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, and 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.

◆ 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.

◆ Define and explain SQL injection attacks.

◆ Compare and contrast different approaches to database access control.

◆ Explain how inference poses a security threat in database systems.

◆ Discuss the use of encryption in a database system.

◆ Present an overview of cloud computing concepts.

◆ Understand the unique security issues related to cloud computing. 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 Hypertext

Transfer Protocol (HTTP) 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.

- **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. A query language provides a uniform interface to the database for users and applications. Following Figure 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.

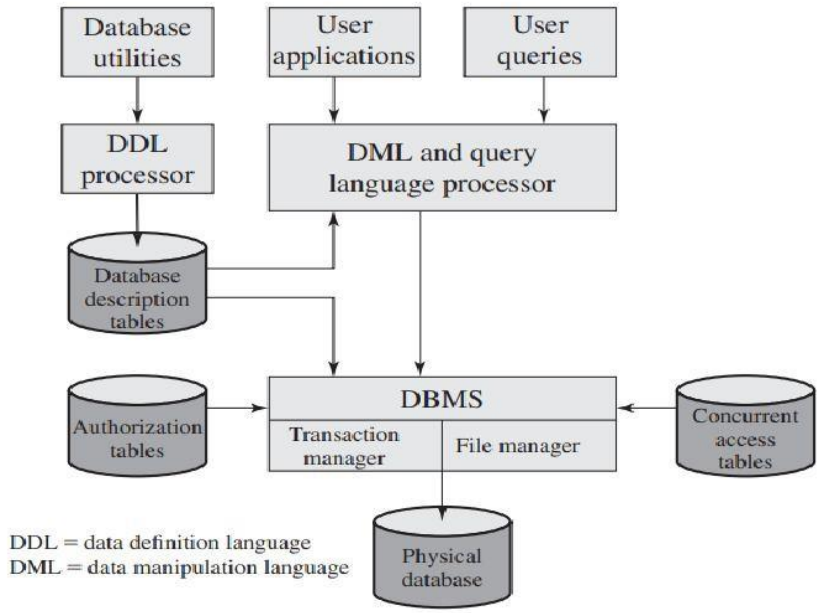


Figure 5.1 **DBMS Architecture**

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.

- **Relational Databases-** 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. . 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. An abstract model of a relational database table is shown as Figure 5.3. There are N individuals, or entities, in the table and M attributes. Each attribute Aj has |Aj| possible values, with xij denoting the value of attribute j for entity i. 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

