CHAPTER 5

DATABASE SECURITY

5.1 **Database Management Systems**

5.2 **Relational Databases**

Elements of a Relational Database System
Structured Query Language

5.3 **Database Access Control**

SQL-Based Access Definition
Cascading Authorizations
Role-Based Access Control

5.4 **Inference**

5.5 **Statistical Databases**

Inference from a Statistical Database
Query Restriction
Perturbation

5.6 **Database Encryption**

5.7 **Recommended Reading**

5.8 **Key Terms, Review Questions, and Problems**

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 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. Finally, we examine database encryption.

5.1 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. Developers 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 standalone security packages.

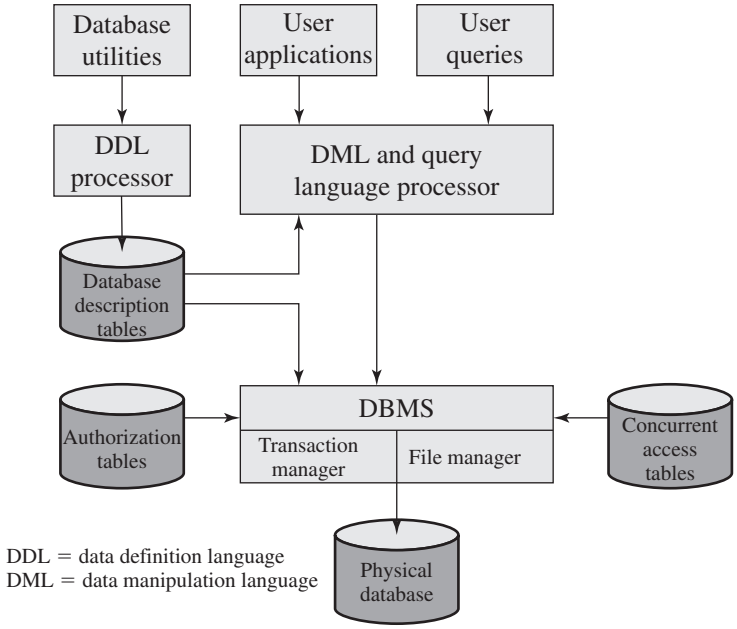

Figure 5.1 **DBMS Architecture**

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 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 designed specifically for, and integrated with, database systems are needed.

5.2 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 data. 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

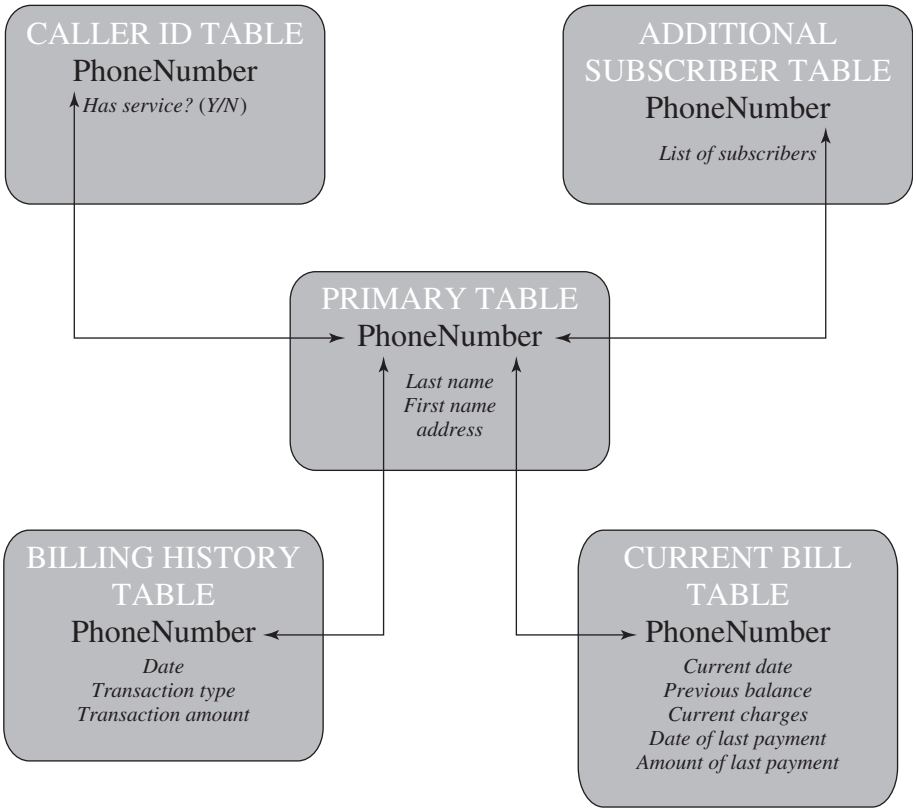

Figure 5.2 Example Relational Database Model. A relational database uses multiple tables related to one another by a designated key; in this case the key is the PhoneNumber field.

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** (Table 5.1). A **primary key** is 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 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.

Table 5.1 Basic Terminology for Relational Databases

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

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	Salary Code	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

Structure Query Language (SQL), originally developed by IBM in the mid-1970s, is a standardized language that can be used to define, manipulate, and query the 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.

5.3 DATABASE ACCESS CONTROL

Commercial 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 that 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 DBMSs provide discretionary or role-based access control. We defer a discussion of mandatory access control considerations to Chapter 10. 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 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 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.

[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.

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 phenomenon in 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 right to Ellen and therefore Jim is revoked when Bob revokes the access right to David. This is because at $t = 40$, when

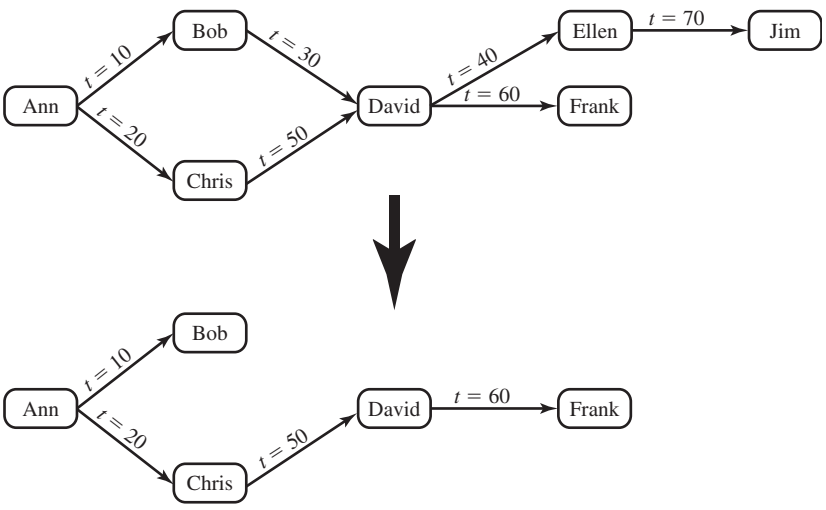

Figure 5.4 Bob Revokes Privilege from David

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 a 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 the 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 that 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 that 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 can not 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

Table 5.2 Fixed Roles in Microsoft SQL Server

Role	Permissions
Fixed Server Roles	
sysadmin	Can perform any activity in SQL Server and have completes 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 (DDL) statements
db_securityadmin	Can manage all permissions, object ownership, 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

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 a password. The role is activated when an application executes the appropriate code. A user that 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 application role with its own password to allow the particular user to obtain and modify any data only during specific hours. So you can realize more complex security management within the application logic.

