23

Database Security and Authorization

This chapter discusses the techniques used for protecting the database against persons who are not authorized to access either certain parts of a database or the whole database. Section 23.1 provides an introduction to security issues and the threats to databases and an overview of the countermeasures that are covered in the rest of this chapter. Section 23.2 discusses the mechanisms used to grant and revoke privileges in relational database systems and in SQL, mechanisms that are often referred to as **discretionary access control**. Section 23.3 offers an overview of the mechanisms for enforcing multiple levels of security—a more recent concern in database system security that is known as **mandatory access control.** It also introduces the more recently developed strategy of **role-based access control.** Section 23.4 briefly discusses the security problem in statistical databases. Section 23.5 introduces flow control and mentions problems associated with covert channels. Section 23.6 is a brief summary of encryption and public key infrastructure schemes. Section 23.7 summarizes the chapter. Readers who are interested only in basic database security mechanisms will find it sufficient to cover the material in Sections 23.1 and 23.2.

23.1 INTRODUCTION TO DATABASE SECURITY ISSUES

23.1.1 Types of Security

Database security is a very broad area that addresses many issues, including the following:

- Legal and ethical issues regarding the right to access certain information. Some information may be deemed to be private and cannot be accessed legally by unauthorized persons. In the United States, there are numerous laws governing privacy of information.

- Policy issues at the governmental, institutional, or corporate level as to what kinds of information should not be made publicly available—for example, credit ratings and personal medical records.

- System-related issues such as the *system levels* at which various security functions should be enforced—for example, whether a security function should be handled at the physical hardware level, the operating system level, or the DBMS level.

- The need in some organizations to identify multiple *security levels* and to categorize the data and users based on these classifications—for example, top secret, secret, confidential, and unclassified. The security policy of the organization with respect to permitting access to various classifications of data must be enforced.

Threats to Databases. Threats to databases result in the loss or degradation of some or all of the following security goals: integrity, availability, and confidentiality.

- *Loss of integrity:* Database integrity refers to the requirement that information be protected from improper modification. Modification of data includes creation, insertion, modification, changing the status of data, and deletion. Integrity is lost if unauthorized changes are made to the data by either intentional or accidental acts. If the loss of system or data integrity is not corrected, continued use of the contaminated system or corrupted data could result in inaccuracy, fraud, or erroneous decisions.

- *Loss of availability:* Database availability refers to making objects available to a human user or a program to which they have a legitimate right.

- *Loss of confidentiality:* Database confidentiality refers to the protection of data from unauthorized disclosure. The impact of unauthorized disclosure of confidential information can range from violation of the Data Privacy Act to the jeopardization of national security. Unauthorized, unanticipated, or unintentional disclosure could result in loss of public confidence, embarrassment, or legal action against the organization.

To protect databases against these types of threats four kinds of countermeasures can be implemented: access control, inference control, flow control, and encryption. We discuss each of these in this chapter.

In a multiuser database system, the DBMS must provide techniques to enable certain users or user groups to access selected portions of a database *without* gaining access to the rest of the database. This is particularly important when a large integrated database is to be used by many different users within the same organization. For example, sensitive

information such as employee salaries or performance reviews should be kept confidential from most of the database system's users. A DBMS typically includes a **database security and authorization subsystem** that is responsible for ensuring the security of portions of a database against unauthorized access. It is now customary to refer to two types of database security mechanisms:

- *Discretionary security mechanisms*: These are used to grant privileges to users, including the capability to access specific data files, records, or fields in a specified mode (such as read, insert, delete, or update).

- *Mandatory security mechanisms*: These are used to enforce multilevel security by classifying the data and users into various security classes (or levels) and then implementing the appropriate security policy of the organization. For example, a typical security policy is to permit users at a certain classification level to see only the data items classified at the user's own (or lower) classification level. An extension of this is *role-based security*, which enforces policies and privileges based on the concept of roles.

We discuss discretionary security in Section 23.2 and mandatory and role-based security in Section 23.3.

A second security problem common to all computer systems is that of preventing unauthorized persons from accessing the system itself, either to obtain information or to make malicious changes in a portion of the database. The security mechanism of a DBMS must include provisions for restricting access to the database system as a whole. This function is called **access control** and is handled by creating user accounts and passwords to control the login process by the DBMS. We discuss access control techniques in Section 23.1.3.

A third security problem associated with databases is that of controlling the access to a **statistical database,** which is used to provide statistical information or summaries of values based on various criteria. For example, a database for population statistics may provide statistics based on age groups, income levels, size of household, education levels, and other criteria. Statistical database users such as government statisticians or market research firms are allowed to access the database to retrieve statistical information about a population but not to access the detailed confidential information on specific individuals. Security for statistical databases must ensure that information on individuals cannot be accessed. It is sometimes possible to deduce or infer certain facts concerning individuals from queries that involve only summary statistics on groups; consequently, this must not be permitted either. This problem, called **statistical database security,** is discussed briefly in Section 23.4. The corresponding countermeasures are called **inference control** measures.

Another security issue is that of **flow control,** which prevents information from flowing in such a way that it reaches unauthorized users. It is discused in Section 23.5. Channels that are pathways for information to flow implicitly in ways that violate the security policy of an organization are called **covert channels.** We briefly discuss some issues related to covert channels in Section 23.5.1.

A final security issue is **data encryption,** which is used to protect sensitive data (such as credit card numbers) that is being transmitted via some type of communications network. Encryption can be used to provide additional protection for sensitive portions of a database as well. The data is **encoded** using some coding algorithm. An unauthorized user who accesses encoded data will have difficulty deciphering it, but authorized users are given decoding or

decrypting algorithms (or keys) to decipher the data. Encrypting techniques that are very difficult to decode without a key have been developed for military applications. Section 23.6 briefly discusses encryption techniques, including popular techniques such as public key encryption, which is heavily used to support Web-based transactions against databases, and digital signatures, which are used in personal communications.

A complete discussion of security in computer systems and databases is outside the scope of this textbook. We give only a brief overview of database security techniques here. The interested reader can refer to several of the references discussed in the selected bibliography at the end of this chapter for a more comprehensive discussion.

23.1.2 Database Security and the DBA

As we discussed in Chapter 1, the database administrator (DBA) is the central authority for managing a database system. The DBA's responsibilities include granting privileges to users who need to use the system and classifying users and data in accordance with the policy of the organization. The DBA has a **DBA account** in the DBMS, sometimes called a **system** or **superuser account,** which provides powerful capabilities that are not made available to regular database accounts and users.[1] DBA-privileged commands include commands for granting and revoking privileges to individual accounts, users, or user groups and for performing the following types of actions:

1. *Account creation:* This action creates a new account and password for a user or a group of users to enable access to the DBMS.

