Chapter 18 Security

Chapter Objectives

In this chapter you will learn:

- The scope of database security.
- Why database security is a serious concern for an organization.
- The types of threat that can affect a database system.
- How to protect a computer system using computer-based controls.
- The security measures provided by Microsoft Access and Oracle DBMSs.
- Approaches for securing a DBMS on the Web.

Data is a valuable resource that must be strictly controlled and managed, as with any corporate resource. Part or all of the corporate data may have strategic importance to an organization and should therefore be kept secure and confidential.

In Chapter 2 we discussed the database environment and, in particular, the typical functions and services of a Database Management System (DBMS). These functions and services include authorization services, such that a DBMS must furnish a mechanism to ensure that only authorized users can access the database. In other words, the DBMS must ensure that the database is secure. The term **security** refers to the protection of the database against unauthorized access, either intentional or accidental. Besides the services provided by the DBMS, discussions on database security could also include broader issues associated with securing the database and its environment. However, these issues are outwith the scope of this book and the interested reader is referred to Pfleeger (1997).

Structure of this Chapter

In Section 18.1 we discuss the scope of database security and examine the types of threat that may affect computer systems in general. In Section 18.2 we consider the range of computer-based controls that are available as countermeasures to these threats. In Sections 18.3 and 18.4 we describe the security measures provided by Microsoft Access 2000 DBMS and Oracle 8/8i DBMS. In Section 18.5 we identify the security measures associated with DBMSs and the Web. The examples used throughout this chapter are taken from the *DreamHome* case study described in Section 10.4 and Appendix A.

18.1 Database Security

In this section we describe the scope of database security and discuss why organizations must take potential threats to their computer systems seriously. We also identify the range of threats and their consequences on computer systems.

Database security	The mechanisms that protect the database against intentional or accidental threats.

Security considerations apply not only to the data held in a database: breaches of security may affect other parts of the system, which may in turn affect the database. Consequently, database security encompasses hardware, software, people, and data. To effectively implement security requires appropriate controls, which are defined in specific mission objectives for the system. This need for security, while often having been neglected or overlooked in the past, is now increasingly recognized by organizations. The reason for this turnaround is the increasing amounts of crucial corporate data being stored on computer and the acceptance that any loss or unavailability of this data could prove to be disastrous.

A database represents an essential corporate resource that should be properly secured using appropriate controls. We consider database security in relation to the following situations:

- theft and fraud;
- loss of confidentiality (secrecy);
- loss of privacy;
- loss of integrity;
- loss of availability.

These situations broadly represent areas in which the organization should seek to reduce risk, that is the possibility of incurring loss or damage. In some situations, these areas are closely related such that an activity that leads to loss in one area may also lead to loss in another. In addition, events such as fraud or loss of privacy may arise because of either intentional or unintentional acts, and do not necessarily result in any detectable changes to the database or the computer system.

Theft and fraud affect not only the database environment but also the entire organization. As it is people who perpetrate such activities, attention should focus on reducing the opportunities for this occurring. Theft and fraud do not necessarily alter data, as is the case for activities that result in either loss of confidentiality or loss of privacy.

Confidentiality refers to the need to maintain secrecy over data, usually only that which is critical to the organization, whereas privacy refers to the need to protect data about individuals. Breaches of security resulting in loss of confidentiality could, for instance, lead to loss of competitiveness, and loss of privacy could lead to legal action being taken against the organization.

Loss of data integrity results in invalid or corrupted data, which may seriously affect the operation of an organization. Many organizations are now seeking virtually continuous operation, the so-called 24 × 7 availability (that is, 24 hours a day, seven days a week). Loss of availability means that the data, or the system, or both cannot be accessed, which can seriously affect an organization's financial performance. In some cases, events that cause a system to be unavailable may also cause data corruption.

Database security aims to minimize losses caused by anticipated events in a cost-effective manner without unduly constraining the users. In recent times, computer-based criminal activities have significantly increased and are forecast to continue to rise over the next few years.

18.1.1 Threats

Threat Any situation or event, whether intentional or accidental, that may adversely affect a system and consequently the organization.

A threat may be caused by a situation or event involving a person, action, or circumstance that is likely to bring harm to an organization. The harm may be tangible, such as loss of hardware, software, or data, or intangible, such as loss of credibility or client confidence. The problem facing any organization is to identify all possible threats. Therefore, as a minimum an organization should invest time and effort in identifying the most serious threats.

In the previous section we identified areas of loss that may result from intentional or unintentional activities. While some types of threat can be either intentional or unintentional, the impact remains the same. Intentional threats involve people and may be perpetrated by both authorized users and unauthorized users, some of whom may be external to the organization.

Any threat must be viewed as a potential breach of security which, if successful, will have a certain impact. Table 18.1 presents examples of various types of threat, listed under the area on which they may have an impact. For example, 'viewing and disclosing unauthorized data' as a threat may result in theft and fraud, loss of confidentiality, and loss of privacy for the organization.

Table 18.1 Examples of threats.

Threat	Theft and fraud	Loss of confidentiality	Loss of privacy	Loss of integrity	Loss of availability
Using another person's means of access	✓	✓	✓		
Unauthorized amendment or copying of data	✓			✓	
Program alteration	✓			✓	✓
Inadequate policies and procedures that allow a mix of confidential and normal output	✓	✓	✓		
Wire tapping	✓	✓	✓		
Illegal entry by hacker	✓	✓	✓		
Blackmail	✓	✓	✓		
Creating 'trapdoor' into system	✓	✓	✓		
Theft of data, programs, and equipment	✓	✓	✓		✓
Failure of security mechanisms, giving greater access than normal		✓	✓	✓	
Staff shortages or strikes				✓	✓
Inadequate staff training		✓	✓	✓	✓
Viewing and disclosing unauthorized data	✓	✓	✓		
Electronic interference and radiation				✓	✓
Data corruption owing to power loss or surge				✓	✓
Fire (electrical fault, lightning strike, arson), flood, bomb				✓	✓
Physical damage to equipment				✓	✓
Breaking cables or disconnection of cables				✓	✓
Introduction of viruses				✓	✓

The extent that an organization suffers as a result of a threat's succeeding depends upon a number of factors, such as the existence of countermeasures and contingency plans. For example, if a hardware failure occurs corrupting secondary storage, all processing activity must cease until the problem is resolved. The recovery will depend upon a number of factors, which include when the last backups were taken and the time needed to restore the system.

An organization needs to identify the types of threat it may be subjected to and initiate appropriate plans and countermeasures, bearing in mind the costs of implementing them. Obviously, it may not be cost-effective to spend considerable time, effort, and money on potential threats that may result only in minor inconvenience. The organization's business may also influence the types of threat that should be considered, some of which may be rare. However, rare events should be taken into account, particularly if their impact would be significant. A summary of the potential threats to computer systems is represented in Figure 18.1.

