entire table. The biggest difficulty is that sometimes a network link might be unavailable or a server might be down. Then the DBMS has to coordinate the databases to make sure they get the current version of the table and do not lose any changes. Figure 11.6 shows the basic concept of publish/subscribe. Once the subscription connection is established, any changes made to the main database are published to all of the subscribers. In many cases, the changes are sent immediately, which results in a continuous flow of changed data across the network.

With bulk synchronization, most of the database is transferred to a second location, the changes are exchanged, and the new database is returned to the original location. With subscribe/publish, databases that want to be informed of changes create a subscription to a main database. When changes are made on the main copy, they are published and sent to the subscribing databases. In both cases, a replication manager in the DBMS determines which changes should be sent and to handle the updates at each location. Replications can be sent automatically at certain times of the day, sent continuously, or triggered manually when someone feels it is necessary to synchronize the data.

Figure 11.7 illustrates the basic concepts of replication. Marketing offices in each location have copies of Customer and Sales data from Britain, France, and Spain. Managers probably do not need up-to-the-minute data from the other countries, so the tables can be replicated as batch updates during the night. The data will be available to managers in all locations without the managers worrying about transfer time, and the company can minimize international transmission costs by performing transfers at off-peak times.

Transaction processing databases generally record many changes—sometimes hundreds of changes per minute. These applications require fast response times at the point of the transaction. It is generally best to run these systems as distributed databases to improve the performance within the local region. On the other hand, managers from different locations often need to analyze the transaction data. If you give them direct access to the distributed transaction databases, the analysis queries might slow the performance of the transaction system. A popular solution is to replicate the transaction data into a data warehouse. Routines extract data from the transaction processing system and store it in the data warehouse. Managers run applications and build queries to retrieve the data from the warehouse and analyze it to make tactical and strategic decisions. Because the managers rarely make changes to the underlying data, the data warehouse is a good candidate for replication. The underlying transaction processing system retains its speed, and the raw data is not shared. Managers have shared access to the warehouse data.

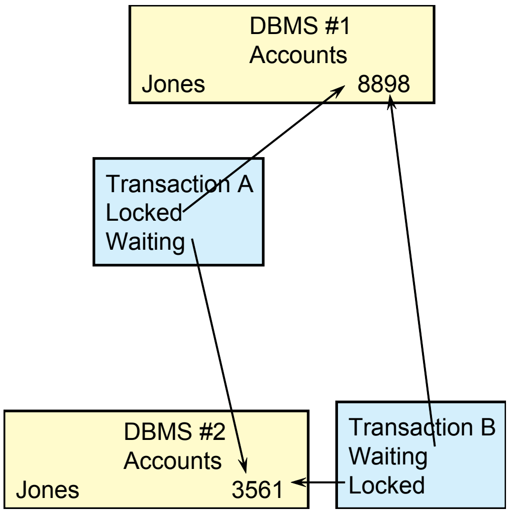
Generating Keys with Replicated Data

Replication seems like an easy solution—each location has a complete copy of the database; performance of local updates and queries is unaffected by the other copies; transactions are completed locally; and data backups are made automatically. In practice, several problems can arise. One problem is the need to generate unique primary key values. A second is the issue of concurrent changes—which is described in the next section.

Automatic key generation is a challenge with replicated databases. What happens if two people in different locations create a new customer? If the key generator is not synchronized, then it is highly likely that both locations will generate the same key, and when the data is updated from the two locations, a collision will occur that must be resolved by hand. Two common methods can be used in dis-

Figure 11.8

Concurrency and deadlock are more complicated in a distributed database. The deadlock can arise across many different databases, making it hard to identify and resolve.



tributed databases to generate keys safely: (1) randomly generated keys, and (2) location-specific keys. Randomly generated keys work if the generator chooses from a sufficiently large number of possible keys. Then there is only a small probability that two keys would ever be generated the same at the same time. To be safe, the generator immediately checks to see if the key just created already exists. The second approach can use either sequential keys or random keys, but it relies on each location being allocated a certain range of values. For instance, one region might be given the range from 1 to 1 million, the next from 1 million to 2 million, and so on. With location-specific generators, you must be careful to isolate the key generation data tables. For example, your key tables would contain the location identifier and the starting or current value of the key. This problem is relatively easy to solve—but you must remember to configure it in your application.

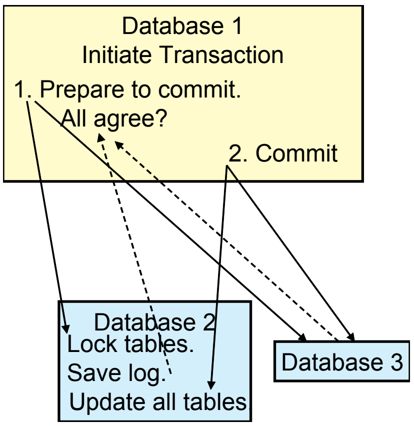
The **globally-unique identifier (GUID)** is often used in distributed databases when a unique value needs to be created. A GUID is essentially a large random number. Microsoft tools use them extensively, and the mechanism for creating them is accessible to most programming tools. The Microsoft algorithm uses the unique ID from the computers' network interface card as part of the GUID, and then adds random digits—for a total of 128 digits. This process ensures that different machines always create different numbers.

Concurrency, Locks, and Transactions

Concurrency and deadlock become complex problems in a distributed database. Remember that concurrency problems arise when two people try to alter the same data at the same time. The situation is prevented by locking a row that is about to be changed. As shown in Figure 11.8, the problem with a distributed database is that the application could create a deadlock that involves different databases on separate computers. One user could hold a lock on a table on one computer and be waiting for a resource on a different computer. Now imagine what happens when there are five databases in five locations. It can be difficult to identify the deadlock

Figure 11.9

Two-phase commit. Each database must agree to save all changes—even if the system crashes. When all systems are prepared, they are asked to commit the changes.

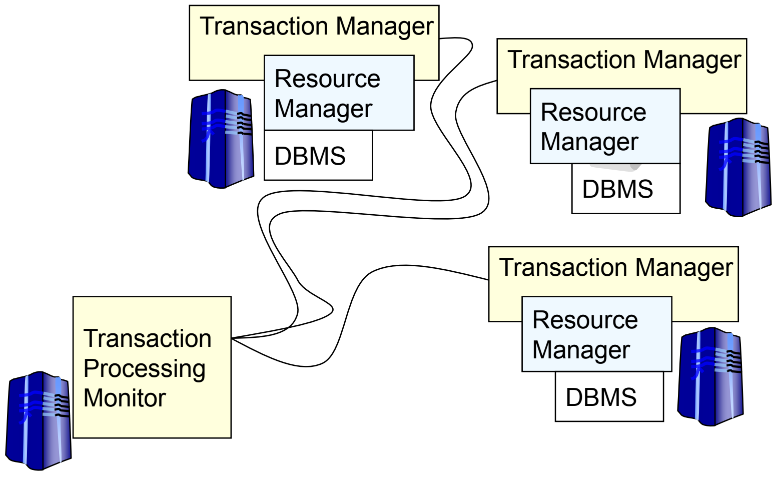


problem. When the locks are on one computer, the DBMS can use a lock graph to catch deadlock problems as they arise. With distributed databases, the DBMS has to monitor the delay while waiting for a resource. If the delay is too long, the system assumes a deadlock has arisen and rolls back the transaction. Of course, the delay might simply be due to a slow network link, so the method is not foolproof. Worse, the time spent waiting is wasted. In a busy system, the DBMS could spend more time waiting than it does processing transactions.

Handling transactions across several databases is also a more complex problem. When changes have to be written to several computers, you still have to be certain that all changes succeed or fail together. To date, the most common mechanism for verifying transactions utilizes a **two-phase commit** process. Figure 11.9 illustrates the process. The database that initiates the transaction becomes a coordinator. In the first phase it sends the updates to the other databases and asks them to prepare the transaction. Each database must then send a reply about its status. Each database must agree to perform the entire transaction or to roll back changes if needed. The local database must agree to make the changes even if a failure occurs. In other words, it writes the changes to a transaction log. Once the log is successfully created, the remote database agrees that it can handle the update. If a database encounters a problem and cannot perform the transaction (perhaps it cannot lock a table), it sends a failure message and the coordinator tells all the databases to roll back their changes. A good DBMS handles the two-phase commit automatically. As a developer, you write standard SQL statements, and the DBMS handles the communication to ensure the transaction is completed successfully. With weaker systems you will have to embed the two-phase commit commands within your program code. If you know that you are building an application that will use many distributed updates, it is generally better to budget for a better DBMS that can handle the two-phase-commit process automatically.

