

# Chương 1

# Giới thiệu

# Hệ quản trị cơ sở dữ liệu



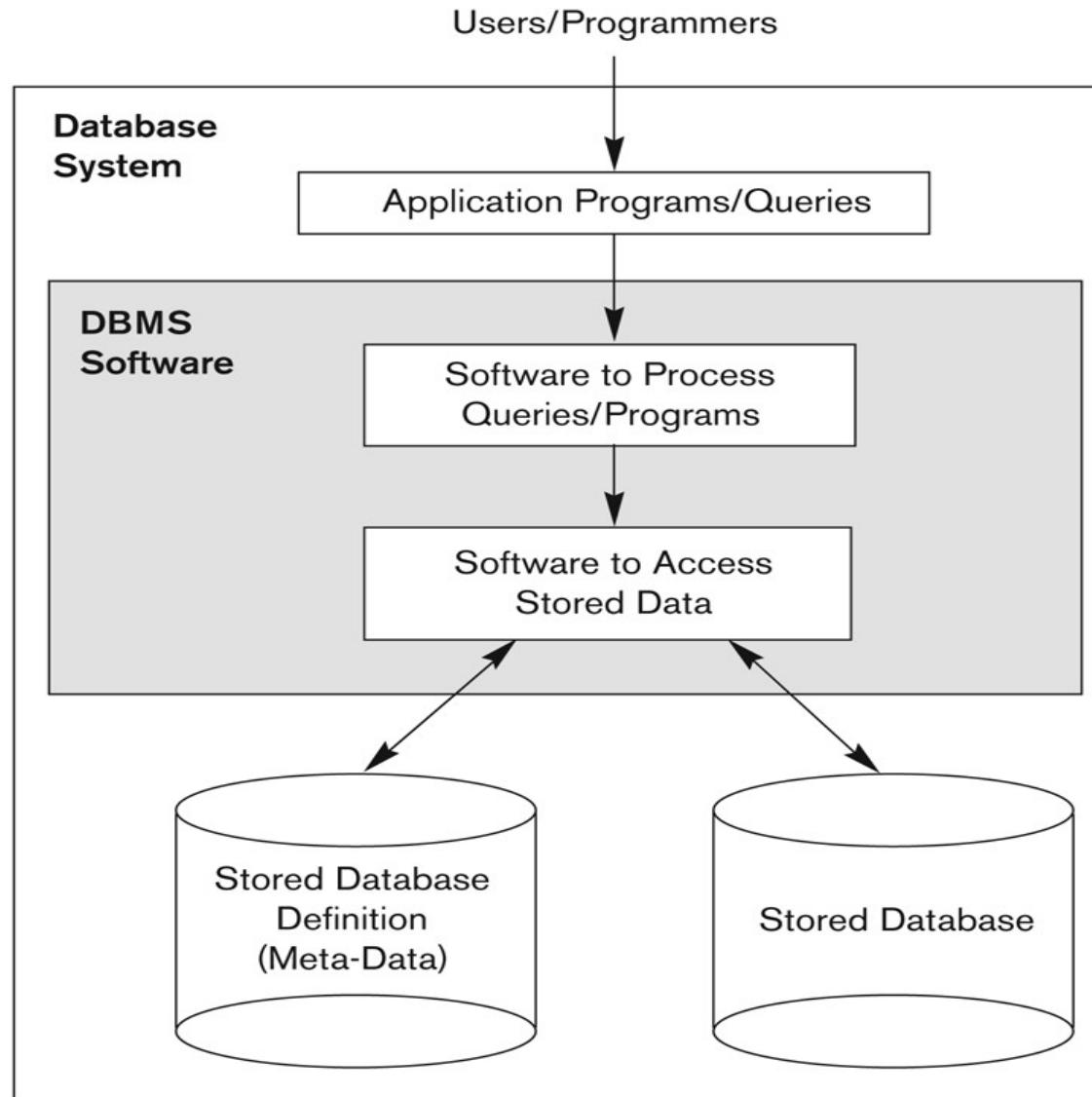
Môn: QUẢN TRỊ CƠ SỞ DỮ LIỆU



# Nội dung

- Môi trường hệ thống cơ sở dữ liệu
- Chức năng của hệ quản trị CSDL
- Kiến trúc của hệ quản trị CSDL
- Những người tham gia trong một hệ thống CSDL
- Nhiệm vụ của người quản trị CSDL (DBA)

# Môi trường hệ thống CSDL



**Figure 1.1**  
A simplified database system environment.

# Cách tiếp cận CSDL

- Hệ quản trị cơ sở dữ liệu  
(Database Management System – DBMS)  
Là một hệ thống phần mềm cho phép người dùng có thể định nghĩa, tạo, bảo trì và quản lý sự truy cập đến cơ sở dữ liệu
- Chương trình ứng dụng cơ sở dữ liệu  
(Database application program)  
Là các chương trình tương tác với CSDL thông qua DBMS bằng cách gửi các yêu cầu thích hợp qua câu lệnh SQL

# Các chức năng của DBMS

- Lưu trữ dữ liệu, sửa chữa và truy xuất
- Danh sách người dùng được phép truy cập
- Hỗ trợ các giao dịch
- Dịch vụ kiểm soát sự truy cập đồng thời
- Dịch vụ phục hồi khi có sự cố
- Dịch vụ xác thực người dùng
- Hỗ trợ giao tiếp dữ liệu
- Dịch vụ đảm bảo toàn vẹn dữ liệu
- Dịch vụ đảm bảo độc lập dữ liệu
- Các dịch vụ tiện ích khác

# Các đặc điểm của DBMS

- Tự mô tả (self-describing)
  - DBMS catalog lưu trữ tất cả các mô tả đặc thù của một CSDL (cấu trúc dữ liệu, kiểu, ràng buộc)
  - Các mô tả này gọi là meta-data.
  - Điều này cho phép DBMS có thể làm việc với nhiều loại ứng dụng CSDL khác nhau
- Cách ly (insulation) chương trình và dữ liệu
  - Gọi là program-data independence.
  - Cho phép thay đổi cấu trúc dữ liệu và tổ chức lưu trữ mà không phải thay đổi chương trình truy cập CSDL

# Ví dụ về database catalog

## RELATIONS

| Relation_name | No_of_columns |
|---------------|---------------|
| STUDENT       | 4             |
| COURSE        | 4             |
| SECTION       | 5             |
| GRADE_REPORT  | 3             |
| PREREQUISITE  | 2             |

## COLUMNS

| Column_name         | Data_type      | Belongs_to_relation |
|---------------------|----------------|---------------------|
| Name                | Character (30) | STUDENT             |
| Student_number      | Character (4)  | STUDENT             |
| Class               | Integer (1)    | STUDENT             |
| Major               | Major_type     | STUDENT             |
| Course_name         | Character (10) | COURSE              |
| Course_number       | XXXXNNNN       | COURSE              |
| ....                | ....           | ....                |
| ....                | ....           | ....                |
| ....                | ....           | ....                |
| Prerequisite_number | XXXXNNNN       | PREREQUISITE        |

Note: Major\_type is defined as an enumerated type with all known majors. XXXXNNNN is used to define a type with four alpha characters followed by four digits

**Figure 1.3**

An example of a database catalog for the database in Figure 1.2.

