Information Security
Fall 2022

WEEK #9

LECTURE # 22, 23 AND 24

DR. AQSA ASLAM

CHAPTER 5

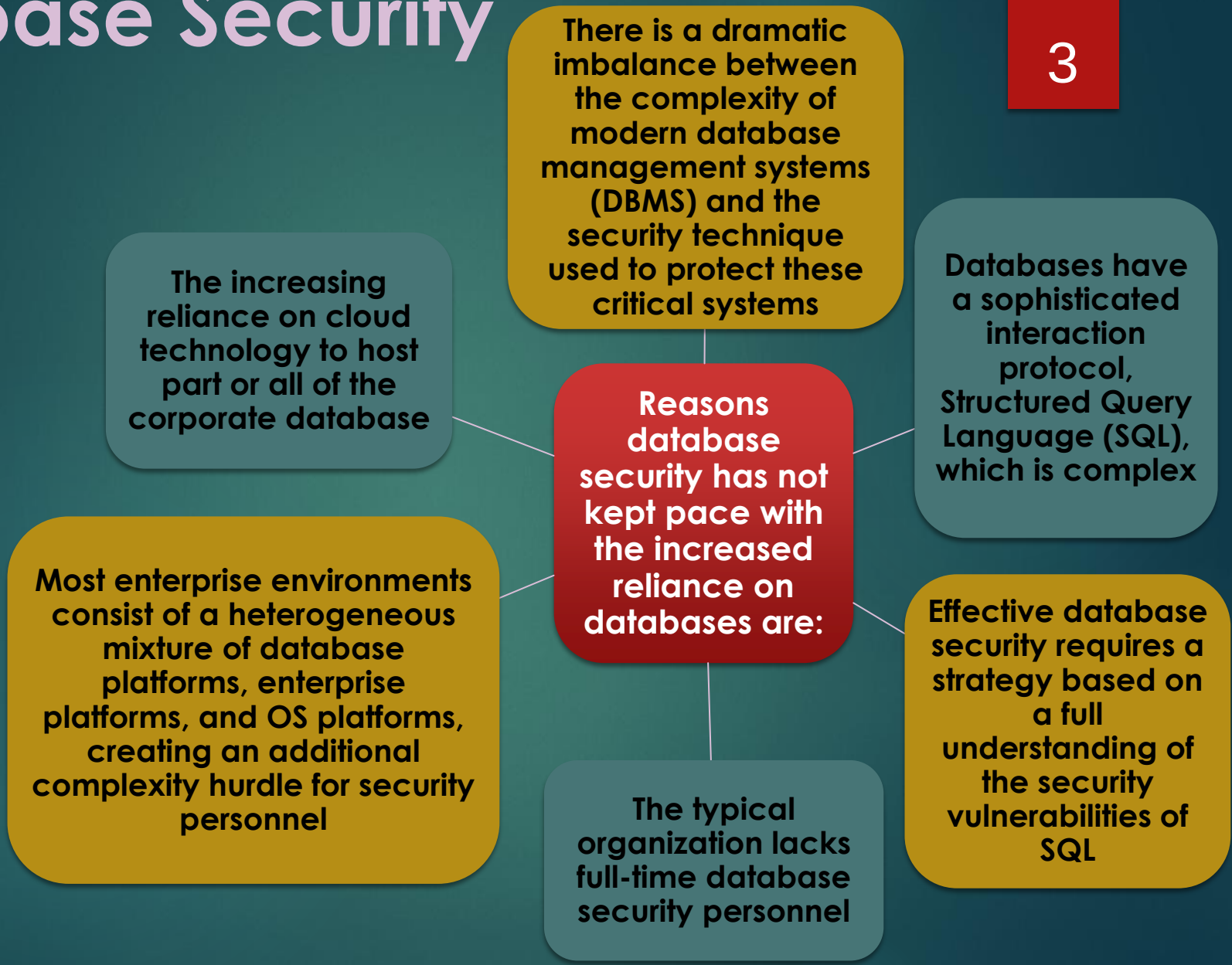
DATABASE AND DATA CENTER SECURITY

LEARNING OBJECTIVES

After studying this chapter, you should be able to:

- Understand the unique need for database security, separate from ordinary computer security measures.
- Present an overview of the basic elements of a database management system.
- Present an overview of the basic elements of a relational database system.
- 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.
- Discuss security issues related to data centers.

Database Security



Databases

- Structured collection of data stored for use by one or more applications
- Contains the relationships between data items and groups of data items
- Can sometimes contain sensitive data that needs to be secured

Query language

- Provides a uniform interface to the database for users and applications

Database management system (DBMS)

- Suite of programs for constructing and maintaining the database
- Offers ad hoc query facilities to multiple users and applications