5.4 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. Metadata refers to knowledge about correlations or dependencies among data items that

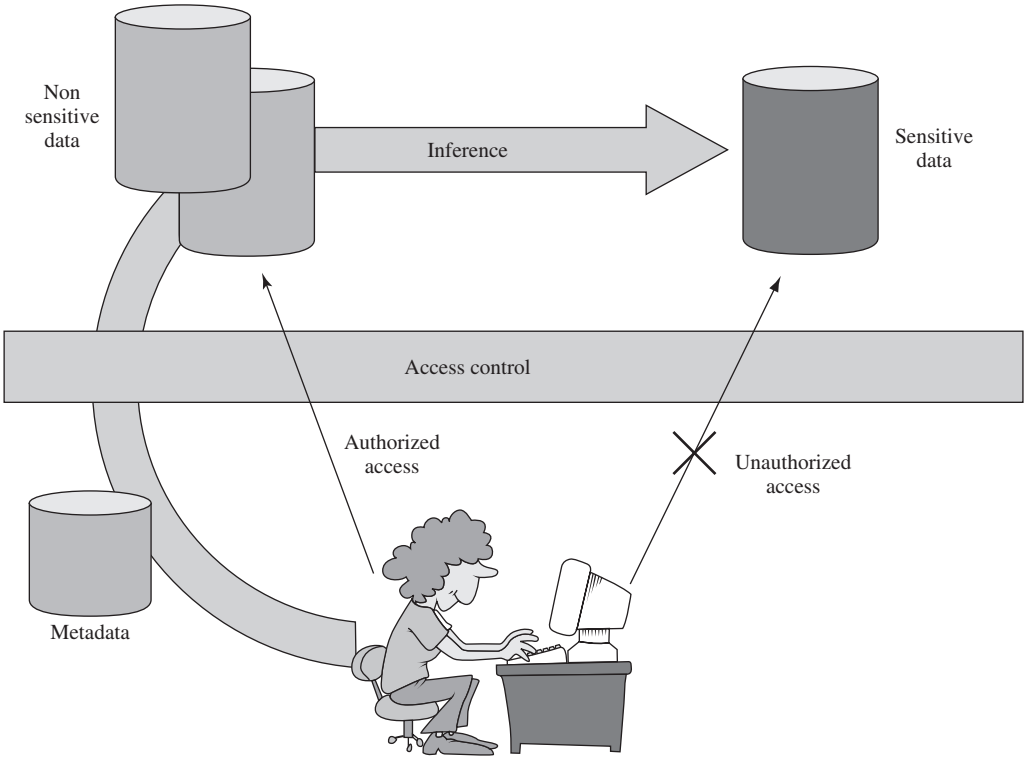

Figure 5.5 Indirect Information Access via Inference Channel
Source: Based on [FARK02].

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 from [SHIE98], shown in Figure 5.6, illustrates the inference problem. Figure 5.6a shows an Employee table with five columns. Figure 5.6b shows two views, defined in SQL as follows:

```
CREATE view V1 AS                    CREATE view V2 AS
SELECT Position, Salary              SELECT Name, Department
FROM Employee                        FROM Employee
WHERE Department = "strip"           WHERE Department = "strip"
```

Users of these views are not authorized to access the relationship between Name and Salary. 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 Name and Salary such that knowing Name and perhaps other information is sufficient to deduce Salary. However, suppose the two views are created with the access constraint that Name and Salary cannot be accessed together. A user who knows the structure of the Employee table and who knows that the view tables maintain the same row order as the Employee table is then able to merge the two

Name	Position	Salary ($)	Department	Dept. Manager
Andy	senior	43,000	strip	Cathy
Calvin	junior	35,000	strip	Cathy
Cathy	senior	48,000	strip	Cathy
Dennis	junior	38,000	panel	Herman
Herman	senior	55,000	panel	Herman
Ziggy	senior	67,000	panel	Herman

(a) Employee table

Position	Salary ($)
senior	43,000
junior	35,000
senior	48,000

Name	Department
Andy	Strip
Calvin	Strip
Cathy	Strip

(b) Two views

Name	Position	Salary ($)	Department
Andy	senior	43,000	strip
Calvin	junior	35,000	strip
Cathy	senior	48,000	strip

(c) Table derived from combining query answers

Figure 5.6 Inference Example

views to construct the table shown in Figure 5.6c. This violates the access control policy that the relationship of attributes Name and Salary must not be disclosed.

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 approaches 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 data 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 10, and statistical databases, discussed in the next section, progress has been made in devising specific techniques.

5.5 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 supports a population of nonstatistical users who are allowed access to selected portions of the database using DAC, RBAC, or MAC. In addition, the database supports a set of statistical users who are only permitted statistical queries. For these latter users, aggregate statistics based on the underlying raw data are generated in response to a user query, or may be precalculated and stored as part of the database.

For the purposes of this section, we are concerned only with the latter type of database and, for convenience, refer to this as an SDB. The access control objective for an SDB system is to provide users with the aggregate information without compromising the confidentiality of any individual entity represented in the database. The security problem is one of inference. The database administrator must prevent, or at least detect, the statistical user who attempts to gain individual information through one or a series of statistical queries.

For this discussion, we use the abstract model of a relational database table shown as Figure 5.7. There are N individuals, or entities, in the table and M attributes. Each attribute A_j has $|A_j|$ possible values, with x_{ij} denoting the value of attribute j for entity i. Table 5.3, taken from [DENN82], is an example that we use in the next few paragraphs. The example is a database containing 13 confidential records of students in a university that has 50 departments.

Statistics are derived from a database by means of a **characteristic formula**, C, which is a logical formula over the values of attributes. A characteristic formula uses the operators OR, AND, and NOT $(+, \cdot, \sim)$, written here in order of increasing priority. A characteristic formula specifies a subset of the records in the database. For example, the formula

$$(Sex = \text{Male}) \cdot ((Major = \text{CS}) + (Major = \text{EE}))$$

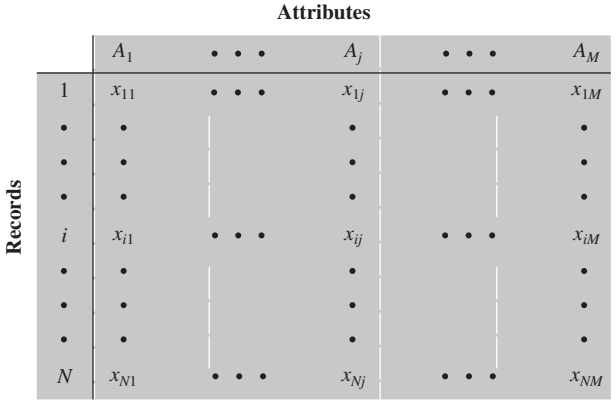

Figure 5.7 **Abstract Model of a Relational Database**

Table 5.3 Statistical Database Example

(a) Database with Statistical Access with $N = 13$ Students

Name	Sex	Major	Class	SAT	GP
Allen	Female	CS	1980	600	3.4
Baker	Female	EE	1980	520	2.5
Cook	Male	EE	1978	630	3.5
Davis	Female	CS	1978	800	4.0
Evans	Male	Bio	1979	500	2.2
Frank	Male	EE	1981	580	3.0
Good	Male	CS	1978	700	3.8
Hall	Female	Psy	1979	580	2.8
Iles	Male	CS	1981	600	3.2
Jones	Female	Bio	1979	750	3.8
Kline	Female	Psy	1981	500	2.5
Lane	Male	EE	1978	600	3.0
Moore	Male	CS	1979	650	3.5

