

# Information Security

## CS3002

Lecture 15  
19th October 2023

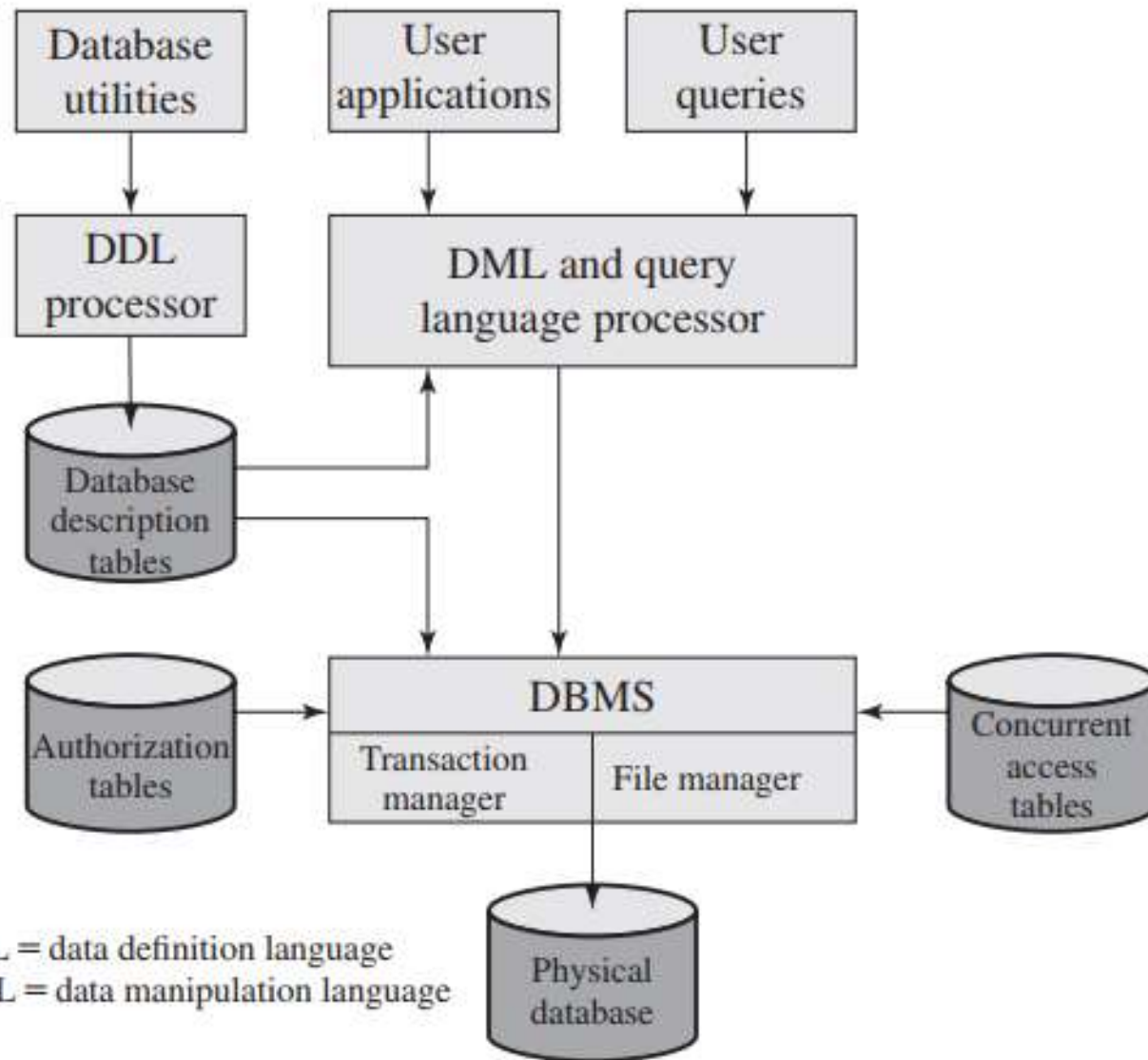
Dr. Rana Asif Rehman  
Email: [r.asif@lhr.nu.edu.pk](mailto:r.asif@lhr.nu.edu.pk)