2. *Privilege granting:* This action permits the DBA to grant certain privileges to certain accounts.

3. *Privilege revocation:* This action permits the DBA to revoke (cancel) certain privileges that were previously given to certain accounts.

4. *Security level assignment:* This action consists of assigning user accounts to the appropriate security classification level.

The DBA is responsible for the overall security of the database system. Action 1 in the preceding list is used to control access to the DBMS as a whole, whereas actions 2 and 3 are used to control *discretionary* database authorization, and action 4 is used to control *mandatory* authorization.

23.1.3 Access Protection, User Accounts, and Database Audits

Whenever a person or a group of persons needs to access a database system, the individual or group must first apply for a user account. The DBA will then create a new **account**

---

1. This account is similar to the *root* or *superuser* accounts that are given to computer system administrators, allowing access to restricted operating system commands.

number and **password** for the user if there is a legitimate need to access the database. The user must **log in** to the DBMS by entering the account number and password whenever database access is needed. The DBMS checks that the account number and password are valid; if they are, the user is permitted to use the DBMS and to access the database. Application programs can also be considered as users and can be required to supply passwords.

It is straightforward to keep track of database users and their accounts and passwords by creating an encrypted table or file with the two fields AccountNumber and Password. This table can easily be maintained by the DBMS. Whenever a new account is created, a new record is inserted into the table. When an account is canceled, the corresponding record must be deleted from the table.

The database system must also keep track of all operations on the database that are applied by a certain user throughout each **login session**, which consists of the sequence of database interactions that a user performs from the time of logging in to the time of logging off. When a user logs in, the DBMS can record the user's account number and associate it with the terminal from which the user logged in. All operations applied from that terminal are attributed to the user's account until the user logs off. It is particularly important to keep track of update operations that are applied to the database so that, if the database is tampered with, the DBA can find out which user did the tampering.

To keep a record of all updates applied to the database and of the particular user who applied each update, we can modify the *system log*. Recall from Chapters 17 and 19 that the **system log** includes an entry for each operation applied to the database that may be required for recovery from a transaction failure or system crash. We can expand the log entries so that they also include the account number of the user and the online terminal ID that applied each operation recorded in the log. If any tampering with the database is suspected, a **database audit** is performed, which consists of reviewing the log to examine all accesses and operations applied to the database during a certain time period. When an illegal or unauthorized operation is found, the DBA can determine the account number used to perform this operation. Database audits are particularly important for sensitive databases that are updated by many transactions and users, such as a banking database that is updated by many bank tellers. A database log that is used mainly for security purposes is sometimes called an **audit trail.**

23.2 DISCRETIONARY ACCESS CONTROL BASED ON GRANTING AND REVOKING PRIVILEGES

The typical method of enforcing **discretionary access control** in a database system is based on the granting and revoking of **privileges.** Let us consider privileges in the context of a relational DBMS. In particular, we will discuss a system of privileges somewhat similar to the one originally developed for the SQL language (see Chapter 8). Many current relational DBMSs use some variation of this technique. The main idea is to include statements in the query language that allow the DBA and selected users to grant and revoke privileges.

23.2.1 Types of Discretionary Privileges

In SQL2, the concept of an **authorization identifier** is used to refer, roughly speaking, to a user account (or group of user accounts). For simplicity, we will use the words *user* or *account* interchangeably in place of *authorization identifier*. The DBMS must provide selective access to each relation in the database based on specific accounts. Operations may also be controlled; thus, having an account does not necessarily entitle the account holder to all the functionality provided by the DBMS. Informally, there are two levels for assigning privileges to use the database system:

- *The account level:* At this level, the DBA specifies the particular privileges that each account holds independently of the relations in the database.

- *The relation (or table) level:* At this level, the DBA can control the privilege to access each individual relation or view in the database.

The privileges at the **account level** apply to the capabilities provided to the account itself and can include the CREATE SCHEMA or CREATE TABLE privilege, to create a schema or base relation; the CREATE VIEW privilege; the ALTER privilege, to apply schema changes such as adding or removing attributes from relations; the DROP privilege, to delete relations or views; the MODIFY privilege, to insert, delete, or update tuples; and the SELECT privilege, to retrieve information from the database by using a SELECT query. Notice that these account privileges apply to the account in general. If a certain account does not have the CREATE TABLE privilege, no relations can be created from that account. Account-level privileges *are not* defined as part of SQL2; they are left to the DBMS implementers to define. In earlier versions of SQL, a CREATETAB privilege existed to give an account the privilege to create tables (relations).

The second level of privileges applies to the **relation level,** whether they are base relations or virtual (view) relations. These privileges *are* defined for SQL2. In the following discussion, the term *relation* may refer either to a base relation or to a view, unless we explicitly specify one or the other. Privileges at the relation level specify for each user the individual relations on which each type of command can be applied. Some privileges also refer to individual columns (attributes) of relations. SQL2 commands provide privileges at the *relation and attribute level only*. Although this is quite general, it makes it difficult to create accounts with limited privileges. The granting and revoking of privileges generally follow an authorization model for discretionary privileges known as the **access matrix model,** where the rows of a matrix M represent *subjects* (users, accounts, programs) and the columns represent *objects* (relations, records, columns, views, operations). Each position $M(i, j)$ in the matrix represents the types of privileges (read, write, update) that subject i holds on object j.

To control the granting and revoking of relation privileges, each relation R in a database is assigned an **owner account,** which is typically the account that was used when the relation was created in the first place. The owner of a relation is given *all* privileges on that relation. In SQL2, the DBA can assign an owner to a whole schema by creating the schema and associating the appropriate authorization identifier with that schema, using the CREATE SCHEMA command (see Section 8.1.1). The owner account holder can pass privileges on any of the owned relations to other users by **granting** privileges to their

accounts. In SQL the following types of privileges can be granted on each individual relation R:

- SELECT (retrieval or read) privilege on R: Gives the account retrieval privilege. In SQL this gives the account the privilege to use the SELECT statement to retrieve tuples from R.

- MODIFY privileges on R: This gives the account the capability to modify tuples of R. In SQL this privilege is further divided into UPDATE, DELETE, and INSERT privileges to apply the corresponding SQL command to R. In addition, both the INSERT and UPDATE privileges can specify that only certain attributes of R can be updated by the account.

