CHAPTER 11

Data and Database Administration

Learning Objectives

After studying this chapter, you should be able to:

- Concisely define each of the following key terms: **data administration, database administration, open source DBMS, database security, authorization rules, user-defined procedures, encryption, smart card, database recovery, backup facilities, journalizing facilities, transaction, transaction log, database change log, before image, after image, checkpoint facility, recovery manager, restore/rerun, transaction boundaries, backward recovery (rollback), forward recovery (rollforward), aborted transaction, database destruction, concurrency control, inconsistent read problem, locking, locking level (lock granularity), shared lock (S lock, or read lock), exclusive lock (X lock, or write lock), deadlock, deadlock prevention, two-phase locking protocol, deadlock resolution, versioning, data dictionary, system catalog, information repository, Information Resource Dictionary System (IRDS), data archiving,** and **heartbeat query**.
- List several major functions of data administration and of database administration.
- Describe the changing roles of the data administrator and database administrator in the current business environment.
- Describe the role of data dictionaries and information repositories and how they are used by data administration.
- Compare the optimistic and pessimistic systems of concurrency control.
- Describe the problem of database security and list five techniques that are used to enhance security.
- Understand the role of databases in Sarbanes-Oxley compliance.
- Describe the problem of database recovery and list four basic facilities that are included with a DBMS to recover databases.
- Describe the problem of tuning a database to achieve better performance, and list five areas where changes may be made when tuning a database.
- Describe the importance of data availability and list several measures to improve availability.

INTRODUCTION

ChoicePoint—More ID theft warnings: ID company says criminals able to obtain almost 140,000 names, addresses and other information.

> ChoicePoint, Inc., a national provider of identification and credential verification services, says it will send an additional 110,000 statements to people informing them of possible identity theft after a group of well-organized criminals was able to obtain personal information on almost 140,000 consumers through the company.
>
> According to a statement on the ChoicePoint Web site, the incident was not the result of its systems being hacked but rather caused by criminals posing as legitimate businesses seeking to gain access to personal information.
>
> ChoicePoint said the criminals may have gained access to people's names, addresses, Social Security numbers, and credit reports.
>
> The company said Tuesday it sent warning letters to 30,000 to 35,000 consumers in California, the only state that requires companies to disclose security breaches.
>
> Although the company knew about the fraud last fall, it said it did not reveal the information until now at the request of authorities, who said it would jeopardize the investigation.
>
> ChoicePoint said 35,000 California residents have already been notified and another 110,000 people outside of California will receive notice soon.
>
> Alpharetta, Ga.-based ChoicePoint maintains personal profiles of nearly every U.S. consumer, which it sells to employers, landlords, marketing companies and about 35 U.S. government agencies.
>
> ChoicePoint's databases contain 19 billion public records, including driving records, sex-offender lists, and FBI lists of wanted criminals and suspected terrorists.
>
> The company says its records enable law enforcers to track down serial killers and have helped find 822 missing children.
>
> (*Source: CNN Money* Web site, February 17, 2005.)

The critical importance of data to organizations is widely recognized. Data are a corporate asset, just as personnel, physical resources, and financial resources are corporate assets. Like these other assets, data and information are too valuable to be managed casually. The development of information technology has made effective management of corporate data far more possible, but data are also vulnerable to accidental and malicious damage and misuse. Data and database administration activities have been developed to help achieve organizations' goals for the effective management of data.

Ineffective data administration, on the other hand, leads to poor data quality, security, and availability and can be characterized by the following conditions, which are all too common in organizations:

1. Multiple definitions of the same data entity and/or inconsistent representations of the same data elements in separate databases, making integration of data across different databases hazardous
2. Missing key data elements, whose loss eliminates the value of existing data
3. Low data quality levels due to inappropriate sources of data or timing of data transfers from one system to another, thus reducing the reliability of the data
4. Inadequate familiarity with existing data, including awareness of data location and meaning of stored data, thus reducing the capability to use the data to make effective strategic or planning decisions
5. Poor and inconsistent query response time, excessive database downtime, and either stringent or inadequate controls to ensure agreed upon data privacy and security
6. Lack of access to data due to damaged, sabotaged, or stolen files or due to hardware failures that eliminate paths to data users need
7. Embarrassment to the organization because of unauthorized access to data

Many of these conditions put an organization at risk for failing to comply with regulations, such as the Sarbanes-Oxley Act (SOX), the Health Insurance Portability and Accountability Act (HIPAA), and the Gramm-Leach-Bliley Act for adequate internal controls and procedures in support of financial control, data transparency, and data privacy. Manual processes for data control are discouraged, so organizations need to implement automated controls, in part through a DBMS (e.g., sophisticated data validation controls, security features, triggers, and stored procedures), to prevent and detect accidental damage of data and fraudulent activities. Databases must be backed-up and recovered to prevent permanent data loss. The who, what, when, and where of data must be documented in metadata repositories for auditor review. Data stewardship programs, aimed at reviewing data quality control procedures, are becoming popular. Collaboration across the organization is needed so data consolidation across distributed databases is accurate. Breaches of data accuracy or security must be communicated to executives and managers.

Morrow (2007) views data as the lifeblood of an organization. Good management of data involves managing data quality (as discussed in Chapter 10) as well as data security and availability (which we cover in this chapter). Organizations have responded to these data management issues with different strategies. Some have created a function called *data administration*. The person who heads this function is called the data administrator (DA), or information resource manager, and he or she takes responsibility for the overall management of data resources. A second function, *database administration*, has been regarded as being responsible for physical database design and for dealing with the technical issues, such as security enforcement, database performance, and backup and recovery, associated with managing a database. Other organizations combine the data administration and database administration functions. The rapidly changing pace of business has caused the roles of the data administrator and the database administrator (DBA) to change, in ways that are discussed next.

THE ROLES OF DATA AND DATABASE ADMINISTRATORS

Several new technologies and trends are driving the changes in the data administration and database administration roles (Mullins, 2001):

1. The proliferation of proprietary and open source technologies and databases on diverse platforms that must be managed concurrently in many organizations
2. Rapid growth in the size of databases, fueled by the storage of complex data types and the business intelligence needs of today's organizations
3. The embedding of business rules in databases in the form of triggers, stored procedures, and user-defined functions
4. The explosion of e-business applications that require linking corporate databases to the Internet and tracking Internet activity, thus making databases more open for unauthorized access from outside the organization

Against the background of these changes, it is important to understand traditional role distinctions. This will help us understand the ways in which the roles are being blended in organizations that have different information technology architectures.

Traditional Data Administration

Databases are shared resources that belong to the entire enterprise; they are not the property of a single function or individual within the organization. Data administration is the custodian of the organization's data, in much the same sense that the controller is custodian of the financial resources. Like the controller, the data administrator must develop procedures to protect and control the resource. Also, data administration must resolve disputes that may arise when data are centralized and shared among users and must play a significant role in deciding where data will be stored and managed. **Data administration** is a high-level function that is responsible for the overall management of data resources in an organization, including maintaining corporate-wide data definitions and standards.

Data administration
A high-level function that is responsible for the overall management of data resources in an organization, including maintaining corporate-wide definitions and standards.

Selecting the data administrator and organizing the function are extremely important organizational decisions. The data administrator must be a highly skilled manager capable of eliciting the cooperation of users and resolving differences that normally arise when significant change is introduced into an organization. The data administrator should be a respected, senior-level manager selected from within the organization, rather than a technical computer expert or a new individual hired for the position. However, the data administrator must have sufficient technical skills to interact effectively with technical staff members such as database administrators, system administrators, and programmers.

Following are some of the core roles of traditional data administration:

- ***Data policies, procedures, and standards*** Every database application requires protection established through consistent enforcement of data policies, procedures, and standards. Data policies are statements that make explicit the goals of data administration, such as "Every user must have a valid password." Data procedures are written outlines of actions to be taken to perform a certain activity. Backup and recovery procedures, for example, should be communicated to all involved employees. Data standards are explicit conventions and behaviors that are to be followed and that can be used to help evaluate database quality. Naming conventions for database objects should be standardized for programmers, for example. Increased use of external data sources and increased access to organizational databases from outside the organization have increased the importance of employees' understanding of data policies, procedures, and standards. Such policies and procedures need to be well documented to comply with the transparency requirements of financial reporting, security, and privacy regulations.
- ***Planning*** A key administration function is providing leadership in developing the organization's information architecture. Effective administration requires both an understanding of the needs of the organization for data and information and the ability to lead the development of an information architecture that will meet the diverse needs of the typical organization.
- ***Data conflict resolution*** Databases are intended to be shared and usually involve data from several different departments of the organization. Ownership of data is a ticklish issue at least occasionally in every organization. Those in data administration are well placed to resolve data ownership issues because they are not typically associated with a certain department. Establishing procedures for resolving such conflicts is essential. If the administration function has been given sufficient authority to mediate and enforce the resolution of the conflict, they may be very effective in this capacity.
- ***Managing the information repository*** Repositories contain the metadata that describe an organization's data and data processing resources. Information repositories are replacing data dictionaries in many organizations. Whereas data dictionaries are simple data element documentation tools, information repositories are used by data administrators and other information specialists to manage the total information processing environment. An information repository serves as an essential source of information and functionality for each of the following:
  1. Users who must understand data definitions, business rules, and relationships among data objects
  2. Automated CASE tools that are used to specify and develop information systems
  3. Applications that access and manipulate data (or business information) in the corporate databases
  4. Database management systems, which maintain the repository and update system privileges, passwords, object definitions, and so on
- ***Internal marketing*** While the importance of data and information to an organization has become more widely recognized within organizations, it is not necessarily true that an appreciation for data management issues—such as information architecture, data modeling, metadata, data quality, and data standards—has also evolved. The importance of following established procedures and policies must be

proactively instituted through data (and database) administrators. Effective internal marketing may reduce resistance to change and data ownership problems.

When the data administration role is not separately defined in an organization, these roles are assumed by database administration and/or others in the IT organization.

Traditional Database Administration

Typically, the role of database administration is taken to be a hands-on, physical involvement with the management of a database or databases. **Database administration** is a technical function responsible for logical and physical database design and for dealing with technical issues, such as security enforcement, database performance, backup and recovery, and database availability. A database administrator (DBA) must understand the data models built by data administration and be capable of transforming them into efficient and appropriate logical and physical database designs (Mullins, 2002). The DBA implements the standards and procedures established by the data administrator, including enforcing programming standards, data standards, policies, and procedures.

Database administration
A technical function that is responsible for physical database design and for dealing with technical issues, such as security enforcement, database performance, and backup and recovery.

Just as a data administrator needs a wide variety of job skills, so does a DBA. Having a broad technical background, including a sound understanding of current hardware and software (operating system and networking) architectures and capabilities and a solid understanding of data processing is essential. An understanding of the database development life cycle, including traditional and prototyping approaches, is also necessary. Strong design and data modeling skills are essential, especially at the logical and physical levels. But managerial skills are also critical; a DBA must manage other information systems (IS) personnel while the database is analyzed, designed, and implemented, and the DBA must also interact with and provide support for the end users who are involved with the design and use of the database.

Following are some of the core roles assumed by database administration:

- ***Analyzing and designing the database*** The key role played by a DBA in the database analysis stage is the definition and creation of the data dictionary repository. The key task in database design for a DBA includes prioritizing application transactions by volume, importance, and complexity. Because these transactions are going to be most critical to the application, specifications for them should be reviewed as quickly as the transactions are developed. Logical data modeling, physical database modeling, and prototyping may occur in parallel. DBAs should strive to provide adequate control of the database environment while allowing the developers space and opportunity to experiment.
- ***Selecting DBMS and related software tools*** The evaluation and selection of hardware and software is critical to an organization's success. The database administration group must establish policies regarding the DBMS and related system software (e.g., compilers, system monitors, etc.) that will be supported within the organization. This requires evaluating vendors and their software products, performing benchmarks, and so on.
- ***Installing and upgrading the DBMS*** Once the DBMS is selected, it must be installed. Before installation, benchmarks of the workload against the database on a computer supplied by the DBMS vendor should be taken. Benchmarking anticipates issues that must be addressed during the actual installation. A DBMS installation can be a complex process of making sure all the correct versions of different modules are in place, all the proper device drivers are present, and the DBMS works correctly with any third-party software products. DBMS vendors periodically update package modules; planning for, testing, and installing upgrades to ensure that existing applications still work properly can be time-consuming and intricate. Once the DBMS is installed, user accounts must be created and maintained.
- ***Tuning database performance*** Because databases are dynamic, it is improbable that the initial design of a database will be sufficient to achieve the best processing performance for the life of the database. The performance of a database (query

and update processing time as well as data storage utilization) needs to be constantly monitored. The design of a database must be frequently changed to meet new requirements and to overcome the degrading effects of many content updates. The database must periodically be rebuilt, reorganized, and re-indexed to recover wasted space and to correct poor data allocation and fragmentation with the new size and use of the database.

- ***Improving database query processing performance*** The workload against a database will expand over time as more users find more ways to use the growing amount of data in a database. Thus, some queries that originally ran quickly against a small database may need to be rewritten in a more efficient form to run in a satisfactory time against a fully populated database. Indexes may need to be added or deleted to balance performance across all queries. Data may need to be relocated to different devices to allow better concurrent processing of queries and updates. The vast majority of a DBA's time is likely to be spent on tuning database performance and improving database query processing time.
- ***Managing data security, privacy, and integrity*** Protecting the security, privacy, and integrity of organizational databases rests with the database administration function. More detailed explanations of the ways in which privacy, security, and integrity are ensured are included later in the chapter. Here it is important to realize that the advent of the Internet and intranets to which databases are attached, along with the possibilities for distributing data and databases to multiple sites, have complicated the management of data security, privacy, and integrity.
- ***Performing data backup and recovery*** A DBA must ensure that backup procedures are established that will allow for the recovery of all necessary data should a loss occur through application failure, hardware failure, physical or electrical disaster, or human error or malfeasance. Common backup and recovery strategies are also discussed later in this chapter. These strategies must be fully tested and evaluated at regular intervals.

Reviewing these data administration and database administration functions should convince any reader of the importance of proper administration, at both the organizational and project levels. Failure to take the proper steps can greatly reduce an organization's ability to operate effectively and may even result in its going out of business. Pressures to reduce application development time must always be reviewed to be sure that necessary quality is not being forgone in order to react more quickly, for such shortcuts are likely to have very serious repercussions. Figure 11-1 summarizes how these data administration and database administration functions are typically viewed with respect to the steps of the systems development life cycle.

Trends in Database Administration

Rapidly changing business conditions are leading to the need for DBAs to possess skills that go above and beyond the ones described above. Here we describe three of these trends and the associated new skills needed:

1. ***Increased used of procedural logic*** Features such as triggers, stored procedures, and persistent stored modules (all described in Chapter 7) provide the ability to define business rules to the DBMS rather than in separate application programs. Once developers begin to rely on the use of these objects, a DBA must address the issues of quality, maintainability, performance, and availability. A DBA is now responsible for ensuring that all such procedural database logic is effectively planned, tested, implemented, shared, and reused (Mullins, 2002). A person filling such a role will typically need to come from the ranks of application programming and be capable of working closely with that group.
2. ***Proliferation of e-business applications*** When a business goes online, it never closes. People expect the site to be available and fully functional on a 24/7 basis. A DBA in such an environment needs to have a full range of DBA skills and also be capable of managing applications and databases that are Internet enabled (Mullins, 2001). Major priorities in this environment include high data availability

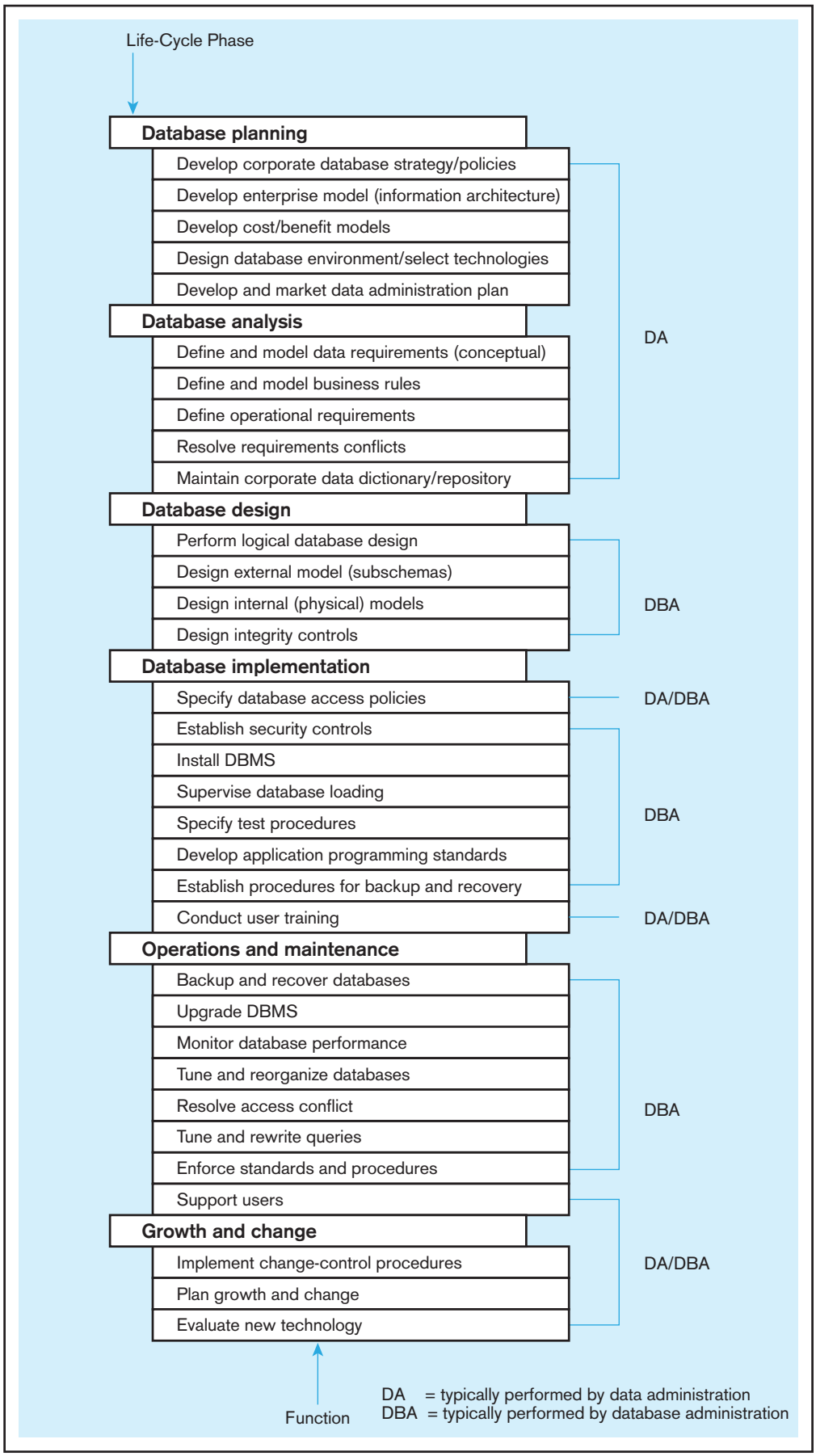


FIGURE 11-1 **Functions of data administration and database administration**

(24/7), integration of legacy data with Web-based applications, tracking of Web activity, and performance engineering for the Internet.

3. ***Increase use of smartphones*** Use of smartphones in organizations is exploding. Most DBMS vendors (e.g., Oracle, IBM, and Sybase) offer small-footprint versions of their products to run on these smartphones, typically in support of specific applications. (This is an example of the personal databases described in Chapter 1.) A small amount of critical data is typically stored on a smartphone, which then is periodically synchronized with data stored on the enterprise data servers. In such an environment, DBAs will often be asked questions about how to design these personal databases (or how to rescue users when they get in trouble). A greater issue is how to manage data synchronization from hundreds (or possibly thousands) of such smartphones while maintaining the data integrity and data availability requirements of the enterprise. However, a number of applications are now available on smartphones that enable DBAs to remotely monitor databases and solve minor issues without requiring physical possession of the devices.