Figure 11.10

Distributed transaction processing monitor. This software handles the transaction decisions and coordinates across the participating systems by communicating with the local transaction managers.



Notice that the two-phase commit system relies on pessimistic locking. Because of transmission delays, it could significantly slow down all of the systems involved in the transaction—as it waits for each machine to lock records. Although optimistic locking might help with some aspects of the transaction, it does not help when a system or communication link fails.

Distributed Transaction Managers

The problem of distributed systems—particularly when database systems are from diverse vendors—is difficult to solve efficiently. One common approach, shown in Figure 11.10, is to use an independent transaction processing monitor or distributed transaction coordinator. This system is a separate piece of software that coordinates all transactions and makes the decision to commit or abort based on interactions with the local transaction managers. This approach is generally provided by the operating system vendor, and the DBMS vendors need to develop interfaces that communicate with the transaction manager. The main transaction manager could run on a separate system, or one of the local transaction managers might be promoted to be the coordinator. For example, Microsoft provides the Distributed Transaction Coordinator, IBM supports Java transactions within its WebSphere Application server, and JBoss Transactions is available independently for UNIX platforms.

Independence is the main strength of the transaction manager. As long as multiple vendors provide support (with the local resource manager software), the system can support diverse products. It is also useful for program-level transactions, where a substantial amount of code is written outside of the databases (e.g., in C++). By relying on the transaction manager, the database system could be changed later if desired—without having to rewrite all of the transaction-processing elements.

Figure 11.11

Design questions. Use these questions to determine whether you should replicate the database, or provide concurrent access to data across the network. Transaction operations are generally run with concurrent access. Decision support systems often use replicated databases. However, the exact choice depends on the use of the data and the needs of the users.

Question	Concurrent	Replication
What level of data consistency is needed?	High	Low – Medium
How expensive is storage?	Medium – High	Low
What are the shared access requirements?	Global	Local
How often are the tables updated?	Often	Seldom
Required speed of updates (transactions)?	Fast	Slow
How important are predictable transaction times?	High	Low
DBMS support for concurrency and locking?	Good – Excellent	Poor
Can shared access be avoided?	No	Yes

Distributed Design Questions

Because of the issues with transmission costs, replication, and concurrency, distributed databases require careful design. As networks gain better transfer rates, database design will eventually become less of a problem. In the meantime you need to analyze your applications to determine how they should be distributed. Figure 11.11 lists some of the questions you need to ask when designing a distributed database. The main point is to determine what portions of the databases should be replicated. If users at all locations require absolute consistency in the database, then replication is probably a bad idea. On the other hand, you might have a weak DBMS that poorly handles locking and concurrency. In this situation it is better to replicate the data, rather than risk destroying the data through incomplete transaction updates.

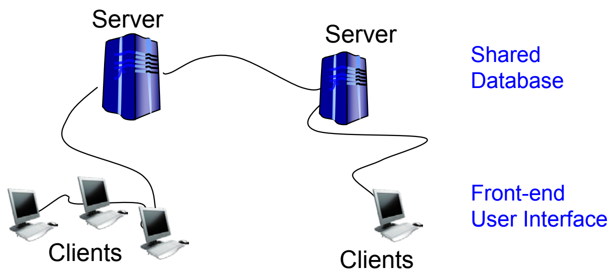
Client/Server Databases

How is data distributed with client/server systems? Many applications run on local area networks using some version of a client/server configuration. This approach is particularly common within retail stores, where checkout registers are based on simple client computers. With this system, the bulk of the data is stored on a centralized server, while the applications run on personal computers. However, some of the data might also be stored on the personal computers, and portions of the application logic might run on middle-tier servers. The client/server approach was driven largely by the limited capabilities of personal computer operating systems. Early operating systems could not support multiple users and provided no security controls. Hence powerful operating systems were installed on servers that handled all the tasks that required sharing data and hardware. The client/server approach is also somewhat easier to manage and control than monitoring hundreds of PCs. Any hardware, software, or data that needs to be shared is stored in a centralized location and controlled by an MIS staff. With the client/server approach, all data that will be shared is first transferred to a server.

As indicated by Figure 11.12, the actual database resides on a server computer. Individual components can be run from client machines, but they store and retrieve data on the servers. The client component is usually a front-end application

Figure 11.12

Client/server system. The client computers run front-end, user interface applications. These applications retrieve and store data in shared databases that are run on the server computers. The network enables clients to access data on any server where they have appropriate permissions.



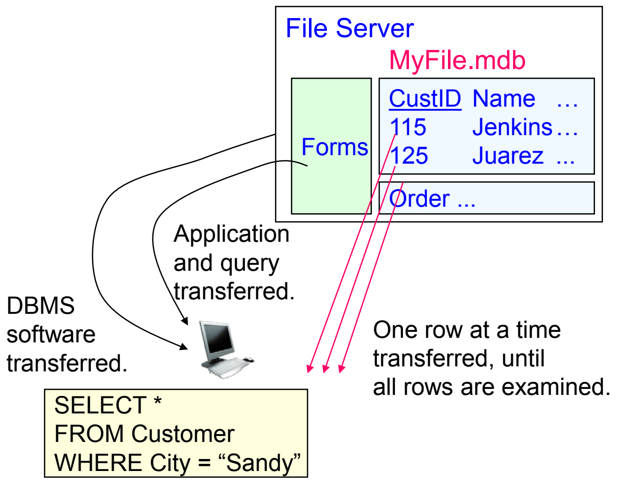


Figure 11.13

File server problems. The file server acts as a large, passive disk drive. The personal computer does all the database processing, so it must retrieve and examine every row of data. For large tables, this process is slow and wastes network bandwidth.

that interacts with the user. For example, a common approach is to store the data tables on a server but run the forms on personal computers. The forms handle user events with a graphical interface, but all data is transferred to the server. You need to understand a few important concepts to design and manage client/server databases. Like any distributed database, where you store the data and how you access it can make a substantial difference in performance. This section also demonstrates some of the tools available to build a client/server database application.

Client/Server versus File Server

To understand the features and power of a client/server database, it is first useful to examine a database application that is not a true client/server database. Initial local area networks were based on file servers. A file server is a centralized computer that can share files with personal computers. However, it does not contain a DBMS. The file server stores files, but to the personal computers it appears as a giant, passive disk drive. The sole purpose of the server is to provide secure shared access to files. The client personal computers do all of the application processing.

Microsoft Access is often used in file server applications. It is relatively easy to split the database and store the data as a file on the server that is accessed by the forms and reports running on personal computers. The file server approach enables you to share a single copy of the data so all users see the same data. However, the file server approach faces some important drawbacks.

Figure 11.13 illustrates the basic problem. The database file is stored on the file server but the DBMS itself runs on the client. Security permissions are set so that each user has read and write permission on the file. The problem arises when your application runs a query. The processing of the query is done on the client computer. That means that the personal computer has to retrieve every row of data from the server, examine it, and decide whether to use it in the computation or display. If the database is small, if the network connection is fast, and if users

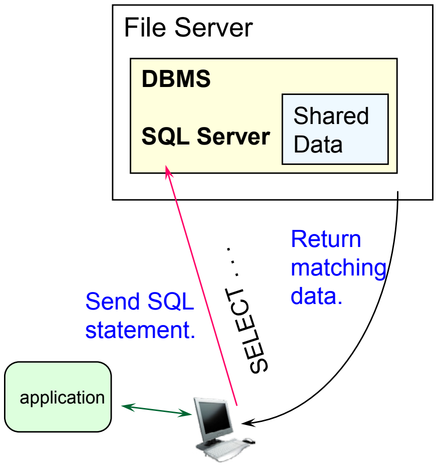


Figure 11.14

Database server. The client computer sends a SQL statement that is processed on the server. Only the result is returned to the client, reducing network traffic.

often want to see the entire table, then this process does not matter. But if the table is large and users need to see only a small portion, then it is a waste of time and network bandwidth to transfer the entire table to the client computer.