- REFERENCES privilege on R: This gives the account the capability to reference relation R when specifying integrity constraints. This privilege can also be restricted to specific attributes of R.

Notice that to create a view, the account must have SELECT privilege on *all relations* involved in the view definition.

23.2.2 Specifying Privileges Using Views

The mechanism of **views** is an important discretionary authorization mechanism in its own right. For example, if the owner A of a relation R wants another account B to be able to retrieve only some fields of R, then A can create a view V of R that includes only those attributes and then grant SELECT on V to B. The same applies to limiting B to retrieving only certain tuples of R; a view V' can be created by defining the view by means of a query that selects only those tuples from R that A wants to allow B to access. We shall illustrate this discussion with the example given in Section 23.2.5.

23.2.3 Revoking Privileges

In some cases it is desirable to grant a privilege to a user temporarily. For example, the owner of a relation may want to grant the SELECT privilege to a user for a specific task and then revoke that privilege once the task is completed. Hence, a mechanism for **revoking** privileges is needed. In SQL a REVOKE command is included for the purpose of canceling privileges. We will see how the REVOKE command is used in the example in Section 23.2.5.

23.2.4 Propagation of Privileges Using the GRANT OPTION

Whenever the owner A of a relation R grants a privilege on R to another account B, the privilege can be given to B *with* or *without* the GRANT OPTION. If the GRANT OPTION is given, this means that B can also grant that privilege on R to other accounts. Suppose that B is given the GRANT OPTION by A and that B then grants the privilege on R to a

third account C, also with GRANT OPTION. In this way, privileges on R can **propagate** to other accounts without the knowledge of the owner of R. If the owner account A now revokes the privilege granted to B, all the privileges that B propagated based on that privilege should automatically be revoked by the system.

It is possible for a user to receive a certain privilege from two or more sources. For example, A4 may receive a certain UPDATE R privilege from *both* A2 and A3. In such a case, if A2 revokes this privilege from A4, A4 will still continue to have the privilege by virtue of having been granted it from A3. If A3 later revokes the privilege from A4, A4 totally loses the privilege. Hence, a DBMS that allows propagation of privileges must keep track of how all the privileges were granted so that revoking of privileges can be done correctly and completely.

23.2.5 An Example

Suppose that the DBA creates four accounts—A1, A2, A3, and A4—and wants only A1 to be able to create base relations; then the DBA must issue the following GRANT command in SQL:

GRANT CREATETAB **TO** A1;

The CREATETAB (create table) privilege gives account A1 the capability to create new database tables (base relations) and is hence an *account privilege*. This privilege was part of earlier versions of SQL but is now left to each individual system implementation to define.

In SQL2, the same effect can be accomplished by having the DBA issue a CREATE SCHEMA command, as follows:

CREATE SCHEMA EXAMPLE **AUTHORIZATION** A1;

Now user account A1 can create tables under the schema called EXAMPLE. To continue our example, suppose that A1 creates the two base relations EMPLOYEE and DEPARTMENT shown in Figure 23.1; A1 is then the **owner** of these two relations and hence has *all the relation privileges* on each of them.

Next, suppose that account A1 wants to grant to account A2 the privilege to insert and delete tuples in both of these relations. However, A1 does not want A2 to be able to propagate these privileges to additional accounts. A1 can issue the following command:

GRANT INSERT, DELETE ON EMPLOYEE, DEPARTMENT **TO** A2;

EMPLOYEE

NAME	SSN	BDATE	ADDRESS	SEX	SALARY	DNO

DEPARTMENT

DNUMBER	DNAME	MGRSSN

FIGURE 23.1 Schemas for the two relations EMPLOYEE and DEPARTMENT.

Notice that the owner account A1 of a relation automatically has the GRANT OPTION, allowing it to grant privileges on the relation to other accounts. However, account A2 cannot grant INSERT and DELETE privileges on the EMPLOYEE and DEPARTMENT tables, because A2 was not given the GRANT OPTION in the preceding command.

Next, suppose that A1 wants to allow account A3 to retrieve information from either of the two tables and also to be able to propagate the SELECT privilege to other accounts. A1 can issue the following command:

```
GRANT SELECT ON EMPLOYEE, DEPARTMENT TO A3 WITH GRANT OPTION;
```

The clause WITH GRANT OPTION means that A3 can now propagate the privilege to other accounts by using GRANT. For example, A3 can grant the SELECT privilege on the EMPLOYEE relation to A4 by issuing the following command:

```
GRANT SELECT ON EMPLOYEE TO A4;
```

Notice that A4 cannot propagate the SELECT privilege to other accounts because the GRANT OPTION was not given to A4.

Now suppose that A1 decides to revoke the SELECT privilege on the EMPLOYEE relation from A3; A1 then can issue this command:

```
REVOKE SELECT ON EMPLOYEE FROM A3;
```

The DBMS must now automatically revoke the SELECT privilege on EMPLOYEE from A4, too, because A3 granted that privilege to A4 and A3 does not have the privilege any more.

Next, suppose that A1 wants to give back to A3 a limited capability to SELECT from the EMPLOYEE relation and wants to allow A3 to be able to propagate the privilege. The limitation is to retrieve only the NAME, BDATE, and ADDRESS attributes and only for the tuples with DNO = 5. A1 then can create the following view:

```
CREATE VIEW A3EMPLOYEE AS
SELECT NAME, BDATE, ADDRESS
FROM EMPLOYEE
WHERE DNO = 5;
```

After the view is created, A1 can grant SELECT on the view A3EMPLOYEE to A3 as follows:

```
GRANT SELECT ON A3EMPLOYEE TO A3 WITH GRANT OPTION;
```

Finally, suppose that A1 wants to allow A4 to update only the SALARY attribute of EMPLOYEE; A1 can then issue the following command:

```
GRANT UPDATE ON EMPLOYEE (SALARY) TO A4;
```

The UPDATE or INSERT privilege can specify particular attributes that may be updated or inserted in a relation. Other privileges (SELECT, DELETE) are not attribute specific, because this specificity can easily be controlled by creating the appropriate views that include only the desired attributes and granting the corresponding privileges on the views. However, because updating views is not always possible (see Chapter 9), the

UPDATE and INSERT privileges are given the option to specify particular attributes of a base relation that may be updated.

