*Handbook of Information Security Management, Auerbach Publishers, 1993, pages 481-499.*

# DATA AND DATABASE SECURITY AND CONTROLS

*Ravi S. Sandhu and Sushil Jajodia*

Center for Secure Information Systems
&
Department of Information and Software Systems Engineering
George Mason University, Fairfax, VA 22030-4444
Telephone: 703-993-1659

## 1 Introduction

This chapter discusses the topic of data security and controls, primarily in the context of Database Management Systems (DBMSs). The emphasis is on basic principles and mechanisms, which have been successfully used by practitioners in actual products and systems. Where appropriate, the limitations of these techniques are also noted. Our discussion focuses on principles and general concepts. It is therefore independent of any particular product (except for section 7 which discusses some products). In the more detailed considerations we limit ourselves specifically to relational DBMSs. The reader is assumed to be familiar with rudimentary concepts of relational databases and SQL. A brief review of essential concepts is given in the appendix.

The chapter begins with a review of basic security concepts in section 2. This is followed, in section 3, by a discussion of access controls in the current generation of commercially available DBMSs. Section 4 introduces the problem of multilevel security. It is shown that the techniques of section 3 are inadequate to solve this problem. Additional techniques developed for multilevel security are reviewed. Section 5, discusses the various kinds of inference threats that arise in a database system, and discusses methods that have been developed for dealing with them. Section 6 addresses the problem of data integrity. Current practice in this area is discussed and limitations are noted. Section 7 discusses some commercially available DBMS products. Section 8 gives a summary of the chapter. A glossary and list of readings follow the summary. Finally, the appendix gives a brief review of essential relational concepts and SQL.

# 2 Basic Security Concepts

In this section we review some basic security concepts which are important for understanding this chapter. We also define some technical terms which will be used throughout this chapter. Whenever we use a technical term for the first time we will italicize it, to draw attention to its definition which is given at that point.

## 2.1 Secrecy, Integrity and Availability

The objective of *data security* can be divided into three separate, but interrelated, areas as follows.

- *Secrecy* is concerned with improper disclosure of information. The terms *confidentiality* or *non-disclosure* are synonyms for secrecy.
- *Integrity* is concerned with improper modification of information or processes.
- *Availability* is concerned with improper denial of access to information. The term *denial of service* is also used as a synonym for availability.

These three objectives arise in practically every information system. For example, in a payroll system secrecy is concerned with preventing an employee from finding out the boss's salary; integrity is concerned with preventing an employee from changing his or her salary; and availability is concerned with ensuring that the paychecks are printed on time as required by law. Similarly, in a military command and control system secrecy is concerned with preventing the enemy from determining the target coordinates of a missile; integrity is concerned with preventing the enemy from altering the target coordinates; and availability is concerned with ensuring that the missile does get launched when the order is given.

Any system will have these three requirements co-existing to some degree. There are of course differences regarding the relative importance of these objectives in a given system. The commercial and military sectors both have similar needs for high integrity systems. The secrecy and availability requirements of the military are often more stringent than in typical commercial applications.

These three objectives also differ with respect to our understanding of the objectives themselves and of the technology to achieve them. It is easiest to understand the objective of secrecy. Integrity is a less tangible objective on which experts in the field have diverse opinions. Availability is technically the least understood aspect. In terms of technology, the dominance of the commercial sector in the marketplace has led vendors to emphasize mechanisms for integrity rather than for military-like secrecy needs. The availability objective is so poorly understood that no product today even tries to address it directly. Availability is discussed only in passing in this chapter.

## 2.2 Security Policy

The purpose of a *security policy* is to elaborate the three generic security objectives of secrecy, integrity and availability, in the context of a particular system. The generic objectives have all used the term “improper” in their definition. A statement of security policy largely consists of defining what is the meaning of “improper” for a particular system.

The meaning of “improper” is sometimes mandated by law, such as for secrecy in the classified military and government sectors. Legal and professional requirements apply to medical records and other sensitive personal information about individuals. Due to conflict of interest considerations so-called Chinese Walls are required to prevent business consultants from accessing confidential information for two or more companies competing in the same market sector. In general, however, security policy is largely determined within an organization rather than imposed by mandate from outside. This is particularly so in the integrity and availability arenas.

## 2.3 Prevention, Detection and Tolerance

The objective of data security can be approached in two distinct, and mutually supportive, ways.

- *Prevention.* Prevention ensures that security breaches cannot occur. The basic technique is that the system examines every action and checks its conformance with the security policy before allowing it to occur. This technique is called *access control.*

- *Detection.* Detection ensures that sufficient history of the activity in the system is recorded in an *audit trail*, so that a security breach can be detected after the fact. This technique is called *auditing.*

Every system employs some mix of these two techniques. Sometimes the distinction between these two techniques gets blurred. For example, consider a system which monitors the audit trail in real time looking for imminent security violations so as to prevent them. Such a system is preventive in nature, yet the technology used is basically a detective one. The distinction is nevertheless a useful one. Our focus in this chapter is on preventive techniques.

Prevention is the more fundamental technique. An effective detection mechanism requires a mechanism to prevent improper modification of the audit trail. Moreover, detection is ultimately useful only to the extent that it prevents improper activity by threatening punitive action.

Finally, there is the third “technique” of *tolerance* in which the potential for some security breaches is tolerated; because either these breaches are too expensive to prevent or detect, or the likelihood of their occurrence is considered sufficiently low, or security measures are acceptable to users only up to some reasonable point. Every practical system tolerates some degree of risk with respect to potential security breaches. It is, however, important to understand what risk is being tolerated and what is being covered by preventive/detective mechanisms.

### 2.4 Assurance

Security mechanisms, whether preventive or detective in nature, can be implemented with various degrees of *assurance*. Assurance is directly related to the effort required to subvert the mechanism. Low assurance mechanisms are easy to implement but also relatively easy to subvert. Subtle bugs in system and/or application software have led to numerous security breaches. On the other hand, high assurance mechanisms are notoriously difficult to implement. They also tend to suffer from degraded performance. Fortunately, rapid advances in hardware performance are making the performance penalty acceptable.

## 3 Access Controls in Current Systems

In this section we discuss the access controls provided in the current generation of commercially available Database Management Systems. Our focus is on relational systems. The access controls described here are often referred to as *discretionary access controls* as opposed to the *mandatory access controls* of multilevel security. Discussion of this distinction is deferred until section 4.

The purpose of access controls is to ensure that a user is only permitted to perform those operations on the database for which that user is authorized. Access controls are based on the premise that the user has been correctly identified to the system by some *authentication* procedure. Authentication typically requires the user to supply his or her claimed identity (e.g., user name, operator number, etc.) along with a password or some other authentication token. Authentication may be performed by the Operating System, the Database Management System, a special Authentication Server, or some combination thereof. Authentication is not discussed any further in this chapter. We simply assume that a suitable mechanism is in place.

### 3.1 Granularity and Modes of Access Control

Access controls can be imposed at various degrees of granularity in a system. Some possibilities are enumerated below.

- The entire database.

- Some collection of relations.
- One relation.
- Some columns of one relation.
- Some rows of one relation.
- Some columns of some rows of one relation.

Access controls are also differentiated with respect to the operation to which they apply. These distinctions are important, e.g., each employee may be authorized to read his own salary but not to write it. In relational databases *access control modes* are expressed in terms of the basic SQL operations (i.e., SELECT, UPDATE, INSERT and DELETE) as follows.

- The ability to INSERT and DELETE is specified on a relation by relation basis.
- SELECT is also usually specified on a relation by relation basis. Finer granularity of authorization for SELECT can be provided by views (see section 3.2.1 below).
- UPDATE can be restricted to certain columns of a relation.

In addition to these access control modes which apply to individual relations or parts thereof, there are also privileges which confer special authority on users. A common example is the DBA privilege for Database Administrators.

