CHAPTER 5

DATABASE SECURITY

Learning Objectives

After studying this chapter, you should be able to:

- ◆ Understand the unique need for database security, separate from ordinary computer security measures.
- ◆ Present an overview of the basic elements of a database management system.
- ◆ Present an overview of the basic elements of a relational database system.
- ◆ Compare and contrast different approaches to database access control.
- ◆ Explain how inference poses a security threat in database systems.
- ◆ Understand the nature of statistical databases and their related security issues.
- ◆ Discuss the use of encryption in a database system.
- ◆ Understand the unique security issues related to cloud computing.

This chapter looks at the unique security issues that relate to databases. The focus of this chapter is on relational database management systems (RDBMS). The relational approach dominates industry, government, and research sectors and is likely to do so for the foreseeable future. We begin with an overview of the need for database-specific security techniques. Then we provide a brief introduction to database management systems, followed by an overview of relational databases. Next, we look at the issue of database access control, followed by a discussion of the inference threat. Then we examine security issues for statistical databases. Next, we examine database encryption. Finally, we examine the issues raised by the use of cloud technology.

5.1 THE NEED FOR DATABASE SECURITY

Organizational databases tend to concentrate sensitive information in a single logical system. Examples include:

- Corporate financial data
- Confidential phone records
- Customer and employee information, such as name, Social Security number, bank account information, credit card information
- Proprietary product information
- Health care information and medical records

For many businesses and other organizations, it is important to be able to provide customers, partners, and employees with access to this information. But such information can be targeted by internal and external threats of misuse or unauthorized change. Accordingly, security specifically tailored to databases is an increasingly important component of an overall organizational security strategy.

[BENN06] cites the following reasons why database security has not kept pace with the increased reliance on databases:

1. There is a dramatic imbalance between the complexity of modern database management systems (DBMS) and the security techniques used to protect these critical systems. A DBMS is a very complex, large piece of software, providing many options, all of which need to be well understood and then secured to avoid data breaches. Although security techniques have advanced, the increasing complexity of the DBMS—with many new features and services—has brought a number of new vulnerabilities and the potential for misuse.
2. Databases have a sophisticated interaction protocol called the Structured Query Language (SQL), which is far more complex, for example, than the HTTP Protocol used to interact with a Web service. Effective database security requires a strategy based on a full understanding of the security vulnerabilities of SQL.
3. The typical organization lacks full-time database security personnel. The result is a mismatch between requirements and capabilities. Most organizations have a staff of database administrators, whose job is to manage the database to ensure availability, performance, correctness, and ease of use. Such administrators may have limited knowledge of security and little available time to master and apply security techniques. On the other hand, those responsible for security within an organization may have very limited understanding of database and DBMS technology.
4. Most enterprise environments consist of a heterogeneous mixture of database platforms (Oracle, IBM DB1 and Informix, Microsoft, Sybase, etc.), enterprise platforms (Oracle E-Business Suite, PeopleSoft, SAP, Siebel, etc.), and OS platforms (UNIX, Linux, z/OS, and Windows, etc.). This creates an additional complexity hurdle for security personnel.

An additional recent challenge for organizations is their increasing reliance on cloud technology to host part or all of the corporate database. This adds an additional burden to the security staff.

5.2 DATABASE MANAGEMENT SYSTEMS

In some cases, an organization can function with a relatively simple collection of files of data. Each file may contain text (e.g., copies of memos and reports) or numerical data (e.g., spreadsheets). A more elaborate file consists of a set of records. However, for an organization of any appreciable size, a more complex structure known as a database is required. A **database** is a structured collection of data stored for use by one or more applications. In addition to data, a database contains the relationships between data items and groups of data items. As an example of the distinction between data files and a database, consider the following. A simple personnel file might consist of a set of records, one for each employee. Each record gives the employee's name, address, date of birth, position, salary, and other details needed by the personnel department. A personnel database includes a personnel file, as just described. It may also include a time and attendance file, showing for each week the hours worked by each employee. With a database organization, these two files are tied together so that a payroll program can extract the information about time worked and salary for each employee to generate paychecks.

Accompanying the database is a **database management system (DBMS)**, which is a suite of programs for constructing and maintaining the database and for offering ad hoc query facilities to multiple users and applications. A **query language** provides a uniform interface to the database for users and applications.