# Database Systems

- 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

# Database Architecture

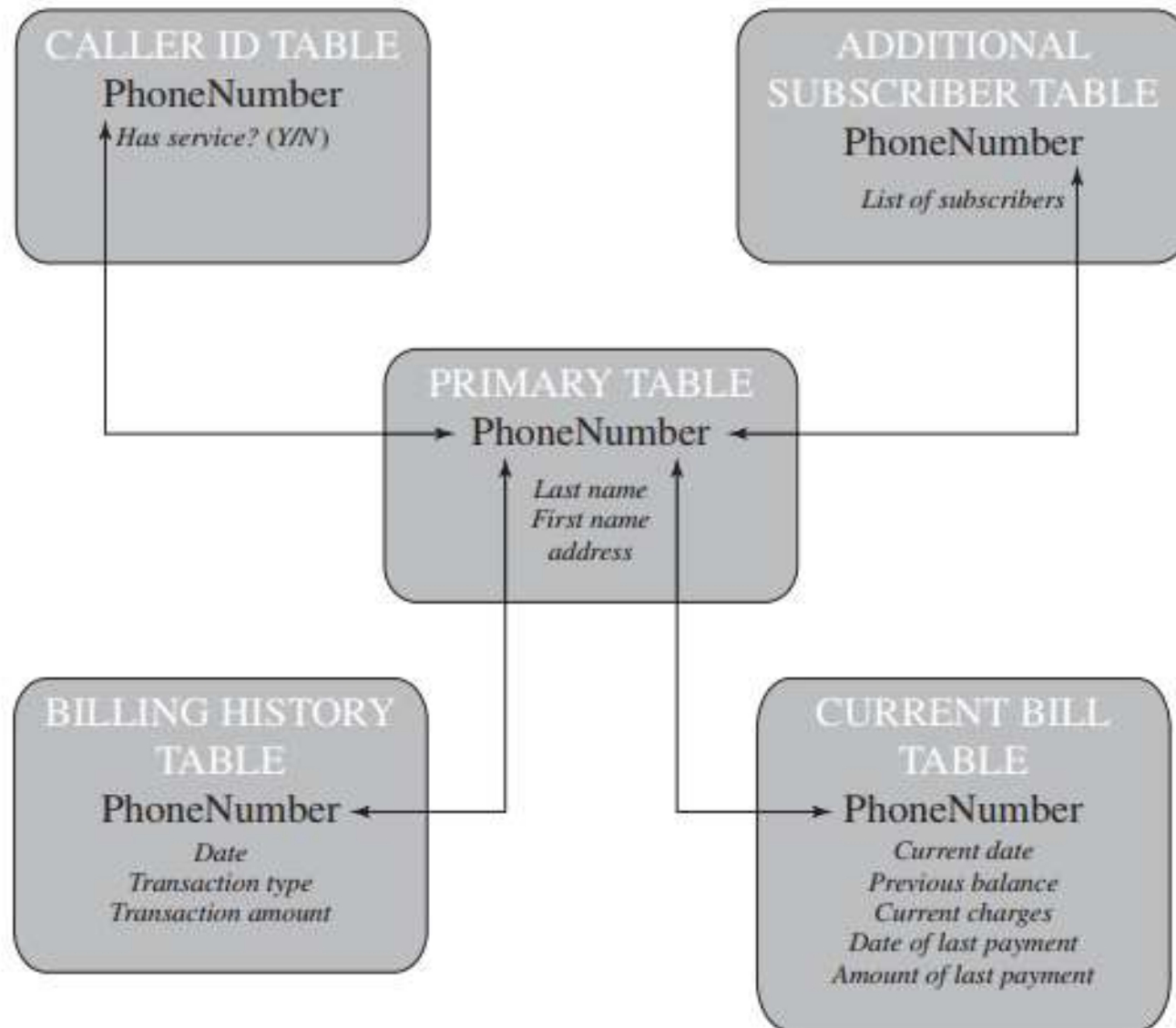


# Relational Databases

- 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

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

# A Relational Database Example



# Relational Database Elements

Department Table			Employee Table				
Did	Dname	Dacctno	Ename	Did	Salarycode	Eid	Ephone
4	human resources	528221	Robin	15	23	2345	6127092485
8	education	202035	Neil	13	12	5088	6127092246
9	accounts	709257	Jasmine	4	26	7712	6127099348
13	public relations	755827	Cody	15	22	9664	6127093148
15	services	223945	Holly	8	23	3054	6127092729
			Robin	8	24	2976	6127091945
			Smith	9	21	4490	6127099380

Primary key

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

# Structured Query Language

- **Structured Query Language (SQL)**
  - originally developed by IBM in the mid-1970s
  - standardized language to define schema, manipulate, and query data in a relational database
  - several similar versions of ANSI/ISO standard

```
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) )
```

```
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
```

# SQL – Background

- SQL
  - Widely used database query language
  - Fetch a set of records  
`SELECT * FROM Person WHERE Username= 'Vitaly'`
  - Add data to the table  
`INSERT INTO Key (Username, Key) VALUES ( 'Vitaly' , 3611BBFF)`
  - Modify data  
`UPDATE Keys SET Key=FA33452D WHERE PersonID=5`
  - Query syntax (mostly) independent of vendor



