Chương 3:
THIẾT KẾ CƠ SỞ DỮ LIỆU

Nội dung

1. Qui trình phát triển cơ sở dữ liệu
2. Phương pháp thiết kế cơ sở dữ liệu
3. Các kiến trúc xây dựng cơ sở dữ liệu
4. Giới thiệu cơ sở dữ liệu hướng đối tượng

Nội dung

5 Giới thiệu cơ sở dữ liệu XML

6 Giới thiệu cơ sở dữ liệu không gian

QUI TRÌNH PHÁT TRIỂN CƠ SỞ DỮ LIỆU


Database planning
(Section 9.3)
System definition
(Section 9.4)
Requirements collection
and analysis
(Section 9.5)
Database design
(Section 9.6)
Conceptual database
design
Logical database
design
Physical database
design
DBMS selection (optional)
(Section 9.7)
Application design
(Section 9.8)
Prototyping (optional)
(Section 9.9)
Implementation
(Section 9.10)
Data conversion
and loading
(Section 9.11)
Testing
(Section 9.12)
Operational maintenance
(Section 9.13)

Stage	Main activities
Database planning	Planning how the stages of the lifecycle can be realized most efficiently and effectively.
System definition	Specifying the scope and boundaries of the database system, including the major user views, its users, and application areas.
Requirements collection and analysis	Collection and analysis of the requirements for the new database system.
Database design	Conceptual, logical, and physical design of the database.
DBMS selection (optional)	Selecting a suitable DBMS for the database system.
Application design	Designing the user interface and the application programs that use and process the database.
Prototyping (optional)	Building a working model of the database system, which allows the designers or users to visualize and evaluate how the final system will look and function.
Implementation	Creating the physical database definitions and the application programs.
Data conversion and loading	Loading data from the old system to the new system and, where possible, converting any existing applications to run on the new database.
Testing	Database system is tested for errors and validated against the requirements specified by the users.
Operational maintenance	Database system is fully implemented. The system is continuously monitored and maintained. When necessary, new requirements are incorporated into the database system through the preceding stages of the lifecycle.

PHƯƠNG PHÁP THIẾT KẾ CƠ SỞ DỮ LIỆU

Conceptual

- The process of constructing a model of the data used in an enterprise, independent of all physical considerations.
- Build conceptual data model
  - Step 1.1 Identify entity types
  - Step 1.2 Identify relationship types
  - Step 1.3 Identify and associate attributes with entity or relationship types
  - Step 1.4 Determine attribute domains
  - Step 1.5 Determine candidate, primary, and alternate key attributes
  - Step 1.6 Consider use of enhanced modeling concepts (optional step)
  - Step 1.7 Check model for redundancy
  - Step 1.8 Validate conceptual model against user transactions
  - Step 1.9 Review conceptual data model with user

Logical

- The process of constructing a model of the data used in an enterprise based on a specific data model, but independent of a particular DBMS design and other physical considerations.
- Build and validate logical data model
  - Step 2.1 Derive relations for logical data model
  - Step 2.2 Validate relations using normalization
  - Step 2.3 Validate relations against user transactions
  - Step 2.4 Check integrity constraints
  - Step 2.5 Review logical data model with user
  - Step 2.6 Merge logical data models into global model (optional step)
  - Step 2.7 Check for future growth

Physical

- The process of producing a description of the implementation of the data base on secondary storage; it describes the base relations, file design organizations, and indexes used to achieve efficient access to the data, and any associated integrity constraints and security measures.
  - Step 3 Translate logical data model for target DBMS
  - Step 3.1 Design base relations
  - Step 3.2 Design representation of derived data
  - Step 3.3 Design general constraints
  - Step 4 Design file organizations and indexes
  - Step 4.1 Analyze transactions
  - Step 4.2 Choose file organizations
  - Step 4.3 Choose indexes
  - Step 4.4 Estimate disk space requirements
  - Step 5 Design user views
  - Step 6 Design security mechanisms
  - Step 7 Consider the introduction of controlled redundancy
  - Step 8 Monitor and tune the operational system