Figure 5.1 provides a simplified block diagram of a DBMS architecture. Database designers and administrators make use of a data definition language (DDL) to define the database logical structure and procedural properties, which are represented by a set of database description tables. A data manipulation language (DML) provides a powerful set of tools for application developers.Query languages are declarative languages designed to support end users. The database management system makes use of the database description tables to manage the physical database. The interface to the database is through a file manager module and a transaction manager module. In addition to the database description table, two other tables support the DBMS. The DBMS uses authorization tables to ensure the user has permission to execute the query language statement on the database. The concurrent access table prevents conflicts when simultaneous, conflicting commands are executed.

Database systems provide efficient access to large volumes of data and are vital to the operation of many organizations. Because of their complexity and criticality, database systems generate security requirements that are beyond the capability of typical OS-based security mechanisms or stand-alone security packages.

Operating system security mechanisms typically control read and write access to entire files. So they could be used to allow a user to read or to write any information in, for example, a personnel file. But they could not be used to limit

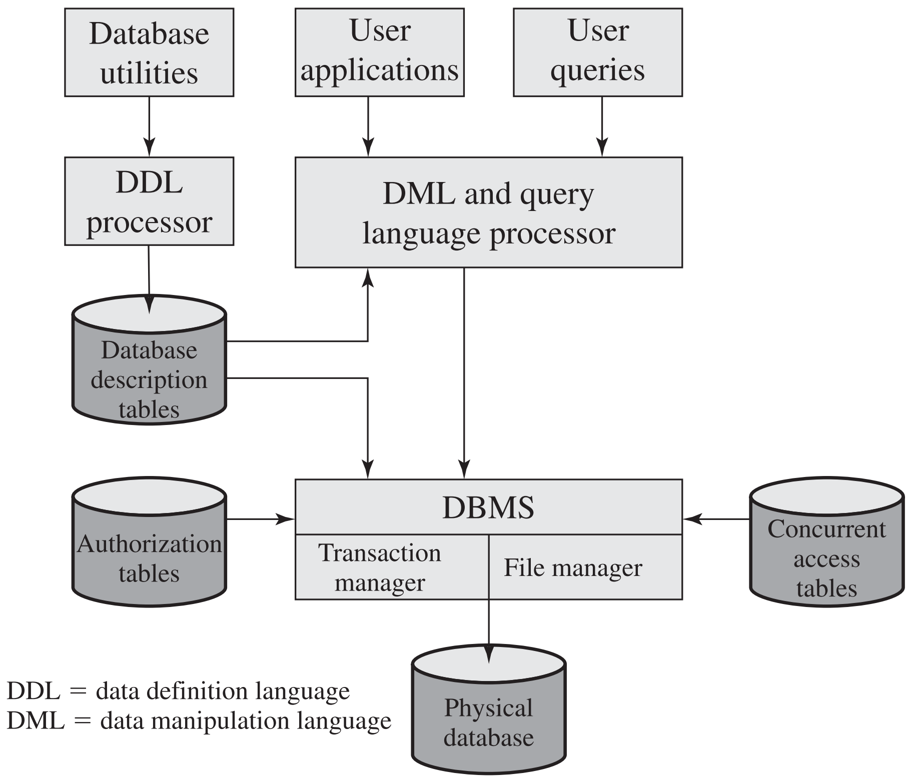

Figure 5.1 **DBMS Architecture**

access to specific records or fields in that file. A DBMS typically does allow this type of more detailed access control to be specified. It also usually enables access controls to be specified over a wider range of commands, such as to select, insert, update, or delete specified items in the database. Thus, security services and mechanisms are needed that are designed specifically for, and integrated with, database systems.

5.3 RELATIONAL DATABASES

The basic building block of a relational database is a table of data, consisting of rows and columns, similar to a spreadsheet. Each column holds a particular type of data, while each row contains a specific value for each column. Ideally, the table has at least one column in which each value is unique, thus serving as an identifier for a given entry. For example, a typical telephone directory contains one entry for each subscriber, with columns for name, telephone number, and address. Such a table is called a flat file because it is a single two-dimensional (rows and columns) file. In a flat file, all of the data are stored in a single table. For the telephone directory, there might be a number of subscribers with the same name, but the telephone numbers should be unique, so that the telephone number serves as a unique identifier for a row. However, two or more people sharing the same phone number might each be listed in the directory. To continue to hold all of the data for the telephone directory in a single table and to provide for a unique identifier for each row, we could require a separate column for secondary subscriber, tertiary subscriber, and so on. The result would be that for each telephone number in use, there is a single entry in the table.

The drawback of using a single table is that some of the column positions for a given row may be blank (not used). Also, any time a new service or new type of information is incorporated in the database, more columns must be added and the database and accompanying software must be redesigned and rebuilt.