# SQL – Background

- Typical Login Prompt



User Login - Microsoft Internet Explorer

File Edit View Favorites Tools Help

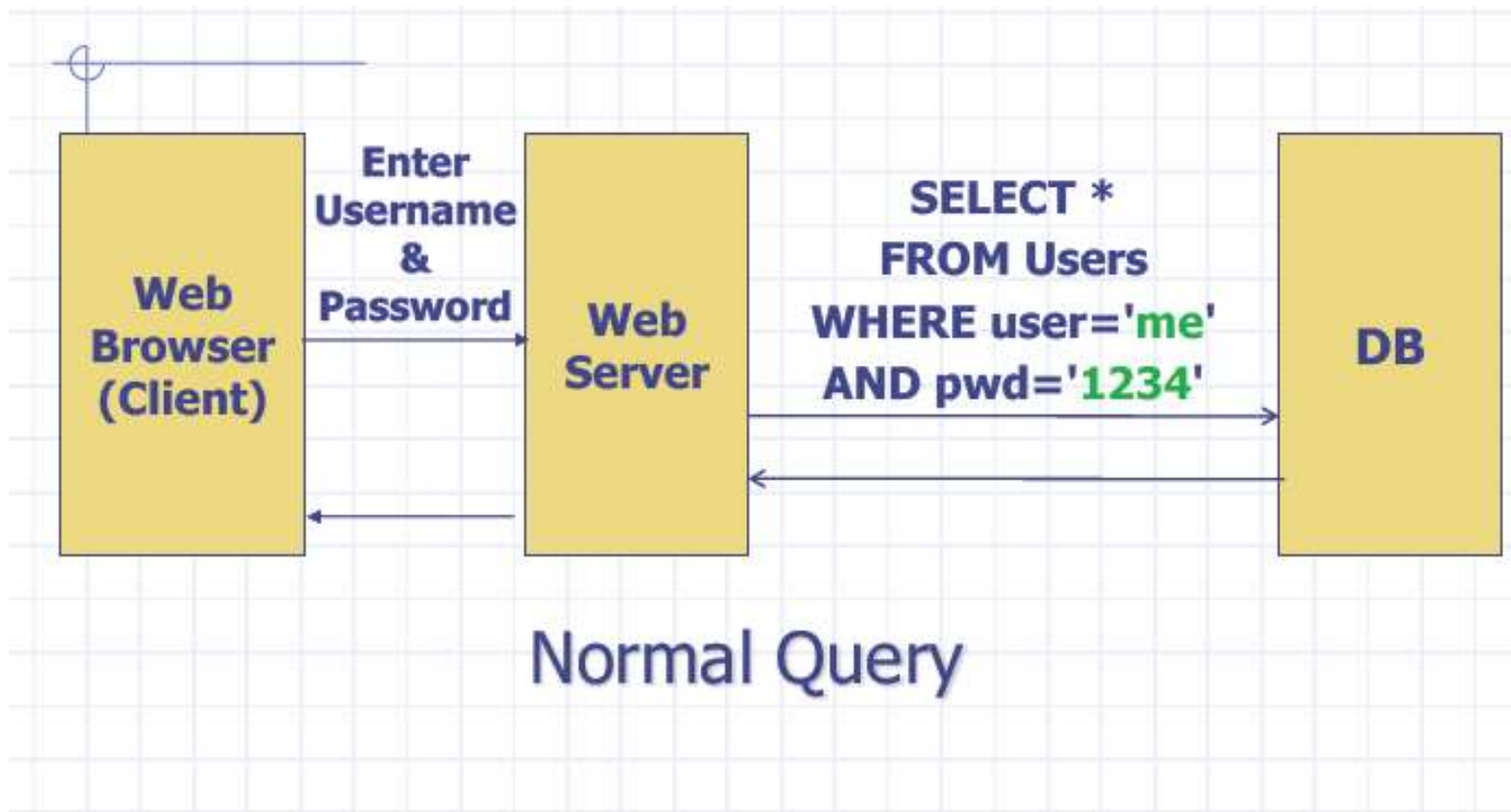
Back Forward Stop Reload Home Search

Enter User Name: smith

Enter Password: ●●●●●●●

Login

# SQL – Background



# SQL – Background

```
| set ok = execute( "SELECT * FROM Users  
|           WHERE user=' ' & form("user") & " '  
|           AND   pwd=' ' & form("pwd") & " ' " );  
  
| if not ok.EOF  
|     login success  
| else fail;
```

# SQL Injection Attacks

- One of the most prevalent and dangerous network-based security threats
- Sends malicious SQL commands to the database server
- 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

