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 24.3. This is a PERSON relation with the attributes Name, Ssn, Income, Address, City, State, Zip, Sex, and Last_degree.

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

Statistical queries involve applying statistical functions to a population of tuples. For example, we may want to retrieve the number of individuals in a population or the average income in the population. However, statistical users are not allowed to retrieve individual data, such as the income of a specific person. Statistical database security techniques must prohibit the retrieval of individual data. This can be 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 24.3
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.

The possibility of accessing individual information from statistical queries is reduced by using the following measures –

Partitioning of Database – This means the records of database must be not be stored as bulk in single record. It must be divided into groups of some minimum size according to confidentiality of records. The advantage of Partitioning of database is queries can refer to any

complete group or set of groups, but queries cannot access the subsets of records within a group. So, attacker can access at most one or two groups which are less private.

If no statistical queries are permitted whenever number of tuples in population specified by selection condition falls below some threshold.

Prohibit sequences of queries that refer repeatedly to same population of tuples.

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 infor-mation 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 data-base 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.