The relational database structure enables the creation of multiple tables tied together by a unique identifier that is present in all tables. Figure 5.2 shows how new services and features can be added to the telephone database without reconstructing the main table. In this example, there is a primary table with basic information for each telephone number. The telephone number serves as a primary key. The database administrator can then define a new table with a column for the primary key and other columns for other information.

Users and applications use a relational query language to access the database. The query language uses declarative statements rather than the procedural instructions of a programming language. In essence, the query language allows the user to request selected items of data from all records that fit a given set of criteria. The software then figures out how to extract the requested data from one or more tables. For example, a telephone company representative could retrieve a subscriber's billing information as well as the status of special services or the latest payment received, all displayed on one screen.

Elements of a Relational Database System

In relational database parlance, the basic building block is a **relation**, which is a flat table. Rows are referred to as **tuples**, and columns are referred to as **attributes**

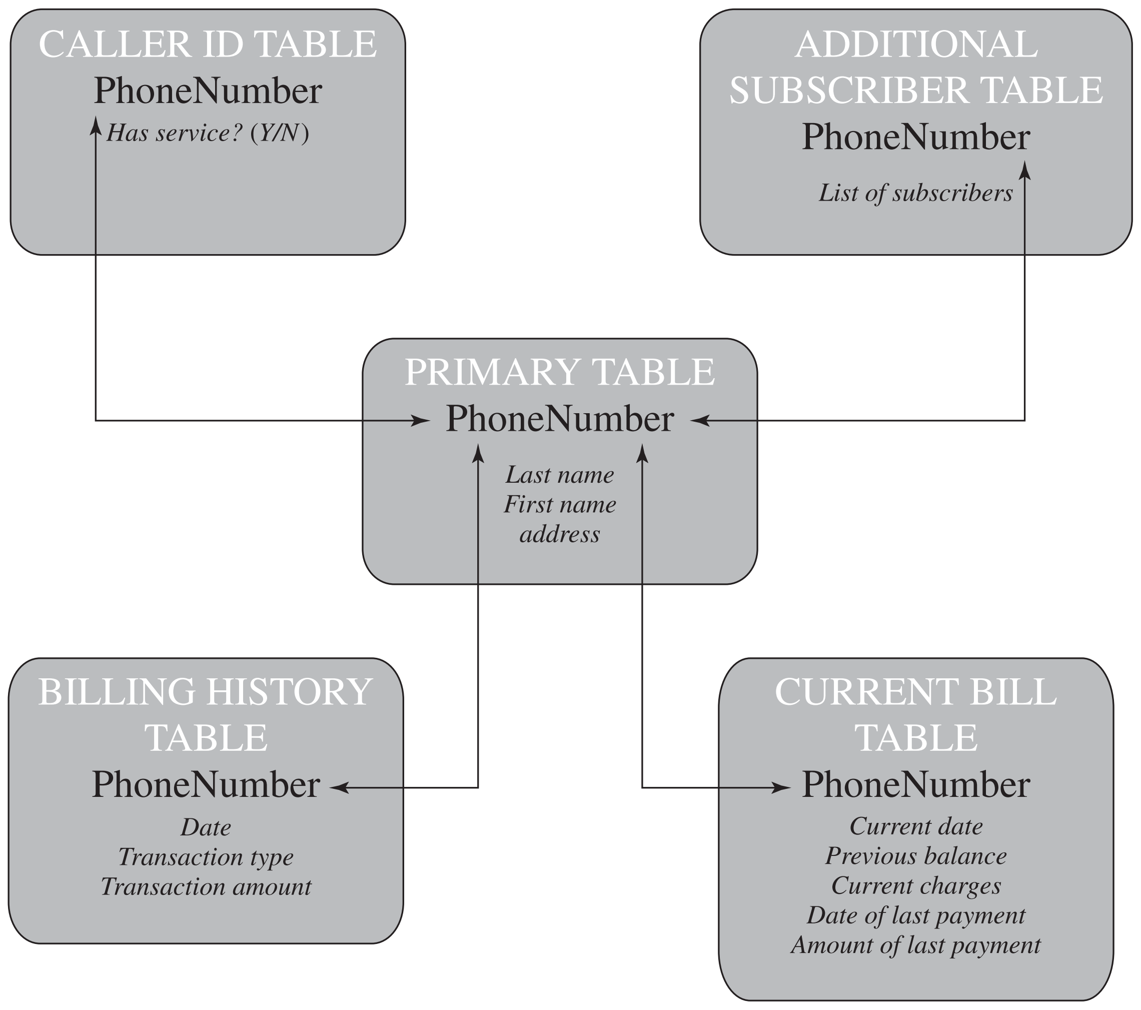

Figure 5.2 **Example Relational Database Model.** A relational database uses multiple tables related to one another by a designated key; in this case the key is the PhoneNumber field.