CÁC KIẾN TRÚC XÂY DỰNG CƠ SỞ DỮ LIỆU

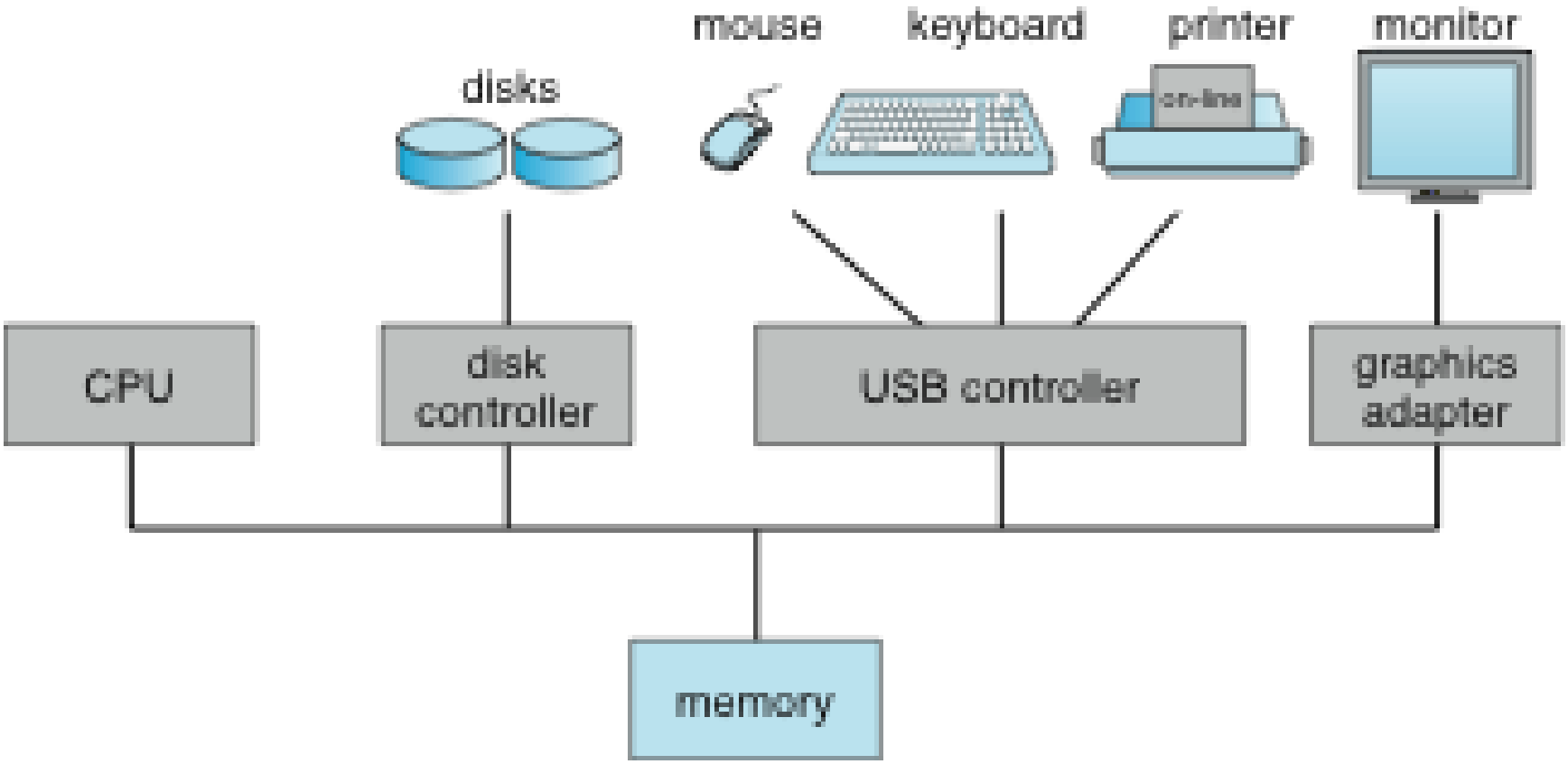
Database System Architectures

- Centralized and Client-Server Systems
- Server System Architectures
- Parallel Systems
- Distributed Systems
- Network Types

Centralized Systems

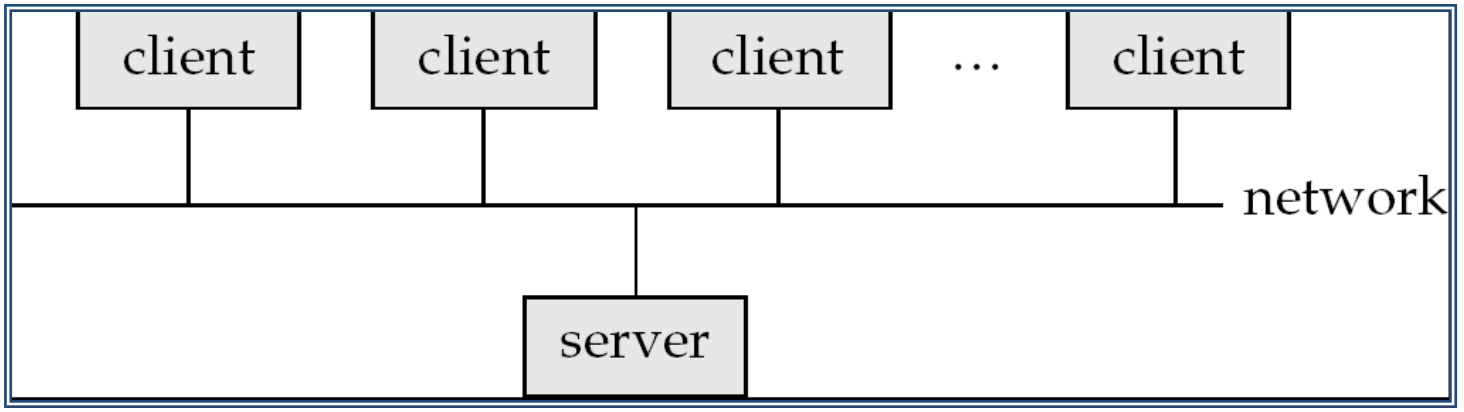
- Run on a single computer system and do not interact with other computer systems.
- General-purpose computer system: one to a few CPUs and a number of device controllers that are connected through a common bus that provides access to shared memory.
- Single-user system (e.g., personal computer or workstation): desk-top unit, single user, usually has only one CPU and one or two hard disks; the OS may support only one user.
- Multi-user system: more disks, more memory, multiple CPUs, and a multi-user OS. Serve a large number of users who are connected to the system vie terminals. Often called *server* systems.

A Centralized Computer System



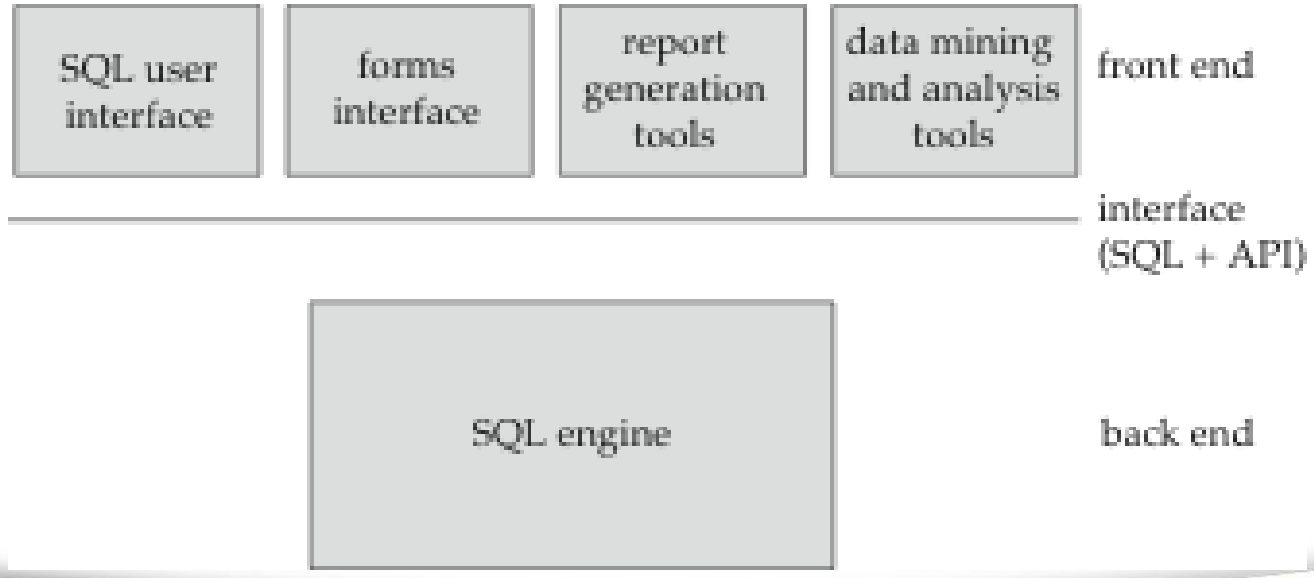
Client-Server Systems

- Server systems satisfy requests generated at *m* client systems, whose general structure is shown below:



Client-Server Systems (Cont.)

- Database functionality can be divided into:
  - **Back-end**: manages access structures, query evaluation and optimization, concurrency control and recovery.
  - **Front-end**: consists of tools such as *forms*, *report-writers*, and graphical user interface facilities.
- The interface between the front-end and the back-end is through SQL or through an application program interface.



Client-Server Systems (Cont.)