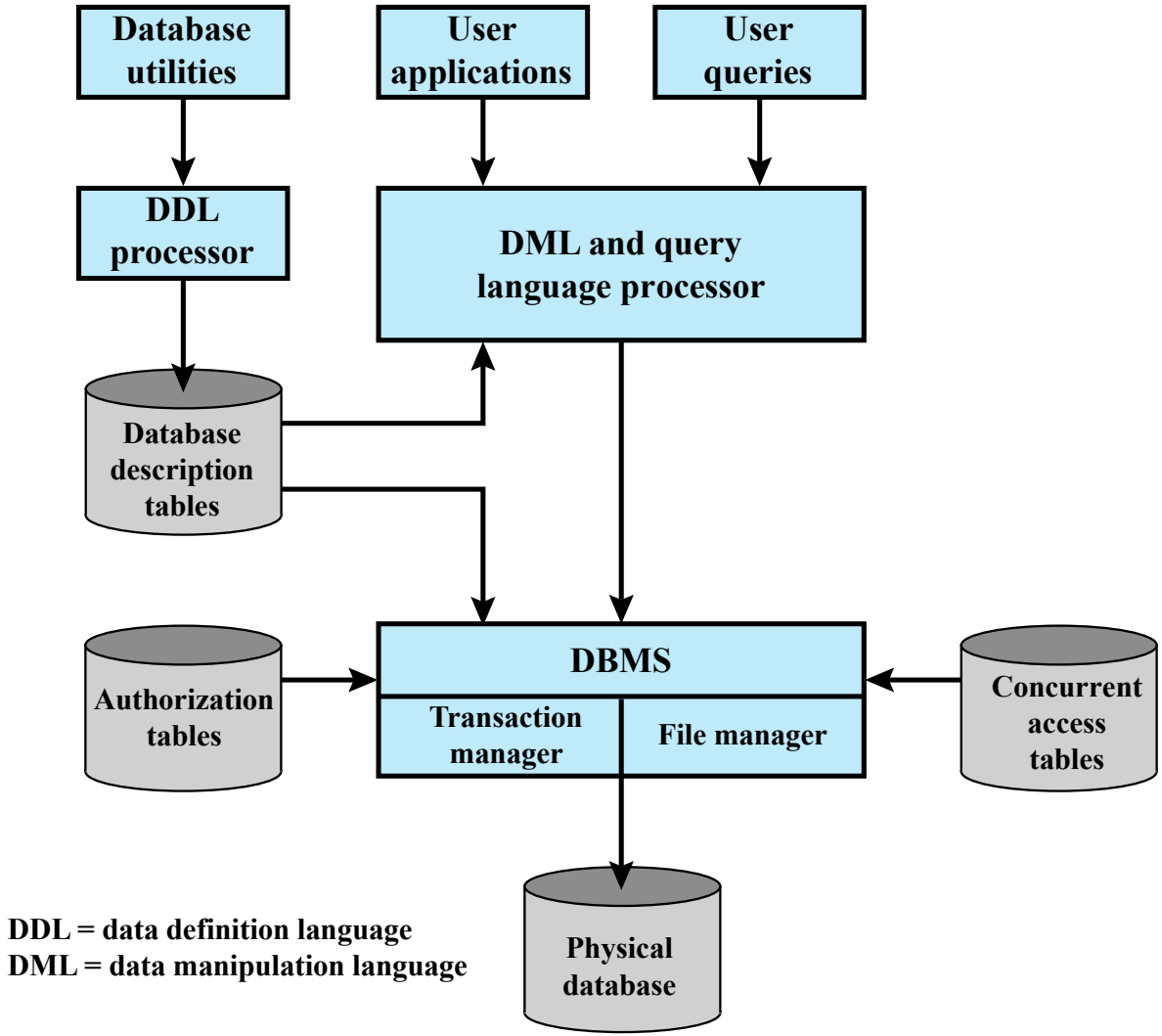


Figure 5.1 DBMS Architecture

Relational Databases

- Table of data consisting of rows and columns
  - Each column holds a particular type of data
  - Each row contains a specific value for each column
  - Ideally has one column where all values are unique, forming an identifier/key for that row
- Enables the creation of multiple tables linked together by a unique identifier that is present in all tables
- Use a relational query language to access the database
  - Allows the user to request data that fit a given set of criteria

- 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.

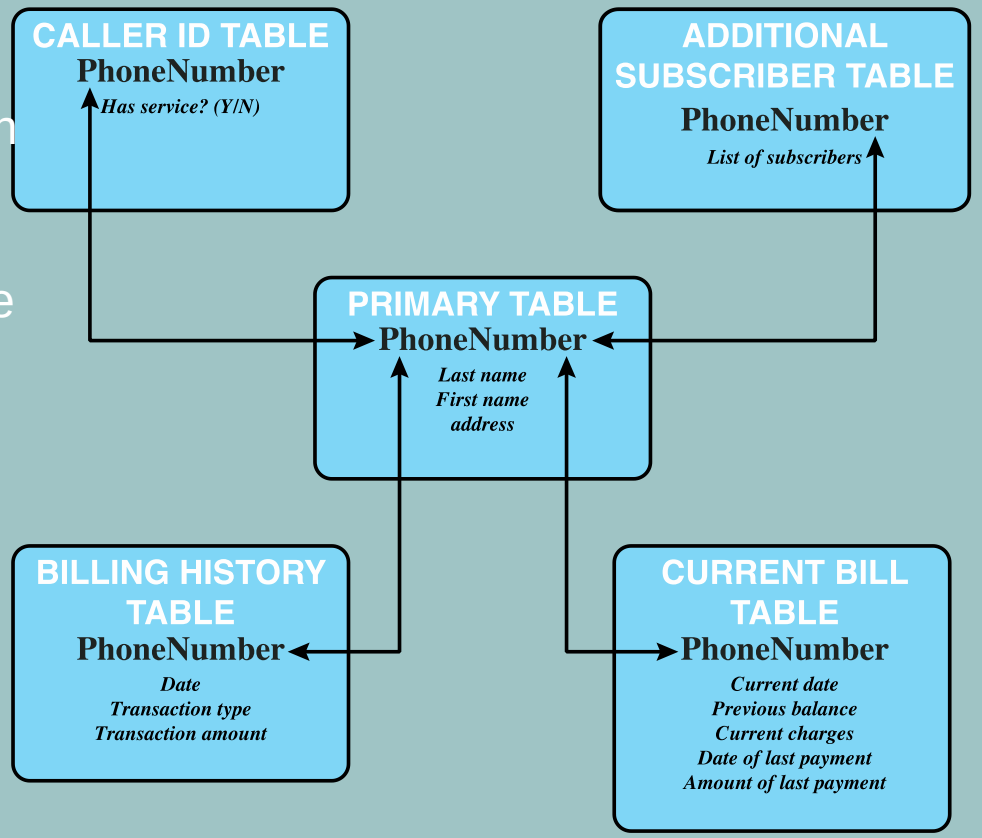


Figure 5.2 Example Relational Database Model. A relational database uses multiple tables related to one another by a designated key; in this case the key is the PhoneNumber field.

Relational Database Elements

- Relation
  - Table/file
- Tuple
  - Row/record
- Attribute
  - Column/field

Primary key

- Uniquely identifies a row
- Consists of one or more column names

Foreign key

- Links one table to attributes in another

View/virtual table

- Result of a query that returns selected rows and columns from one or more tables
- Views are often used for security purposes

Table 5.1
Basic Terminology for Relational Databases

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

- An abstract model of a relational database table
- 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.

