Chapter 10: Database Security and Authorization

10.1 Introduction to Database Security Issues

10.1.1 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 results 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 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*.

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.

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 log-in process by the DBMS. We discuss access control techniques in Section 10.1.3.

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 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 10.4.

A fourth 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** by 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. We will not discuss encryption algorithms here.

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 one of the references at the end of this chapter for a more comprehensive discussion.

10.1.2 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. 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 them to access 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 authorizations, and action 4 is used to control *mandatory* authorization.

10.1.3 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 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 **log-in 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 Chapter 19 and Chapter 21 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 on-line 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.**

10.2 Discretionary Access Control Based on Granting/Revoking of 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. Many current relational DBMSs use some variation of this technique. The main idea is to include additional statements in the query language that allow the DBA and selected users to grant and revoke privileges.

10.2.1 Types of Discretionary Privileges

In SQL2, the concept of **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, we 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 follows 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. 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.

10.2.2 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 10.2.5.

10.2.3 Revoking Privileges

In some cases it is desirable to grant some 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 10.2.5.

10.2.4 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, *A*4 may receive a certain UPDATE *R* privilege from *both A*2 and *A*3. In such a case, if *A*2 revokes this privilege from *A*4, *A*4 will still continue to have the privilege by virtue of having been granted it from *A*3. If *A*3 later revokes the privilege from *A*4, *A*4 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.

10.5 An Example

Suppose that the DBA creates four accounts—*A*1, *A*2, *A*3, and *A*4—and wants only *A*1 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 *A*1 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 *A*1 can create tables under the schema called EXAMPLE. To continue our example, suppose that *A*1 creates the two base relations EMPLOYEE and DEPARTMENT shown in Figure 10.1; then *A*1 is the **owner** of these two relations and hence has *all the relation privileges* on each of them.

Next, suppose that account *A*1 wants to grant to account *A*2 the privilege to insert and delete tuples in both of these relations. However, *A*1 does not want *A*2 to be able to propagate these privileges to additional accounts. Then *A*1 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 10.1 Schemas for the two relations EMPLOYEE and DEPARTMENT.

Notice that the owner account *A*1 of a relation automatically has the GRANT OPTION, allowing it to grant privileges on the relation to other accounts. However, account *A*2 cannot grant INSERT and DELETE privileges on the EMPLOYEE and DEPARTMENT tables, because *A*2 was not given the GRANT OPTION in the preceding command.

Next, suppose that *A*1 wants to allow account *A*3 to retrieve information from either of the two tables and also to be able to propagate the SELECT privilege to other accounts. Then *A*1 can issue the following command:

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

The clause WITH GRANT OPTION means that *A*3 can now propagate the privilege to other accounts by using GRANT. For example, *A*3 can grant the SELECT privilege on the EMPLOYEE relation to *A*4 by issuing the following command:
GRANT SELECT **ON** EMPLOYEE **TO** A4;

Notice that *A*4 cannot propagate the SELECT privilege to other accounts because the GRANT OPTION was not given to *A*4. Now suppose that *A*1 decides to revoke the SELECT privilege on the EMPLOYEE relation from *A*3; *A*1 then can issue this command: **REVOKE** SELECT **ON** EMPLOYEE **FROM** A3;

The DBMS must now automatically revoke the SELECT privilege on EMPLOYEE from *A*4, too, because *A*3 granted that privilege to *A*4 and *A*3 does not have the privilege any more. Next, suppose that *A*1 wants to give back to *A*3 a limited capability to SELECT from the EMPLOYEE relation and wants to allow *A*3 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. *A*1 then can create the following view:

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

After the view is created, *A*1 can grant SELECT on the view A3EMPLOYEE to *A*3 as follows:

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

Finally, suppose that *A*1 wants to allow *A*4 to update only the SALARY attribute of EMPLOYEE; *A*1 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, as 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, the UPDATE and INSERT privileges are given the option to specify particular attributes of a base relation that may be updated.

10.2.6 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 vertical propagation of zero is equivalent to granting the privilege with *no* GRANT OPTION. If account *A* grants a privilege to account *B* with 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 *A*1 grants SELECT to *A*2 on the EMPLOYEE relation with horizontal propagation = 1 and vertical propagation = 2. *A*2 can then grant SELECT to at most one account because the horizontal propagation limitation is set to 1. In addition, *A*2 cannot grant the privilege to another account except with vertical propagation = 0 (no GRANT OPTION) or 1; this is because *A*2 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.

10.3 Mandatory Access Control for Multilevel Security

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 22.2. 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 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 $TS \geq S \geq C \geq U$, 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 $class(S) \geq class(O)$. This is known as the **simple security property.**
2. A subject S is not allowed to write an object O unless $class(S) \leq class(O)$. 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 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 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,** 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 10.2(a), 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 10.2(a), 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 10.2(b), with `Salary` and `JobPerformance` *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 10.2c). Thus filtering introduces null values for attribute values whose security classification is higher than the user's security clearance.

(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 10.2 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.

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 10.2 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 10.2b) 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. 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 10.2(d). 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.

10.3.1 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 tom 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 and they are only applicable in limited environments. In many practical situations, discretionary policies are preferred because they offer a better trade-off between security and applicability.

10.3.2 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 time and duration of role activations, and timed triggering of a role by an activation of another role. Using an RBAC model is 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, discretionary access control (DAC) and mandatory access control (MAC) models lack capabilities needed to support the security requirements 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.

10.3.3 Access Control Policies for E-Commerce and the Web

E-Commerce environments require elaborate access control policies that go beyond traditional DBMSs. In an e-commerce environment the resources to be protected are not only traditional data but also knowledge and experience. The access control mechanism should be flexible enough to support a wide spectrum of heterogeneous protection objects.

A related requirement is the support for content-based access-control. Control-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.

Another 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.

10.4 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 10.3. 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 controlled 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.**

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 two statistical queries:

Q1: **SELECT COUNT** (*) **FROM** PERSON
WHERE,condition.;

Q2: **SELECT AVG** (INCOME) **FROM** PERSON
WHERE,condition.;

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. The interested reader is referred to the bibliography for a discussion of these techniques.