**Unit-6**

**Introduction to Database Security Issues**

**1. Types of Security**

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

        Various legal and ethical issues regarding the right to access certain information—for example, some information may be deemed to be private and can-not be accessed legally by unauthorized organizations or 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 can result in the loss or degradation of some or all of the following commonly accepted security goals: integrity, avail-ability, and confidentiality.

        **Loss of integrity.**Database integrity refers to the requirement that information be protected from improper modification. Modification of data includes creation, insertion, updating, 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 availableto a human user or a program to which they have a legitimate right.

**Loss of confidentiality.**Database confidentiality refers to the protection ofdata 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, it is common to implement *four* *kinds of control measures*: 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 tousers, 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 multilevelsecurity by classifying the data and users into various security classes (or lev-\els) and then implementing the appropriate security policy of the organization. For example, a typical security policy is to permit users at a certain classification (or clearance) 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 organizational roles.

We discuss discretionary security in Section 24.2 and mandatory and role-based security in Section 24.3.

**2. Control Measures**

Four main control measures are used to provide security of data in databases:

* Access control
* Inference control
* Flow control
* Data encryption

A security problem common to 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, called **access control**, is handled by creating user accounts and passwords to control the login process by the DBMS. We discuss access control techniques in Section 24.1.3.

S**tatistical databases** are 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, household size, 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 about specific individuals. Security for statistical databases must ensure that information about 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 24.4. The corresponding control measures 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 discussed in Section 24.6. Channels that are pathways for information to flow implicitly in ways that vio-late the security policy of an organization are called **covert channels**. We briefly dis-cuss some issues related to covert channels in Section 24.6.1.

A final control measure is **data encryption**, which is used to protect sensitive data (such as credit card numbers) that is 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 24.7 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 comprehensive discussion of security in computer systems and databases is out-side 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 dis-cussed in the Selected Bibliography at the end of this chapter for a more comprehensive discussion.

**3. 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:

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

        **Privilege granting.**This action permits the DBA to grant certain privilegesto certain accounts.

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

        **Security level assignment.**This action consists of assigning user accounts tothe appropriate security clearance 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.

**4. Access Control, 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 users and are required to log in to the database (see Chapter 13).

It is straightforward to keep track of database users and their accounts and pass-words by creating an encrypted table or file with 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 computer or device from which the user logged in. All operations applied from that computer or device 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 determine which user did the tampering.

To keep a record of all updates applied to the database and of particular users who applied each update, we can modify the *system log.* Recall from Chapters 21 and 23 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 computer or device 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 the 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**.

**5. Sensitive Data and Types of Disclosures**

**Sensitivity of data**is a measure of the importance assigned to the data by its owner,for the purpose of denoting its need for protection. Some databases contain only sensitive data while other databases may contain no sensitive data at all. Handling databases that fall at these two extremes is relatively easy, because these can be cov-ered by access control, which is explained in the next section. The situation becomes tricky when some of the data is sensitive while other data is not.

Several factors can cause data to be classified as sensitive:

        **Inherently sensitive.**The value of the data itself may be so revealing or con-fidential that it becomes sensitive—for example, a person’s salary or that a patient has HIV/AIDS.

        **From a sensitive source.**The source of the data may indicate a need forsecrecy—for example, an informer whose identity must be kept secret.

        **Declared sensitive.**The owner of the data may have explicitly declared it assensitive.

        **A sensitive attribute or sensitive record.**The particular attribute or recordmay have been declared sensitive—for example, the salary attribute of an employee or the salary history record in a personnel database.

        **Sensitive in relation to previously disclosed data.**Some data may not besensitive by itself but will become sensitive in the presence of some other data—for example, the exact latitude and longitude information for a loca-tion where some previously recorded event happened that was later deemed sensitive.