(b) Attribute Values and Counts

Attribute A_J	Possible Values	$\lvert A_J \rvert$
Sex	Male, Female	2
Major	Bio, CS, EE, Psy, . . .	50
Class	1978, 1979, 1980, 1981	4
SAT	310, 320, 330, . . . 790, 800	50
GP	0.0, 0.1, 0.2, . . . 3.9, 4.0	41

Table 5.4 Some Queries of a Statistical Database

Name	Formula	Description
count(C)	$\lvert X(C)\rvert$	Number of records in the query set
sum(C, A_j)	$\sum_{i \in X(C)} x_{ij}$	Sum of the values of numerical attribute A_j over all the records in $X(C)$
rfreq(C)	$\frac{\textbf{count}\ (C)}{N}$	Fraction of all records that are in $X(C)$
avg(C, A_j)	$\frac{\textbf{sum}(C,\ A_j)}{\textbf{count}(C)}$	Mean value of numerical attribute A_j over all the records in $X(C)$
median (C, A_j)		The $\lceil \lvert X(C)\rvert/2 \rceil$ largest value of attribute over all the records in $X(C)$. Note that when the query set size is even, the median is the smaller of the two middle values. $\lceil x \rceil$ denotes the smallest integer greater than x.
max (C, A_j)	$\underset{i \in X(C)}{\text{Max}}\ (x_{ij})$	Maximum value of numerical attribute A_j over all the records in $X(C)$
min (C, A_j)	$\underset{i \in X(C)}{\text{Min}}\ (x_{ij})$	Minimum value of numerical attribute A_j over all the records in $X(C)$

Note: C = a characteristic formula, consisting of a logical formula over the values of attributes. X = query set of C, the set of records satisfying C.

specifies all male students majoring in either CS or EE. For numerical attributes, relational operators may be used. For example, $(GP > 3.7)$ specifies all students whose grade point average exceeds 3.7. For simplicity, we omit attribute names when they are clear from context. Thus, the preceding formula becomes Male · (CS + EE).

The **query set** of characteristic formula C, denoted as X(C), is the set of records matching that characteristic. For example, for C = Female · CS, X(C) consists of records 1 and 4, the records for Allen and Davis.

A statistical query is a query that produces a value calculated over a query set. Table 5.4 lists some simple statistics that can be derived from a query set. Examples: **count**(Female · CS) = 2; **sum**(Female · CS, SAT) = 1400.

Inference from a Statistical Database

A statistical user of an underlying database of individual records is restricted to obtaining only aggregate, or statistical, data from the database and is prohibited access to individual records. The inference problem in this context is that a user may infer confidential information about individual entities represented in the SDB. Such an inference is called a **compromise**. The compromise is positive if the user deduces the value of an attribute associated with an individual entity and is negative if the user deduces that a particular value of an attribute is not associated with an

individual entity. For example, the statistic **sum**(EE · Female, GP) = 2.5 compromises the database if the user knows that Baker is the only female EE student.

In some cases, a sequence of queries may reveal information. For example, suppose a questioner knows that Baker is a female EE student but does not know if she is the only one. Consider the following sequence of two queries:

count (EE · Female) = 1
sum (EE · Female, GP) = 2.5

This sequence reveals the sensitive information.

The preceding example shows how some knowledge of a single individual in the database can be combined with queries to reveal protected information. For a large database, there may be few or no opportunities to single out a specific record that has a unique set of characteristics, such as being the only female student in a department. Another angle of attack is available to a user aware of an incremental change to the database. For example, consider a personnel database in which the sum of salaries of employees may be queried. Suppose a questioner knows the following information:

Salary range for a new systems analyst with a BS degree is $[50K, 60K]

Salary range for a new systems analyst with a MS degree is $[60K, 70K]

Suppose two new systems analysts are added to the payroll and the change in the sum of the salaries is $130K. Then the questioner knows that both new employees have an MS degree.

In general terms, the inference problem for an SDB can be stated as follows. A characteristic function C defines a subset of records (rows) within the database. A query using C provides statistics on the selected subset. If the subset is small enough, perhaps even a single record, the questioner may be able to infer characteristics of a single individual or a small group. Even for larger subsets, the nature or structure of the data may be such that unauthorized information may be released.

Query Restriction

SDB implementers have developed two distinct approaches to protection of an SDB from inference attacks (Figure 5.8):

- **Query restriction:** Rejects a query that can lead to a compromise. The answers provided are accurate.
- **Perturbation:** Provides answers to all queries, but the answers are approximate.

We examine query restriction in this section and perturbation in the next. Query restriction techniques defend against inference by restricting statistical queries so that they do not reveal user confidential information. Restriction in this context simply means that some queries are denied.

Query Size Restriction The simplest form of query restriction is query size restriction. For a database of size N (number of rows, or records), a query $q(C)$ is permitted only if the number of records that match C satisfies

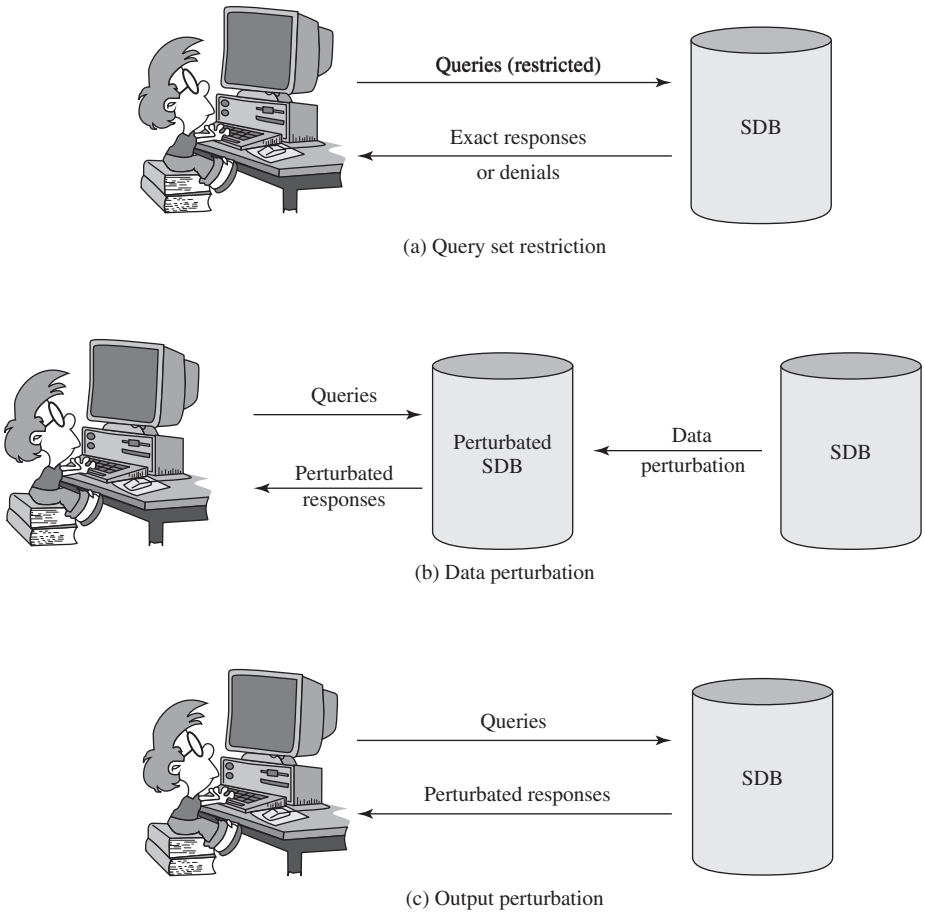

Figure 5.8 **Approaches to Statistical Database Security**
Source: Based on [ADAM89].

$$k \leq |X(C)| \leq N - k \tag{5.1}$$

where k is a fixed integer greater than 1. Thus, the user may not access any query set of less than k records. Note that the upper bound is also needed. Designate *All* as the set of all records in the database. If $q(C)$ is disallowed because $|X(C)| < k$, and there is no upper bound, then a user can compute $q(C) = q(All) - q(\sim C)$. The upper bound of $N - k$ guarantees that the user does not have access to statistics on query sets of less than k records. In practice, queries of the form $q(All)$ are allowed, enabling users to easily access statistics calculated on the entire database.