(Table 5.1). A **primary key** is defined to be a portion of a row used to uniquely identify a row in a table; the primary key consists of one or more column names. In the example of Figure 5.2, a single attribute, PhoneNumber, is sufficient to uniquely identify a row in a particular table.

To create a relationship between two tables, the attributes that define the primary key in one table must appear as attributes in another table, where they are referred to as a **foreign key**. Whereas the value of a primary key must be unique for each tuple (row) of its table, a foreign key value can appear multiple times in a table, so that there is a one-to-many relationship between a row in the table with

Table 5.1 Basic Terminology for Relational Databases

Formal Name	Common Name	Also Known As
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

the primary key and rows in the table with the foreign key. Figure 5.3a provides an example. In the Department table, the department ID (*Did*) is the primary key; each value is unique. This table gives the ID, name, and account number for each department. The Employee table contains the name, salary code, employee ID, and phone number of each employee. The Employee table also indicates the department to which each employee is assigned by including *Did*. *Did* is identified as a foreign key and provides the relationship between the Employee table and the Department table.

A **view** is a virtual table. In essence, a view is the result of a query that returns selected rows and columns from one or more tables. Figure 5.3b is a view that includes the employee name, ID, and phone number from the Employee table and the corresponding department name from the Department table. The linkage is the *Did*, so that the view table includes data from each row of the Employee table, with additional data from the Department table. It is also possible to construct a view from a single table. For example, one view of the Employee table consists of all rows, with the salary code column deleted. A view can be qualified to include only some rows and/or some columns. For example, a view can be defined consisting of all rows in the Employee table for which the *Did* = 15.

Views are often used for security purposes. A view can provide restricted access to a relational database so that a user or application only has access to certain rows or columns.

Department Table

Did	Dname	Dacctno
4	human resources	528221
8	education	202035
9	accounts	709257
13	public relations	755827
15	services	223945

Primary key

Employee Table

Ename	Did	Salarycode	Eid	Ephone
Robin	15	23	2345	6127092485
Neil	13	12	5088	6127092246
Jasmine	4	26	7712	6127099348
Cody	15	22	9664	6127093148
Holly	8	23	3054	6127092729
Robin	8	24	2976	6127091945
Smith	9	21	4490	6127099380

Foreign key

Primary key

(a) Two tables in a relational database

Dname	Ename	Eid	Ephone
human resources	Jasmine	7712	6127099348
education	Holly	3054	6127092729
education	Robin	2976	6127091945
accounts	Smith	4490	6127099380
public relations	Neil	5088	6127092246
services	Robin	2345	6127092485
services	Cody	9664	6127093148

(b) A view derived from the database

Figure 5.3 **Relational Database Example**

Structured Query Language

Structured Query Language (SQL), originally developed by IBM in the mid-1970s, is a standardized language that can be used to define schema, manipulate, and query data in a relational database. There are several versions of the ANSI/ISO standard and a variety of different implementations, but all follow the same basic syntax and semantics.

For example, the two tables in Figure 5.3a are defined as follows:

```
CREATE TABLE department (
   Did INTEGER PRIMARY KEY,
   Dname CHAR (30),
   Dacctno CHAR (6) )
CREATE TABLE employee (
   Ename CHAR (30),
   Did INTEGER,
   SalaryCode INTEGER,
   Eid INTEGER PRIMARY KEY,
   Ephone CHAR (10),
   FOREIGN KEY (Did) REFERENCES department (Did) )
```

The basic command for retrieving information is the SELECT statement. Consider this example:

```
SELECT Ename, Eid, Ephone
   FROM Employee
   WHERE Did = 15
```

This query returns the Ename, Eid, and Ephone fields from the Employee table for all employees assigned to department 15.

The view in Figure 5.3b is created using the following SQL statement:

```
CREATE VIEW newtable (Dname, Ename, Eid, Ephone)
AS SELECT D.Dname E.Ename, E.Eid, E.Ephone
FROM Department D Employee E
WHERE E.Did = D.Did
```

The preceding are just a few examples of SQL functionality. SQL statements can be used to create tables, insert and delete data in tables, create views, and retrieve data with query statements.

5.4 DATABASE ACCESS CONTROL

Commercial and open-source DBMSs typically provide an access control capability for the database. The DBMS operates on the assumption that the computer system has authenticated each user. As an additional line of defense, the computer system may use the overall access control system described in Chapter 4 to determine whether a user may have access to the database as a whole. For users who are authenticated and granted access to the database, a database access control system provides a specific capability that controls access to portions of the database.