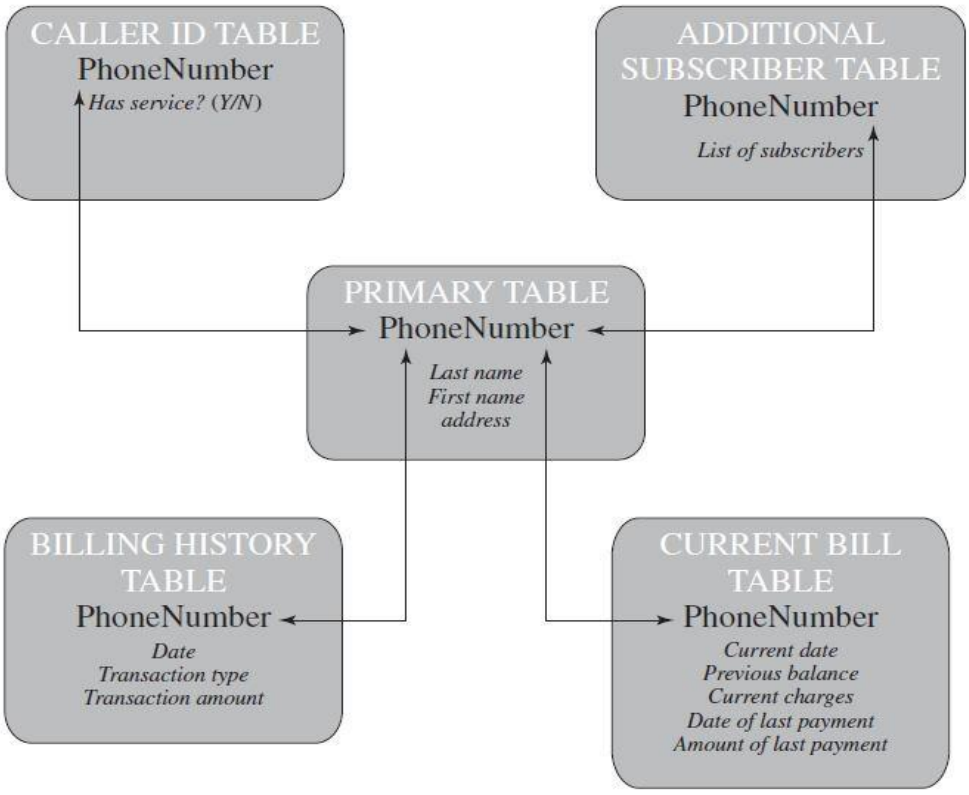


Figure 5.2 **Example Relational Database Model.** A relational database uses multiple tables related to one another by a designated key; in this case the key is the Phone-Number field.

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. An abstract model of a relational database table is shown as Figure 5.3. There are N individuals, or entities, in the table and M attributes. Each attribute Aj has |Aj| possible values, with xij denoting the value of attribute j for entity i.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. In following figure, 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 keyand provides the relationship between the Employee table and the Department table.

Structured Query Language

Structured Query Language (SQL) 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 above Figure 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) )
```

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

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

```
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.

```
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.

- **DATABASE ACCESS CONTROL**

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.

Following are some access rights SQL based access rights 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:

GRANT {privileges (access permissions granted) | role (role based access rights supported)}
[ON table (table name)]
TO {user | role | PUBLIC}
[IDENTIFIED BY password (Authentication)]
[WITH GRANT OPTION]

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.
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

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 completeControl. 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 administrativeResponsibilities. Others, such as db_datareader. Following table shows some role based access permissions.

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

- **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 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 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 5.8, illustrates the inference problem. Figure 5.8a shows an Inventory table with four columns. Figure 5.8b shows two views, defined in SQL as follows:

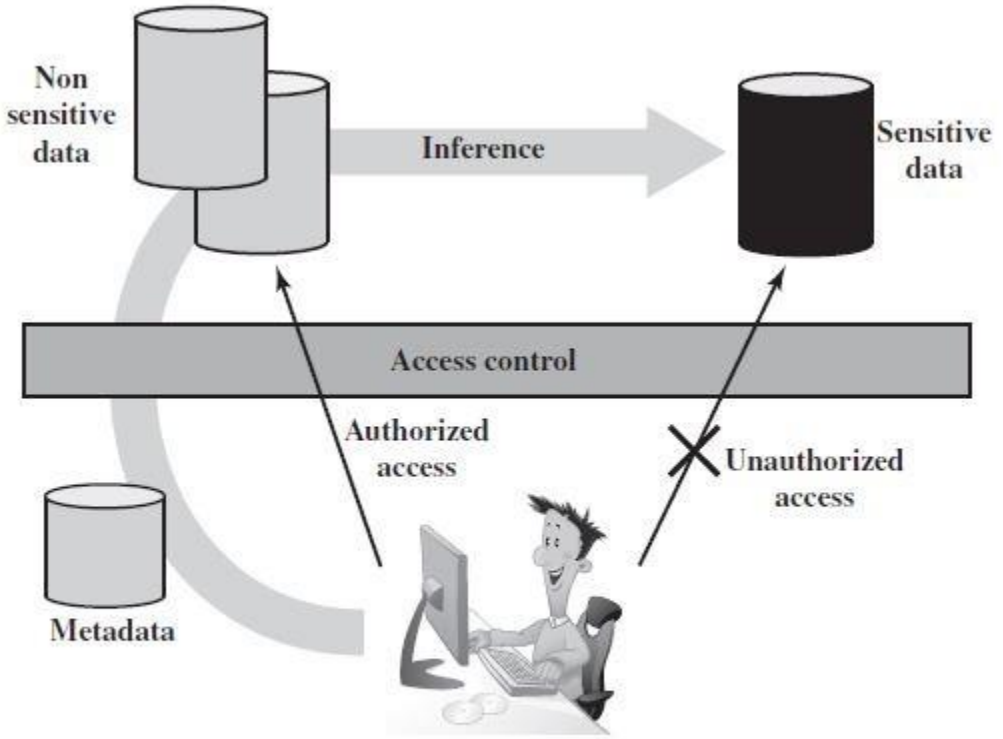


Figure 5.7 **Indirect Information Access via Inference Channel**