It is the responsibility of the database administrator and security administrator to collectively enforce the security policies of an organization. This dictates whether access should be permitted to a certain database attribute (also known as a *table col-umn*or a*data element*) or not for individual users or for categories of users. Severalfactors need to be considered before deciding whether it is safe to reveal the data. The three most important factors are data availability, access acceptability, and authenticity assurance.

**Data availability.**If a user is updating a field, then this field becomes inaccessible and other users should not be able to view this data. This blocking is only temporary and only to ensure that no user sees any inaccurate data. This is typically handled by the concurrency control mechanism (see Chapter 22).

        **Access acceptability.**Data should only be revealed to authorized users. Adatabase administrator may also deny access to a user request even if the request does not directly access a sensitive data item, on the grounds that the requested data may reveal information about the sensitive data that the user is not authorized to have.

        **Authenticity assurance.**Before granting access, certain external characteristics about the user may also be considered. For example, a user may only be permitted access during working hours. The system may track previous queries to ensure that a combination of queries does not reveal sensitive data. The latter is particularly relevant to statistical database queries (see Section 24.5).

The term *precision*, when used in the security area, refers to allowing as much as possible of the data to be available, subject to protecting exactly the subset of data that is sensitive. The definitions of *security* versus *precision* are as follows:

        **Security:**Means of ensuring that data is kept safe from corruption and thataccess to it is suitably controlled. To provide security means to disclose only nonsensitive data, and reject any query that references a sensitive field.

        **Precision**: To protect all sensitive data while disclosing as much nonsensitivedata as possible.

The ideal combination is to maintain perfect security with maximum precision. If we want to maintain security, some sacrifice has to be made with precision. Hence there is typically a tradeoff between security and precision.

**6. Relationship between Information Security versus Information Privacy**

The rapid advancement of the use of information technology (IT) in industry, government, and academia raises challenging questions and problems regarding the protection and use of personal information. Questions of *who* has *what* rights to information about individuals for *which* purposes become more important as we move toward a world in which it is technically possible to know just about anything about anyone.

Deciding how to design privacy considerations in technology for the future includes philosophical, legal, and practical dimensions. There is a considerable overlap between issues related to access to resources (security) and issues related to appropriate use of information (privacy). We now define the difference between *security* versus *privacy*.

**Security**in information technology refers to many aspects of protecting a systemfrom unauthorized use, including authentication of users, information encryption, access control, firewall policies, and intrusion detection. For our purposes here, we will limit our treatment of security to the concepts associated with how well a system can protect access to information it contains. The concept of **privacy** goes beyond security. Privacy examines how well the use of personal information that the system acquires about a user conforms to the explicit or implicit assumptions regarding that use. From an end user perspective, privacy can be considered from two different perspectives: *preventing storage* of personal information versus *ensuring appropriate use*of personal information.

For the purposes of this chapter, a simple but useful definition of **privacy** is *the ability of individuals to control the terms under which their personal information is acquired and used*. In summary, security involves technology to ensure that information is appropriately protected. Security is a required building block for privacy to exist. Privacy involves mechanisms to support compliance with some basic principles and other explicitly stated policies. One basic principle is that people should be informed about information collection, told in advance what will be done with their information, and given a reasonable opportunity to approve of such use of the information. A related concept**, trust**, relates to both security and privacy, and is seen as increasing when it is perceived that both security and privacy are provided for.

**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 Chapters 4 and 5). Many current relational DBMSs use some variation of this tech-nique. The main idea is to include statements in the query language that allow the DBA and selected users to grant and revoke privileges.

**1. Types of Discretionary Privileges**

In SQL2 and later versions, 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 databasebased 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 privilegesthat each account holds independently of the relations in the database.

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

        **References privilege on *R.***This gives the account the capability to*reference*(or refer to) a 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 the SELECT privilege on *all relations*involved in the view definition in order to specify the query that correspondsto the view.

**2. Specifying Privileges through the Use of 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 will illustrate this discussion with the example given in Section 24.2.5.