The problem is that the file server approach relies on transferring huge amounts of data when the application needs only some of the data. The client/server database approach was designed to solve this problem. With a client/server database, the binary code for the database actually runs on the server. As shown in Figure 11.14, the server database receives SQL statements, processes them, and returns only the results of the query. Notice the reduction in network transfers. The initial SQL statement is small, and only the data needed by the application is transferred over the network. This result is particularly important for decision support systems. The server database might contain millions of rows of data. The manager is analyzing the data and may want summary statistics, such as an average. The server database optimizes the query, computes the result, and transfers a few simple numbers back to the client. Without the server database millions of rows of data would be transferred across the network. Remember that even fast LAN transfer rates are substantially slower than disk drive transfers.

Of course, the drawback to the server database approach is that the server spends more time processing data. Consequently, the server computer has to be configured so that it can efficiently run processes for many users at the same time. Fortunately, processor speeds have historically increased much more rapidly than disk drive and network transfer speeds. The other drawback is that this approach requires the purchase of a powerful DBMS that runs on the server. However, you rarely have a choice. Only small applications used by a few users can be run without a database server.

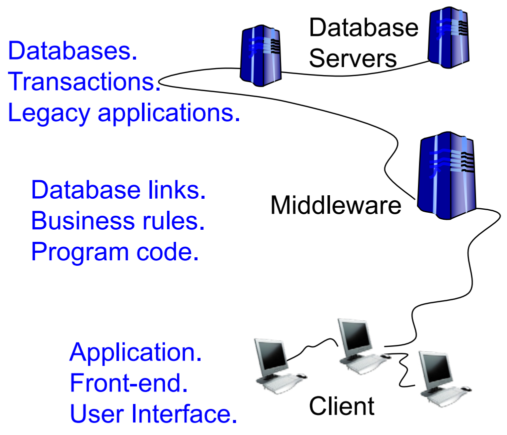


Figure 11.15

Three-tier client/server model. The middle layer separates the business rules and program code from the databases and applications. Independence makes it easier to alter each component without interfering with the other elements.

Three-Tier Client/Server Model

The **three-tier client/server** model has been suggested as an approach that has some advantages over the two-tier model. The three-tier approach adds a layer between the clients and the servers. The three-tier approach is particularly useful for systems having several database servers with many different applications. The method is useful when some of the servers are running legacy applications.

As shown in Figure 11.15, one role of the middle layer is to create links to the databases. If necessary, the middle layer translates SQL requests and retrieves data from legacy COBOL applications. By placing the access links in one location, the server databases can be moved or altered without affecting the client front-end applications. Developers simply change the location pointers, or alter the middleware routines. Some people refer to this approach as n-tier because you can have any number of middle-level computers—each specializing in a particular aspect of the business rules.

Another important role of the middle layer is to host the business rules. For example, creating identification numbers for customers and products should follow a standard process. The routine that generates these numbers should be stored in one location, and all the applications that need it will call that function. Similarly, common application functions can be written once and stored on the middle-layer servers.

This middleware system is well suited to an object-oriented development approach. Common objects that are used for multiple business applications can be written once and stored on the middle servers. Any application can use those objects as needed. As the business rules change or as systems are updated, developers can alter or improve the base objects without interfering with the operations of the applications on the client side. The three-tier approach separates the business rules and program code from the databases and from the applications. The independence makes the system more flexible and easier to expand. Several middleware development tools exist to create and manage objects, but many are proprietary to specific platforms, such as those by Oracle and IBM.

The Back End: Server Databases

Server database systems tend to be considerably more complex and require more administrative tasks than personal computer-based systems. The server environment also provides more options, which makes administration and development more complicated. The DBA must work closely with the system administrator to set up the software, define user accounts, and monitor performance.

Server databases also use trigger procedures to define and enforce business rules. One of the more difficult design questions you must address is whether to store these rules on the back-end database as database triggers and procedures, or move them to a middle-level server using lower-level languages such as C++ or Java. Sometimes you are constrained by the tools and time available. But when possible, you should consider the various alternatives in terms of cost, performance, and expandability.

Placing procedures in the back-end database ensures that all rules are enforced by the DBMS, regardless of how the data is accessed. But, this approach ties you in to a particular DBMS vendor. Because most systems contain proprietary elements, it is difficult to switch to a different DBMS in the future. Placing rules in a middle tier also makes it easier to physically move the database. Generally, the systems are built with reference links to the databases. To move the database, you simply change the reference pointers.

One rule of thumb is to write user-interface code for the client computers and to write data manipulation and control programs to run on the server. Middle-layer programs are used to encode business rules and provide data translation and database independence. The primary objective is to minimize the transfer of data across the network. However, if some computers are substantially slower than others, you will have to accept more data transfers in order to execute the code on faster machines.

The Front End: Windows Clients

Windows-based computers are commonly used as client machines, so Microsoft has created several technologies to provide database connections from the PCs to back-end databases. Various tools and many vendors support the technologies, so they are relatively standardized. The tools have evolved over time as hardware and networks have improved and applications became more complex. Visual Studio is often used as a front-end tool to create the forms and reports. The application is compiled and distributed to user machines, which connect through the Microsoft data components to a back-end server. The PC has a network connection and a database connection that enables it to find the central database on the server. The application code simply selects the appropriate database connection. From that point, your application no longer cares where the data is located—it simply passes SQL requests to the server.

The current Microsoft technology to connect programs to databases is **active data objects (ADO)**. Most DBMS vendors have written ADO connectors, so your application code can retrieve and save data to the most common DBMSs. Whenever you build an application in Visual Studio, you will use ADO to connect to the DBMS. All of the commonly-used commands are embedded in the objects, so you can retrieve data to display it in a form and save changes back to the database with a couple of calls to the object methods. ADO is also used in Microsoft's Web-based applications and the underlying concepts are the same. Java (now supported by Oracle) uses JDBC to connect the language to backend databases.

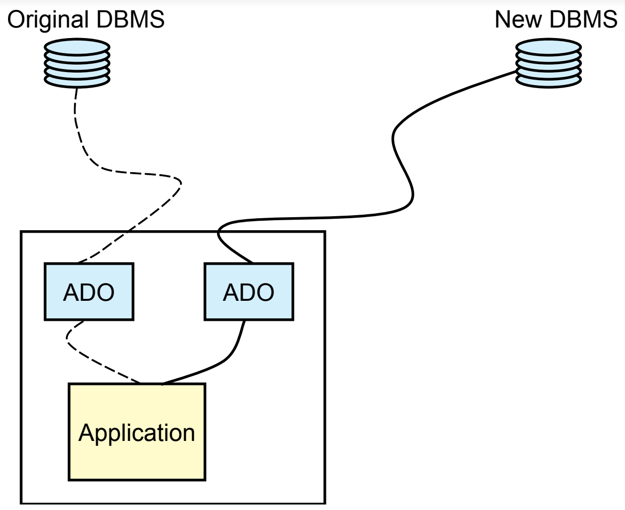


Figure 11.16

Database independence. ADO is a useful buffer between the application and the DBMS. Changing the connection makes it relatively easy to switch the back-end DBMS.

Maintaining Database Independence in the Client

One of the trickiest aspects of distributed databases is the issue of maintaining database independence. When you first build an application, it is often created to run with a single, specified database on the back end. Consequently, it is tempting to simply build the application assuming that the same database will always be there, and use the tools and shortcuts available for that particular system. But what happens later when someone wants to change the back-end database? In extreme cases, the entire application will have to be rewritten. As a developer pressed for time, you might ask why it matters. If someone wants to change the database at a later date, then should they be willing to pay the costs at that time? Yet, with only a little extra effort up front, the application can support most common database systems on the back end; making it easy to change later.

The database connection is one issue in building a generic application. Using a standard such as ADO makes it easier to change databases. Figure 11.16 shows that by changing the connection, your application can connect to a different DBMS. Of course, it is never quite that simple, but the ADO buffer is an important element. In many cases, you can specify the ADO connection string dynamically, making it easy for the application to connect to a different DBMS without rewriting the code. If you are careful, you can build the application so that it can switch DBMS connections at any time. You can build the system using one DBMS and run the production system against a different one.