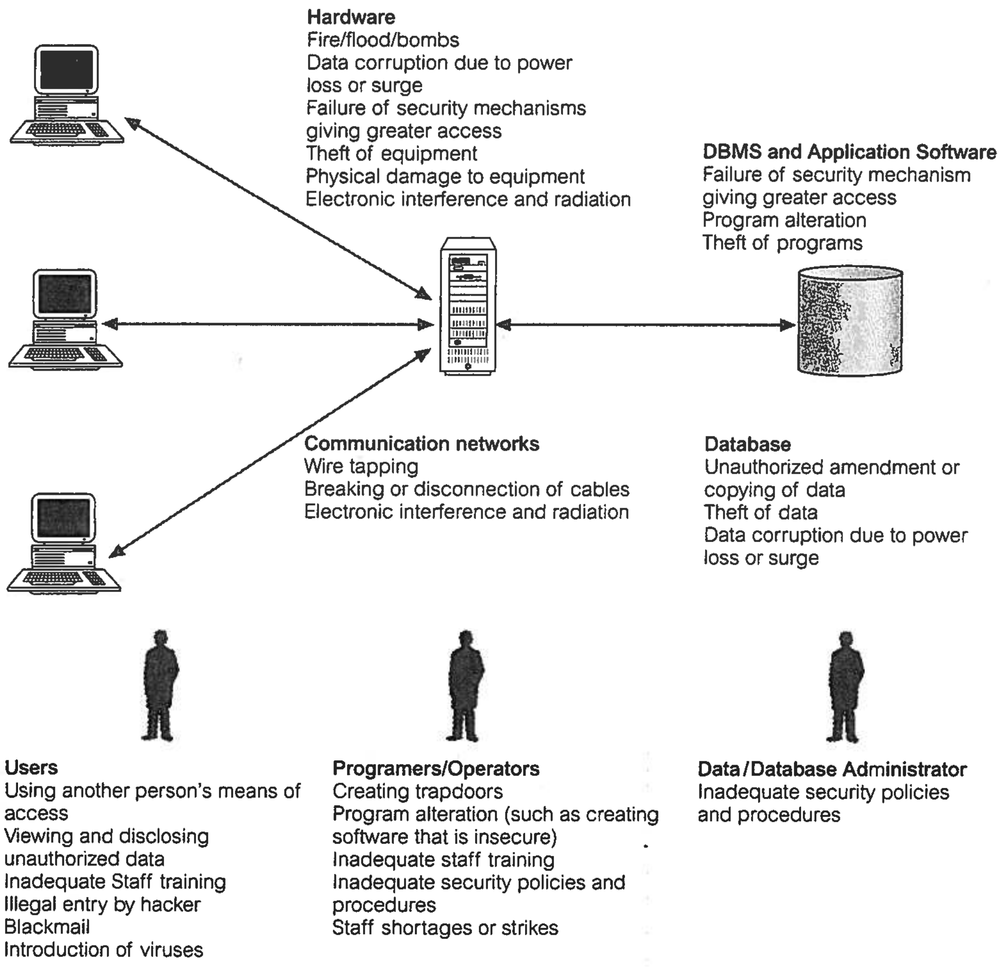

Hardware
Fire/flood/bombs
Data corruption due to power loss or surge
Failure of security mechanisms giving greater access
Theft of equipment
Physical damage to equipment
Electronic interference and radiation
DBMS and Application Software
Failure of security mechanism giving greater access
Program alteration
Theft of programs
Communication networks
Wire tapping
Breaking or disconnection of cables
Electronic interference and radiation
Database
Unauthorized amendment or copying of data
Theft of data
Data corruption due to power loss or surge
Users
Using another person's means of access
Viewing and disclosing unauthorized data
Inadequate Staff training
Illegal entry by hacker
Blackmail
Introduction of viruses
Programers/Operators
Creating trapdoors
Program alteration (such as creating software that is insecure)
Inadequate staff training
Inadequate security policies and procedures
Staff shortages or strikes
Data/Database Administrator
Inadequate security policies and procedures

Figure 18.1 Summary of potential threats to computer systems.

Countermeasures – Computer-Based Controls **18.2**

The types of countermeasure to threats on computer systems range from physical controls to administrative procedures. Despite the range of computer-based controls that are available, it is worth noting that, generally, the security of a DBMS is only as good as that of the operating system, owing to their close association. Representation of a typical multi-user computer environment is shown in Figure 18.2. In this section we focus on the following computer-based security controls for a multi-user environment (some of which may not be available in the PC environment):

T

- authorization
- views
- backup and recovery
- integrity
- encryption
- RAID technology.

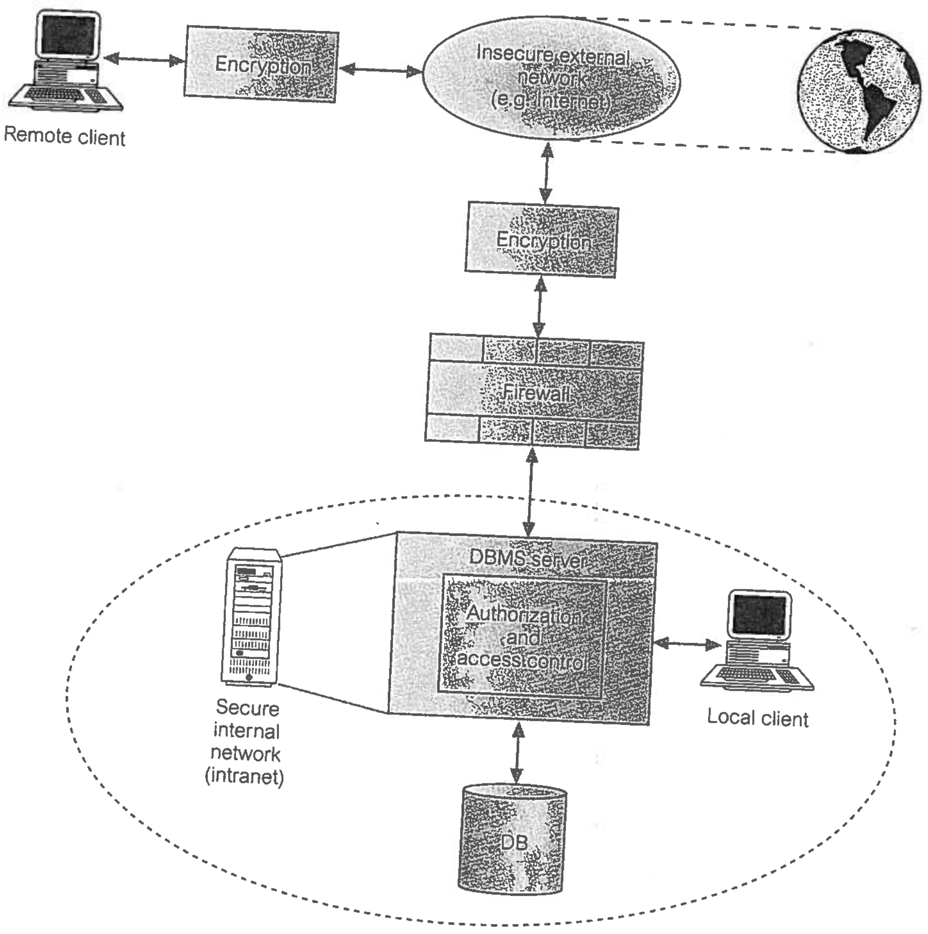

Remote client
Encryption
Insecure external network (e.g. Internet)
Encryption
Firewall
DBMS server
Authorization and access control
Local client
Secure internal network (intranet)
DB

Figure 18.2 Representation of a typical multi-user computer environment.

18.2.1 Authorization

Authorization The granting of a right or privilege that enables a subject to have legitimate access to a system or a system's object.

Authorization controls can be built into the software, and govern not only what system or object a specified user can access, but also what the user may do with it. For this reason, authorization controls are sometimes referred to as **access controls**. The process of authorization involves authentication of subjects requesting access to objects, where 'subject' represents a user or program and 'object' represents a database table, view, procedure, trigger, or any other object that can be created within the system.

Authentication A mechanism that determines whether a user is who he or she claims to be.

A system administrator is usually responsible for allowing users to have access to a computer system by creating individual user accounts. Each user is given a unique identifier, which is used by the operating system to determine who they are. Associated with each identifier is a password, chosen by the user and known to the operating system, which must be supplied to enable the operating system to verify (or authenticate) who the user claims to be.

This procedure allows authorized use of a computer system but does not necessarily authorize access to the DBMS or any associated application programs. A separate, similar procedure may have to be undertaken to give a user the right to use the DBMS. The responsibility to authorize use of the DBMS usually rests with the Database Administrator (DBA), who must also set up individual user accounts and passwords using the DBMS itself.

Some DBMSs maintain a list of valid user identifiers and associated passwords, which can be distinct from the operating system's list. However, other DBMSs maintain a list whose entries are validated against the operating system's list based on the current user's login identifier. This prevents a user from logging on to the DBMS with one name, having already logged on to the operating system using a different name.

Privileges

Once a user is given permission to use a DBMS, various other privileges may also be automatically associated with it. For example, privileges may include the right to access or create certain database objects such as relations, views, and indexes, or to run various DBMS utilities. Privileges are granted to users to accomplish the tasks required for their jobs. As excessive granting of unnecessary privileges can compromise security, a privilege should only be granted to a user who absolutely requires the privilege to accomplish his or her work.

Some DBMSs operate as **closed systems** so that while users may be authorized to access the DBMS, they require authorization to access specific objects. Either the DBA or owners of particular objects provide this authorization. On the other hand, an **open system** allows users to have complete access to all objects within the database. In this case, privileges have to be explicitly removed from users to control access. We discussed authorization and privileges using SQL in Section 6.6.

Table 18.2 User and group identifiers.

User identifier	Type	Group	Member identifier
SG37	User	Sales	SG37
SG14	User	Sales	SG14
SG5	User		
Sales	Group		

Ownership and privileges

Some objects in the DBMS are owned by the DBMS itself, usually in the form of a specific superuser, such as the DBA. Accordingly, ownership of objects gives the owner all appropriate privileges on the objects owned. The same situation applies to other authorized users if they own objects. The creator of an object owns the object and can assign appropriate privileges for the object. For example, although a user owns a view, he or she may be authorized only to query the view. This may happen when the user is authorized only to query the underlining base relation. These privileges can be passed on to other authorized users. For example, an owner of several relations may authorize other users to query the relations but not to carry out any updates. In SQL, whenever a user passes on a privilege, he or she can indicate whether the recipient can pass the privilege on.

Where a DBMS supports several different types of authorization identifier, there may be different priorities associated with each type. For example, a DBMS may permit both individual user identifiers and group identifiers to be created, with the user identifier having a higher priority than the group identifier. For such a DBMS, user and group identifiers may be defined as shown in Table 18.2.

The columns with headings *User identifier* and *Type* list each user on the system together with the user type, which distinguishes individuals from groups. The columns with headings *Group* and *Member identifier* list each group and the user members of that group. Certain privileges may be associated with specific identifiers, which indicate what kind of privilege (such as Select, Update, Insert, Delete, or All) is allowed with certain database objects. Each privilege has a binary value associated with it, for example:

SELECT	**UPDATE**	**INSERT**	**DELETE**	**ALL**
0001	0010	0100	1000	1111

The binary values are summed, as appropriate, and the total value indicates what privileges, if any, are allowed for a specific user or group with a particular object. Table 18.3 is an example of an **access control matrix**, which illustrates different privileges for the Sales group and for users SG37 and SG5.