**3. Revoking of 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 24.2.5.

**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 the 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 privi-leges must keep track of how all the privileges were granted so that revoking of priv-ileges can be done correctly and completely.

**5. An Example to Illustrate Granting and Revoking of Privileges**

Suppose that the DBA creates four accounts—A1, A2, A3, and A4—and wants only A1 to be able to create base relations. To do this, 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 imple-mentation 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;

User account A1 can now create tables under the schema called EXAMPLE. To con-tinue our example, suppose that A1 creates the two base relations EMPLOYEE and DEPARTMENT shown in Figure 24.1; 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 com-mand:

GRANT INSERT, DELETE ON EMPLOYEE, DEPARTMENT TO A2;

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 revoke the SELECT privilege on EMPLOYEE from A3, and it must also *automatically revoke* the SELECT privilege on EMPLOYEE from A4. This is because A3 granted that privilege to A4, but 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 and INSERT privileges can specify particular attributes that may be updated or inserted in a relation. Other privileges (SELECT, DELETE) are not attrib-ute specific, because this specificity can easily be controlled by creating the appro-priate 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 5), the UPDATE and INSERT privileges are given the option to specify the particular attributes of a base relation that may be updated.

**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 ofprivileges. 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 briefly illustrate horizontal and vertical propagation limits—which are *not* *available*currently in SQL or other relational systems—with an example. Supposethat 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. Additionally, 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. In addition, the horizontal propagation must be less than or equal to the originally granted hor-izontal propagation. For example, if account *A* grants a privilege to account *B* with the horizontal propagation set to an integer number *j* > 0, this means that *B* can grant the privilege to other accounts only with a horizontal propagation *less than or* *equal to j.*As this example shows, horizontal and vertical propagation techniques aredesigned to limit the depth and breadth of propagation of privileges.

**Mandatory Access Control and Role-Based 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 (MAC)**, would typically be*combined*with the discretionary accesscontrol mechanisms described in Section 24.2. 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. Some DBMS vendors—for example, Oracle—have released special versions of their RDBMSs that incorporate mandatory access control for government use.

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. The model we describe here is known as the *multilevel model*, because it allows classifications at multiple security levels. A **multilevel relation** schema *R* with *n* attributes would be represented as:

*R*(*A*1,*C*1,*A*2,*C*2, ...,*An*,*Cn*,*TC*)

where each *Ci* represents the *classification attribute* associated with attribute *Ai*.

The value of the tuple classification attribute *TC* in each tuple *t—*which is the *highest*of all attribute classification values within*t—*provides a general classification for the tuple itself. Each attribute classification *Ci* provides a finer security classification for each attribute value within the tuple. The value of *TC* in each tuple *t* is the *highest* of all attribute classification values *Ci* within *t*.

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 clearance levels.

We illustrate these concepts with the simple example of a multilevel relation shown in Figure 24.2(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 24.2(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 the values for Salary of ‘Brown’ and Job\_performance of ‘Smith’, since they have higher classification. The tuples would be *filtered* to appear as shown in Figure 24.2(b), with Salary and Job\_performance *appearing as null.* For a user with security clearance *U*, the filtering allows only theNameattribute of ‘Smith’ to appear, with all the other



attributes appearing as null (Figure 24.2(c)). 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. Additionally, 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. 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 Job\_performance of ‘Smith’ in Figure 24.2 to ‘Excellent’; this corresponds to the following SQL update being submitted by that user:

UPDATE  EMPLOYEE SET  Job\_performance = ‘Excellent’ WHERE  Name = ‘Smith’;

Since the view provided to users with security clearance *C* (see Figure 24.2(b)) per-mits such an update, the system should not reject it; otherwise, the user could *infer* that some nonnull value exists for the Job\_performance 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 24.6.1). However, the user should not be allowed to overwrite the existing value of Job\_performance 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 24.2(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 prob-lem is outside the scope of our presentation. We refer the interested reader to the Selected Bibliography at the end of this chapter for further details.