It is important that you understand that the connection is only one element in making an application DBMS independent. In most situations, the actual SQL commands are a much bigger issue. DBMS vendors tend to provide different levels of support for the SQL standard. They also add proprietary options and commands that are enticing. In particular, vendors offer many variations within the SELECT command. For example, string and date operations are notoriously non-

```
Generic application query:
SELECT SaleID, SaleDate, CustomerID, CustomerName
FROM SaleCustomer

Saved Oracle query:
SELECT SaleID, SaleDate, CustomerID,
        LastName || ', ' || FirstName AS CustomerName
FROM Sale, Customer
WHERE Sale.CustomerID=Customer.CustomerID

Saved SQL Server query:
SELECT SaleID, SaleDate, CustomerID,
        LastName + ', ' + FirstName AS CustomerName
FROM Sale INNER JOIN Customer
ON Sale.CustomerID = Customer.CustomerID
```

Figure 11.17

Database query independence. The application contains only simple queries that do not use vendor-specific functions. All detail queries are created and saved within the DBMS.

standardized across vendors. And, if you are using an older version of Oracle or SQL Server, you will not be able to use the INNER JOIN syntax. Because of these differences, a key step in making an application DBMS independent is to move all queries to the DBMS and save them as views. Then your application only contains simple SELECT (or INSERT/UPDATE/DELETE) queries that pull data from the saved view. These simple queries should use only basic standard SQL elements. All of the vendor-specific functions are coded into the query that is saved in the DBMS. To transfer to a new DBMS, you just recreate the queries on the new DBMS using that vendor's specific tools and syntax. This technique is particularly important for applications that might begin small and grow. At a small size, you might be able to use a small, inexpensive DBMS, but as the number of users grows and demand on the system increases, you will have to scale up to a larger DBMS. If the application queries were carefully built to remain independent, it will be relatively easy to transfer to a new DBMS. Figure 11.17 shows an example of using simple queries to maintain DBMS vendor independence.

Trigger functions are a more complex issue. Some systems do not support triggers at all, and those that do generally provide different functionality. At this point in time, the only method to guarantee compatibility across vendors is to avoid database triggers. Instead, write the same functionality into middleware code that also relies on generic queries.

Centralizing with a Web Server

Can a Web approach solve the data distribution issues? The **World Wide Web** was designed to enable people (initially physicists and researchers) to share information with their colleagues. The fundamental problem was that everyone used different hardware and software. The solution was to define a set of standards. These evolving standards are the heart of the Web. They define how computers can connect, how data can be transferred, and how data can be found. Additional standards define how data should be stored and how it can be displayed. As long

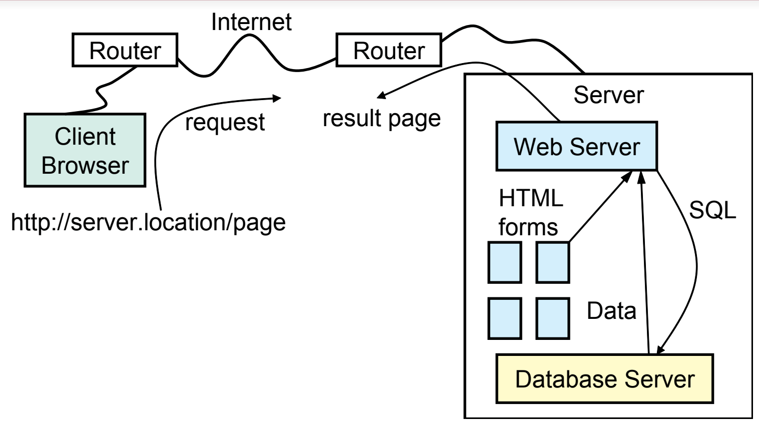


Figure 11.18

Web servers and client browsers. Browsers are standardized display platforms. Servers are accessible from any browser.

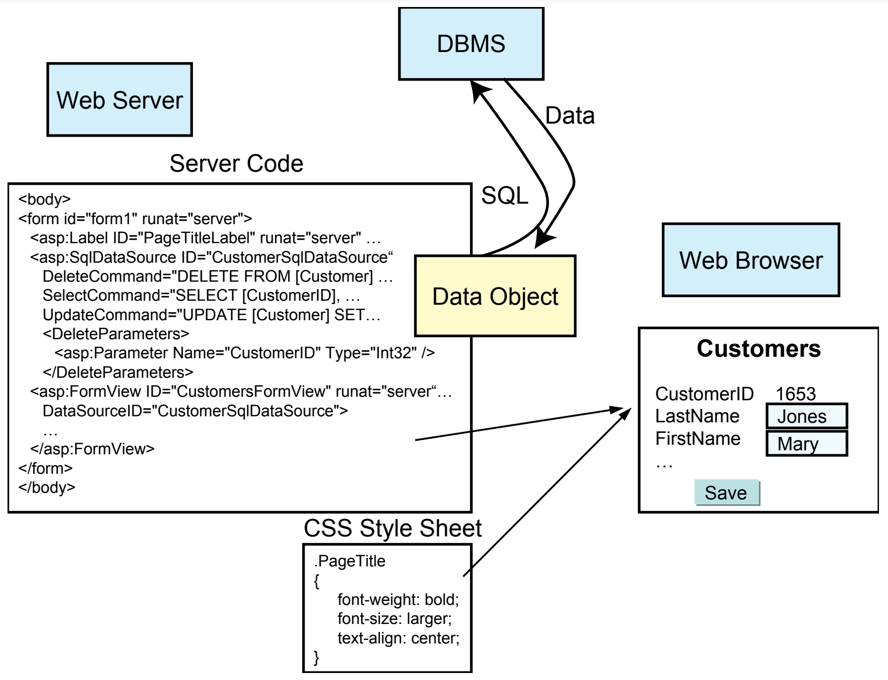
as a computer runs **browser** software that receives and displays data files, it can access and interact with data stored on Web servers. The servers run Web server software that can do almost anything—as long as it formats the data for standard browsers. Both the servers and browsers are becoming more sophisticated, but the essence of the method is presented in Figure 11.18.

An interesting consequence of the rapid acceptance of the Web approach is that it encourages a return to a centralized database. All of the data and applications are stored in one location. Users can be located anywhere in the world—as long as they have a Web browser and an Internet connection. The issues of concurrency and security are simplified again, since everything is handled by one DBMS. The issues of distributed data are minimized, since the data is now stored in one place, and all users share the same data. Placing the data in one location does not remove the issue of data transfer speeds. Users with slow Internet connections might complain about sluggish performance. However, most of the bulk data transfers should take place at the server itself on high-speed lines. The majority of the communication with users can be reduced to simple pages consisting of input screens or simple data results.

The Web-based approach does not yet solve all distributed problems. For instance, a retail chain that has stores in multiple locations will probably want to keep most data locally. Each store would use a server to handle local transactions. This data can be transferred in bulk to the headquarters server a couple of times a day, where it can be made available for analysis over the Web. Even though the data eventually winds up on a central Web server, local server are still needed to improve transaction performance and reliability at each store.

Web Server Database Fundamentals

There is no standard mechanism for connecting databases to the Web server. Consequently, the method you use depends on the specific software (Web server and DBMS) that you install. Most of the methods follow a similar structure but vary in the details. Several tools exist to help you build forms in a graphical designer, supported by a programming language to process the data. These tools then gen-



```
<body>
<form id="form1" runat="server">
  <asp:Label ID="PageTitleLabel" runat="server" …
  <asp:SqlDataSource ID="CustomerSqlDataSource"
    DeleteCommand="DELETE FROM [Customer] …
    SelectCommand="SELECT [CustomerID], …
    UpdateCommand="UPDATE [Customer] SET…
    <DeleteParameters>
      <asp:Parameter Name="CustomerID" Type="Int32" />
    </DeleteParameters>
  <asp:FormView ID="CustomersFormView" runat="server"…
    DataSourceID="CustomerSqlDataSource">
    …
  </asp:FormView>
</form>
</body>
```

```
.PageTitle
{
     font-weight: bold;
     font-size: larger;
     text-align: center;
}
```