Data Warehouse Administration

The significant growth in data warehousing (see Chapter 9) in the past five years has caused a new role to emerge: that of a data warehouse administrator (DWA). Two generalizations are true about the DWA role:

1. A DWA plays many of the same roles as do DAs and DBAs for the data warehouse and data mart databases for the purpose of supporting decision-making applications (rather than transaction-processing applications for the typical DA and DBA).
2. The role of a DWA emphasizes integration and coordination of metadata and data (extraction agreements, operational data stores, and enterprise data warehouses) across many data sources, not necessarily the standardization of data across these separately managed data sources outside the control and scope of the DWA. Specifically, Inmon (1999) suggests that a DWA has a unique charter to perform the following functions:
   - Build and administer an environment supportive of decision support applications. Thus, a DWA is more concerned with the time to make a decision than with query response time.
   - Build a stable architecture for the data warehouse. A DWA is more concerned with the effect of data warehouse growth (scalability in the amount of data and number of users) than with redesigning existing applications. Inmon refers to this architecture as the *corporate information factory*. For a detailed discussion of this architecture, see Chapter 9 and Inmon, Imhoff, and Sousa (2001).
   - Develop service-level agreements with suppliers and consumers of data for the data warehouse. Thus, a DWA works more closely with end users and operational system administrators to coordinate vastly different objectives and to oversee the development of new applications (data marts, ETL procedures, and analytical services) than do DAs and DBAs.
3. These responsibilities are in addition to the responsibilities typical of any DA or DBA, such as selecting technologies, communicating with users about data needs, making performance and capacity decisions, and budgeting and planning data warehouse requirements.

Inmon (1999)has estimated that every 100 gigabytes of data in an EDW necessitates another DWA. Another metric is that a DWA is needed for each year of data kept in the EDW. The use of custom-built tools for ETL usually increases the number of DWAs needed.

DWAs typically report through the IT unit of an organization but have strong relationships with marketing and other business areas that depend on the EDW for applications, such as customer or supplier relationship management, sales analysis, channel management, and other analytical applications. DWAs should not be part of traditional systems development organizations, as are many DBAs, because data warehousing applications are developed differently than operational systems are and need to be

viewed as independent from any particular operational system. Alternatively, DWAs can be placed in the primary end-user organization for the EDW, but this runs the risk of creating many data warehouses or marts, rather than leading to a true, scalable EDW.

Summary of Evolving Data Administration Roles

The DA and DBA roles are some of the most challenging roles in any organization. The DA has renewed visibility with the recent enactment of financial control regulations and greater interest in data quality. The DBA is always expected to keep abreast of rapidly changing new technologies and is usually involved with mission-critical applications. A DBA must be constantly available to deal with problems, so the DBA is constantly on call. In return, the DBA position ranks among the best compensated in the IS profession.

Many organizations have blended together the data administration and database administration roles. These organizations emphasize the capability to build a database quickly, tune it for maximum performance, and restore it to production quickly when problems develop. These databases are more likely to be departmental, client/server databases that are developed quickly using newer development approaches, such as prototyping, which allow changes to be made more quickly. The blending of data administration and database administration roles also means that DBAs in such organizations must be able to create and enforce data standards and policies.

It is expected that the DBA role will continue to evolve toward increased specialization, with skills such as distributed database/network capacity, server programming, customization of off-the-shelf packages, and support for data warehousing DBAs (Dowgiallo et al., 1997) becoming more important. The ability to work with multiple databases, communication protocols, and operating systems will continue to be highly valued. DBAs who gain broad experience and develop the ability to adapt quickly to changing environments will have many opportunities. It is possible that some current DBA activities, such as tuning, will be replaced by decision support systems able to tune systems by analyzing usage patterns. Some operational duties, such as backup and recovery, can be outsourced and offshored with remote database administration services. Opportunities in large companies to continue working with very large databases (VLDBs) and opportunities in small and midsize companies to manage desktop and midrange servers should remain strong.

THE OPEN SOURCE MOVEMENT AND DATABASE MANAGEMENT

As mentioned previously, one role of a DBA is to select the DBMS(s) to be used in the organization. Database administrators and systems developers in all types of organizations have new alternatives when selecting a DBMS. Increasingly, organizations of all sizes are seriously considering open source DBMSs, such as MySQL and PostgreSQL, as viable choices along with Oracle, DB2, Microsoft SQL Server, Informix, and Teradata. This interest is spurred by the success of the Linux operating system and the Apache Web server. The open source movement began in roughly 1984, with the start of the Free Software Foundation. Today, the Open Source Initiative (**www.opensource.org**) is a nonprofit organization dedicated to managing and promoting the open source movement.

Why has open source software become so popular? It's not all about cost. Advantages of open source software include the following:

- A large pool of volunteer testers and developers facilitate the construction of reliable, low-cost software in a relatively short amount of time. (But be aware that only the most widely used open source software comes close to achieving this advantage; for example, MySQL has over 11 million installations.)
- The availability of the source code allows people to make modifications to add new features, which are easily inspected by others. (In fact, the agreement is that you do share all modifications for the good of the community.)
- Because the software is not proprietary to one vendor, you do not become locked into the product development plans (i.e., new features, time lines) of a single vendor, which might not be adding the features you need for your environment.

- Open source software often comes in multiple versions, and you can select the version that is right for you (from simple to complex, from totally free to some costs for special features).
- Distributing application code dependent on and working with the open source software does not incur any additional costs for copies or licenses. (Deploying software across multiple servers even within the same organization has no marginal cost for the DBMS.)

There are, however, some risks or disadvantages of open source software:

- Often there is not complete documentation (although for-fee services might provide quite sufficient documentation).
- Systems with specialized or proprietary needs across organizations do not have the commodity nature that makes open source software viable, so not all kinds of software lend themselves to being provided via an open source arrangement. (However, DBMSs are viable.)
- There are different types of open source licenses, and not all open source software is available under the same terms; thus, you have to know the ins and outs of each type of license (see Michaelson, 2004).
- An open source tool may not have all the features needed. For example, early versions of MySQL did not support subqueries (although it has now supported subqueries for several releases). An open source tool may not have options for certain functionality, so it may require that "one size fits all."
- Open source software vendors often do not have certification programs. This may not be a major factor for you, but some organizations (often software development contractors) want staff to be certified as a way to demonstrate competence in competitive bidding.

Open source DBMS
Free DBMS source code software that provides the core functionality of an SQL-compliant DBMS.

An **open source DBMS** is free or nearly free database software whose source code is publicly available. (Some people refer to open source as "sharing with rules.") The free DBMS is sufficient to run a database, but vendors provide additional fee-based components and support services that make the product more full featured and comparable to the more traditional product leaders. Because many vendors often provide the additional fee-based components, use of an open source DBMS means that an organization is not tied to one vendor's proprietary product.

A core open source DBMS is not competitive with IBM's DB2, Oracle, or Teradata, but it is more than competitive against Microsoft Access and other PC-oriented packages. As of this chapter's writing, the commercial version of MySQL is priced at $495 for one license, compared to $5,000 to $40,000 for Oracle, DB2, or Microsoft SQL Server, depending on the edition chosen. According to Hall (2003), a typical Oracle database annual license is $300,000, and a comparable MySQL annual subscription for bug fixes and code updates would be $4,000.

Open source DBMSs are improving rapidly to include more powerful features, such as the transaction controls described later in this chapter, needed for mission-critical applications. Open source DBMSs are fully SQL compliant and run on most popular operating systems. For organizations that cannot afford to spend a lot on software or staff (e.g., small businesses, nonprofits, and educational institutions), an open source DBMS can be an ideal choice. For example, many Web sites are supported by MySQL or PostgreSQL database back ends. Visit **www.postgresql.org** and **www.mysql.com** for **more** details on these two leading open source DBMSs.

When choosing an open source (or really any) DBMS, you need to consider the following types of factors:

- ***Features*** Does the DBMS include capabilities you need, such as subqueries, stored procedures, views, and transaction integrity controls?
- ***Support*** How widely is the DBMS used, and what alternatives exist for helping you solve problems? Does the DBMS come with documentation and ancillary tools?
- ***Ease of use*** This often depends on the availability of tools that make any piece of system software, such as a DBMS, easier to use through things like a GUI interface.

- ***Stability*** How frequently and how seriously does the DBMS malfunction over time or with high-volume use?
- ***Speed*** How rapid is the response time to queries and transactions with proper tuning of the database? (Because open source DBMSs are often not as fully loaded with advanced, obscure features, their performance can be attractive.)
- ***Training*** How easy is it for developers and users to learn to use the DBMS?
- ***Licensing*** What are the terms of the open source license, and are there commercial licenses that would provide the types of support needed?

MANAGING DATA SECURITY

Consider the following situations:

- At the university of one of this book's authors, anyone with access to the university's main automated system for student and faculty data can see everyone's Social Security number.
- A previously loyal employee is given access to sensitive documents, and within a few weeks leaves the organization, purportedly with a trove of trade secrets to share with competing firms.
- The FBI reports (Morrow, 2007) that there are 3,000 clandestine organizations in the United States whose sole purpose is to steal secrets and acquire technology for foreign organizations.
- Sarbanes-Oxley requires that companies audit the access of privileged users to sensitive data, and the payment card industry standards require companies to track user identity information whenever credit card data are used.

Database security
Protection of database data against accidental or intentional loss, destruction, or misuse.

The goal of **database security** is to protect data from accidental or intentional threats to their integrity and access. The database environment has grown more complex, with distributed databases located on client/server architectures and personal computers as well as on mainframes. Access to data has become more open through the Internet and corporate intranets and from mobile computing devices. As a result, managing data security effectively has become more difficult and time-consuming. Some security procedures for client/server and Web-based systems were introduced in Chapter 8.

Because data are a critical resource, all persons in an organization must be sensitive to security threats and take measures to protect the data within their domains. For example, computer listings or computer disks containing sensitive data should not be left unattended on desktops. Data administration is often responsible for developing overall policies and procedures to protect databases. Database administration is typically responsible for administering database security on a daily basis. The facilities that database administrators have to use in establishing adequate data security are discussed later, but first it is important to review potential threats to data security.

Threats to Data Security

Threats to data security may be direct threats to the database. For example, those who gain unauthorized access to a database may then browse, change, or even steal the data to which they have gained access. (See the news story at the beginning of this chapter for a good example.) Focusing on database security alone, however, will not ensure a secure database. All parts of the system must be secure, including the database, the network, the operating system, the building(s) in which the database resides physically, and the personnel who have any opportunity to access the system. Figure 11-2 diagrams many of the possible locations for data security threats. Accomplishing this level of security requires careful review, establishment of security procedures and policies, and implementation and enforcement of those procedures and policies. The following threats must be addressed in a comprehensive data security plan:

- ***Accidental losses, including human error, software, and hardware-caused breaches*** Establishing operating procedures such as user authorization, uniform software installation procedures, and hardware maintenance schedules are examples of actions that may be taken to address threats from accidental losses. As in any effort

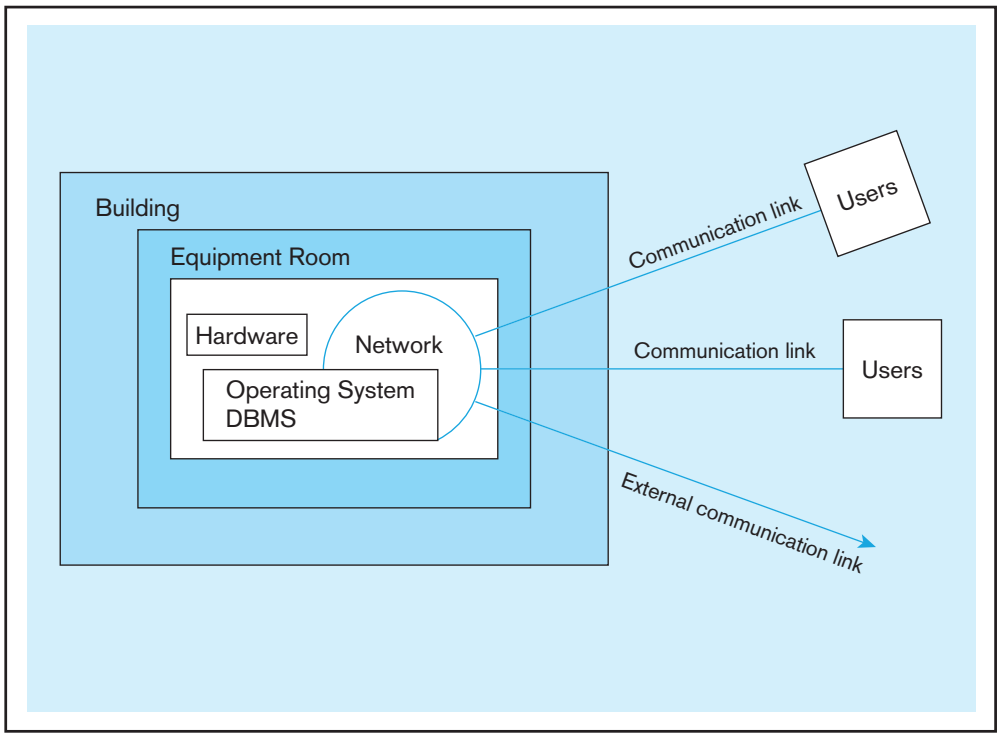


FIGURE 11-2 **Possible locations of data security threats**

that involves human beings, some losses are inevitable, but well-thought-out policies and procedures should reduce the amount and severity of losses. Of potentially more serious consequence are the threats that are not accidental.

- ***Theft and fraud*** These activities are going to be perpetrated by people, quite possibly through electronic means, and may or may not alter data. Attention here should focus on each possible location shown in Figure 11-2. For example, physical security must be established so that unauthorized persons are unable to gain access to rooms where computers, servers, telecommunications facilities, or computer files are located. Physical security should also be provided for employee offices and any other locations where sensitive data are stored or easily accessed. Establishment of a firewall to protect unauthorized access to inappropriate parts of the database through outside communication links is another example of a security procedure that will hamper people who are intent on theft or fraud.
- ***Loss of privacy or confidentiality*** Loss of privacy is usually taken to mean loss of protection of data about individuals, whereas loss of confidentiality is usually taken to mean loss of protection of critical organizational data that may have strategic value to the organization. Failure to control privacy of information may lead to blackmail, bribery, public embarrassment, or stealing of user passwords. Failure to control confidentiality may lead to loss of competitiveness. State and federal laws now exist to require some types of organizations to create and communicate policies to ensure privacy of customer and client data. Security mechanisms must enforce these policies, and failure to do so can mean significant financial and reputation loss.
- ***Loss of data integrity*** When data integrity is compromised, data will be invalid or corrupted. Unless data integrity can be restored through established backup and recovery procedures, an organization may suffer serious losses or make incorrect and expensive decisions based on the invalid data.
- ***Loss of availability*** Sabotage of hardware, networks, or applications may cause the data to become unavailable to users, which again may lead to severe operational difficulties. This category of threat includes the introduction of viruses intended to corrupt data or software or to render the system unusable. It is important to counter this threat by always installing the most current antivirus software, as well as educating employees on the sources of viruses. We discuss data availability later in this chapter.

As noted earlier, data security must be provided within the context of a total program for security. Two critical areas that strongly support data security are client/server security and Web application security. We address these two topics next, before outlining approaches aimed more directly at data security.

Establishing Client/Server Security

Database security is only as good as the security of the whole computing environment. Physical security, logical security, and change control security must be established across all components of the client/server environment, including the servers, the client workstations, the network and its related components, and the users.

SERVER SECURITY In a modern client/server environment, multiple servers, including database servers, need to be protected. Each should be located in a secure area, accessible only to authorized administrators and supervisors. Logical access controls, including server and administrator passwords, provide layers of protection against intrusion.

Most modern DBMSs have database-level password security that is similar to system-level password security. Database management systems, such as Oracle and SQL Server, provide database administrators with considerable capabilities that can provide aid in establishing data security, including the capability to limit each user's access and activity permissions (e.g., *select*, *update*, *insert*, or *delete*) to tables within the database. While, it is also possible to pass authentication information through from the operating system's authentication capability, this reduces the number of password security layers. Thus, in a database server, sole reliance on operating system authentication should not be encouraged.

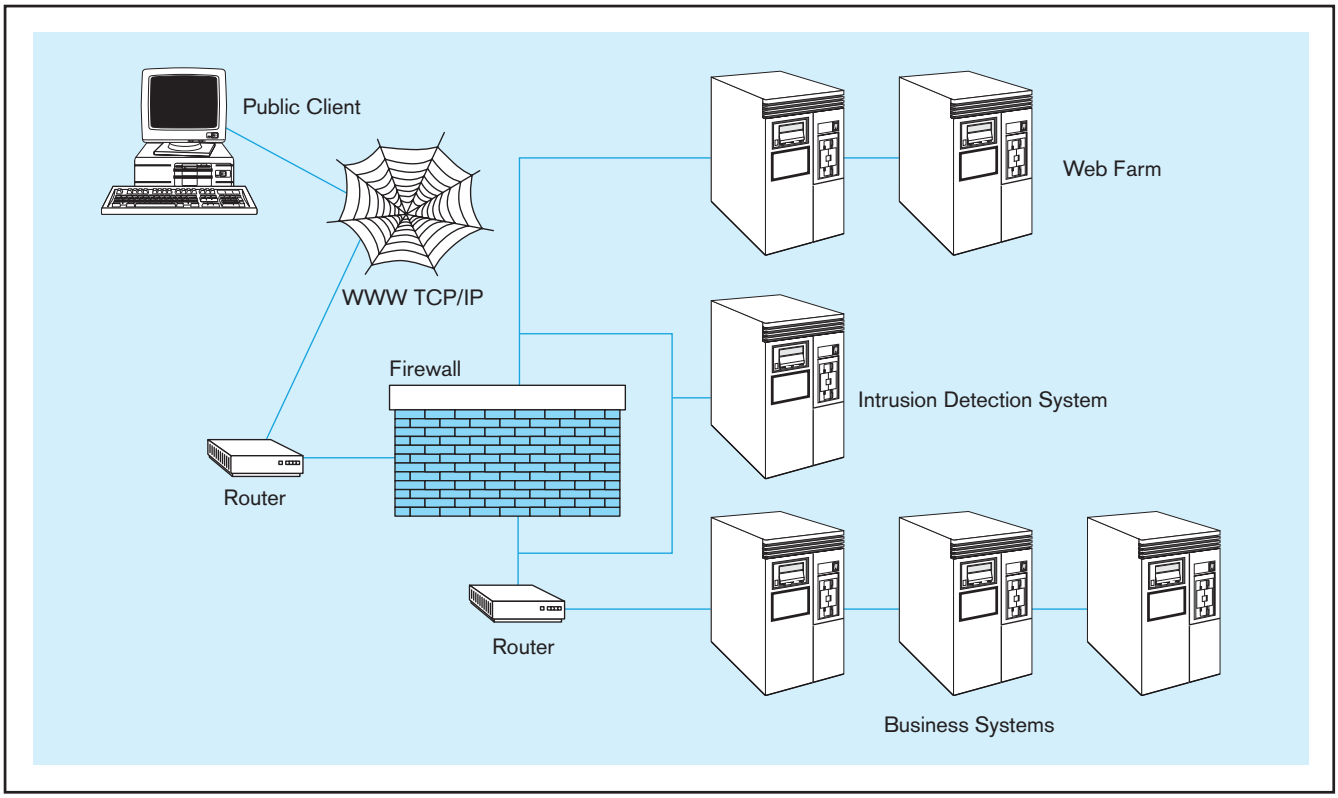
NETWORK SECURITY Securing client/server systems includes securing the network between client and server. Networks are susceptible to breaches of security through eavesdropping, unauthorized connections, or unauthorized retrieval of packets of information that are traversing the network. Thus, encryption of data so that attackers cannot read a data packet that is being transmitted is obviously an important part of network security. (We discuss encryption later in the chapter.) In addition, authentication of the client workstation that is attempting to access the server also helps to enforce network security, and application authentication gives the user confidence that the server being contacted is the real server needed by the user. Audit trails of attempted accesses can help administrators identify unauthorized attempts to use the system. Other system components, such as routers, can also be configured to restrict access to authorized users, IP addresses, and so forth.

Application Security Issues in Three-Tier Client/Server Environments

The explosion of Web sites that make data accessible to viewers through their Internet connections raises new issues that go beyond the general client/server security issues just addressed. In a three-tier environment, the dynamic creation of a Web page from a database requires access to the database, and if the database is not properly protected, it is vulnerable to inappropriate access by any user. This is a new point of vulnerability that was previously avoided by specialized client access software. Also of interest is privacy. Companies are able to collect information about those who access their Web sites. If they are conducting e-commerce activities, selling products over the Web, they can collect information about their customers that has value to other businesses. If a company sells customer information without those customers' knowledge or if a customer believes that may happen, ethical and privacy issues are raised that must be addressed.

Figure 11-3 illustrates a typical environment for Web-enabled databases. The Web farm includes Web servers and database servers supporting Web-based applications. If an organization wishes to make only static HTML pages available, protection must be established for the HTML files stored on a Web server. Creation of a static Web page with extracts from a database uses traditional application development languages such as Visual Basic.NET or Java, and thus their creation can be controlled by using

FIGURE 11-3 Establishing Internet security



standard methods of database access control. If some of the HTML files loaded on the Web server are sensitive, they can be placed in directories that are protected using operating system security or they may be readable but not published in the directory. Thus, the user must know the exact file name to access the sensitive HTML page. It is also common to segregate the Web server and limit its contents to publicly browsable Web pages. Sensitive files may be kept on another server accessible through an organization's intranet.