## 3.2 Data Dependent Access Control

Database access controls are often *data dependent.* For example, some users may be limited to seeing salaries which are less than $30,000. Similarly, a manager may be restricted to seeing salaries for employees in his department. We now discuss two basic techniques, viz., view-based access controls and query modification, for implementing data-dependent access controls in relational databases.

### 3.2.1 View Based Access Control

A *base relation* is a "real" relation in the database, that is it is actually stored in the Database. A *view* is a "virtual" relation which is derived from base relations and other views. The database stores the view definitions and materializes the view as needed.

To illustrate the concept of a view, and its security application, consider the EMPLOYEE relation of Table 1. (The value NULL indicates that Harding has no manager.) The following SQL statement defines a view called TOY-DEPT.

| NAME | DEPT | SALARY | MANAGER |
|---|---|---|---|
| Smith | Toy | 10,000 | Jones |
| Jones | Toy | 15,000 | Baker |
| Baker | Admin | 40,000 | Harding |
| Adams | Candy | 20,000 | Harding |
| Harding | Admin | 50,000 | NULL |

Table 1: Base Relation EMPLOYEE

| NAME | SALARY | MANAGER |
|---|---|---|
| Smith | 10,000 | Jones |
| Jones | 15,000 | Baker |

Table 2: View TOY-DEPT

```
CREATE   VIEW TOY-DEPT
AS       SELECT   NAME, SALARY, MANAGER
         FROM     EMPLOYEE
         WHERE    DEPT = 'Toy'
```

This defines the virtual relation shown in Table 2. A user who has read access to TOY-DEPT is thereby limited to retrieving information about employees in the Toy Department. To illustrate the dynamic aspect of views suppose that a new employee Brown is inserted in base relation EMPLOYEE, as shown in Table 3. The view TOY-DEPT will be automatically modified to include Brown, as shown in Table 4.

Views can also be used to provide access to statistical information. For example, the following view gives the average salary for each department.

```
CREATE   VIEW AVSAL(DEPT, AVG)
AS       SELECT     DEPT, AVG(SALARY)
         FROM       EMPLOYEE
         GROUP BY   DEPT
```

For retrieval purposes users need not distinguish between views and base relations. A view is simply another relation in the database, which happens to be automatically modified by the DBMS whenever its base relations are modified. Views, therefore, provide a very powerful mechanism for specifying data-dependent authorization for data retrieval. There are, however, significant problems if views are modified by users directly (rather than by side effect of modifying base relations). This is due to our theoretical inability, in general, to translate updates on views into updates of base

| NAME | DEPT | SALARY | MANAGER |
|---|---|---|---|
| Smith | Toy | 10,000 | Jones |
| Jones | Toy | 15,000 | Baker |
| Baker | Admin | 40,000 | Harding |
| Adams | Candy | 20,000 | Harding |
| Harding | Admin | 50,000 | NULL |
| Brown | Toy | 22,000 | Harding |

Table 3: Modified Base Relation EMPLOYEE

| NAME | SALARY | MANAGER |
|---|---|---|
| Smith | 10,000 | Jones |
| Jones | 15,000 | Baker |
| Brown | 22,000 | Harding |

Table 4: Automatically Modified View TOY-DEPT

relations (see section 6.3.2). This limits the usefulness of views for data-dependent authorization of update operations.

#### 3.2.2 Query Modification

Query modification is another technique for enforcing data-dependent access controls for retrieval. (It is not supported in SQL, but is discussed here for the sake of completeness.) In this technique, a query submitted by a user is modified to include further restrictions as determined by the user's authorization.

Suppose that the Database Administrator has granted Thomas the ability to query the EMPLOYEE base relation for employees in the Toy Department, as follows.

```
GRANT   SELECT
ON      EMPLOYEE
TO      Thomas
WHERE   DEPT = 'Toy'
```

Now suppose that Thomas executes the following query.

```
SELECT  NAME, DEPT, SALARY, MANAGER
FROM    EMPLOYEE
```

In the absence of access controls this query would return the entire EMPLOYEE relation. Due to the above GRANT, however, the DBMS will automatically modify this query to the following.

```
SELECT  NAME, DEPT, SALARY, MANAGER
FROM    EMPLOYEE
WHERE   DEPT = 'Toy'
```

This will limit Thomas to retrieving that portion of the EMPLOYEE relation for which he was granted SELECT access.

## 3.3 Granting and Revocation of Access

Granting and revocation allow users to selectively and dynamically grant privileges to other users, and subsequently revoke them if so desired. In SQL granting is accomplished by means of the GRANT statement which has the following general format.

```
GRANT   privileges
[ON     relation]
TO      users
[WITH   GRANT OPTION]
```

The GRANT command applies to base relations as well as views. The brackets on the ON and WITH clauses denote that these are optional and may not be present in every GRANT command.

Some examples of GRANT statements are given below.

- GRANT SELECT ON EMPLOYEE TO TOM

  Allows Tom to execute SELECT queries on the EMPLOYEE relation.

- GRANT SELECT, UPDATE(SALARY) ON EMPLOYEE TO TOM

  As above, and in addition allows Tom to modify the SALARY of existing employees in the EMPLOYEE relation.

- GRANT INSERT, DELETE ON EMPLOYEE TO TOM, DICK, HARRY

  Allows Tom, Dick and Harry to insert new rows (i.e, new employees) in the EMPLOYEE relation, and to delete existing rows (i.e., existing employees) from the EMPLOYEE relation.

- GRANT SELECT ON EMPLOYEE TO TOM WITH GRANT OPTION

  Allows Tom to execute SELECT queries on EMPLOYEE relation, and further allows him to GRANT this privilege to other users (with or without the GRANT OPTION).

- GRANT DBA TO JILL WITH GRANT OPTION

  This gives the DBA privilege to Jill. This allows Jill to act as the Database Administrator and confers a large number of privileges on her. No specific relation is mentioned because the DBA privilege confers system-wide authority. In this case, Jill can in turn GRANT this privilege to other users (with or without the GRANT OPTION).

Note that it is not possible to grant a user the grant option on a privilege, without allowing the grant option itself to be further granted.

Revocation in SQL is accomplished by means of the REVOKE statement which has the following general format.

```
REVOKE   privileges
[ON      relation]
FROM     users
```

Some examples of REVOKE statements are given below. The meaning of REVOKE depends upon who executes it. The examples are therefore given as a sequence of actions. Each step identifies the user who executes it.

- DICK: GRANT SELECT ON EMPLOYEE TO TOM
  DICK: REVOKE SELECT ON EMPLOYEE FROM TOM

  Dick grants a privilege to Tom and then revokes it.

- DICK: GRANT SELECT ON EMPLOYEE TO TOM
  HARRY: GRANT SELECT ON EMPLOYEE TO TOM
  DICK: REVOKE SELECT ON EMPLOYEE FROM TOM

  Dick revokes his grant of the SELECT privilege to Tom. However, Tom continues to retain the SELECT privilege due to the grant by Harry.

- DICK: GRANT SELECT ON EMPLOYEE TO JOE WITH GRANT OPTION
  JOE: GRANT SELECT ON EMPLOYEE TO TOM
  DICK: REVOKE SELECT ON EMPLOYEE FROM JOE

  This revokes Dick's grant of the SELECT privilege to Joe, and also indirectly revokes Joe's grant of the SELECT privilege to Tom. This is called *cascading revocation*.

- DICK: GRANT SELECT ON EMPLOYEE TO JOE WITH GRANT OPTION
  HARRY: GRANT SELECT ON EMPLOYEE TO JOE WITH GRANT OPTION
  JOE: GRANT SELECT ON EMPLOYEE TO TOM
  DICK: REVOKE SELECT ON EMPLOYEE FROM JOE

  This revokes Dick's grant. However, Joe and Tom continue to retain the SELECT privilege due to the grant by Harry.