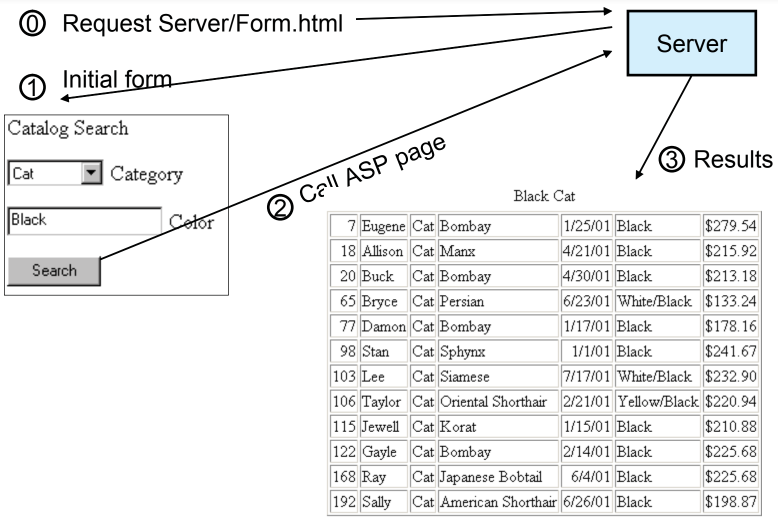
Figure 11.19

Web server database fundamentals. The server executes a script or code page that utilizes a data object to pass SQL commands to the DBMS and receive data and results. The page generates HTML and browser script that is sent to the browser along with a style sheet to establish the page data and layout.

erate the HTML files that are sent to the client browsers. One issue to watch for when selecting tools is that some of them require users to download special add-in software for the browsers. Most users are wary of downloading nonstandard components. When you are dealing with users outside of the main company, it is best to stick with tools that use only standard browser features. At the moment, the three leading tools are: (1) Microsoft ASP .NET, (2) Java, such as Oracle's JDeveloper and IBM's WebSphere, (3) UNIX-based scripting, particularly LAMP (Linux, Apache Web Server, MySql, and PHP or Python).

Figure 11.19 shows the basic process of connecting a DBMS to a Web server. As the developer, you create a code file on the server. The file contains data objects that pass SQL commands to the DBMS and receive results (and error codes). The page code can contain complex SQL statements and conditions. Ultimately, it generates the HTML code and browser script that is sent to the client's Web browser. Most systems also use a **cascading style sheet (CSS)** to establish the overall page design elements. The syntax of the code varies enormously, depending on the server. Pages written for one server system generally cannot be translated to run on a different system. Consequently, the choice of server and tools is a critical first step in building data-driven Web applications.

Systems that separate the code from the HTML and the styles have a substantial advantage. In particular, most organizations hire Web graphics designers and usability experts to design the final Web page that is seen by users. These people



Black Cat

7	Eugene	Cat	Bombay	1/25/01	Black	$279.54
18	Allison	Cat	Manx	4/21/01	Black	$215.92
20	Buck	Cat	Bombay	4/30/01	Black	$213.18
65	Bryce	Cat	Persian	6/23/01	White/Black	$133.24
77	Damon	Cat	Bombay	1/17/01	Black	$178.16
98	Stan	Cat	Sphynx	1/1/01	Black	$241.67
103	Lee	Cat	Siamese	7/17/01	White/Black	$232.90
106	Taylor	Cat	Oriental Shorthair	2/21/01	Yellow/Black	$220.94
115	Jewell	Cat	Korat	1/15/01	Black	$210.88
122	Gayle	Cat	Bombay	2/14/01	Black	$225.68
168	Ray	Cat	Japanese Bobtail	6/4/01	Black	$225.68
192	Sally	Cat	American Shorthair	6/26/01	Black	$198.87

Figure 11.20

Client perspective. The client enters data into a form. Clicking the Search button sends the data to a server page. The server page retrieves the matching data from the DBMS and formats a new HTML page. This table is returned to the user, along with additional choices.

rarely know how to write database code. If the system mixes code with layout and design elements, it is difficult for the designers to alter the server files. By separating the three elements (code, layout, and style), the designers can fine tune their work without affecting any of the underlying procedural code. Similarly, the style sheet is easily modified by designers. If the style sheet is used for every page item, it is possible to change the overall look and feel of a Web site simply by editing this one file.

Browser and Server Perspectives

On the client browser the user will see a simple sequence like the forms shown in Figure 11.19. Once someone chooses a Search option, the AnimalSearch form is displayed on his or her browser. The user chooses a category and enters a color. When the Search button is clicked, the choices are sent to a new page on the server. This page retrieves the data and formats a new page. The data is generally stored in a table, similar to the one shown in Figure 11.20. The user never needs to know anything about the DBMS: Users see only forms and new pages. Each new page should provide additional choices and links to other pages.

Vendors are busy creating and refining tools to create Web-based forms that can interact easily with the database. The details vary enormously depending on which tool you use. However, the basic process remains similar. The server takes the values from the client form, validates them for rules you specify, and writes SQL queries to insert them or update existing rows in the database. You should review all of the issues discussed in Chapter 6 for building forms and reports. The main difference with the Web is that today's tools generally require more programming and individualized attention. On the other hand, Web-based applications create some potentially difficult problems that must be addressed. Data transmission,

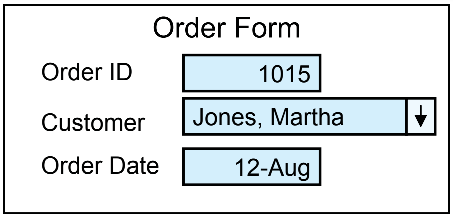


Figure 11.21

Data transfer in forms. What if there are 10,000 customers? How long will it take to load the selection box? How long will it take to refresh a page with several selection boxes? How can a user possibly read and scroll all 10,000 entries?

concurrency, and server loads are significant issues that arise in Web-based applications. In fact, some of the most important differences in vendor tools can be found in how these problems are solved.

Data Transmission Issues in Applications

How much data can you send to a client form? At first glance, it seems straightforward to build a client/server or Web-based application. You simply move the database to a central server and use the network connections to handle the data transfer. However, these applications can present some challenging issues for data transfer and usability of forms. One of the most difficult issues is the use of drop down list boxes on a form.

Consider the main section of a standard order form shown in Figure 11.21. If you build this form in Access or Visual Studio and run it locally over a fast network, it will run fine. But what happens if there are 10,000 customers or the form runs at a remote location with a slow network? For simplicity, assume those names and identification numbers average 20 Unicode characters for each customer. So the selection box needs 10,000 times 20 times 2 or 400,000 bytes of data. At 8 bits per byte that is 3.2 million bits of data to transfer. Even at 3 mbps it would take a little over 3 seconds just to transfer the data for that one lookup box. If the network is slower, such as 300 kbps on a cell hone, it would take 30 seconds. Anytime you need to refresh the form or reload the drop down list box, it takes another 30 seconds. If your form has several selection boxes, the form takes even longer to load. Most users will be unhappy with the performance if forms take more than a couple seconds to load.

So why not just remove the drop down list box? To understand the issues, you need to remember why the selection box is useful. In a relational database, data is stored in separate tables that are joined through key data. In this case, the Order table contains the CustomerID. Theoretically, to place an order you simply need the customer's ID. (Eventually you will also need individual identification numbers for products as well.) You cannot expect your customers or clerks to memorize ID numbers, so the order form uses the drop down list box to look up customers alphabetically and return the matching ID number for the selected customer. If you remove the list box, you need to rethink the usability and find another method for customers and clerks to enter data.

Even without the data transfer issue, a selection box might not be the best solution when it has thousands of entries. Some boxes try to automatically find a

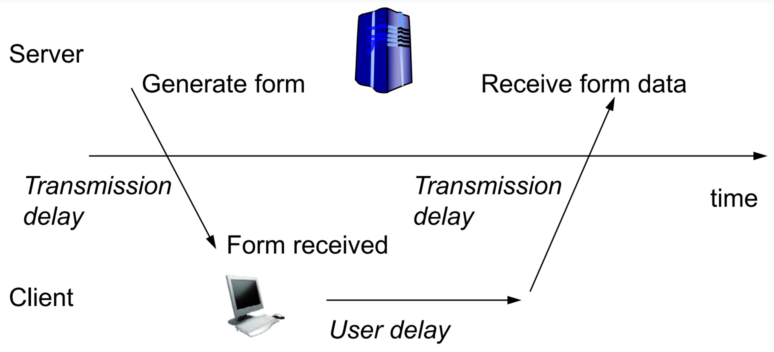


Figure 11.22

Latency. Transmission delays and user delays can create long latency for Web forms. Avoid pessimistic locking and carefully test for changes in the underlying database.

matching entry as a user enters the first few characters of a name or product, but this method still requires the user to know the first few characters. Hence, a potentially better solution is to create a more detailed search mechanism. Instead of a selection box containing all customers, the user could enter the first few characters of a customer's last name, click a button, and receive a small list of matching names.