Commercial and open-source DBMSs provide discretionary or role-based access control. We defer a discussion of mandatory access control considerations to Chapter 13. Typically, a DBMS can support a range of administrative policies, including the following:

- **Centralized administration:** A small number of privileged users may grant and revoke access rights.
- **Ownership-based administration:** The owner (creator) of a table may grant and revoke access rights to the table.
- **Decentralized administration:** In addition to granting and revoking access rights to a table, the owner of the table may grant and revoke authorization rights to other users, allowing them to grant and revoke access rights to the table.

As with any access control system, a database access control system distinguishes different access rights, including create, insert, delete, update, read, and write. Some DBMSs provide considerable control over the granularity of access rights. Access rights can be to the entire database, to individual tables, or to selected rows or columns within a table. Access rights can be determined based on the contents of a table entry. For example, in a personnel database, some users may be limited to seeing salary information only up to a certain maximum value. And a department manager may only be allowed to view salary information for employees in his or her department.

SQL-Based Access Definition

SQL provides two commands for managing access rights, GRANT and REVOKE. For different versions of SQL, the syntax is slightly different. In general terms, the GRANT command has the following syntax:[1]

```
GRANT              { privileges | role }
[ON                table]
TO                 { user | role | PUBLIC }
[IDENTIFIED BY     password]
[WITH              GRANT OPTION]
```

This command can be used to grant one or more access rights or can be used to assign a user to a role. For access rights, the command can optionally specify that it applies only to a specified table. The TO clause specifies the user or role to which the rights are granted. A PUBLIC value indicates that any user has the specified access rights. The optional IDENTIFIED BY clause specifies a password that must be used to revoke the access rights of this GRANT command. The GRANT

[1]The following syntax definition conventions are used. Elements separated by a vertical line are alternatives. A list of alternatives is grouped in curly brackets. Square brackets enclose optional elements. That is, the elements inside the square brackets may or may not be present.

OPTION indicates that the grantee can grant this access right to other users, with or without the grant option.

As a simple example, consider the following statement.

```
GRANT SELECT ON ANY TABLE TO ricflair
```

This statement enables user ricflair to query any table in the database.

Different implementations of SQL provide different ranges of access rights. The following is a typical list:

- Select: Grantee may read entire database; individual tables; or specific columns in a table.
- Insert: Grantee may insert rows in a table; or insert rows with values for specific columns in a table.
- Update: Semantics is similar to INSERT.
- Delete: Grantee may delete rows from a table.
- References: Grantee is allowed to define foreign keys in another table that refer to the specified columns.

The REVOKE command has the following syntax:

```
REVOKE          { privileges | role }
[ON             table]
FROM            { user | role | PUBLIC }
```

Thus, the following statement revokes the access rights of the preceding example:

```
REVOKE SELECT ON ANY TABLE FROM ricflair
```

Cascading Authorizations

The grant option enables an access right to cascade through a number of users.We consider a specific access right and illustrate the cascade phenomenonin Figure 5.4. The figure indicates that Ann grants the access right to Bob at time $t = 10$ and to Chris at time $t = 20$. Assume that the grant option is always used. Thus, Bob is able to grant the access right to David at $t = 30$. Chris redundantly grants the access right to David at $t = 50$. Meanwhile, David grants the right to Ellen, who in turn grants it to Jim; and subsequently David grants the right to Frank.

Just as the granting of privileges cascades from one user to another using the grant option, the revocation of privileges also cascaded. Thus, if Ann revokes the access right to Bob and Chris, then the access right is also revoked to David, Ellen, Jim, and Frank. A complication arises when a user receives the same access right multiple times, as happens in the case of David. Suppose that Bob revokes the privilege from David. David still has the access right because it was granted by Chris at $t = 50$. However, David granted the access right to Ellen after receiving the right, with grant option, from Bob but prior to receiving it from Chris.Most implementations dictate that in this circumstance, the access