23.2.6 Specifying Limits on Propagation of Privileges

Techniques to limit the propagation of privileges have been developed, although they have not yet been implemented in most DBMSs and are not a part of SQL. Limiting **horizontal propagation** to an integer number i means that an account B given the GRANT OPTION can grant the privilege to at most i other accounts. **Vertical propagation** is more complicated; it limits the depth of the granting of privileges. Granting a privilege with a vertical propagation of zero is equivalent to granting the privilege with no GRANT OPTION. If account A grants a privilege to account B with the vertical propagation set to an integer number $j > 0$, this means that the account B has the GRANT OPTION on that privilege, but B can grant the privilege to other accounts only with a vertical propagation *less than* j. In effect, vertical propagation limits the sequence of GRANT OPTIONs that can be given from one account to the next based on a single original grant of the privilege.

We now briefly illustrate horizontal and vertical propagation limits—which are *not available* currently in SQL or other relational systems—with an example. Suppose that A1 grants SELECT to A2 on the EMPLOYEE relation with horizontal propagation equal to 1 and vertical propagation equal to 2. A2 can then grant SELECT to at most one account because the horizontal propagation limitation is set to 1. In addition, A2 cannot grant the privilege to another account except with vertical propagation set to 0 (no GRANT OPTION) or 1; this is because A2 must reduce the vertical propagation by at least 1 when passing the privilege to others. As this example shows, horizontal and vertical propagation techniques are designed to limit the propagation of privileges.

23.3 MANDATORY ACCESS CONTROL AND ROLE-BASED ACCESS CONTROL FOR MULTILEVEL SECURITY[2]

The discretionary access control technique of granting and revoking privileges on relations has traditionally been the main security mechanism for relational database systems. This is an all-or-nothing method: A user either has or does not have a certain privilege. In many applications, an *additional security policy* is needed that classifies data and users based on security classes. This approach, known as **mandatory access control,** would typically be *combined* with the discretionary access control mechanisms described in Section 23.2. It is important to note that most commercial DBMSs currently provide mechanisms only for discretionary access control. However, the need for multilevel security exists in

---

2. The contribution of Fariborz Farahmand to this and subsequent sections is appreciated.

government, military, and intelligence applications, as well as in many industrial and corporate applications.

Typical **security classes** are top secret (TS), secret (S), confidential (C), and unclassified (U), where TS is the highest level and U the lowest. Other more complex security classification schemes exist, in which the security classes are organized in a lattice. For simplicity, we will use the system with four security classification levels, where TS ≥ S ≥ C ≥ U, to illustrate our discussion. The commonly used model for multilevel security, known as the Bell-LaPadula model, classifies each **subject** (user, account, program) and **object** (relation, tuple, column, view, operation) into one of the security classifications TS, S, C, or U. We will refer to the **clearance** (classification) of a subject S as **class(S)** and to the **classification** of an object O as **class(O)**. Two restrictions are enforced on data access based on the subject/object classifications:

1. A subject S is not allowed read access to an object O unless class(S) ≥ class(O). This is known as the **simple security property.**

2. A subject S is not allowed to write an object O unless class(S) ≤ class(O). This is known as the **star property** (or ***-property**).

The first restriction is intuitive and enforces the obvious rule that no subject can read an object whose security classification is higher than the subject's security clearance. The second restriction is less intuitive. It prohibits a subject from writing an object at a lower security classification than the subject's security clearance. Violation of this rule would allow information to flow from higher to lower classifications, which violates a basic tenet of multilevel security. For example, a user (subject) with TS clearance may make a copy of an object with classification TS and then write it back as a new object with classification U, thus making it visible throughout the system.

To incorporate multilevel security notions into the relational database model, it is common to consider attribute values and tuples as data objects. Hence, each attribute A is associated with a **classification attribute** C in the schema, and each attribute value in a tuple is associated with a corresponding security classification. In addition, in some models, a **tuple classification** attribute TC is added to the relation attributes to provide a classification for each tuple as a whole. Hence, a **multilevel relation** schema R with n attributes would be represented as

$$R(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TC)$$

where each C_i represents the *classification attribute* associated with attribute A_i.

The value of the TC attribute in each tuple t—which is the *highest* of all attribute classification values within t—provides a general classification for the tuple itself, whereas each C_i provides a finer security classification for each attribute value within the tuple. The **apparent key** of a multilevel relation is the set of attributes that would have formed the primary key in a regular (single-level) relation. A multilevel relation will appear to contain different data to subjects (users) with different clearance levels. In some cases, it is possible to store a single tuple in the relation at a higher classification level and produce the corresponding tuples at a lower-level classification through a process known as **filtering.** In other cases, it is necessary to store two or more tuples at different classification levels with the same value for the *apparent key*. This leads to the concept of

polyinstantiation,[3] where several tuples can have the same apparent key value but have different attribute values for users at different classification levels.

We illustrate these concepts with the simple example of a multilevel relation shown in Figure 23.2a, where we display the classification attribute values next to each attribute's value. Assume that the Name attribute is the apparent key, and consider the query SELECT * FROM employee. A user with security clearance S would see the same relation shown in Figure 23.2a, since all tuple classifications are less than or equal to S. However, a user with security clearance C would not be allowed to see values for Salary of Brown and JobPerformance of Smith, since they have higher classification. The tuples would be *filtered* to appear as shown in Figure 23.2b, with Salary and JobPerformance

(a) EMPLOYEE

Name		Salary		JobPerformance		TC
Smith	U	40000	C	Fair	S	S
Brown	C	80000	S	Good	C	S

(b) EMPLOYEE

Name		Salary		JobPerformance		TC
Smith	U	40000	C	null	C	C
Brown	C	null	C	Good	C	C

(c) EMPLOYEE

Name		Salary		JobPerformance		TC
Smith	U	null	U	null	U	U

(d) EMPLOYEE

Name		Salary		JobPerformance		TC
Smith	U	40000	C	Fair	S	S
Smith	U	40000	C	Excellent	C	C
Brown	C	80000	S	Good	C	S

FIGURE 23.2 A multilevel relation to illustrate multilevel security. (a) The original EMPLOYEE tuples. (b) Appearance of EMPLOYEE after filtering for classification C users. (c) Appearance of EMPLOYEE after filtering for classification U users. (d) Polyinstantiation of the Smith tuple.

---

3. This is similar to the notion of having multiple versions in the database that represent the same real-world object.

appearing as null. For a user with security clearance U, the filtering allows only the Name attribute of Smith to appear, with all the other attributes appearing as null (Figure 23.2c). Thus, filtering introduces null values for attribute values whose security classification is higher than the user's security clearance.