- Advantages of replacing mainframes with networks of workstations or personal computers connected to back-end server machines:
  - better functionality for the cost
  - flexibility in locating resources and expanding facilities
  - better user interfaces
  - easier maintenance

Server System Architecture

- Server systems can be broadly categorized into two kinds:
  - **transaction servers** which are widely used in relational database systems, and
  - **data servers**, used in object-oriented database systems

Transaction Servers

- Also called **query server** systems or SQL *server* systems
  - Clients send requests to the server
  - Transactions are executed at the server
  - Results are shipped back to the client.
- Requests are specified in SQL, and communicated to the server through a *remote procedure call* (RPC) mechanism.
- Transactional RPC allows many RPC calls to form a transaction.
- *Open Database Connectivity* (ODBC) is a C language application program interface standard from Microsoft for connecting to a server, sending SQL requests, and receiving results.
- JDBC standard is similar to ODBC, for Java

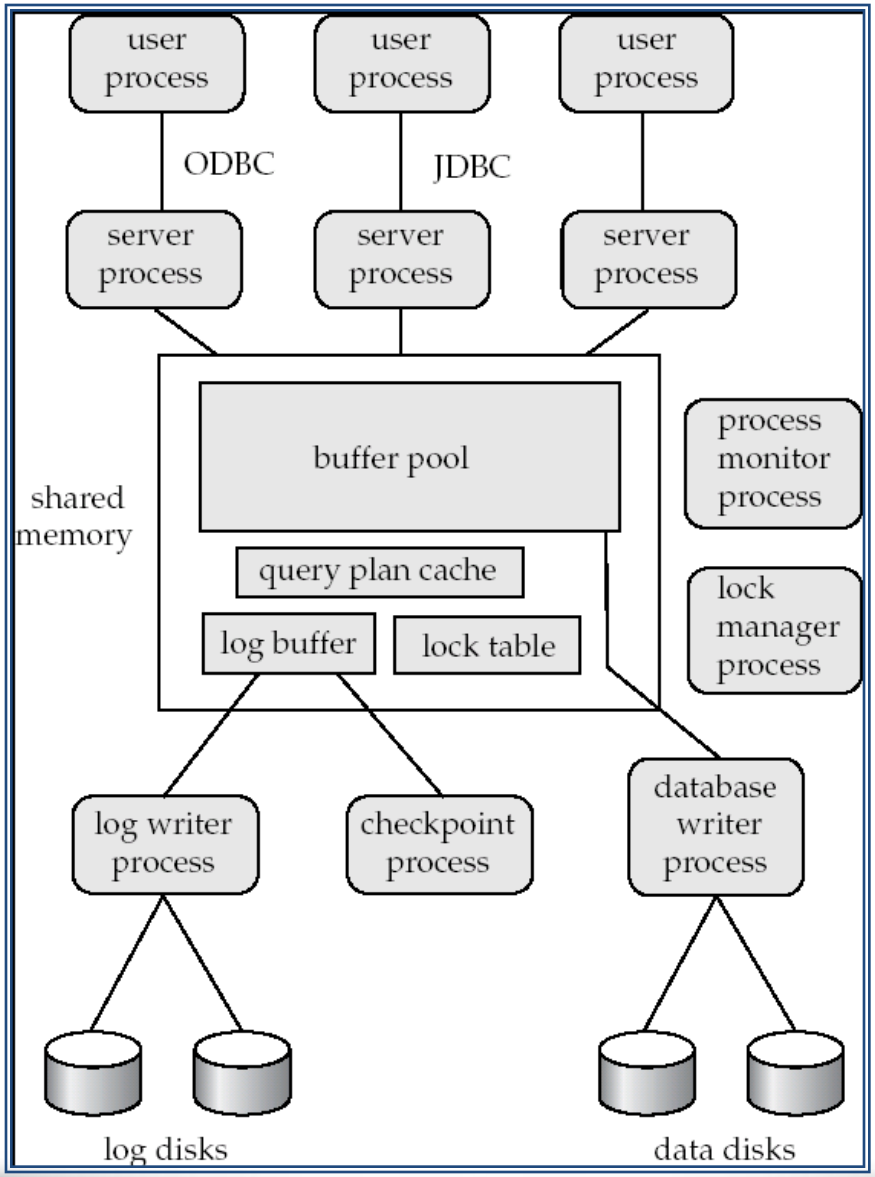
Transaction Server Process Structure

- A typical transaction server consists of multiple processes accessing data in shared memory.
- Server processes
  - These receive user queries (transactions), execute them and send results back
  - Processes may be **multithreaded**, allowing a single process to execute several user queries concurrently
  - Typically multiple multithreaded server processes
- Lock manager process
  - More on this later
- Database writer process
  - Output modified buffer blocks to disks continually

Transaction Server Processes (Cont.)

- Log writer process
  - Server processes simply add log records to log record buffer
  - Log writer process outputs log records to stable storage.
- Checkpoint process
  - Performs periodic checkpoints
- Process monitor process
  - Monitors other processes, and takes recovery actions if any of the other processes fail
    - E.g. aborting any transactions being executed by a server process and restarting it

Transaction System Processes (Cont.)



Transaction System Processes (Cont.)

- Shared memory contains shared data
  - Buffer pool
  - Lock table
  - Log buffer
  - Cached query plans (reused if same query submitted again)
- All database processes can access shared memory
- To ensure that no two processes are accessing the same data structure at the same time, databases systems implement **mutual exclusion** using either
  - Operating system semaphores
  - Atomic instructions such as test-and-set
- To avoid overhead of interprocess communication for lock request/grant, each database process operates directly on the lock table
  - instead of sending requests to lock manager process
- Lock manager process still used for deadlock detection

Data Servers

- Used in high-speed LANs, in cases where
  - The clients are comparable in processing power to the server
  - The tasks to be executed are compute intensive.
- Data are shipped to clients where processing is performed, and then shipped results back to the server.
- This architecture requires full back-end functionality at the clients.
- Used in many object-oriented database systems
- Issues:
  - Page-Shipping versus Item-Shipping
  - Locking
  - Data Caching
  - Lock Caching

Data Servers (Cont.)

- **Page-shipping** versus **item-shipping**
  - Smaller unit of shipping ⇒ more messages
  - Worth **prefetching** related items along with requested item
  - Page shipping can be thought of as a form of prefetching
- Locking
  - Overhead of requesting and getting locks from server is high due to message delays
  - Can grant locks on requested and prefetched items; with page shipping, transaction is granted lock on whole page.
  - Locks on a prefetched item can be P{called back} by the server, and returned by client transaction if the prefetched item has not been used.
  - Locks on the page can be **deescalated** to locks on items in the page when there are lock conflicts. Locks on unused items can then be returned to server.

Data Servers (Cont.)

- **Data Caching**
  - Data can be cached at client even in between transactions
  - But check that data is up-to-date before it is used (**cache coherency**)
  - Check can be done when requesting lock on data item
- **Lock Caching**
  - Locks can be retained by client system even in between transactions
  - Transactions can acquire cached locks locally, without contacting server
  - Server **calls back** locks from clients when it receives conflicting lock request. Client returns lock once no local transaction is using it.
  - Similar to deescalation, but across transactions.