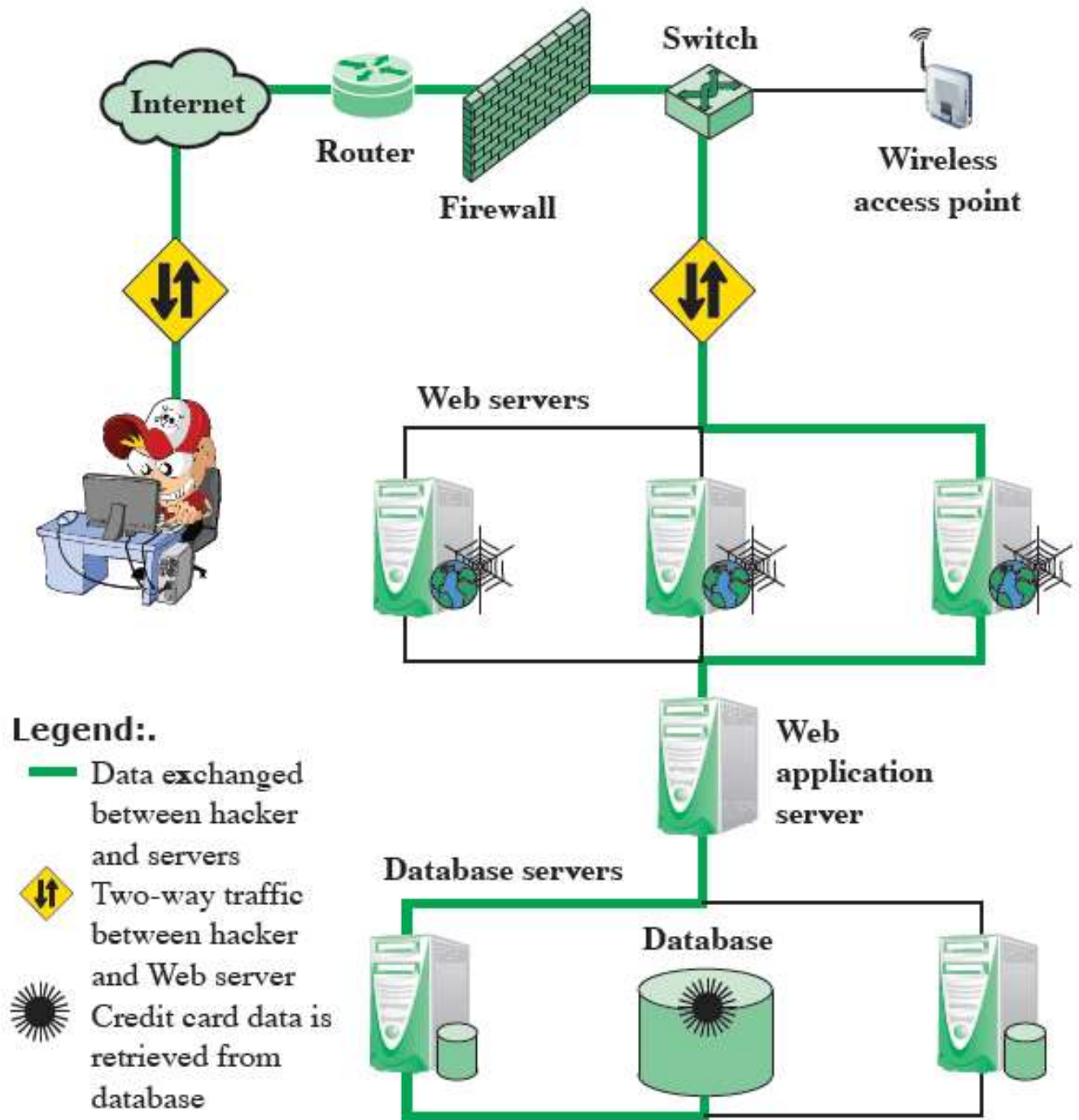
# SQL injection (SQLi)

- An application security weakness that allows attackers to control an application's database Access or delete data
- Change an application's data-driven behavior and do other undesirable things – by
- Method: Sending unexpected SQL commands.

# SQL injection (SQLi)

- Control application behavior that's based on data in the database, for example by tricking an application into allowing a login without a valid password
- Alter data in the database without authorization, for example by creating fraudulent records, adding users or “promoting” users to higher access levels, or deleting data
- Access data without authorization, for example by tricking the database into providing too many results for a query

# A Typical Injection Attack



# Injection Attack Steps

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



# Technique of SQL Injection

- **Union Operator:** can be used when the SQL injection flaw happens in a SELECT statement, making it possible to combine two queries into a single result or result set.
- **Boolean:** use Boolean condition(s) to verify whether certain conditions are true or false.
- **Error based:** this technique forces the database to generate an error, giving the attacker or tester information upon which to refine their injection.
- **Out-of-band:** technique used to retrieve data using a different channel (e.g., make a HTTP connection to send the results to a web server).
- **Time delay:** use database commands (e.g. sleep) to delay answers in conditional queries. It is useful when attacker doesn't have some kind of answer (result, output, or error) from the application.

# SQL Injection Attacks – Bad Input

◆ Suppose `user = " 'or 1=1 -- "` (URL encoded)

◆ Then script does:

```
ok = execute( SELECT ...  
              WHERE user= ' ' or 1=1 -- ... )
```

- The `--` causes rest of line to be ignored.
- Now `ok.EOF` is always false and login succeeds.