**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. Therefore, they are suitable for military and high security types of applications, which require a higher 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 there-fore they are applicable to few environments. In many practical situations, discretionary policies are preferred because they offer a better tradeoff between security and applicability.

**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 enterprise-wide systems. Its basic notion is that privileges and other permissions are associated with organizational **roles**, rather than individual users. Individual users are then assigned to appropriate roles. Roles can be created using the CREATE ROLE and DESTROY ROLE commands. The GRANT and REVOKE commands discussed in Section 24.2 can then be used to assign and revoke privileges from roles, as well as for individual users when needed. For example, a company may have roles such as sales account manager, purchasing agent, mailroom clerk, department manager, and so on. Multiple individuals can be assigned to each role. Security privileges that are common to a role are granted to the role name, and any individual assigned to this role would automatically have those privileges granted.

RBAC can be used with traditional discretionary and mandatory access controls; it ensures that only authorized users in their specified roles 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 several roles, but it maps to one user or a single subject only. Many DBMSs have allowed the concept of roles, where privileges can be assigned to roles.

Separation of duties is another important requirement in various commercial DBMSs. It is needed to prevent one user from doing work that requires the involvement of two or more people, thus preventing collusion. One method in which sepa-ration of duties can be successfully implemented is with mutual exclusion of roles. Two roles are said to be **mutually exclusive** if both the roles cannot be used simultaneously by the user. **Mutual exclusion of roles** can be categorized into two types, namely *authorization time exclusion (static)* and *runtime exclusion (dynamic)*. In authorization time exclusion, two roles that have been specified as mutually exclusive cannot be part of a user’s authorization at the same time. In runtime exclusion, both these roles can be authorized to one user but cannot be activated by the user at the same time. Another variation in mutual exclusion of roles is that of complete and partial exclusion.

The **role hierarchy** in RBAC is a natural way to organize 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. In other words, if a user has one role, the user automatically has roles lower in the hierarchy. Defining a role hierarchy involves choosing the type of hierarchy and the roles, and then implementing the hierarchy by granting roles to other roles. Role hierarchy can be implemented in the following manner:

GRANT ROLE full\_time TO employee\_type1

GRANT ROLE intern TO employee\_type2

The above are examples of granting the roles *full\_time* and *intern* to two types of employees.

Another issue related to security is *identity management*. **Identity** refers to a unique name of an individual person. Since the legal names of persons are not necessarily unique, the identity of a person must include sufficient additional information to make the complete name unique. Authorizing this identity and managing the schema of these identities is called **Identity Management.** Identity Management addresses how organizations can effectively authenticate people and manage their access to confidential information. It has become more visible as a business requirement across all industries affecting organizations of all sizes. Identity Management administrators constantly need to satisfy application owners while keeping expenditures under control and increasing IT efficiency.

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 only for a certain duration.

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. These features are lacking in DAC and MAC models. In addition, RBAC models include the capabilities available in traditional DAC and MAC policies. Furthermore, an RBAC model provides mechanisms for addressing the security issues related to the execution of tasks and workflows, and for specifying user-defined and organization-specific policies. Easier deployment over the Internet has been another reason for the success of RBAC models.

**3. Label-Based Security and Row-Level Access Control**

Many commercial DBMSs currently use the concept of row-level access control, where sophisticated access control rules can be implemented by considering the data row by row. In row-level access control, each data row is given a label, which is used to store information about data sensitivity. Row-level access control provides finer granularity of data security by allowing the permissions to be set for each row and not just for the table or column. Initially the user is given a default session label by the database administrator. Levels correspond to a hierarchy of data-sensitivity levels to exposure or corruption, with the goal of maintaining privacy or security. Labels are used to prevent unauthorized users from viewing or altering certain data. A user having a low authorization level, usually represented by a low number, is denied access to data having a higher-level number. If no such label is given to a row, a row label is automatically assigned to it depending upon the user’s session label.