Security measures for dynamic Web page generation are different. Dynamic Web pages are stored as a template into which the appropriate and current data are inserted from the database or user input once any queries associated with the page are run. This means that the Web server must be able to access the database. To function appropriately, the connection usually requires full access to the database. Thus, establishing adequate server security is critical to protecting the data. The server that owns the database connection should be physically secure, and the execution of programs on the server should be controlled. User input, which could embed SQL commands, also needs to be filtered so unauthorized scripts are not executed.

Access to data can also be controlled through another layer of security: user-authentication security. Use of an HTML login form will allow the database administrator to define each user's privileges. Each session may be tracked by storing a piece of data, or cookie, on the client machine. This information can be returned to the server and provide information about the login session. Session security must also be established to ensure that private data are not compromised during a session, because information is broadcast across a network for reception by a particular machine and is thus susceptible to being intercepted. TCP/IP is not a very secure protocol, and encryption systems, such as the ones discussed later in this chapter, are essential. A standard encryption method, Secure Sockets Layer (SSL), is used by many developers to encrypt all data traveling between client and server during a session. URLs that begin with https:// use SSL for transmission.

Additional methods of Web security include ways to restrict access to Web servers:

- Restrict the number of users on the Web server as much as possible. Of those users, give as few as possible superuser or administrator rights. Only those given these privileges should also be allowed to load software or edit or add files.
- Restrict access to the Web server, keeping a minimum number of ports open. Try to open a minimum number of ports, and preferably only http and https ports.
- Remove any unneeded programs that load automatically when setting up the server. Demo programs are sometimes included that can provide a hacker with the access desired. Compilers and interpreters such as Perl should not be on a path that is directly accessible from the Internet.

DATA PRIVACY Protection of individual privacy when using the Internet has become an important issue. E-mail, e-commerce and marketing, and other online resources have created new computer-mediated communication paths. Many groups have an interest in people's Internet behavior, including employers, governments, and businesses. Applications that return individualized responses require that information be collected about the individual, but at the same time proper respect for the privacy and dignity of employees, citizens, and customers should be observed.

Concerns about the rights of individuals to not have personal information collected and disseminated casually or recklessly have intensified as more of the population has become familiar with computers and as communications among computers have proliferated. Information privacy legislation generally gives individuals the right to know what data have been collected about them and to correct any errors in those data. As the amount of data exchanged continues to grow, the need is also growing to develop adequate data protection. Also important are adequate provisions to allow the data to be used for legitimate legal purposes so that organizations that need the data can access them and rely on their quality. Individuals need to be given the opportunity to state with whom data retained about them may be shared, and then these wishes must be enforced; enforcement is more reliable if access rules based on privacy wishes are developed by the DBA staff and handled by the DBMS.

Individuals must guard their privacy rights and must be aware of the privacy implications of the tools they are using. For example, when using a browser, users may elect to allow cookies to be placed on their machines, or they may reject that option. To make a decision with which they would be comfortable, they must know several things. They must be aware of cookies, understand what they are, evaluate their own desire to receive customized information versus their wish to keep their browsing behavior to themselves, and learn how to set their machine to accept or reject cookies. Browsers and Web sites have not been quick to help users understand all of these aspects. Abuses of privacy, such as selling customer information collected in cookies, has helped increase general awareness of the privacy issues that have developed as use of the Web for communication, shopping, and other uses has developed.

At work, the individual needs to realize that communication executed through their employer's machines and networks is not private. Courts have upheld the rights of employers to monitor all employee electronic communication.

On the Internet, privacy of communication is not guaranteed. Encryption products, anonymous remailers, and built-in security mechanisms in commonly used software help to preserve privacy. Protecting the privately owned and operated computer networks that now make up a very critical part of our information infrastructure is essential to the further development of electronic commerce, banking, health care, and transportation applications over the Web.

The W3C has created a standard, the Platform for Privacy Preferences (P3P), that will communicate a Web site's stated privacy policies and compare that statement with the user's own policy preferences. P3P uses XML code on Web site servers that can be fetched automatically by any browser or plug-in equipped for P3P. The client browser

or plug-in can then compare the site's privacy policy with the user's privacy preferences and inform the user of any discrepancies. P3P addresses the following aspects of online privacy:

- Who is collecting the data?
- What information is being collected, and for what purpose?
- What information will be shared with others, and who are those others?
- Can users make changes in the way their data will be used by the collector?
- How are disputes resolved?
- What policies are followed for retaining data?
- Where can the site's detailed policies be found, in readable form?

Anonymity is another important facet of Internet communication that has come under pressure. Although U.S. law protects a right to anonymity, chat rooms and e-mail forums have been required to reveal the names of people who have posted messages anonymously. A 1995 European Parliament directive that would cut off data exchanges with any country lacking adequate privacy safeguards has led to an agreement that the United States will provide the same protection to European customers as European businesses do. This may lead Congress to establish legislation that is more protective than that previously enacted.

DATABASE SOFTWARE DATA SECURITY FEATURES

A comprehensive data security plan will include establishing administrative policies and procedures, physical protections, and data management software protections. Physical protections, such as securing data centers and work areas, disposing of obsolete media, and protecting portable devices from theft, are not covered here. We discuss administrative policies and procedures later in this section. All the elements of a data security plan work together to achieve the desired level of security. Some industries, for example health care, have regulations that set standards for the security plan and, hence, put requirements on data security. (See Anderson, 2005, for a discussion of the HIPAA security guidelines.) The most important security features of data management software follow:

1. Views or subschemas, which restrict user views of the database
2. Domains, assertions, checks, and other integrity controls defined as database objects, which are enforced by the DBMS during database querying and updating
3. Authorization rules, which identify users and restrict the actions they may take against a database
4. User-defined procedures, which define additional constraints or limitations in using a database
5. Encryption procedures, which encode data in an unrecognizable form
6. Authentication schemes, which positively identify persons attempting to gain access to a database
7. Backup, journaling, and checkpointing capabilities, which facilitate recovery procedures

Views

In Chapter 6, we defined a view as a subset of a database that is presented to one or more users. A view is created by querying one or more of the base tables, producing a dynamic result table for the user at the time of the request. Thus, a view is always based on the current data in the base tables from which it is built. The advantage of a view is that it can be built to present only the data (certain columns and/or rows) to which the user requires access, effectively preventing the user from viewing other data that may be private or confidential. The user may be granted the right to access the view, but not to access the base tables upon which the view is based. So, confining a user to a view may be more restrictive for that user than allowing him or her access to the involved base tables.

For example, we could build a view for a Pine Valley employee that provides information about materials needed to build a Pine Valley furniture product without

providing other information, such as unit price, that is not relevant to the employee's work. This command creates a view that will list the wood required and the wood available for each product:


```
CREATE VIEW MATERIALS_V AS
  SELECT Product_T.ProductID, ProductName, Footage,
      FootageOnHand
  FROM Product_T, RawMaterial_T, Uses_T
  WHERE Product_T.ProductID = Uses_T.ProductID
  AND RawMaterial_T.MaterialID = Uses_T.MaterialID;
```

The contents of the view created will be updated each time the view is accessed, but here are the current contents of the view, which can be accessed with the SQL command:

```
SELECT * FROM MATERIALS_V;
```

ProductID	ProductName	Footage	FootageOnHand
1	End Table	4	1
2	Coffee Table	6	11
3	Computer Desk	15	11
4	Entertainment Center	20	84
5	Writer's Desk	13	68
6	8-Drawer Desk	16	66
7	Dining Table	16	11
8	Computer Desk	15	9

8 rows selected.

The user can write SELECT statements against the view, treating it as though it were a table. Although views promote security by restricting user access to data, they are not adequate security measures because unauthorized persons may gain knowledge of or access to a particular view. Also, several persons may share a particular view; all may have authority to read the data, but only a restricted few may be authorized to update the data. Finally, with high-level query languages, an unauthorized person may gain access to data through simple experimentation. As a result, more sophisticated security measures are normally required.

Integrity Controls

Integrity controls protect data from unauthorized use and update. Often, integrity controls limit the values a field may hold and the actions that can be performed on data, or trigger the execution of some procedure, such as placing an entry in a log to record which users have done what with which data.

One form of integrity control is a domain. In essence, a domain can be used to create a user-defined data type. Once a domain is defined, any field can be assigned that domain as its data type. For example, the following PriceChange domain (defined in SQL) can be used as the data type of any database field, such as PriceIncrease and PriceDiscount, to limit the amount standard prices can be augmented in one transaction:

```
CREATE DOMAIN PriceChange AS DECIMAL
  CHECK (VALUE BETWEEN .001 and .15);
```

Then, in the definition of, say, a pricing transaction table, we might have the following:

```
PriceIncrease PriceChange NOT NULL,
```

One advantage of a domain is that, if it ever has to change, it can be changed in one place—the domain definition—and all fields with this domain will be changed automatically. Alternatively, the same CHECK clause could be included in a constraint on both the PriceIncrease and PriceDiscount fields, but in this case, if the limits of the check were to change, a DBA would have to find every instance of this integrity control and change it in each place separately.

Assertions are powerful constraints that enforce certain desirable database conditions. Assertions are checked automatically by the DBMS when transactions are run involving tables or fields on which assertions exist. For example, assume that an employee table has the fields EmpID, EmpName, SupervisorID, and SpouseID. Suppose that a company rule is that no employee may supervise his or her spouse. The following assertion enforces this rule:

```
CREATE ASSERTION SpousalSupervision
  CHECK (SupervisorID < > SpouseID);
```

If the assertion fails, the DBMS will generate an error message.

Assertions can become rather complex. Suppose that Pine Valley Furniture has a rule that no two salespersons can be assigned to the same territory at the same time. Suppose a Salesperson table includes the fields SalespersonID and TerritoryID. This assertion can be written using a correlated subquery, as follows:

```
CREATE ASSERTION TerritoryAssignment
  CHECK (NOT EXISTS
    (SELECT * FROM Salesperson_T SP WHERE SP.TerritoryID IN
      (SELECT SSP.TerritoryID FROM Salesperson_T SSP WHERE
        SSP.SalespersonID < > SP.SalespersonID)));
```

Finally, *triggers* (defined and illustrated in Chapter 7) can be used for security purposes. A trigger, which includes an event, a condition, and an action, is potentially more complex than an assertion. For example, a trigger can do the following:

- Prohibit inappropriate actions (e.g., changing a salary value outside the normal business day)
- Cause special handling procedures to be executed (e.g., if a customer invoice payment is received after some due date, a penalty can be added to the account balance for that customer)
- Cause a row to be written to a log file to echo important information about the user and a transaction being made to sensitive data, so that the log can be reviewed by human or automated procedures for possible inappropriate behavior (e.g., the log can record which user initiated a salary change for which employee)

As with domains, a powerful benefit of a trigger, as with any other stored procedure, is that the DBMS enforces these controls for all users and all database activities. The control does not have to be coded into each query or program. Thus, individual users and programs cannot circumvent the necessary controls.

Assertions, triggers, stored procedures, and other forms of integrity controls may not stop all malicious or accidental use or modification of data. Thus, it is recommended (Anderson, 2005) that a change audit process be used in which all user activities are logged and monitored to check that all policies and constraints are enforced. Following this recommendation means that every database query and transaction is logged to record characteristics of all data use, especially modifications: who accessed the data, when it was accessed, what program or query was run, where in the computer network the request was generated, and other parameters that can be used to investigate suspicious activity or actual breaches of security and integrity.

Authorization Rules

Authorization rules are controls incorporated in a data management system that restrict access to data and also restrict the actions that people may take when they access data. For example, a person who can supply a particular password may be authorized to read any record in a database but cannot necessarily modify any of those records.

Authorization rules
Controls incorporated in a data management systems that restrict access to data and also restrict the actions that people may take when they access data.

Fernandez et al. (1981) have developed a conceptual model of database security. Their model expresses authorization rules in the form of a table (or matrix) that includes subjects, objects, actions, and constraints. Each row of the table indicates that a particular subject is authorized to take a certain action on an object in the database, perhaps subject to some constraint. Figure 11-4 shows an example of such an authorization matrix. This table contains several entries pertaining to records in an accounting database. For example, the first row in the table indicates that anyone in the Sales Department is authorized to insert a new customer record in the database, provided that the customer's credit limit does not exceed $5,000. The last row indicates that the program AR4 is authorized to modify order records without restriction. Data administration is responsible for determining and implementing authorization rules that are implemented at the database level. Authorization schemes can also be implemented at the operating system level or the application level.

Most contemporary database management systems do not implement an authorization matrix such as the one shown in Figure 11-4; they normally use simplified versions. There are two principal types: authorization tables for subjects and authorization tables for objects. Figure 11-5 shows an example of each type. In Figure 11-5a, for example, we see that salespersons are allowed to modify customer records but not delete these records. In Figure 11-5b, we see that users in Order Entry or Accounting can modify order records, but salespersons cannot. A given DBMS product may provide either one or both of these types of facilities.

Authorization tables, such as those shown in Figure 11-5, are attributes of an organization's data and their environment; they are therefore properly viewed as metadata. Thus, the tables should be stored and maintained in the repository. Because authorization tables contain highly sensitive data, they themselves should be protected by stringent security rules. Normally, only selected persons in data administration have authority to access and modify these tables.

For example, in Oracle, the privileges included in Figure 11-6 can be granted to users at the database level or table level. INSERT and UPDATE can be granted at the column level. Where many users, such as those in a particular job classification, need similar privileges, roles may be created that contain a set of privileges, and then all the privileges can be granted to a user simply by granting the role. To grant the ability to read the product table and update prices to a user with the log in ID of SMITH, the following SQL command may be given:

```
GRANT SELECT, UPDATE (UnitPrice) ON Product_T TO SMITH;
```

There are eight data dictionary views that contain information about privileges that have been granted. In this case, DBA_TAB_PRIVS contains users and objects for every user who has been granted privileges on objects, such as tables. DBA_COL_PRIVS contains users who have been granted privileges on columns of tables.

Subject	Object	Action	Constraint
Sales Dept.	Customer record	Insert	Credit limit LE $5000
Order trans.	Customer record	Read	None
Terminal 12	Customer record	Modify	Balance due only
Acctg. Dept.	Order record	Delete	None
Ann Walker	Order record	Insert	Order aml LT $2000
Program AR4	Order record	Modify	None

FIGURE 11-4 Authorization matrix

FIGURE 11-5 Implementing authorization rules
(a) Authorization table for subjects (salespersons)

	Customer records	Order records
Read	Y	Y
Insert	Y	Y
Modify	Y	N
Delete	N	N

(b) Authorization table for objects (order records)

	Salespersons (password BATMAN)	Order entry (password JOKER)	Accounting (password TRACY)
Read	Y	Y	Y
Insert	N	Y	N
Modify	N	Y	N
Delete	N	N	Y

FIGURE 11-6 Oracle privileges

Privilege	Capability
SELECT	Query the object.
INSERT	Insert records into the table/view. Can be given for specific columns.
UPDATE	Update records in table/view. Can be given for specific columns.
DELETE	Delete records from table/view.
ALTER	Alter the table.
INDEX	Create indexes on the table.
REFERENCES	Create foreign keys that reference the table.
EXECUTE	Execute the procedure, package, or function.

User-Defined Procedures

User-defined procedures
User exits (or interfaces) that allow system designers to define their own security procedures in addition to the authorization rules.

Some DBMS products provide user exits (or interfaces) that allow system designers or users to create their own **user-defined procedures** for security, in addition to the authorization rules we have just described. For example, a user procedure might be designed to provide positive user identification. In attempting to log on to the computer, the user might be required to supply a procedure name in addition to a simple password. If valid password and procedure names are supplied, the system then calls the procedure, which asks the user a series of questions whose answers should be known only to that password holder (e.g., mother's maiden name).

Encryption

Encryption
The coding or scrambling of data so that humans cannot read them.

Data encryption can be used to protect highly sensitive data such as customer credit card numbers or account balances. **Encryption** is the coding or scrambling of data so that humans cannot read them. Some DBMS products include encryption routines that automatically encode sensitive data when they are stored or transmitted over communications channels. For example, encryption is commonly used in electronic funds transfer (EFT) systems. Other DBMS products provide exits that allow users to code their own encryption routines.

Any system that provides encryption facilities must also provide complementary routines for decoding the data. These decoding routines must be protected by adequate security, or else the advantages of encryption are lost. They also require significant computing resources.

Two common forms of encryption exist: one key and two key. With a one-key method, also called Data Encryption Standard (DES), both the sender and the receiver need to know the key that is used to scramble the transmitted or stored data. A two-key method, also called asymmetric encryption, employs a private and a public key. Two-key methods (see Figure 11-7) are especially popular in e-commerce applications to provide secure transmission and database storage of payment data, such as credit card numbers.

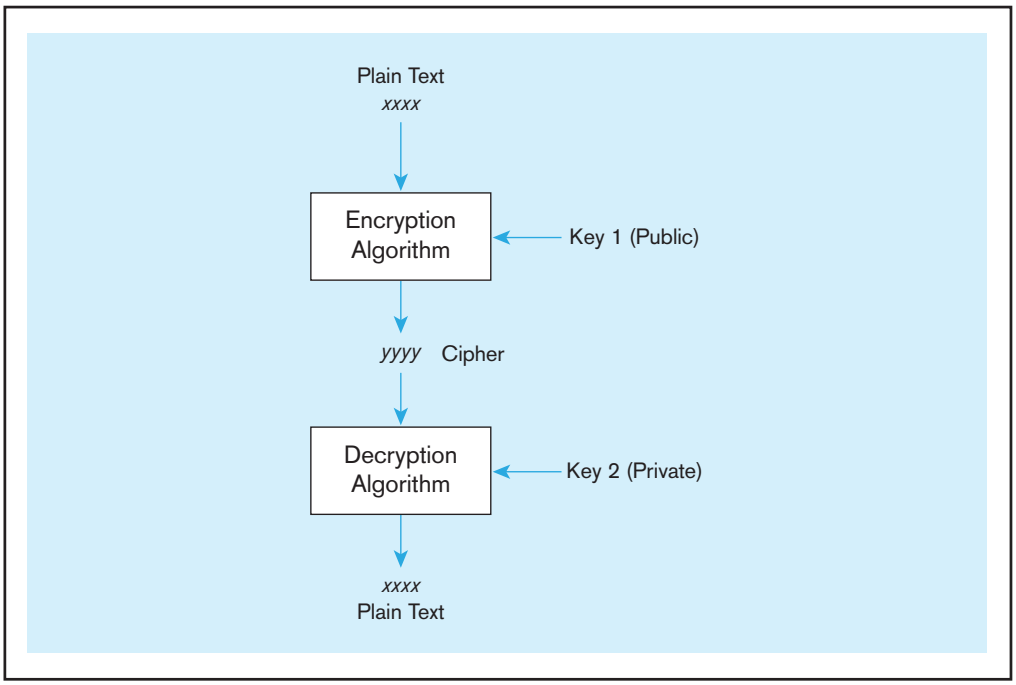


FIGURE 11-7 Basic two-key encryption

A popular implementation of the two-key method is Secure Sockets Layer (SSL), developed by Netscape Communications Corporation. SSL is built into most major browsers and Web servers. It provides data encryption, server authentication, and other services in a TCP/IP connection. For example, the U.S. banking industry uses a 128-bit version of SSL (the most secure level in current use) to secure online banking transactions.

Details about encryption techniques are beyond the scope of this book and are generally handled by the DBMS without significant involvement of a DBA; it is simply important to know that database data encryption is a strong measure available to a DBA.

Authentication Schemes

A long-standing problem in computer circles is how to identify persons who are trying to gain access to a computer or its resources, such as a database or DBMS. In an electronic environment, a user can prove his or her identity by supplying one or more of the following factors:

1. Something the user knows, usually a password or personal identification number (PIN)
2. Something the user possesses, such as a smart card or token
3. Some unique personal characteristic, such as a fingerprint or retinal scan

Authentication schemes are called one-factor, two-factor, or three-factor authentication, depending on how many of these factors are employed. Authentication becomes stronger as more factors are used.

PASSWORDS The first line of defense is the use of passwords, which is a one-factor authentication scheme. With such a scheme, anyone who can supply a valid password can log on to a database system. (A user ID may also be required, but user IDs are typically not secured.) A DBA (or perhaps a system administrator) is responsible for managing schemes for issuing or creating passwords for the DBMS and/or specific applications.