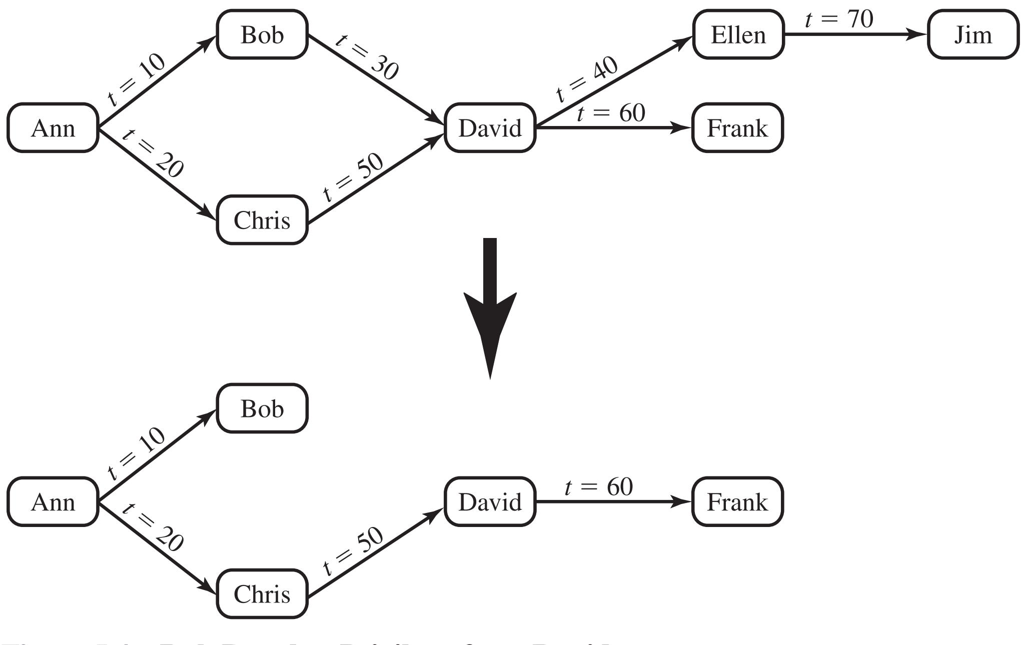

Figure 5.4 **Bob Revokes Privilege from David**

right to Ellen and therefore Jim is revoked when Bob revokes the access right to David. This is because at $t = 40$, when David granted the access right to Ellen, David only had the grant option to do this from Bob. When Bob revokes the right, this causes all subsequent cascaded grants that are traceable solely to Bob via David to be revoked. Because David granted the access right to Frank after David was granted the access right with grant option from Chris, the access right to Frank remains. These effects are shown in the lower portion of Figure 5.4.

To generalize, the convention followed by most implementations is as follows. When user A revokes an access right, any cascaded access right is also revoked, unless that access right would exist even if the original grant from A had never occurred. This convention was first proposed in [GRIF76].

Role-Based Access Control

A role-based access control (RBAC) scheme is a natural fit for database access control. Unlike a file system associated with a single or a few applications, a database system often supports dozens of applications. In such an environment, an individual user may use a variety of applications to perform a variety of tasks, each of which requires its own set of privileges. It would be poor administrative practice to simply grant users all of the access rights they require for all the tasks they perform. RBAC provides a means of easing the administrative burden and improving security.

In a discretionary access control environment, we can classify database users in three broad categories:

- **Application owner:** An end user who owns database objects (tables, columns, rows) as part of an application. That is, the database objects are generated by the application or are prepared for use by the application.

- **End user other than application owner:** An end user who operates on database objects via a particular application but does not own any of the database objects.
- **Administrator:** User who has administrative responsibility for part or all of the database.

We can make some general statements about RBAC concerning these three types of users. An application has associated with it a number of tasks, with each task requiring specific access rights to portions of the database. For each task, one or more roles can be defined that specify the needed access rights. The application owner may assign roles to end users. Administrators are responsible for more sensitive or general roles, including those having to do with managing physical and logical database components, such as data files, users, and security mechanisms. The system needs to be set up to give certain administrators certain privileges. Administrators in turn can assign users to administrative-related roles.

A database RBAC facility needs to provide the following capabilities:

- Create and delete roles.
- Define permissions for a role.
- Assign and cancel assignment of users to roles.

A good example of the use of roles in database security is the RBAC facility provided by Microsoft SQL Server. SQL Server supports three types of roles: server roles, database roles, and user-defined roles. The first two types of roles are referred to as fixed roles (Table 5.2); these are preconfigured for a system with specific access rights. The administrator or user cannot add, delete, or modify fixed roles; it is only possible to add and remove users as members of a fixed role.

Fixed server roles are defined at the server level and exist independently of any user database. They are designed to ease the administrative task. These roles have different permissions and are intended to provide the ability to spread the administrative responsibilities without having to give out complete control. Database administrators can use these fixed server roles to assign different administrative tasks to personnel and give them only the rights they absolutely need.

Fixed database roles operate at the level of an individual database. As with fixed server roles, some of the fixed database roles, such as db_accessadmin and db_securityadmin, are designed to assist a DBA with delegating administrative responsibilities. Others, such as db_datareader and db_datawriter, are designed to provide blanket permissions for an end user.