- DICK: GRANT SELECT ON EMPLOYEE TO JOE WITH GRANT OPTION
  JOE: GRANT SELECT ON EMPLOYEE TO TOM
  HARRY: GRANT SELECT ON EMPLOYEE TO JOE WITH GRANT OPTION
  DICK: REVOKE SELECT ON EMPLOYEE FROM JOE

  This revokes Dick's grant. Since Joe has received a later grant from Harry, Joe continues to hold the SELECT privilege (with grant option). The SELECT privilege from Tom is, however, revoked. This is because the grant from Joe to Tom occurred before Harry's grant to Joe.

- DICK: GRANT UPDATE(SALARY,DEPT) ON EMPLOYEE TO JOE
  DICK: REVOKE UPDATE ON EMPLOYEE FROM JOE

  This revokes Joe's UPDATE privilege on both columns SALARY and DEPT. It is not possible to revoke UPDATE for only one of these columns.

## 4 Multilevel Security Requirements

In this section we introduce the problem of multilevel security. The focus of multilevel security is on secrecy, and this section reflects this focus.

The access controls of section 3 are said to be *discretionary*, because the granting of access is under user control. Users who possess a privilege with the grant option are free to grant it to whomever they choose to. We first show that this approach has serious limitations with respect to secrecy requirements. We then describe how *mandatory access controls* get around this limitation. Next we introduce the *covert channel problem* which even mandatory controls are unable to solve. Finally, we briefly discuss the evaluation criteria for secure computer systems developed by the U.S. Department of Defense. It should be noted that although multilevel security has been developed primarily for the military sector, it is applicable in the commercial sector also.

### 4.1 Limitations of Discretionary Access Controls

To illustrate the basic limitation of discretionary access controls, consider the following grant operation.

TOM: GRANT SELECT ON EMPLOYEE TO DICK

Tom has not conferred the grant option on Dick. Tom's intention is that Dick should not be allowed to further grant SELECT access on EMPLOYEE to other users. However, this intent is easily subverted as follows. Dick creates a new relation, call it COPY-OF-EMPLOYEE, into which he copies all the rows of EMPLOYEE. As the creator of COPY-OF-EMPLOYEE, Dick has the authority to grant any privileges for it to any user. Dick can therefore grant Harry access to COPY-OF-EMPLOYEE as follows.

```
DICK: GRANT SELECT ON COPY-OF-EMPLOYEE TO HARRY
```

At this point Harry has access to all the information in the original EMPLOYEE relation. For all practical purposes Harry has SELECT access to EMPLOYEE, so long as Dick keeps COPY-OF-EMPLOYEE reasonably up to date with respect to EMPLOYEE.

The situation is actually worse than the above scenario indicates. So far, we have portrayed Dick as a cooperative participant in this process. Now suppose that Dick is a trusted confidant of Tom and would not deliberately subvert Tom's intentions regarding the EMPLOYEE relation. However, Dick uses a fancy text editor supplied to him by Harry. This editor provides all the editing services that Dick needs. In addition Harry has also programmed it to create the COPY-OF-EMPLOYEE relation and execute the above grant operation. Such software is said to be a *Trojan Horse*, because in addition to the normal functions expected by its user it also engages in surreptitious actions to subvert security. Note that a Trojan Horse executed by Tom could actually grant Harry the privilege to SELECT on EMPLOYEE.

In summary, even if the users are trusted not to deliberately breach security we have to contend with Trojan Horses which have been programmed to deliberately do so. We can require that all software that is run on the system is free of Trojan Horses. But this is hardly a practical option. The solution is to impose mandatory controls which cannot be violated, even by Trojan Horses.

## 4.2 Mandatory Access Controls

Mandatory access controls are based on *security labels* associated with each data item and each user. A label on a data item is called a *security classification*, while a label on a user is called a *security clearance*. In a computer system every program run by a user inherits the user's security clearance. In other words the user's clearance applies not only to the user as a human being, but also to every program executed by that user. It is important to understand that a particular program, such as a text editor, when executed by a Secret user is run as a Secret process; whereas when it is executed by an Unclassified user it is run as an Unclassified process. Moreover, the classifications and clearances once assigned cannot be changed (except by the security officer).

Security labels in the military and government sectors consist of two components: a *hierarchical component* and a (possibly empty) set of *categories*. The hierarchical component consists of the following, listed in decreasing order of sensitivity.

- Top secret (TS)
- Secret (S)
- Confidential (C)
- Unclassified (U)

The categories consist of items such as NUCLEAR, CONVENTIONAL, NAVY, ARMY, NATO, etc.

The label X is said to *dominate* label Y provided the hierarchical component of X is greater than or equal to the hierarchical component of Y, and the categories of X contain all the categories of Y. For example:

- X = (TOP-SECRET, {NUCLEAR, ARMY}) dominates Y = (SECRET, {ARMY})
- X = (SECRET, {NUCLEAR, ARMY}) dominates Y = (SECRET, {NUCLEAR})
- X = (TOP-SECRET, {NUCLEAR}) is *incomparable* to Y = (SECRET, {ARMY}), i.e., neither one dominates the other.

Note that two labels which dominate each other are exactly identical.

Commercial organizations also use similar labels for protecting sensitive information. The main difference is that procedures for assigning clearances to users are much less formal than in the military or government sectors.

We will henceforth limit ourselves to hierarchical labels, i.e., labels without any categories. The reader is cautioned that there are many subtle issues which arise due to incomparable labels (i.e., labels with categories). However, the basic concepts can be demonstrated with hierarchical labels. We usually use the labels Secret and Unclassified in our discussion.

When a user signs on (i.e., logs in) to the system he or she specifies the security level of that session. The level of the session must be dominated by the user's clearance. That is, a Secret user can sign on as Unclassified, but an Unclassified user cannot sign on as Secret. Once the user is signed on at a specific level all programs executed by him or her will be run at that level.

The following rules for mandatory access control were formulated by Bell and LaPadula.

- *Simple Security*: A *subject* (i.e., a running program) with label X can read an *object* (i.e., a data item) with label Y only if X dominates Y.
- *Star-Property*: A subject with label X can write an object with label Y only if Y dominates X.

For example, a Secret subject can read Secret and Unclassified data, but cannot write Unclassified data. So even with Trojan Horses in the software, it is not possible to copy information from Secret data items to Unclassified data items.

The simple-security requirement applies equally to humans and programs. The star-property on the other hand is not applied to the human users, but rather to programs. Human users are trusted not to leak information. A Secret user can write an Unclassified document, because it is assumed that he or she will only put Unclassified information in it. Programs, on the other hand, are not trusted because they may have Trojan Horses embedded in them (recall the Trojan Horse example in section 4.1). A program running at the Secret level is therefore not allowed to write to Unclassified data items.

## 4.3 Covert Channels

Unfortunately, mandatory controls do not solve the Trojan Horse problem completely. A program running at the Secret level is prevented from writing directly to Unclassified data item. There are, however, other ways of communicating information to Unclassified programs.

For example, the Secret program can acquire large amounts of memory in the system. This fact can be detected by an Unclassified program which is able to observe how much memory is available. Even if the Unclassified program is prevented from directly observing the amount of free memory, it can do so indirectly by making a request for a large amount of memory itself. Granting or denial of this request will convey some information about free memory to the Unclassified program.

Such indirect methods of communication are called *covert channels*. Covert channels present a formidable problem for multilevel security. They are difficult to detect, and once detected are difficult to close without incurring significant performance penalties. Covert channels do tend to be noisy due to interference by the activity of other users in the system. Nevertheless, standard coding techniques for communication on noisy channels can be employed by the Trojan Horses to achieve error-free communication, with data rates which can be as high as several million bits per second.

## 4.4 Evaluation Criteria

In 1985 the U.S. Department of Defense published the *Trusted Computer System Evaluation Criteria*, popularly known as the *Orange Book*. This document established a metric against which computers systems can be evaluated for security. The metric consists of a number of levels, A1, B3, B2, B1, C2, C1, and D; listed here in decreasing order of how secure the system is.