Although requiring passwords is a good starting point for authentication, it is well known that they this method has a number of deficiencies. People assigned passwords for different devices quickly devise ways to remember these passwords, ways that tend to compromise the password scheme. The passwords get written down, where others may find them. They get shared with other users; it is not unusual for an entire department to use one common password for access. Passwords get included in

automatic logon scripts, which removes the inconvenience of remembering them and typing them but also eliminates their effectiveness. And passwords usually traverse a network in cleartext, not encrypted, so if intercepted they may be easily interpreted. Also, passwords cannot, by themselves, ensure the security of a computer and its databases because they give no indication of who is trying to gain access. Thus, for example, a log should be kept and analyzed of attempted logons with incorrect passwords.

STRONG AUTHENTICATION More reliable authentication techniques have become a business necessity, with the rapid advances in e-commerce and increased security threats in the form of hacking, identity theft, and so on.

Two-factor authentication schemes require two of the three factors: something the user has (usually a card or token) and something the user knows (usually a PIN). You are already familiar with this system from using automated teller machines (ATMs). This scheme is much more secure than using only passwords because (barring carelessness) it is quite difficult for an unauthorized person to obtain both factors at the same time.

Although an improvement over password-only authentication, two-factor schemes are not infallible. Cards can be lost or stolen, and PINs can be intercepted. *Three-factor authentication* schemes add an important third factor: a biometric attribute that is unique for each individual user. Personal characteristics that are commonly used include fingerprints, voiceprints, eye pictures, and signature dynamics.

Smart card
A credit card–sized plastic card with an embedded microprocessor chip that can store, process, and output electronic data in a secure manner.

Three-factor authentication is normally implemented with a high-tech card called a smart card (or smart badge). A **smart card** is a credit card–sized plastic card with an embedded microprocessor chip that can store, process, and output electronic data in a secure manner. Smart cards are replacing the familiar magnetic-stripe-based cards we have used for decades. Using smart cards can be a very strong means to authenticate a database user. In addition, smart cards can themselves be database storage devices; today smart cards can store well over 100MB bytes of data, and this number is increasing rapidly. Smart cards can provide secure storage of personal data such as medical records or a summary of medications taken.

All of the authentication schemes described here, including use of smart cards, can be only as secure as the process that is used to issue them. For example, if a smart card is issued and personalized to an imposter (either carelessly or deliberately), it can be used freely by that person. Thus, before allowing any form of authentication—such as issuing a new card to an employee or other person—the issuing agency must validate beyond any reasonable doubt the identity of that person. Because paper documents are used in this process—birth certificates, passports, driver's licenses, and so on—and these types of documents are often unreliable because they can be easily copied, forged, and so on, significant training of the personnel, use of sophisticated technology, and sufficient oversight of the process are needed to ensure that this step is rigorous and well controlled.

SARBANES-OXLEY (SOX) AND DATABASES

The Sarbanes-Oxley Act (SOX) and other similar global regulations were designed to ensure the integrity of public companies' financial statements. A key component of this is ensuring sufficient control and security over the financial systems and IT infrastructure in use within an organization. This has resulted in an increased emphasis on understanding controls around information technology. Given that the focus of SOX is on the integrity of financial statements, controls around the databases and applications that are the source of these data is key.

The key focus of SOX audits is around three areas of control:

1. IT change management
2. Logical access to data
3. IT operations

Most audits start with a walkthrough—that is, a meeting with business owners (of the data that fall under the scope of the audit) and technical architects of the applications and databases. During this walkthrough, the auditors will try to understand how the above three areas are handled by the IT organization.

IT Change Management

IT change management refers to the process by which changes to operational systems and databases are authorized. Typically any change to a production system or database has to be approved by a change control board that is made up of representatives from the business and IT organizations. Authorized changes must then be put through a rigorous process (essentially a mini systems development life cycle) before being put into production. From a database perspective, the most common types of changes are changes to the database schema, changes to database configuration parameters, and patches/updates to the DBMS software itself.

A key issue related to change management that was a top deficiency found by SOX auditors was adequate segregation of duties between people who had access to databases in the three common environments: development, test, and production. SOX mandates that the DBAs who have the ability to modify data in these three environments be different. This is primarily to ensure that changes to the operating environment have been adequately tested before being implemented. In cases where the size of the organization does not allow this, other personnel should be authorized to do periodic reviews of database access by DBAs, using features such as database audits (described in the next section).

Logical Access to Data

Logical access to data is essentially about the security procedures in place to prevent unauthorized access to the data. From a SOX perspective, the two key questions to ask are: Who has access to what? and Who has access to too much? In response to these two questions, organizations must establish administrative policies and procedures that serve as a context for effectively implementing these measures. Two types of security policies and procedures are personnel controls and physical access controls.

PERSONNEL CONTROLS Adequate controls of personnel must be developed and followed, for the greatest threat to business security is often internal rather than external. In addition to the security authorization and authentication procedures just discussed, organizations should develop procedures to ensure a selective hiring process that validates potential employees' representations about their backgrounds and capabilities. Monitoring to ensure that personnel are following established practices, taking regular vacations, working with other employees, and so forth should be done. Employees should be trained in those aspects of security and quality that are relevant to their jobs and encouraged to be aware of and follow standard security and data quality measures. Standard job controls, such as separating duties so no one employee has responsibility for an entire business process or keeping application developers from having access to production systems, should also be enforced. Should an employee need to be let go, there should be an orderly and timely set of procedures for removing authorizations and authentications and notifying other employees of the status change. Similarly, if an employee's job profile changes, care should be taken to ensure that his or her new set of roles and responsibilities do not lead to violations of separation of duties.

PHYSICAL ACCESS CONTROLS Limiting access to particular areas within a building is usually a part of controlling physical access. Swipe, or proximity access, cards can be used to gain access to secure areas, and each access can be recorded in a database, with a time stamp. Guests, including vendor maintenance representatives, should be issued badges and escorted into secure areas. Access to sensitive equipment, including hardware and peripherals such as printers (which may be used to print classified reports) can be controlled by placing these items in secure areas. Other equipment may be locked to a desk or cabinet or may have an alarm attached. Backup data tapes should be kept in fireproof data safes and/or kept offsite, at a safe location. Procedures that make explicit the schedules for moving media and disposing of media and that establish labeling and indexing of all materials stored must be established.

Placement of computer screens so that they cannot be seen from outside the building may also be important. Control procedures for areas external to the office building

should also be developed. Companies frequently use security guards to control access to their buildings or use a card swipe system or handprint recognition system (smart badges) to automate employee access to the building. Visitors should be issued an identification card and required to be accompanied throughout the building.

New concerns are raised by the increasingly mobile nature of work. Laptop computers are very susceptible to theft, which puts data on a laptop at risk. Encryption and multiple-factor authentication can protect data in the event of laptop theft. Antitheft devices (e.g., security cables, geographic tracking chips) can deter theft or help quickly recover stolen laptops on which critical data are stored.

IT Operations

IT operations refers to the policies and procedures in place related to the day-to-day management of the infrastructure, applications, and databases in an organization. Key areas in this regard that are relevant to data and database administrators are database backup and recovery, as well as data availability. These are discussed in detail in later sections.

An area of control that helps to maintain data quality and availability but that is often overlooked is vendor management. Organizations should periodically review external maintenance agreements for all hardware and software they are using to ensure that appropriate response rates are agreed to for maintaining system quality and availability. It is also important to consider reaching agreements with the developers of all critical software so that the organization can get access to source code should the developer go out of business or stop supporting the programs. One way to accomplish this is by having a third party hold the source code, with an agreement that it will be released if such a situation develops. Controls should be in place to protect data from inappropriate access and use by outside maintenance staff and other contract workers.

DATABASE BACKUP AND RECOVERY

Database recovery
Mechanisms for restoring a database quickly and accurately after loss or damage.

Database recovery is database administration's response to Murphy's law. Inevitably, databases are damaged or lost or become unavailable because of some system problem that may be caused by human error, hardware failure, incorrect or invalid data, program errors, computer viruses, network failures, conflicting transactions, or natural catastrophes. It is the responsibility of a DBA to ensure that all critical data in a database are protected and can be recovered in the event of loss. Because an organization depends heavily on its databases, a DBA must be able to minimize downtime and other disruptions while a database is being backed up or recovered. To achieve these objectives, a database management system must provide mechanisms for backing up data with as little disruption of production time as possible and restoring a database quickly and accurately after loss or damage.

Basic Recovery Facilities

A database management system should provide four basic facilities for backup and recovery of a database:

1. ***Backup facilities,*** which provide periodic backup (sometimes called *fallback*) copies of portions of or the entire database
2. ***Journalizing facilities,*** which maintain an audit trail of transactions and database changes
3. ***A checkpoint facility,*** by which the DBMS periodically suspends all processing and synchronizes its files and journals to establish a recovery point
4. ***A recovery manager,*** which allows the DBMS to restore the database to a correct condition and restart processing transactions

Backup facility
A DBMS COPY utility that produces a backup copy (or save) of an entire database or a subset of a database.

BACKUP FACILITIES A DBMS should provide **backup facilities** that produce a backup copy (or save) of the entire database plus control files and journals. Each DBMS normally provides a COPY utility for this purpose. In addition to the database files, the backup facility should create a copy of related database objects including the repository

(or system catalog), database indexes, source libraries, and so on. Typically, a backup copy is produced at least once per day. The copy should be stored in a secured location where it is protected from loss or damage. The backup copy is used to restore the database in the event of hardware failure, catastrophic loss, or damage.

Some DBMSs provide backup utilities for a DBA to use to make backups; other systems assume that the DBA will use the operating system commands, export commands, or SELECT . . . INTO SQL commands to perform backups. Because performing the nightly backup for a particular database is repetitive, creating a script that automates regular backups will save time and result in fewer backup errors.

With large databases, regular full backups may be impractical because the time required to perform a backup may exceed the time available. Or, a database may be a critical system that must always remain available; in such a case, a cold backup, where the database is shut down, is not practical. As a result, backups may be taken of dynamic data regularly (a so-called *hot backup*, in which only a selected portion of the database is shut down from use), but backups of static data, which don't change frequently, may be taken less often. Incremental backups, which record changes made since the last full backup, but which do not take so much time to complete, may also be taken on an interim basis, allowing for longer periods of time between full backups. Thus, backup strategies must be based on the demands being placed on the database systems.

Database downtime can be very expensive. The lost revenue from downtime (e.g., inability to take orders or place reservations) needs to be balanced against the cost of additional technology, primarily disk storage, to achieve a desired level of availability. To help achieve the desired level of reliability, some DBMSs will automatically make backup (often called fallback) copies of the database in real time as the database is updated. These fallback copies are usually stored on separate disk drives and disk controllers, and they are used as live backup copies if portions of the database become inaccessible due to hardware failures. As the cost of secondary storages steadily decreases, the cost to make redundant copies becomes more practical in more situations. Fallback copies are different from RAID storage, discussed in Chapter 5, because the DBMS is making copies of only the database as database transactions occur, whereas RAID is used by the operating system for making redundant copies of all storage elements as any page is updated.

JOURNALIZING FACILITIES A DBMS must provide **journalizing facilities** to produce an audit trail of **transactions** and database changes. In the event of a failure, a consistent database state can be reestablished, using the information in the journals together with the most recent complete backup. As Figure 11-8 shows, there are two basic journals, or logs. The first is the **transaction log**, which contains a record of the essential data for each transaction that is processed against the database. Data that are typically recorded for each transaction include the transaction code or identification, action or type of transaction (e.g., insert), time of the transaction, terminal number or user ID, input data values, table and records accessed, records modified, and possibly the old and new field values.

The second kind of log is a **database change log**, which contains before and after images of records that have been modified by transactions. A **before image** is simply a copy of a record before it has been modified, and an **after image** is a copy of the same record after it has been modified. Some systems also keep a security log, which can alert the DBA to any security violations that occur or are attempted. The recovery manager uses these logs to undo and redo operations, which we explain later in this chapter. These logs may be kept on disk or tape; because they are critical to recovery, they, too, must be backed up.

CHECKPOINT FACILITY A **checkpoint facility** in a DBMS periodically refuses to accept any new transactions. All transactions in progress are completed, and the journal files are brought up-to-date. At this point, the system is in a quiet state, and the database and transaction logs are synchronized. The DBMS writes a special record (called a *checkpoint record*) to the log file, which is like a snapshot of the state of the database. The checkpoint record contains information necessary to restart the system. Any dirty data blocks (i.e., pages of memory that contain changes that have not yet been written out to disk)

Journalizing facility
An audit trail of transactions and database changes.

Transaction
A discrete unit of work that must be completely processed or not processed at all within a computer system. Entering a customer order is an example of a transaction.

Transaction log
A record of the essential data for each transaction that is processed against the database.

Database change log
A log that contains before and after images of records that have been modified by transactions.

Before image
A copy of a record (or page of memory) before it has been modified.

After image
A copy of a record (or page of memory) after it has been modified.

Checkpoint facility
A facility by which a DBMS periodically refuses to accept any new transactions. The system is in a quiet state, and the database and transaction logs are synchronized.

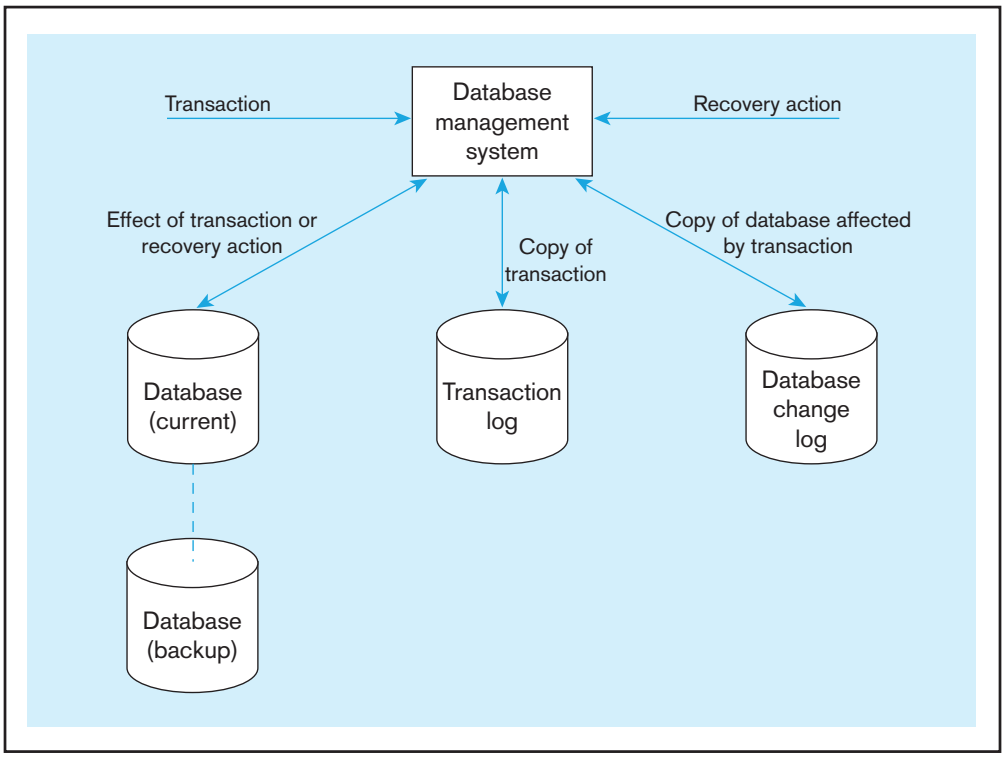


FIGURE 11-8 **Database audit trail**

are written from memory to disk storage, thus ensuring that all changes made prior to taking the checkpoint have been written to long-term storage.

A DBMS may perform checkpoints automatically (which is preferred) or in response to commands in user application programs. Checkpoints should be taken frequently (say, several times an hour). When failures occur, it is often possible to resume processing from the most recent checkpoint. Thus, only a few minutes of processing work must be repeated, compared with several hours for a complete restart of the day's processing.

Recovery manager
A module of a DBMS that restores the database to a correct condition when a failure occurs and then resumes processing user questions.

RECOVERY MANAGER The **recovery manager** is a module of a DBMS that restores the database to a correct condition when a failure occurs and then resumes processing user requests. The type of restart used depends on the nature of the failure. The recovery manager uses the logs shown in Figure 11-8 (as well as the backup copy, if necessary) to restore the database.

Recovery and Restart Procedures

The type of recovery procedure that is used in a given situation depends on the nature of the failure, the sophistication of the DBMS recovery facilities, and operational policies and procedures. Following is a discussion of the techniques that are most frequently used.

DISK MIRRORING To be able to switch to an existing copy of a database, the database must be mirrored. That is, at least two copies of the database must be kept and updated simultaneously. When a media failure occurs, processing is switched to the duplicate copy of the database. This strategy allows for the fastest recovery and has become increasingly popular for applications requiring high availability as the cost of long-term storage has dropped. Level 1 RAID systems implement mirroring. A damaged disk can be rebuilt from the mirrored disk with no disruption in service to the user. Such disks are referred to as being *hot-swappable*. This strategy does not protect against loss of power or catastrophic damage to both databases, though. See Chapter 5 for a more detailed discussion of RAID.

RESTORE/RERUN The **restore/rerun** technique involves reprocessing the day's transactions (up to the point of failure) against the backup copy of the database or portion of the database being recovered. First, the database is shut down, and then the most recent copy of the database or file to be recovered (say, from the previous day) is mounted, and all transactions that have occurred since that copy (which are stored on the transaction log) are rerun. This may also be a good time to make a backup copy and clear out the transaction, or redo, log.

Restore/rerun
A technique that involves reprocessing the day's transactions (up to the point of failure) against the backup copy of the database.

The advantage of restore/rerun is its simplicity. The DBMS does not need to create a database change journal, and no special restart procedures are required. However, there are two major disadvantages. First, the time to reprocess transactions may be prohibitive. Depending on the frequency with which backup copies are made, several hours of reprocessing may be required. Processing new transactions will have to be deferred until recovery is completed, and if the system is heavily loaded, it may be impossible to catch up. The second disadvantage is that the sequencing of transactions will often be different from when they were originally processed, which may lead to quite different results. For example, in the original run, a customer deposit may be posted before a withdrawal. In the rerun, the withdrawal transaction may be attempted first and may lead to sending an insufficient funds notice to the customer. For these reasons, restore/rerun is not a sufficient recovery procedure and is generally used only as a last resort in database processing.

MAINTAINING TRANSACTION INTEGRITY A database is updated by processing transactions that result in changes to one or more database records. If an error occurs during the processing of a transaction, the database may be compromised, and some form of database recovery is required. Thus, to understand database recovery, we must first understand the concept of transaction integrity.

A business transaction is a sequence of steps that constitute some well-defined business activity. Examples of business transactions are Admit Patient in a hospital and Enter Customer Order in a manufacturing company. Normally, a business transaction requires several actions against the database. For example, consider the transaction Enter Customer Order. When a new customer order is entered, the following steps may be performed by an application program:

1. Input the order data (keyed by the user).
2. Read the CUSTOMER record (or insert record if a new customer).
3. Accept or reject the order. If Balance Due plus Order Amount does not exceed Credit Limit, accept the order; otherwise, reject it.
4. If the order is accepted, increase Balance Due by Order Amount. Store the updated CUSTOMER record. Insert the accepted ORDER record in the database.

When processing transactions, a DBMS must ensure that the transactions follow four well-accepted properties, called the ACID properties:

1. ***Atomic,*** meaning that the transaction cannot be subdivided and, hence, it must be processed in its entirety or not at all. Once the whole transaction is processed, we say that the changes are *committed*. If the transaction fails at any midpoint, we say that it has aborted. For example, suppose that the program accepts a new customer order, increases Balance Due, and stores the updated CUSTOMER record. However, suppose that the new ORDER record is not inserted successfully (perhaps due to a duplicate Order Number key or insufficient physical file space). In this case, we want none of the parts of the transaction to affect the database.
2. ***Consistent,*** meaning that any database constraints that must be true before the transaction must also be true after the transaction. For example, if the inventory on-hand balance must be the difference between total receipts minus total issues, this will be true both before and after an order transaction, which depletes the on-hand balance to satisfy the order.
3. ***Isolated,*** meaning that changes to the database are not revealed to users until the transaction is committed. For example, this property means that other users

do not know what the on-hand inventory is until an inventory transaction is complete; this property then usually means that other users are prohibited from simultaneously updating and possibly even reading data that are in the process of being updated. We discuss this topic in more detail later under concurrency controls and locking. A consequence of transactions being isolated from one another is that concurrent transactions (i.e., several transactions in some partial state of completion) all affect the database as if they were presented to the DBMS in serial fashion.