In general, the **entity integrity** rule for multilevel relations states that all attributes that are members of the apparent key must not be null and must have the *same* security classification within each individual tuple. In addition, all other attribute values in the tuple must have a security classification greater than or equal to that of the apparent key. This constraint ensures that a user can see the key if the user is permitted to see any part of the tuple at all. Other integrity rules, called **null integrity** and **interinstance integrity,** informally ensure that if a tuple value at some security level can be filtered (derived) from a higher-classified tuple, then it is sufficient to store the higher-classified tuple in the multilevel relation.

To illustrate polyinstantiation further, suppose that a user with security clearance C tries to update the value of JobPerformance of Smith in Figure 23.2 to 'Excellent'; this corresponds to the following SQL update being issued:

```
UPDATE EMPLOYEE
SET JobPerformance = 'Excellent'
WHERE Name = 'Smith';
```

Since the view provided to users with security clearance C (see Figure 23.2b) permits such an update, the system should not reject it; otherwise, the user could infer that some nonnull value exists for the JobPerformance attribute of Smith rather than the null value that appears. This is an example of inferring information through what is known as a **covert channel,** which should not be permitted in highly secure systems (see Section 23.5.1). However, the user should not be allowed to overwrite the existing value of JobPerformance at the higher classification level. The solution is to create a **polyinstantiation** for the Smith tuple at the lower classification level C, as shown in Figure 23.2d. This is necessary since the new tuple cannot be filtered from the existing tuple at classification S.

The basic update operations of the relational model (insert, delete, update) must be modified to handle this and similar situations, but this aspect of the problem is outside the scope of our presentation. We refer the interested reader to the end-of-chapter bibliography for further details.

23.3.1 Comparing Discretionary Access Control and Mandatory Access Control

Discretionary Access Control (DAC) policies are characterized by a high degree of flexibility, which makes them suitable for a large variety of application domains. The main drawback of DAC models is their vulnerability to malicious attacks, such as Trojan horses embedded in application programs. The reason is that discretionary authorization models do not impose any control on how information is propagated and used once it has been accessed by users authorized to do so. By contrast, mandatory policies ensure a high

degree of protection—in a way, they prevent any illegal flow of information. They are therefore suitable for military types of applications, which require a high degree of protection. However, mandatory policies have the drawback of being too rigid in that they require a strict classification of subjects and objects into security levels, and therefore they are applicable to very few environments. In many practical situations, discretionary policies are preferred because they offer a better trade-off between security and applicability.

23.3.2 Role-Based Access Control

Role-based access control (RBAC) emerged rapidly in the 1990s as a proven technology for managing and enforcing security in large-scale enterprisewide systems. Its basic notion is that permissions are associated with roles, and users are assigned to appropriate roles. Roles can be created using the CREATE ROLE and DESTROY ROLE commands. The GRANT and REVOKE commands discussed under DAC can then be used to assign and revoke privileges from roles.

RBAC appears to be a viable alternative to traditional discretionary and mandatory access controls; it ensures that only authorized users are given access to certain data or resources. Users create sessions during which they may activate a subset of roles to which they belong. Each session can be assigned to many roles, but it maps to only one user or a single subject. Many DBMSs have allowed the concept of roles, where privileges can be assigned to roles.

Role hierarchy in RBAC is a natural way of organizing roles to reflect the organization's lines of authority and responsibility. By convention, junior roles at the bottom are connected to progressively senior roles as one moves up the hierarchy. The hierarchic diagrams are partial orders, so they are reflexive, transitive, and antisymmetric.

Another important consideration in RBAC systems is the possible temporal constraints that may exist on roles, such as the time and duration of role activations, and timed triggering of a role by an activation of another role. Using an RBAC model is a highly desirable goal for addressing the key security requirements of Web-based applications. Roles can be assigned to workflow tasks so that a user with any of the roles related to a task may be authorized to execute it and may play a certain role for a certain duration only.

RBAC models have several desirable features, such as flexibility, policy neutrality, better support for security management and administration, and other aspects that make them attractive candidates for developing secure Web-based applications. In contrast, DAC and mandatory access control (MAC) models lack capabilities needed to support the security requirements of emerging enterprises and Web-based applications. In addition, RBAC models can represent traditional DAC and MAC policies as well as user-defined or organization-specific policies. Thus, RBAC becomes a superset model that can in turn mimic the behavior of DAC and MAC systems. Furthermore, an RBAC model provides a natural mechanism for addressing the security issues related to the execution of tasks and workflows. Easier deployment over the Internet has been another reason for the success of RBAC models.

23.3.3 Access Control Policies for E-Commerce and the Web

Electronic commerce (**E-commerce**) environments are characterized by any transactions that are done electronically. They require elaborate access control policies that go beyond traditional DBMSs. In conventional database environments, access control is usually performed using a set of authorizations stated by security officers or users according to some security policies. Such a simple paradigm is not well suited for a dynamic environment like e-commerce. Furthermore, in an e-commerce environment the resources to be protected are not only traditional data but also knowledge and experience. Such peculiarities call for more flexibility in specifying access control policies. The access control mechanism must be flexible enough to support a wide spectrum of heterogeneous protection objects.

A second related requirement is the support for content-based access control. Content-based access control allows one to express access control policies that take the protection object content into account. In order to support content-based access control, access control policies must allow inclusion of conditions based on the object content.

A third requirement is related to the heterogeneity of subjects, which requires access control policies based on user characteristics and qualifications rather than on very specific and individual characteristics (e.g., user IDs). A possible solution, to better take into account user profiles in the formulation of access control policies, is to support the notion of credentials. A **credential** is a set of properties concerning a user that are relevant for security purposes (for example, age, position within an organization). For instance, by using credentials, one can simply formulate policies such as "Only permanent staff with 5 or more years of service can access documents related to the internals of the system."

It is believed that the XML language can play a key role in access control for e-commerce applications.[4] The reason is that XML is becoming the common representation language for document interchange over the Web, and is also becoming the language for e-commerce. Thus, on the one hand there is the need to make XML representations secure, by providing access control mechanisms specifically tailored to the protection of XML documents. On the other hand, access control information (that is, access control policies and user credentials) can be expressed using XML itself. The Directory Service Markup Language provides a foundation for this: a standard for communicating with the directory services that will be responsible for providing and authenticating user credentials. The uniform presentation of both protection objects and access control policies can be applied to policies and credentials themselves. For instance, some credential properties (such as the user name) may be accessible to everyone, whereas other properties may be visible only to a restricted class of users. Additionally, the use of an XML-based language for specifying credentials and access control policies facilitates secure credential submission and export of access control policies.