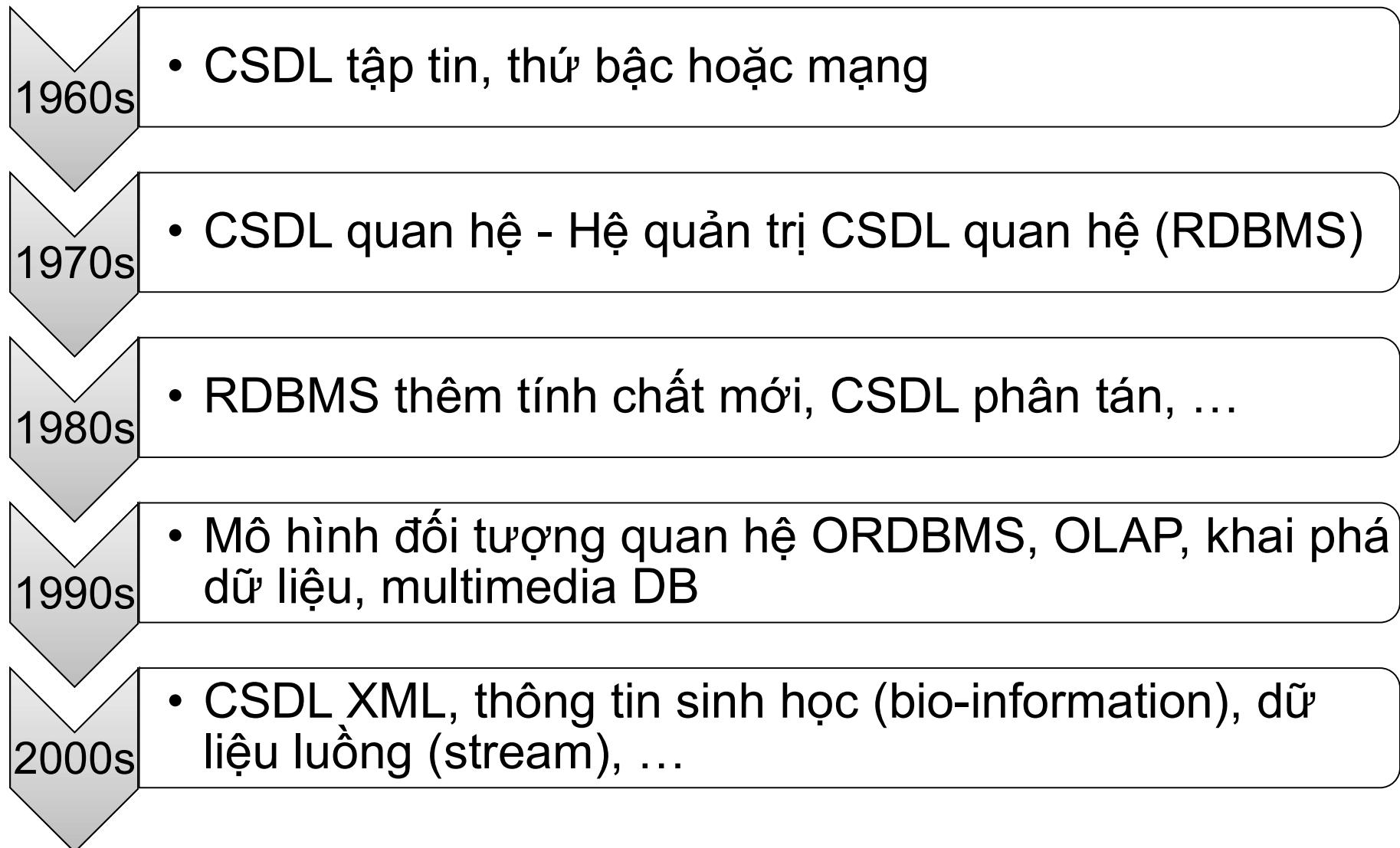
# Các đặc điểm của DBMS

- Trừu tượng hóa dữ liệu:
  - Mô hình dữ liệu (data model) dùng để che dấu các chi tiết lưu trữ vật lý và chỉ thể hiện cho người dùng góc nhìn ở mức ý niệm
  - Các chương trình tham chiếu đến cấu trúc của mô hình dữ liệu chứ không phải đến chi tiết lưu trữ của dữ liệu
- Hỗ trợ nhiều góc nhìn (multiple views)
  - Mỗi user có thể nhìn CSDL ở các góc nhìn khác nhau, DBMS sẽ hỗ trợ để mô tả nhiều góc nhìn khác nhau đối với CSDL

# Các đặc điểm của DBMS

- Chia sẻ dữ liệu và xử lý giao dịch đồng thời của nhiều người dùng
  - Cho phép nhiều người đồng thời truy cập CSDL
  - OLTP (Online Transaction Processing) cho phép nhiều giao dịch (transaction) đồng thời được thực thi cùng một thời điểm
  - Kiểm soát truy cập đồng thời đảm bảo mỗi giao dịch đều được thực hiện đầy đủ hoặc bị hủy bỏ
  - Hệ thống phục hồi (recovery) đảm bảo giao dịch hoàn tất sẽ được lưu trữ bền vững trong CSDL

# Lịch sử phát triển của DBMS



# Ngôn ngữ CSDL

## Ngôn ngữ định nghĩa dữ liệu

(Data Definition Language - DDL)

- Cho phép xác định sơ đồ CSDL, bằng cách đặc tả kiểu dữ liệu, cấu trúc và các ràng buộc dữ liệu
- Kết quả được lưu trong từ điển dữ liệu (data dictionary) của hệ thống → metadata
- Từ điển dữ liệu sẽ được cập nhật mỗi khi có sự sửa chữa cấu trúc của các bảng dữ liệu (table)

# Ngôn ngữ CSDL

## Ngôn ngữ thao tác dữ liệu

(Data Manipulation Language - DML)

- Cho phép người dùng thao tác đến dữ liệu trong CSDL, bao gồm việc truy vấn (query), thêm (insert), xóa (delete) và cập nhật (update) dữ liệu
- Có hai kiểu: có thủ tục (Procedural DML) và phi thủ tục (Nonprocedural DML)