The matrix indicates that the group identifier Sales has only the Select privilege (0001) for the propertyNo, type, and price attributes, and a limit of 15 rows for any query result set. As user SG14 (David Ford) is a member of this group and has no additional privileges of his own, these are also the restrictions that apply to him. On the other hand, user SG37 (Ann Beech) has Select and Insert privileges (shown as 0001 + 0100 = 0101) for the

Table 18.3 Access control matrix.

User identifier	propertyNo	type	price	ownerNo	staffNo	branchNo	Query row limit
Sales	0001	0001	0001	0000	0000	0000	15
SG37	0101	0101	0111	0101	0111	0000	100
SG5	1111	1111	1111	1111	1111	1111	none

propertyNo, type, and ownerNo attributes and Select, Update, and Insert privileges (shown as 0001 + 0010 + 0100 = 0111) for the price and staffNo attributes, with a limit of 100 rows for any query result set. Finally, user SG5 (Susan Brand) has Select, Update, Insert, and Delete privileges (shown as 0001 + 0010 + 0100 + 1000 = 1111), in other words All privileges for all attributes, with no limit set on the number of rows for any query result set.

DBMSs use similar matrices to implement access control, although the precise details of implementation vary from one system to another. On some DBMSs, a user has to tell the system under which identifier he or she is operating, especially if the user is a member of more than one group. It is essential to become familiar with the available authorization and other control mechanisms provided by the DBMS, particularly where priorities may be applied to different authorization identifiers and where privileges can be passed on. This will enable the correct types of privileges to be granted to users based on their requirements and those of the application programs that many of them will use.

Views (Subschemas) 18.2.2

View A view is the dynamic result of one or more relational operations operating on the base relations to produce another relation. A view is a *virtual relation* that does not actually exist in the database, but is produced upon request by a particular user, at the time of request.

The view mechanism provides a powerful and flexible security mechanism by hiding parts of the database from certain users. The user is not aware of the existence of any attributes or rows that are missing from the view. A view can be defined over several relations with a user being granted the appropriate privilege to use it, but not to use the base relations. In this way, using a view is more restrictive than simply having certain privileges granted to a user on the base relation(s). We discussed views in detail in Sections 3.4 and 6.4.

Backup and Recovery 18.2.3

Backup The process of periodically taking a copy of the database and log file (and possibly programs) on to offline storage media.

A DBMS should provide backup facilities to assist with the recovery of a database following failure. It is always advisable to make backup copies of the database and log file at regular intervals and to ensure that the copies are in a secure location. In the event of a failure that renders the database unusable, the backup copy and the details captured in the log file are used to restore the database to the latest possible consistent state. A description of how a log file is used to restore a database is described in more detail in Section 19.3.3.

Journaling The process of keeping and maintaining a log file (or journal) of all changes made to the database to enable recovery to be undertaken effectively in the event of a failure.

A DBMS should provide logging facilities, sometimes referred to as journaling, which keep track of the current state of transactions and database changes, to provide support for recovery procedures. The advantage of journaling is that, in the event of a failure, the database can be recovered to its last known consistent state using a backup copy of the database and the information contained in the log file. If no journaling is enabled on a failed system, the only means of recovery is to restore the database using the latest backup version of the database. However, without a log file, any changes made after the last backup to the database will be lost. The process of journaling is discussed in more detail in Section 19.3.3.

18.2.4 Integrity

Integrity constraints also contribute to maintaining a secure database system by preventing data from becoming invalid, and hence giving misleading or incorrect results. Integrity constraints were discussed in detail in Section 3.3.

18.2.5 Encryption

Encryption The encoding of the data by a special algorithm that renders the data unreadable by any program without the decryption key.

If a database system holds particularly sensitive data, it may be deemed necessary to encode it as a precaution against possible external threats or attempts to access it. Some DBMSs provide an encryption facility for this purpose. The DBMS can access the data (after decoding it), although there is a degradation in performance because of the time taken to decode it. Encryption also protects data transmitted over communication lines. There are a number of techniques for encoding data to conceal the information; some are termed 'irreversible' and others 'reversible'. Irreversible techniques, as the name implies, do not permit the original data to be known. However, the data can be used to obtain valid statistical information. Reversible techniques are more commonly used. To transmit data securely over insecure networks requires the use of a **cryptosystem**, which includes:

- an *encryption key* to encrypt the data (plaintext);
- an *encryption algorithm* that, with the encryption key, transforms the plaintext into *ciphertext*;
- a *decryption key* to decrypt the ciphertext;
- a *decryption algorithm* that, with the decryption key, transforms the ciphertext back into plaintext.

One technique, called **symmetric encryption**, uses the same key for both encryption and decryption and relies on safe communication lines for exchanging the key. However, most users do not have access to a secure communication line and, to be really secure, the keys need to be as long as the message (Leiss, 1982). However, most working systems are based on user keys shorter than the message. One scheme used for encryption is the **Data Encryption Standard (DES)**, which is a standard encryption algorithm developed by IBM. This scheme uses one key for both encryption and decryption, which must be kept secret, although the algorithm need not be. The algorithm transforms each 64-bit block of plaintext using a 56-bit key. The DES is not universally regarded as being very secure, and some authors maintain that a larger key is required. For example, a scheme called PGP (Pretty Good Privacy) uses a 128-bit symmetric algorithm for bulk encryption of the data it sends.

Keys with 64 bits are now probably breakable by major governments with special hardware, albeit at substantial cost. However, this technology will be within the reach of organized criminals, major organizations, and smaller governments in a few years. While it is envisaged that keys with 80 bits will also become breakable in the future, it is probable that keys with 128 bits will remain unbeakable for the foreseeable future. The terms 'strong authentication' and 'weak authentication' are sometimes used to distinguish between algorithms that, to all intents and purposes, cannot be broken with existing technologies and knowledge (strong) from those that can be (weak).

Another type of cryptosystem uses different keys for encryption and decryption, and is referred to as **asymmetric encryption**. One example is **public key** cryptosystems, which use two keys, one of which is public and the other private. The encryption algorithm may also be public, so that anyone wishing to send a user a message can use the user's publicly known key in conjunction with the algorithm to encrypt it. Only the owner of the private key can then decipher the message. Public key cryptosystems can also be used to send a 'digital signature' with a message and prove that the message came from the person who claimed to have sent it. The most well known asymmetric encryption is **RSA** (the name is derived from the initials of the three designers of the algorithm).

Generally, symmetric algorithms are much faster to execute on a computer than those that are asymmetric. However, in practice, they are often used together, so that a public key algorithm is used to encrypt a randomly generated encryption key, and the random key is used to encrypt the actual message using a symmetric algorithm. We discuss encryption in the context of the Web in Section 18.5.

18.2.6 RAID (Redundant Array of Independent Disks)

The hardware that the DBMS is running on must be *fault-tolerant*, meaning that the DBMS should continue to operate even if one of the hardware components fails. This suggests having redundant components that can be seamlessly integrated into the working system whenever there is one or more component failures. The main hardware components that should be fault-tolerant include disk drives, disk controllers, CPU, power supplies, and cooling fans. Disk drives are the most vulnerable components with the shortest times between failure of any of the hardware components.

One solution is the use of **RAID** technology. RAID originally stood for *Redundant Array of Inexpensive Disks*, but more recently the 'I' in RAID has come to stand for *Independent*. RAID works on having a large disk array comprising an arrangement of several independent disks that are organized to improve reliability and at the same time increase performance.

Performance is increased through *data striping*: the data is segmented into equal-size partitions (the *striping unit*) which are transparently distributed across multiple disks. This gives the appearance of a single large, fast disk where in actual fact the data is distributed across several smaller disks. Striping improves overall I/O performance by allowing multiple I/Os to be serviced in parallel. At the same time, data striping also balances the load among disks.

Reliability is improved through storing redundant information across the disks using a *parity* scheme or an *error-correcting* scheme, such as Reed-Solomon codes (see, for example, Pless, 1989). In a parity scheme, each byte may have a parity bit associated with it that records whether the number of bits in the byte that are set to 1 is even or odd. If the number of bits in the byte becomes corrupted, the new parity of the byte will not match the stored parity. Similarly, if the stored parity bit becomes corrupted, it will not match the data in the byte. Error-correcting schemes store two or more additional bits, and can reconstruct the original data if a single bit becomes corrupt. These schemes can be used through striping bytes across disks.