A policy defined by an administrator is called a **Label Security policy.** Whenever data affected by the policy is accessed or queried through an application, the policy is automatically invoked. When a policy is implemented, a new column is added to each row in the schema. The added column contains the label for each row that reflects the sensitivity of the row as per the policy. Similar to MAC, where each user has a security clearance, each user has an identity in label-based security. This user’s identity is compared to the label assigned to each row to determine whether the user has access to view the contents of that row. However, the user can write the label value himself, within certain restrictions and guidelines for that specific row. This label can be set to a value that is between the user’s current session label and the user’s minimum level. The DBA has the privilege to set an initial default row label.

The Label Security requirements are applied on top of the DAC requirements for each user. Hence, the user must satisfy the DAC requirements and then the label security requirements to access a row. The DAC requirements make sure that the user is legally authorized to carry on that operation on the schema. In most applica-tions, only some of the tables need label-based security. For the majority of the application tables, the protection provided by DAC is sufficient.

Security policies are generally created by managers and human resources personnel. The policies are high-level, technology neutral, and relate to risks. Policies are a result of management instructions to specify organizational procedures, guiding principles, and courses of action that are considered to be expedient, prudent, or advantageous. Policies are typically accompanied by a definition of penalties and countermeasures if the policy is transgressed. These policies are then interpreted and converted to a set of label-oriented policies by the **Label Security administra-tor**, who defines the security labels for data and authorizations for users; these labelsand authorizations govern access to specified protected objects.

Suppose a user has SELECT privileges on a table. When the user executes a SELECT statement on that table, Label Security will automatically evaluate each row returned by the query to determine whether the user has rights to view the data. For example, if the user has a sensitivity of 20, then the user can view all rows having a security level of 20 or lower. The level determines the sensitivity of the information contained in a row; the more sensitive the row, the higher its security label value. Such Label Security can be configured to perform security checks on UPDATE, DELETE, and INSERT statements as well.

**4. XML Access Control**

With the worldwide use of XML in commercial and scientific applications, efforts are under way to develop security standards. Among these efforts are digital  signatures and encryption standards for XML. The XML Signature Syntax and Processing specification describes an XML syntax for representing the associations between cryptographic signatures and XML documents or other electronic resources. The specification also includes procedures for computing and verifying XML signatures. An XML digital signature differs from other protocols for message signing, such as **PGP** (**Pretty Good Privacy**—a confidentiality and authentication service that can be used for electronic mail and file storage application), in its sup-port for signing only specific portions of the XML tree (see Chapter 12) rather than the complete document. Additionally, the XML signature specification defines mechanisms for countersigning and transformations—so-called *canonicalization* to ensure that two instances of the same text produce the same digest for signing even if their representations differ slightly, for example, in typographic white space.

The XML Encryption Syntax and Processing specification defines XML vocabulary and processing rules for protecting confidentiality of XML documents in whole or in part and of non-XML data as well. The encrypted content and additional pro-cessing information for the recipient are represented in well-formed XML so that the result can be further processed using XML tools. In contrast to other commonly used technologies for confidentiality such as SSL (Secure Sockets Layer—a leading Internet security protocol), and virtual private networks, XML encryption also applies to parts of documents and to documents in persistent storage.

**5. Access Control Policies for E-Commerce and the Web**

Electronic commerce (**e-commerce**) environments are characterized by any trans-actions 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 takethe 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 specific and individual characteristics (for example, 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 or position or role within an organization). For instance, by using credentials, one can simply formulate policies such as *Only permanent staff with five or more years of service can access* *documents related to the internals of the system.*

It is believed that the XML is expected to play a key role in access control for e-commerce applications5 because 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 Services Markup Language** (DSML) is a representation of directory service information in XML syntax. It provides a foundation for 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 specify-ing credentials and access control policies facilitates secure credential submission and export of access control policies.