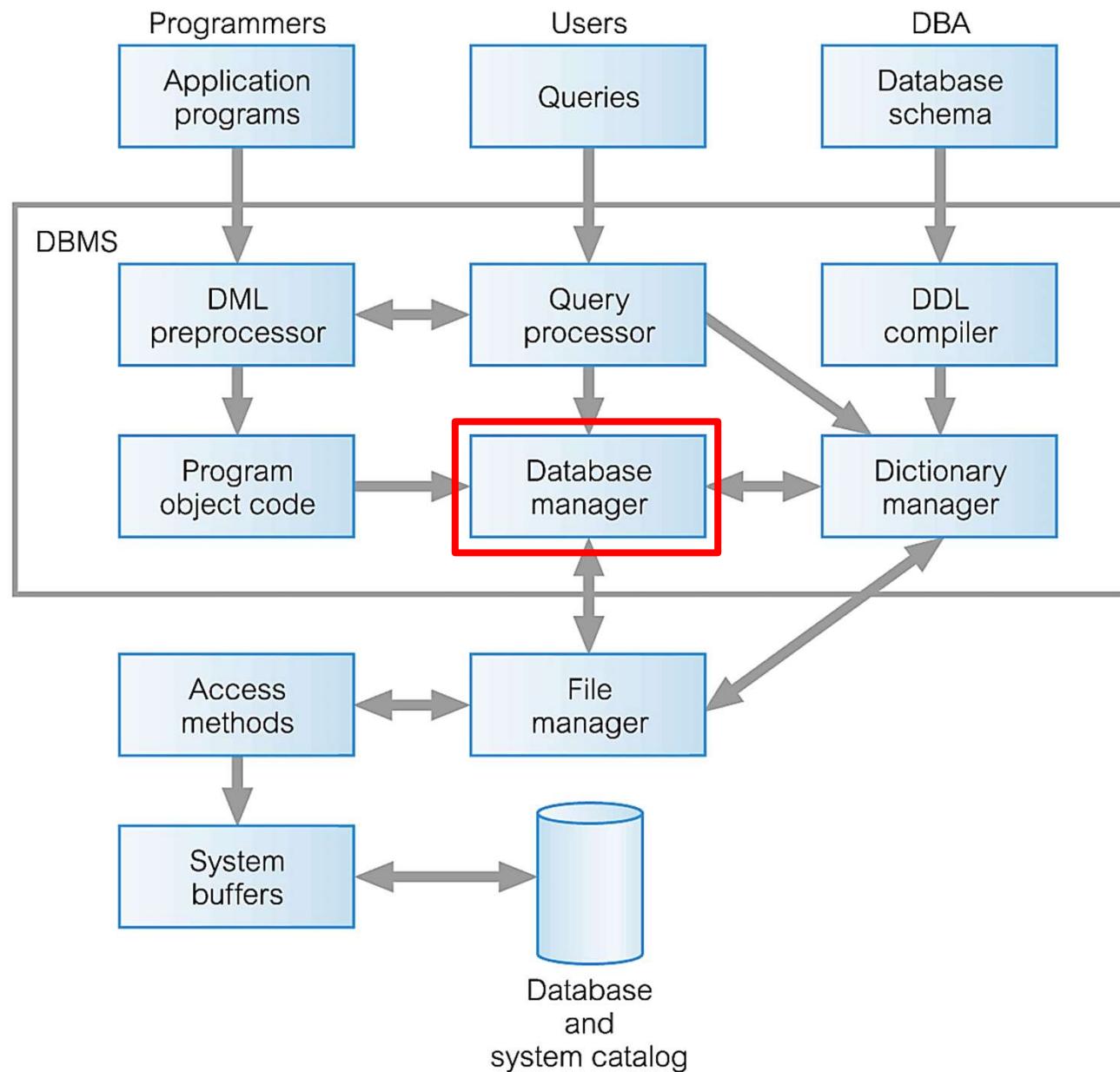
# Ngôn ngữ CSDL

## Ngôn ngữ kiểm soát dữ liệu

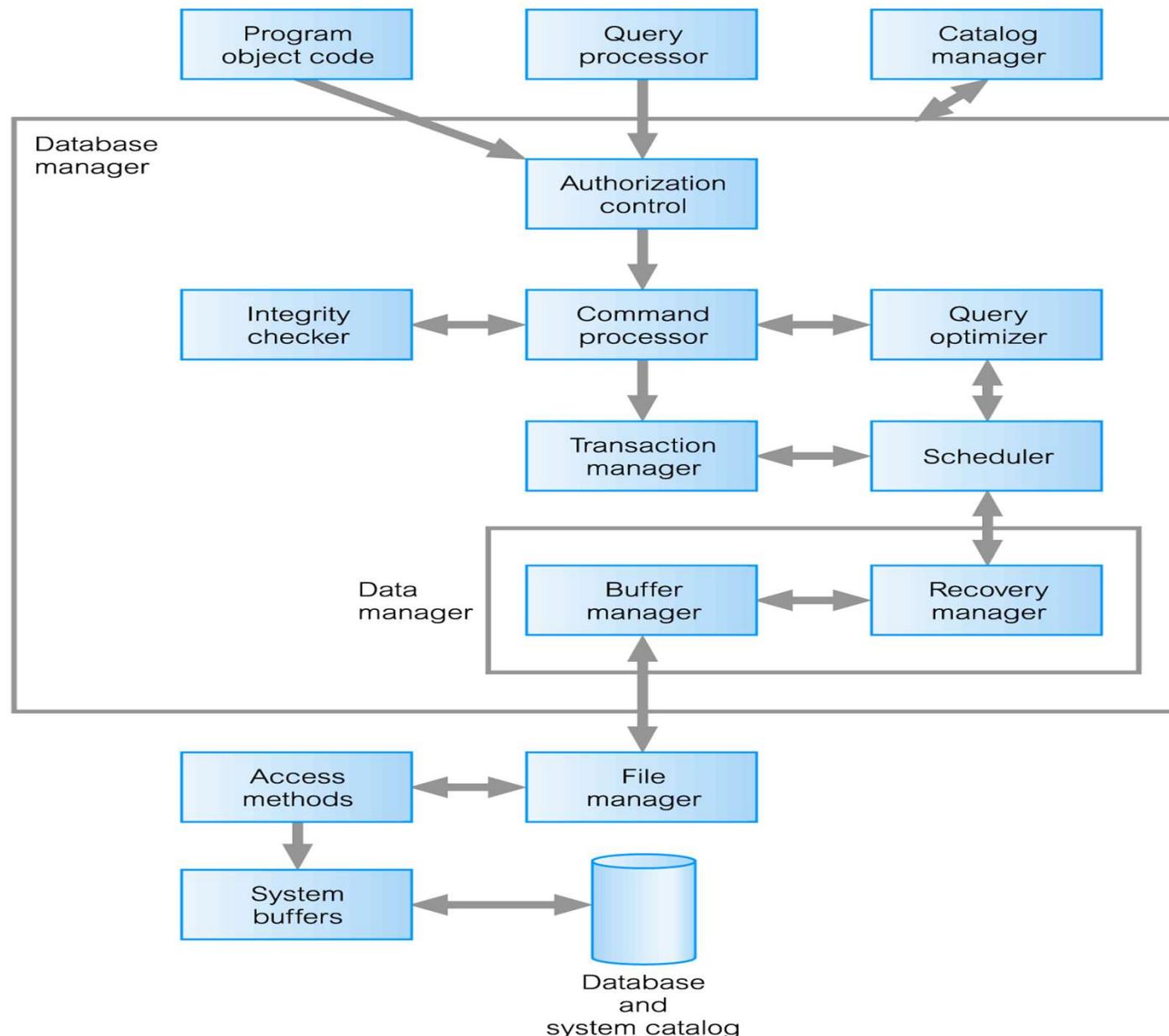
(Data Control Language - DCL)

- Cho phép kiểm soát việc truy cập vào CSDL
- Việc kiểm soát có thể liên quan đến:
  - Bảo mật (security)
  - Kiểm soát truy cập đồng thời (concurrency control)
  - Kiểm soát việc phục hồi (recovery control)
  - ...