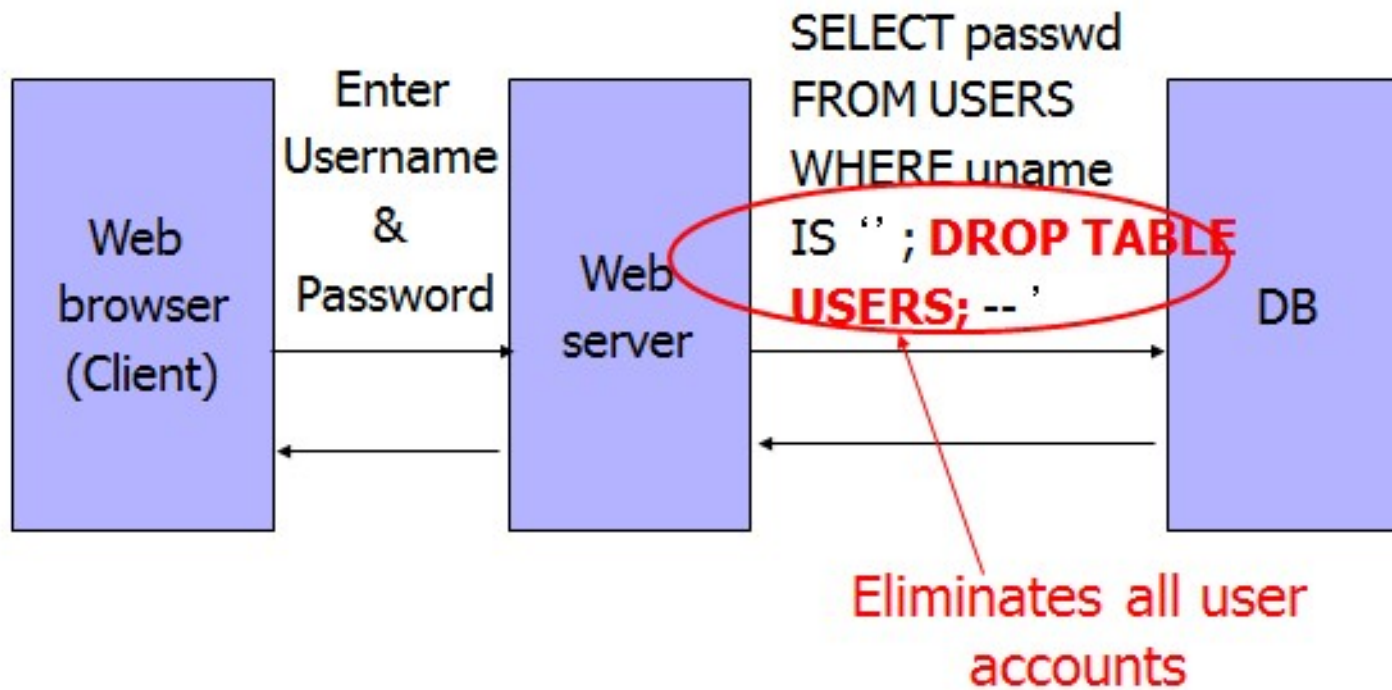
◆ The bad news: easy login to many sites this way.

# SQL Injection Attacks (cont.)

- Malicious User Input



# SQL Injection Attacks (cont.)



# SQL Injection Attacks (cont.)

◆ Then script does:

```
ok = execute( SELECT ...  
              WHERE user= ' ' ; DROP TABLE Users ... )
```

◆ Deletes user table

- Similarly: attacker can add users, reset pwds, etc.

# SQL Injection Attacks – Even Worse

◆ Suppose user =

```
' ; exec cmdshell
```

```
'net user badguy badpwd' / ADD --
```

◆ Then script does:

```
ok = execute( SELECT ...
```

```
WHERE username= ' ' ; exec ... )
```

If SQL server context runs as "sa", attacker gets account on DB server

# Types of Attack

- In-band Attack
- Out-of-band Attack
- Inferential or Blind Attack

# 1. In-band Attacks

Data is extracted using the same channel that is used to inject the SQL code. This is the most straightforward kind of attack, in which the retrieved data is presented directly in the application web page.

- **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



## 2. Out-of-bound 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 less strict

### 3. Inferential Attack (gathering info)

- 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
  - **Illegal/logically incorrect queries**: lets an attacker gather important information about the type and structure of the backend database of a Web application
    - The vulnerability leveraged by this attack is that the default error page returned by application servers is often overly descriptive.
  - **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 to observe the functionality in each case.

The **Common Vulnerabilities and Exposures (CVE)** system provides a reference-method for publicly known [information-security vulnerabilities](#) and exposures.

## CVE Details

*The ultimate security vulnerability datasource*

(e.g.: CVE-20

[Log In](#) [Register](#)

[Switch to https://](#)

[Home](#)

### Browse :

[Vendors](#)

[Products](#)

[Vulnerabilities By Date](#)

[Vulnerabilities By Type](#)

### Reports :

[CVSS Score Report](#)

[CVSS Score Distribution](#)

### Search :

[Vendor Search](#)

[Product Search](#)

[Version Search](#)

[Vulnerability Search](#)

[By Microsoft References](#)

### Top 50 :

[Vendors](#)

[Vendor Cvss Scores](#)

[Products](#)

[Product Cvss Scores](#)

[Versions](#)

### Other :

[Microsoft Bulletins](#)

[Bugtraq Entries](#)

[CWE Definitions](#)

[About & Contact](#)

[Feedback](#)

[CVE Help](#)

## Security Vulnerabilities (SQL Injection)

CVSS Scores Greater Than: [0](#) [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#)

Sort Results By : [CVE Number Descending](#) [CVE Number Ascending](#) [CVSS Score Descending](#) [Number Of Exploits Descending](#)

[Copy Results](#) [Download Results](#)