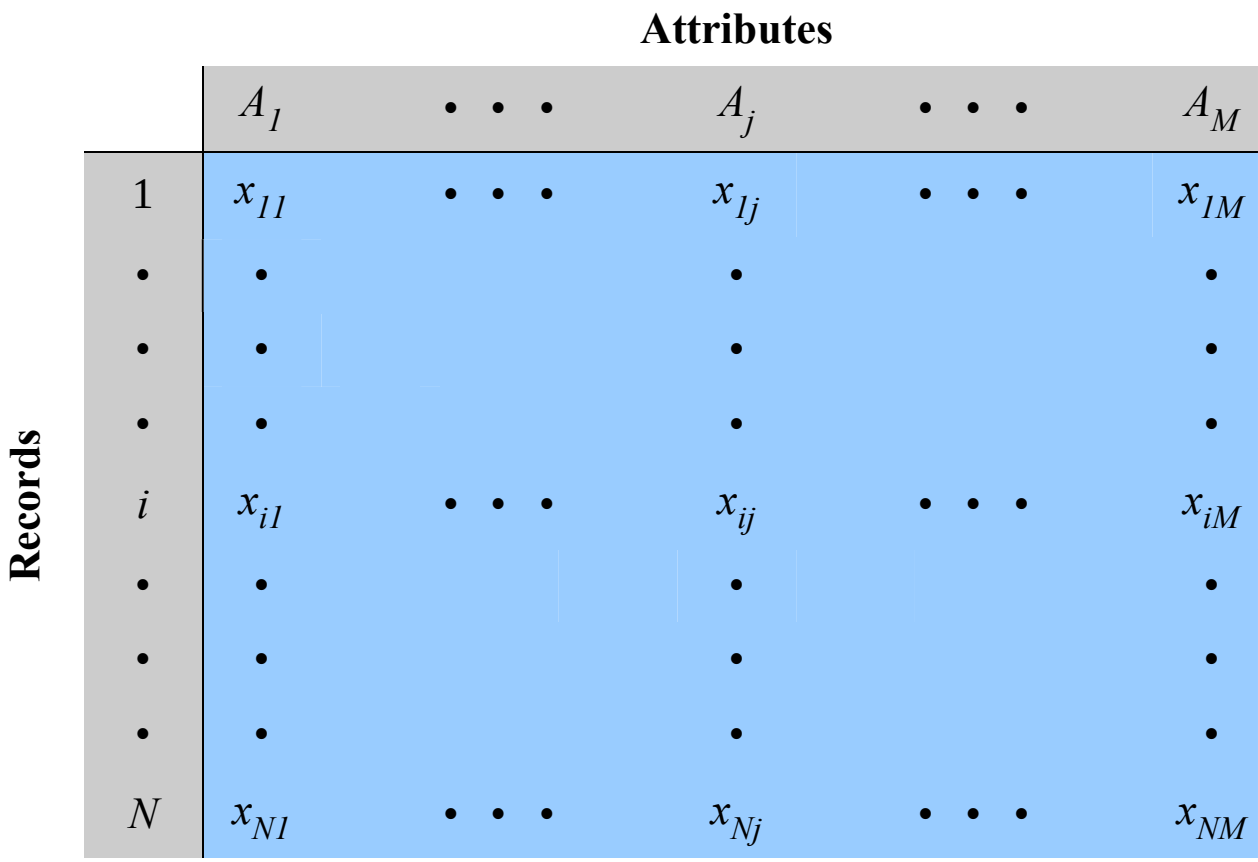


Figure 5.3 Abstract Model of a Relational Database

Department Table

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

primary key

Employee Table

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

foreign key

primary key

(a) Two tables in a relational database

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

(b) A view derived from the database

Figure 5.4 Relational Database Example

Structured Query Language (SQL)

- Standardized language to define schema, manipulate, and query data in a relational database
- Several similar versions of ANSI/ISO standard
- All follow the same basic syntax and semantics

SQL statements can be used to:

- Create tables
- Insert and delete data in tables
- Create views
- Retrieve data with query statements

Structured Query Language (SQL)

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

- For example, the two tables in Figure 5.4a 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) )
```

Structured Query Language (SQL)

- 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.

Structured Query Language (SQL)

- The view in Figure 5.4b 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
```

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 5.4 Relational Database Example

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.

SQL Injection Attacks (SQLi)

- One of the most prevalent and dangerous network-based security threats
- Designed to exploit the nature of Web application pages
- Sends malicious SQL commands to the database server
- Most common attack goal is bulk extraction of data
- Depending on the environment SQL injection can also be exploited to:
  - Modify or delete data
  - Execute arbitrary operating system commands
  - Launch denial-of-service (DoS) attacks

A Typical SQLi Attack

Figure 5.5, from [ACUN13], is a typical example of an SQLi attack. The steps involved are as follows:

1. Hacker finds a vulnerability in a custom Web application and injects an SQL command to a database by sending the command to the Web server. The command is injected into traffic that will be accepted by the firewall.
2. The Web server receives the malicious code and sends it to the Web application server.
3. The Web application server receives the malicious code from the Web server and sends it to the database server.
4. The database server executes the malicious code on the database. The database returns data from credit cards table.
5. The Web application server dynamically generates a page with data including credit card details from the database.
6. The Web server sends the credit card details to the hacker.