# Kiến trúc tổng quát của DBMS



# Kiến trúc tổng quát của Database Manager



# Buffer và Buffer manager

- Dữ liệu trong CSDL bình thường nằm trên thiết bị lưu trữ thứ cấp (secondary storage) như đĩa từ.
- Khi xử lý dữ liệu phải được nạp vào bộ nhớ chính.
- DBMS dùng các vùng đệm (**buffer**) để chứa các dữ liệu cần xử lý. Thay vì xin cấp phát mỗi khi cần, DBMS sẽ xin cấp phát trước các vùng đệm có kích thước đủ lớn để dùng lâu dài
- Bộ quản lý vùng đệm (**Buffer manager**) phân vùng đệm thành các trang (**page**) dùng để chép các khối (**block**) dữ liệu từ đĩa vào các trang này và chép các trang này vào đĩa.

# Buffer và Buffer manager

- Dữ liệu mà các thành phần của DBMS cần gồm:
  - Dữ liệu (**data**): nội dung của CSDL
  - Siêu dữ liệu (**metadata**): mô tả cấu trúc của lược đồ CSDL và các ràng buộc trên đó
  - Dữ kiện thống kê (**statistics**): thông tin về các đặc tính của dữ liệu như kích thước và các giá trị khác được DBMS thu thập và lưu trữ lại (dùng trong việc tối ưu hóa truy vấn chẵng hạn)
  - Chỉ mục (**Index**): cấu trúc dữ liệu hỗ trợ việc truy đạt dữ liệu một cách hiệu quả

# Bộ xử lý truy vấn

- Bộ xử lý truy vấn sẽ xử lý các truy vấn đầu vào và trả lời kết quả cho người dùng
- Đây là phần có ảnh hưởng lớn đến hiệu năng của DBMS mà người dùng thấy rõ nhất
- Bao gồm các thành phần:
  - Bộ biên dịch câu truy vấn (**query compiler**)
  - Bộ máy thực thi (**execution engine**)

# Bộ biên dịch truy vấn

- Bộ biên dịch sẽ chuyển các câu truy vấn thành dạng thức nội (internal form) gọi là kế hoạch truy vấn (**query plan**) là một chuỗi các tác vụ cần phải thực thi trên dữ liệu
- Thường thì một kế hoạch truy vấn là các tác vụ đại số quan hệ (**relational algebra**)
- Bộ biên dịch sử dụng siêu dữ liệu (**metadata**) và dữ kiện thống kê (**statistics**) để quyết định thứ tự các tác vụ trong kế hoạch truy vấn để việc thực thi là tốt nhất

# Bộ biên dịch truy vấn

Gồm 3 thành phần chính:

- Bộ phân tích ngữ pháp truy vấn (**query parser**):
  - Xây dựng cấu trúc cây từ dạng văn bản của câu truy vấn
- Bộ tiền xử lý truy vấn (**query preprocessor**):
  - Kiểm tra ngữ nghĩa của câu truy vấn, ví dụ kiểm tra sự tồn tại của các quan hệ có trong câu truy vấn
  - Chuyển đổi cây ngữ pháp thành cây các tác vụ đại số (algebraic operator) để tạo kế hoạch truy vấn ban đầu
- Bộ tối ưu hóa truy vấn (**query optimizer**)
  - Chuyển đổi kế hoạch ban đầu thành một chuỗi tốt nhất của các tác vụ trên dữ liệu thực tế

# Bộ máy thực thi (Execution Engine)

- Thực thi các bước trong kế hoạch truy vấn
- Giao tiếp với các thành phần khác của DBMS một cách trực tiếp hoặc thông qua các vùng đệm
- Nạp dữ liệu từ CSDL vào trong vùng đệm để thao tác trên các dữ liệu đó
- Giao tiếp với bộ lập lịch (**scheduler**) để tránh truy cập vào các dữ liệu đã bị khoá
- Giao tiếp với bộ quản lý sổ ghi (**log manager**) để đảm bảo các thay đổi trong CSDL phải được ghi lại một cách đầy đủ

# Xử lý giao tác (Transaction)

- Các câu truy vấn và tác vụ được nhóm thành các giao tác (transaction) để thực thi. Giao tác có tính **ACID** (atomicity, consistency, isolation, durability)
- Bộ quản lý giao tác gồm:
  - Bộ quản lý điều khiển tương tranh (concurrency-control manager), hay bộ lập lịch (scheduler)
  - Bộ quản lý sổ ghi và khôi phục (logging and recovery manager)

# Các tác vụ của bộ xử lý giao tác

## ■ Ghi sổ (Logging):

- Để đảm bảo tính bền vững (durability), bất kỳ sự thay đổi nào trong CSDL sẽ được ghi lại vào một sổ ghi độc lập trên đĩa
- Bộ quản lý ghi sổ (**log manager**) đảm bảo việc ghi sổ để khi xảy ra sự cố (system failure) thì bộ quản lý khôi phục (recovery manager) có thể dùng sổ ghi để phục hồi CSDL về trạng thái nhất quán (**consistency**)
- Bộ quản lý sổ sẽ ghi sổ vào vùng đệm (buffer) và sắp xếp với bộ quản lý vùng đệm để đảm bảo sổ sẽ được ghi vào đĩa ở thời điểm thích hợp

# Các tác vụ của bộ xử lý giao tác

- Điều khiển tương tranh (concurrency control):
  - Hệ thống thường có nhiều giao tác thực thi cùng lúc
  - Bộ lập lịch (**scheduler**) đảm bảo các tác vụ trong các giao tác đồng thời được thực thi sao cho hiệu quả giống như khi từng giao tác được thực thi riêng lẻ
  - Scheduler thường dùng cách khoá (**lock**) các mảnh dữ liệu, để tránh hai giao tác truy cập cùng một dữ liệu. Các khoá được lưu trong lock table
  - Scheduler sẽ cấm execution engine truy cập vào các phần của CSDL đã bị khoá

# Các tác vụ của bộ xử lý giao tác

- Giải quyết khoá chết (**deadlock** resolution)
  - Khi các giao tác tranh chấp các tài nguyên bằng cách khoá (theo bộ lập lịch), chúng có thể rơi vào tình trạng là không giao tác nào có thể được thực thi tiếp vì mỗi giao tác cần tài nguyên được giữ bởi các giao tác khác
  - Bộ quản lý giao tác lúc đó can thiệp bằng cách huỷ một trong các giao tác gây deadlock để cho phép các giao tác khác được thực thi