There are a number of different disk configurations with RAID, termed RAID *levels*. The RAID levels are as follows:

- RAID 0 – Nonredundant This level maintains no redundant data and so has the best write performance since updates do not have to be replicated. Data striping is performed at the level of blocks.
- RAID 1 – Mirrored This level maintains (*mirrors*) two identical copies of the data across different disks. To maintain consistency in the presence of disk failure, writes may not be performed simultaneously. This is the most expensive storage solution.
- RAID 0+1 – Nonredundant and Mirrored This level combines striping and mirroring.
- RAID 2 – Memory-Style Error-Correcting Codes With this level, the striping unit is a single bit and Hamming codes are used as the redundancy scheme.
- RAID 3 – Bit-Interleaved Parity This level provides redundancy by storing parity information on a single disk in the array. This parity information can be used to recover the data on other disks should they fail. This level uses less storage space than RAID 1 but the parity disk can become a bottleneck.

- RAID 4 – Block-Interleaved Parity With this level, the striping unit is a disk block – a parity block is maintained on a separate disk for corresponding blocks from a number of other disks. If one of the disks fails, the parity block can be used with the corresponding blocks from the other disks to restore the blocks of the failed disk.
- RAID 5 – Block-Interleaved Distributed Parity This level uses parity data for redundancy in a similar way to RAID 3 but stripes the parity data across all the disks, similar to the way in which the source data is striped. This alleviates the bottleneck on the parity disk.
- RAID 6 – P+Q Redundancy This level is similar to RAID 5 but additional redundant data is maintained to protect against multiple disk failures. Error-correcting codes are used instead of using parity.

Oracle, for example, recommends use of RAID 1 for the redo log files. For the database files, Oracle recommends either RAID 5, provided the write overhead is acceptable, otherwise Oracle recommends either RAID 1 or RAID 0+1. A fuller discussion of RAID is outwith the scope of this book and the interested reader is referred to the papers by Chen and Patterson (1990) and Chen *et al.* (1994).

18.3 Security in Microsoft Access DBMS

In Section 8.1 we provided an overview of Microsoft Access 2000 DBMS. In this section we focus on the security measures provided by Access. In Section 6.6 we described the SQL GRANT and REVOKE statements; Microsoft Access 2000 does not support these statements but instead provides the following two methods for securing a database:

- setting a password for opening a database (referred to as *system security* by Microsoft Access);
- user-level security, which can be used to limit the parts of the database that a user can read or update (referred to as *data security* by Microsoft Access).

In this section we briefly discuss how Microsoft Access provides these two types of security mechanism.

Setting a Password

The simpler security method is to set a password for opening the database. Once a password has been set (from the **Tools, Security** menu), a dialog box requesting the password will be displayed whenever the database is opened. Only users who type the correct password will be allowed to open the database. This method is secure as Microsoft Access encrypts the password so that it cannot be accessed by reading the database file directly. However, once a database is open, all the objects contained within the database

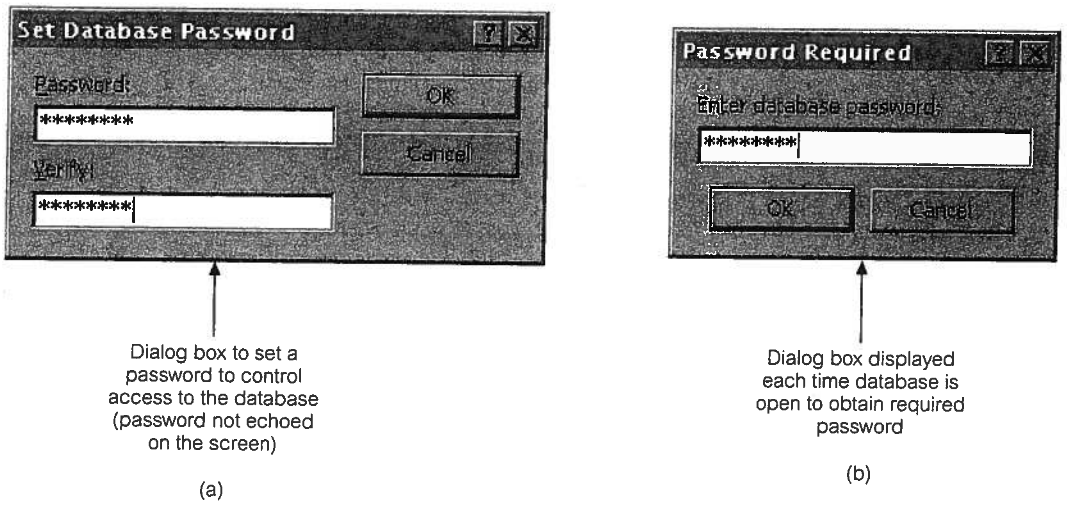

Set Database Password
Password:
****
Verify:
****
OK
Cancel
Dialog box to set a password to control access to the database (password not echoed on the screen)
(a)
Password Required
Enter database password:
****
OK
Cancel
Dialog box displayed each time database is open to obtain required password
(b)

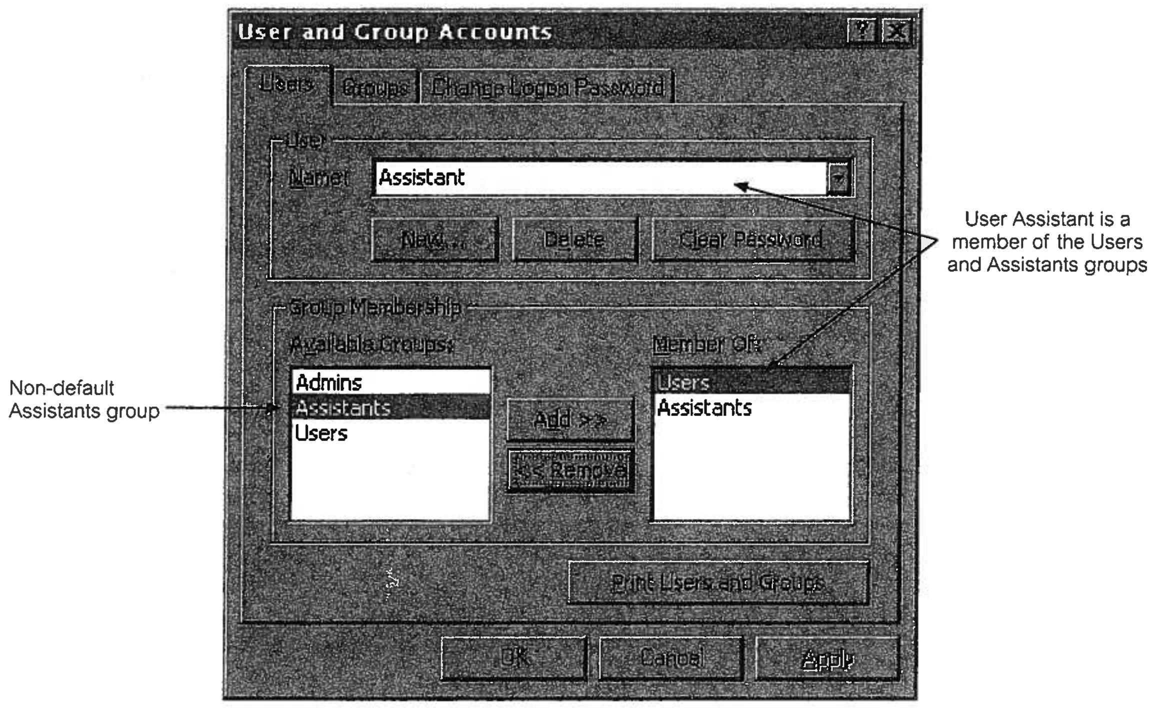
Figure 18.3 Securing the *DreamHome* database using a password: (a) the Set Database Password dialog box; (b) the Password Required dialog box shown at startup.

are available to the user. Figure 18.3(a) shows the dialog box to set the password and Figure 18.3(b) shows the dialog box requesting the password whenever the database is opened.

User-Level Security

User-level security in Microsoft Access is similar to methods used in most network systems. Users are required to identify themselves and type a password when they start Microsoft Access. Within the Microsoft Access *workgroup information file*, users are identified as members of a **group**. Access provides two default groups: administrators (*Admins* group) and users (*Users* group), but additional groups can be defined. Figure 18.4 displays the dialog box used to define the security level for user and group accounts. It shows a non-default group called Assistants, and a user called Assistant who is a member of the Users and Assistants groups.

Permissions are granted to groups and users to regulate how they are allowed to work with each object in the database using the User and Group Permissions dialog box. Table 18.4 shows the permissions that can be set in Microsoft Access. For example, Figure 18.5 shows the dialog box for a user called Assistant who has only read access to a stored query called Staff1_View. In a similar way, all access to the base table Staff would be removed so that the Assistant user could only view the data in the Staff table using this view.