Query size restriction counters attacks based on very small query sets. For example, suppose a user knows that a certain individual I satisfies a given characteristic formula C (e.g., Allen is a female CS major). If the query **count**(C) returns 1, then the user has uniquely identified I. Then the user can test whether I has a particular characteristic D with the query **count**($C \cdot D$). Similarly, the user can learn the value of a numerical attribute A for I with the query **sum**(C, A).

Although query size restriction can prevent trivial attacks, it is vulnerable to more sophisticated attacks, such as the use of a tracker [DENN79]. In essence, the questioner divides his or her knowledge of an individual into parts, such that queries can be made based on the parts without violating the query size restriction. The combination of parts is called a *tracker*, because it can be used to track down characteristics of an individual. We can describe a tracker in general terms using the case from the preceding paragraph. The formula $C \cdot D$ corresponds to zero or one record, so that the query **count**$(C \cdot D)$ is not permitted. But suppose that the formula C can be decomposed into two parts $C = C1 \cdot C2$, such that the query sets for both $C1$ and $T = (C1 \cdot \sim C2)$ satisfy the query size restriction. Figure 5.9 illustrates this situation; in the figure, the size of the circle corresponds to the number of records in the query set. If it is not known if I is uniquely identified by C, the following formula can be used to determine if **count**$(C) = 1$:

$$\textbf{count}(C) = \textbf{count}(C1) - \textbf{count}(T) \tag{5.2}$$

That is, you count the number of records in $C1$ and then subtract the number of records that are in $C1$ but not in $C2$. The result is the number of records that are in both $C1$ and $C2$, which is equal to the number of records in C. By a similar reasoning, it can be shown that we can determine whether I has attribute D with

$$\textbf{count}(C \cdot D) = \textbf{count}(T + C1 \cdot D) - \textbf{count}(T) \tag{5.3}$$

For example, in Table 5.3, Evans is identified by $C = \text{Male} \cdot \text{Bio} \cdot 1979$. Let $k = 3$ in Equation 501. We can use $T = (C1 \cdot \sim C2) = \text{Male} \cdot \sim (\text{Bio} \cdot 1979)$. Both $C1$ and $C2$ satisfy the query size restriction. Using Equations (5.2) and (5.3), we determine that Evans is uniquely identified by C and whether his SAT score is at least 600:

$$\textbf{count}(\text{Male} \cdot \text{Bio} \cdot 1979) = \textbf{count}(\text{Male}) - \textbf{count}(\text{Male} \cdot \sim (\text{Bio} \cdot 1979))$$
$$= 7 - 6 = 1$$

$$\textbf{count}((\text{Male} \cdot \text{Bio} \cdot 1979) \cdot (\text{SAT} \geq 600)) =$$
$$\textbf{count}((\text{Male} \cdot \sim (\text{Bio} \cdot 1979) + (\text{Male} \cdot (\text{SAT} \geq 600)))$$
$$-\textbf{count}(\text{Male} \cdot \sim (\text{Bio} \cdot 1979)) = 6 - 6 = 0$$

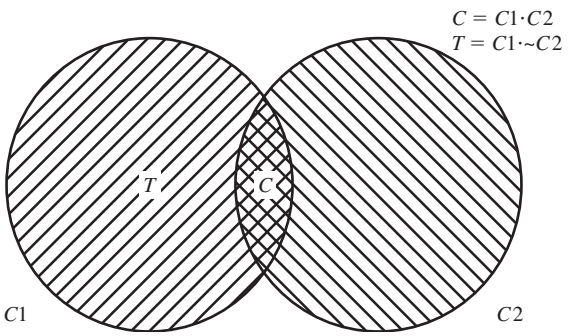

Figure 5.9 **Example of Tracker**

In a large database, the use of just a few queries will typically be inadequate to compromise the database. However, it can be shown that more sophisticated tracker attacks may succeed even against large databases in which the threshold k is set at a relatively high level [DENN79].

We have looked at query size restriction in some detail because it is easy to grasp both the mechanism and its vulnerabilities. A number of other query restriction approaches have been studied, all of which have their own vulnerabilities. However, several of these techniques in combination do reduce vulnerability.

Query Set Overlap Control A query size restriction is defeated by issuing queries in which there is considerable overlap in the query sets. For example, in one of the preceding examples the query sets Male and Male · ~ (Bio · 1979) overlap significantly, allowing an inference. To counter this, the query set overlap control provides the following limitation.

A query $q(C)$ is permitted only if the number of records that match C satisfies

$$\left|X(C) \cap X(D)\right| \leq r \tag{5.4}$$

for all $q(D)$ that have been answered for this user, and where r is a fixed integer greater than 0.

This technique has a number of problems, including the following [ADAM89]:

1. This control mechanism is ineffective for preventing the cooperation of several users to compromise the database.
2. Statistics for both a set and its subset (e.g., all patients and all patients undergoing a given treatment) cannot be released, thus limiting the usefulness of the database.
3. For each user, a user profile has to be kept up to date.

Partitioning Partitioning can be viewed as taking query set overlap control to its logical extreme, by not allowing overlapping queries at all. With partitioning, the records in the database are clustered into a number of mutually exclusive groups. The user may only query the statistical properties of each group as a whole. That is, the user may not select a subset of a group. Thus, with multiple queries, there must either be complete overlap (two different queries of all the records in a group) or zero overlap (two queries from different groups).

The rules for partitioning the database are as follows:

1. Each group G has $g = |G|$ records, where $g = 0$ or $g \geq n$, and g even, where n is a fixed integer parameter.
2. Records are added or deleted from G in pairs.
3. Query sets must include entire groups. A query set may be a single group or multiple groups.

A group of a single record is forbidden, for obvious reasons. The insertion or deletion of a single record enables a user to gain information about that record by taking before and after statistics. As an example, the database of Table 5.3a can be

Table 5.5 Partitioned Database

Sex	Class			
	1978	1979	1980	1981
Female	4	2	2	0
Male		2	0	2

partitioned as shown in Table 5.5. Because the database has an odd number of records, the record for Kline has been omitted. The database is partitioned by year and sex, except that for 1978, it is necessary to merge the Female and Male records to satisfy the design requirement.

Partitioning solves some security problems but has some drawbacks. The user's ability to extract useful statistics is reduced, and there is a design effort in constructing and maintaining the partitions.