# Database Users

User có thể được chia thành:

- Những người thực sự sử dụng và kiểm soát nội dung trong CSDL và những người thiết kế, phát triển và bảo trì ứng dụng CSDL → gọi là “**Actors on the Scene**”
- Những người thiết kế và phát triển phần mềm DBMS và các công cụ liên quan, và những người trông coi, vận hành (operator) hệ thống máy tính → “**Workers Behind the Scene**”

# Database Users

Actors on the scene

- Database administrators (DBA):
  - Chịu trách nhiệm cấp quyền truy cập CSDL, phối hợp và giám sát việc họ sử dụng các tài nguyên của hệ thống, theo dõi và hiệu chỉnh để tăng hiệu suất của hệ thống, sao lưu và phục hồi, ...
- Database Designers:
  - Chịu trách nhiệm định nghĩa nội dung, cấu trúc, ràng buộc và các chức năng hoặc giao dịch CSDL. Họ phải giao tiếp với người dùng để hiểu rõ nhu cầu của họ

# Phân loại End-users

- Casual:
  - Thỉnh thoảng truy cập CSDL khi cần
- Naïve (Parametric):
  - Đây là nhóm đông nhất, sử dụng các chức năng đã được cài đặt sẵn. Ví dụ các nhân viên bán hàng
- Sophisticated:
  - Phân tích viên, các kỹ sư, người quen thuộc hoàn toàn với các chức năng của hệ thống. Thường sử dụng các gói công cụ để làm việc trực tiếp với CSDL
- Stand-alone:
  - Sử dụng các gói ứng dụng để thao tác với CSDL cá nhân trên máy tính riêng

# Trách nhiệm của DBA

## 1. Bảo mật (Security):

Mỗi người dùng trong hệ thống có những quyền gì? được xem những dữ liệu gì? được sử dụng những chức năng gì? xác thực như thế nào? ...

## 2. Độ sẵn sàng cao (High Availability):

Đảm bảo CSDL luôn sẵn sàng phục vụ ngay cả khi có sự cố. SQL Server cung cấp một số giải pháp như Failover Clustering, Transaction Log Shipping, Database Mirroring,. Tùy độ phức tạp của ứng dụng, cơ sở hạ tầng mà chọn giải pháp thích hợp.

# Trách nhiệm của DBA

## 3. Độ tin cậy (Reliability):

DBA phải có kế hoạch để chẩn đoán và bảo trì hệ thống, phát hiện kịp thời các sự cố tiềm ẩn, đảm bảo hiệu năng của hệ thống. Công cụ cho DBA làm việc đó như System Monitor, SQL Profiler, ...

## 4. Khả năng phục hồi (Recoverability):

Khi sự cố đã xảy ra, Database không thể tiếp tục làm việc, thì nó phải được phục hồi càng nhanh càng tốt với lượng dữ liệu bị mất là ít nhất. Vì vậy DBA phải có kế hoạch Backup và chiến lược phục hồi hiệu quả

# SQL Server DBA Responsibilities

## Security

Auditing, permissions, SSL / TDE  
encryption, least privilege

## Availability

SLA, server design, RAID &  
component redundancy, monitoring  
clustering, log shipping, mirroring

## Reliability

SQLDIAG, service packs,  
performance tuning, load testing,  
DBCC, change control

## Recoverability

Backups, restore verification,  
DR planning, documentation

# Tóm tắt

- Môi trường hệ thống cơ sở dữ liệu
- Chức năng của hệ quản trị CSDL
- Kiến trúc của hệ quản trị CSDL
- Những người tham gia trong một hệ thống CSDL
- Nhiệm vụ của người quản trị CSDL (DBA)