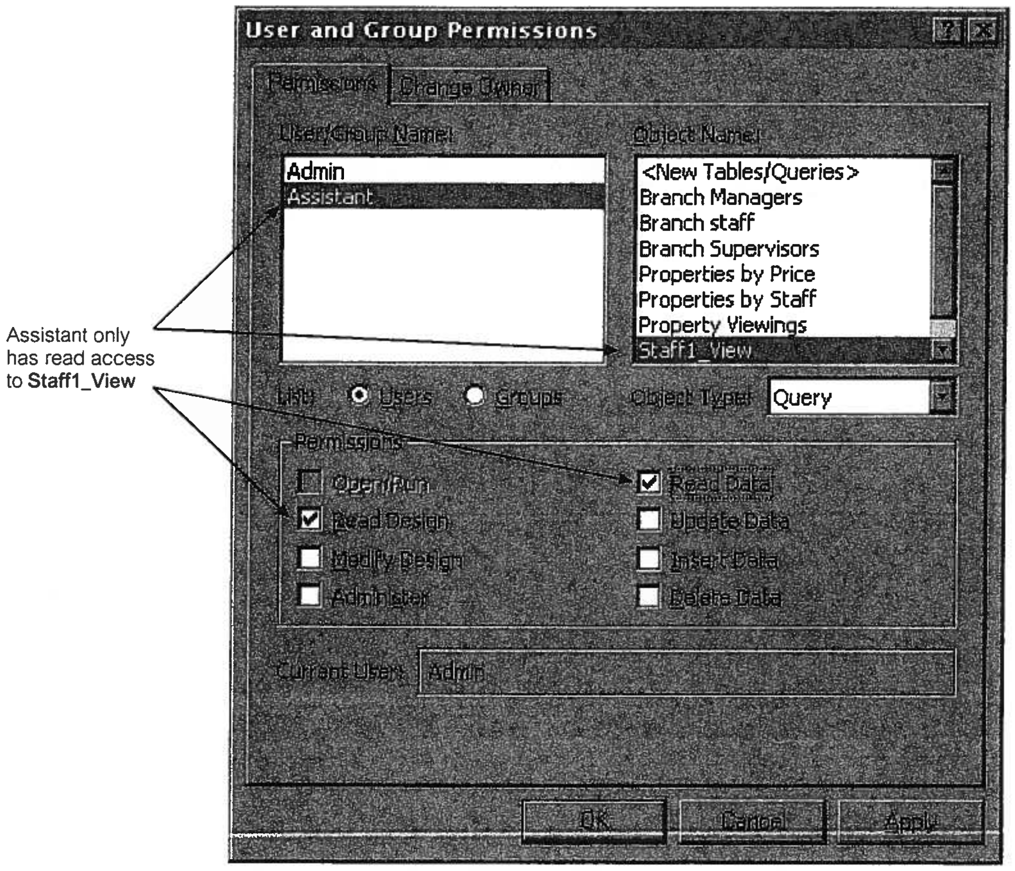


Name: Assistant

Figure 18.4 The User and Group Accounts dialog box for the *DreamHome* database.

Table 18.4 Microsoft Access permissions.

Permission	Description
Open/Run	Open a database, form, report, or run a macro
Open Exclusive	Open a database with exclusive access
Read Design	View objects in Design view
Modify Design	View and change database objects, and delete them
Administer	For databases, set database password, replicate database, and change startup properties
	Full access to database objects including ability to assign permissions
Read Data	View data
Update Data	View and modify data (but not insert or delete data)
Insert Data	View and insert data (but not update or delete data)
Delete Data	View and delete data (but not insert or update data)



User and Group Permissions

Permissions | Change Owner

User/Group Name:
Admin
Assistant

Object Name:
<New Tables/Queries>
Branch Managers
Branch staff
Branch Supervisors
Properties by Price
Properties by Staff
Property Viewings
Staff1_View

List: [✓ Users] Groups

Object Type: Query

Permissions
- ☐ Open/Run
- ☑ Read Design
- ☐ Modify Design
- ☐ Administer
- ☑ Read Data
- ☐ Update Data
- ☐ Insert Data
- ☐ Delete Data

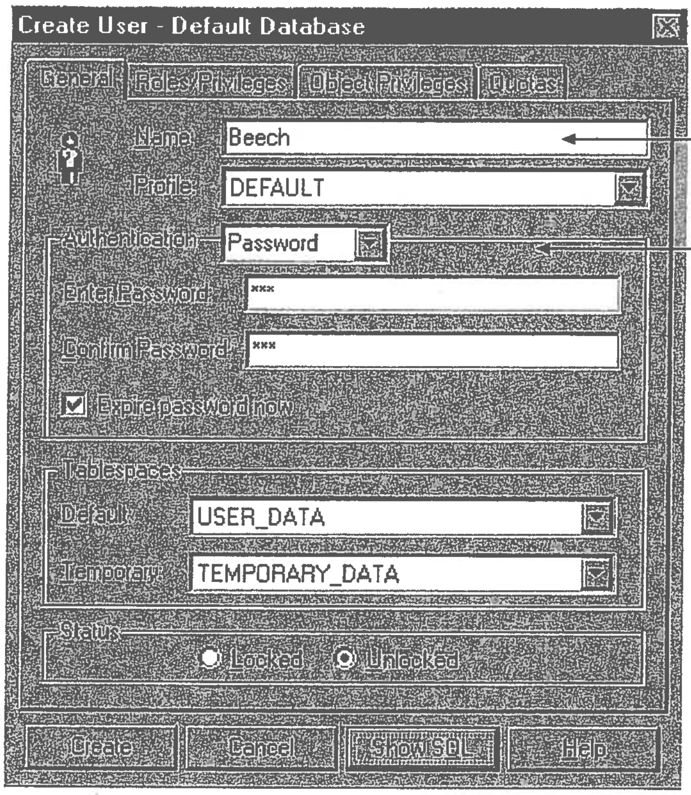
Current User: Admin

OK | Cancel | Apply

Figure 18.5 User and Group Permissions dialog box showing the Assistant user has only read access to the Staff1_View query.

18.4 Security in Oracle DBMS

In Section 8.2 we provided an overview of Oracle 8/8i DBMS. In this section, we focus on the security measures provided by Oracle. In the previous section we examined two types of security in Microsoft Access: system security and data security. In this section we examine how Oracle provides these two types of security. As with Access, one form of system security used by Oracle is the standard user name and password mechanism, whereby a user has to provide a valid user name and password before access can be gained to the database, although the responsibility to authenticate users can be devolved to the operating system. Figure 18.6 illustrates the creation of a new user called Beech with password authentication set. Whenever user Beech tries to connect to the database, this user will be presented with a Connect or Log On dialog box similar to the one illustrated in Figure 18.7, prompting for a user name and password to access the specified database.


Create User - Default Database

General | Roles/Privileges | Object Privileges | Quotas

Name: Beech

Profile: DEFAULT

Authentication: Password

Enter Password: ***

Confirm Password: ***

[✓] Expire password now

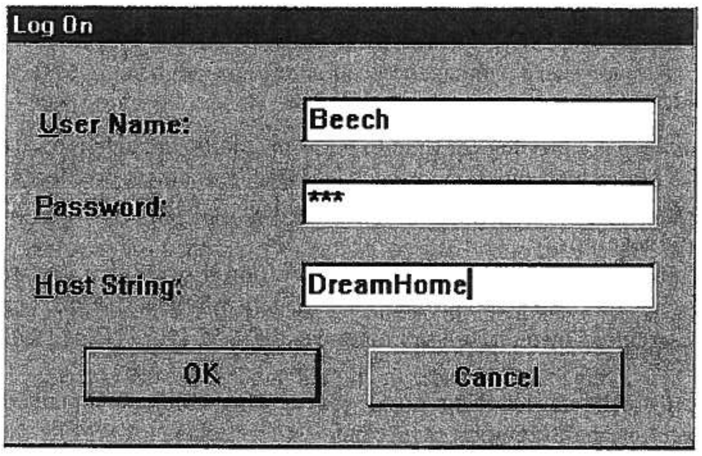
Tablespaces

Default: USER_DATA

Temporary: TEMPORARY_DATA

Status: ( ) Locked [✓ Unlocked]

Figure 18.6 Creation of a new user called Beech with password authentication set.


Log On

User Name: Beech

Password: ***

Host String: DreamHome

Figure 18.7 Log On dialog box requesting user name, password, and the name of the database the user wishes to connect to.

Privileges

As we discussed in Section 18.2.1, a **privilege** is a right to execute a particular type of SQL statement or to access another user's objects. Some examples of Oracle privileges include the right to:

- connect to the database (create a session);
- create a table;
- select rows from another user's table.

In Oracle, there are two distinct categories of privileges:

- system privileges;
- object privileges.

System privileges

A **system privilege** is the right to perform a particular action or to perform an action on any schema objects of a particular type. For example, the privileges to create tablespaces and to create users in a database are system privileges. There are over eighty distinct system privileges in Oracle. System privileges are granted to, or revoked from, users and **roles** (discussed below) using either of the following:

- Grant System Privileges/Roles dialog box and Revoke System Privileges/Roles dialog box of the Oracle Security Manager;
- SQL GRANT and REVOKE statements (see Section 6.6).

However, only users who are granted a specific system privilege with the ADMIN OPTION or users with the GRANT ANY PRIVILEGE system privilege can grant or revoke system privileges.

Object privileges

An **object privilege** is a privilege or right to perform a particular action on a specific table, view, sequence, procedure, function, or package. Different object privileges are available for different types of object. For example, the privilege to delete rows from the Staff table is an object privilege.