ItemIDs present similar problems. Two common solutions exist: (1) For in-store sales, attach the product ID numbers to the individual items (e.g., bar codes); or (2) for Web sales, let the customer search for items and keep a collection of the selected ID numbers in a shopping cart.

For situations where you still want to use selection boxes, you need to be more creative with programming. For example, Oracle recommends that you do not use selection boxes for lists with more than 30 items. Instead, Oracle suggests the use of a standard text box, along with a **list of values (LOV)**. A list of values can be defined as a query. When the user enters the text box, the item can be selected from the list of values. How is this approach different from a selection box? The main difference lies beneath the surface. The LOV retrieves data in chunks instead of trying to transfer the entire set at one time. To the user, the list appears continuous, but by transferring only the currently displayed section of the list, the LOV reduces transmission time. It also transfers data while the user is reading, so the delay is less noticeable. This approach can be used, even if your tool or vendor does not support it directly. In Web forms, simply create a second form that holds the list of items on multiple pages with a search function. When users find the specific item, you can write a function to transfer the selected item to the main form. On the Web, this approach requires some slightly tricky Javascript (or ECMA script) coding.

For similar data transmission reasons, concurrency is also a problem with Web forms. In engineering terms, **latency** is a time delay in a system. In the context of forms, latency is the time between generating the form and receiving a response from the user. With a long latency, there is a greater opportunity for someone else to modify the same data elements, so concurrency is a greater problem. As shown in Figure 11.22, latency is typically longer on Web forms because of slow trans-

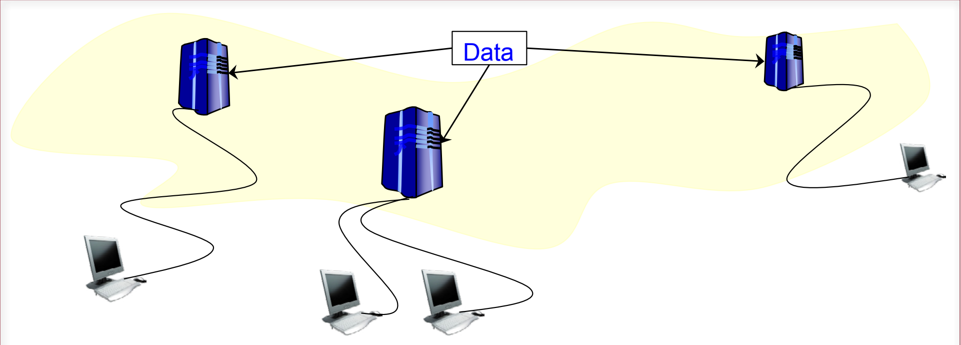


Figure 11.23

Cloud computing. Data is replicated to multiple, connected servers in the Internet cloud. Client requests are filled from the nearest available server, spreading the bandwidth and processor demands across the network.

mission lines and because users may be casual browsers who wander off and do other tasks before submitting the form. Consequently, Web-based applications should avoid pessimistic locking so that the data is available to more people at the same time. As a result, your application has to test and handle optimistic concurrency issues when the data has been changed by another process.

Cloud Databases

What benefits are provided by cloud computing and data storage? As the Internet has expanded, some of the structures have changed. In particular, a few large companies have become leaders in developing tools and in establishing multiple high-speed connections. These companies are led by Amazon, Microsoft, and possibly Google. All of them have huge data centers with very high-capacity Internet connections. Beyond their underlying businesses, they offer other firms the ability to use their computing and network facilities—for a fee of course.

Think about Web-based servers for a second. Typically, you think about one big server and its Internet connection. An expensive Internet connection might be capable of handling 155 megabits per second, which sounds like a lot of capacity. But, what happens if your server (and database) needs to deliver content to a million users—at the same time? The network becomes congested and you are averaging only 155 bits per second per user—way too slow for transferring large amounts of data. OK, perhaps a million simultaneous hits is high, but similar problems quickly arise with a smaller number of users when the Web server is delivering complex content such as images or video. As shown in Figure 11.23, some companies—notably Akamai—help reduce this problem by creating hundreds of data centers around the globe. Your data gets duplicated and distributed. Users interacting with your server might actually receive data from a nearby data center—reducing the overall processing and transmission load on any one server.

Cloud Computing Basics

Cloud computing arises by locating multiple servers and databases on the Internet. You can purchase time, space, and network data transfers by renting a virtual machine and virtual storage space on these distributed servers. At the simplest level, you can store files on a virtual Web folder (e.g., Amazon S3). You are charged a monthly storage fee along with monthly fees based on the amount of data transferred. This approach is typically used to store large files (books, music, videos, and so on) that need to be downloaded by people around the world. The service provider has high-speed data connections that can support a huge number of users, but you only pay for the actual usage rates.

In more complex scenarios, you can rent SQL Server data storage and query processing (e.g., Microsoft). The data is actually stored in Microsoft's cloud, and that company runs the network, hardware, and backup facilities. Your Web site issues queries and stores data, but the data is stored on these distributed servers.

It is even possible to rent virtual machine servers (e.g., Amazon EC2). These computers can be configured in a variety of sizes and with almost any operating system or software you can find. You pay a monthly fee based on the relative size/performance needed. Again, the service provider ensures that the computers keep running and provides network access. One of the key features of these virtual machines and cloud databases is that they can be expanded (or contracted) at almost any time. Also, initial fixed costs are low.

Think about the problem from the perspective of an Internet entrepreneur with a new idea. You could build your own data center, hire a staff, and buy hardware, software, and network access for hundreds of thousands of dollars. But you really do not know how much capacity you need. Alternatively, you can rent all of the initial capacity you need from a cloud provider for a monthly fee with little or no startup cost. If your company takes off and you get a huge increase in customers, you simply scale up the processing and storage with the cloud vendor.

However, you must continually re-evaluate your costs. The monthly rates charged by the cloud providers will ultimately be higher than those that you might achieve on your own. Once a company reaches a relatively steady-state, it might be cheaper to install your own computers or lease servers directly from a hosting company. But the decision depends on how much your demand fluctuates, and whether you want to hire people to configure and run your own servers.

Data Storage in the Cloud

Web-based data is interesting. It might or might not resemble traditional transaction data. For example, images, files, audio, and video are common on the Web. These items could be stored in a relational DBMS, but it is generally easier to store them as separate files—perhaps storing the file location in a relational table. Cloud vendors have initially concentrated on tools to handle these types of object data. For example, Amazon's S3 and Google's Bigtable are designed to store large chunks of object data. They are not relational databases and support only rudimentary query operations. However, they are distributed and have the ability to deliver data quickly—to anywhere in the world. For the most part, think of them as keyed storage—you provide a key value and the database returns the matching item stored at that location. (The systems, particularly Bigtable, are more flexible than that but it is a good starting point.)

When you need more traditional relational database services, you can also rent those—such as Microsoft's SQL Server offerings through its Azure platform.

In all cases, the data storage is most commonly used in combination with Web applications. But, Web applications are often used for in-house systems as well as public data, so this usage is not a limitation. Essentially, you treat the cloud server as a basic database server. The Web server (which could also be in the cloud), sends key values or queries to the cloud database, then formats and displays the returned data to place it into an HTML page.