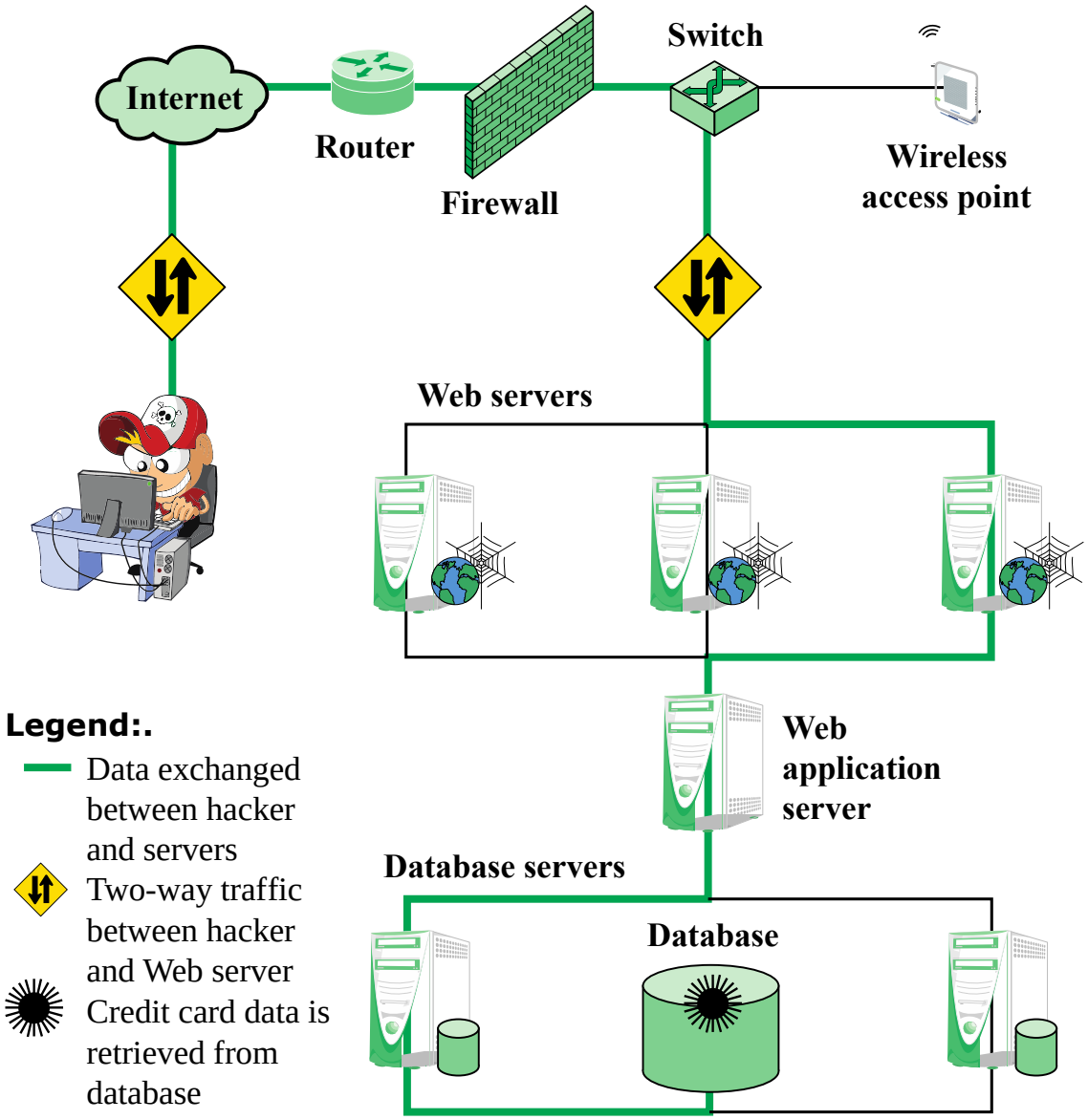
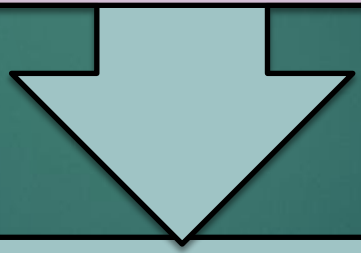


Figure 5.5 Typical SQL Injection Attack

Injection Technique

The SQLi attack typically works by prematurely terminating a text string and appending a new command

Because the inserted command may have additional strings appended to it before it is executed the attacker terminates the injected string with a comment mark "- -"



Subsequent text is ignored at execution time

Injection Technique

As a simple example, consider a script that build an SQL query by combining predefined strings with text entered by a user:

```
var Shipcity;
ShipCity = Request.form ("ShipCity");
var sql = "select * from OrdersTable where ShipCity = '" +
ShipCity + "'";
```


The intention of the script's designer is that a user will enter the name of a city. For example, when the script is executed, the user is prompted to enter a city, and if the user enters Redmond, then the following SQL query is generated:

```
SELECT * FROM OrdersTable WHERE ShipCity = 'Redmond'
```

Suppose, however, the user enters the following:

```
Boston'; DROP table OrdersTable--
```

This results in the following SQL query:

```
SELECT * FROM OrdersTable WHERE ShipCity =
'Redmond'; DROP table OrdersTable--
```

The semicolon is an indicator that separates two commands, and the double dash is an indicator that the remaining text of the current line is a comment and not to be executed. When the SQL server processes this statement, it will first select all records in `OrdersTable where ShipCity is Redmond`. Then, it executes the DROP request, which deletes the table.

SQLi Attack Avenues

User input

- Attackers inject SQL commands by providing suitable crafted user input

Server variables

- Attackers can forge the values that are placed in HTTP and network headers and exploit this vulnerability by placing data directly into the headers

Second-order injection

- A malicious user could rely on data already present in the system or database to trigger an SQL injection attack, so when the attack occurs, the input that modifies the query to cause an attack does not come from the user, but from within the system itself

Cookies

- An attacker could alter cookies such that when the application server builds an SQL query based on the cookie's content, the structure and function of the query is modified

Physical user input

- Applying user input that constructs an attack outside the realm of web requests

SQLi Attack Avenues

- **User input:** In this case, attackers inject SQL commands by providing suitably crafted user input. A Web application can read user input in several ways based on the environment in which the application is deployed. In most SQLi attacks that target Web applications, user input typically comes from form submissions that are sent to the Web application via HTTP GET or POST requests. Web applications are generally able to access the user input contained in these requests as they would access any other variable in the environment.

- **Server variables:** Server variables are a collection of variables that contain HTTP headers, network protocol headers, and environmental variables. Web applications use these server variables in a variety of ways, such as logging usage statistics and identifying browsing trends. If these variables are logged to a database without sanitization, this could create an SQL injection vulnerability. Because attackers can forge the values that are placed in HTTP and network headers, they can exploit this vulnerability by placing data directly into the headers. When the query to log the server variable is issued to the database, the attack in the forged header is then triggered.

SQLi Attack Avenues

- **Second-order injection:** Second-order injection occurs when incomplete prevention mechanisms against SQL injection attacks are in place. In second-order injection, a malicious user could rely on data already present in the system or database to trigger an SQL injection attack, so when the attack occurs, the input that modifies the query to cause an attack does not come from the user, but from within the system itself.

- **Cookies:** When a client returns to a Web application, cookies can be used to restore the client's state information. Because the client has control over cookies, an attacker could alter cookies such that when the application server builds an SQL query based on the cookie's content, the structure and function of the query is modified.

- **Physical user input:** SQL injection is possible by supplying user input that constructs an attack outside the realm of Web requests. This user-input could take the form of conventional barcodes, RFID tags, or even paper forms which are scanned using

Types of SQL injection attacks

- There are three broad categories to classify SQL injections, depending on the methods they use to gain access to back-end data and the extent of the potential damage they can cause:
  - **Inband**
  - **Inferential SQLi (Blind SQLi)**
  - **Out-of-band**

Types of SQL injection attacks

- **Inband:**
  - An **inband attack** uses the same communication channel for injecting SQL code and retrieving results. The retrieved data are presented directly in the application webpage.
- **Inferential attack:**
  - There is no actual transfer of data, but the attacker is able to reconstruct the information by sending particular requests and observing the resulting behavior of the website/database server.
- **Out-of-band attack**:
  - In an **out-of-band attack**, data are retrieved using a different channel (e.g., an e-mail with the results of the query is generated and sent to the tester). This can be used when there are limitations on information retrieval, but outbound connectivity from the database server is lax.