Parallel Systems

- Parallel database systems consist of multiple processors and multiple disks connected by a fast interconnection network.
- A **coarse-grain parallel** machine consists of a small number of powerful processors
- A **massively parallel** or **fine grain parallel** machine utilizes thousands of smaller processors.
- Two main performance measures:
  - **throughput** --- the number of tasks that can be completed in a given time interval
  - **response time** --- the amount of time it takes to complete a single task from the time it is submitted

Speed-Up and Scale-Up

- **Speedup**: a fixed-sized problem executing on a small system is given to a system which is *N*-times larger.
  - Measured by:

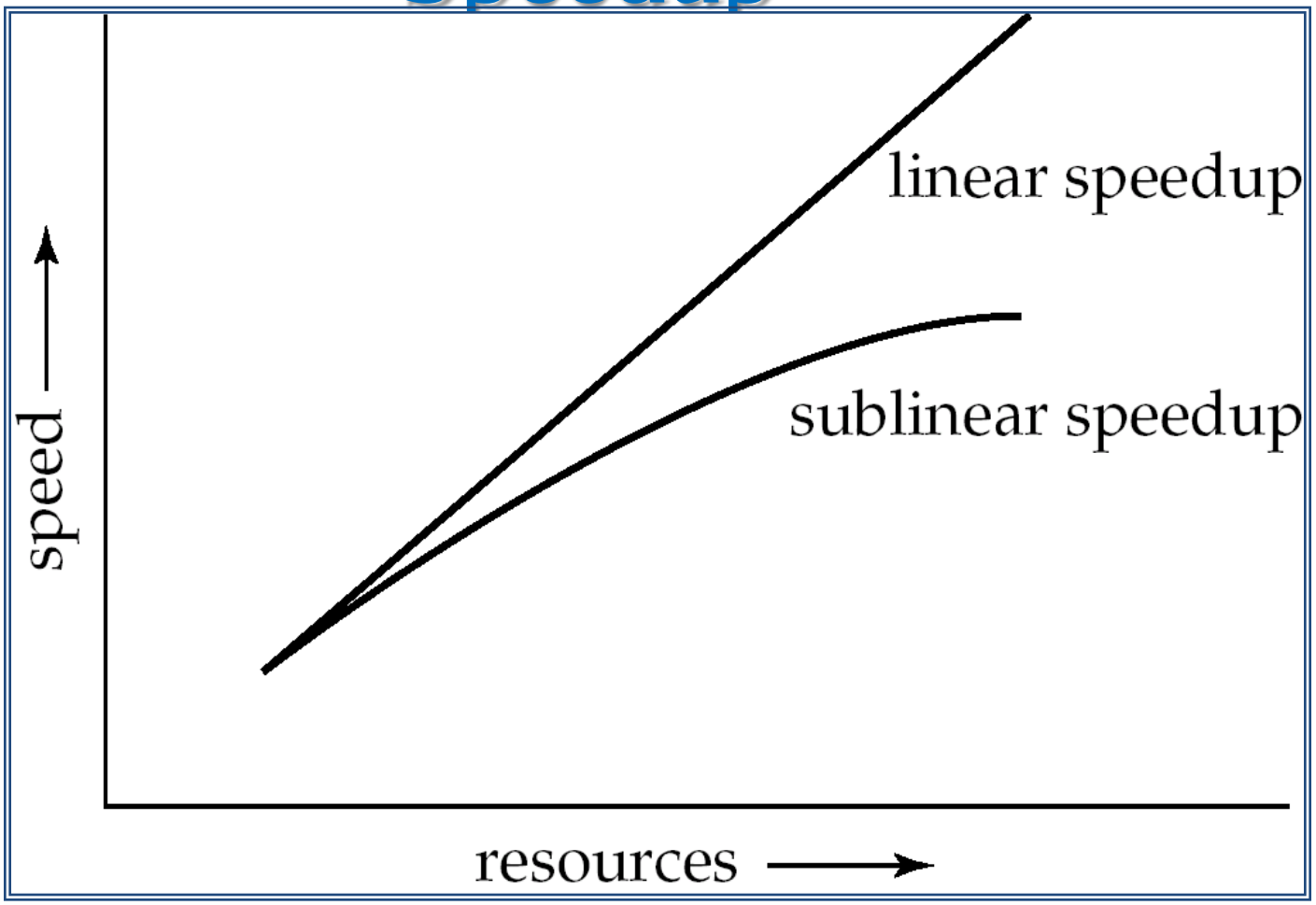
$$speedup = \frac{small\ system\ elapsed\ time}{large\ system\ elapsed\ time}$$

  - Speedup is **linear** if equation equals N.
- **Scaleup**: increase the size of both the problem and the system
  - *N*-times larger system used to perform *N*-times larger job
  - Measured by:

$$scaleup = \frac{small\ system\ small\ problem\ elapsed\ time}{big\ system\ big\ problem\ elapsed\ time}$$

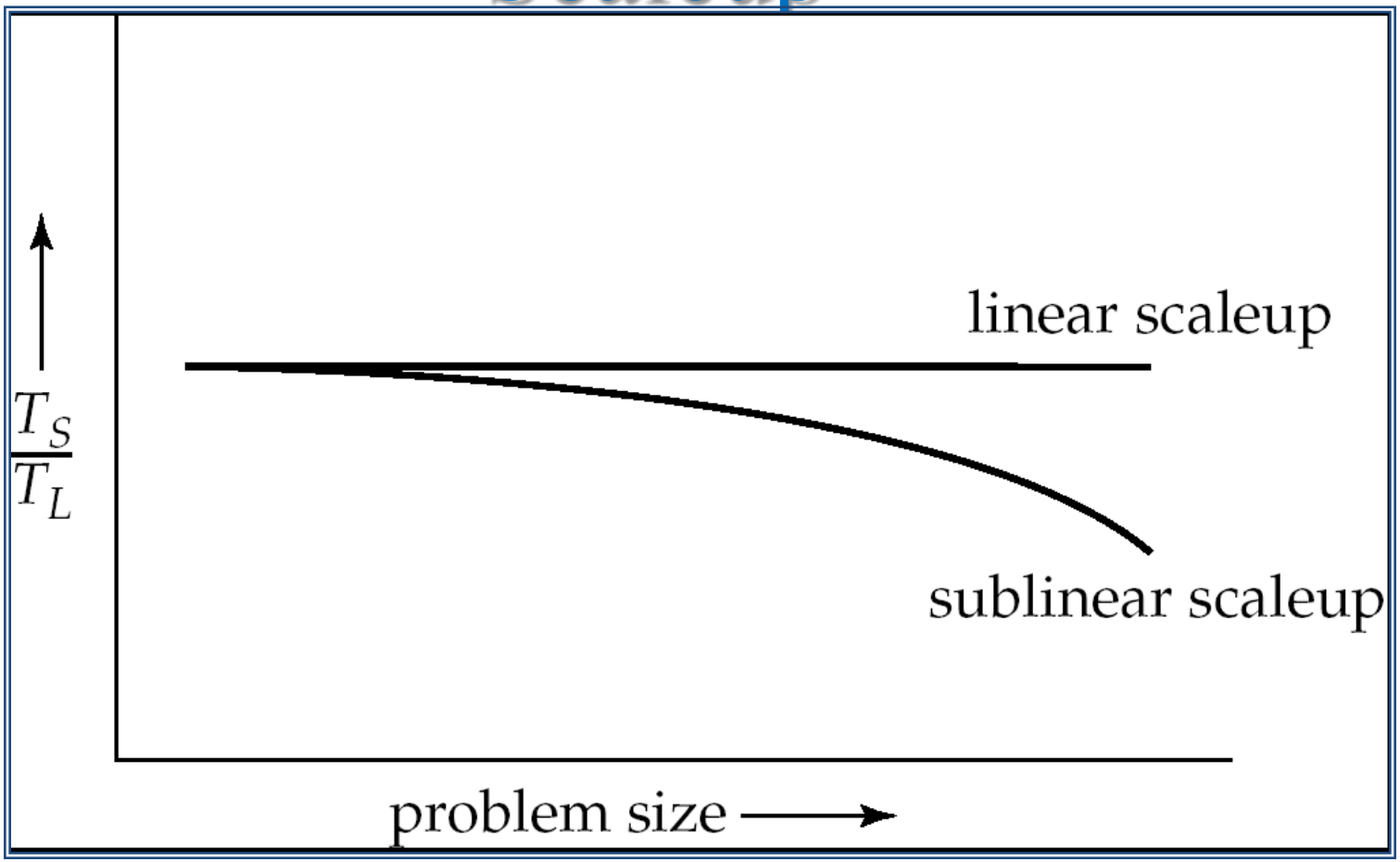
  - Scale up is **linear** if equation equals 1.