SQL Server allows users to create roles. These **user-defined roles** can then be assigned access rights to portions of the database. A user with proper authorization (typically, a user assigned to the db_securityadmin role) may define a new role and associate access rights with the role. There are two types of user-defined roles: standard and application. For a standard role, an authorized user can assign other users to the role. An application role is associated with an application rather than with a group of users and requires

Table 5.2 Fixed Roles in Microsoft SQL Server

Role	Permissions
Fixed Server Roles	
sysadmin	Can perform any activity in SQL Server and have complete control over all database functions
serveradmin	Can set server-wide configuration options, shut down the server
setupadmin	Can manage linked servers and startup procedures
securityadmin	Can manage logins and CREATE DATABASE permissions, also read error logs and change passwords
processadmin	Can manage processes running in SQL Server
Dbcreator	Can create, alter, and drop databases
diskadmin	Can manage disk files
bulkadmin	Can execute BULK INSERT statements
Fixed Database Roles	
db_owner	Has all permissions in the database
db_accessadmin	Can add or remove user IDs
db_datareader	Can select all data from any user table in the database
db_datawriter	Can modify any data in any user table in the database
db_ddladmin	Can issue all data definition language statements
db_securityadmin	Can manage all permissions, object ownerships, roles and role memberships
db_backupoperator	Can issue DBCC, CHECKPOINT, and BACKUP statements
db_denydatareader	Can deny permission to select data in the database
db_denydatawriter	Can deny permission to change data in the database

a password. The role is activated when an application executes the appropriate code. A user who has access to the application can use the application role for database access. Often database applications enforce their own security based on the application logic. For example, you can use an application role with its own password to allow the particular user to obtain and modify any data only during specific hours. Thus, you can realize more complex security management within the application logic.

5.5 INFERENCE

Inference, as it relates to database security, is the process of performing authorized queries and deducing unauthorized information from the legitimate responses received. The inference problem arises when the combination of a number of data items is more sensitive than the individual items, or when a combination of data items can be used to infer data of a higher sensitivity. Figure 5.5 illustrates the process. The attacker may make use of nonsensitive data as well as metadata.

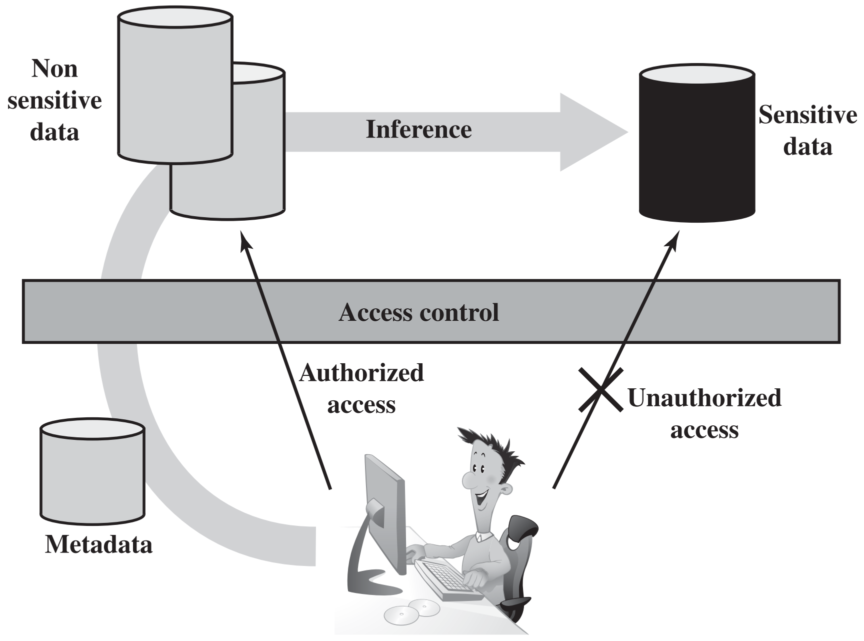

Figure 5.5 Indirect Information Access via Inference Channel

Metadata refers to knowledge about correlations or dependencies among data items that can be used to deduce information not otherwise available to a particular user. The information transfer path by which unauthorized data is obtained is referred to as an **inference channel**.

In general terms, two inference techniques can be used to derive additional information: analyzing functional dependencies between attributes within a table or across tables, and merging views with the same constraints.