For each level, the Orange Book lists a set of requirements that a system must have to achieve that level of security. Briefly, the D level consists of all systems which are not secure enough to qualify for any of A, B, or C levels. Systems at levels C1 and C2 provide discretionary protection of data, systems at level B1 provide mandatory access controls, and systems at levels B2 or above provide increasing assurance, in particular against covert channels. The level A1, which is most rigorous, requires verified protection of data.

In 1991 the U.S. Department of Defense published the *Trusted Database Interpretation of the Trusted Computer System Evaluation Criteria*, popularly known as the TDI. The TDI describes how a DBMS and the underlying OS can be evaluated separately and in conjunction. Several efforts are underway to build secure DBMS products satisfying these criteria.

## 5 Inference and Aggregation

Even in multilevel secure DBMSs, it is possible for users to draw inferences from the information they obtain from the database. The inference could be derived purely from the data obtained from the database system, or it could additionally depend on some prior knowledge which was obtained by users from outside the database system. An inference presents a security breach if more highly classified information can be inferred from less classified information.

There is a significant difference between the inference and covert channel problems. Inference is a unilateral activity, in which an Unclassified user legitimately accesses Unclassified information from which that user is able to deduce Secret information. Covert channels, on the other hand, require cooperation of a Secret Trojan Horse which transmits information to an Unclassified user by indirect means of communication. The inference problem will exist even in an ideal system which is completely free of covert channels.

There are many difficulties associated with determining when more highly classified information can be inferred from less classified information. The biggest problem is that it is impossible to determine precisely what a user knows. The inference problem is somewhat manageable if we adopt the *closed-world assumption* and assume that if information $Y$ can be derived using information $X$, then both $X$ and $Y$ are contained in the database. In reality, however, the outside knowledge that users bring plays a significant role in inference.

There are two important cases of the inference problem, which often arise in database systems.

- An *aggregation problem* occurs whenever there is a collection of data items that is classified at a higher level than the levels of individual data items by themselves. A classic example from a military context occurs when the location of individual ships is unclassified, but the aggregate information concerning the location of all ships in the fleet is secret. Similarly, in the commercial sector the individual sales figures for branch offices might be considered less sensitive than the aggregate sales figures for the entire company.

- A *data association problem* occurs whenever two values seen together are classified at a higher level than the classification of either value individually. As an example, the list consisting of the names of all employees and the list containing all employee salaries are unclassified, while a combined list giving employee names with their salaries is classified.

Notice that the data association problem is different from the aggregation problem since what is really sensitive is not the aggregate of the two lists, but the exact association giving an employee name and his or her salary.

We now describe some techniques for resolving the inference problem. Although these methods can be extremely useful, a complete and generally applicable solution to the inference problem remains an elusive goal.

## 5.1 Appropriate Labeling

If Unclassified information $x$ permits disclosure of Secret information $y$, one way to prevent this is to reclassify all or part of information $x$ such that it is no longer possible to derive $y$ from the disclosed subset of $x$. To illustrate, suppose that an attribute A is Unclassified while attribute B is Secret. Suppose the database enforces the constraint $\mathrm{A} + \mathrm{B} \leq 20$, and that the constraint is known to Unclassified users. The value of B does not affect the value of A directly, but it does constrain the set of possible values A can take. Thus we have an inference problem. This inference can be prevented by reclassifying A as Secret.

## 5.2 Query Restriction

Many inference violations arise as a result of a query which returns data at the user's level, but its evaluation requires accessing data above the user's level. As an example, suppose that data are classified at the relation level, and that we have two relations, an Unclassified relation, called EP, with attributes EMPLOYEE-NAME and PROJECT-NAME, and a Secret relation, called PT, with attributes PROJECT-NAME and PROJECT-TYPE. Let EMPLOYEE-NAME be the key of the first relation and PROJECT-NAME the key of the second. (The existence of the relation scheme PT is Unclassified.)

Suppose an Unclassified user makes the following SQL query.

```
SELECT   EP.PROJECT-NAME
FROM     EP, PT
WHERE    EP.PROJECT-NAME = PT.PROJECT-NAME AND
         EP.PROJECT-TYPE = 'NUCLEAR'
```

The data returned by this query, viz., project names, is extracted from the Unclassified relation EP. As such the output of this query contains Unclassified data. Nevertheless it reveals Secret information, viz., names of nuclear projects, which is stored in the Secret relation PT. The point of this example is that even though the output of this query is wholly contained in the Unclassified relation EP, it reveals Secret information by virtue of being selected based on Secret data in the PT relation.

Query restriction ensures that all data used in the process of evaluating the query is dominated by the level of the user, and therefore prevents such inferences. To this end, the system can either modify the user query such that the query involves only the authorized data or simply abort the query.

## 5.3 Polyinstantiation

Polyinstantiation is another technique that can be used to prevent inference violations. To illustrate, suppose an Unclassified user wants to enter a row in a relation in which each row is labeled as S (Secret) or U (Unclassified). If the same key is already occurring in an S row, we cannot prevent the Unclassified from inserting the U row without leakage of 1 bit of information by inference. In other words the classification of the row has to be treated as part of the relation key. Thus U rows and S rows will always have different keys, since the keys will have different security classes.

An example is given in the SD relation of Table 5, which has the key STARSHIP, CLASS. Suppose a Secret user inserts the first row in this relation. Later, an Unclassified user inserts the second row. This later insertion cannot be rejected without leaking the fact to the Unclassified user that a Secret row for the Enterprise already exists. The insertion is therefore allowed resulting in the relation of Table 5. Unclassified users see only one row for the Enterprise, viz., the U row. Secret users see two rows. There are two different ways these two rows might be interpreted as follows.

- There are two distinct starships named Enterprise going to two distinct destinations. Unclassified users know of the existence of only one of them, viz., the one going to Mars. Secret users know about both of them.
- There is a single starship named Enterprise. Its real destination is Rigel, which is known to Secret users. There is an Unclassified cover story alleging that the destination is Mars.

Presumably, Secret users know which interpretation is intended.

| STARSHIP | DESTINATION | CLASS |
|---|---|---|
| Enterprise | Rigel | S |
| Enterprise | Mars | U |

Table 5: Relation SD

## 5.4 Auditing

Auditing can be used to control inferences. For instance, a history can be kept of all queries made by a user. Whenever the user makes a query, the history is analyzed to determine whether the response to this query, correlated with the responses to earlier queries, could result in an inference violation. If a violation could arise, the systems can take appropriate action (for example, abort the query).

There is a side benefit of this approach: it may deter many inference attacks by threatening discovery of violations. There are two disadvantages of this approach: One, it may be too cumbersome to be useful in practical situation. Two, it can detect very limited types of inferences (since it is based on the hypothesis that a violation can always be detected by analyzing the audit records for abnormal behavior.)

## 5.5 Tolerating Limited Inferences

Tolerance methods are useful in those cases in which the inference bandwidth is so small that these violations do not pose any threat. Consider the following example. Suppose that data are classified at the column level, and that we have two relations, one called PD with the Unclassified attribute PLANE and Secret attribute DESTINATION, and another called DF with the Unclassified attribute DESTINATION and Unclassified attribute FUEL-NEEDED. Suppose also that, although knowledge of the fuel needed for a particular plane can give information about the destination of the plane, there are too many destinations requiring the same amount of fuel for this to be a serious inference threat. Moreover, we do not want to go to the trouble of clearing everybody responsible for fueling the plane to the Secret level. Thus we wish to make the derived relation with attributes PLANE and FUEL-NEEDED available to Unclassified users.

Even though we have decided that this information does not provide a serious inference threat, we cannot allow Unclassified users to extract the required information from PD and PF by, say, executing the following query.

```
SELECT   PLANE, FUEL-NEEDED
FROM     PD, DF
WHERE    PD.DESTINATION = DF.DESTINATION
```

Doing so opens up a covert channel for leaking Secret information to Unclassified users.