Inband Attacks

- Inband attack types include the following:

Tautology

This form of attack injects code in one or more conditional statements so that they always evaluate to true

End-of-line comment

After injecting code into a particular field, legitimate code that follows are nullified through usage of end of line comments

Piggybacked queries

The attacker adds additional queries beyond the intended query, piggy-backing the attack on top of a legitimate request

Inband Attacks

- **Tautologies:**
  - **Attack Intent:** Identifying injectable parameters bypassing authentication, extracting data.
  - **Description:** In this type of attack, a code is injected into the input fields of the web application and one or more conditional statements are executed that are always true when executed.
    - This is one of the most widely and commonly **used method to bypass authentication and data extraction**. If the attack was performed successfully, it returns a set of record, or a result if some action is performed.
  - **Example:**
    - **The query for login is: SELECT * FROM user_info WHERE logID=“‘ or 1=1 --AND pass1=‘’**
  - The conditional statement OR (1=1) makes the entire WHERE clause to a tautology.
  - The query evaluated true for all records in the table and returns them.
  - In the above example, there is a value returned and hence a not null value is evaluated, making the authentication process successful allowing the user to login successfully without valid credentials.
    - This method is also employed to extract data from the database.

Inband Attacks

- **End-of-line comment:** After injecting code into a particular field, legitimate code that follows are nullified through usage of end of line comments.
  - An example would be to add "- -" after inputs so that remaining queries are not treated as executable code, but comments. The preceding tautology example is also of this form.
  - **Example: Select * from <table name> where userId = <id> and password = <wrongPassword> or 1=1;**
    - This is a query for sending an error message for wrong password.
- **Piggybacked queries:**
  - **Attack Intent**: Extracting data, adding or modifying data, performing denial of service, executing remote commands**.**
  - **Description:** In this type of attack, the statement after the ";" is executed. The attacker adds additional queries beyond the intended query, piggy-backing the attack on top of a legitimate request. This technique relies on server configurations that allow several different queries within a single string of code. The example in the preceding section is of this form.
  - **Example: SELECT account FROM user WHERE login="abc" AND pass=""; drop table user --" AND pin=123**
    - **The above query is generated if the attacker inputs ""; drop table users --" into the pass field.**

Inferential Attack

- With an inferential attack, there is no actual transfer of data, but the attacker is able to reconstruct the information by sending particular requests and observing the resulting behavior of the website/database server.

- Inferential attack types include the following:
  - Illegal/logically incorrect queries
  - Blind SQL injection

Inferential Attack

- **Illegal/logically incorrect queries:**
  - **Attack Intent:** Extraction of data, performing database finger-printing, identifies injectable parameters
  - This attack lets an attacker gather important information about the type and structure of the backend database of a Web application. The attack is considered a preliminary, information-gathering step for other attacks. The vulnerability leveraged by this attack is that the default error page returned by application servers is often overly descriptive. In fact, the simple fact that an error messages is generated can often reveal vulnerable/injectable parameters to an attacker.
  - **Example: Select * from <table name> where userId = <id> and password = <wrongPassword> or 1=1;**
    - **This is a query for sending an error message for wrong password**

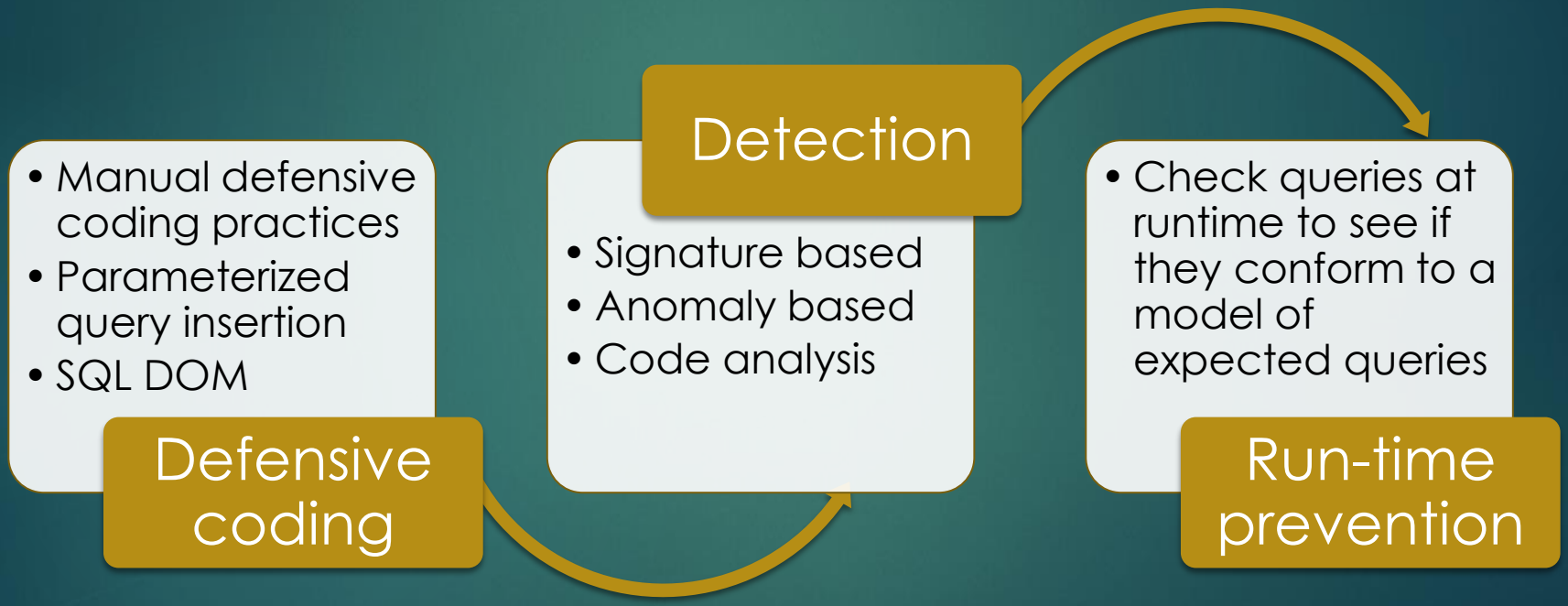
- **Blind SQL injection:**
  - **Attack Intent: Extraction of data, Theft of data**
  - Blind SQL injection allows attackers to infer the data present in a database system even when the system is sufficiently secure to not display any erroneous information back to the attacker. The attacker asks the server true/false questions. If the injected statement evaluates to true, the site continues to function normally. If the statement evaluates to false, although there is no descriptive error message, the page differs significantly from the normally functioning page.
  - **Example: SELECT name FROM WHERE id= and 1 =0 -- AND pass = SELECT name FROM WHERE id= and 1 = 1 -- AND pass =**

Out-of-Band Attack

- **In an out-of-band attack:** data are retrieved using a different channel (e.g., an e-mail with the results of the query is generated and sent to the tester). This can be used when there are limitations on information retrieval, but outbound connectivity from the database server is lax.