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access
1	<a href="#">CVE-2017-100208</a>			Sql	2017-09-14	2017-09-15	0.0	None	???
Vulnerability in wordpress plugin wordpress-gallery-transformation v1.0, SQL injection is in ./wordpress-gallery-transformation/ga into an SQL query.									
2	<a href="#">CVE-2017-1002022</a>	<a href="#">89</a>		Sql	2017-09-14	2017-09-18	7.5	None	Remote
Vulnerability in wordpress plugin surveys v1.01.8, The code in questions.php does not sanitize the survey variable before placing i									
3	<a href="#">CVE-2017-1002021</a>	<a href="#">89</a>		Sql	2017-09-14	2017-09-18	7.5	None	Remote
Vulnerability in wordpress plugin surveys v1.01.8, The code in individual_responses.php does not sanitize the survey_id variable b									
4	<a href="#">CVE-2017-1002019</a>	<a href="#">89</a>		Sql	2017-09-14	2017-09-18	7.5	None	Remote
Vulnerability in wordpress plugin eventr v1.02.2, The edit.php form and event_form.php code do not sanitize input, this allows for									
5	<a href="#">CVE-2017-1002018</a>	<a href="#">89</a>		Sql	2017-09-14	2017-09-18	7.5	None	Remote
Vulnerability in wordpress plugin eventr v1.02.2, The edit.php form and attendees.php code do not sanitize input, this allows for bl									
6	<a href="#">CVE-2017-1002015</a>			Sql	2017-09-14	2017-09-14	0.0	None	???
Vulnerability in wordpress plugin image-gallery-with-slideshow v1.5.2, Blind SQL Injection in image-gallery-with-slideshow/admin_									
7	<a href="#">CVE-2017-1002014</a>			Sql	2017-09-14	2017-09-14	0.0	None	???
Vulnerability in wordpress plugin image-gallery-with-slideshow v1.5.2, Blind SQL Injection in image-gallery-with-slideshow/admin_									
8	<a href="#">CVE-2017-1002013</a>			Sql	2017-09-14	2017-09-14	0.0	None	???

Information Security (CS3002)

27

# SQLi Countermeasures

- **Defensive coding:** Stronger data validation
  - type checking, to check that inputs that are supposed to be numeric contain no characters other than digits
  - pattern matching to try to distinguish normal input from abnormal input
- **Detection**
  - Signature based
  - Anomaly based
  - Code analysis
- **Runtime prevention:** Check queries at runtime to see if they conform to a model of expected queries

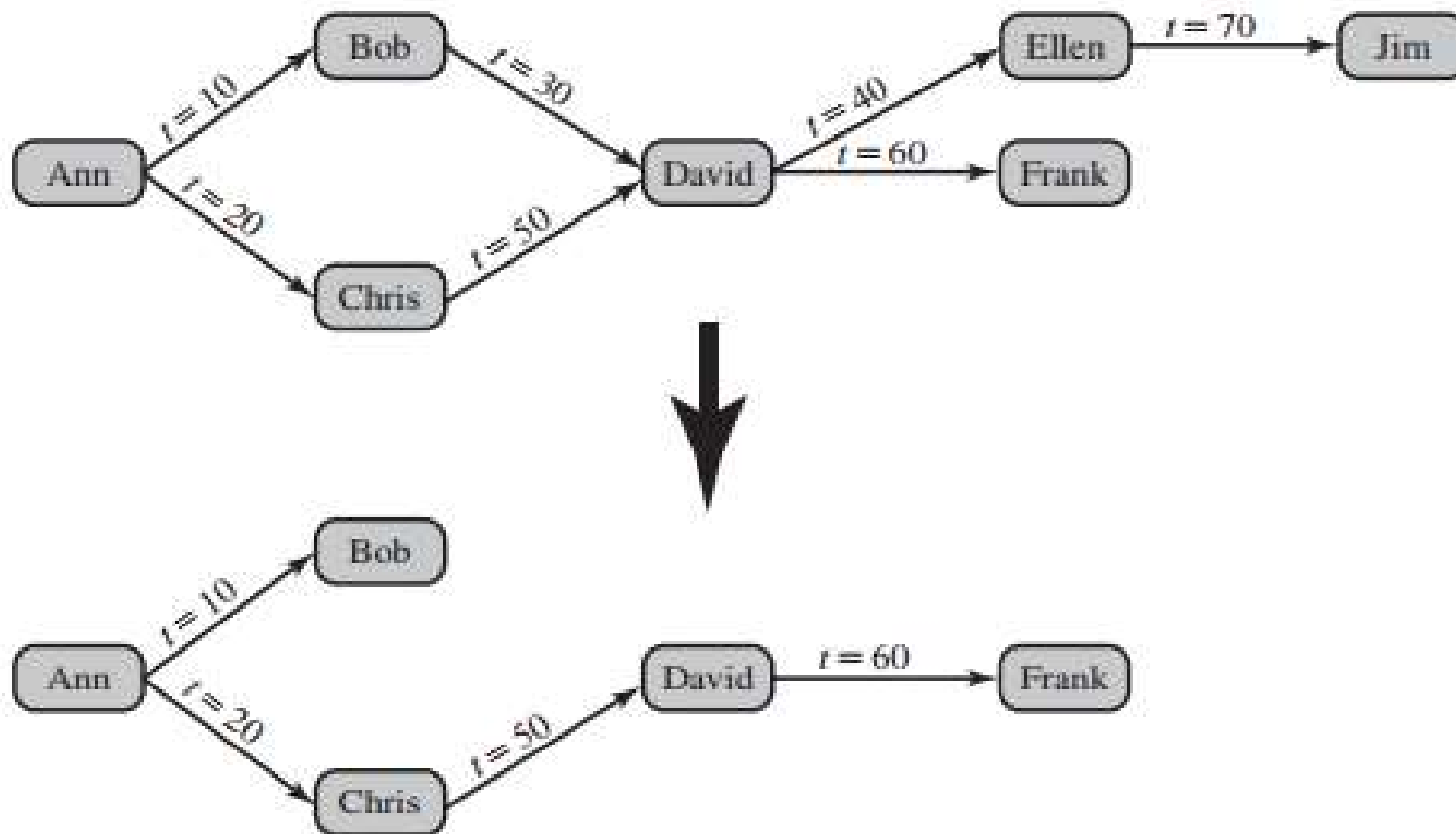
# Database Access Control

- DBMS provide access control for database
- It operates on assumption that user is already authenticated
- DBMS provides specific access rights to portions of the database
  - e.g. create, insert, delete, update, read, write
  - to entire database, tables, selected rows or columns
  - possibly dependent on contents of a table entry
- can support a range of policies:
  - centralized administration
  - ownership-based administration
  - decentralized administration