---

4. See Thuraisingham et al. (2001).

23.4 INTRODUCTION TO STATISTICAL DATABASE SECURITY

Statistical databases are used mainly to produce statistics on various populations. The database may contain confidential data on individuals, which should be protected from user access. However, users are permitted to retrieve statistical information on the populations, such as averages, sums, counts, maximums, minimums, and standard deviations. The techniques that have been developed to protect the privacy of individual information are outside the scope of this book. We will only illustrate the problem with a very simple example, which refers to the relation shown in Figure 23.3. This is a PERSON relation with the attributes NAME, SSN, INCOME, ADDRESS, CITY, STATE, ZIP, SEX, and LAST_DEGREE.

A **population** is a set of tuples of a relation (table) that satisfy some selection condition. Hence each selection condition on the PERSON relation will specify a particular population of PERSON tuples. For example, the condition SEX = 'M' specifies the male population; the condition ((SEX = 'F') AND (LAST_DEGREE = 'M.S.' OR LAST_DEGREE = 'PH.D.')) specifies the female population that has an M.S. or PH.D. degree as their highest degree; and the condition CITY = 'Houston' specifies the population that lives in Houston.

Statistical queries involve applying statistical functions to a population of tuples. For example, we may want to retrieve the number of individuals in a population or the average income in the population. However, statistical users are not allowed to retrieve individual data, such as the income of a specific person. **Statistical database security** techniques must prohibit the retrieval of individual data. This can be achieved by prohibiting queries that retrieve attribute values and by allowing only queries that involve statistical aggregate functions such as COUNT, SUM, MIN, MAX, AVERAGE, and STANDARD DEVIATION. Such queries are sometimes called **statistical queries.**

It is the responsibility of a database management system to ensure the confidentiality of information about individuals, while still providing useful statistical summaries of data about those individuals to users. Provision of **privacy protection** of users in a statistical database is paramount; its violation is illustrated in the following example.

In some cases it is possible to **infer** the values of individual tuples from a sequence of statistical queries. This is particularly true when the conditions result in a population consisting of a small number of tuples. As an illustration, consider the following two statistical queries:

> Q1: **SELECT COUNT** (*) **FROM** PERSON
> **WHERE** <CONDITION>;
> Q2: **SELECT AVG** (INCOME) **FROM** PERSON
> **WHERE** <CONDITION>;

PERSON

NAME	SSN	INCOME	ADDRESS	CITY	STATE	ZIP	SEX	LAST_DEGREE

FIGURE 23.3 The PERSON relation schema for illustrating statistical database security.

Now suppose that we are interested in finding the SALARY of 'Jane Smith', and we know that she has a PH.D. degree and that she lives in the city of Bellaire, Texas. We issue the statistical query Q1 with the following condition:

```
(LAST_DEGREE='PH.D.' AND SEX='F' AND CITY='Bellaire' AND
STATE='Texas')
```

If we get a result of 1 for this query, we can issue Q2 with the same condition and find the income of Jane Smith. Even if the result of Q1 on the preceding condition is not 1 but is a small number—say, 2 or 3—we can issue statistical queries using the functions MAX, MIN, and AVERAGE to identify the possible range of values for the income of Jane Smith.

The possibility of inferring individual information from statistical queries is reduced if no statistical queries are permitted whenever the number of tuples in the population specified by the selection condition falls below some threshold. Another technique for prohibiting retrieval of individual information is to prohibit sequences of queries that refer repeatedly to the same population of tuples. It is also possible to introduce slight inaccuracies or "noise" into the results of statistical queries deliberately, to make it difficult to deduce individual information from the results. Another technique is partitioning of the database. Partitioning implies that records are stored in groups of some minimum size; queries can refer to any complete group or set of groups, but never to subsets of records within a group. The interested reader is referred to the bibliography for a discussion of these techniques.

23.5 INTRODUCTION TO FLOW CONTROL

Flow control regulates the distribution or flow of information among accessible objects. A flow between object X and object Y occurs when a program reads values from X and writes values into Y. **Flow controls** check that information contained in some objects does not flow explicitly or implicitly into less protected objects. Thus, a user cannot get indirectly in Y what he or she cannot get directly from X. Active flow control began in the early 1970s. Most flow controls employ some concept of security class; the transfer of information from a sender to a receiver is allowed only if the receiver's security class is at least as privileged as the sender's. Examples of a flow control include preventing a service program from leaking a customer's confidential data, and blocking the transmission of secret military data to an unknown classified user.

A **flow policy** specifies the channels along which information is allowed to move. The simplest flow policy specifies just two classes of information: confidential (C) and nonconfidential (N), and allows all flows except those from class C to class N. This policy can solve the confinement problem that arises when a service program handles data such as customer information, some of which may be confidential. For example, an income-tax computing service might be allowed to retain the customer's address and the bill for services rendered, but not the customer's income or deductions.

Access control mechanisms are responsible for checking users' authorizations for resource access: Only granted operations are executed. Flow controls can be enforced by

an extended access control mechanism, which involves assigning a security class (usually called the *clearance*) to each running program. The program is allowed to read a particular memory segment only if its security class is as high as that of the segment. It is allowed to write in a segment only if its class is as low as that of the segment. This automatically ensures that no information transmitted by the person can move from a higher to a lower class. For example, a military program with a secret clearance can read only from objects that are unclassified and confidential and it can only write into objects that are secret or top secret.

Two types of flow can be distinguished: *explicit flows*, occurring as a consequence of assignment instructions, such as $Y := f(X_1, X_n)$; and *implicit flows* generated by conditional instructions, such as if $f(X_{m+1}, \ldots, X_n)$ then $y := f(X_1, X_m)$.

Flow control mechanisms must verify that only authorized flows, both explicit and implicit, are executed. A set of rules must be satisfied to ensure secure information flows. Rules can be expressed using flow relations among classes and assigned to information, stating the authorized flows within a system. (An information flow from A to B occurs when information associated with A affects the value of information associated with B. The flow results from operations that cause information transfer from one object to another.) These relations can define, for a class, the set of classes where information (classified in that class) can flow, or can state the specific relations to be verified between two classes to allow information flow from one to the other. In general, flow control mechanisms implement the controls by assigning a label to each object and by specifying the security class of the object. Labels are then used to verify the flow relations defined in the model.