One solution is to use the *snapshot approach*, where a trusted user creates a derived Secret relation with attributes PLANE and FUEL-NEEDED and then downgrades it to Unclassified. Although this "snapshot" cannot be updated automatically without opening a covert channel, it can be kept more or less up-to-date by having the trusted user re-create it from time to time.

A "snapshot" or a "sanitized file" is an important technique for controlling inferences, especially in offline, static databases. In particular, it has been used quite effectively by the United States Bureau of the Census.

## 6 Integrity Principles and Mechanisms

In this section we discuss the problem of data integrity. Integrity is a much less tangible objective than secrecy. Our approach to integrity is pragmatic and utilitarian. We define integrity as being concerned with the improper modification of information (much as confidentiality is concerned with improper disclosure). We understand modification to include insertion of new information, deletion of existing information as well as changes to existing information.

The reader may have seen similar definitions of integrity using "unauthorized" instead of "improper." Our use of the latter term is significant and should not be dismissed lightly. Integrity breaches can and do occur without authorization violations. In other words authorization is only one piece of the solution and we must also deal with the malicious user who exercises his or her authority improperly.

### 6.1 The Insider Threat

It is important to understand that the threat posed by a corrupt authorized user is quite different in the context of integrity as compared to secrecy.

A corrupt user can leak secrets by (i) using the computer to legitimately access confidential information, and then (ii) passing on this information to an improper destination by some non-computer means of communication (e.g., a telephone call). It is simply impossible for the computer to know whether or not step (i) was followed by step (ii). We therefore have no choice but to trust our insiders to be honest and alert. The military and government sectors have established elaborate procedures for this purpose, while the commercial sector is much more informal in this respect. Security research which focuses on secrecy therefore considers the principal threat to be Trojan Horses embedded in programs. That is, the focus is on corrupt programs rather than corrupt users.

Analogously, a corrupt user can compromise integrity by (i) manipulating stored data, or (ii) falsifying source or output documents. A computer system can do little by itself to solve the problem of false source or output documents, for which we must rely on the traditional techniques of paper-based manual systems. However, the manipulation of stored data simply cannot be done without use of the computer. In principle, the computer system is in a position to detect or prevent such manipulation. Integrity must therefore focus on the corrupt user as the principal problem. In fact the Trojan Horse problem can itself be viewed as a problem of corrupt system or application programmers, who improperly modify the software under their control. Also note that the problem of the corrupt user remains even if we are willing to trust all our software to be free of Trojan Horses.

## 6.2 Integrity Principles

In this section we identify basic *principles* for achieving data integrity. Principles lay down broad goals without specifying how to achieve them. In section 6.3 we will map these principles to DBMS *mechanisms*. Principles lay out *what* needs to be done while mechanisms establish *how* these principles are to be achieved.

Seven integrity principles are enumerated below.

1. *Well-formed Transactions.* The concept of the well-formed transaction is that users should not manipulate data arbitrarily, but only in restricted ways that preserve integrity of the database.

2. *Least Privilege.* Programs and users should be given the least privilege necessary to accomplish their task.

3. *Separation of Duties.* Separation of duties is a time honored principle for prevention of fraud and errors, going back to the very beginning of commerce. Simply stated, no single individual should be in a position to misappropriate assets on his own. Operationally this means that a chain of events which affects the balance of assets must require different individuals to be involved at key points, so that without their collusion the overall chain cannot take effect.

4. *Reconstruction of Events.* This principle seeks to deter improper behavior by threatening its discovery. The ability to reconstruct what happened in a system stems from the notion of accountability. Users are accountable for their actions to the extent that it is possible to determine what they did.

5. *Delegation of Authority.* This principle concerns the critical issue of how privileges are acquired and distributed in an organization. Clearly the procedures to do so must reflect the structure of the organization and allow for effective delegation of authority.

6. *Reality Checks.* Cross-checks with external reality are an essential part of integrity control. For example, if an internal inventory record does not correctly reflect the number of items in the warehouse, it makes little difference if the value of the recorded inventory is being correctly recorded in the balance sheet.

7. *Continuity of Operation.* This principle states that system operations should be maintained to some appropriate degree in the face of potentially devastating events which are beyond the organization's control. This catch-all description is intended to include natural disasters, power outages, disk crashes and the like. With this principle we are clearly stepping into the scope of availability. We have mentioned it here for the sake of completeness.

## 6.3 Integrity Mechanisms

We now consider DBMS mechanisms to facilitate application of the principles defined in the previous section.

### 6.3.1 Well-formed Transactions

The concept of a well-formed transaction corresponds very well to the standard DBMS concept of a transaction. A transaction is defined as a sequence of primitive actions which satisfies the following properties.

- *Correct state transform*: each transaction if run by itself in isolation and given a consistent state to begin with will leave the database in a consistent state.
- *Serializability*: the net effect of executing a set of transactions is equivalent to executing them in some sequential order, even though they may actually be executed concurrently (i.e., their actions are interleaved or simultaneous).
- *Failure atomicity*: either all or none of the updates of a transaction take effect. (We understand update to mean modification, i.e., it includes insertion of new data, deletion of existing data and changes to existing data.)
- *Progress*: every transaction will eventually complete, i.e., there is no indefinite blocking due to deadlock and no indefinite restarts due to livelocks.

The basic requirement is that the DBMS must ensure that updates are restricted to transactions. Clearly, if users are allowed to bypass transactions and directly manipulate relations in a database, we have no foundation to build upon. In other words updates should be encapsulated within transactions. This restriction may seem too strong, because in practice there will always be a need to perform ad hoc updates. However, ad hoc updates can themselves be carried out by means of special transactions! Of course the authorization for these special ad hoc transactions should be carefully controlled and their usage properly audited.

DBMS mechanisms can help in assuring the correctness of a state by enforcing *consistency constraints* on the data. Consistency constraints are also often called integrity constraints or integrity rules in the database literature. Since we are using integrity in a wider sense we prefer the former term.

The relational data model in particular imposes two consistency constraints.

- *Entity integrity* stipulates that attributes in the primary key of a relation cannot have NULL values. This amounts to requiring that each entity represented in the database must be uniquely identifiable.

- *Referential integrity* is concerned with references from one entity to another. A foreign key is a set of attributes in one relation whose values are required to match those of the primary key of some specific relation. Referential integrity requires that a foreign key either be all NULL or a matching tuple exist in the latter relation. This amounts to ruling out dangling references to non-existent entities.

Entity integrity is easily enforced. Referential integrity on the other hand requires more effort and has seen limited support in commercial products. The precise manner in which to achieve it is also very dependent on the semantics of the application. This is particularly so when the referenced tuple is deleted. There are several choices as follows: (i) prohibit this delete operation, (ii) delete the referencing tuple (with a possibility of further cascading deletes), or (iii) set the foreign key attributes in the referencing tuple to NULL.

The relational model in addition encourages the use of *domain constraints* whereby the values in a particular attribute (column) are constrained to come from some given set. These constraints are particularly easy to state and enforce, at least so long as the domains are defined in terms of primitive types such as integers, decimal numbers and character strings. A variety of *dependency constraints* which constrain the tuples in a given relation have been extensively studied in the database literature.

In the limit a consistency constraint can be viewed as an arbitrary predicate which all correct states of the database must satisfy. The predicate may involve any number of relations. Although this concept is theoretically appealing and flexible in its expressive power, in practice the overhead in checking the predicates for every transaction has been prohibitive. As a result relational DBMS's typically confine their enforcement of consistency constraints to domain constraints and entity integrity.