# 1.1. SQL Access Controls

- If the user has access to the entire database or just portions of it
- Two commands:
  - `GRANT {privileges | role} [ON table] TO {user | role | PUBLIC} [IDENTIFIED BY password] [WITH GRANT OPTION]`
    - e.g. `GRANT SELECT ON ANY TABLE TO john`
  - `REVOKE {privileges | role} [ON table] FROM {user | role | PUBLIC}`
    - e.g. `REVOKE SELECT ON ANY TABLE FROM john`
  - `WITH GRANT OPTION`: whether grantee can grant “GRANT” option to other users
- Typical access rights are:
  - `SELECT, INSERT, UPDATE, DELETE, REFERENCES`

# Cascading Authorizations



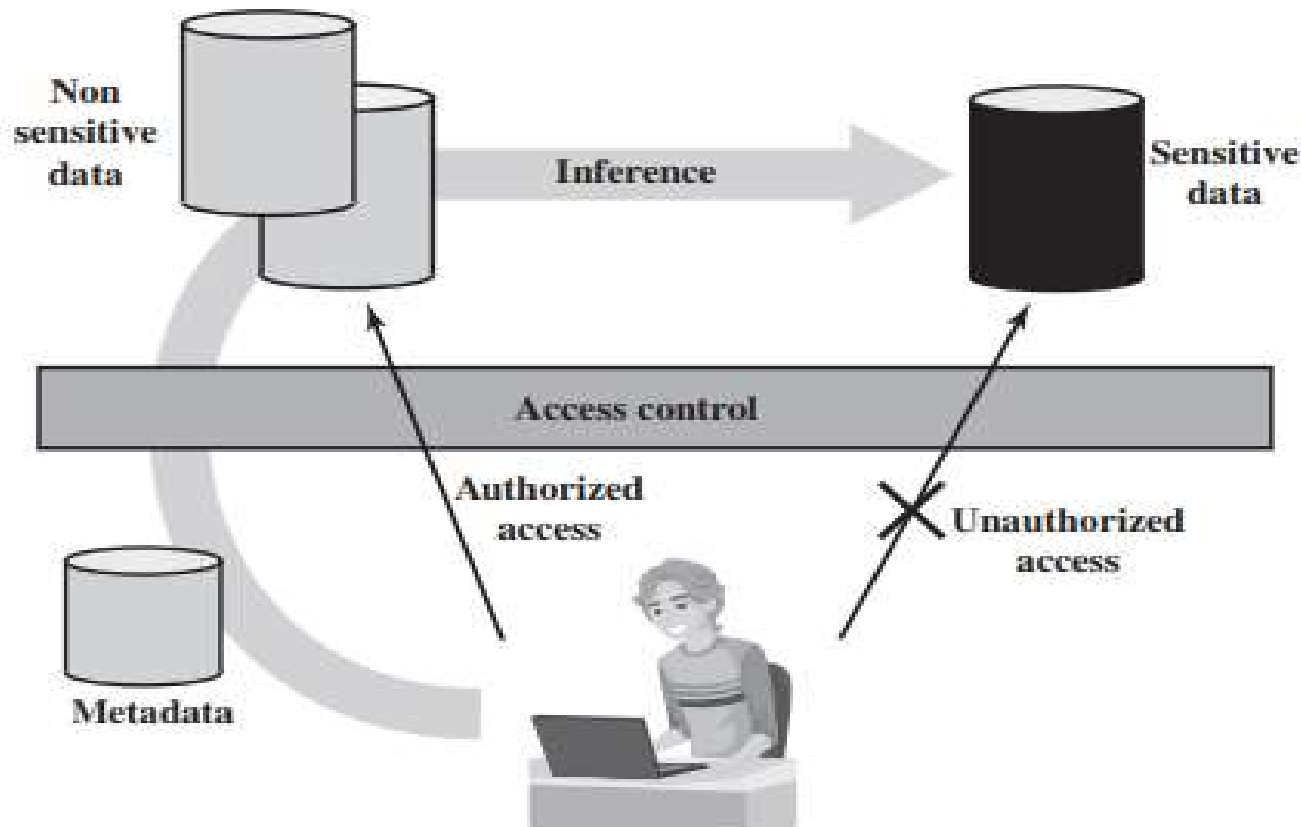
## 1.2. Role-Based Access Control (RBAC)

- Role-based access control work well for DBMS
  - eases admin burden, improves security
- Categories of database users:
  - application owner
  - end user
  - Administrator
- DB RBAC must manage roles and their users