SQLi Countermeasures

- Three types:



SQLi Countermeasures

- **Defensive Coding**
  - **Manual defensive coding practices:** A common vulnerability exploited by **SQLi attacks is insufficient input validation.** The straightforward solution for eliminating these vulnerabilities is to **apply suitable defensive coding practices**.
    - **An example is input type checking, to check that inputs that are supposed to be numeric contain no characters other than digits.**
      - This type of technique can avoid attacks based on forcing errors in the database management system.
    - **Another type of coding practice is one that performs pattern matching to try to distinguish normal input from abnormal input.**
  - **Parameterized query insertion**: This approach attempts to prevent SQLi by allowing the application developer to more accurately specify the structure of an SQL query, and pass the value parameters to it separately such that any unsanitary user input is not allowed to modify the query structure.
  - **SQL DOM:** SQL DOM is a set of classes that enables automated data type validation and escaping [MCCL05].
    - This approach uses encapsulation of database queries to provide a safe and reliable way to access databases.
    - This changes the query-building process from an unregulated one that uses string concatenation to a systematic one that uses a type-checked API.
    - Within the API, developers are able to systematically apply coding best practices such as input filtering and rigorous type checking of user input.

SQLi Countermeasures

- A variety of **detection methods** have been developed, including the following:
  - **Signature-based:** This technique attempts to match specific attack patterns. Such an approach must be constantly updated and may not work against self modifying attacks.
  - **Anomaly-based:** This approach attempts to define normal behavior then detect behavior patterns outside the normal range. A number of approaches have been used.
    - In general terms, there is a training phase, in which the system learns the range of normal behavior, followed by the actual detection phase.
  - **Code analysis:** Code analysis techniques involve the use of a test suite to detect SQLi vulnerabilities. The test suite is designed to generate a wide range of SQLi attacks and assess the response of the system.

Database Access Control

Database access control system determines:

If the user has access to the entire database or just portions of it

What access rights the user has (create, insert, delete, update, read, write)

Can support a range of administrative policies

Centralized administration

- Small number of privileged users may grant and revoke access rights

Ownership-based administration

- The creator of a table may grant and revoke access rights to the table

Decentralized administration

- 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

SQL Access Definition

- Two commands for managing access rights:
  - **Grant**
    - Used to grant one or more access rights or can be used to assign a user to a role
  - **Revoke**
    - Revokes the access rights
- In general terms, the GRANT command has the following syntax:

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

- **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 the user **ricflair** to query any table in the database

SQL Access Definition

- 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

The REVOKE command has the following syntax:

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

The following statement revokes the access rights of the preceding example:

REVOKE SELECT ON ANY TABLE FROM ricflair

Cascading Authorizations

- The figure indicates that Ann grants the access right to Bob at time t = 10 and to Chris at time t = 20.
- Assume 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.

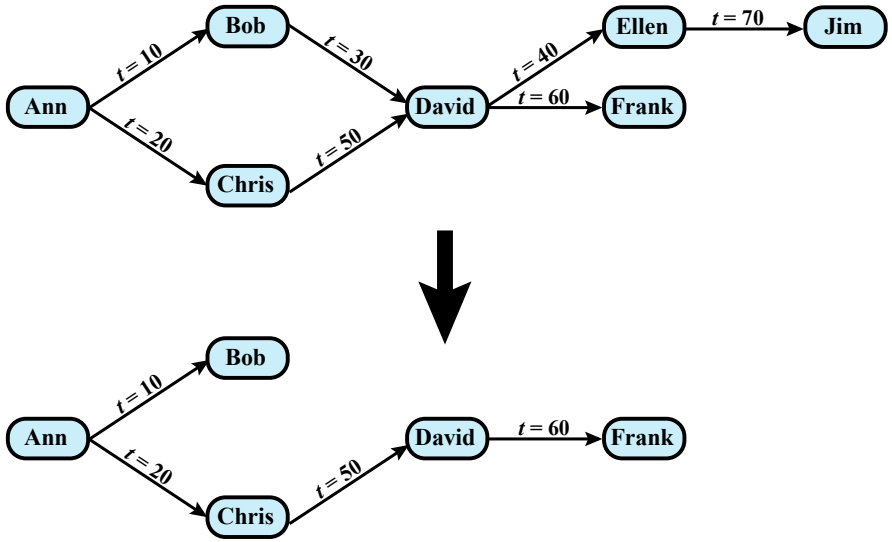


Figure 5.6 Bob Revokes Privilege from David

Role-Based Access Control (RBAC)

- 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.
- Role-based access control eases administrative burden and improves security
- A database RBAC needs to provide the following capabilities:
  - Create and delete roles
  - Define permissions for a role
  - Assign and cancel assignment of users to roles
- Categories of database users:

Application owner	End user	Administrator
• An end user who owns database objects as part of an application	• An end user who operates on database objects via a particular application but does not own any of the database objects	• User who has administrative responsibility for part or all of the database

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 (DDL) 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

(Table is on page 165 in the textbook)

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 higher sensitivity

Figure 5.7 illustrates the process. The attacker may make use of non-sensitive 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**.

