# Ch 24: Introduction to Database Security Issues

**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 cannot 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 and Protection measures**: Threats to databases can result in the loss or degradation of some or all of the following commonly accepted **security goals**: *integrity*, *availability*, and *confidentiality* (protection from unauthorized disclosure). To protect databases against these types of threats, it is common to implement four kinds of **control measures**: (1) **access control**, (2) **inference control**, (3) **flow control**, and (4) **encryption**.

Control Measures:
Four main control measures are used to provide security of data in databases:
■ **Access control**: creating user accounts and passwords to control the login process by the DBMS.
■ **Inference control**: Cannot deduce or infer certain facts concerning individuals from the DB.
■ **Flow control**: Protect information from reaching unauthorized users.
■ **Data encryption**: The data is encoded using some coding algorithm.

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 (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.

**Database Security and the DBA**:
The DBA has a DBA account in the DBMS, sometimes called a system or super user account, which provides powerful capabilities that are not made available to regular database accounts and users.

**1. Account creation**: This action creates a new account and password for a user or a group of users to enable access to the DBMS.
**2. Privilege granting**: This action permits the DBA to grant certain privileges to certain accounts.
**3. Privilege revocation**: This action permits the DBA to revoke (cancel) certain privileges that were previously given to certain accounts.
**4. Security level assignment**: This action consists of assigning user accounts to the appropriate security clearance level.

Among other DBA functions, one can list: **Access Control**, **User Accounts**, and **Database Audits.**

**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 covered 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:

1. **Inherently sensitive**: The value of the data itself may be so revealing or confidential that it becomes sensitive—for example, a person's salary or that a patient has HIV/AIDS.

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

3. **Declared sensitive**: The owner of the data may have explicitly declared it as sensitive.

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

5. **Sensitive in relation to previously disclosed data**: Some data may not be sensitive by itself but will become sensitive in the presence of some other data—for example, the exact latitude and longitude information for a location 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 column or a data element) or not for **individual users** or for **categories of users**. Several factors 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.**

1. **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 (concurrency control mechanism).

2. **Access acceptability**: Data should only be revealed to authorized users. A database 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.

3. **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. 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 that access to it is suitably controlled. To provide security means to disclose only non-sensitive data, and reject any query that references a sensitive field.

■ **Precision**: To protect all sensitive data while disclosing as much non-sensitive data 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.

## 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 system from 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. 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**. The concept of an **authorization identifier** is used to refer, roughly speaking, to a **user account**. 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 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.

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

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.

The granting and revoking of privileges generally follow an authorization model for discretionary privileges known as the access matrix model, where the rows of a matrix M represent subjects (users, accounts, programs) and the columns represent objects (relations, records, columns, views, operations).

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.

■ **Modification** privileges on R: This gives the account the capability to modify the tuples of R. In SQL this includes three privileges: **UPDATE**, **DELETE**, and **INSERT**. These correspond to the three SQL commands for modifying a table R. Additionally, both the **INSERT** and **UPDATE** privileges can specify that only certain attributes of R can be modified by the account.

■ **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 corresponds to the view.

**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.

**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.

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

**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 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;**

User account A1 can now create tables under the schema called **EXAMPLE**. To continue our example, suppose that A1 creates the two base relations **EMPLOYEE** and **DEPARTMENT** shown in Figure 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 command:

**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.

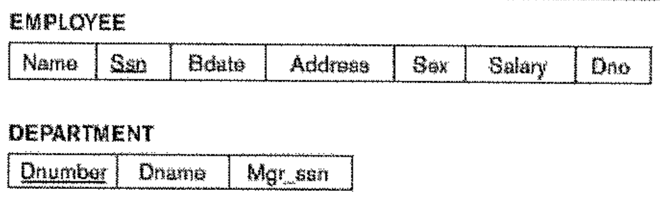

**Figure 24.1**
Schemas for the two relations EMPLOYEE and DEPARTMENT.

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;
```

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 attribute specific, because this specificity can easily be controlled by creating the appropriate views that include only the desired attributes and granting the corresponding privileges on the views. However, because updating views is not always possible the UPDATE and INSERT privileges are given the option to specify the particular attributes of a base relation that may be updated.

**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 access control mechanisms. **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.

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($A_1$, $C_1$, $A_2$, $C_2$, ..., $A_n$, $C_n$, TC)**

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

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 the Name attribute of 'Smith' to appear, with all the other attributes appearing as null (Figure 24.2(c)).

(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 24.2**
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.

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)) permits such an update, the system should not reject it; otherwise, the user could infer that some non-null 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** . 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 problem is outside the scope of our presentation.

**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 therefore they are applicable to few environments. In many practical situations, discretionary policies are preferred because they offer a better tradeoff between security and applicability.

**Challenges of Database Security:**

Considering the vast growth in volume and speed of threats to databases and information assets, research efforts need to be devoted to the following issues: **data quality, intellectual property rights**, and **database survivability**. These are only some of the main challenges that researchers in database security are trying to address.

**Data Quality:**

The database community needs techniques and organizational solutions to assess and attest the quality of data. These techniques may include simple mechanisms such as quality stamps that are posted on Web sites. We also need techniques that provide more effective integrity semantics verification and tools for the assessment of data quality, based on techniques such as record linkage. Application-level recovery techniques are also needed for automatically repairing incorrect data. The ETL (extract, transform, load) tools widely used to load data in data warehouses are presently grappling with these issues.

**Intellectual Property Rights:**

With the widespread use of the Internet and intranets, legal and informational aspects of data are becoming major concerns of organizations. To address these concerns, watermarking techniques for relational data have been proposed. The main purpose of digital watermarking is to protect content from unauthorized duplication and distribution by enabling provable ownership of the content. It has traditionally relied upon the availability of a large noise domain within which the object can be altered while retaining its essential properties. However, research is needed to assess the robustness of such techniques and to investigate different approaches aimed at preventing intellectual property rights violations.

**Database Survivability:**

Database systems need to operate and continue their functions, even with reduced capabilities, despite disruptive events such as information warfare attacks. A DBMS, in addition to making every effort to prevent an attack and detecting one in the event of occurrence, should be able to do the following:

- **Confinement**: Take immediate action to eliminate the attacker's access to the system and to isolate or contain the problem to prevent further spread.

- **Damage assessment**: Determine the extent of the problem, including failed functions and corrupted data.

- **Reconfiguration**: Reconfigure to allow operation to continue in a degraded mode while recovery proceeds.

- **Repair**: Recover corrupted or lost data and repair or reinstall failed system functions to reestablish a normal level of operation.

- **Fault treatment**: To the extent possible, identify the weaknesses exploited in the attack and take steps to prevent a recurrence.

The goal of the information warfare attacker is to damage the organization's operation and fulfillment of its mission through disruption of its information systems. The specific target of an attack may be the system itself or its data. While attacks that bring the system down outright are severe and dramatic, they must also be well timed to achieve the attacker's goal, since attacks will receive immediate and concentrated attention in order to bring the system back to operational condition, diagnose how the attack took place, and install preventive measures. To date, issues related to database survivability have not been sufficiently investigated. Much more research needs to be devoted to techniques and methodologies that ensure database system survivability.

**Oracle Label-Based Security:**
Restricting access to entire tables or isolating sensitive data into separate databases is a costly operation to administer. Oracle Label Security overcomes the need for such measures by enabling row-level access control. Figure 24.4 shows the sequence of discretionary access control (DAC) and label security checks.

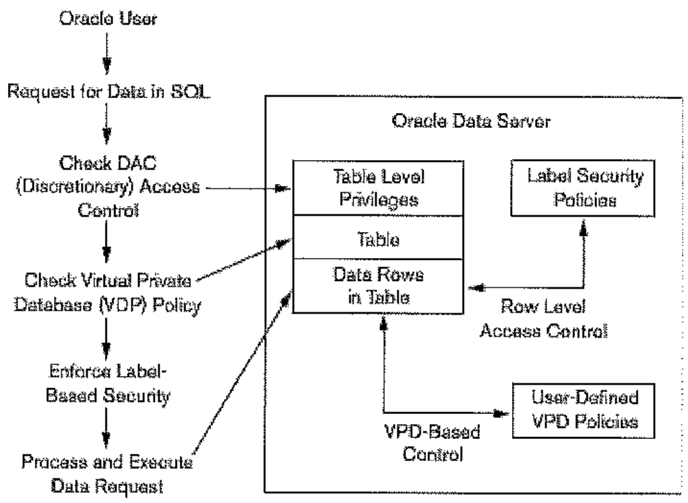
Oracle User
Request for Data in SQL
Check DAC (Discretionary) Access Control
Check Virtual Private Database (VDP) Policy
Enforce Label-Based Security
Process and Execute Data Request
Oracle Data Server
Table Level Privileges
Table
Data Rows in Table
Label Security Policies
Row Level Access Control
User-Defined VPD Policies
VPD-Based Control

**Figure 24.4**
Oracle Label Security architecture.
*Source:* Oracle (2007)

**Virtual Private Databases** (VPDs) is a feature of the Oracle Enterprise Edition that adds predicates to user statements to limit their access in a transparent manner to the user and the application. The VPD concept allows server-enforced, fine-grained access control for a secure application.

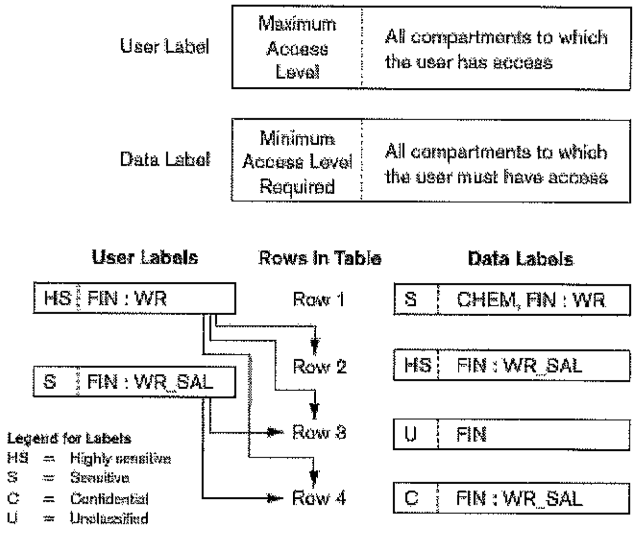

**Figure 24.5**
Data labels and user labels in Oracle.
*Source:* Oracle (2007)

As shown in Figure 24.5, User 1 can access the rows 2, 3, and 4 because his maximum level is HS (Highly_Sensitive). He has access to the FIN (Finance) compartment, and his access to group WR (Western Region) hierarchically includes group WR_SAL (WR Sales). He cannot access row 1 because he does not have the CHEM (Chemical) compartment. It is important that a user has authorization for all compartments in a row's data label to be able to access that row. Based on this example, user 2 can access both rows 3 and 4,and has a maximum level of S, which is less than the HS in row 2.So, although user 2 has access to the FIN compartment, he can only access the group WR_SAL, and thus cannot access row 1.