4. ***Durable,*** meaning that changes are permanent. Thus, once a transaction is committed, no subsequent failure of the database can reverse the effect of the transaction.

Transaction boundaries
The logical beginning and end of a transaction.

To maintain transaction integrity, the DBMS must provide facilities for the user or application program to define **transaction boundaries**—that is, the logical beginning and end of a transaction. In SQL, the BEGIN TRANSACTION statement is placed in front of the first SQL command within the transaction, and the COMMIT command is placed at the end of the transaction. Any number of SQL commands may come in between these two commands; these are the database processing steps that perform some well-defined business activity, as explained earlier. If a command such as ROLLBACK is processed after a BEGIN TRANSACTION is executed and before a COMMIT is executed, the DBMS aborts the transaction and undoes the effects of the SQL statements processed so far within the transaction boundaries. The application would likely be programmed to execute a ROLLBACK when the DBMS generates an error message performing an UPDATE or INSERT command in the middle of the transaction. The DBMS thus commits (makes durable) changes for successful transactions (those that reach the COMMIT statement) and effectively rejects changes from transactions that are aborted (those that encounter a ROLLBACK). Any SQL statement encountered after a COMMIT or ROLLBACK and before a BEGIN TRANSACTION is executed as a single statement transaction, automatically committed if it executed without error, aborted if any error occurs during its execution.

Although conceptually a transaction is a logical unit of business work, such as a customer order or receipt of new inventory from a supplier, you may decide to break the business unit of work into several database transactions for database processing reasons. For example, because of the isolation property, a transaction that takes many commands and a long time to process may prohibit other uses of the same data at the same time, thus delaying other critical (possibly read-only) work. Some database data are used frequently, so it is important to complete transactional work on these so-called hotspot data as quickly as possible. For example, a primary key and its index for bank account numbers will likely need to be accessed by every ATM transaction, so the database transaction must be designed to use and release this data quickly. Also, remember, all the commands between the boundaries of a transaction must be executed, even those commands seeking input from an online user. If a user is slow to respond to input requests within the boundaries of a transaction, other users may encounter significant delays. Thus, if possible, collect all user input before beginning a transaction. Also, to minimize the length of a transaction, check for possible errors, such as duplicate keys or insufficient account balance, as early in the transaction as possible, so portions of the database can be released as soon as possible for other users if the transaction is going to be aborted. Some constraints (e.g., balancing the number of units of an item received with the number placed in inventory less returns) cannot be checked until many database commands are executed, so the transaction must be long to ensure database integrity. Thus, the general guideline is to make a database transaction as short as possible while still maintaining the integrity of the database.

Backward recovery (rollback)
The backout, or undo, of unwanted changes to a database. Before images of the records that have been changed are applied to the database, and the database is returned to an earlier state. Rollback is used to reverse the changes made by transactions that have been aborted, or terminated abnormally.

BACKWARD RECOVERY With **backward recovery** (also called **rollback**), the DBMS backs out of or undoes unwanted changes to the database. As Figure 11-9a shows, before images of the records that have been changed are applied to the database.

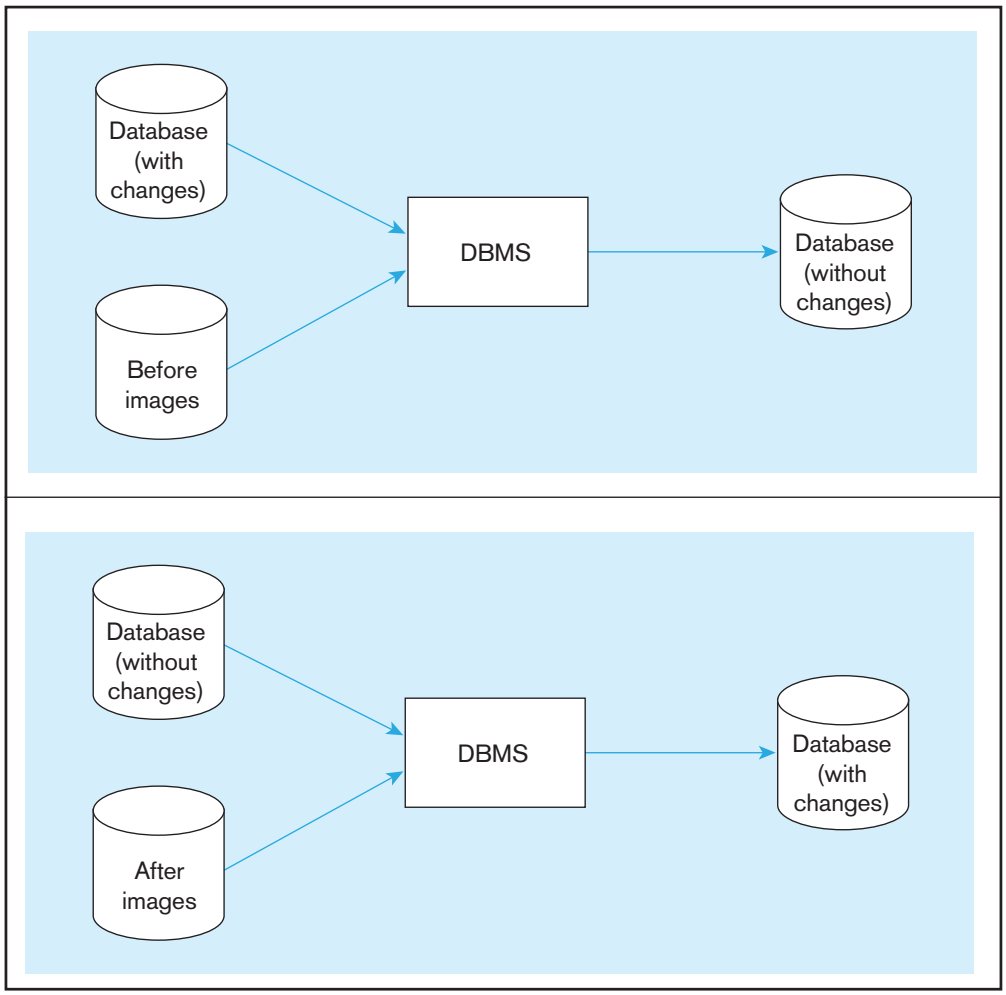


FIGURE 11-9 **Basic recovery techniques**
(a) Rollback

(b) Rollforward

As a result, the database is returned to an earlier state; the unwanted changes are eliminated.

Backward recovery is used to reverse the changes made by transactions that have aborted, or terminated abnormally. To illustrate the need for backward recovery (or UNDO), suppose that a banking transaction will transfer $100 in funds from the account for customer A to the account for customer B. The following steps are performed:

1. The program reads the record for customer A and subtracts $100 from the account balance.
2. The program then reads the record for customer B and adds $100 to the account balance. Now the program writes the updated record for customer A to the database. However, in attempting to write the record for customer B, the program encounters an error condition (e.g., a disk fault) and cannot write the record. Now the database is inconsistent—record A has been updated but record B has not—and the transaction must be aborted. An UNDO command will cause the recovery manager to apply the before image for record A to restore the account balance to its original value. (The recovery manager may then restart the transaction and make another attempt.)

FORWARD RECOVERY With **forward recovery** (also called **rollforward**), the DBMS starts with an earlier copy of the database. Applying after images (the results of good transactions) quickly moves the database forward to a later state (see Figure 11-9b).

Forward recovery (rollforward) A technique that starts with an earlier copy of a database. After images (the results of good transactions) are applied to the database, and the database is quickly moved forward to a later state.

Forward recovery is much faster and more accurate than restore/rerun, for the following reasons:

- The time-consuming logic of reprocessing each transaction does not have to be repeated.
- Only the most recent after images need to be applied. A database record may have a series of after images (as a result of a sequence of updates), but only the most recent, "good" after image, is required for rollforward.

With rollforward, the problem of different sequencing of transactions is avoided, because the results of applying the transactions (rather than the transactions themselves) are used.

Types of Database Failure

A wide variety of failures can occur in processing a database, ranging from the input of an incorrect data value to complete loss or destruction of the database. Four of the most common types of problems are aborted transactions, incorrect data, system failure, and database loss or destruction. Each of these types of problems is described in the following sections, and possible recovery procedures are indicated (see Table 11-1).

Aborted transaction
A transaction in progress that terminates abnormally.

ABORTED TRANSACTIONS As we noted earlier, a transaction frequently requires a sequence of processing steps to be performed. An **aborted transaction** terminates abnormally. Some reasons for this type of failure are human error, input of invalid data, hardware failure, and deadlock (covered in the next section). A common type of hardware failure is the loss of transmission in a communications link when a transaction is in progress.

When a transaction aborts, we want to "back out" the transaction and remove any changes that have been made (but not committed) to the database. The recovery manager accomplishes this through backward recovery (applying before images for the transaction in question). This function should be accomplished automatically by the DBMS, which then notifies the user to correct and resubmit the transaction. Other procedures, such as rollforward or transaction reprocessing, could be applied to bring the database to the state it was in just prior to the abort occurrence, but rollback is the preferred procedure in this case.

INCORRECT DATA A more complex situation arises when the database has been updated with incorrect, but valid, data. For example, an incorrect grade may be recorded for a student, or an incorrect amount could be input for a customer payment.

Incorrect data are difficult to detect and often lead to complications. To begin with, some time may elapse before an error is detected and the database record (or records)

TABLE 11-1 Responses to Database Failure

Type of Failure	Recovery Technique
Aborted transaction	Rollback (preferred)
	Rollforward/return transactions to state just prior to abort
Incorrect data (update inaccurate)	Rollback (preferred)
	Reprocess transactions without inaccurate data updates
	Compensating transactions
System failure (database intact)	Switch to duplicate database (preferred)
	Rollback
	Restart from checkpoint (rollforward)
Database destruction	Switch to duplicate database (preferred)
	Rollforward
	Reprocess transactions

corrected. By this time, numerous other users may have used the erroneous data, and a chain reaction of errors may have occurred as various applications made use of the incorrect data. In addition, transaction outputs (e.g., documents and messages) based on the incorrect data may be transmitted to persons. An incorrect grade report, for example, may be sent to a student or an incorrect statement sent to a customer.

When incorrect data have been processed, the database may be recovered in one of the following ways:

- If the error is discovered soon enough, backward recovery may be used. (However, care must be taken to ensure that all subsequent errors have been reversed.)
- If only a few errors have occurred, a series of compensating transactions may be introduced through human intervention to correct the errors.
- If the first two measures are not feasible, it may be necessary to restart from the most recent checkpoint before the error occurred, and subsequent transactions processed without the error.

Any erroneous messages or documents that have been produced by the erroneous transaction will have to be corrected by appropriate human intervention (letters of explanation, telephone calls, etc.).

SYSTEM FAILURE In a system failure, some component of the system fails, but the database is not damaged. Some causes of system failure are power loss, operator error, loss of communications transmission, and system software failure.

When a system crashes, some transactions may be in progress. The first step in recovery is to back out those transactions, using before images (backward recovery). Then, if the system is mirrored, it may be possible to switch to the mirrored data and rebuild the corrupted data on a new disk. If the system is not mirrored, it may not be possible to restart because status information in main memory has been lost or damaged. The safest approach is to restart from the most recent checkpoint before the system failure. The database is rolled forward by applying after images for all transactions that were processed after that checkpoint.

DATABASE DESTRUCTION In the case of **database destruction**, the database itself is lost, destroyed, or cannot be read. A typical cause of database destruction is a disk drive failure (or head crash).

Database destruction
The database itself is lost, destroyed, or cannot be read.

Again, using a mirrored copy of the database is the preferred strategy for recovering from such an event. If there is no mirrored copy, a backup copy of the database is required. Forward recovery is used to restore the database to its state immediately before the loss occurred. Any transactions that may have been in progress when the database was lost are restarted.

Disaster Recovery

Every organization requires contingency plans for dealing with disasters that may severely damage or destroy its data center. Such disasters may be natural (e.g., floods, earthquakes, tornadoes, hurricanes) or human-caused (e.g., wars, sabotage, terrorist attacks). For example, the 2001 terrorist attacks on the World Trade Center resulted in the complete destruction of several data centers and widespread loss of data.

Planning for disaster recovery is an organization-wide responsibility. Database administration is responsible for developing plans for recovering the organization's data and for restoring data operations. Following are some of the major components of a recovery plan (Mullins, 2002):

- Develop a detailed written disaster recovery plan. Schedule regular tests of the plan.
- Choose and train a multidisciplinary team to carry out the plan.
- Establish a backup data center at an offsite location. This site must be located a sufficient distance from the primary site so that no foreseeable disaster will disrupt both sites. If an organization has two or more data centers, each site may

serve as a backup for one of the others. If not, the organization may contract with a disaster recovery service provider.
- Send backup copies of databases to the backup data center on a scheduled basis. Database backups may be sent to the remote site by courier or transmitted by replication software.

CONTROLLING CONCURRENT ACCESS

Concurrency control
The process of managing simultaneous operations against a database so that data integrity is maintained and the operations do not interfere with each other in a multiuser environment.

Databases are shared resources. Database administrators must expect and plan for the likelihood that several users will attempt to access and manipulate data at the same time. With concurrent processing involving updates, a database without **concurrency control** will be compromised due to interference between users. There are two basic approaches to concurrency control: a pessimistic approach (involving locking) and an optimistic approach (involving versioning). We summarize both of these approaches in the following sections.

If users are only reading data, no data integrity problems will be encountered because no changes will be made in the database. However, if one or more users are updating data, then potential problems with maintaining data integrity arise. When more than one transaction is being processed against a database at the same time, the transactions are considered to be concurrent. The actions that must be taken to ensure that data integrity is maintained are called *currency control actions*. Although these actions are implemented by a DBMS, a database administrator must understand these actions and may expect to make certain choices governing their implementation.

Remember that a CPU can process only one instruction at a time. As new transactions are submitted while other processing is occurring against the database, the transactions are usually interleaved, with the CPU switching among the transactions so that some portion of each transaction is performed as the CPU addresses each transaction in turn. Because the CPU is able to switch among transactions so quickly, most users will not notice that they are sharing CPU time with other users.

The Problem of Lost Updates

The most common problem encountered when multiple users attempt to update a database without adequate concurrency control is lost updates. Figure 11-10 shows a common situation. John and Marsha have a joint checking account, and both want to withdraw some cash at the same time, each using an ATM terminal in a different location. Figure 11-10 shows the sequence of events that might occur in the absence of a concurrency control mechanism. John's transaction reads the account balance (which is $1,000) and he proceeds to withdraw $200. Before the transaction writes the new account balance ($800), Marsha's transaction reads the account balance (which is still $1,000). She then withdraws $300, leaving a balance of $700. Her transaction then writes this account balance, which replaces the one written by John's transaction. The effect of John's update has been lost due to interference between the transactions, and the bank is unhappy.

Inconsistent read problem
An unrepeatable read, one that occurs when one user reads data that have been partially updated by another user.

Another similar type of problem that may occur when concurrency control is not established is the **inconsistent read problem**. This problem occurs when one user reads data that have been partially updated by another user. The read will be incorrect, and is sometimes referred to as a *dirty read* or an *unrepeatable read*. The lost update and inconsistent read problems arise when the DBMS does not isolate transactions, part of the ACID transaction properties.

Serializability

Concurrent transactions need to be processed in isolation so that they do not interfere with each other. If one transaction were entirely processed before another transaction, no interference would occur. Procedures that process transactions so that the outcome is the same as this are called *serializable*. Processing transactions using a serializable schedule will give the same results as if the transactions had been processed one after the other.

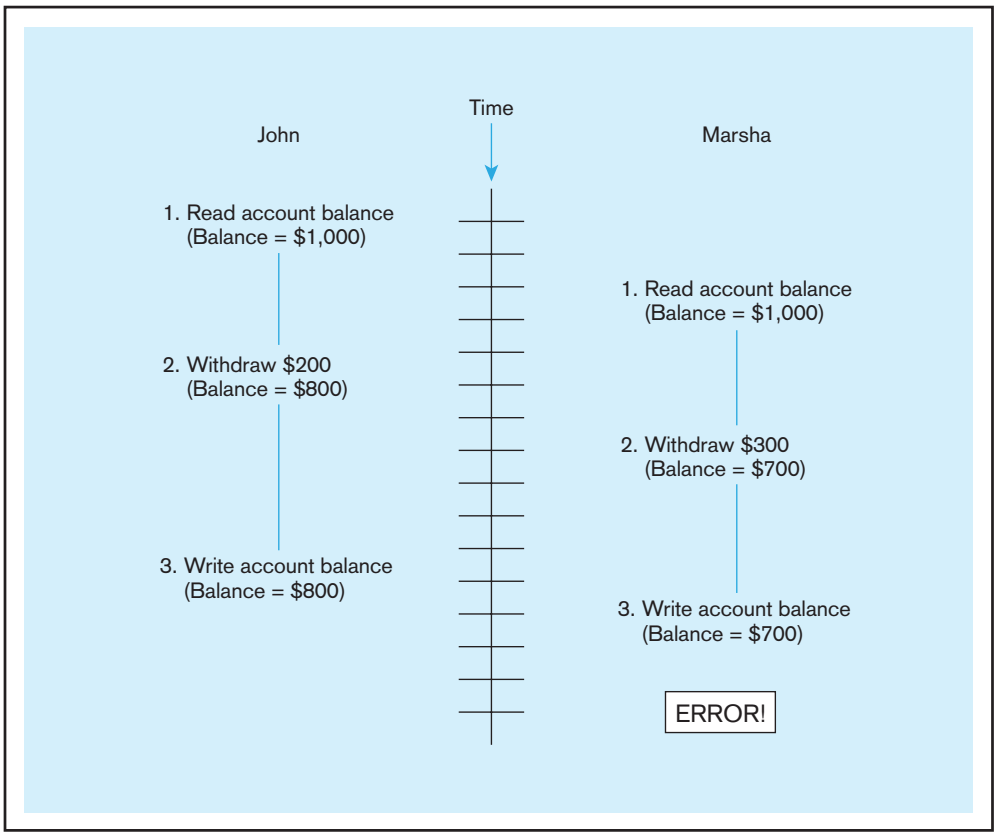


FIGURE 11-10 **Lost update (no concurrency control in effect)**

Schedules are designed so that transactions that will not interfere with each other can still be run in parallel. For example, transactions that request data from different tables in a database will not conflict with each other and can be run concurrently without causing data integrity problems. Serializability is achieved by different means, but locking mechanisms are the most common type of concurrency control mechanism. With **locking**, any data that are retrieved by a user for updating must be locked, or denied to other users, until the update is complete or aborted. Locking data is much like checking a book out of the library; it is unavailable to others until the borrower returns it.

Locking
A process in which any data that are retrieved by a user for updating must be locked, or denied to other users, until the update is completed or aborted.