23.5.1 Covert Channels

A covert channel allows a transfer of information that violates the security or the policy. Specifically, a **covert channel** allows information to pass from a higher classification level to a lower classification level through improper means. Covert channels can be classified into two broad categories: storage and timing channels. The distinguishing feature between the two is that in a **timing channel** the information is conveyed by the timing of events or processes, whereas **storage channels** do not require any temporal synchronization, in that information is conveyed by accessing system information or what is otherwise inaccessible to the user.

In a simple example of a covert channel, consider a distributed database system in which two nodes have user security levels of secret (S) and unclassified (U). In order for a transaction to commit, both nodes must agree to commit. They mutually can only do operations that are consistent with the *-property, which states that in any transaction, the S site cannot write or pass information to the U site. However, if these two sites collude to set up a covert channel between them, a transaction involving secret data may be committed unconditionally by the U site, but the S site may do so in some predefined agreed-upon way so that certain information may be passed on from the S site to the U site, violating the *-property. This may be achieved where the transaction runs repeatedly, but the actions taken by the S site implicitly convey information to the U site. Measures such as

locking that we discussed in Chapters 17 and 18 prevent concurrent writing of the information by users with different security levels into the same objects, preventing the storage-type covert channels. Operating systems and distributed databases provide control over the multiprogramming of operations that allow a sharing of resources without the possibility of encroachment of one program or process into another's memory or other resources in the system, thus preventing timing-oriented covert channels. In general, covert channels are not a major problem in well-implemented robust database implementations. However, certain schemes may be contrived by clever users that implicitly transfer information.

Some security experts believe that one way to avoid covert channels is for programmers to not actually gain access to sensitive data that a program is supposed to process after the program has been put into operation. For example, a programmer for a bank has no need to access the names or balances in depositors' accounts. Programmers for brokerage firms do not need to know what buy and sell orders exist for clients. During program testing, access to a form of real data or some sample test data may be justifiable, but not after the program has been accepted for regular use.

23.6 ENCRYPTION AND PUBLIC KEY INFRASTRUCTURES

The previous methods of access and flow control, despite being strong countermeasures, may not be able to protect databases from some threats. Suppose we communicate data, but our data falls into the hands of some nonlegitimate user. In this situation, by using encryption we can disguise the message so that even if the transmission is diverted, the message will not be revealed. Encryption is a means of maintaining secure data in an insecure environment. Encryption consists of applying an **encryption algorithm** to data using some prespecified **encryption key**. The resulting data has to be **decrypted** using a **decryption key** to recover the original data.

23.6.1 The Data and Advanced Encryption Standards

The Data Encryption Standard (DES) is a system developed by the U.S. government for use by the general public. It has been widely accepted as a cryptographic standard both in the United States and abroad. DES can provide end-to-end encryption on the channel between the sender A and receiver B. The DES algorithm is a careful and complex combination of two of the fundamental building blocks of encryption: substitution and permutation (transposition). The algorithm derives its strength from repeated application of these two techniques for a total of 16 cycles. Plaintext (the original form of the message) is encrypted as blocks of 64 bits. Although the key is 64 bits long, in effect the key can be any 56-bit number. After questioning the adequacy of DES, the National Institute of Standards (NIST) introduced the Advanced Encryption Standards (AES). This algorithm has a block size of 128 bits, compared with DES's 56-block size, and can use keys of

128, 192, or 256 bits, compared with DES's 56-bit key. AES introduces more possible keys, compared with DES, and thus takes a much longer time to crack.

23.6.2 Public Key Encryption

In 1976 Diffie and Hellman proposed a new kind of cryptosystem, which they called **public key encryption.** Public key algorithms are based on mathematical functions rather than operations on bit patterns. They also involve the use of two separate keys, in contrast to conventional encryption, which uses only one key. The use of two keys can have profound consequences in the areas of confidentiality, key distribution, and authentication. The two keys used for public key encryption are referred to as the **public key** and the **private key.** Invariably, the private key is kept secret, but it is referred to as a *private key* rather than a *secret key* (the key used in conventional encryption) to avoid confusion with conventional encryption.

A public key encryption scheme, or infrastructure, has six ingredients:

1. *Plaintext:* This is the data or readable message that is fed into the algorithm as input.

2. *Encryption algorithm:* The encryption algorithm performs various transformations on the plaintext.

3 and 4. *Public and private keys:* These are a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the encryption algorithm depend on the public or private key that is provided as input.

5. *Ciphertext:* This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.

6. *Decryption algorithm:* This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

As the name suggests, the public key of the pair is made public for others to use, whereas the private key is known only to its owner. A general-purpose public key cryptographic algorithm relies on one key for encryption and a different but related one for decryption. The essential steps are as follows:

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.

2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private.

3. If a sender wishes to send a private message to a receiver, the sender encrypts the message using the receiver's public key.

4. When the receiver receives the message, he or she decrypts it using the receiver's private key. No other recipient can decrypt the message because only the receiver knows his or her private key.

The RSA Public Key Encryption Algorithm. One of the first public key schemes was introduced in 1978 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and is named after them as the RSA scheme. The RSA scheme has since then reigned supreme as the most widely accepted and implemented approach to public key encryption. The RSA encryption algorithm incorporates results from number theory, combined with the difficulty of determining the prime factors of a target. The RSA algorithm also operates with modular arithmetic—mod n.

Two keys, d and e, are used for decryption and encryption. An important property is that they can be interchanged. n is chosen as a large integer that is a product of two large distinct prime numbers, a and b. The encryption key e is a randomly chosen number between 1 and n that is relatively prime to $(a - 1) \times (b - 1)$. The plaintext block P is encrypted as P^e mod n. Because the exponentiation is performed mod n, factoring P^e to uncover the encrypted plaintext is difficult. However, the decrypting key d is carefully chosen so that $(P^e)^d$ mod $n = P$. The decryption key d can be computed from the condition that $d \times e = 1$ mod $((a - 1) \times (b - 1))$. Thus, the legitimate receiver who knows d simply computes $(P^e)^d$ mod $n = P$ and recovers P without having to factor P^e.

23.6.3 Digital Signatures

A digital signature is an example of using encryption techniques to provide authentication services in electronic commerce applications. Like a handwritten signature, a **digital signature** is a means of associating a mark unique to an individual with a body of text. The mark should be unforgettable, meaning that others should be able to check that the signature does come from the originator.