Some schema objects (such as clusters, indexes, and triggers) do not have associated object privileges; their use is controlled with system privileges. For example, to alter a cluster, a user must own the cluster or have the ALTER ANY CLUSTER system privilege.

A user automatically has all object privileges for schema objects contained in his or her schema. A user can grant any object privilege on any schema object he or she owns to any other user or role. If the grant includes the WITH GRANT OPTION (of the GRANT statement), the grantee can further grant the object privilege to other users; otherwise, the grantee can use the privilege but cannot grant it to other users. The object privileges for tables and views are shown in Table 18.5.

Table 18.5 What each object privilege allows a grantee to do with tables and views.

Object privilege	Table	View
ALTER	Change the table definition with the ALTER TABLE statement.	N/A
DELETE	Remove rows from the table with the DELETE statement. Note: SELECT privilege on the table must be granted along with the DELETE privilege.	Remove rows from the view with the DELETE statement.
INDEX	Create an index on the table with the CREATE INDEX statement.	N/A
INSERT	Add new rows to the table with the INSERT statement.	Add new rows to the view with the INSERT statement.
REFERENCES	Create a constraint that refers to the table. Cannot grant this privilege to a role.	N/A
SELECT	Query the table with the SELECT statement.	Query the view with the SELECT statement.
UPDATE	Change data in the table with the UPDATE statement. Note: SELECT privilege on the table must be granted along with the UPDATE privilege.	Change data in the view with the UPDATE statement.

Roles

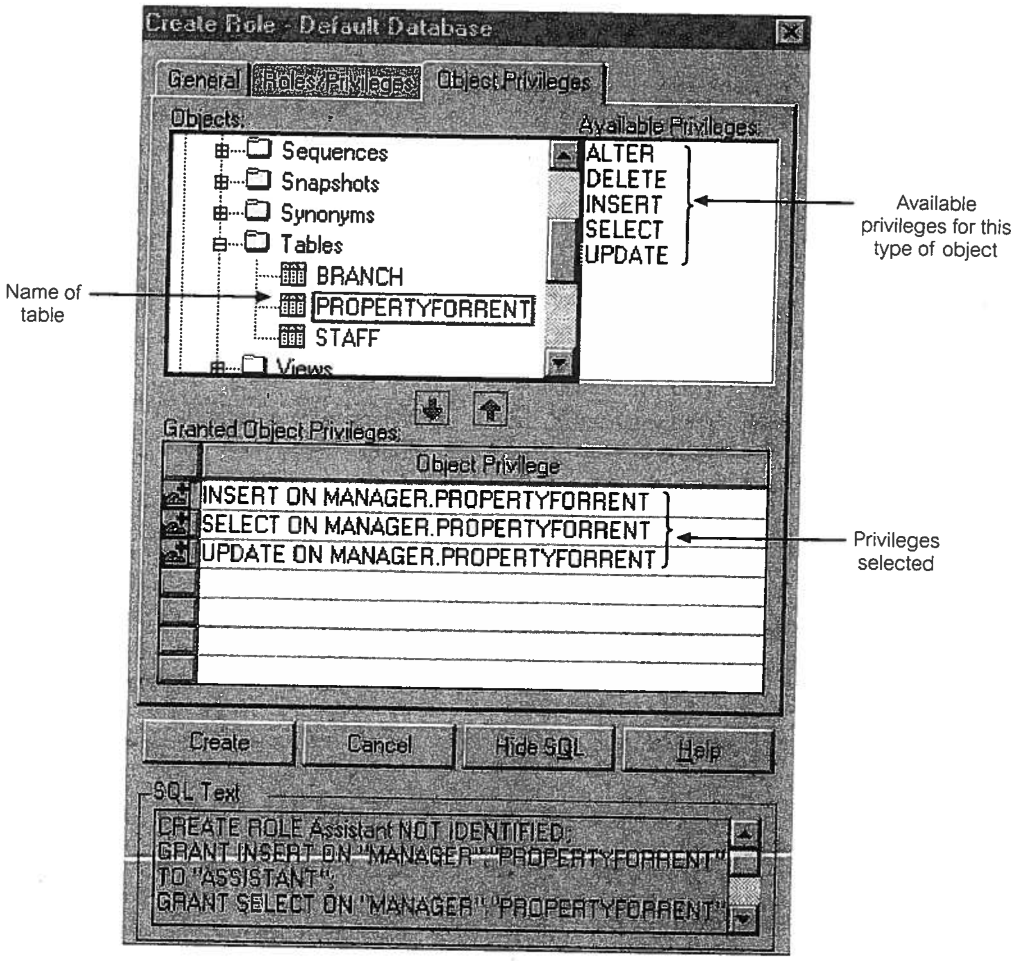
A user can receive a privilege in two different ways:

- Privileges can be granted to users explicitly. For example, a user can explicitly grant the privilege to insert rows into the PropertyForRent table to the user Beech:

  GRANT INSERT ON PropertyForRent **TO** Beech;

- Privileges can also be granted to a **role** (a named group of privileges), and then the role granted to one or more users. For example, a user can grant the privileges to select, insert, and update rows from the PropertyForRent table to the role named Assistant, which in turn can be granted to the user Beech. A user can have access to several roles, and several users can be assigned the same roles. Figure 18.8 illustrates the granting of these privileges to the role Assistant using the Oracle Security Manager.

Because roles allow for easier and better management of privileges, privileges should normally be granted to roles and not to specific users.



```
INSERT ON MANAGER.PROPERTYFORRENT
SELECT ON MANAGER.PROPERTYFORRENT
UPDATE ON MANAGER.PROPERTYFORRENT
```

```
CREATE ROLE Assistant NOT IDENTIFIED;
GRANT INSERT ON "MANAGER"."PROPERTYFORRENT"
TO "ASSISTANT";
GRANT SELECT ON "MANAGER"."PROPERTYFORRENT"
```

Figure 18.8 Setting the Insert, Select, and Update privileges on the PropertyForRent table to the role Assistant.

18.5 DBMSs and Web Security

In Chapter 28 we provide a general overview of DBMSs on the Web. In this section we focus on how to make a DBMS secure on the Web. Those readers unfamilair with the terms and technologies associated with DBMSs on the Web are advised to read Chapter 28 before reading this section.

Internet communication relies on TCP/IP as the underlying protocol. However, TCP/IP and HTTP were not designed with security in mind. Without special software, all Internet traffic travels 'in the clear' and anyone who monitors traffic can read it. This form of attack is relatively easy to perpetrate using freely available 'packet sniffing' software, since the Internet has traditionally been an open network. Consider, for example, the implications of credit card numbers being intercepted by unethical parties during transmission when customers use their cards to purchase products over the Internet. The challenge is to transmit and receive information over the Internet while ensuring that:

- it is inaccessible to anyone but the sender and receiver (privacy);
- it has not been changed during transmission (integrity);
- the receiver can be sure it came from the sender (authenticity);
- the sender can be sure the receiver is genuine (non-fabrication);
- the sender cannot deny he or she sent it (non-repudiation).

However, protecting the transaction only solves part of the problem. Once the information has reached the Web server, it must also be protected there. With the three-tier architecture that is popular in a Web environment, we also have the complexity of ensuring secure access to, and of, the database. Today, most parts of such architecture can be secured, but it generally requires different products and mechanisms.

One other aspect of security that has to be addressed in the Web environment is that information transmitted to the client's machine may have executable content. For example, HTML pages may contain ActiveX controls, JavaScript/VBScript, and/or one or more Java applets. Executable content can perform the following malicious actions, and measures need to be taken to prevent them:

- corrupt data or the execution state of programs;
- reformat complete disks;
- perform a total system shutdown;
- collect and download confidential data, such as files or passwords, to another site;
- usurp identity and impersonate the user or user's computer to attack other targets on the network;
- lock up resources making them unavailable for legitimate users and programs;
- cause non-fatal but unwelcome effects, especially on output devices.

In earlier sections we identified general security mechanisms for database systems. However, the increasing accessibility of databases on the public Internet and private intranets requires a re-analysis and extension of these approaches. In this section we address some of the issues associated with database security in these environments.

Proxy Servers 18.5.1

In a Web environment, a proxy server is a computer that sits between a Web browser and a Web server. It intercepts all requests to the Web server to determine if it can fulfill the requests itself. If not, it forwards the requests to the Web server. Proxy servers have two main purposes: to improve performance and filter requests.

Improve performance

Since a proxy server saves the results of all requests for a certain amount of time, it can significantly improve performance for groups of users. For example, assume that user A and user B access the Web through a proxy server. First, user A requests a certain Web page and, slightly later, user B requests the same page. Instead of forwarding the request to the Web server where that page resides, the proxy server simply returns the cached page that it had already fetched for user A. Since the proxy server is often on the same network as the user, this is a much faster operation. Real proxy servers, such as those employed by Compuserve and America Online, can support thousands of users.

Filter requests

Proxy servers can also be used to filter requests. For example, an organization might use a proxy server to prevent its employees from accessing a specific set of Web sites.