Locking Mechanisms

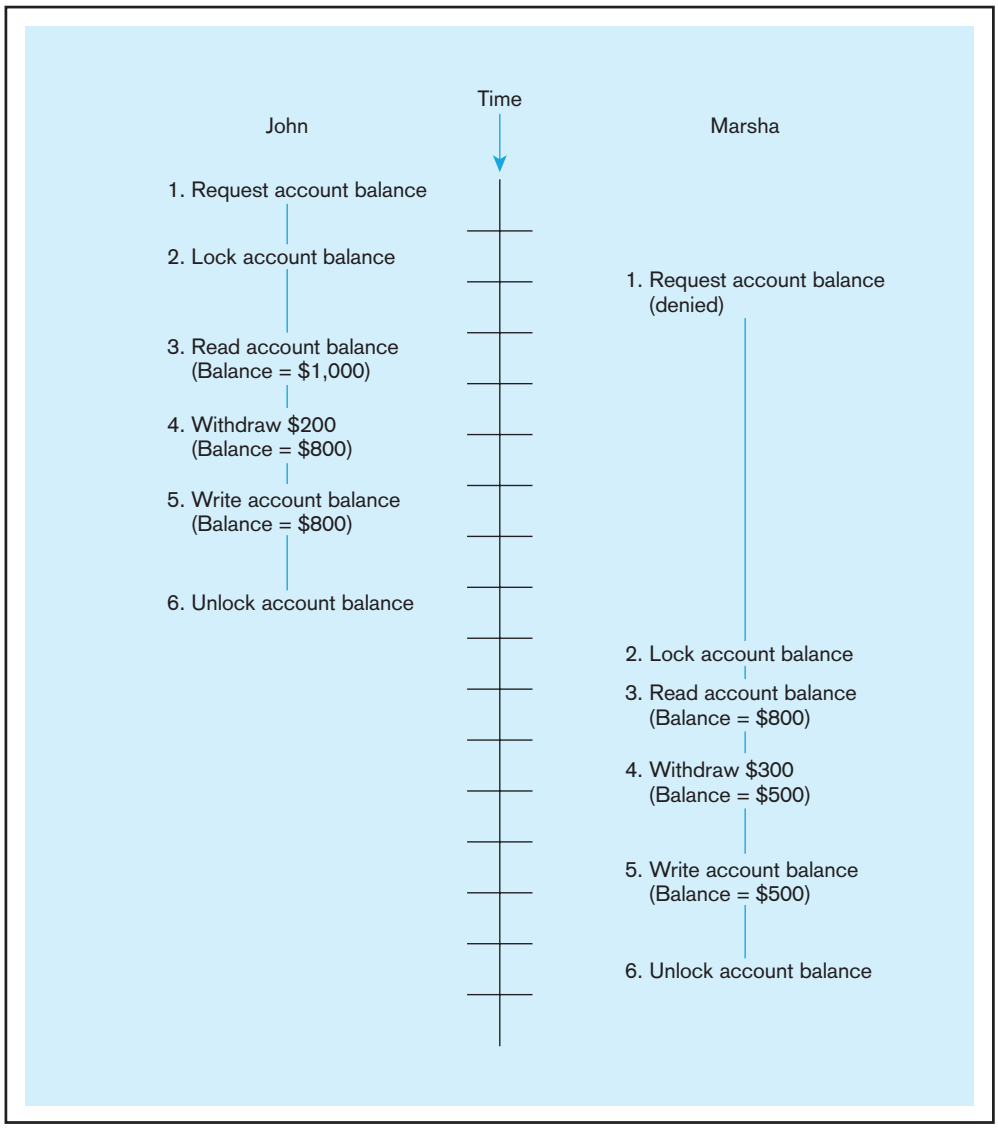
Figure 11-11 shows the use of record locks to maintain data integrity. John initiates a withdrawal transaction from an ATM. Because John's transaction will update this record, the application program locks this record before reading it into main memory. John proceeds to withdraw $200, and the new balance ($800) is computed. Marsha has initiated a withdrawal transaction shortly after John, but her transaction cannot access the account record until John's transaction has returned the updated record to the database and unlocked the record. The locking mechanism thus enforces a sequential updating process that prevents erroneous updates.

LOCKING LEVEL An important consideration in implementing concurrency control is choosing the locking level. The **locking level** (also called **lock granularity**) is the extent of the database resource that is included with each lock. Most commercial products implement locks at one of the following levels:

Locking level (lock granularity)
The extent of a database resource that is included with each lock.

- ***Database*** The entire database is locked and becomes unavailable to other users. This level has limited application, such as during a backup of the entire database (Rodgers, 1989).
- ***Table*** The entire table containing a requested record is locked. This level is appropriate mainly for bulk updates that will update the entire table, such as giving all employees a 5 percent raise.

FIGURE 11-11 Updates with locking (concurrency control)



- ***Block or page*** The physical storage block (or page) containing a requested record is locked. This level is the most commonly implemented locking level. A page will be a fixed size (4K, 8K, etc.) and may contain records of more than one type.
- ***Record*** Only the requested record (or row) is locked. All other records, even within a table, are available to other users. It does impose some overhead at run time when several records are involved in an update.
- ***Field*** Only the particular field (or column) in a requested record is locked. This level may be appropriate when most updates affect only one or two fields in a record. For example, in inventory control applications the quantity-on-hand field changes frequently, but other fields (e.g., description and bin location) are rarely updated. Field-level locks require considerable overhead and are seldom used.

TYPES OF LOCKS So far, we have discussed only locks that prevent all access to locked items. In reality, a database administrator can generally choose between two types of locks:

Shared lock (S lock, or read lock)
A technique that allows other transactions to read but not update a record or another resource.

1. ***Shared locks*** **Shared locks** (also called **S locks**, or **read locks**) allow other transactions to read (but not update) a record or other resource. A transaction should place a shared lock on a record or data resource when it will only read but not update that record. Placing a shared lock on a record prevents another user from placing an exclusive lock, but not a shared lock, on that record.

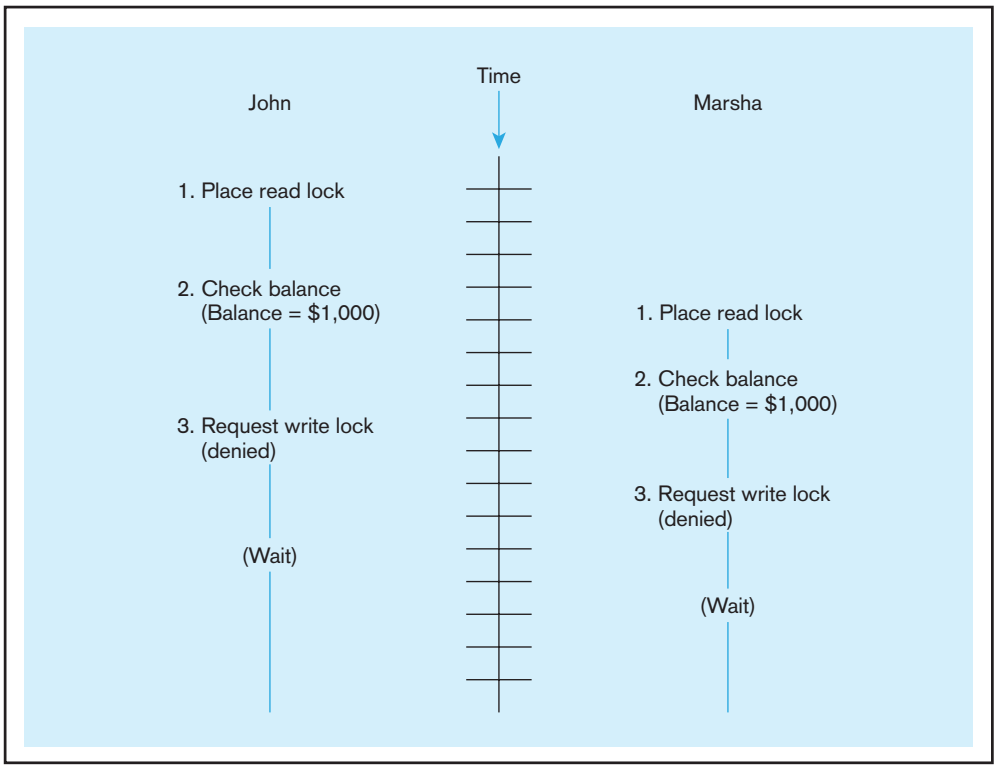


FIGURE 11-12 The problem of deadlock

2. *Exclusive locks* **Exclusive locks** (also called **X locks**, or **write locks**) prevent another transaction from reading (and therefore updating) a record until it is unlocked. A transaction should place an exclusive lock on a record when it is about to update that record (Descollonges, 1993). Placing an exclusive lock on a record prevents another user from placing any type of lock on that record.

Exclusive lock (X lock, or write lock)
A technique that prevents another transaction from reading and therefore updating a record until it is unlocked.

Figure 11-12 shows the use of shared and exclusive locks for the checking account example. When John initiates his transaction, the program places a read lock on his account record, because he is reading the record to check the account balance. When John requests a withdrawal, the program attempts to place an exclusive lock (write lock) on the record because this is an update operation. However, as you can see in the figure, Marsha has already initiated a transaction that has placed a read lock on the same record. As a result, his request is denied; remember that if a record is a read lock, another user cannot obtain a write lock.

DEADLOCK Locking solves the problem of erroneous updates but may lead to a problem called **deadlock**—an impasse that results when two or more transactions have locked a common resource, and each must wait for the other to unlock that resource. Figure 11-12 shows a simple example of deadlock. John's transaction is waiting for Marsha's transaction to remove the read lock from the account record, and vice versa. Neither person can withdraw money from the account, even though the balance is more than adequate.

Deadlock
An impasse that results when two or more transactions have locked a common resource, and each waits for the other to unlock that resource.

Figure 11-13 shows a slightly more complex example of deadlock. In this example, user A has locked record X, and user B has locked record Y. User A then requests record Y (intending to update), and user B requests record X (also intending to update). Both requests are denied, because the requested records are already locked. Unless the DBMS intervenes, both users will wait indefinitely.

MANAGING DEADLOCK There are two basic ways to resolve deadlocks: deadlock prevention and deadlock resolution. When **deadlock prevention** is employed, user programs must lock all records they will require at the beginning of a transaction, rather than one at a time. In Figure 11-13, user A would have to lock both records X and Y before processing the transaction. If either record is already locked, the

Deadlock prevention
A method for resolving deadlocks in which user programs must lock all records they require at the beginning of a transaction (rather than one at a time).

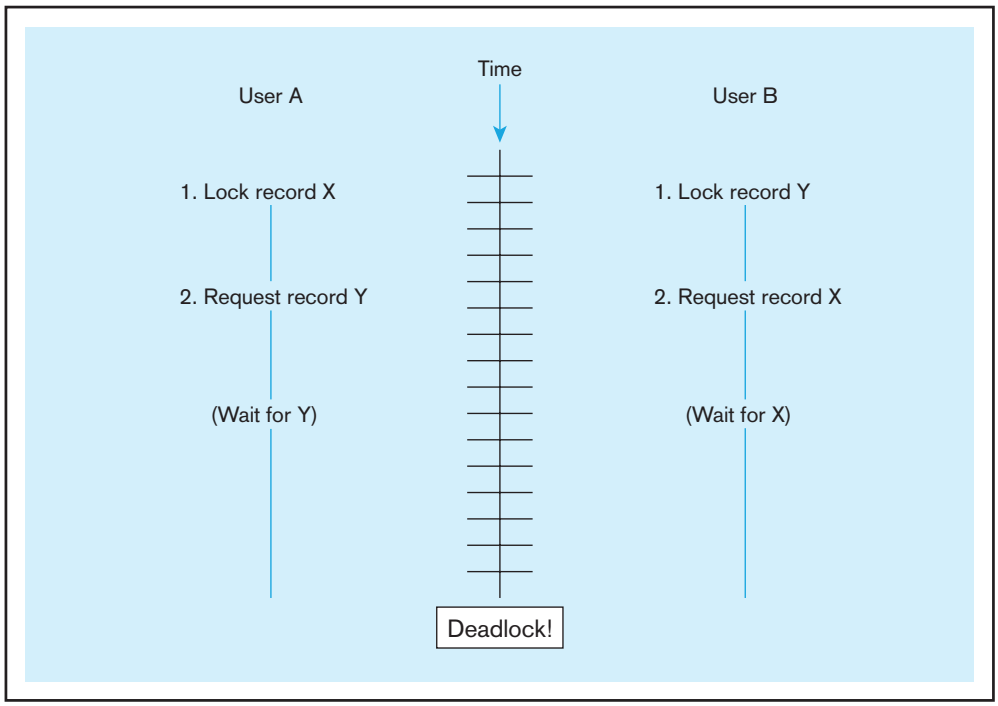


FIGURE 11-13 Another example of deadlock

program must wait until it is released. Where all locking operations necessary for a transaction occur before any resources are unlocked, a **two-phase locking protocol** is being used. Once any lock obtained for the transaction is released, no more locks may be obtained. Thus, the phases in the two-phase locking protocol are often referred to as a growing phase (where all necessary locks are acquired) and a shrinking phase (where all locks are released). Locks do not have to be acquired simultaneously. Frequently, some locks will be acquired, processing will occur, and then additional locks will be acquired as needed.

Two-phase locking protocol
A procedure for acquiring the necessary locks for a transaction in which all necessary locks are acquired before any locks are released, resulting in a growing phase when locks are acquired and a shrinking phase when they are released.

Locking all the required records at the beginning of a transaction (called *conservative two-phase locking*) prevents deadlock. Unfortunately, it is often difficult to predict in advance what records will be required to process a transaction. A typical program has many processing parts and may call other programs in varying sequences. As a result, deadlock prevention is not always practical.

Two-phase locking, in which each transaction must request records in the same sequence (i.e., serializing the resources), also prevents deadlock, but again this may not be practical.

The second, and more common, approach is to allow deadlocks to occur but to build mechanisms into the DBMS for detecting and breaking the deadlocks. Essentially, these **deadlock resolution** mechanisms work as follows: The DBMS maintains a matrix of resource usage, which, at a given instant, indicates what subjects (users) are using what objects (resources). By scanning this matrix, the computer can detect deadlocks as they occur. The DBMS then resolves the deadlocks by "backing out" one of the deadlocked transactions. Any changes made by that transaction up to the time of deadlock are removed, and the transaction is restarted when the required resources become available. We will describe the procedure for backing out shortly.

Deadlock resolution
An approach to dealing with deadlocks that allows deadlocks to occur but builds mechanisms into the DBMS for detecting and breaking the deadlocks.

Versioning

Locking, as described here, is often referred to as a pessimistic concurrency control mechanism because each time a record is required, the DBMS takes the highly cautious approach of locking the record so that other programs cannot use it. In reality, in most cases other users will not request the same documents, or they may only want to read them, which is not a problem (Celko, 1992). Thus, conflicts are rare.

A newer approach to concurrency control, called **versioning**, takes the optimistic approach that most of the time other users do not want the same record, or if they do, they only want to read (but not update) the record. With versioning, there is no form of

Versioning
An approach to concurrency control in which each transaction is restricted to a view of the database as of the time that transaction started, and when a transaction modifies a record, the DBMS creates a new record version instead of overwriting the old record. Hence, no form of locking is required.

locking. Each transaction is restricted to a view of the database as of the time that transaction started, and when a transaction modifies a record, the DBMS creates a new record version instead of overwriting the old record.

The best way to understand versioning is to imagine a central records room, corresponding to the database (Celko, 1992). The records room has a service window. Users (corresponding to transactions) arrive at the window and request documents (corresponding to database records). However, the original documents never leave the records room. Instead, the clerk (corresponding to the DBMS) makes copies of the requested documents and time stamps them. Users then take their private copies (or versions) of the documents to their own workplace and read them and/or make changes. When finished, they return their marked-up copies to the clerk. The clerk merges the changes from marked-up copies into the central database. When there is no conflict (e.g., when only one user has made changes to a set of database records), that user's changes are merged directly into the public (or central) database.

Suppose instead that there is a conflict; for example, say that two users have made conflicting changes to their private copy of the database. In this case, the changes made by one of the users are committed to the database. (Remember that the transactions are time-stamped, so that the earlier transaction can be given priority.) The other user must be told that there was a conflict, and his work cannot be committed (or incorporated into the central database). He must check out another copy of the data records and repeat the previous work. Under the optimistic assumption, this type of rework will be the exception rather than the rule.

Figure 11-14 shows a simple example of the use of versioning for the checking account example. John reads the record containing the account balance, successfully withdraws $200, and the new balance ($800) is posted to the account with a COMMIT statement. Meanwhile, Marsha has also read the account record and requested a withdrawal, which is posted to her local version of the account record. However, when the transaction attempts to COMMIT, it discovers the update conflict, and her transaction is aborted (perhaps with a message such as "Cannot complete transaction at this time"). Marsha can then restart the transaction, working from the correct starting balance of $800.

The main advantage of versioning over locking is performance improvement. Read-only transactions can run concurrently with updating transactions, without loss of database consistency.

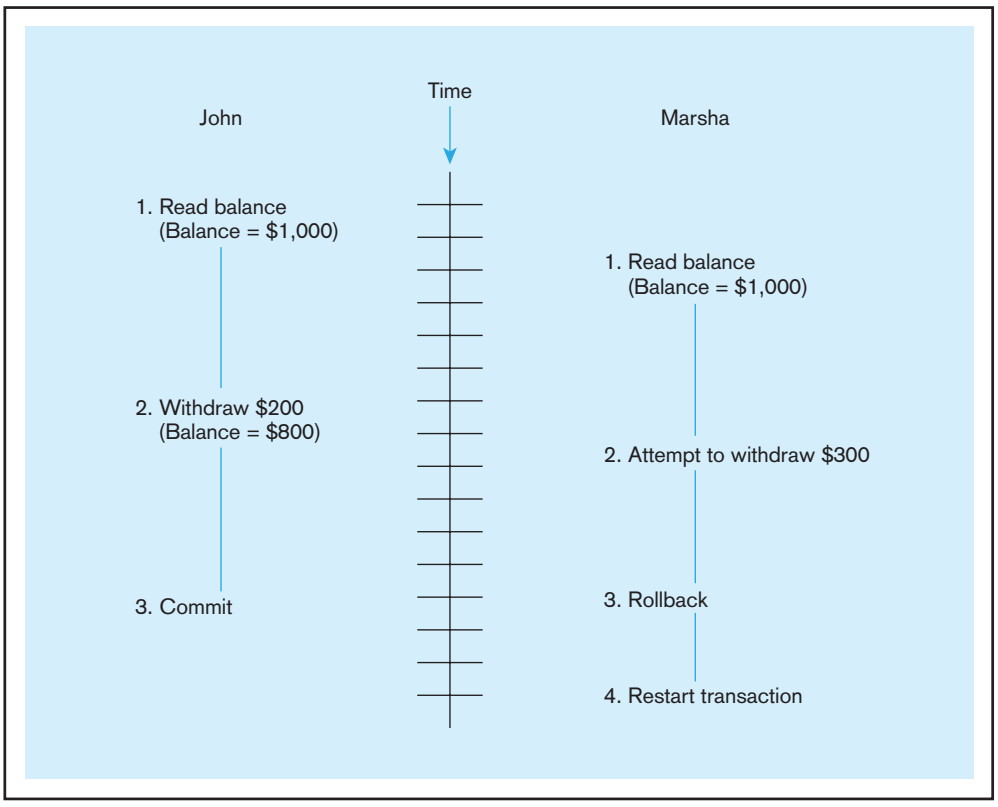


FIGURE 11-14 **The use of versioning**

DATA DICTIONARIES AND REPOSITORIES

In Chapter 1, we defined *metadata* as data that describe the properties or characteristics of end-user data and the context of that data. To be successful, an organization must develop sound strategies to collect, manage, and utilize its metadata. These strategies should address identifying the types of metadata that need to be collected and maintained and developing methods for the orderly collection and storage of that metadata. Data administration is usually responsible for the overall direction of the metadata strategy.

Metadata must be stored and managed using DBMS technology. The collection of metadata is referred to as a *data dictionary* (an older term) or a *repository* (a modern term). We describe each of these terms in this section. Some facilities of RDBMSs to access the metadata stored with a database were described in Chapter 7.

Data Dictionary

Data dictionary
A repository of information about a database that documents data elements of a database.

System catalog
A system-created database that describes all database objects, including data dictionary information, and also includes user access information.

An integral part of relational DBMSs is the **data dictionary**, which stores metadata, or information about the database, including attribute names and definitions for each table in the database. The data dictionary is usually a part of the **system catalog** that is generated for each database. The system catalog describes all database objects, including table-related data such as table names, table creators or owners, column names and data types, foreign keys and primary keys, index files, authorized users, user access privileges, and so forth. The system catalog is created and maintained automatically by the database management system, and the information is stored in systems tables, which may be queried in the same manner as any other data table, if the user has sufficient access privileges.

Data dictionaries may be either active or passive. An *active* data dictionary is managed automatically by the database management software. Active systems are always consistent with the current structure and definition of the database because they are maintained by the system itself. Most relational database management systems now contain active data dictionaries that can be derived from their system catalog. A *passive* data dictionary is managed by the user(s) of the system and is modified whenever the structure of the database is changed. Because this modification must be performed manually by the user, it is possible that the data dictionary will not be current with the current structure of the database. However, the passive data dictionary may be maintained as a separate database. This may be desirable during the design phase because it allows developers to remain independent from using a particular RDBMS for as long as possible. Also, passive data dictionaries are not limited to information that can be discerned by the database management system. Because passive data dictionaries are maintained by the user, they may be extended to contain information about organizational data that is not computerized.

Repositories

Information repository
A component that stores metadata that describe an organization's data and data processing resources, manages the total information processing environment, and combines information about an organization's business information and its application portfolio.

Whereas data dictionaries are simple data element documentation tools, information repositories are used by data administrators and other information specialists to manage the total information processing environment. The **information repository** is an essential component of both the development environment and the production environment. In the application development environment, people (either information specialists or end users) use CASE tools, high-level languages, and other tools to develop new applications. CASE tools may tie automatically to the information repository. In the production environment, people use applications to build databases, keep the data current, and extract data from databases. To build a data warehouse and develop business intelligence applications, it is absolutely essential that an organization build and maintain a comprehensive repository.