Speedup



Speedup

Scaleup



Scaleup

Batch and Transaction Scaleup

- **Batch scaleup:**
  - A single large job; typical of most decision support queries and scientific simulation.
  - Use an *N*-times larger computer on *N*-times larger problem.
- **Transaction scaleup**:
  - Numerous small queries submitted by independent users to a shared database; typical transaction processing and timesharing systems.
  - *N*-times as many users submitting requests (hence, *N*-times as many requests) to an *N*-times larger database, on an *N*-times larger computer.
  - Well-suited to parallel execution.

Factors Limiting Speedup and Scaleup

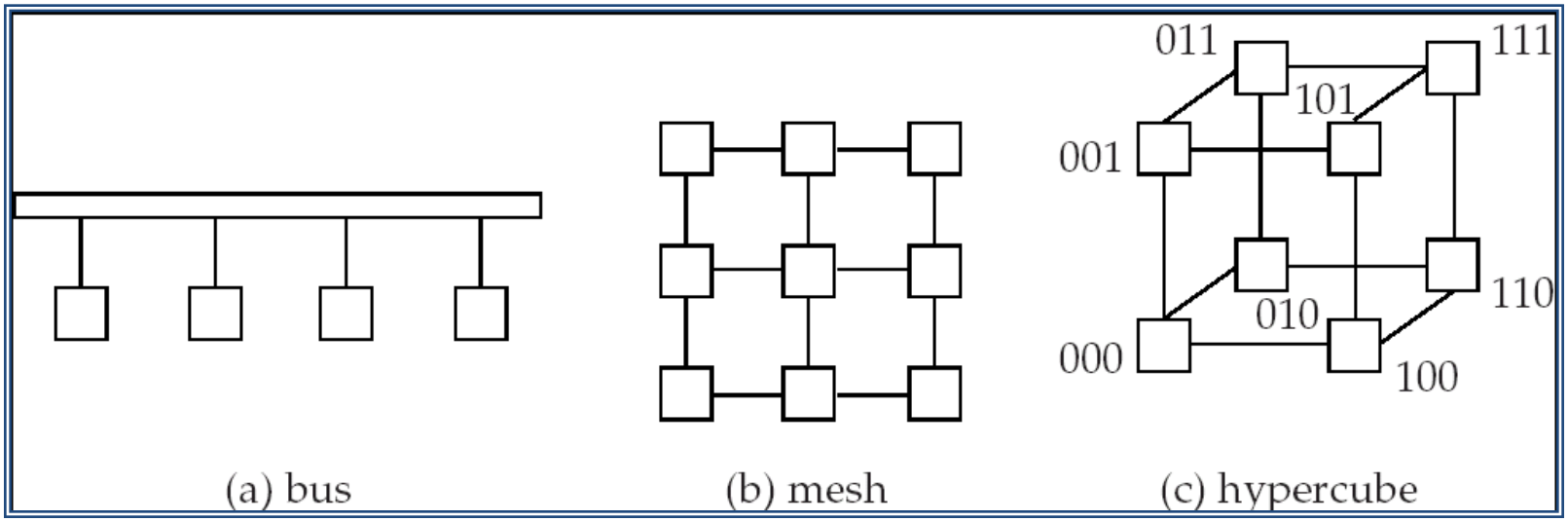
Speedup and scaleup are often sublinear due to:

- **Startup costs**: Cost of starting up multiple processes may dominate computation time, if the degree of parallelism is high.
- **Interference**: Processes accessing shared resources (e.g.,system bus, disks, or locks) compete with each other, thus spending time waiting on other processes, rather than performing useful work.
- **Skew**: Increasing the degree of parallelism increases the variance in service times of parallely executing tasks. Overall execution time determined by **slowest** of parallely executing tasks.

Interconnection Network Architectures

- **Bus**. System components send data on and receive data from a single communication bus;
  - Does not scale well with increasing parallelism.
- **Mesh**. Components are arranged as nodes in a grid, and each component is connected to all adjacent components
  - Communication links grow with growing number of components, and so scales better.
  - But may require $2\sqrt{n}$ hops to send message to a node (or $\sqrt{n}$ with wraparound connections at edge of grid).
- **Hypercube**. Components are numbered in binary; components are connected to one another if their binary representations differ in exactly one bit.
  - *n* components are connected to *log(n)* other components and can reach each other via at most *log(n)* links; reduces communication delays.