# Inference

- The process of performing authorized queries and deducing unauthorized information from the legitimate responses received
- A combination of data items can be used to infer data of a higher sensitivity



# Inference Example

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

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

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

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

# Inference example

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

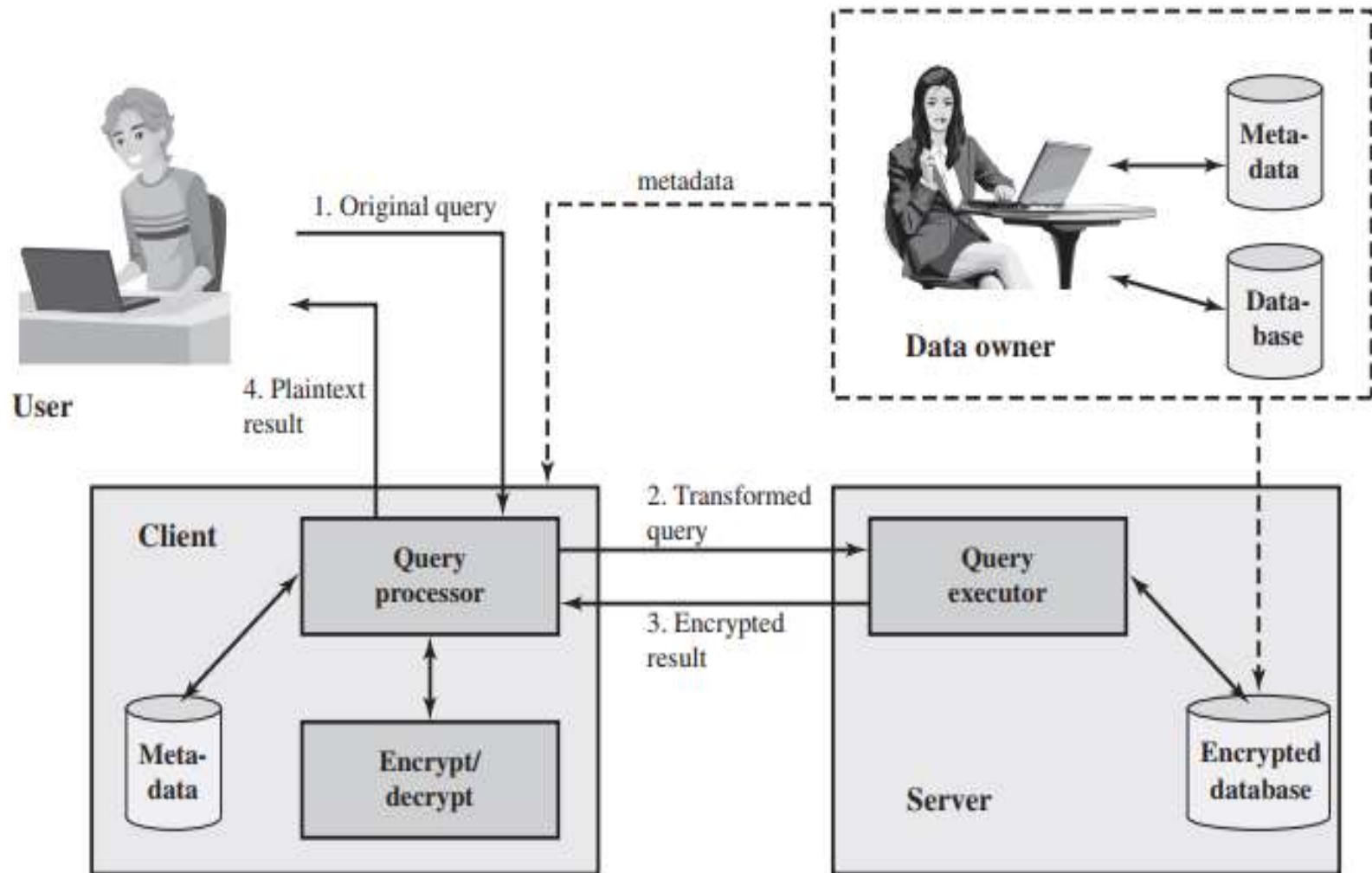
# Inference Countermeasures

- Inference detection at database design
  - alter database structure or access controls
  - Splitting a table into multiple tables
  - More fine grain access control roles
- Inference detection at query time
  - by monitoring and altering or rejecting queries

# Database Encryption

- Databases typically a valuable info resource
  - protected by multiple layers of security: firewalls, authentication, O/S access control systems, DB access control systems, and database encryption
- Can encrypt
  - entire database - very inflexible and inefficient
  - individual fields - simple but inflexible
  - records (rows) or columns (attributes) - best
    - also need attribute indexes to help data retrieval
- Varying trade-offs

# Database Encryption



# Secured Operations

- Suppose that each individual item in the database is encrypted separately.
- The encrypted database is stored at the server, but the server does not have the key, so that the data is secure at the server. 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.

# Example

```
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
```



# Issues

- This method is certainly straightforward but 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.

# Record level encryption and indexing

- Each record (row) of a table in the database is encrypted as a block which is treated as a contiguous block.
- 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 in the original database, there is one row in the encrypted database.
- 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.

# Example

- 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.

# 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