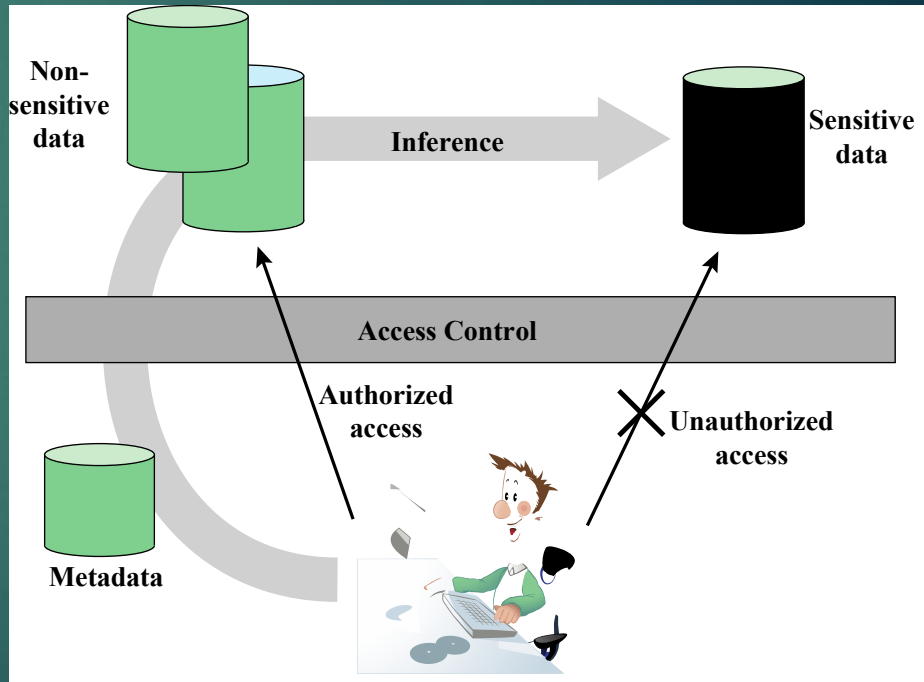


Figure 5.7 Indirect Information Access Via Inference Channel

Inference

- 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:

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

```
CREATE view V2 AS
SELECT Item, Department
FROM Inventory
WHERE Department = "hardware"
```

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.**

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

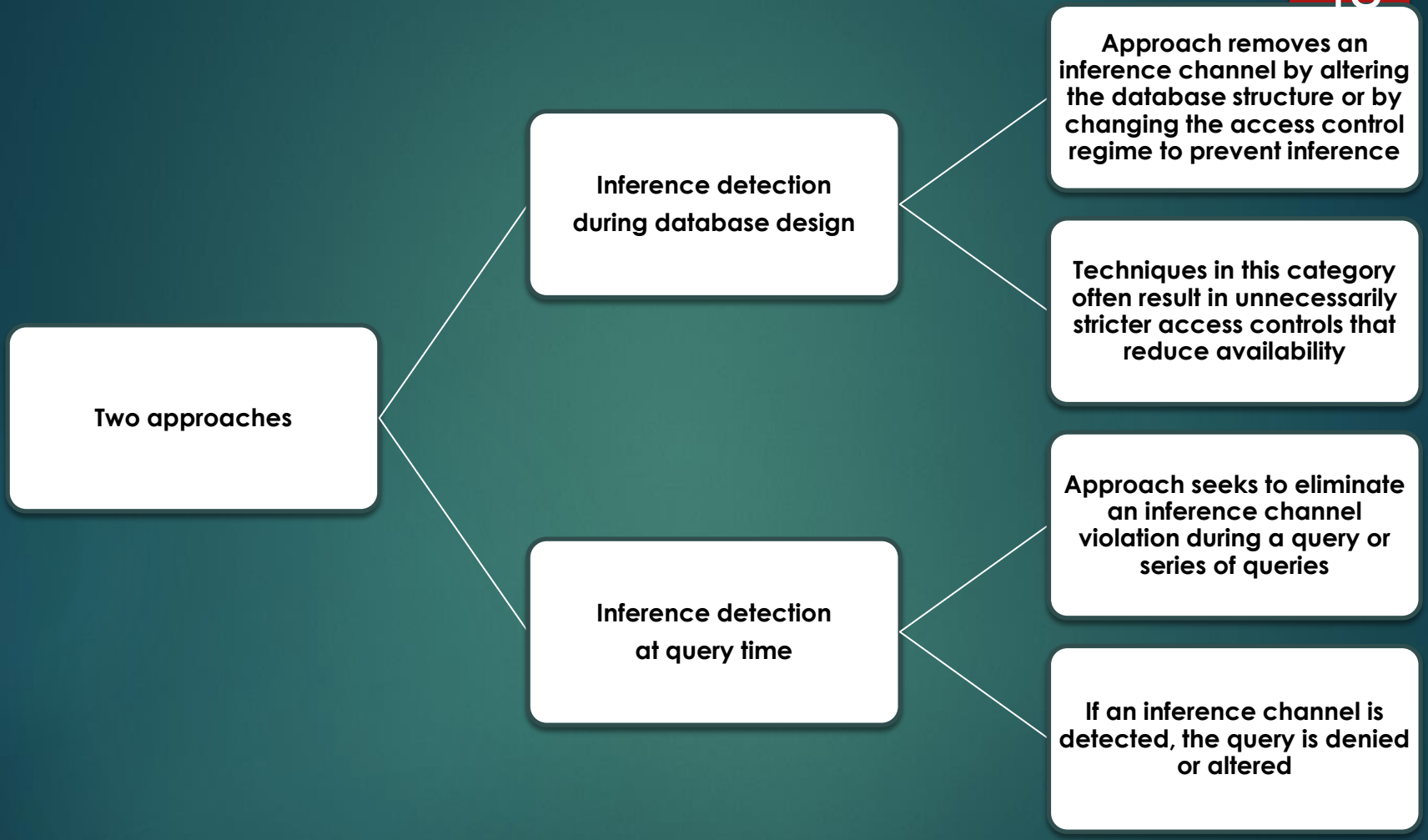
(b) Two views

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**

Inference Detection



- Some inference detection algorithm is needed for either of these approaches
- Progress has been made in devising specific inference detection techniques for multilevel secure databases and statistical databases

Inference

- 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
- Figure 5.8. 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.

Inference

- Now, suppose we want to add a new attribute, employee start date, which is not sensitive. This could be added to the Salaries table as follows:

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

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

Database Encryption

- The database is typically the most valuable information resource for any organization
  - Protected by multiple layers of security
    - Firewalls, authentication, general access control systems, DB access control systems, database encryption
    - Encryption becomes the last line of defense in database security
  - **Encryption can be applied to the entire database, at the record level, the attribute level, or level of the individual field**
- Disadvantages to encryption:
  - **Key management**
    - Authorized users must have access to the decryption key for the data for which they have access
  - **Inflexibility**
    - When part or all of the database is encrypted it becomes more difficult to perform record searching