Query Denial and Information Leakage A general problem with query restriction techniques is that the denial of a query may provide sufficient clues that an attacker can deduce underlying information. This is generally described by saying that query denial can leak information.

Here is a simple example from [KENT05]. Suppose that the underlying database consists of real-valued entries and that a query is denied only if it would enable the requestor to deduce a value. Now suppose the requester poses the query **sum**(x_1, x_2, x_3) and the response is 15. Then the requester queries **max**(x_1, x_2, x_3) and the query is denied. What can the requester deduce from this? We know that the **max**(x_1, x_2, x_3) cannot be less than 5 because then the sum would be less than 15. But if **max**(x_1, x_2, x_3) > 5, the query would not be denied because the answer would not reveal a specific value. Therefore, it must be the case that **max**(x_1, x_2, x_3) = 5, which enables the requester to deduce that $x_1 = x_2 = x_3 = 5$.

[KENT05] describes an approach to counter this threat, referred to as *simulatable auditing*. The details of this approach are beyond the scope of this chapter. In essence, the system monitors all of the queries from a given source and decides on the basis of the queries so far posed whether to deny a new query. The decision is based solely on the history of queries and answers and the specific new query. In deciding whether to deny the query, the system does not consider the actual values of database elements that will contribute to generating the answer and therefore does not consider the actual value of the answer. Thus, the system makes the denial decision solely on the basis of information that is already available to the requester (the history of prior requests). Hence the decision to deny a query cannot leak any information. For this approach, the system determines whether any collection of database values might lead to information leakage and denies the query if leakage is possible. In practice, a number of queries will be denied even if leakage is not possible. In the example of the preceding paragraph, this strategy would deny the **max** query whether or not the three underlying values were equal. Thus, this approach is more conservative in that it issues more denials than an approach that considers the actual values in the database.

Perturbation

Query restriction techniques can be costly and are difficult to implement in such a way as to completely thwart inference attacks, especially if a user has supplementary knowledge. For larger databases, a simpler and more effective technique is to, in effect, add noise to the statistics generated from the original data. This can be done in one of two ways (Figure 5.8): The data in the SDB can be modified (perturbed) so as to produce statistics that cannot be used to infer values for individual records; we refer to this as **data perturbation**. Alternatively, when a statistical query is made, the system can generate statistics that are modified from those that the original database would provide, again thwarting attempts to gain knowledge of individual records; this is referred to as **output perturbation**.

Regardless of the specific perturbation technique, the designer must attempt to produce statistics that accurately reflect the underlying database. Because of the perturbation, there will be differences between perturbed results and ordinary results from the database. However, the goal is to minimize the differences and to provide users with consistent results.

As with query restriction, there are a number of perturbation techniques. In this section, we highlight a few of these.

Data Perturbation Techniques We look at two techniques that consider the SDB to be a sample from a given population that has a given population distribution. Two methods fit into this category. The first transforms the database by substituting values that conform to the same assumed underlying probability distribution. The second method is, in effect, to generate statistics from the assumed underlying probability distribution.

The first method is referred to as **data swapping**. In this method, attribute values are exchanged (swapped) between records in sufficient quantity so that nothing can be deduced from the disclosure of individual records. The swapping is done in such a way that the accuracy of at least low-order statistics is preserved. Table 5.6, from [DENN82], shows a simple example, transforming the database D into the database D′. The transformed database D has the same statistics as D for statistics derived from one or two attributes. However, three-attribute statistics are not preserved. For example, **count**(Female · CS · 3.0) has the value 1 in D but the value 0 in D′.

Table 5.6 Example of Data Swapping

	D			D′		
Record	**Sex**	**Major**	**GP**	**Sex**	**Major**	**GP**
1	Female	Bio	4.0	Male	Bio	4.0
2	Female	CS	3.0	Male	CS	3.0
3	Female	EE	3.0	Male	EE	3.0
4	Female	Psy	4.0	Male	Psy	4.0
5	Male	Bio	3.0	Female	Bio	3.0
6	Male	CS	4.0	Female	CS	4.0
7	Male	EE	4.0	Female	EE	4.0
8	Male	Psy	3.0	Female	Psy	3.0

Another method is to generate a modified database using the estimated underlying probability distribution of attribute values. The following steps are used:

1. For each confidential or sensitive attribute, determine the probability distribution function that best matches the data and estimate the parameters of the distribution function.
2. Generate a sample series of data from the estimated density function for each sensitive attribute.
3. Substitute the generated data of the confidential attribute for the original data in the same rank order. That is, the smallest value of the new sample should replace the smallest value in the original data, and so on.

Output Perturbation Techniques A simple output perturbation technique is known as **random-sample query**. This technique is suitable for large databases and is similar to a technique employed by the U.S. Census Bureau. The technique works as follows:

1. A user issues a query $q(C)$ that is to return a statistical value. The query set so defined is $X(C)$.
2. The system replaces $X(C)$ with a sampled query set, which is a properly selected subset of $X(C)$.
3. The system calculates the requested statistic on the sampled query set and returns the value.

Other approaches to output perturbation involve calculating the statistic on the requested query set and then adjusting the answer up or down by a given amount in some systematic or randomized fashion. All of these techniques are designed to thwart tracker attacks and other attacks that can be made against query restriction techniques.

With all of the perturbation techniques, there is a potential loss of accuracy as well as the potential for a systematic bias in the results.

Limitations of Perturbation Techniques The main challenge in the use of perturbation techniques is to determine the average size of the error to be used. If there is too little error, a user can infer close approximations to protected values. If the error is, on average, too great, the resulting statistics may be unusable.

For a small database, it is difficult to add sufficient perturbation to hide data without badly distorting the results. Fortunately, as the size of the database grows, the effectiveness of perturbation techniques increases. This is a complex topic, beyond the scope of this chapter. Examples of recent work include [DWOR06], [EVFI03], and [DINU03].

The last-mentioned reference reported the following result. Assume the size of the database, in terms of the number of data items or records, is n. If the number of queries from a given source is linear to the size of the database (i.e., on the order of n), then a substantial amount of noise must be added to the system in terms of perturbation, to preserve confidentiality. Specifically, suppose the perturbation is imposed on the system by adding a random amount of perturbation $\leq x$. Then, if the query magnitude is linear, the perturbation must be at least of order $\sqrt{n}$. This amount

of noise may be sufficient to make the database effectively unusable. However, if the number of queries is sublinear (e.g., of order $\sqrt{n}$), then much less noise must be added to the system to maintain privacy. For a large database, limiting queries to a sublinear number may be reasonable.

5.6 DATABASE ENCRYPTION

The database is typically the most valuable information resource for any organization and is therefore protected by multiple layers of security, including firewalls, authentication mechanisms, general access control systems, and database access control systems. In addition, for particularly sensitive data, database encryption is warranted and often implemented. Encryption becomes the last line of defense in database security.

There are two disadvantages to database encryption:

- **Key management:** Authorized users must have access to the decryption key for the data for which they have access. Because a database is typically accessible to a wide range of users and a number of applications, providing secure keys to selected parts of the database to authorized users and applications is a complex task.
- **Inflexibility:** When part or all of the database is encrypted, it becomes more difficult to perform record searching.

Encryption can be applied to the entire database, at the record level (encrypt selected records), at the attribute level (encrypt selected columns), or at the level of the individual field.

A number of approaches have been taken to database encryption. In this section, we look at a representative approach for a multiuser database.