# Chương 2

# Tổ chức lưu trữ dữ liệu



Môn: QUẢN TRỊ CƠ SỞ DỮ LIỆU



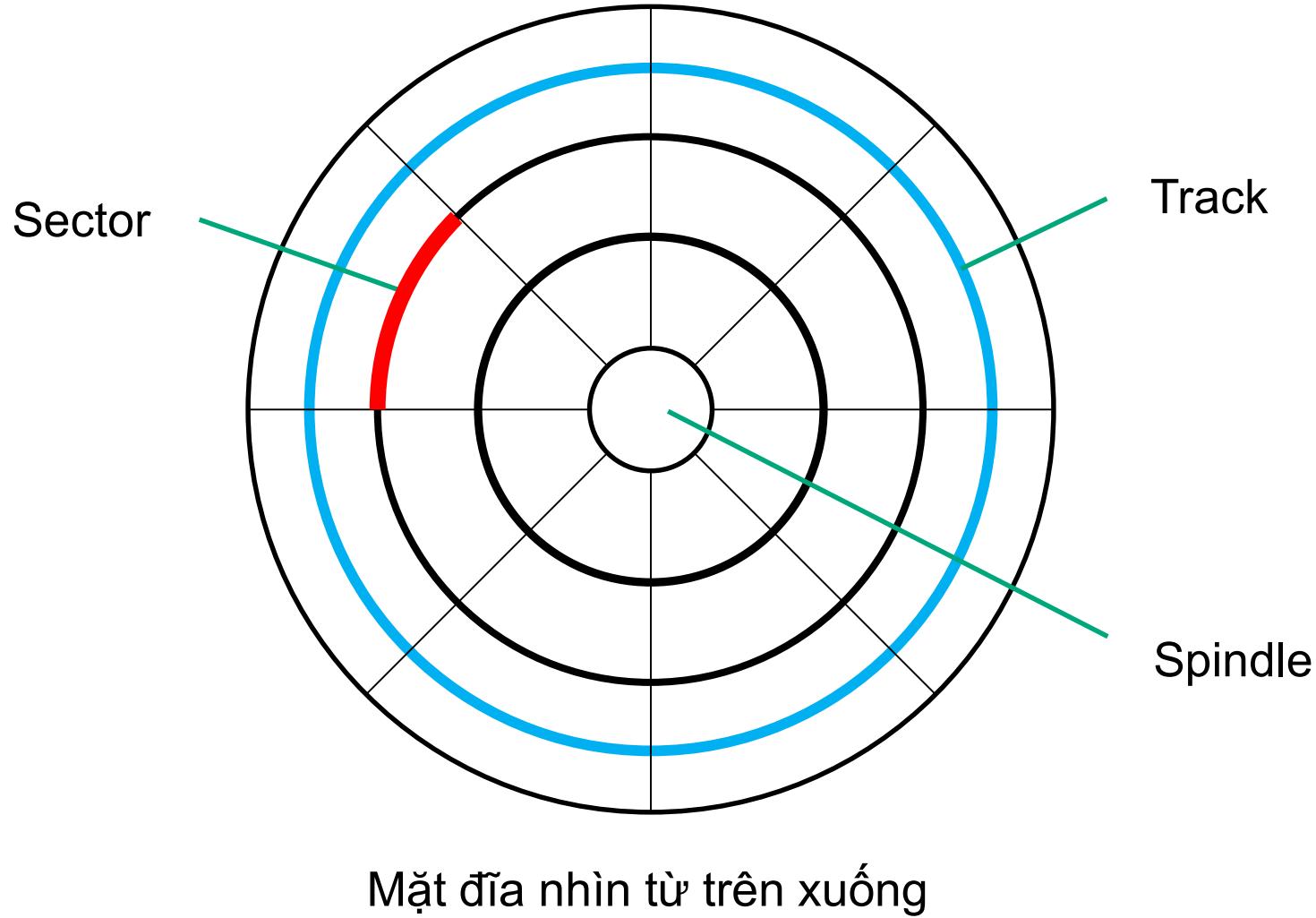
# Nội dung

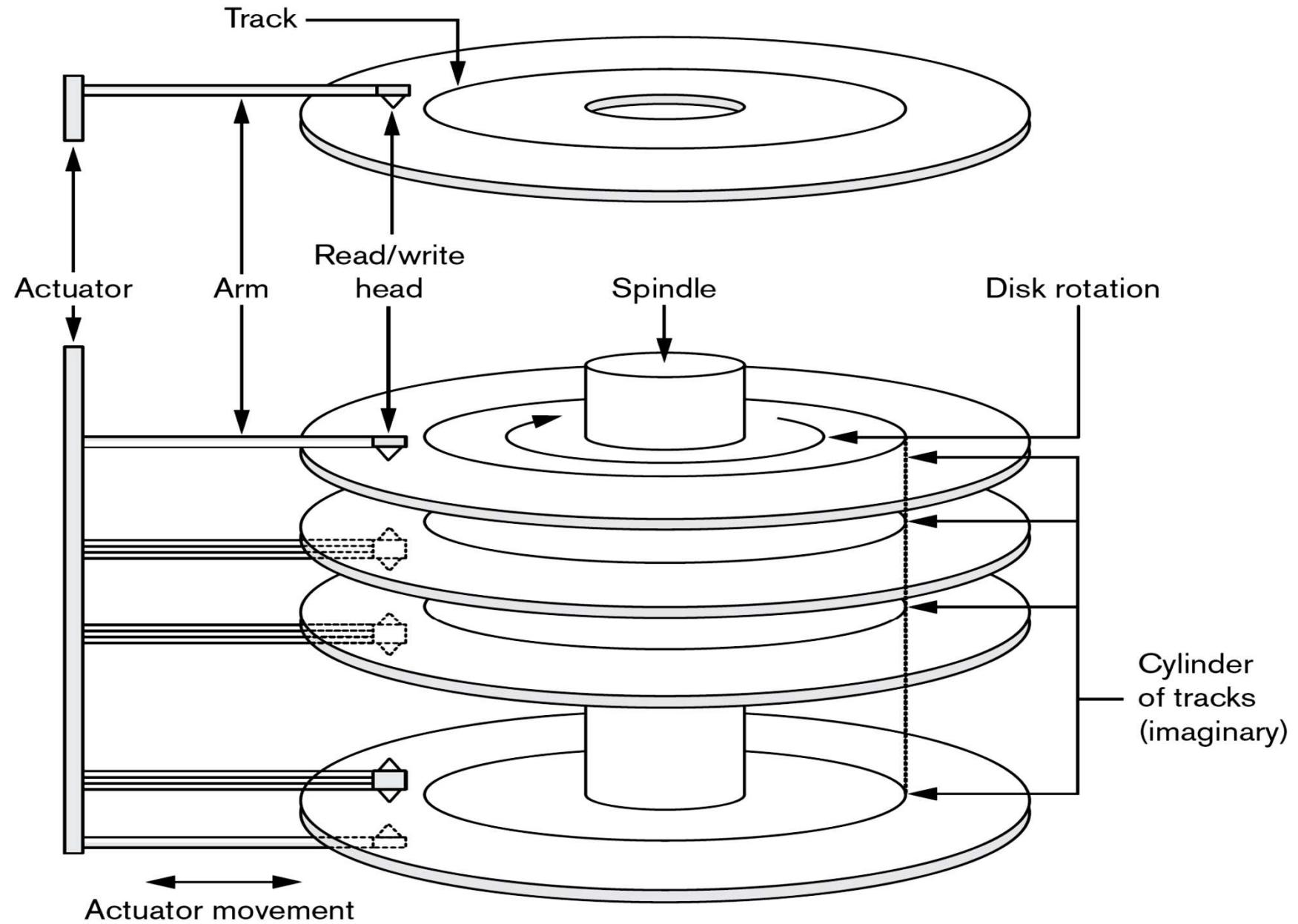
- Cấu trúc đĩa từ
- Tổ chức tập tin
- Tập tin với mẫu tin không có thứ tự
- Tập tin với mẫu tin có thứ tự
- Tập tin dùng kỹ thuật băm
- RAID
- Tổ chức tập tin trong database của SQL server

# Đĩa lưu trữ

- Một ổ đĩa cứng bao gồm nhiều tám đĩa từ quay trên cùng một trục
- Mỗi mặt đĩa được chia thành nhiều vòng tròn đồng tâm gọi là **tracks**.
- Một track được chia thành các mảnh nhỏ hơn gọi là **blocks** hay **sectors**.
- Kích thước block (Block size **B**) cố định ứng với mỗi hệ thống. Kích thước thường từ 512 đến 4096 bytes
- Block sẽ được chuyển cả khối từ đĩa sang bộ nhớ để xử lý hoặc ngược lại

# Đĩa lưu trữ





# Dung lượng đĩa

- Địa chỉ vật lý của một block bao gồm số cylinder, số mặt đĩa và số block (trong track)
- Dung lượng đĩa:

$$\begin{aligned} &= (\text{Số byte/sector}) \times (\text{Số sector/track}) \\ &\quad \times (\text{Số Cylinder}) \times (\text{số đầu đọc/ghi}) \end{aligned}$$

- Giữa các khối có khe hở (interblock gap **G**) là các vùng bị bỏ qua khi đọc 2 khối liên tiếp nhau
- Dung lượng toàn phần:  $B + G$
- Dung lượng sử dụng:  $B$

# Ví dụ

- Giả sử có ổ đĩa cứng với thông số sau:
  - Ổ đĩa gồm 15 tấm đĩa 2 mặt
  - Block B = 512 bytes, Interblock gap G = 128 bytes
  - Số block / track = 20, số track / mặt = 400
- Dung lượng của mỗi track
  - Toàn phần :  $(512 + 128) * 20 = 12800 \text{ bytes} = 12.5 \text{ KB}$
  - Sử dụng :  $512 * 20 = 10240 \text{ bytes} = 10 \text{ KB}$
- Dung lượng của mỗi cylinder
  - Toàn phần =  $12.5 * 15 * 2 = 375 \text{ KB}$
  - Sử dụng =  $10 * 15 * 2 = 300 \text{ KB}$
- Dung lượng của cả ổ đĩa
  - Toàn phần =  $375 * 400 = 150,000 \text{ KB} \sim 146.5 \text{ MB}$
  - Sử dụng =  $300 * 400 = 120,000 \text{ KB} \sim 117.2 \text{ MB}$