Database Encryption

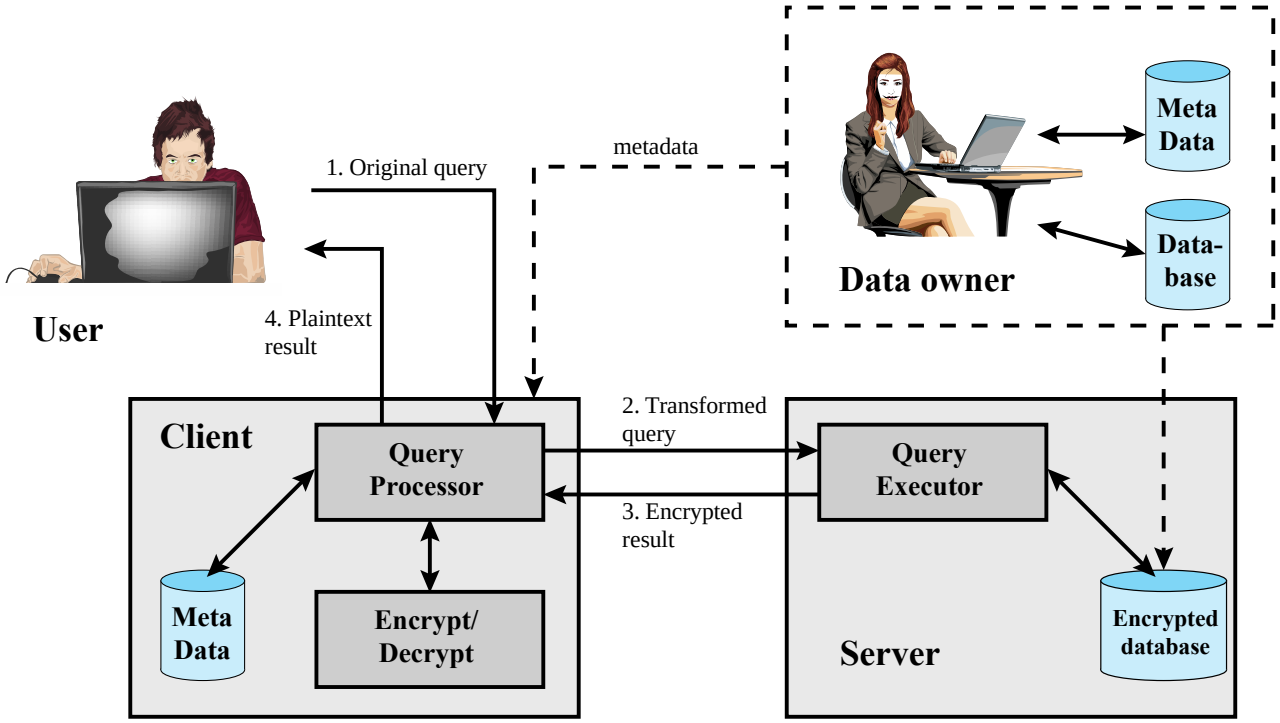


Figure 5.9 A Database Encryption Scheme

Database Encryption

An example of such an approach, depicted in Figure 5.9, 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.
- **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.

Database Encryption

Let us first examine the simplest possible arrangement based on this scenario.
Suppose 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 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.

Database Encryption

- For example, consider this query, which was introduced in database of Figure 5.4a:

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

- Assume the encryption key *k* is used and the encrypted value of the department id 15 is $E(k, 15) = 1000110111001110$.

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

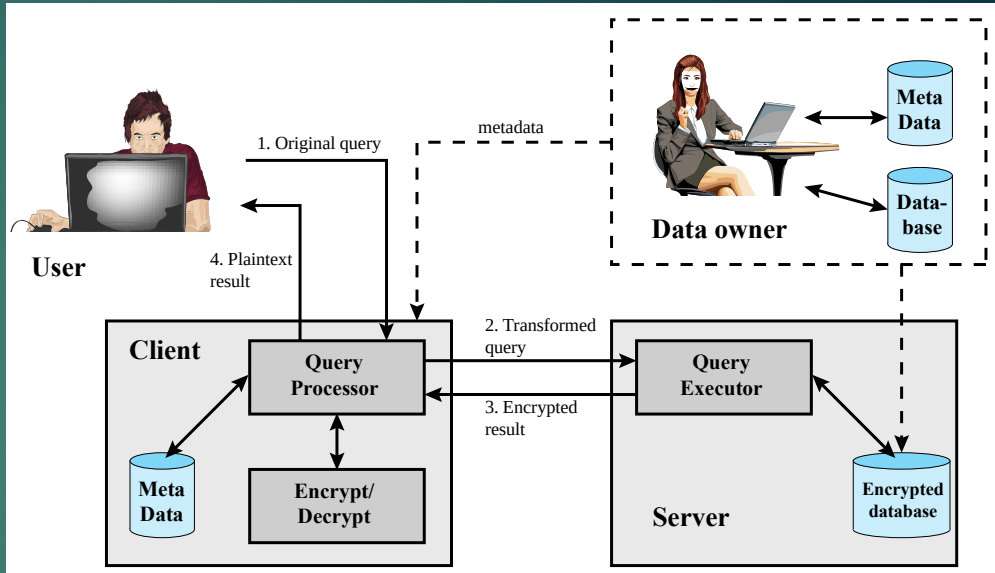


Figure 5.9 A Database Encryption Scheme

- 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

- 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 5.3, each row R_i is treated as a contiguous block

$$B_i = (x_{i1} \| x_{i2} \| c \| x_{iM}).$$

- The entire row is encrypted, expressed as

$$E(k, B_i) = E(k, (x_{i1} \| x_{i2} \| c \| x_{iM})).$$

- 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 (see Figure 5.10):

$$(x_{i1}, x_{i2}, c, x_{iM}) \; S[E(k, B_i), I_{i1}, I_{i2}, c, I_{iM}]$$

$E(k, B_1)$	I_{11}	• • •	I_{1j}	• • •	I_{1M}
•	•		•		•
•	•		•		•
•	•		•		•
$E(k, B_i)$	I_{i1}	• • •	I_{ij}	• • •	I_{iM}
•	•		•		•
•	•		•		•
•	•		•		•
$E(k, B_N)$	I_{N1}	• • •	I_{Nj}	• • •	I_{NM}

$B_i = (x_{i1} \| x_{i2} \| \ldots \| x_{iM})$

Figure 5.10 Encryption Scheme for Database of Figure 5.3

- Table 5.3 provides an example of this mapping.
- Suppose 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]; then assign index values 1, 2, 3, 4, and 5, respectively

- Table 5.3b 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.

Table 5.3 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(eid)	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

- This arrangement provides for more efficient data retrieval.
- Suppose, for example, a user requests records for all employees with eid 600.
  - The query processor requests all records with I (eid) = 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.

- To increase the efficiency of accessing records by means of the primary key, the system could use the encrypted 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 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.