```
CREATE view V1 AS
    SELECT Availability, Cost
    FROM Inventory
    WHERE Department = ”hardware”
```

```
CREATE view V2 AS
    SELECT Item, Department
    FROM Inventory
    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. 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 5.8c. This violates the access control policy that the relationship of attributes Item and Cost must not be disclosed

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.

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 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#)

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 5.8 **Inference Example**

DATABASE ENCRPTION

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.

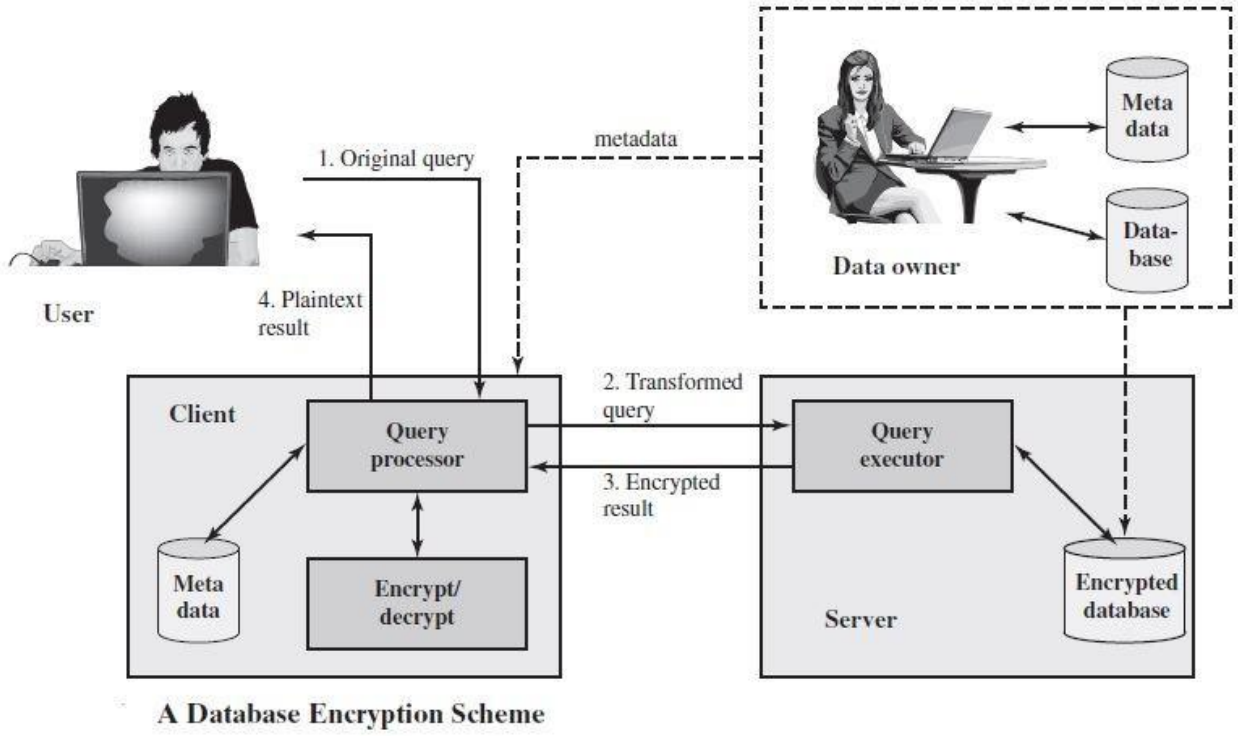
There are two disadvantages to database encryption:

• **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 Database Encryption Scheme

• Data owner: An organization that produces data to be made available for controlled release, either within the organization or to external users.
• 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 thedatabase of Figure 5.4a:

```
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 do not preserve the ordering of values in the original attribute.