# Đọc ghi dữ liệu từ đĩa

- Thời gian định vị - s (seek time)
  - Là thời gian cần thiết để đầu đọc di chuyển đến track có chứa block cần đọc/ghi dữ liệu
  - Thường từ 10 đến 60 ms
- Thời gian trễ quay - rd (Rotational delay / latency)
  - Là thời gian để chờ đĩa quay và đưa block cần đọc/ghi đến dưới đầu đọc
  - Trung bình tốn khoảng 1/2 vòng quay đĩa
  - Giả sử tốc độ đĩa quay là  $p = 5400 \text{ rpm}$ 
    - $$rd = 1 / (2 * 5400) \text{ phút}$$
$$= (60 * 1000) / (2 * 5400) = 5.56 \text{ ms}$$

# Tốc độ truyền trên đĩa

- Tốc độ truyền - **tr** (Transfer rate)
  - $tr = (\text{track\_size}) / (60 * 1000 / p)$   
Ví dụ: tốc độ quay của đĩa là 5400 rpm và kích thước 1 track là 12,800 bytes
    - $tr = 12,800 / (60 * 1000 / 5400) = 1152 \text{ bytes/ms}$
- Thời gian truyền khối - **btt** (Block transfer time)
  - Là thời gian để truyền (đọc/ghi) một khối
  - $btt = B / tr$
- Ví dụ: block size = 512 bytes và tr = 1152 bytes/ms
  - $btt = 512 / 1152 \sim 0.44 \text{ ms}$

# Tốc độ truyền trên địa

- Thời gian tìm và truyền một khối
  - Là thời gian để truyền một khối nếu biết được địa chỉ
  - Phụ thuộc vào thời gian định vị, thời gian trễ quay và thời gian truyền khối
    - $(s + rd + btt)$
- Ví dụ:  $btt = 0.44 \text{ ms}$ ,  $s = 30 \text{ ms}$ ,  $rd = 5.56 \text{ ms}$ 
  - Thời gian tìm và truyền =  $(30 + 5.56 + 0.44) = 36 \text{ ms}$

# Tốc độ truyền trên đĩa

- Thời gian tìm và truyền liên tục các khối trên một track
  - Các khối không liên tục (noncontiguous blocks):
    - $(s + (k * (rd + btt)))$
  - Các khối liên tục (consecutive blocks):
    - $s + rd + (k * btt)$
  - Ví dụ:  $btt = 0.44$  ms,  $s = 30$  ms,  $rd = 5.56$  ms,  $k= 10$ 
    - Khối không liên tục:  $(30 + 10*(5.56 + 0.44) ) = 90$  ms
    - Khối liên tục =  $(30 + 5.56 + 10*0.44 ) = 39.96$  ms
- Kỹ thuật **Double buffering** có thể dùng để tăng tốc việc chuyển các khối dữ liệu liên tiếp nhau trên cùng 1 track

# Tốc độ truyền trên đĩa

## ■ Tốc độ truyền gộp - btr (Bulk transfer rate)

- Trên thực tế, khi truyền các khối liên tục, các block và interblock gap đều được truyền nên tốc độ truyền chỉ nên tính cho phần hữu dụng
  - $btr = (B/(B + G)) * tr$  bytes/ms
  - Ví dụ:  $B = 512, G = 128, tr = 1152$  bytes/ms
    - $btr = 512/(512 + 128) * 1152 = 921.6$  bytes/ms
  - Thời gian truyền k khối liên tục khi đó:
    - $(s + rd + (k * (B/btr)))$  ms
    - Ví dụ:  $30 + 5.56 + 10 * 512/921.6 = 41.12$  ms

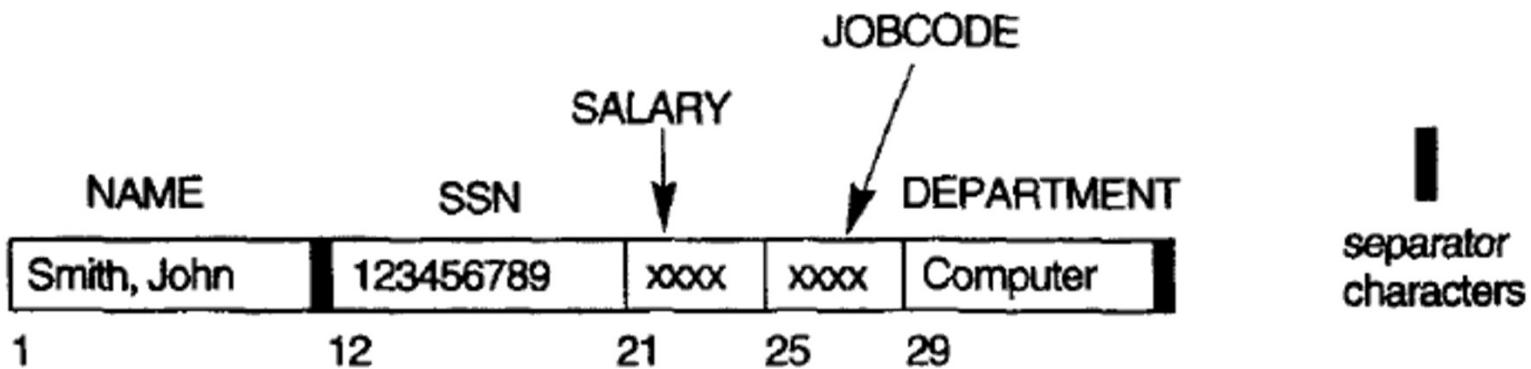
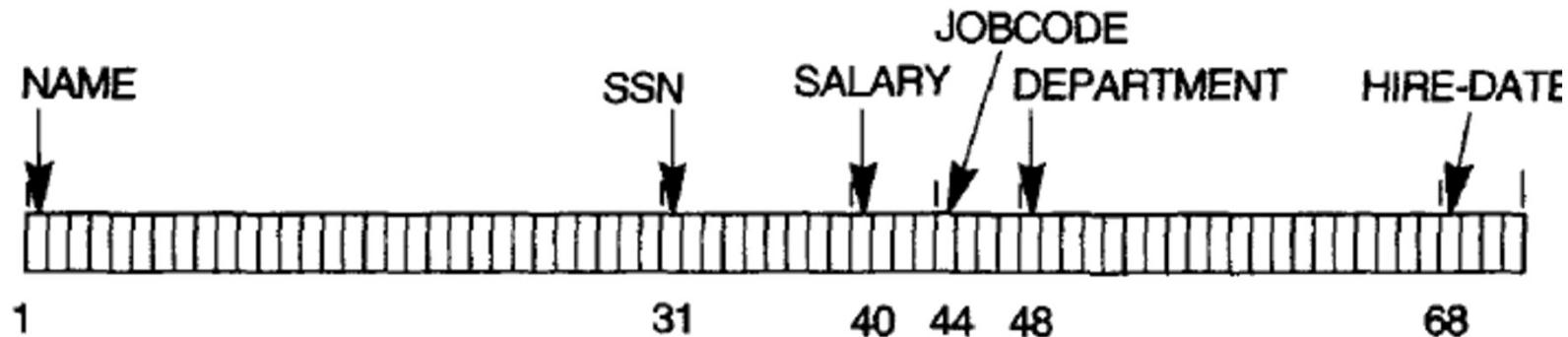
**Table 17.1** Specifications of Typical High-End Cheetah Disks from Seagate