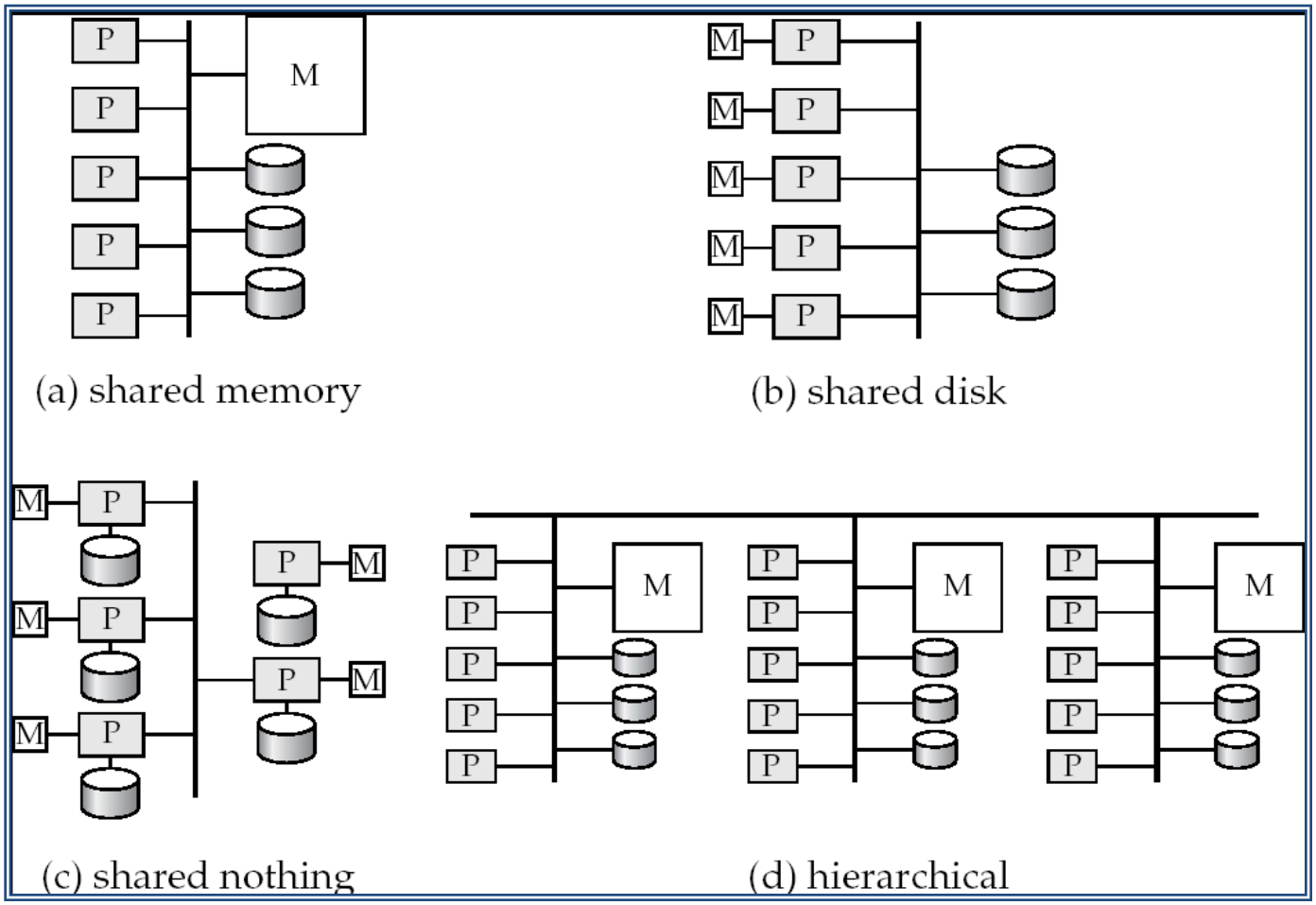
Interconnection Architectures



Parallel Database Architectures

- **Shared memory --** processors share a common memory
- **Shared disk --** processors share a common disk
- **Shared nothing --** processors share neither a common memory nor common disk
- **Hierarchical --** hybrid of the above architectures

Parallel Database Architectures



Shared Memory

- Processors and disks have access to a common memory, typically via a bus or through an interconnection network.
- Extremely efficient communication between processors — data in shared memory can be accessed by any processor without having to move it using software.
- Downside – architecture is not scalable beyond 32 or 64 processors since the bus or the interconnection network becomes a bottleneck
- Widely used for lower degrees of parallelism (4 to 8).

Shared Disk

- All processors can directly access all disks via an interconnection network, but the processors have private memories.
  - The memory bus is not a bottleneck
  - Architecture provides a degree of **fault-tolerance** — if a processor fails, the other processors can take over its tasks since the database is resident on disks that are accessible from all processors.
- Examples: IBM Sysplex and DEC clusters (now part of Compaq) running Rdb (now Oracle Rdb) were early commercial users
- Downside: bottleneck now occurs at interconnection to the disk subsystem.
- Shared-disk systems can scale to a somewhat larger number of processors, but communication between processors is slower.

Shared Nothing

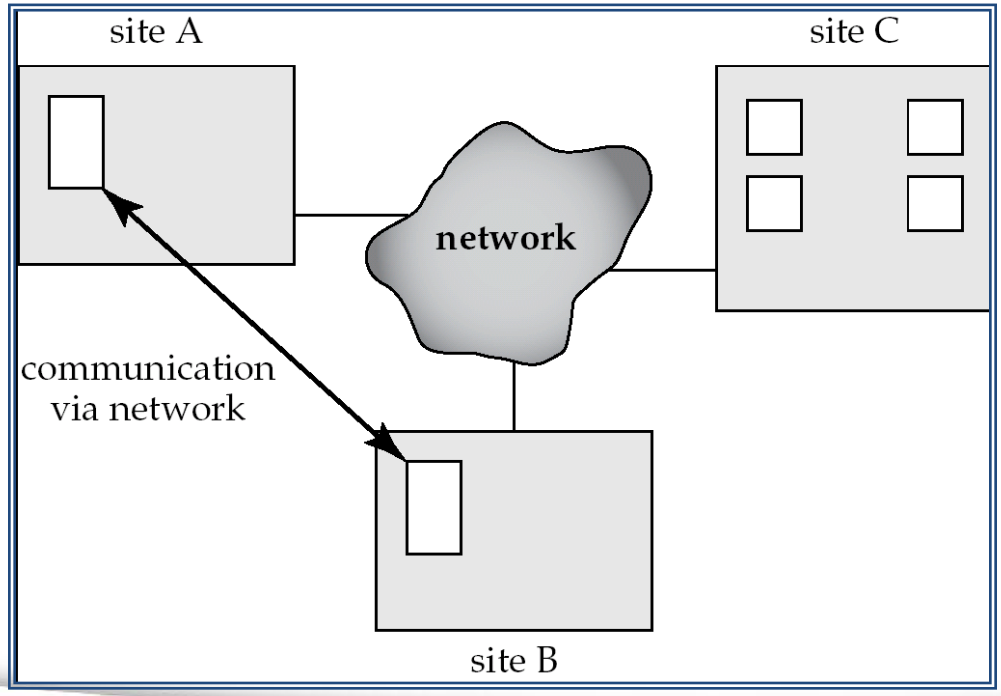
- Node consists of a processor, memory, and one or more disks. Processors at one node communicate with another processor at another node using an interconnection network. A node functions as the server for the data on the disk or disks the node owns.
- Examples: Teradata, Tandem, Oracle-n CUBE
- Data accessed from local disks (and local memory accesses) do not pass through interconnection network, thereby minimizing the interference of resource sharing.
- Shared-nothing multiprocessors can be scaled up to thousands of processors without interference.
- Main drawback: cost of communication and non-local disk access; sending data involves software interaction at both ends.