18.5.2 Firewalls

The standard security advice is to ensure that Web servers are unconnected to any in-house networks and regularly backed up to recover from inevitable attacks. When the Web server has to be connected to an internal network, for example to access the company database, firewall technology can help to prevent unauthorized access, provided it has been installed and maintained correctly.

A **firewall** is a system designed to prevent unauthorized access to or from a private network. Firewalls can be implemented in both hardware and software, or a combination of both. They are frequently used to prevent unauthorized Internet users from accessing private networks connected to the Internet, especially intranets. All messages entering or leaving the intranet pass through the firewall, which examines each message and blocks those that do not meet the specified security criteria. There are several types of firewall technique:

- **Packet filter**, which looks at each packet entering or leaving the network and accepts or rejects it based on user-defined rules. Packet filtering is a fairly effective mechanism and transparent to users, but can be difficult to configure. In addition, it is susceptible to IP spoofing. (IP spoofing is a technique used to gain unauthorized access to computers, whereby the intruder sends messages to a computer with an IP address indicating that the message is coming from a trusted port.)
- **Application gateway**, which applies security mechanisms to specific applications, such as FTP and Telnet servers. This is a very effective mechanism, but can degrade performance.
- **Circuit-level gateway**, which applies security mechanisms when a TCP or UDP (User Datagram Protocol) connection is established. Once the connection has been made, packets can flow between the hosts without further checking.
- **Proxy server**, which intercepts all messages entering and leaving the network. The proxy server in effect hides the true network addresses.

In practice, many firewalls provide more than one of these techniques. A firewall is considered a first line of defense in protecting private information. For greater security, data can be encrypted, as discussed below and earlier in Section 18.2.

Message Digest Algorithms and Digital Signatures 18.5.3

A message digest algorithm, or one-way hash function, takes an arbitrary-sized string (the *message*) and generates a fixed-length string (the *digest* or *hash*). A digest has the following characteristics:

- it should be computationally infeasible to find another message that will generate the same digest;
- the digest does not reveal anything about the message.

A digital signature consists of two pieces of information: a string of bits that is computed from the data that is being 'signed', along with the private key of the individual or organization wishing the signature. The signature can be used to verify that the data comes from this individual or organization. Like a handwritten signature, a digital signature has many useful properties:

- its authenticity can be verified, using a computation based on the corresponding public key;
- it cannot be forged (assuming the private key is kept secret);
- it is a function of the data signed and cannot be claimed to be the signature for any other data;
- the signed data cannot be changed, otherwise the signature will no longer verify the data as being authentic.

Some digital signature algorithms use message digest algorithms for parts of their computations; others, for efficiency, compute the digest of a message and digitally sign the digest rather than signing the message itself.

Digital Certificates 18.5.4

A digital certificate is an attachment to an electronic message used for security purposes, most commonly to verify that a user sending a message is who he or she claims to be, and to provide the receiver with the means to encode a reply.

An individual wishing to send an encrypted message applies for a digital certificate from a Certificate Authority (CA). The CA issues an encrypted digital certificate containing the applicant's public key and a variety of other identification information. The CA makes its own public key readily available through printed material or perhaps on the Internet.

The recipient of an encrypted message uses the CA's public key to decode the digital certificate attached to the message, verifies it as issued by the CA, and then obtains the sender's public key and identification information held within the certificate. With this information, the recipient can send an encrypted reply.

Clearly, the CA's role in this process is critical, acting as a go-between for the two parties. In a large, distributed complex network like the Internet, this third-party trust model is necessary as clients and servers may not have an established mutual trust yet both parties want to have a secure session. However, because each party trusts the CA, and because the CA is vouching for each party's identification and trustworthiness by signing their certificates, each party recognizes and implicitly trusts each other. The most widely used standard for digital certificates is X.509.

18.5.5 Kerberos

Kerberos is a server of secured user names and passwords (named after the three-headed monster in Greek mythology that guarded the gate of hell). The importance of Kerberos is that it provides one centralized security server for all data and resources on the network. Database access, login, authorization control, and other security features are centralized on trusted Kerberos servers. Kerberos has a similar function to that of a Certificate server: to identify and validate a user. Security companies are currently investigating a merger of Kerberos and Certificate servers to provide a network-wide secure system.

18.5.6 Secure Sockets Layer and Secure HTTP

Many large Internet product developers agreed to use an encryption protocol known as Secure Sockets Layer (SSL) developed by Netscape for transmitting private documents over the Internet. SSL works by using a private key to encrypt data that is transferred over the SSL connection. Both Netscape Navigator and Internet Explorer support SSL, and many Web sites use this protocol to obtain confidential user information, such as credit card numbers. The protocol, layered between application-level protocols such as HTTP and the TCP/IP transport-level protocol, is designed to prevent eavesdropping, tampering, and message forgery. Since SSL is layered under application-level protocols, it may be used for other application-level protocols such as FTP and NNTP.

Another protocol for transmitting data securely over the Web is Secure HTTP (S-HTTP), a modified version of the standard HTTP protocol. S-HTTP was developed by Enterprise Integration Technologies (EIT), which was acquired by Verifone, Inc. in 1995. Whereas SSL creates a secure connection between a client and a server, over which any amount of data can be sent securely, S-HTTP is designed to transmit individual messages securely. SSL and S-HTTP, therefore, can be seen as complementary rather than competing technologies. Both protocols have been submitted to the Internet Engineering Task Force (IETF) for approval as standards. By convention, Web pages that require an SSL connection start with **https:** instead of **http:**. Not all Web browsers and servers support SSL/S-HTTP.

Basically, these protocols allow the browser and server to authenticate one another and secure information that subsequently flows between them. Through the use of cryptographic techniques such as encryption, and digital signatures, these protocols:

- allow Web browsers and servers to authenticate each other;
- permit Web site owners to control access to particular servers, directories, files, or services;
- allow sensitive information (for example, credit card numbers) to be shared between browser and server, yet remain inaccessible to third parties;
- ensure that data exchanged between browser and server is reliable (that is, cannot be corrupted either accidentally or deliberately, without detection).

A key component in the establishment of secure Web sessions using the SSL or S-HTTP protocols is the digital certificate, discussed above. Without authentic and trustworthy certificates, protocols like SSL and S-HTTP offer no security at all.

Secure Electronic Transactions and Secure Transaction Technology 18.5.7

The Secure Electronic Transactions (SET) protocol is an open, interoperable standard for processing credit card transactions over the Internet, created jointly by Netscape, Microsoft, Visa, Mastercard, GTE, SAIC, Terisa Systems, and VeriSign. SET's goal is to allow credit card transactions to be as simple and secure on the Internet as they are in retail stores. To address privacy concerns, the transaction is split in such a way that the merchant has access to information about what is being purchased, how much it costs, and whether the payment is approved, but no information on what payment method the customer is using. Similarly, the card issuer (for example, Visa) has access to the purchase price but no information on the type of merchandise involved.

Certificates are heavily used by SET, both for certifying a cardholder and for certifying that the merchant has a relationship with the financial institution. The mechanism is illustrated in Figure 18.9. While both Microsoft and Visa International are major participants in the SET specifications, they currently provide the Secure Transaction Technology (STT)

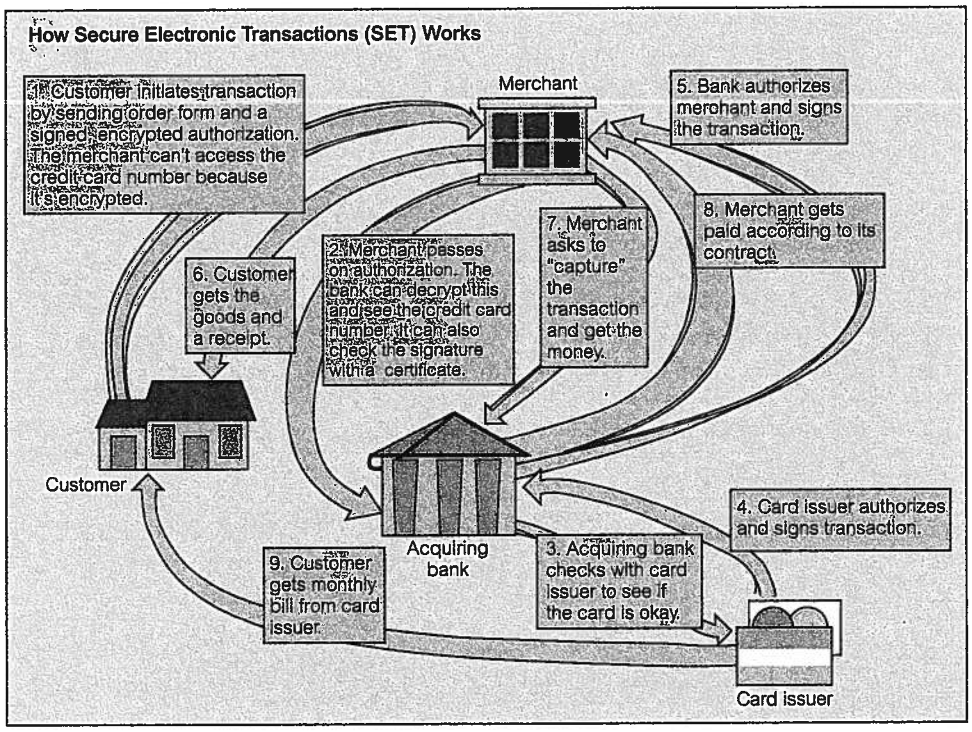

How Secure Electronic Transactions (SET) Works
1. Customer initiates transaction by sending order form and a signed, encrypted authorization. The merchant can't access the credit card number because it's encrypted.
Merchant
5. Bank authorizes merchant and signs the transaction.
8. Merchant gets paid according to its contract.
7. Merchant asks to "capture" the transaction and get the money.
2. Merchant passes on authorization. The bank can decrypt this and see the credit card number. It can also check the signature with a certificate.
6. Customer gets the goods and a receipt.
Customer
Acquiring bank
4. Card issuer authorizes and signs transaction.
3. Acquiring bank checks with card issuer to see if the card is okay.
9. Customer gets monthly bill from card issuer.
Card issuer