A DBMS is a complex collection of hardware and software. It requires a large storage capacity and requires skilled personnel to perform maintenance, disaster protection, update, and security. For many small and medium-sized organizations, an attractive solution is to outsource the DBMS and the database to a service provider. The service provider maintains the database off site and can provide high availability, disaster prevention, and efficient access and update. The main concern with such a solution is the confidentiality of the data.

A straightforward solution to the security problem in this context is to encrypt the entire database and not provide the encryption/decryption keys to the service provider. This solution by itself is inflexible. The user has little ability to access individual data items based on searches or indexing on key parameters, but rather would have to download entire tables from the database, decrypt the tables, and work with the results. To provide more flexibility, it must be possible to work with the database in its encrypted form.

An example of such an approach, depicted in Figure 5.10, is reported in [DAMI05] and [DAMI03]. A similar approach is described in [HACI02]. Four entities are involved:

- **Data owner:** An organization that produces data to be made available for controlled release, either within the organization or to external users.

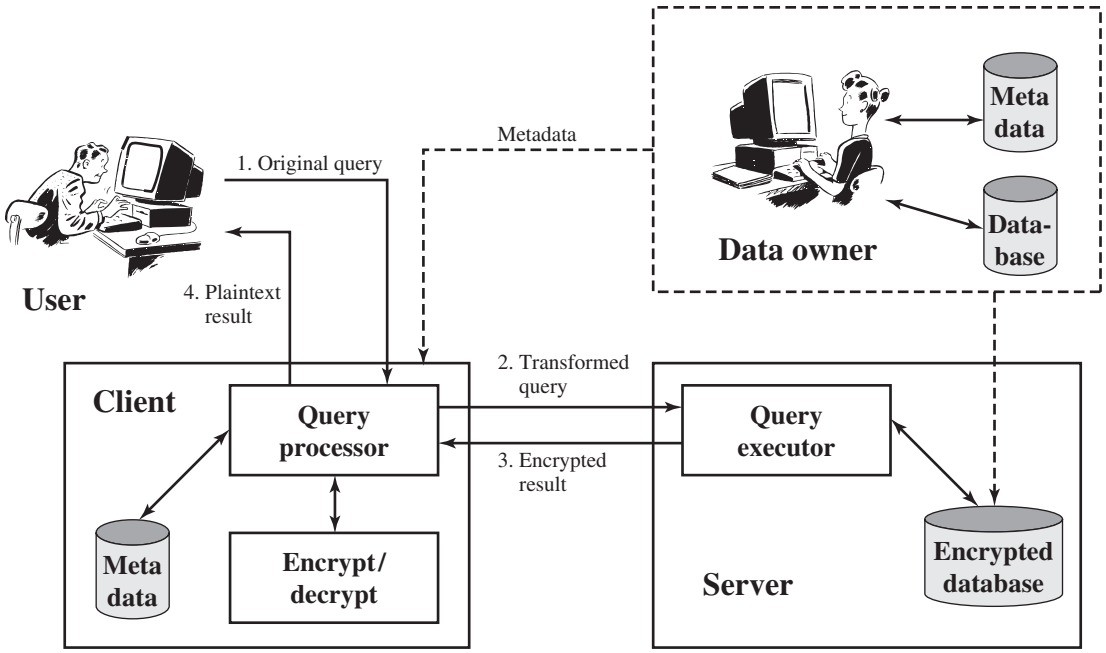

Figure 5.10 **A Database Encryption Scheme**

- **User:** Human entity that presents requests (queries) to the system. The user could be an employee of the organization who is granted access to the database via the server, or a user external to the organization who, after authentication, is granted access.
- **Client:** Front end that transforms user queries into queries on the encrypted data stored on the server.
- **Server:** An organization that receives the encrypted data from a data owner and makes them available for distribution to clients. The server could in fact be owned by the data owner but, more typically, is a facility owned and maintained by an external provider.

Let us first examine the simplest possible arrangement based on this scenario. Suppose that each individual item in the database is encrypted separately, all using the same encryption key. The encrypted database is stored at the server, but the server does not have the key, so that the data are secure at the server. Even if someone were able to hack into the server's system, all he or she would have access to is encrypted data. The client system does have a copy of the encryption key. A user at the client can retrieve a record from the database with the following sequence:

1. The user issues an SQL query for fields from one or more records with a specific value of the primary key.
2. The query processor at the client encrypts the primary key, modifies the SQL query accordingly, and transmits the query to the server.

3. The server processes the query using the encrypted value of the primary key and returns the appropriate record or records.
4. The query processor decrypts the data and returns the results.

For example, consider this query, which was introduced in Section 5.1, on the database of Figure 5.3a:

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

Assume that the encryption key k is used and that the encrypted value of the department id 15 is $E(k, 15) = 1000110111001110$. Then the query processor at the client could transform the preceding query into

```
SELECT Ename, Eid, Ephone
      FROM Employee
      WHERE Did = 1000110111001110
```

This method is certainly straightforward but, as was mentioned, lacks flexibility. For example, suppose the Employee table contains a salary attribute and the user wishes to retrieve all records for salaries less than $70K. There is no obvious way to do this, because the attribute value for salary in each record is encrypted. The set of encrypted values does not preserve the ordering of values in the original attribute.

To provide more flexibility, the following approach is taken. Each record (row) of a table in the database is encrypted as a block. Referring to the abstract model of a relational database in Figure 5.7, each row R_i is treated as a contiguous block $B_i = (x_{i1} \| x_{i2} \| \ldots \| x_{iM})$. Thus, each attribute value in R_i, regardless of whether it is text or numeric, is treated as a sequence of bits, and all of the attribute values for that row are concatenated together to form a single binary block. The entire row is encrypted, expressed as $E(k, B_i) = E(k, (x_{i1} \| x_{i2} \| \ldots \| x_{iM}))$. To assist in data retrieval, attribute indexes are associated with each table. For some or all of the attributes an index value is created. For each row R_i of the unencrypted database, the mapping is as follows (Figure 5.11):

$$(x_{i1}, x_{i2}, \ldots, x_{iM}) \rightarrow [E(k, B_i), I_{i1}, I_{i2}, \ldots, I_{iM}]$$

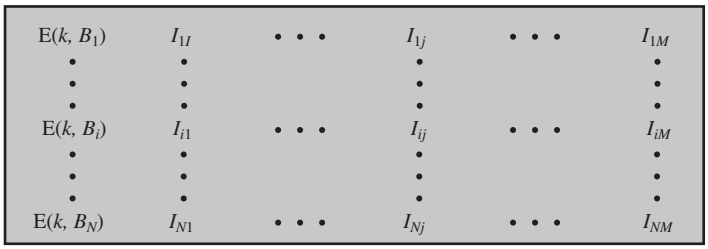

Figure 5.11 Encryption Scheme for Database of Figure 5.7

For each row in the original database, there is one row in the encrypted database. The index values are provided to assist in data retrieval. We can proceed as follows. For any attribute, the range of attribute values is divided into a set of non-overlapping partitions that encompass all possible values, and an index value is assigned to each partition.