Hierarchical

- Combines characteristics of shared-memory, shared-disk, and shared-nothing architectures.
- Top level is a shared-nothing architecture – nodes connected by an interconnection network, and do not share disks or memory with each other.
- Each node of the system could be a shared-memory system with a few processors.
- Alternatively, each node could be a shared-disk system, and each of the systems sharing a set of disks could be a shared-memory system.
- Reduce the complexity of programming such systems by **distributed virtual-memory** architectures
  - Also called **non-uniform memory architecture (NUMA)**

Distributed Systems

- Data spread over multiple machines (also referred to as **sites** or **nodes**).
- Network interconnects the machines
- Data shared by users on multiple machines



Distributed Databases

- Homogeneous distributed databases
  - Same software/schema on all sites, data may be partitioned among sites
  - Goal: provide a view of a single database, hiding details of distribution
- Heterogeneous distributed databases
  - Different software/schema on different sites
  - Goal: integrate existing databases to provide useful functionality
- Differentiate between *local* and *global* transactions
  - A local transaction accesses data in the *single* site at which the transaction was initiated.
  - A global transaction either accesses data in a site different from the one at which the transaction was initiated or accesses data in several different sites.

Trade-offs in Distributed Systems

- Sharing data – users at one site able to access the data residing at some other sites.
- Autonomy – each site is able to retain a degree of control over data stored locally.
- Higher system availability through redundancy — data can be replicated at remote sites, and system can function even if a site fails.
- Disadvantage: added complexity required to ensure proper coordination among sites.
  - Software development cost.
  - Greater potential for bugs.
  - Increased processing overhead.

Implementation Issues for Distributed Databases

- Atomicity needed even for transactions that update data at multiple sites
- The two-phase commit protocol (2PC) is used to ensure atomicity
  - Basic idea: each site executes transaction until just before commit, and the leaves final decision to a coordinator
  - Each site must follow decision of coordinator, even if there is a failure while waiting for coordinators decision
- 2PC is not always appropriate: other transaction models based on persistent messaging, and workflows, are also used
- Distributed concurrency control (and deadlock detection) required
- Data items may be replicated to improve data availability
- Details of above in Chapter 22

Network Types

- **Local-area networks (**LANs) – composed of processors that are distributed over small geographical areas, such as a single building or a few adjacent buildings.
- **Wide-area networks (**WANs) – composed of processors distributed over a large geographical area.

Networks Types (Cont.)

- WANs with continuous connection (e.g. the Internet) are needed for implementing distributed database systems
- Groupware applications such as Lotus notes can work on WANs with discontinuous connection:
  - Data is replicated.
  - Updates are propagated to replicas periodically.
  - Copies of data may be updated independently.
  - Non-serializable executions can thus result. Resolution is application dependent.

GIỚI THIỆU CƠ SỞ DỮ LIỆU HƯỚNG ĐỐI TƯỢNG

Nhược điểm của cơ sở dữ liệu quan hệ

Weakness
Poor representation of 'real world' entities
Semantic overloading
Poor support for integrity and enterprise constraints
Homogeneous data structure
Limited operations
Difficulty handling recursive queries
Impedance mismatch
Other problems with RDBMSs associated with concurrency, schema changes, and poor navigational access

GIỚI THIỆU CƠ SỞ DỮ LIỆU HƯỚNG ĐỐI TƯỢNG

CÁC KHÁI NIỆM TRONG CSDL HƯỚNG ĐỐI TƯỢNG

Structured Types

- User-defined row types

  create type *CustomerType* **as** (

  name Name,

  address Address,

  dateOfBirth **date**)

  not final

- Can then create a table whose rows are a user-defined type

  create table *customer* **of** *CustomerType*

Methods

- Can add a method declaration with a structured type.

  method *ageOnDate* (*onDate* **date**)
  returns interval year

- Method body is given separately.

  create instance method *ageOnDate* (*onDate* **date**)
  returns interval year
  for *CustomerType*
  begin
  return *onDate* - **self**.*dateOfBirth*;
  end

- We can now find the age of each customer:

  select *name.lastname, ageOnDate* (**current_date**)
  from *customer*

Inheritance

- Suppose that we have the following type definition for people:

```
create type Person
    (name varchar(20),
     address varchar(20))
```

- Using inheritance to define the student and teacher types

```
create type Student
  under Person
  (degree       varchar(20),
   department   varchar(20))
create type Teacher
  under Person
  (salary       integer,
   department   varchar(20))
```

- Subtypes can redefine methods by using **overriding method** in place of **method** in the method declaration

Array and Multiset Types in SQL

- Example of array and multiset declaration:

```
create type Publisher as
   (name          varchar(20),
    branch        varchar(20))
create type Book as
   (title         varchar(20),
    author-array  varchar(20) array [10],
    pub-date      date,
    publisher     Publisher,
    keyword-set   varchar(20) multiset )
create table books of Book
```

- Similar to the nested relation books, but with array of authors instead of set

Object-Identity and Reference Types

- Define a type *Department* with a field *name* and a field *head* which is a reference to the type *Person*, with table *people* as scope:

  create type *Department* (
  name **varchar** (20),
  head **ref** (*Person*) **scope** *people*)

- We can then create a table *departments* as follows

  create table *departments* **of** *Department*

- We can omit the declaration **scope** people from the type declaration and instead make an addition to the **create table** statement:

  create table *departments* **of** *Department*
  (*head* **with options scope** *people*)

Nesting

- **Nesting** is the opposite of unnesting, creating a collection-valued attribute
- NOTE: SQL:1999 does not support nesting
- Nesting can be done in a manner similar to aggregation, but using the function **colect**() in place of an aggregation operation, to create a multiset
- To nest the *flat-books* relation on the attribute *keyword*:

  select *title*, *author*, *Publisher* (*pub_name*, *pub_branch*) **as** *publisher*,
  collect (*keyword*) **as** *keyword_set*
  from *flat-books*
  groupby *title*, *author*, *publisher*
- To nest on both authors and keywords:

  select *title*, **collect** (*author*) **as** *author_set*,
  Publisher (*pub_name*, *pub_branch*) **as** *publisher*,
  collect (*keyword*) **as** *keyword_set*
  from *flat-books*
  group by *title*, *publisher*

GIỚI THIỆU CƠ SỞ DỮ LIỆU XML

Phân loại cơ sở dữ liệu XML

- XML-Enabled Databases
- Native XML Databases
- Hybrid XML databases

XML-Enabled Databases

- Define a (logical) model for an XML document | as opposed to the data in that document { and stores and retrieves documents according to that model. At a minimum, the model must include elements, attributes, PCDATA, and document order. Examples of such models are the XPath data model, the XML Infoset, and the models implied by the DOM and the events in SAX 1.0.
- Has an XML document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage.
- Is not required to have any particular underlying physical storage model. For example, it can be built on a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed, compressed files.

Relational-to-XML schema mapping.