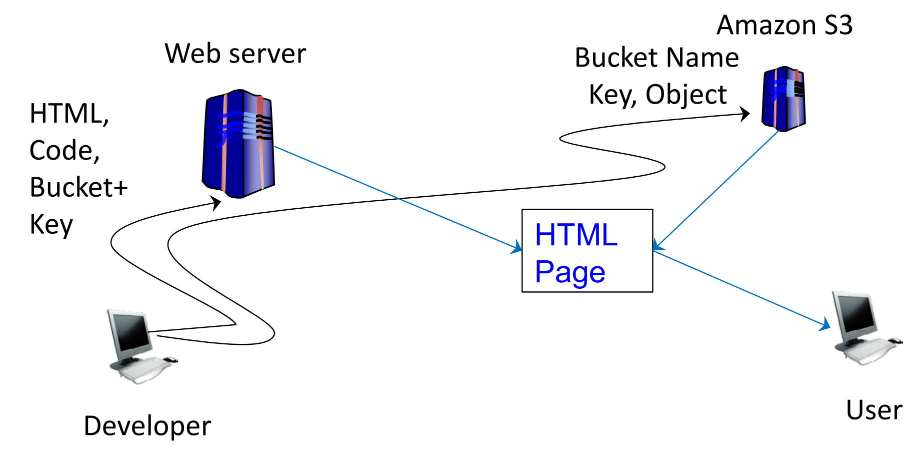


Figure 11.24

Amazon S3 process. Developer uploads object to an Amazon S3 account into a named bucket and gives the object a key value. The Web server code includes the HTML and the link to the S3 service with the bucket name and key. The Web.server delivers the HTML page and the nearest S3 server delivers the referenced content.

Figure 11.24 shows the basic process for the Amazon S3 service. Developers define buckets in Amazon, which are similar to file folders. Objects are uploaded to a bucket and assigned a key value. The developer creates pages on the Web server (which does not have to be at Amazon), and includes the bucket and key names to tell the browser to insert objects from the S3 service. The bucket and key names might actually be stored in a relational database and assigned based on some actions or choices by the user. For example, a page request might retrieve the bucket name and ID value to display a photo of an item being purchased. The Web server encodes the bucket name and key into a URL such as http://s3.amazonaws.com/mybucket/mykey. When the browser sees that link, it retrieves the specified object from the nearest available S3 server. Objects are generally uploaded to the S3 service using a simple transfer tool such as the add-in for the Fire Fox browser.

Microsoft's Azure SQL service is even easier. You interact with the service by writing SQL statements and upload data into tables. Then you write the Web page code using a database connection that specifies the Azure SQL service as the database provider. Your Web server runs the SQL statements and sends them to the Azure server which returns the values, and you insert the results into the Web page. The details are straightforward with most current Web programming platforms including Microsoft .NET and Java's JDBC.

Sally's Pet Store

How will Sally's employees access the database? Even with relatively simple applications, you need to think about how employees and managers will access the database. As a retail store, Sally's employees will need access to terminals or personal computers to record sales transactions and purchases or lookup customer or item data. Managers will have to print reports and browse for trends over time. You still need to determine where these computers will be located, how they will be connected, and where the data will be stored. Some of the answers depend on which DBMS you are using and the physical layout of the store. For example, will Sally's have a single checkout counter near the entrance with one terminal, or will there be multiple terminals scattered throughout the store? If you use Microsoft Access as the DBMS with multiple transaction computers, you will need to split the database to store the shared data tables on a file server. Or, you might transfer the tables to SQL Server and just use Access for the front-end forms and reports. With Oracle, a single server would hold all of the data, and the front-end tools would run on basic Web browsers. Since the machines are all contained within the store, you can install a relatively high-speed network and not worry about transfer speeds. If Sally wants access to the database from home, you will have to experiment with options.

If Sally wants to expand and add a second store, the decisions become more complex. She is also pushing for creation of a Web site, so that customers can order products, check on adopting animals, and get some help in caring for their pets.

Sally's request to expand the database to a second store raises many questions. Does she need "instant" access to the sales data from both stores all the time? Do the stores need to share data with each other? For example, if a product is out of stock at one store, does Sally want the system to automatically check the other store? Will the stores operate somewhat independently—so that sales and financial data are maintained separately for each store—or will the data always be merged into one entity? How up-to-date does data need to be? Is it acceptable to have inventory data from yesterday, or does it need to be up-to-the-minute?

The answers to these questions determine some crucial design aspects. In particular, the primary design question to answer is whether one central database should handle all sales or separate, distributed databases should handle each store. The answer depends on how the stores are managed, the type of data needed, the network capabilities and costs, and the capabilities of the DBMS.

In many ways, initially the cheapest solution is to keep the second store completely independent. Then there is no need to share data except for some basic financial information at the end of each accounting period. A second advantage of this approach is that it is easy to expand since each new store is independent. Similarly, if something goes wrong with the computer system at one store, it will not affect the other stores.

However, at some point Sally will probably want a tighter integration of the data. For example, the ability to check inventory at other local stores can be a useful feature to customers, which means that the application will need to retrieve data from several databases, located in different stores. These distributed databases must be networked through a telecommunications channel. There are many ways to physically link computers, and you should take a telecommunications course to understand the various options. Once the computers are physically linked, you need to deal with some additional issues in terms of creating and managing the distributed databases.

Summary

As organizations grow, distributed databases become useful. Distributed databases enable the company to expand individual departments without directly affecting everyone else. Distributed databases also give individual departments increased control and responsibility for their data. However, distributed databases, with in-

dependent database engines running in different locations, increase the complexity of developing and managing applications. One of the primary goals is to make the location of the data transparent to the user. To accomplish this goal, developers and DBAs need to carefully define the databases, networks, and applications.

Some of the major complications generated by distributed databases are query optimization; data replication questions; and support for transactions, concurrency controls, and deadlock resolution. These issues become even more complex when multiple databases are involved. Network transfers of data are substantially slower than transfers from local disk drives. Transfers over wide area networks can be slow and costly. These factors imply that developers must carefully design the applications and the data distribution strategy. The applications also have to be tested and monitored for performance and cost.

One of the major strategies in designing and controlling distributed databases is to replicate data. Instead of maintaining one source, it is often more efficient to replicate data that is heavily used in multiple locations. Of course, replication requires additional disk space, along with periodic updates and transfers of the data changes to each copy. Replication saves time by providing local access to data. It reduces costs by reducing the need for a full-time high-speed connection. Instead, bulk data is transferred at regular intervals—preferably at off-peak communication rates.

Client/server networks and client/server databases are a common means to design applications and distribute databases. Clients usually run applications on personal computers, and most of their power is devoted to the user interface. The data is maintained on a limited number of database servers, which are more efficient than simple file-server transfers. With a server database, the client sends an SQL query, and the server processes the request and returns only the desired data. With a file server, the client computer performs all the processing and must retrieve and examine all the data.

Larger, object-oriented applications are being built using a three-tier client/server architecture. The additional layer is in the middle and consists of business rules and program code (business objects) that execute on servers. The middle layer is also responsible for pulling data from the database servers and reformatting it for use by the client applications. Separating the three layers makes it easier to modify each component without interfering with the other elements.

The World Wide Web is becoming a popular mechanism to centralize applications and solve some of the distributed database issues. Web browsers have limited capabilities, but standards make it easier for everyone to get access to the applications and data. Keeping the data in one location simplifies security and concurrency issues. You still have to think about reducing the data transferred to client computers, such as avoiding huge drop down lists. You also have to deal with transferring data to external suppliers and customers. XML is a standard method of transferring bulk data. XML stores data in hierarchical files and XQuery provides searches of those files.

A Developer's View

Like Miranda, most developers understand the importance of the Web. The client standards make it easier to distribute data and connect with users around the world. Additionally, as applications expand, it becomes necessary to create distributed databases to improve performance and to support different regions. Distributed databases can significantly complicate application development. First be sure the application runs on one computer. Then get the best software you can afford. As much as possible, let the server databases perform the data manipulation and computation tasks. Use the client computers to display the results. Learn as much as you can about the Internet: It changes constantly, but will become increasingly important in your applications. For your class project, you should identify where the company might expand and where you would position distributed computers to support it. Explain how the database design would change in a distributed environment.

Key Terms

active data objects (ADO)
browser
cascading style sheet (CSS)
cloud computing
cluster
distributed database
fault tolerance
globally-unique identifier (GUID)
latency
list of values (LOV)
local area network (LAN)
replicate
replication manager
scalability
three-tier client/server
two-phase commit
wide area network (WAN)
World Wide Web

Review Questions

1. What are the strengths and weaknesses of distributed databases?
2. Why might a query on a distributed database take a long time to run?
3. When would you want to replicate data in a distributed database?
4. Why is concurrency a bigger problem with distributed databases than with stand-alone databases?
5. How does the two-phase commit process work?
6. Why is a client/server database more efficient than a database on a simple file server?
7. What are the advantages of the three-tier client/server approach?
8. How does a central Web site reduce problems with distributed databases?
9. How do you reduce transmission delays within data-driven Web sites?
10. What benefits are provided by databases run on the Internet as cloud computing?