As indicated previously, CASE tools often generate information that should be a part of the information repository, as do documentation tools, project management tools, and, of course, the database management software itself. When they were first developed, the information recorded by each of these products was not easily integrated. Now, however, there has been an attempt to make this information more accessible

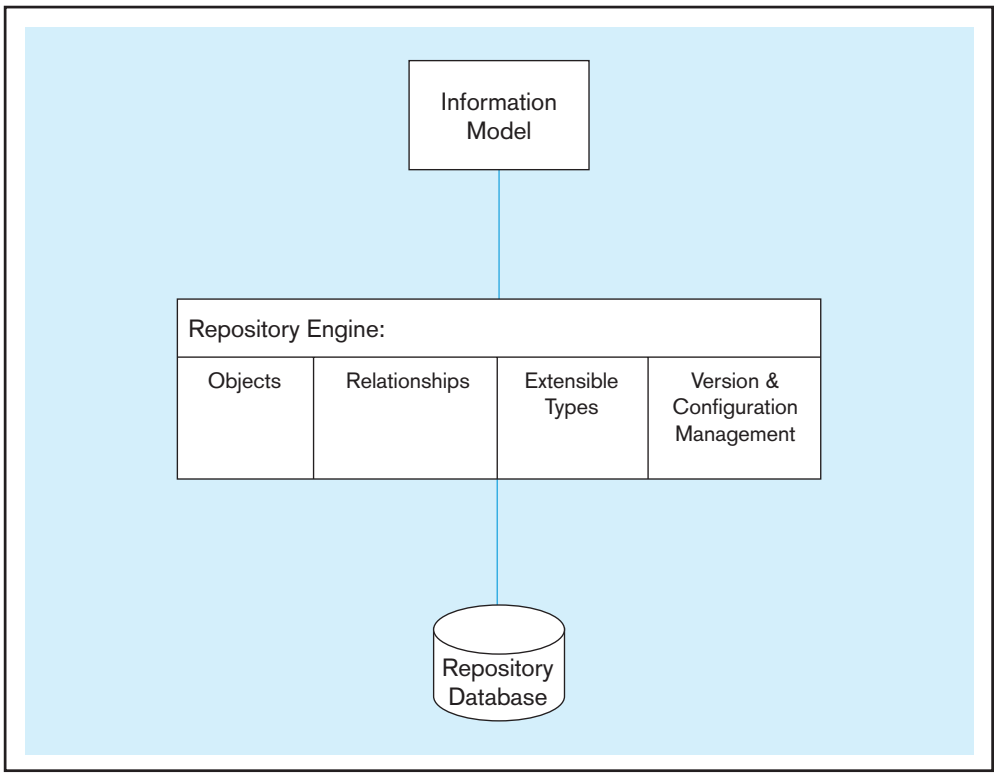


FIGURE 11-15 Three components of repository system architecture

Source: Based on Bernstein (1996)

and shareable. The **Information Resource Dictionary System (IRDS)** is a computer software tool that is used to manage and control access to the information repository. It provides facilities for recording, storing, and processing descriptions of an organization's significant data and data processing resources (Lefkovitz, 1985). When systems are compliant with IRDS, it is possible to transfer data definitions among the data dictionaries generated by the various products. IRDS, which has been adopted as a standard by the International Standards Organization as the ISO/IEC 10027 standard **www.iso.org/iso/catalogue_detail.htm?csnumber=17985** includes a set of rules for storing data dictionary information and for accessing it.

Information Resource Dictionary System (IRDS)
A computer software tool that is used to manage and control access to the information repository.

Figure 11-15 shows the three components of a repository system architecture (Bernstein, 1996). First is an information model. This model is a schema of the information stored in the repository, which can then be used by the tools associated with the database to interpret the contents of the repository. Next is the repository engine, which manages the repository objects. Services, such as reading and writing repository objects, browsing, and extending the information model, are included. Last is the repository database, in which the repository objects are actually stored. Notice that the repository engine supports five core functions (Bernstein, 1996):

1. ***Object management*** Object-oriented repositories store information about objects. As databases become more object oriented, developers will be able to use the information stored about database objects in the information repository. The repository can be based on an object-oriented database or it can add the capability to support objects.
2. ***Relationship management*** The repository engine contains information about object relationships that can be used to facilitate the use of software tools that attach to the database.
3. ***Dynamic extensibility*** The repository information model defines types, which should be easy to extend, that is, to add new types or to extend the definitions of those that already exist. This capability can make it easier to integrate a new software tool into the development process.
4. ***Version management*** During development, it is important to establish version control. The information repository can be used to facilitate version control for

software design tools. Version control of objects is more difficult to manage than version control of files, because there are many more objects than files in an application, and each version of an object may have many relationships.

5. ***Configuration management*** It is necessary to group versioned objects into configurations that represent the entire system, which are also versioned. It may help you to think of a configuration as similar to a file directory, except configurations can be versioned and they contain objects rather than files. Repositories often use checkout systems to manage objects, versions, and configurations. A developer who wishes to use an object checks it out, makes the desired changes, and then checks the object back in. At that time, a new version of the object will be created, and the object will become available to other developers.

As object-oriented database management systems become more available, and as object-oriented programming associated with relational databases increases, the importance of the information repository is also going to increase because object-oriented development requires the use (and reuse) of the metadata contained in the information repository. Also, the metadata and application information generated by different software tools will be more easily integrated into the information repository now that the IRDS standard has been accepted. Although information repositories are already included in the enterprise-level development tools, the increasing emphasis on object-oriented development and the explosion of data warehouse solutions are leading to more widespread use of information repositories.

OVERVIEW OF TUNING THE DATABASE FOR PERFORMANCE

Effective database support results in a reliable database where performance is not subject to interruption from hardware, software, or user problems and where optimal performance is achieved. Tuning a database is not an activity that is undertaken at the time of DBMS installation and/or at the time of implementation of a new application and then disregarded. Rather, performance analysis and tuning is an ongoing part of managing any database, as hardware and software configurations change and as user activity changes. Five areas of DBMS management that should be addressed when trying to maintain a well-tuned database are addressed here: installation of the DBMS, memory and storage space usage, input/output contention, CPU usage, and application tuning. The extent to which the database administrator can affect each of these areas will vary across DBMS products. Oracle 11g will be used as the exemplar DBMS throughout this section, but it should be noted that each product has its own set of tuning capabilities.

Tuning a database application requires familiarity with the system environment, the DBMS, the application, and the data used by the application. It is here that the skills of even an experienced database administrator are tested. Achieving a quiet environment, one that is reliable and allows users to secure desired information in a timely manner, requires skills and experience that are obtained by working with databases over time. The areas discussed next are quite general and are intended to provide an initial understanding of the scope of activities involved in tuning a database rather than providing the type of detailed understanding necessary to tune a particular database application.

Installation of the DBMS

Correct installation of the DBMS product is essential to any environment. Products often include README files, which may include detailed installation instructions, revisions of procedures, notification of increased disk space needed for installation, and so on. A quick review of any README files may save time during the installation process and result in a better installation. Failing to review general installation instructions may result in default parameter values being set during installation that are not optimal for the situation. Some possible considerations are listed here.

Before beginning installation, the database administrator should ensure that adequate disk space is available. You will need to refer to manuals for the specific

DBMS to be able to translate logical database size parameters (e.g., field length, number of table rows, and estimated growth) into actual physical space requirements. It is possible that the space allocation recommendations are low, as changes made to a DBMS tend to make it larger, but the documentation may not reflect that change. To be safe, allocate at least 20 percent more space than suggested by standard calculations. After installation, review any log files generated during the installation process. Their contents will reveal installation problems that were not noticed or provide assurance that the installation proceeded as expected.

Allocation of disk space for the database should also receive consideration. For example, some UNIX backup systems have trouble with data files that exceed a gigabyte in size. Keeping data files under one gigabyte will avoid possible problems. Allocation of data files in standard sizes will make it easier to balance I/O, because data file locations can be swapped more easily should a bottleneck need to be resolved.

Memory and Storage Space Usage

Efficient usage of main memory involves understanding how the DBMS uses main memory, what buffers are being used, and what needs the programs in main memory have. For example, Oracle has many background processes that reside in memory and handle database management functions when a database is running. Some operating systems require a contiguous chunk of memory to be able to load Oracle, and a system with insufficient memory will have to free up memory space first. Oracle maintains in main memory a data dictionary cache that ideally should be large enough so that at least 90 percent of the requests to the data dictionary can be located in the cache rather than having to retrieve information from disk. Each of these is an example of typical memory management issues that should be considered when tuning a database.

Storage space management may include many activities, some of which have already been discussed in this book, such as denormalization and partitioning. One other activity is **data archiving**. Any database that stores history, such as transaction history or a time series of values for some field, will eventually include obsolete data—data that no longer has any use. Database statistics showing location access frequencies for records or pages can be a clue that data no longer has a purpose. Business rules may also indicate that data older than some value (e.g., seven years) does not need to be kept for active processing. However, there may be legal reasons or infrequently needed business intelligence queries that suggest data should simply not be discarded. Thus, database administrations should develop a program of archiving inactive data. Data may be archived to separate database tables (thus making active tables more compact and, hence, more likely to be more quickly processed) or to files stored outside the database (possibly on magnetic tape or optical storage). Archive files may also be compressed to save space. Methods also need to be developed to restore, in an acceptable time, archived data to the database if and when they are needed. (Remember, archived data are inactive, not totally obsolete.) Archiving reclaims disk space, saves disk storage costs, and may improve database performance by allowing the active data to be stored in less expansive space.

Data archiving
The process of moving inactive data to another storage location where it can be accessed when needed.

Input/Output (I/O) Contention

Database applications are very I/O intensive; a production database will usually both read and write large amounts of data to disk as it works. While CPU clock speeds have increased dramatically, I/O speeds have not increased proportionately, and increasingly complex distributed database systems have further complicated I/O functioning.

Understanding how data are accessed by end users is critical to managing I/O contention. When hot spots (physical disk locations that are accessed repeatedly) develop, understanding the nature of the activity that is causing the hot spot affords the database administrator a much better chance of reducing the I/O contention being experienced. Oracle allows the DBA to control the placement of tablespaces, which contain data files. The DBA's in-depth understanding of user activity facilitates her or

his ability to reduce I/O contention by separating data files that are being accessed together. Where possible, large database objects that will be accessed concurrently may be striped across disks to reduce I/O contention and improve performance. An overall objective of distributing I/O activity evenly across disks and controllers should guide the DBA in tuning I/O.

CPU Usage

Most database operations will require CPU work activity. Because of this, it is important to evaluate CPU usage when tuning a database. Using multiple CPUs allows query processing to be shared when the CPUs are working in parallel, and performance may be dramatically improved. DBAs need to maximize the performance of their existing CPUs while planning for the gains that may be achieved with each new generation of CPUs.

Monitoring CPU load so that typical load throughout a 24-hour period is known provides DBAs with basic information necessary to begin to rebalance CPU loading. The mixture of online and background processing may need to be adjusted for each environment. For example, establishing a rule that all jobs that can be run in off-hours must be run in off-hours will help to unload the machine during peak working hours. Establishing user accounts with limited space will help manage the CPU load also.

Application Tuning

The previous sections have concentrated on activities to tune a DBMS. Examining the applications that end users are using with the database may also increase performance. While normalization to at least 3NF is expected in many organizations that are using relational data models, carefully planned denormalization (see Chapter 5) may improve performance, often by reducing the number of tables that must be joined when running an SQL query.

Examination and modification of the SQL code in an application may also lead to performance improvement. Queries that do full table scans should be avoided, for example, because they are not selective and may not remain in memory very long. This necessitates more retrievals from long-term storage. Multitable joins should be actively managed when possible with the DBMS being used, because the type of join can dramatically affect performance, especially if a join requires a full table join. A general rule of thumb is that any query whose ratio of CPU to I/O time exceeds 13:1 is probably poorly designed. Active monitoring of queries by the DBMS can be used to actually terminate a query of job that exhibits exceeding this ratio. Alternatively, such queries may be put into a "penalty box" to wait until the job scheduler determines that sufficient CPU time is available to continue processing the query.

Similarly, statements containing views and those containing subqueries should be actively reviewed. Tuning of such statements so that components are resolved in the most efficient manner possible may achieve significant performance gains. Chapter 5 discussed a variety of techniques a DBA could use to tune application processing speed and disk space utilization (e.g., re-indexing, overriding automatic query plans, changing data block sizes, reallocating files across storage devices, and guidelines for more efficient query design). A DBA plays an important role in advising programmers and developers which techniques will be the most effective.

The same database activity may take vastly different amounts of time, depending on the workload mix at the time the query or program is run. Some DBMSs have job schedulers that look at statistics about the history of running queries and will schedule batch jobs to achieve a desirable mix of CPU usage and I/O. A DBA can actively monitor query processing times by running so called "heartbeat" or "canary" queries. A **heartbeat query** is a very simple query (possibly SELECT * FROM table WHERE some condition) that a DBA runs many times during the day to monitor variations in processing times. When heartbeat queries are taking extraordinarily long to run, there is probably either an inappropriate mix of jobs running or some inefficient queries are consuming too many DBMS resources. A heartbeat query may also be exactly like certain regularly run user queries for which there are service-level agreements (SLAs) with users on maximum response times. In this case, the heartbeat query is

Heartbeat query
A query submitted by a DBA to test the current performance of a database or to predict the response time for queries that have promised response times. Also called a canary query.

run periodically to make sure that if the user were to submit this query, the SLA goals would be met.

Another aspect of application tuning is setting realistic user expectations. Users should be trained to realize that more complex queries, especially if submitted ad hoc, will take more processing and response time. Users should also be trained to submit queries first using the EXPLAIN or similar function that will not actually run the query but rather estimate the time for query processing from database statistics. This way, many poorly written queries can be avoided. To effectively set realistic user expectations, the DBA needs to realize that database statistics (e.g., number of table rows and distribution of values for certain fields often used for qualifications) must to be recalculated frequently. Recalculation of statistics should occur at least after every batch load of a table, and more frequently for tables that are constantly being updated online. Statistics affect the query optimizer, so reasonable up-to-date statistics are essential for the DBMS to develop a very good query processing plan (i.e., which indexes to use and in which order to execute joins).

The preceding description of potential areas where database performance may be affected should convince you of the importance of effective database management and tuning. As a DBA achieves an in-depth understanding of a DBMS and the applications for which responsibility is assigned, the importance of tuning the database for performance should become apparent. Hopefully this brief section on database tuning will whet your appetite for learning more about one or more database products in order to develop tuning skills.

DATA AVAILABILITY

Ensuring the availability of databases to their users has always been a high-priority responsibility of database administrators. However, the growth of e-business has elevated this charge from an important goal to a business imperative. An e-business must be operational and available to its customers 24/7. Studies have shown that if an online customer does not get the service he or she expects within a few seconds, the customer will take his or her business to a competitor.

Costs of Downtime

The costs of downtime (when databases are unavailable) include several components: lost business during the outage, costs of catching up when service is restored, inventory shrinkage, legal costs, and permanent loss of customer loyalty. These costs are often difficult to estimate accurately and vary widely from one type of business to another. Table 11-2 shows the estimated *hourly* costs of downtime for several business types (Mullins, 2002).

A DBA needs to balance the costs of downtime with the costs of achieving the desired availability level. Unfortunately, it is seldom (if ever) possible to provide 100 percent service levels. Failures may occur (as discussed earlier in this chapter) that may

TABLE 11-2 Cost of Downtime, by Type of Business

Type of Business	Estimated Hourly Cost
Retail brokerage	$6.45 million
Credit card sales authorization	$2.6 million
Home shopping channel	$113,750
Catalog sales centers	$90,000
Airline reservation centers	$89,500
Package shipping service	$28,250
ATM service fees	$14,500

Source: Mullins (2002), p. 226

TABLE 11-3 Cost of Downtime, by Availability

	Downtime Per Year		
Availability	**Minutes**	**Hours**	**Cost Per Year**
99.999%	5	.08	$8,000
99.99%	53	.88	$88,000
99.9%	526	8.77	$877,000
99.5%	2,628	43.8	$4,380,000
99%	5,256	87.6	$8,760,000

Source: Mullins (2002), p. 226

interrupt service. Also, it is necessary to perform periodic database reorganizations or other maintenance activities that may cause service interruptions. It is the responsibility of database administration to minimize the impact of these interruptions. The goal is to provide a high level of availability that balances the various costs involved. Table 11-3 shows several availability levels (stated as percentages) and, for each level, the approximate downtime per year (in minutes and hours). Also shown is the annual cost of downtime for an organization whose hourly cost of downtime is $100,000 (say, a shopping network or online auction). Notice that the annual costs escalate rapidly as the availability declines, yet in the worst case shown in the table the downtime is only 1 percent.

Measures to Ensure Availability

A new generation of hardware, software, and management techniques has been developed (and continues to be developed) to assist database administrators in achieving the high availability levels expected in today's organizations. We have already discussed many of these techniques in this chapter (e.g., database recovery) or in earlier ones (e.g., RAID storage); in this section we provide only a brief summary of potential availability problems and measures for coping with them. A number of other techniques, such as component failure impact analysis (CFIA), fault-tree analysis (FTA), CRAMM, and so on, as well as a wealth of guidance on how to manage availability are described in the IT Infrastructure Library (ITIL) framework (**www.itil-officialsite.com**).

HARDWARE FAILURES Any hardware component, such as a database server, disk subsystem, power supply, or network switch, can become a point of failure that will disrupt service. The usual solution is to provide redundant or standby components that replace a failing system. For example, with clustered servers, the workload of a failing server can be reallocated to another server in the cluster.

LOSS OR CORRUPTION OF DATA Service can be interrupted when data are lost or become inaccurate. Mirrored (or backup) databases are almost always provided in high-availability systems. Also, it is important to use the latest backup and recovery systems (discussed earlier in this chapter).

HUMAN ERROR "Most . . . outages . . . are not caused by the technology, they're caused by people making changes" (Morrow, 2007, p. 32). The use of standard operating procedures, which are mature and repeatable, is a major deterrent to human errors. In addition, training, documentation, and insistence on following internationally recognized standard procedures (see, for example, COBIT [**www.isaca.org/cobit**] or ITIL [**www.itil-officialsite.com**]) are essential for reducing human errors.

MAINTENANCE DOWNTIME Historically, the greatest source of database downtime was attributed to planned database maintenance activities. Databases were taken offline during periods of low activity (nights, weekends) for database reorganization, backup, and other activities. This luxury is no longer available for high-availability applications. New database products are now available that automate maintenance functions. For example, some utilities (called *nondisruptive utilities*) allow routine maintenance to be

performed while the systems remain operational for both read and write operations, without loss of data integrity.

NETWORK-RELATED PROBLEMS High-availability applications nearly always depend on the proper functioning of both internal and external networks. Both hardware and software failures can result in service disruption. However, the Internet has spawned new threats that can also result in interruption of service. For example, a hacker can mount a denial-of-service attack by flooding a Web site with computer-generated messages. To counter these threats, an organization should carefully monitor its traffic volumes and develop a fast-response strategy when there is a sudden spike in activity. An organization also must employ the latest firewalls, routers, and other network technologies.

Summary

The importance of managing data was emphasized in this chapter. The functions of data administration, which takes responsibility for the overall management of data resources, include developing procedures to protect and control data, resolving data ownership and use issues, conceptual data modeling, and developing and maintaining corporate-wide data definitions and standards. The functions of database administration, on the other hand, are those associated with the direct management of a database or databases, including DBMS installation and upgrading, database design issues, and technical issues such as security enforcement, database performance, data availability, and backup and recovery. The data administration and database administration roles are changing in today's business environment, with pressure being exerted to maintain data quality while building high-performing systems quickly.

Threats to data security include accidental losses, theft and fraud, loss of privacy, loss of data integrity, and loss of availability. A comprehensive data security plan will address all of these potential threats, partly through the establishment of views, authorization rules, user-defined procedures, and encryption procedures.

Databases, especially data security, play a key role in an organization's compliance with Sarbanes-Oxley (SOX). SOX audits focus on three key areas: IT change management, logical access to data, and IT operations.

Database recovery and backup procedures are another set of essential database administration activities. Basic recovery facilities that should be in place include backup facilities, journalizing facilities, checkpoint facilities, and a recovery manager. Depending on the type of problem encountered, backward recovery (rollback) or forward recovery (rollforward) may be needed.

The problems of managing concurrent access in multiuser environments must also be addressed. A DBMS must ensure that database transactions possess the ACID properties: atomic, consistent, isolated, and durable. Proper transaction boundaries must be chosen to achieve these properties at an acceptable performance. If concurrency controls on transactions are not established, lost updates may occur, which will cause data integrity to be impaired. Locking mechanisms, including shared and exclusive locks, can be used. Deadlocks may also occur in multiuser environments and may be managed by various means, including using a two-phase locking protocol or other deadlock-resolution mechanism. Versioning is an optimistic approach to concurrency control.