- table-based mapping
- object-relational mapping

Native XML Databases

- A database that has an added XML mapping layer

provided either by the database vendor or a third party. This mapping layer manages the storage and retrieval of XML data. Data that is mapped into the database is mapped into application specific formats and the original XML meta-data and structure may be lost. Data retrieved as XML is NOT guaranteed to have originated in XML form. Data manipulation may occur via either XML specicific technologies (e.g. XPath, XSLT, DOM or SAX) or other database technologies (e.g. SQL). The fundamental unit of storage in an XML Enabled Database is implementation dependent.

Element	Node Type
Flights	Root
Flight	First node
Airline	Second node
Airplane	Second node
Origin	Second node
Destination	Second node
Departure	Second node
Arrival	Second node

Attribute name
AirlineID
CityID

Text	Text Type
American Airlines	Character
Delta Airlines	Character
DAL	Character
CHT	Character
RDU	Character
JFK	Character
Boeing	Character
Airbus	Character
9:15	Time
11:19	Time
12:15	Time
13:46	Time
15:30	Time
19:50	Time

XML-enabled Database vs Native XML Database

XML-enabled Database

- Maps XML schema to database schema
- Data is transferred using this mapping
- It has its own data model to map XML instances into database
- Creates tables like Airlines, Airports, Airplanes, etc.

Native XML Database (NXD)

- NXD has fixed (native) schema
- Uses XML model directly to map XML instances into database
- Uses tables for holding arbitrary XML document like Elements, Attributes, Text, etc.

Hybrid XML databases

- A database that can be treated as either a Native XML Database or as an XML Enabled Database depending on the requirements of the application.

Một số hệ quản trị cơ sở dữ liệu hỗ trợ quản lý hybrid XML

- Oracle XML DB
- IBM DB2 XML Extender
- Microsoft SQL Server 2005 ->

GIỚI THIỆU CƠ SỞ DỮ LIỆU KHÔNG GIAN

Spatial Databases Background

- Spatial databases provide structures for storage and analysis of spatial data
- Spatial data is comprised of objects in multi-dimensional space
- Storing spatial data in a standard database would require excessive amounts of space
- Queries to retrieve and analyze spatial data from a standard database would be long and cumbersome leaving a lot of room for error
- Spatial databases provide much more efficient storage, retrieval, and analysis of spatial data

Types of Data Stored in Spatial Databases

- Two-dimensional data examples
  - Geographical
  - Cartesian coordinates (2-D)
  - Networks
  - Direction
- Three-dimensional data examples
  - Weather
  - Cartesian coordinates (3-D)
  - Topological
  - Satellite images

Spatial Databases Uses and Users

- Three types of uses
  - Manage spatial data
  - Analyze spatial data
  - High level utilization
- A few examples of users
  - Transportation agency tracking projects
  - Insurance risk manager considering location risk profiles
  - Doctor comparing Magnetic Resonance Images (MRIs)
  - Emergency response determining quickest route to victim
  - Mobile phone companies tracking phone usage

Spatial Databases Uses and Users

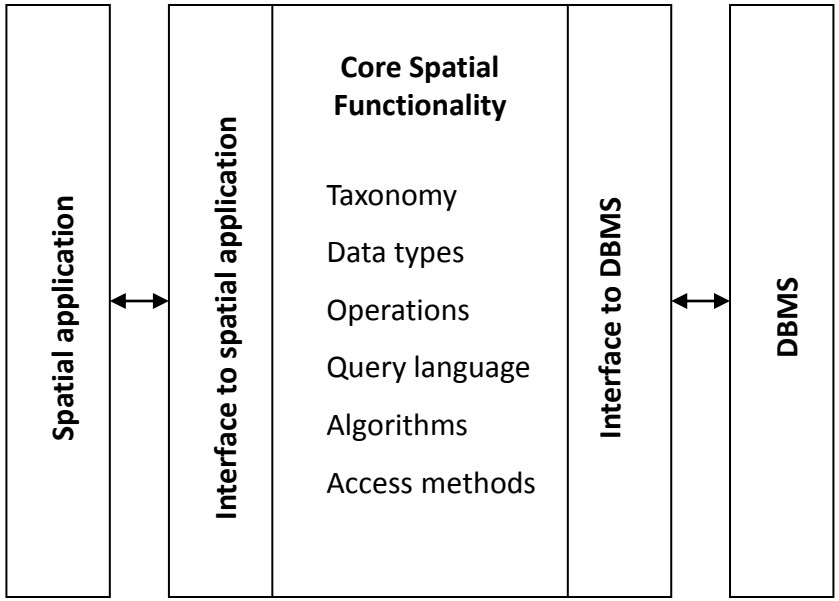
- Three types of uses
  - Manage spatial data
  - Analyze spatial data
  - High level utilization
- A few examples of users
  - Transportation agency tracking projects
  - Insurance risk manager considering location risk profiles
  - Doctor comparing Magnetic Resonance Images (MRIs)
  - Emergency response determining quickest route to victim
  - Mobile phone user determining current relative location of businesses

Spatial Database Management System

- Spatial Database Management System (SDBMS) provides the capabilities of a traditional database management system (DBMS) while allowing special storage and handling of spatial data.
- SDBMS:
  - Works with an underlying DBMS
  - Allows spatial data models and types
  - Supports querying language specific to spatial data types
  - Provides handling of spatial data and operations

SDBMS Three-layer Structure

- SDBMS works with a spatial application at the front end and a DBMS at the back end
- SDBMS has three layers:
  - Interface to spatial application
  - Core spatial functionality
  - Interface to DBMS



Spatial Query Language

- Number of specialized adaptations of SQL
  - Spatial query language
  - Temporal query language (TSQL2)
  - Object query language (OQL)
  - Object oriented structured query language (O_2SQL)
- Spatial query language provides tools and structures specifically for working with spatial data
- SQL3 provides 2D geospatial types and functions

References

- Database Systems ,A Practical Approach to Design, Implementation, and Management 4th Edition, 2004, homas M.Connolly, Carolyn E.Begg.
- Database Systems Design Implementation and Management 9th Edition, 2009, Carlos Coronel, Steven Morris, Peter Rod.
- Database System Concepts 5^{th} , 2005, Silberschatz, Korth. Sudarshan.
- XML Database, George Papamarkos, Lucas Zamboulis, Alexandra Poulovassilis, School of Computer Science and Information Systems, Birkbeck College, University of London


ThankYou
Merci
Salamat
Kop
Tack
Hvala
Xie
Gra
kun
Diolch
Kiitos
Sheun
umesc
Kasih
Mamnoon
Todah
Shnorhakalutiun
Shokriya
Spaas
Mul
Ačі
Gamsahapnida
Ngiyabonga
Dziękuję
Shokrun
Dank
Takk
Cam
or
Dhanyavaad
Dakujem
Waad
Daw
Dhanyavadaalu
Dankie
Go
Grazie
Falemiderit
Dhanyavad
Khopjai
Kruthagnathalu
krap
Arigatou
Or
Dhonnobaad
ederim
Asante
Hain
Dhan
daa
Grazzi
raibh
Gracias
Nandree
Blagodariya
Gomapsupnida
Fyrir
Terima
Enkosi
Danke
dank
Euxaristo
Kun
Shukriya