Exercises

1. In each of the following situations, identify the best method of structuring the databases.

   A. A single retail store with 7 checkout lanes, a manager's office, and an owner who wants to review reports from home.

   B. Three retail stores with the same owner located in different cities about 50 miles apart. Each with 5 checkout lanes.

   C. An engineering firm with computers used for design work where engineers spend considerable time at production sites.

   D. A large marketing firm with major offices in Los Angeles and New York, where each office mostly works with local clients, but some work is shared.

   E. A large agricultural firm with a custom application used to collect and analyze production data with headquarters in one state and farms located in at least three other states.

2. ✓ Assuming your DBMS cannot generate distributed safe keys automatically write a procedure to generate key values based on a location.

3. Compare the cost of using Amazon RDS versus Microsoft SQL Server cloud services. Assume the database content is about 10 GB with monthly data transfer rates around 1 GB per month.

4. You have the following distributed databases:

Location	Link Speed	Tables	Sizes

London	53 kbps	Contact(ContactID, Name, ClientID, Title, Phone)
		Employee(EmployeeID, Name, Phone, Title)
		WorkHours(WorkID, EmployeeID, ClientID, Date, Hours)
			5,000 rows
			2,300 rows
			3,000,000 rows

Paris	1.544 mbps	Contact(ContactID, Name, ClientID, Title, Phone)
		Employee(EmployeeID, Name, Phone, Title)
		WorkHours(WorkID, EmployeeID, ClientID, Date, Hours)
			10,000 rows
			1,000 rows
			20,000,000 rows

Frankfort	128 kbps	Contact(ContactID, Name, ClientID, Title, Phone)
		Employee(EmployeeID, Name, Phone, Title)
		WorkHours(WorkID, EmployeeID, ClientID, Date, Hours)
			7,000 rows
			3,500 rows
			30,000,000 rows

Madrid (HQ)	local	Client(ClientID, Name, Lead contact, Main city)
		Employee(EmployeeID, Name, Phone, Title)
		Project(ProjectID, ClientID, StartDate, Topic)
			20,000 rows
			10,000 rows
			1,000,000 rows

You are working for an accounting firm with headquarters in Madrid and major offices in London, Paris, and Frankfort. Many of the client companies have offices in three or four of these cities. Some clients are smaller and work with a single office. The accounting teams in the various offices need to share documents with teams in the other offices when they are working for the same client. Each office maintains a database of working papers, spreadsheets, questions, answers, and workflow data for the team. It also tracks billable hours for each employee and client. You need to get a list of all employees who have worked for a particular client in the last month; along with hours worked. Based on the communication speeds and table sizes, design the best performing query to answer this question. Could the database and query performance be improved by changing the distributed design?

5. A company has a database and an application where managers often generate and read a report that consists of 5 pages of dense text and numbers plus a chart on each page. If 100 people routinely view this report (with different data) each hour, and if the company wants to run the report on a central Web server, how much bandwidth capacity is needed? If the application is converted to tablets using cell-phone connections, how much will it cost in monthly cell phone bills?

6. You are working for a company that has two offices and is using a replicated database approach. Both offices have copies of the database of about 100 GB. Assuming that the entire database has to be transferred (both ways) between the two offices to handle synchronization, estimate the time required to handle the updates using at least three common network speeds.

7. A company wants managers to use mobile devices (phones, tablets, laptops) to access its database-driven applications. Briefly explain the different ways there are to develop the client (tablet/phone) applications. Which method would you select?

8. Find a development tool that can be used to create database-driven application on a Web server. Briefly describe the commands used to connect a server page to a database and explain what needs to be changed if the underlying DBMS is changed.

9. A company is building a Web-based application with SQL Server (or Oracle) as the backend DBMS. The middle tier uses a server-based programming language to generate HTML pages and insert data retrieved from the DBMS. The project manager wants on section of the application to implement pessimistic locking but does not think the application software can adequately handle it. Over half of the large application has already been written. What can you do to address the issue?

Sally's Pet Store

10. Sally is planning to add a second store. Write a plan that describes how the data will be shared. How will you control and monitor the new system? Which tools will you add?

11. Sally wants to connect to suppliers so that she can get information on orders and shipments electronically. The data needs to be imported into her database and matched to the orders. Describe a system that can handle these tasks.

12. Sally is thinking about implementing a Web site to sell pet products. Estimate the costs of storing the data for the site on Microsoft Azure SQL servers. The database would need to store the tables for Customer, Merchandise, Sale, and SaleItem. As an initial estimate, assume there will be 3,000 merchandise items, 10,000 customers with each customer placing an average of one purchase a month consisting of an average of three items on each order.

13. Use the tables for a different DBMS, or create them if necessary. Try to connect your primary DBMS to the new tables. Write a query that pulls data from both DBMSs.

14. Build a front-end application that handles the Sales form and connects to a database server.
15. Briefly explain the steps required (and estimate the time) to build an application so that store clerks could carry tablets on the floor to look up inventory or take special orders from customers.

Rolling Thunder Bicycles

16. Rolling Thunder is planning to expand to a second location across the country. How should the database be distributed? Where should each table be stored? Which tables should be replicated, and how should the data changes be reconciled?
17. Rolling Thunder is planning to expand by sending sales representatives around the country to various bike shops. They will use portable devices and a Web interface to configure bicycles and take new orders. The system should at least be able to run on an Apple iPad browser, and perhaps even a cell-phone browser. Describe how this system will work. What security provisions will be needed?
18. Describe a method to create a Web application that enables customers to check on the progress of their bicycle orders.
19. Create a second copy of your database running on a second computer. Create a link from the first database to the copy. Write a query that combines data from at least one table in each database.
20. Assume the owners want to convert the entire application to the cloud (such as Amazon RDS or Microsoft SQL Server/cloud). In the process, they want to increase marketing to increase sales and production to ten times the level in 2012. Estimate the new database size. Assuming customers place their own orders over the Web, estimate the cloud hosting costs.
21. Using the DBMS tools available to you, create a replica of the database; make changes to data in both copies and then synchronize the database to see the changes. Test what happens if you change the same data (such as customer phone number) in both copies before synchronizing.
22. Assume the managers want to temporarily connect a SQL Server of the database to an Oracle database with other tables. How can you build a link between SQL Server and Oracle that lets you run Oracle queries inside SQL Server?

Corner Med

23. The company owners basically want to franchise the operations. The headquarters will run database operations for all of the local clinics. Describe how you will configure the database to support this operational process. List any potential problems you might encounter.
24. Using one DBMS, research the capabilities for replicating data. Build at least one replica of a table, make changes to both copies, and synchronize the copies. Describe how the system handles generated keys.

25. If Corner Med decides to franchise and move the database to cloud computing, what security and privacy problems might arise and how could they be managed?
26. Assume the company has several offices, but physicians (and sometimes patients) move among the offices during the month. So the company builds the application on a centralized server with access through Web-based forms and reports. Would you add pessimistic locking to any of the tables or forms?
27. Related to distributed databases, assume that insurance companies send you payment data by downloading data files. The CSV files contain a line for each payment and list: Customer SSN, LastName, FirstName, VisitDate, PaymentAmount, and AmountDenied (not paid). Create an application to read this data and update the insurance company payment data. You might need to add more columns to existing tables, and you should create a sample file to test your application.
28. Create a second copy of your database running on a second computer. Create a link from the first database to the copy. Write a query that combines data from at least one table in each database.

Web Site References

http://www.w3.org/	Web standards body.
http://msdn.microsoft.com	Search for Microsoft's .NET framework and documentation.
http://www.oracle.com/technetwork/java/index.html	Java and JDBC documentation and references.

Additional Reading

R. Burns, D. Long, and R. Rees, Consistency and Locking For Distributing Updates to Web Servers using a File System, *ACM SIGMETRICS Performance Evaluation Review*, 28(2) September 2000, 15-21. [Performance issues in replicated databases.]

Date, C. J., *An Introduction to Database Systems*, 8th ed. Reading, MA: Addison-Wesley, 2003. [In-depth discussion of distributed databases.]

Fisher, M., J. Ellis, and J. Bruce, *JDBC API Tutorial and Reference/3e*, Boston: Addison-Wesley, 2003. [Using JDBC and Java to connect to database.]