Table 5.7 provides an example of this mapping. Suppose that employee ID (*eid*) values lie in the range [1, 1000]. We can divide these values into five partitions—[1, 200], [201, 400], [401, 600], [601, 800], and [801, 1000]—and then assign index values 1, 2, 3, 4, and 5, respectively. For a text field, we can derive an index from the first letter of the attribute value. For the attribute *ename*, let us assign index 1 to values starting with A or B, index 2 to values starting with C or D, and so on. Similar partitioning schemes can be used for each of the attributes. Table 5.7b shows the resulting table. The values in the first column represent the encrypted values for each row. The actual values depend on the encryption algorithm and the encryption key. The remaining columns show index values for the corresponding attribute values. The mapping functions between attribute values and index values constitute metadata that are stored at the client and data owner locations but not at the server.

This arrangement provides for more efficient data retrieval. Suppose, for example, a user requests records for all employees with *eid* <300. The query processor requests all records with I(*eid*) $\leq$ 2. These are returned by the server. The query processor decrypts all rows returned, discards those that do not match the original query, and returns the requested unencrypted data to the user.

The indexing scheme just described does provide a certain amount of information to an attacker, namely a rough relative ordering of rows by a given attribute. To obscure such information, the ordering of indexes can be randomized. For example, the *eid* values could be partitioned by mapping [1, 200], [201, 400], [401, 600], [601, 800], and [801, 1000] into 2, 3, 5, 1, and 4, respectively. Because the metadata are not stored at the server, an attacker could not gain this information from the server.

Other features may be added to this scheme. To increase the efficiency of accessing records by means of the primary key, the system could use the encrypted

Table 5.7 Encrypted Database Example

(a) Employee Table

eid	ename	salary	addr	did
23	Tom	70K	Maple	45
860	Mary	60K	Main	83
320	John	50K	River	50
875	Jerry	55K	Hopewell	92

(b) Encrypted Employee Table with Indexes

E(*k*, *B*)	I(ied)	I(ename)	I(salary)	I(addr)	I(did)
1100110011001011 . . .	1	10	3	7	4
0111000111001010 . . .	5	7	2	7	8
1100010010001101 . . .	2	5	1	9	5
0011010011111101 . . .	5	5	2	4	9

value of the primary key attribute values, or a hash value. In either case, the row corresponding to the primary key value could be retrieved individually. Different portions of the database could be encrypted with different keys, so that users would only have access to that portion of the database for which they had the decryption key. This latter scheme could be incorporated into a role-based access control system.

5.7 RECOMMENDED READING

[BERT05] is an excellent survey of database security. Two surveys of access control for database systems are [BERT95] and [LUNT90]. [VIEI05] analyzes ways to characterize and assess security mechanisms in database systems. [DISA95] is a lengthy discussion of database security topics, focusing on the features available in commercial DBMSs.

[FARK02] is a brief overview of the inference problem. [THUR05] provides a thorough treatment. [ADAM89] provides a useful overview of statistical database security. [JONG83] illustrates the extent of the vulnerability of statistical databases to a simple series of queries.

For a brief but useful overview of databases, see [LEYT01]. [SHAS04] is an instructive discussion on the use of database systems by application developers. The concepts on which relational databases are based were introduced in a classic paper by Codd [CODD70]. An early survey paper on relational databases is [KIM79].

ADAM89 Adam, N., and Wortmann, J. "Security-Control Methods for Statistical Databases: A Comparative Study." *ACM Computing Surveys*, December 1989.

BERT95 Bertino, E.; Japonica, S.; and Samurai, P. "Database Security: Research and Practice." *Information Systems*, Vol. 20, No. 7, 1995.

BERT05 Bertino, E., and Sandhog, R. "Database Security—Concepts, Approaches, and Challenges." *IEEE Transactions on Dependable and Secure Computing*, January–March, 2005.

CODD70 Codd, E. "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM*, June 1970.

DISA95 Defense Information Systems Agency. *Database Security Technical Implementation Guide.* Department of Defense, 30 November 2005. csrc.nist.gov/pcig/STIGs/database-stig-v7r2.pdf

FARK02 Farkas, C., and Jajodia, S. "The Inference Problem: A Survey." *ACM SIGKDD Explorations*, Vol. 4, No. 2, 2002.

JONG83 Jonge, W. "Compromising Statistical Database Responding to Queries About Means." *ACM Transactions on Database Systems*, March 1983.

KIM79 Kim, W. "Relational Database Systems." *Computing Surveys*, September 1979,

LEYT01 Leyton, R. "A Quick Introduction to Database Systems." *;login*, December 2001.

LUNT90 Lunt, T., and Fernandez, E. "Database Security." *ACM SIGMOD Record*, December 1990.

SHAS04 Shasha, D., and Bonnet, P. "Database Systems: When to Use Them and How to Use Them Well." *Dr. Dobb's Journal*, December 2004.

THUR05 Thuraisingham, B. *Database and Applications Security.* New York: Auerbach, 2005.

VIEI05 Vieira, M, and Madeira, H. " Towards a Security Benchmark for Database Management Systems." *Proceedings of the 2005 International Conference on Dependable Systems and Networks*, 2005.

5.8 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

attribute	foreign key	query size restriction
cascading authorizations	inference	relation
characteristic formula	inference channel	relational database
compromise	output perturbation	relational database management system (RDBMS)
data perturbation	partitioning	SQL
data swapping	perturbation	statistical database
database	primary key	tuple
database access control	query language	view
database encryption	query restriction	
database management system (DBMS)	query set	
	query set overlap control	

Review Questions

5.1 Define the terms *database*, *database management system*, and *query language.*

5.2 What is a relational database and what are its principal ingredients?

5.3 How many primary keys and how many foreign keys may a table have in a relational database?

5.4 List and briefly describe some administrative policies that can be used with a RDBMS.

5.5 Explain the concept of cascading authorizations.

5.6 Explain the nature of the inference threat to a RDBMS.

5.7 What are the two main types of statistical databases?

5.8 List and briefly describe two approaches to inference prevention for a statistical database.

5.9 What are the disadvantages to database encryption?

Problems

5.1 Consider a simplified university database that includes information on courses (name, number, day, time, room number, max enrollment) and on faculty teaching courses and students attending courses. Suggest a relational database for efficiently managing this information.

5.2 The following table below provides information on members of a mountain climbing club.

Climber-ID	Name	Skill-Level	Age
123	Edmund	Experienced	80
214	Arnold	Beginner	25
313	Bridget	Experienced	33
212	James	Medium	27

The primary key is *Climber-ID*. Explain whether or not each of the following rows can be added to the table.

Climber-ID	Name	Skill-Level	Age
214	Abbot	Medium	40
	John	Experienced	19
15	Jeff	Medium	42

5.3 The following table shows a list of pets and their owners that is used by a veterinarian service.

P_Name	Type	Breed	DOB	Owner	O_Phone	O_Email
Kino	Dog	Std. Poodle	3/27/97	M. Downs	5551236	md@abc.com
Teddy	Cat	Chartreaux	4/2/98	M. Downs	1232343	md@abc.com
Filo	Dog	Std. Poodle	2/24/02	R. James	2343454	rj@abc.com
AJ	Dog	Collie Mix	11/12/95	Liz Frier	3456567	liz@abc.com
Cedro	Cat	Unknown	12/10/96	R. James	7865432	rj@abc.com
Woolley	Cat	Unknown	10/2/00	M. Trent	9870678	mt@abc.com
Buster	Dog	Collie	4/4/01	Ronny	4565433	ron@abc.com

a. Describe four problems that are likely to occur when using this table.
b. Break the table into two tables in a way that fixes the four problems.

5.4 We wish to create a Student Table containing the students ID number, name, and telephone number. Write an SQL statement to accomplish this.