A digital signature consists of a string of symbols. If a person's digital signature were always the same for each message, then one could easily counterfeit it by simply copying the string of symbols. Thus, signatures must be different for each use. This can be achieved by making each digital signature a function of the message that it is signing, together with a time stamp. To be unique to each signer and counterfeitproof, each digital signature must also depend on some secret number that is unique to the signer. Thus, in general, a counterfeitproof digital signature must depend on the message and a unique secret number of the signer. The verifier of the signature, however, should not need to know any secret number. Public key techniques are the best means of creating digital signatures with these properties.

23.7 SUMMARY

This chapter discussed several techniques for enforcing security in database systems. It presented the different threats to databases in terms of loss of integrity, availability, and confidentiality. The four types of countermeasures to deal with these problems are access control, inference control, flow control, and encryption. We discussed all of these measures in this chapter.

Security enforcement deals with controlling access to the database system as a whole and controlling authorization to access specific portions of a database. The former is usually done by assigning accounts with passwords to users. The latter can be accomplished by using a system of granting and revoking privileges to individual accounts for accessing specific parts of the database. This approach is generally referred to as discretionary access control. We presented some SQL commands for granting and revoking privileges, and we illustrated their use with examples. Then we gave an overview of mandatory access control mechanisms that enforce multilevel security. These require the classifications of users and data values into security classes and enforce the rules that prohibit flow of information from higher to lower security levels. Some of the key concepts underlying the multilevel relational model, including filtering and polyinstantiation, were presented. Role-based access control was introduced, which assigns privileges based on roles that users play. We briefly discussed the problem of controlling access to statistical databases to protect the privacy of individual information while concurrently providing statistical access to populations of records. The issues related to flow control and the problems associated with covert channels were discussed next. Finally, we covered the area of encryption of data, including the public key infrastructure and digital signatures.

Review Questions

23.1. Discuss what is meant by each of the following terms: *database authorization*, *access control*, *data encryption*, *privileged (system) account*, *database audit*, *audit trail*.
   a. Discuss the types of privileges at the account level and those at the relation level.

23.2. Which account is designated as the owner of a relation? What privileges does the owner of a relation have?

23.3. How is the view mechanism used as an authorization mechanism?

23.4. What is meant by granting a privilege?

23.5. What is meant by revoking a privilege?

23.6. Discuss the system of propagation of privileges and the restraints imposed by horizontal and vertical propagation limits.

23.7. List the types of privileges available in SQL.

23.8. What is the difference between *discretionary* and *mandatory* access control?

23.9. What are the typical security classifications? Discuss the simple security property and the *-property, and explain the justification behind these rules for enforcing multilevel security.

23.10. Describe the multilevel relational data model. Define the following terms: *apparent key*, *polyinstantiation*, *filtering*.

23.11. What are the relative merits of using DAC or MAC?

23.12. What is role-based access control? In what ways is it superior to DAC and MAC?

23.13. What is a statistical database? Discuss the problem of statistical database security.

23.14. How is privacy related to statistical database security? What meaures can be taken to ensure some degree of privacy in statistical databases?

23.15. What is flow control as a security measure? What types of flow control exist?

23.16. What are covert channels? Give an example of a covert channel.

23.17. What is the goal of encryption? What process is involved in encrypting data and then recovering it at the other end?

23.18. Give an example of an encryption algorithm and explain how it works.

23.19. Repeat the previous question for the popular RSA algorithm.

23.20. What is the public key infrastructure scheme? How does it provide security?

23.21. What are digital signatures? How do they work?

Exercises

23.22. Consider the relational database schema of Figure 5.5. Suppose that all the relations were created by (and hence are owned by) user X, who wants to grant the following privileges to user accounts A, B, C, D, and E:

a. Account A can retrieve or modify any relation except dependent and can grant any of these privileges to other users.

b. Account B can retrieve all the attributes of employee and department except for salary, mgrssn, and mgrstartdate.

c. Account C can retrieve or modify works_on but can only retrieve the fname, minit, lname, and ssn attributes of employee and the pname and pnumber attributes of project.

d. Account D can retrieve any attribute of employee or dependent and can modify dependent.

e. Account E can retrieve any attribute of employee but only for employee tuples that have dno = 3.

f. Write SQL statements to grant these privileges. Use views where appropriate.

23.23. Suppose that privilege (a) of Exercise 23.1 is to be given with GRANT OPTION but only so that account A can grant it to at most five accounts, and each of these accounts can propagate the privilege to other accounts but *without* the GRANT OPTION privilege. What would the horizontal and vertical propagation limits be in this case?

23.24. Consider the relation shown in Figure 23.2d. How would it appear to a user with classification U? Suppose a classification U user tries to update the salary of 'Smith' to $50,000; what would be the result of this action?

Selected Bibliography

Authorization based on granting and revoking privileges was proposed for the SYSTEM R experimental DBMS and is presented in Griffiths and Wade (1976). Several books discuss security in databases and computer systems in general, including the books by Leiss (1982a) and Fernandez et al. (1981). Denning and Denning (1979) is a tutorial paper on data security.

Many papers discuss different techniques for the design and protection of statistical databases. These include McLeish (1989), Chin and Ozsoyoglu (1981), Leiss (1982), Wong (1984), and Denning (1980). Ghosh (1984) discusses the use of statistical databases for

quality control. There are also many papers discussing cryptography and data encryption, including Diffie and Hellman (1979), Rivest et al. (1978), Akl (1983), Pfleeger (1997), Omura et al. (1990), and Stalling (2000).

Multilevel security is discussed in Jajodia and Sandhu (1991), Denning et al. (1987), Smith and Winslett (1992), Stachour and Thuraisingham (1990), Lunt et al. (1990), and Bertino et al. (2001). Overviews of research issues in database security are given by Lunt and Fernandez (1990), Jajodia and Sandhu (1991), Bertino et al. (1998), Castano et al. (1995), and Thuraisingham et al. (2001). The effects of multilevel security on concurrency control are discussed in Atluri et al. (1997). Security in next-generation, semantic, and object-oriented databases is discussed in Rabbiti et al. (1991), Jajodia and Kogan (1990), and Smith (1990). Oh (1999) presents a model for both discretionary and mandatory security. Security models for Web-based applications and role-based access control are discussed in Joshi et al. (2001). Security issues for managers in the context of e-commerce applications and the need for risk assessment models for selection of appropriate security countermeasures are discussed in Farahmand et al. (2002).