An example of the latter shown in Figure 5.6, illustrates the inference problem. Figure 5.6a shows an Inventory table with four columns. Figure 5.6b shows two views, defined in SQL as follows:

```
CREATE view V1 AS              CREATE view V2 AS
SELECT Availability, Cost        SELECT Item, Department
FROM Inventory                  FROM Inventory
WHERE Department = ″hardware″   WHERE Department = ″hardware″
```

Users of these views are not authorized to access the relationship between Item and Cost. A user who has access to either or both views cannot infer the relationship by functional dependencies. That is, there is not a functional relationship between Item and Cost such that knowing Item and perhaps other information is sufficient to deduce Cost. However, suppose the two views are created with the access constraint that Item and Cost cannot be accessed together. A user who knows the structure of the Inventory table and who knows that the view tables maintain the same row order as the Inventory table is then able to merge the two views to construct the table shown in Figure 5.6c. This violates the access control policy that the relationship of attributes Item and Cost must not be disclosed.

Item	Availability	Cost ($)	Department
Shelf support	in-store/online	7.99	hardware
Lid support	online only	5.49	hardware
Decorative chain	in-store/online	104.99	hardware
Cake pan	online only	12.99	housewares
Shower/tub cleaner	in-store/online	11.99	housewares
Rolling pin	in-store/online	10.99	housewares

(a) Inventory table

Availability	Cost ($)
in-store/online	7.99
online only	5.49
in-store/online	104.99

Item	Department
Shelf support	hardware
Lid support	hardware
Decorative chain	hardware

(b) Two views

Item	Availability	Cost ($)	Department
Shelf support	in-store/online	7.99	hardware
Lid support	online only	5.49	hardware
Decorative chain	in-store/online	104.99	hardware

(c) Table derived from combining query answers

Figure 5.6 Inference Example

In general terms, there are two approaches to dealing with the threat of disclosure by inference:

- **Inference detection during database design:** This approach removes an inference channel by altering the database structure or by changing the access control regime to prevent inference. Examples include removing data dependencies by splitting a table into multiple tables or using more fine-grained access control roles in an RBAC scheme. Techniques in this category often result in unnecessarily stricter access controls that reduce availability.
- **Inference detection at query time:** This approach seeks to eliminate an inference channel violation during a query or series of queries. If an inference channel is detected, the query is denied or altered.

For either of the preceding approaches, some inference detection algorithm is needed. This is a difficult problem and the subject of ongoing research. To give some appreciation of the difficulty, we present an example taken from [LUNT89]. Consider a database containing personnel information, including names, addresses, and salaries of employees. Individually, the name, address, and salary information is available to a subordinate role, such as Clerk, but the association of names and salaries is restricted to a superior role, such as Administrator. This is similar to the problem illustrated in Figure 5.6.

One solution to this problem is to construct three tables, which include the following information:

Employees (Emp#, Name, Address)
Salaries (S#, Salary)
Emp-Salary (Emp#, S#)

where each line consists of the table name followed by a list of column names for that table. In this case, each employee is assigned a unique employee number (Emp#) and a unique salary number (S#). The Employees table and the Salaries table are accessible to the Clerk role, but the Emp-Salary table is only available to the Administrator role. In this structure, the sensitive relationship between employees and salaries is protected from users assigned the Clerk role. Now suppose that we want to add a new attribute, employee start date, which is not sensitive. This could be added to the Salaries table as follows:

Employees (Emp#, Name, Address)
Salaries (S#, Salary, Start-Date)
Emp-Salary (Emp#, S#)

However, an employee's start date is an easily observable or discoverable attribute of an employee. Thus a user in the Clerk role should be able to infer (or partially infer) the employee's name. This would compromise the relationship between employee and salary. A straightforward way to remove the inference channel is to add the start-date column to the Employees table rather than to the Salaries table.

The first security problem indicated in this sample, that it was possible to infer the relationship between employee and salary, can be detected through analysis of the data structures and security constraints that are available to the DBMS. However, the second security problem, in which the start-date column was added to the Salaries table, cannot be detected using only the information stored in the database. In particular, the database does not indicate that the employee name can be inferred from the start date.

In the general case of a relational database, inference detection is a complex and difficult problem. For multilevel secure databases, discussed in Chapter 13, and statistical databases, discussed in the next section, progress has been made in devising specific inference detection techniques.

5.6 STATISTICAL DATABASES

A statistical database (SDB) is one that provides data of a statistical nature, such as counts and averages. The term *statistical database* is used in two contexts:

- **Pure statistical database:** This type of database only stores statistical data. An example is a census database. Typically, access control for a pure SDB is straightforward: certain users are authorized to access the entire database.
- **Ordinary database with statistical access:** This type of database contains individual entries; this is the type of database discussed so far in this chapter. The database