5.5 Assume that A, B, and C grant certain privileges on the Employee table to X, who in turn grants them to Y, as shown in the following table, with the numerical entries indicating the time of granting:

UserID	Table	Grantor	READ	INSERT	DELETE
X	Employee	A	15	15	—
X	Employee	B	20	—	20
Y	Employee	X	25	25	25
X	Employee	C	30	—	30

At time $t = 35$, B issues the command REVOKE ALL RIGHTS ON Employee FROM X. Which access rights, if any, of Y must be revoked, using the conventions defined in Section 5.2?

5.6 Figure 5.12 shows a sequence of grant operations for a specific access right on a table. Assume that at $t = 70$, B revokes the access right from C. Using the conventions defined in Section 5.2, show the resulting diagram of access right dependencies.

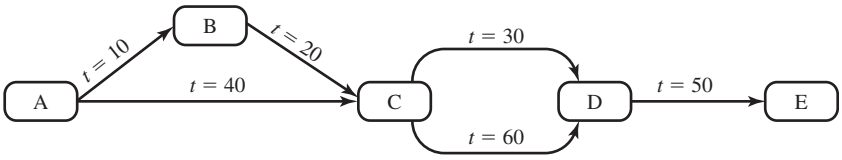

Figure 5.12 **Cascaded Privileges**

5.7 Figure 5.13 shows an alternative convention for handling revocations of the type illustrated in Figure 5.4.

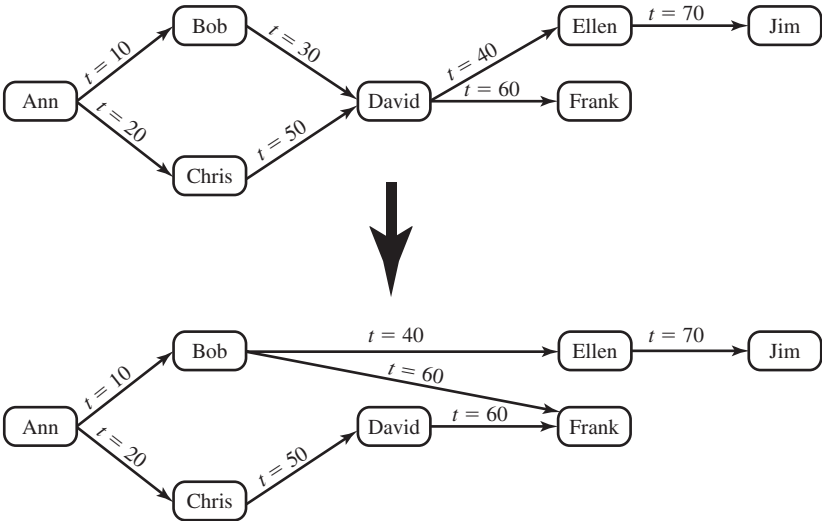

Figure 5.13 Bob Revokes Privilege from David, Second Version

a. Describe an algorithm for revocation that fits this figure.
b. Compare the relative advantages and disadvantages of this method to the original method, illustrated in Figure 5.4.

5.8 Consider the parts department of a plumbing contractor. The department maintains an inventory database that includes parts information (part number, description, color, size, number in stock, etc.) and information on vendors from whom parts are obtained (name, address, pending purchase orders, closed purchase orders, etc.). In an RBAC system, supposed that roles are defined for accounts payable clerk, an installation foreman, and a receiving clerk. For each role, indicate which items should be accessible for read-only and read-write access.

5.9 Imagine that you are the database administrator for a military transportation system. You have a table named cargo in your database that contains information on the various cargo holds available on each outbound airplane. Each row in the table represents a single shipment and lists the contents of that shipment and the flight identification number. Only one shipment per hold is allowed. The flight identification number may be cross-referenced with other tables to determine the origin, destination, flight time, and similar data. The cargo table appears as follows:

Flight ID	Cargo Hold	Contents	Classification
1254	A	Boots	Unclassified
1254	B	Guns	Unclassified
1254	C	Atomic bomb	Top Secret
1254	D	Butter	Unclassified

Suppose that two roles are defined: Role 1 has full access rights to the cargo table. Role 2 has full access rights only to rows of the table in which the Classification field has the value Unclassified. Describe a scenario in which a user assigned to role 2 uses one or more queries to determine that there is a classified shipment on board the aircraft.

5.10 Users hulkhogan and undertaker do not have the SELECT access right to the Inventory Table and the Item table. These tables were created by and are owned by user bruno-s. Write the SQL commands that would enable bruno-s to grant SELECT access to these tables to hulkhogan and undertaker,

5.11 In the example of Section 5.4 involving the addition of a Start-Date column to a set of tables defining employee information, it was stated that a straightforward way to remove the inference channel is to add the Start-Date column to the Employees table. Suggest another way.

5.12 The query size restriction for a statistical database is defined in Section 5.4 as $k \leq |\mathrm{X}(C)| \leq N - k$. What is the upper bound on the value of k? Explain.

5.13 In Section 5.4 it was mentioned that for the query size restriction, queries of the form $q(All)$ are allowed. If such queries are not allowed, how can the user access statistics calculated on the entire database?

5.14 Suppose a user knows that Evans is represented in the database of Table 5.3 and that Evans is a male biology student in the class of 1979.

a. What query can be used to test whether Evans is the only such student?

b. What query can be used to determine Evans SAT score?

5.15 Draw a diagram similar to that of Figure 5.9 that illustrates the relationship $\mathbf{count}(C \cdot D) = \mathbf{count}(T + C1 \cdot D) - \mathbf{count}(T)$.

5.16 **a.** Explain why the following statement is true. If $\mathbf{count}(C) = 1$ for individual I, the value of a numerical attribute A for I can be computed from $\mathbf{sum}(C, A) = \mathbf{sum}(C1, A) - \mathbf{sum}(T, A)$.

b. Continuing the query restriction example from Section 5.5, show how to calculate the GP value for Evans.

5.17 This question relates to the statistical database of Table 5.8.

a. Assume no query size restriction and that a questioner knows that Dodd is a female CS professor. Show a sequence of two queries that the questioner could use to determine Dodd's salary.

b. Suppose there is a lower query size limit of 2, but no upper limit. Show a sequence of queries that could be used to determine Dodd's salary.

c. Suppose that there is a lower and upper query size limit that satisfies Equation (5.1) with $k = 2$. Show a sequence of queries that could be used to determine Dodd's salary.

Table 5.8 Statistical Database Problem

Name	Sex	Department	Position	Salary ($K)
Adams	Male	CS	Prof	80
Baker	Male	Math	Prof	60
Cook	Female	Math	Prof	100
Dodd	Female	CS	Prof	60
Engel	Male	Stat	Prof	72
Flynn	Female	Stat	Prof	88
Grady	Male	CS	Admin	40
Hayes	Male	Math	Prof	72
Irons	Female	CS	Stu	12
Jones	Male	Stat	Adm	80
Knapp	Female	Math	Prof	100
Lord	Male	CS	Stu	12

5.18 Consider a database table that includes a salary attribute. Suppose the three queries **sum**, **count**, and **max** (in that order) are made on the salary attribute, all conditioned on the same predicate involving other attributes. That is, a specific subset of records is selected and the three queries are performed on that subset. Suppose that the first two queries are answered and the third query is denied. Is any information leaked?

5.19 For Table 5.7, deduce the partitioning scheme used for attributes *salary*, *addr*, and *did*.