### 6.3.2 Least Privilege

The principle of least privilege translates into a requirement for fine-grained access control. For purpose of controlling read access DBMSs have employed mechanisms

| EMP | DEPT |
|---|---|
| Smith | Toy |
| Jones | Toy |
| Adams | Candy |

| DEPT | MANAGER |
|---|---|
| Toy | Brown |
| Candy | Baker |

Table 6: Base Relations EMP-DEPT and DEPT-MANAGER

| EMP | MANAGER |
|---|---|
| Smith | Brown |
| Jones | Brown |
| Adams | Baker |

Table 7: View EMPLOYEE-MANAGER

| EMP | MANAGER |
|---|---|
| Smith | Green |
| Jones | Brown |
| Adams | Baker |

Table 8: Updated View EMPLOYEE-MANAGER

based on views or query modification. These mechanisms are extremely flexible and can be as fine grained as desired. However, neither one of these mechanisms provides the same flexibility for fine-grained control of updates. The fundamental reason for this is our theoretical inability to translate updates on views into updates of base relations, in general. As a result authorization to control updates is often less sophisticated than authorization for read access.

To appreciate the difficulty in updating views, consider the two base relations shown in table 6, and the view shown in table 7. This view is created from the base relations as follows.

```
CREATE   VIEW EMP-MANAGER
AS       SELECT   EMP, MANAGER
         FROM     EMP-DEPT, DEPT-MANAGER
         WHERE    EMP-DEPT.DEPT = DEPT-MANAGER.DEPT
```

Consider the following UPDATE statement.

```
UPDATE   EMP-MANAGER
SET      MANAGER = 'Green'
WHERE    EMP = 'Smith'
```

If EMP-MANAGER was a base relation, this update would have the effect shown in table 8. This effect, however, cannot be attained by updating existing tuples in the base relations of table 6. Suppose we change the manager of the Toy department as follows.

```
UPDATE   DEPT-MANAGER
SET      MANAGER = 'Green'
WHERE    DEPT = 'Toy'
```

We will obtain the following view

| EMP | MANAGER |
|---|---|
| Smith | Green |
| Jones | Green |
| Adams | Baker |

where the manager of Jones has also been changed to Green. The view of table 8 can be realized by modifying the base relations of table 6 as follows.

| EMP | DEPT |
|---|---|
| Smith | X |
| Jones | Toy |
| Adams | Candy |

| DEPT | MANAGER |
|---|---|
| X | Green |
| Toy | Brown |
| Candy | Baker |

In this case Smith is assigned to some department, say X, whose manager is Green. It is, however, difficult to justify that this is the intended result of the original UPDATE on the view EMP-MANAGER. Moreover, the UPDATE statement gives us no clue as to what X might be.

For these reasons fine-grained control of updates by means of views does not work well in practice. View are, however, extremely useful for fine-grained control of retrieval.

### 6.3.3 Separation of Duties

Separation of duties finds little support in existing products. Although it is possible to use existing mechanisms for this purpose, these mechanisms have not been designed with this end in mind. As a result their use is awkward at best. Separation of duties is inherently concerned with sequences of transactions, rather than individual transactions in isolation. For example consider a situation in which payment in the form of a check is prepared and issued by the following sequence of events.

1. A clerk prepares a voucher and assigns an account.

2. The voucher and account are approved by a supervisor.

3. The check is issued by a clerk who must be different from the clerk in step 1. Issuing the check also debits the assigned account. (Strictly speaking we should debit one account and credit another in equal amounts. The important point for our purpose is that issuing a check modifies account balances.)

This sequence embodies separation of duties since the three steps must be executed by different people. The policy moreover has a dynamic flavor in that a particular clerk can prepare vouchers as well as, on different occasions, issue checks. However, a clerk cannot issue a check for a voucher prepared by himself.

### 6.3.4 Reconstruction of Events

The ability to reconstruct events in a system serves as a deterrent to improper behavior. In the DBMS context the mechanism to record the history of a system is traditionally called an audit trail. As with the principle of least privilege, a high-end DBMS should be capable of reconstructing events to the finest detail. In practise this ability must be tempered with the reality that gathering audit data indiscriminately can generate overwhelming volume. Therefore a DBMS must also allow fine-grained selectivity regarding what is audited. It should also structure the audit trail logically so that it is easy to query. For instance, logging every keystroke does give us the ability to reconstruct the system history accurately. However, with this primitive logical structure one needs substantial effort to reconstruct a particular transaction. In addition to the actual recording of all events that take place in the database, an audit trail must also provide support for true auditing, i.e., an audit trail must have the capability for an auditor to examine it in a systematic manner. In this respect DBMSs have a significant advantage, since their powerful querying abilities can be used for this purpose.

### 6.3.5 Delegation of Authority

The need to delegate authority and responsibility within an organization is essential to its smooth functioning. It appears in its most developed form with respect to monetary budgets. However the concept applies equally well to the control of other assets and resources of the organization.

In most organizations the ability to grant authorization is never completely unconstrained. For example, a department manger may be able to delegate substantial authority over departmental resources to project managers within his department and yet be prohibited to delegate this authority to project managers outside the department. Traditional delegation mechanisms based on the concept of ownership, e.g., as embodied in the SQL GRANT and REVOKE statements, are not adequate in this context. Further work remains to be done in this area.

### 6.3.6 Reality Checks

This principle inherently requires activity outside of the DBMS. The DBMS does have obligation to provide an internally consistent view of that portion of the database which is being externally verified. This is particularly so if the external inspection is conducted on an ad hoc on-demand basis.

### 6.3.7 Continuity of Operation

The basic technique to deal for maintaining continuity of operation in the face of natural disasters, hardware failures and other disruptive events, is redundancy in various forms. Recovery mechanisms in DBMS's must also ensure that we arrive at a consistent state.

## 6.4 Conclusion

The integrity principles, identified in section 6.3 can be divided into two groups as follows, on the basis of how well existing DBMS mechanisms can support them.

| Group I | Group II |
|---|---|
| Well-formed transactions | Least privilege |
| Continuity of operation | Separation of duties |
| Reality checks | Reconstruction of events |
|  | Delegation of authority |

Group I principles are adequately supported in existing products (to the extent that a DBMS can address these issues), whereas Group II principles are not so well understood and require improvement.

# 7 Examples

In this section we briefly review some representative commercial DBMS products, viz., DB2, Oracle, and dBase IV. Our objective is to illustrate how the general principles and concepts discussed in the previous sections have been applied in actual products. The products described here are complex and large pieces of software. The descriptions given below should be taken as high-level bird's eye perspectives of some salient features.

The examples in this section are limited to conventional DBMSs, with security features based on discretionary access controls. There are a number of development efforts to build DBMSs which incorporate the mandatory controls discussed in section 4. Most major DBMS vendors, with the notable exception of IBM, have formally announced some effort in this area. These include familiar names such as Informix, Ingres, Oracle, Sybase, Teradata, and DEC. Lesser known names include Atlantic Research Corporation (TRUDATA) and InfoSystems Technology Inc (Rubix).

## 7.1 DB2

DB2, or Database 2, is IBM's relational DBMS for the MVS (Multiple Virtual Storages) operating system. It provides a dialect of SQL and QBE (Query by Example) as its query languages. Users of DB2 first sign on to one of the following MVS subsystems: IMS (Information Management System), CICS (Customer Information Control System), or TSO (Time Sharing Option). DB2 identifies a user by means of a system administrator assigned authorization identifier (ID). DB2 relies on the individual MVS subsystems to authenticate the authorization ID.

The DB2 system catalog contains the metadata (i.e., data about data) of the system, including definitions of base tables, views, authorization IDs, access privileges, etc. The system catalog is itself stored as tables. Two important tables in the system catalog are SYSTABLES and SYSCOLUMNS. SYSTABLES contains an entry for each base table in the system specifying its name, creator, and number of columns among other things. SYSCOLUMNS contains an entry for each column giving its name, the name of the table to which it belongs and its domain.

Access to the databases as well as the system catalog can be controlled by views. The system keyword USER refers to the current user's authorization ID. The keyword PUBLIC refers to all authorization IDs. Keywords such as USER and PUBLIC can be used in view definitions and associated GRANTs. DB2 avoids the view update problem by restricting view updates to a subset of single-relation views.

There is one authorization ID which has the SYSADM (for system administrator) privilege. SYSADM is the highest privilege and includes all other privileges in the system. DBADM is another comprehensive privilege which confers the ability to execute any operation on a particular database.