Figure 18.9
A SET transaction.

protocol, which has been designed to handle secure bank payments over the Internet. STT uses DES encryption of information, RSA encryption of bankcard information, and strong authentication of all parties involved in the transaction .

18.5.8 Java Security

In Section 28.8 we introduce the Java language as an increasingly important language for Web development. Those readers unfamiliar with Java are advised to read Section 28.8 before reading this section.

Safety and security are integral parts of Java's design, with the 'sandbox' ensuring that an untrusted, possibly malicious, application cannot gain access to system resources. To implement this sandbox, three components are used: a class loader, a bytecode verifier, and a security manager. The safety features are provided by the Java language and the Java Virtual Machine (JVM), and enforced by the compiler and the runtime system; security is a policy that is built on top of this safety layer.

Two safety features of the Java language relate to strong typing and automatic garbage collection. In this section we look at two other features: the class loader and the bytecode verifier. To complete this section on Java security, we examine the JVM Security Manager.

The class loader

The class loader, as well as loading each required class and checking it is in the correct format, additionally checks that the application/applet does not violate system security by allocating a *namespace*. Namespaces are hierarchical and allow the JVM to group classes based on where they originate (local or remote). A class loader never allows a class from a 'less protected' namespace to replace a class from a more protected namespace. In this way, the file system's I/O primitives, which are defined in a local Java class, cannot be invoked or indeed overridden by classes from outside the local machine. An executing JVM allows multiple class loaders, each with its own namespace, to be active simultaneously. As browsers and Java applications can typically provide their own class loader, albeit based on a recommended template from Sun Microsystems, this may be viewed as a weakness in the security model. However, some argue that this is a strength of the language, allowing system administrators to implement their own (presumably tighter) security measures.

The bytecode verifier

Before the JVM will allow an application/applet to run, its code must be verified. The verifier assumes that all code is meant to crash or violate system security and performs a series of checks, including the execution of a theorem prover, to ensure that this is not the case. Typical checks include verifying that:

- compiled code is correctly formatted;
- internal stacks will not overflow/underflow;

- no 'illegal' data conversions will occur (for example, integer to pointer) – this ensures that variables will not be granted access to restricted memory areas;
- bytecode instructions are appropriately typed;
- all class member accesses are valid.

The Security Manager

The Java security policy is application specific. A Java application, such as a Java-enabled Web browser or a Web server, defines and implements its own security policy. Each of these applications implements its own Security Manager. A Java-enabled Web browser contains its own applet Security Manager, and any applets downloaded by this browser are subject to its policies. Generally, the Security Manager performs runtime verification of potentially 'dangerous' methods, that is, methods that request I/O, network access, or wish to define a new class loader. In general, downloaded applets are prevented from:

- reading and writing files on the client's file system. This also prevents applets storing persistent data (for example, a database) on the client side, although the data could be sent back to the host for storage;
- making network connections to machines other than the host that provided the compiled '.class' files. This is either the host where the HTML page came from, or the host specified in the CODEBASE parameter in the applet tag, with CODEBASE taking precedence;
- starting other programs on the client;
- loading libraries;
- defining method calls. Allowing an applet to define native method calls would give the applet direct access to the underlying operating system.

These restrictions apply to applets that are downloaded over the public Internet or company intranet. They do not apply to applets on the client's local disk and in a directory that is on the client's CLASSPATH. Local applets are loaded by the file system loader and, as well as being able to read and write files, are allowed to exit the virtual machine and are not passed through the bytecode verifier. The JDK (Java Development Kit) Appletviewer also slightly relaxes these restrictions, by letting the user define an explicit list of files that can be accessed by downloaded applets. In a similar way, Microsoft's Internet Explorer 4.0 introduced the concept of 'zones', and some zones may be trusted and others untrusted. Java applets loaded from certain zones are able to read and write to files on the client's hard drive. The zones with which this is possible are customizable by the Network Administrators.

Enhanced applet security

The sandbox model was introduced with the first release of the Java applet API in January 1996. Although this model does generally protect systems from untrusted code obtained from the network, it does not address several other security and privacy issues. Authentication is needed to ensure that an applet comes from where it claims to have come from. Further, digitally signed and authenticated applets can then be raised to the status of trusted applets, and subsequently allowed to run with fewer security restrictions.

The Java Security API, available in JDK 1.1, contains APIs for digital signatures, message digests, key management, and encryption/decryption (subject to United States export control regulations). Work is in progress to define an infrastructure that allows flexible security policies for signed applets.

18.5.9 ActiveX Security

The ActiveX security model is considerably different from Java applets. Java achieves security by restricting the behavior of applets to a safe set of instructions. ActiveX, on the other hand, places no restrictions on what a control can do. Instead, each ActiveX control can be digitally signed by its author using a system called Authenticode™. The digital signatures are then certified by a Certificate Authority (CA). This security model places the responsibility for the computer's security on the user. Before the browser downloads an ActiveX control that has not been signed or has been certified by an unknown CA, it presents a dialog box warning the user that this action may not be safe. The user can then abort the transfer or continue and accept the consequences.

Chapter Summary

- **Database security** is the mechanisms that protect the database against intentional or accidental threats.
- Database security is concerned with avoiding the following situations: theft and fraud, loss of confidentiality (secrecy), loss of privacy, loss of integrity, and loss of availability.
- A **threat** is any situation or event, whether intentional or accidental, that will adversely affect a system and consequently an organization.
- **Computer-based security controls** for the multi-user environment include: authorization, views, backup and recovery, integrity, encryption, and RAID technology.
- **Authorization** is the granting of a right or privilege that enables a subject to have legitimate access to a system or a system's object. **Authentication** is a mechanism that determines whether a user is who he or she claims to be.
- A **view** is the dynamic result of one or more relational operations operating on the base relations to produce another relation. A view is a **virtual relation** that does not actually exist in the database but is produced upon request by a particular user at the time of request. The view mechanism provides a powerful and flexible security mechanism by hiding parts of the database from certain users.
- **Backup** is the process of periodically taking a copy of the database and log file (and possibly programs) on to offline storage media. **Journaling** is the process of keeping and maintaining a log file (or journal) of all changes made to the database to enable recovery to be undertaken effectively in the event of a failure.
- **Integrity constraints** also contribute to maintaining a secure database system by preventing data from becoming invalid, and hence giving misleading or incorrect results.
- **Encryption** is the encoding of the data by a special algorithm that renders the data unreadable by any program without the decryption key.

- **Microsoft Access DBMS** and **Oracle DBMS** provide two types of security measure: system security and data security. **System security** enables the setting of a password for opening a database, and **data security** provides user-level security which can be used to limit the parts of a database that a user can read and update.
- The security measures associated with **DBMSs on the Web** include: proxy servers, firewalls, message digest algorithms and digital signatures, digital certificates, kerberos, Secure Sockets Layer (SSL) and Secure HTTP (S-HTTP), Secure Electronic Transactions (SET) and Secure Transaction Technology (SST), Java security, and ActiveX security.

Review Questions

18.1 Explain the purpose and scope of database security.

18.2 List the main types of threat that could affect a database system, and for each describe the controls that you would use to counteract each of them.

18.3 Explain the following in terms of providing security for a database:
(a) authorization;
(b) views;
(c) backup and recovery;
(d) integrity;
(e) encryption;
(f) RAID technology.

18.4 Describe the security measure provided by Microsoft Access or Oracle DBMSs.

18.5 Describe the approaches for securing DBMSs on the Web.

Exercises

18.6 Examine any DBMS used by your organization and identify the security measures provided.

18.7 Identify the types of security approach that are used by your organization to secure any DBMSs that are accessible over the Web.

18.8 Consider the *DreamHome* case study described in Chapter 10. List the potential threats that could occur and propose countermeasures to overcome them.

18.9 Consider the *Wellmeadows Hospital* case study described in Appendix B.3. List the potential threats that could occurr and propose countermeasures to overcome them.