Managing the data dictionary, which is part of the system catalog in most relational database management systems, and the information repository help the DBA maintain high-quality data and high-performing database systems. The establishment of the Information Resource Dictionary System (IRDS) standard has helped with the development of repository information that can be integrated from multiple sources, including the DBMS itself, CASE tools, and software development tools.

Ensuring the availability of databases to users has become a high priority for the modern DBA. Use of batch windows to perform periodic maintenance (e.g., database reorganization) is no longer permissible for mission-critical applications. A new generation of hardware, software, and management techniques is being introduced to assist the DBA in managing data availability.

Effective data administration is not easy, and it encompasses all of the areas summarized here. Increasing emphasis on object-oriented development methods and rapid development are changing the data administration function, but better tools to achieve effective administration and database tuning are becoming available.

Chapter Review

Key Terms

Aborted transaction *490*
After image *485*
Authorization rules *479*
Backup facility *484*
Backward recovery (rollback) *488*
Before image *485*
Checkpoint facility *485*
Concurrency control *492*
Data administration *463*
Data archiving *501*
Data dictionary *498*
Database administration *465*
Database change log *485*
Database destruction *491*
Database recovery *484*
Database security *471*

Deadlock *495*
Deadlock prevention *495*
Deadlock resolution *496*
Encryption *480*
Exclusive lock (X lock, or write lock) *495*
Forward recovery (rollforward) *489*
Heartbeat query *502*
Inconsistent read problem *492*
Information repository *498*
Information Resource Dictionary System (IRDS) *499*
Journalizing facility *485*
Locking *493*
Locking level (lock granularity) *493*
Open source DBMS *470*
Recovery manager *486*
Restore/rerun *487*
Shared lock (S lock, or read lock) *494*
Smart card *482*
System catalog *498*
Transaction *485*
Transaction boundaries *488*
Transaction log *485*
Two-phase locking protocol *496*
User-defined procedures *480*
Versioning *496*

Review Questions

1. Define each of the following terms:
   a. data administration
   b. database administration
   c. two-phase locking protocol
   d. information repository
   e. locking
   f. versioning
   g. deadlock
   h. transaction
   i. encryption
   j. data availability
   k. data archiving
   l. heartbeat query

2. Match the following terms to the appropriate definitions:

____ backup facilities	a. protects data from loss or misuse
____ biometric device	b. reversal of abnormal or aborted transactions
____ checkpoint facility	c. describes all database objects
____ database recovery	d. automatically produces a saved copy of an entire database
____ database security	e. application of after images
____ granularity	f. might analyze your signature
____ recovery manager	g. restoring a database after a loss
____ rollback	h. DBMS module that restores a database after a failure
____ rollforward	i. extent to which a database is locked for transaction
____ system catalog	j. records database state at moment of synchronization

3. Compare and contrast the following terms:
   a. data administration; database administration
   b. repository; data dictionary
   c. deadlock prevention; deadlock resolution
   d. backward recovery; forward recovery
   e. active data dictionary; passive data dictionary
   f. optimistic concurrency control; pessimistic concurrency control
   g. shared lock; exclusive lock
   h. before image; after image
   i. two-phase locking protocol; versioning
   j. authorization; authentication
   k. data backup; data archiving

4. What is an open source DBMS?

5. Indicate whether data administration or database administration is typically responsible for each of the following functions:
   a. Managing the data repository
   b. Installing and upgrading the DBMS
   c. Conceptual data modeling
   d. Managing data security and privacy
   e. Database planning
   f. Tuning database performance
   g. Database backup and recovery
   h. Running heartbeat queries

6. Describe the changing roles of a data administrator and database administrator in the current business environment.

7. List four common problems of ineffective data administration.

8. List four job skills necessary for data administrators. List four job skills necessary for database administrators.

9. Briefly describe four new specialized DBA roles that are emerging today.

10. What changes can be made in data administration at each stage of the traditional database development life cycle to deliver high-quality, robust systems more quickly?

11. List and discuss five areas where threats to data security may occur.

12. Explain how creating a view may increase data security. Also explain why one should not rely completely on using views to enforce data security.

13. List and briefly explain how integrity controls can be used for database security.

14. What is the difference between an authentication scheme and an authorization scheme?

15. What are the key areas of IT that are examined during a Sarbanes-Oxley audit?

16. What are the two key types of security policies and procedures that must be established to aid in Sarbanes-Oxley compliance?

17. What is the advantage of optimistic concurrency control compared with pessimistic concurrency control?

18. What is the difference between shared locks and exclusive locks?

19. What is the difference between deadlock prevention and deadlock resolution?

20. Briefly describe four DBMS facilities that are required for database backup and recovery.

21. What is transaction integrity? Why is it important?

22. List and describe four common types of database failure.

23. Briefly describe four threats to high data availability and at least one measure that can be taken to counter each of these threats.

24. What is an Information Resource Dictionary System (IRDS)?

25. List and briefly explain the ACID properties of a database transaction.

26. Explain the two common forms of encryption.

27. Briefly describe four components of a disaster recovery plan.
28. Explain the purpose of heartbeat queries.
29. How can views be used as part of data security? What are the limitations of views for data security?
30. What is the purpose of the GRANT and REVOKE SQL commands? List some actions that can be granted to or revoked from a user.

Problems and Exercises

1. Fill in the two authorization tables for Pine Valley Furniture Company below, based on the following assumptions (enter Y for yes or N for no):

Authorizations for Inventory Clerks

	Inventory Records	Receivables Records	Payroll Records	Customer Records
Read				
Insert				
Modify				
Delete				

Authorizations for Inventory Records

	Salespersons	A/R Personnel	Inventory Clerks	Carpenters
Read				
Insert				
Modify				
Delete				

- Salespersons, managers, and carpenters may read inventory records but may not perform any other operations on these records.
- Persons in Accounts Receivable and Accounts Payable may read and/or update (insert, modify, delete) receivables records and customer records.
- Inventory clerks may read and/or update (modify, delete) inventory records. They may not view receivables records or payroll records. They may read but not modify customer records.

2. Five recovery techniques are listed below. For each situation described, decide which of the following recovery techniques is most appropriate.
   - Backward recovery
   - Forward recovery (from latest checkpoint)
   - Forward recovery (using backup copy of database)
   - Reprocessing transactions
   - Switch

   a. A phone disconnection occurs while a user is entering a transaction.
   b. A disk drive fails during regular operations.
   c. A lightning storm causes a power failure.
   d. An incorrect amount is entered and posted for a student tuition payment. The error is not discovered for several weeks.
   e. Data entry clerks have entered transactions for two hours after a full database backup when the database becomes corrupted. It is discovered that the journalizing facility of the database has not been activated since the backup was made.

3. Whitlock Department Stores runs a multiuser DBMS on a LAN file server. Unfortunately, at the present time, the DBMS does not enforce concurrency control. One Whitlock customer had a balance due of $250.00 when the following three transactions related to this customer were processed at about the same time:
   - Payment of $250.00
   - Purchase on credit of $100.00
   - Merchandise return (credit) of $50.00

   Each of the three transactions read the customer record when the balance was $250.00 (i.e., before any of the other transactions were completed). The updated customer record was returned to the database in the order shown in the bulleted list above.
   a. What balance will be included for the customer after the last transaction was completed?
   b. What balance should be included for the customer after the three transactions have been processed?
4. For each of the situations described below, indicate which of the following security measures is most appropriate:
   - Authorization rules
   - Encryption
   - Authentication schemes

   a. A national brokerage firm uses an electronic funds transfer (EFT) system to transmit sensitive financial data between locations.
   b. An organization has set up an offsite computer-based training center. The organization wishes to restrict access to the site to authorized employees. Because each employee's use of the center is occasional, the center does not wish to provide the employees with keys to access the center.
   c. A manufacturing firm uses a simple password system to protect its database but finds it needs a more comprehensive system to grant different privileges (e.g., read, versus create or update) to different users.
   d. A university has experienced considerable difficulty with unauthorized users accessing files and databases by appropriating passwords from legitimate users.
5. Metro Marketers, Inc., wants to build a data warehouse for storing customer information that will be used for data marketing purposes. Building the data warehouse will require much more capacity and processing power than they have previously needed, and they are considering Oracle and Red Brick as their database and data warehousing products. As part of their implementation plan, Metro has decided to organize a data administration function. At present, they have four major candidates for the data administrator position:
   a. Monica Lopez, a senior database administrator with five years of experience as an Oracle database administrator managing a financial database for a global banking firm, but no data warehousing experience.
   b. Gerald Bruester, a senior database administrator with six years of experience as an Informix database administrator managing a marketing-oriented database for a *Fortune* 1000 food products firm. Gerald has been to several data

warehousing seminars over the past 12 months and is interested in being involved with a data warehouse.
c. Jim Reedy, currently project manager for Metro Marketers. Jim is very familiar with Metro's current systems environment and is well respected by his coworkers. He has been involved with Metro's current database system but does not have any data warehousing experience.
d. Marie Weber, a data warehouse administrator with two years of experience using a Red Brick–based application that tracks accident information for an automobile insurance company.

Based on this limited information, rank the four candidates for the data administration position. Support your rankings by indicating your reasoning.

6. Referring to Problem and Exercise 5, rank the four candidates for the position of data warehouse administrator at Metro Marketing. Again, support your rankings.
7. Referring to Problem and Exercise 5, rank the four candidates for the position of database administrator at Metro Marketing. Again, support your rankings.
8. What concerns would you have if you accept a job as a database administrator and discover that the database users are entering one common password to log on to the database each morning when they arrive for work? You also learn that they leave their workstations connected to the database all day, even when they are away from their machines for extended periods of time.
9. During the Sarbanes-Oxley audit of a financial services company, you note the following issues. Categorize each of them into the area to which they belong: IT change management, logical access to data, and IT operations.
   a. Five database administrators have access to the sa (system administrator) account that has complete access to the database.
   b. Several changes to database structures did not have appropriate approval by management.
   c. Some users continued to have access to the database even after having been terminated.
   d. Databases are backed up on a regular schedule, using an automated system.
10. Revisit the four issues identified in Problem and Exercise 9. What risk, if any, do each of them pose to the firm?
11. An organization has a database server with three disk devices. The accounting and payroll applications share one of these disk devices and are experiencing performance problems. You have been asked to investigate the problem and tune the databases. What might you suggest to reduce I/O contention?
12. You take a new job as a database administrator at an organization that has a globally distributed database. You are asked to analyze the performance of the database, and as part of your analysis, you discover that all of the processing for regional monthly sales reports is being conducted at the corporate headquarters location. Operations are categorized by five regions: Eastern United States, Western United States, Canada, South America, and Mexico. Data for each region are kept on a server located at the regional headquarters. What would you try to improve the time needed to create the monthly sales reports?
13. An e-business operates a high-volume catalog sales center. Through the use of clustered servers and mirrored disk drives, the data center has been able to achieve data availability of 99.9 percent. Although this exceeds industry norms, the organization still receives periodic customer complaints that the Web site is unavailable (due to data outages). A vendor has proposed several software upgrades as well as expanded disk capacity to improve data availability. The cost of these proposed improvements would be about $25,000 per month. The vendor estimates that the improvements should improve availability to 99.99 percent.
   a. If this company is typical for a catalog sales center, what is the current annual cost of system unavailability? (You will need to refer to Tables 11-2 and 11-3 to answer this question.)
   b. If the vendor's estimates are accurate, can the organization justify the additional expenditure?
14. Review the tables for data availability (Tables 11-2 and 11-3). For the retail brokerage firm shown in Table 11-2, calculate the expected annual cost of downtime for the following availability levels: 99.9 percent and 99.5 percent. Do you think that either of these levels are acceptable for this organization?
15. The mail order firm described in Problem and Exercise 13 has about 1 million customers. The firm is planning a mass mailing of its spring sales catalog to all of its customers. The unit cost of the mailing (postage and catalog) is $6.00. The error rate in the database (duplicate records, erroneous addresses, etc.) is estimated to be 12 percent. Calculate the expected loss of this mailing due to poor-quality data.
16. The average annual revenue per customer for the mail order firm described in Problems and Exercises 13 and 15 is $100. The organization is planning a data quality improvement program that it hopes will increase the average revenue per customer by 5 percent per year. If this estimate proves accurate, what will be the annual increase in revenue due to improved quality?
17. Referring to the Fitchwood Insurance Company case study at the end of Chapter 9, what types of security issues would you expect to encounter when building a data warehouse? Would there be just one set of security concerns related to user access to the data warehouse, or would you also need to be concerned with security of data during the extracting, cleansing, and loading processes?
18. How would Fitchwood's security have to be different if the data mart were made available to customers via the Internet?
19. What security and data quality issues need to be addressed when developing a B2B application using Web services?
20. Research available data quality software. Describe in detail at least one technique employed by one of these tools (e.g., an expert system).
21. Visit some Web sites for open source databases, such as **www.postgresql.org** and **www.mysql.com**. What do you see as major differences in administration between open source databases, such as MySQL, and commercial database products, such as Oracle? How might these differences come into play when choosing a database platform? Summarize the DBA functions of MySQL versus PostgreSQL.
22. Compare the concurrency issues that must be dealt with when developing an OLTP system versus a data warehouse.

Field Exercises

1. Visit an organization that has implemented a database approach. Evaluate each of the following:
   a. The organizational placement of data administration, database administration, and data warehouse administration
   b. The assignment of responsibilities for each of the functions listed in part a
   c. The background and experience of the person chosen as head of data administration
   d. The status and usage of an information repository (passive, active-in-design, active-in-production)
2. Visit an organization that has implemented a database approach and interview an MIS department employee who has been involved in disaster recovery planning. Before you go for the interview, think carefully about the relative probabilities of various disasters for the organization you are visiting. For example, is the area subject to earthquakes, tornadoes, or other natural disasters? What type of damage might the physical plant be subject to? What is the background and training of the employees who must use the system? Find out about the organization's disaster recovery plans and ask specifically about any potential problems you have identified.
3. Visit an organization that has implemented a database approach and interview individuals there about the security measures they take routinely. Evaluate each of the following at the organization:
   a. Database security measures
   b. Network security measures
   c. Operating system security measures
   d. Physical plant security measures
   e. Personnel security measures
4. Identify an organization that handles large, sporadic data loads. For example, organizations that have implemented data warehouses may have large data loads as they populate their data warehouses. Determine what measures the organization has taken to handle these large loads as part of its capacity planning.
5. Databases tend to grow larger over time, not smaller, as new transaction data are added. Interview at least three companies that use databases extensively and identify their criteria and procedures for purging or archiving old data. Find out how often data are purged and what type of data are purged. Identify the data each organization archives and how long those data are archived.
6. Visit an organization that relies heavily on e-commerce applications. Interview the database administrator (or a senior person in that organization) to determine the following:
   a. What is the organizational goal for system availability? (Compare with Table 11-3.)
   b. Has the organization estimated the cost of system downtime ($/hour)? If not, use Table 11-2 and select a cost for a similar type of organization.
   c. What is the greatest obstacle to achieving high data availability for this organization?
   d. What measures has the organization taken to ensure high availability? What measures are planned for the future?
7. Visit an organization that uses an open source DBMS. Why did the organization choose open source software? Does it have other open source software besides a DBMS? Has it purchased any fee-based components or services? Does it have a DA or DBA staff, and, if so, how do these people evaluate the open source DBMS they are using? (This could especially provide insight if the organization also has some traditional DBMS products, such as Oracle or DB2.)

References

Anderson, D. 2005. "HIPAA Security and Compliance," available at **www.tdan.com** (July).

Bernstein, P. A. 1996. "The Repository: A Modern Vision." *Database Programming & Design* 9,12 (December): 28–35.

Celko, J. 1992. "An Introduction to Concurrency Control." *DBMS* 5,9 (September): 70–83.

Descollonges, M. 1993. "Concurrency for Complex Processing." *Database Programming & Design* 6,1 (January): 66–71.

Dowgiallo, E., H. Fosdick, Y. Lirov, A. Langer, T. Quinlan, and C. Young. 1997. "DBA of the Future." *Database Programming & Design* 10,6 (June): 33–41.

Fernandez, E. B., R. C. Summers, and C. Wood. 1981. *Database Security and Integrity*. Reading, MA: Addison-Wesley.

Hall, M. 2003. "MySQL Breaks into the Data Center," available at **www.computerworld.com/printthis/2003/0,4814,85900,00.html**.

Inmon, W. H. 1999. "Data Warehouse Administration." Found at **www.billinmon.com/library/other/dwaadmin.asp** (no longer available).

Inmon, W. H., C. Imhoff, and R. Sousa. 2001. *Corporate Information Factory*, 2nd ed. New York: Wiley.

Lefkovitz, H. C. 1985. *Proposed American National Standards Information Resource Dictionary System*. Wellesley, MA: QED Information Sciences.

Michaelson, J. 2004. "What Every Developer Should Know About Open Source Licensing." *Queue* 2,3 (May): 41–47. (Note: This whole issue of *Queue* is devoted to the open source movement and contains many interesting articles.)

Morrow, J. T. 2007. "The Three Pillars of Data." *InfoWorld* (March 12): 20–33.

Mullins, C. 2001. "Modern Database Administration, Part 1." *DM Review* 11,9 (September): 31, 55–57.

Mullins, C. 2002. *Database Administration: The Complete Guide to Practices and Procedures*. Boston: Addison-Wesley.

Rodgers, U. 1989. "Multiuser DBMS Under UNIX." *Database Programming & Design* 2,10 (October): 30–37.

Further Reading

Loney, K. 2000. "Protecting Your Database." *Oracle Magazine*. 14,3 (May/June): 101–106.

Surran, M. 2003. "Making the Switch to Open Source Software." *THE Journal*. 31,2 (September): 36–41. (This journal is available at **www.thejournal.com**)

Quinlan, T. 1996. "Time to Reengineer the DBA?" *Database Programming & Design* 9,3 (March): 29–34.

Web Resources

http://gost.isi.edu/publications/kerberos-neuman-tso.html A guide to the Kerberos method of user authentication.

www.abanet.org/scitech/ec/isc/dsg-tutorial.html An excellent guide to digital signatures from the American Bar Association Section of Science and Technology, Information Security Committee.

http://tpc.org Web site of the Transaction Processing Performance Council, a nonprofit corporation founded to define transaction processing and database benchmarks and to disseminate objective, verifiable transaction processing performance data to the industry. This is an excellent site for learning more about evaluating DBMSs and database designs through technical articles on database benchmarking.

CASE

Mountain View Community Hospital

Case Description

Refer to the case presented for Mountain View Community Hospital (MVCH) in Chapter 10.

Case Questions

1. Do EMR and CPOE systems seem to have the potential to help MVCH achieve its goals of achieving high-quality care and cost containment? Support your answers with examples of how you think these goals may or may not be achieved.
2. In light of HIPAA and other regulations, securing and protecting patient records is a primary requirement for MVCH.
   a. What data security issues would you expect MVCH to encounter if an EMR system is implemented that is accessible by physicians in the community, by laboratories, and by health-care organizations?
   b. What data security techniques described in this chapter could be used to address these issues?
3. If MVCH decides to implement a CPOE system, how could access problems such as the one that Dr. Z experienced at another hospital be prevented?
4. Given that the MVCH database you developed in SQL Server already includes tables for physicians, orders, and so on, do you think a full-fledged CPOE system could or should be developed internally? Why or why not?
5. Dr. Z indicated that physicians might resist the implementation of a CPOE system. Do you think that would also be true for an EMR system? Why or why not? What would be critical success factors for implementing an electronic medical record at MVCH?
6. Should MVCH adopt a continuous data protection (CDP) system? Why or why not? What other backup strategies might the hospital pursue?
7. Do you think data storage at MVCH should be treated as a strategic issue? Why or why not?
8. Which data and database administration issues described in this chapter should be addressed by MVCH's special study team as part of the long-range business and information systems plan? Why?

Case Exercises

1. List all the possible types of users who would need authorization to use (a) an ERM system and (b) a CPOE system at MVCH. Include user groups external to the hospital that may need to be included.
2. For each user type you listed in Case Exercise 1, indicate what permissions (read, insert, delete, modify) you would grant.
3. Investigate how a hospital such as MVCH could use RFID in connection with an EMR system. How would that affect data storage requirements?
4. In light of HIPAA's security rules (data backup, access to data, data retention, etc.) and the tremendous growth of data at MVCH, outline the pros and cons of various data storage options that the hospital may be using. Are there storage media that can potentially lead to violations under HIPAA? Which ones? Why?
5. Access HIPAA's security requirements online and outline a contingency plan for MVCH.

Project Assignments

P1. Password protect the MVCH database you created in SQL Server (or other database management systems required by your instructor).

P2. Create a matrix to indicate the permissions (read, insert, delete, modify) you would grant to different users of the database you identify.

P3. Create at least two different users and implement their permissions using SQL statements.