The use of DB2's authorization and security mechanisms is optional; that is, they may be disabled if so desired. On the other hand, the controls of DB2 can be supplemented by those of MVS, IMS, CICS and TSO to provide additional protection.

## 7.2 ORACLE

The Oracle DBMS is a product of Oracle Corporation. It is available on a wide range of operating systems, including IBM's MVS, DEC's VMS, various flavors of Unix, MS-DOS etc. Reflecting this diversity, it comes in various modes such as multi-user, single-user, networked and distributed. Oracle supports its own dialect of SQL.

Each Oracle database has a collection of user accounts, each with a user name and password. Oracle defines three types basic privileges called Connect, Resource and DBA (Data Base Administrator). Users with the Connect privilege can log on to the database, and access and update those tables to which they have been granted appropriate access permissions. They cannot create tables, but they may create views. The Resource privilege allows users to create their own tables, for which access may be granted to other users. Users with the DBA privilege essentially have all privileges, including the ability to access and update any other user's tables. In particular, all user accounts are created and owned by DBA accounts.

An Oracle system is initially installed with three special accounts called Sys, System and Public. Sys and System both have DBA privileges. Together they own the tables and views comprising the Oracle data dictionary (which is analogous to the DB2 system catalog). All privileges granted by a user to the special user Public are automatically granted to all user accounts. Much like DB2, Oracle avoids the view update problem by restricting view updates to a subset of single-relation views.

### 7.3 dBase IV

dBase IV is a popular microcomputer DBMS. The dBase series of products provides a set of standard dBase commands, as a query language for defining, storing and manipulating data. In addition, dBase IV also provides an SQL interface. It is possible to configure dBase IV to control access on the basis of what can be done to an individual table, in terms of reading, changing, adding and deleting rows. It is also possible to control access on a column by column basis allowing read-write access, read-only access, or no access. This is usually done in a network environment. User accessing dBase IV over the network are asked to first log in. The valid users and their permissions are specified by the system administrator. Note that dBase IV controls access only to the base tables and not to views.

## 8 Summary

The problem of data security has three aspects: secrecy, integrity and availability.

A complete solution to the secrecy problem requires high-assurance multilevel systems, which impose mandatory controls and are known to be free of covert channels. Such systems are not currently available in the market, and are at the research and development stage. In due course of time products should become available. In the interim, security administrators must be aware of the limitations of discretionary access controls for achieving secrecy. Discretionary access controls cannot cope with Trojan Horse attacks. It is therefore important to ensure that only high quality software of known origin is used in the system. (This attitude also supports the integrity objectives.) Moreover, security administrators must appreciate that even the mandatory controls of high-assurance multilevel systems do not directly prevent inference of secret information.

The integrity problem, somewhat paradoxically, is less well understood than secrecy but is better supported in existing products. The basic foundation for integrity is to ensure that all updates are carried out by well-formed transactions. This facility is reasonably well supported by current products. On the other hand integrity principles such as least privilege, separation of duties and delegation of authority are not so well supported. These are still in the domain of research and development.

The availability problem is very poorly understood, and therefore existing products do not address it to any significant degree.

## Glossary

**Access control** The basic preventive technique for computer security, wherein the system examines every action and checks its conformance with the security policy before allowing it to occur.

**Assurance** The guarantee that a security mechanism is correctly implemented and cannot be subverted.

**Auditing** Examination of the system history recorded in an audit trail for purpose of detecting security breaches and/or determining the users responsible for security breaches.

**Authentication** The procedure used to establish the identity of a user to the system (or one system to another).

**Availability** The objective of preventing improper denial of access to information. Denial-of-service is a synonym for lack of availability.

**Cascading revocation** The indirect revocation of a user's privilege due to its revocation from the source from whom this user obtained it.

**Consistency constraints** Rules which determine whether or not a given database state is consistent in context of a given application. The relational model provides for entity and referential integrity constraints, as well as domain and dependency constraints. In the most general case a consistency constraint is an arbitrary predicate which all correct states of the database must satisfy.

**Covert channels** Indirect means of communication in a computer system which can be used to subvert the system's security policy.

**Data-dependent access controls** The authorization to access data is a function of the value of the data being accessed.

**Discretionary access controls** The granting of access is entirely under control of the users.

**Inference** The ability of users to deduce further information based on information obtained from the database, possibly combined with prior knowledge obtained by users from outside the database system. An inference presents a breach of secrecy if more highly classified information can be inferred from less classified information.

**Integrity** The objective of preventing improper modification of information or processes.

**Least privilege principle** Programs and users should be given the least privilege necessary to accomplish their task.

**Mandatory access controls** The granting of access is constrained by the system security policy. The following mandatory controls are required by the higher evaluation classes of the Orange Book: (i) Simple Security—a subject (i.e., a running program) with label X can read an object (i.e., a data item) with label Y only if X dominates Y, and (ii) Star-Property—a subject with label X can write an object with label Y only if Y dominates X.

**Orange Book** The popular name for the U.S. Department of Defense *Trusted Computer System Evaluation Criteria*, 1985. This document established a metric against which computers systems can be evaluated for security. The metric consists of a number of levels, A1, B3, B2, B1, C2, C1, and D; listed here in decreasing order of how secure the system is.

**Reality Checks** Cross-checks with external reality are an essential part of integrity control.

**Secrecy** The objective of preventing improper disclosure of information. Confidentiality and non-disclosure are synonyms for secrecy.

**Security labels** Security labels in the military and government sectors consist of two components: a hierarchical component and a (possibly empty) set of categories. A label on an object (e.g., a file) is called a security classification, while a label on a subject (e.g., a user) is called a security clearance. The label X is said to dominate label Y provided the hierarchical component of X is greater than or equal to the hierarchical component of Y, and the categories of X contain all the categories of Y. Labels X and Y are incomparable if neither one dominates the other.

**Separation of duties principle** A chain of events which affects the balance of assets must require different individuals to be involved at key points, so that without their collusion the overall chain cannot take effect.

**Trojan horse** Software which, in addition to the normal functions expected by its user, also engages in surreptitious actions to subvert security.

**View** A virtual relation which is derived from base relations and other views.

**Well-formed transaction** A sequence of primitive actions which if run by itself in isolation and given a consistent state to begin with will leave the database in a consistent state.

## Readings

Basic material on database systems can be found in the following books (among others).

- C.J. Date. *An Introduction to Database Systems, Volume I.* Fifth edition, Addison-Wesley, 1990.
- H.F. Korth and A. Silberschatz. *Database System Concepts.* Second edition, McGraw-Hill, 1991.

These books devote roughly a chapter to security. However, their view of security is limited to include only the discretionary access controls. Discussion of the SQL standard is available in the following book.

- C.J. Date. *A Guide to the SQL Standard.* Second edition, Addison-Wesley, 1989.

The following are some recent books on security.

- M. Gasser. *Building a Secure Computer System.* Van Nostrand Reinhold, 1988.
- C. Pfleeger. *Security in Computing.* Prentice-Hall, 1989.

These cover multilevel security but the coverage of database security issues is limited.

One complimentary copy of each document in the U.S. Department of Defense "Rainbow Series," which includes the Orange Book (*Trusted Computer System Evaluation Criteria*) and the TDI (*Trusted Database Interpretation of the Trusted Computer System Evaluation Criteria*), is available from:

INFOSEC Awareness Division
Attn: IAOC
Ft. George G. Meade, MD 20755-6000
USA
Telephone: +1 410-766-8729

# Appendix: Relational Databases and SQL

This appendix gives a brief review of essential concepts of relational databases and SQL.

## Relational Databases

A relational database stores data in relations which have well defined mathematical properties. Each *relation* $R$ has two parts as follows.

1. A state-invariant *relation scheme* $R(A_1, A_2, \ldots, A_n)$, where each $A_i$ is an *attribute* over some *domain* $D_i$ which is a set of values.

2. A state-dependent *relation instance* $R$, which is a set of distinct *tuples* of the form $(a_1, a_2, \ldots, a_n)$ where each *element* $a_i$ is a value in domain $D_i$, or $a_i$ is NULL (i.e., undefined).

Roughly speaking, a relation can be thought of as a table, and is often shown as such. The attributes correspond to columns, and tuples correspond to rows. The relation scheme gives us the names of the columns and the permissible values (i.e., the domain) for each column. The relation instance gives us the rows of the table at a given instant. For example, the relation scheme for the EMPLOYEE relation of Table 1 (in section 3) is

EMPLOYEE(NAME, DEPT, SALARY, MANAGER)

with the domain of NAME, DEPT and MANAGER as character strings, and the domain of SALARY as integers. Table 1 show a particular instance of the EMPLOYEE relation, reflecting the employees who are currently employed.

It should be noted that in a relation there is no significance to the order of the columns or rows. Also a relation does not allow duplicate rows (i.e., two rows which have identical values for all columns).

Let $X$ and $Y$ denote sets of one or more of the attributes $A_i$ in a relation scheme. We say $Y$ is *functionally dependent* on $X$, written $X \rightarrow Y$, if and only if it is not possible to have two tuples with the same values for $X$ but different values for $Y$. A *candidate key* of a relation is a minimal set of attributes on which all other attributes are functionally dependent. Intuitively, it is not allowed to have two tuples with the same values of the candidate key in a relation instance. A candidate key is required to be minimal, in the sense that no attribute can be discarded without destroying this property. It is guaranteed that a candidate key always exists, since in the absence of any functional dependencies it consists of the entire set of attributes. In general, there can be more than one candidate key for a relation with a given collection of functional dependencies.

In the EMPLOYEE relation of Table 1 it would appear that NAME is the only candidate key. This, of course, assumes there are no duplicate names. (Realistically one would use something like the social security number or employee identity number as the key.) For sake of example, further assume that duplicate salaries are not allowed in this company. In that case SALARY would also be a candidate key. Note that the relation instance of Table 1 actually satisfies this requirement. The identification of the candidate key is, however, a property of the relation scheme and would apply to every instance; not merely to the particular one that happens to exist at a given moment.

The *primary key* of a relation is one of its candidate keys which has been specifically designated as such. The primary key serves the purpose of uniquely identifying a specific tuple from a relation instance. It also serves the purpose of linking relations together. The relational model incorporates two application independent integrity rules, called *entity integrity* and *referential integrity*, respectively to ensure these purposes are properly served.

Entity integrity simply requires that no tuple in a relation instance can have NULL (i.e., undefined) values for any of the primary key attributes. This property guarantees that each tuple will be uniquely identifiable.

Referential integrity is concerned with references from one relation to another. To understand this property in context of Table 1 let us suppose there is a second relation with the scheme

DEPARTMENT(DEPT, LOCATION, BUDGET)

with primary key DEPT. The DEPT attribute of the EMPLOYEE relation is said to be a *foreign key.* In general a foreign key is an attribute (or set of attributes) in one relation $R_1$ whose values are required to match those of the primary key of a tuple in some other relation $R_2$. $R_1$ and $R_2$ need not be distinct. In fact, since managers are employees, the MANAGER attribute in EMPLOYEE is a foreign key with $R_1 = R_2 =$ EMPLOYEE.

Referential integrity stipulates that if a foreign key $FK$ of relation $R_1$ is the primary key $PK$ of $R_2$, then for every tuple in $R_1$ the value of $FK$ must either be NULL or equal to the value of $PK$ of a tuple in $R_2$. In context of Table 1, and the above discussion, referential integrity requires the following.

- Due to the DEPT foreign key, there should be tuples for the Toy, Admin, and Candy departments in the DEPARTMENT relation.

- Due to the MANAGER foreign key, there should be tuples for Jones, Baker and Harding in the EMPLOYEE relation.

The motivation of referential integrity is to prevent employees from being assigned to departments or managers who do not exist in the database. Note that it is all right for employee Harding to have a NULL manager. It would similarly be acceptable for an employee to have a NULL department.

## SQL

Every DBMS needs a language for defining, storing, retrieving, and manipulating data. SQL is the de facto standard, for this purpose, in relational DBMSs. SQL emerged from several projects at the IBM San Jose (now called Almaden) Research Center in the mid-1970s. The name SQL was originally an abbreviation for Structured Query Language. Most relational DBMSs today support some dialect of SQL. There is also an official standard approved by the American National Standards Institute (ANSI) in 1986, and accepted by the International Standards Organization (ISO) in 1987. Our objective here is to explain SQL in just enough detail to understand the examples in this chapter.

Consider the EMPLOYEE relation of Table 1 (in section 3). The relation scheme is defined in SQL by the following command.

```
CREATE  TABLE         EMPLOYEE
      ( NAME          CHARACTER    NOT NULL,
        DEPT          CHARACTER,
        SALARY        INTEGER,
        MANAGER       CHARACTER,
        PRIMARY KEY   (NAME),
        FOREIGN KEY   (DEPT)       REFERENCES DEPARTMENT,
        FOREIGN KEY   (MANAGER)    REFERENCES EMPLOYEE )
```

This statement creates a table called EMPLOYEE with four columns. The NAME, DEPT and MANAGER columns have character strings (of unspecified length) as values, whereas the SALARY column has integer values. NAME is the primary key. DEPT is a foreign key which reference the primary key of table DEPARTMENT. MANAGER is a foreign key which references the primary key (i.e., NAME) of the EMPLOYEE table itself.

The EMPLOYEE table is initially empty. Tuples are inserted into it by means of the SQL INSERT statement. For example, the last tuple of table 1 is inserted by the following statement.

```
INSERT
INTO     EMPLOYEE(NAME, DEPT, SALARY, MANAGER)
VALUES   VALUES('Harding', 'Admin', 50000, NULL)
```

The remaining tuples can be similarly inserted. Insertion of the tuples for Baker, Jones and Smith must occur in this order, so as to maintain referential integrity. (Alternately, these tuples can be inserted in any order with NULL managers which are later updated to their actual values.) There is a DELETE statement to delete tuples from a relation.

Retrieval of data is effected in SQL by the SELECT statement. For example, the NAME, SALARY and MANAGER data for employees in the Toy department is extracted as follows.

```
SELECT   NAME, SALARY, MANAGER
FROM     EMPLOYEE
WHERE    DEPT = 'Toy'
```

This query applied to Table 1 returns the data shown in Table 2.

The WHERE clause in a SELECT statement is optional. SQL also allows the retrieved records to be grouped together for statistical computations by means of built-in statistical functions. For example, the following query gives the average salary for employees in each department.

```
SELECT      DEPT, AVG(SALARY)
FROM        EMPLOYEE
GROUP BY    DEPT
```

Data from two or more relations can be retrieved and linked together in a SELECT statement. For example, the location of employees can be retrieved by linking the data in EMPLOYEE with that in DEPARTMENT, as follows.

```
SELECT   NAME, LOCATION
FROM     EMPLOYEE, DEPARTMENT
WHERE    EMPLOYEE.DEPT = DEPARTMENT.DEPT
```

This query will attempt to match every tuple in EMPLOYEE with every tuple in DEPARTMENT, but will select only those pairs for which the DEPT attribute in the EMPLOYEE tuple matches the DEPT attribute in the DEPARTMENT tuple. (Since DEPT is a common attribute to both relations, every use of it is explicitly identified as occurring with respect to one of the two relations.)

Finally the UPDATE statement allows one or more attributes of existing tuples in a relation to be modified. For example, the following statement gives all employees in the Toy department a raise of $1000.

```
UPDATE   EMPLOYEE
SET      SALARY = SALARY + 1000
WHERE    DEPT = 'Toy'
```

This statement selects those tuples in EMPLOYEE which have the value of Toy for the DEPT attribute. It then increases the value of the SALARY attribute for all these tuples by $1000 each.

# Contents