| <b>Description</b>               | <b>Cheetah 15K.6</b> | <b>Cheetah NS 10K</b> |
|----------------------------------|----------------------|-----------------------|
| Model Number                     | ST3450856SS/FC       | ST3400755FC           |
| Height                           | 25.4 mm              | 26.11 mm              |
| Width                            | 101.6 mm             | 101.85 mm             |
| Length                           | 146.05 mm            | 147 mm                |
| Weight                           | 0.709 kg             | 0.771 kg              |
| <b>Capacity</b>                  |                      |                       |
| Formatted Capacity               | 450 Gbytes           | 400 Gbytes            |
| <b>Configuration</b>             |                      |                       |
| Number of disks (physical)       | 4                    | 4                     |
| Number of heads (physical)       | 8                    | 8                     |
| <b>Performance</b>               |                      |                       |
| <b>Transfer Rates</b>            |                      |                       |
| Internal Transfer Rate (min)     | 1051 Mb/sec          |                       |
| Internal Transfer Rate (max)     | 2225 Mb/sec          | 1211 Mb/sec           |
| Mean Time Between Failure (MTBF) |                      | 1.4 M hours           |
| <b>Seek Times</b>                |                      |                       |
| Avg. Seek Time (Read)            | 3.4 ms (typical)     | 3.9 ms (typical)      |
| Avg. Seek Time (Write)           | 3.9 ms (typical)     | 4.2 ms (typical)      |
| Track-to-track, Seek, Read       | 0.2 ms (typical)     | 0.35 ms (typical)     |
| Track-to-track, Seek, Write      | 0.4 ms (typical)     | 0.35 ms (typical)     |
| Average Latency                  | 2 ms                 | 2.98 msec             |
| Courtesy Seagate Technology      |                      |                       |

# Mẫu tin (Record)

- Mỗi mẫu tin (record) chứa các trường (field) của từng kiểu dữ liệu chuyên biệt (ngày tháng, tuổi, tên...)
- Mẫu tin và trường có 2 dạng kích thước:
  - chiều dài cố định (fixed length)
  - chiều dài biến thiên (variable length)
- Các trường có chiều dài biến thiên có thể được trộn vào trong một mẫu tin theo cách:
  - Dùng ký tự phân cách
  - Dựa vào số liệu chiều dài

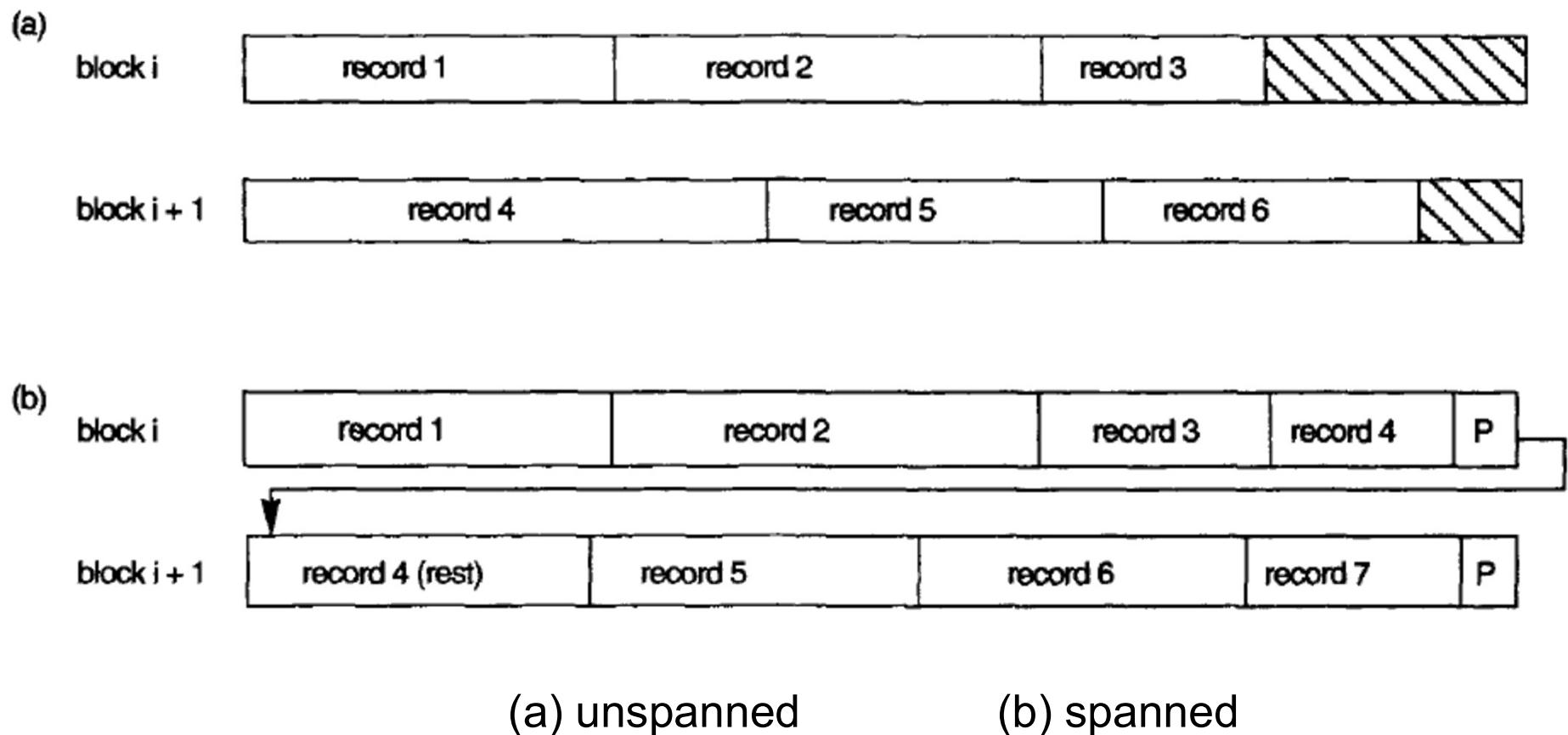
# Mẫu tin



# Blocking

- Là cách phân chia các mẫu tin vào một block
- Hệ số phân khói (Blocking factor) **bfr** là số mẫu tin trong một block
- Có thể có khoảng trống trong block nếu một số lượng nguyên mẫu tin không chiếm hết một block  
 $bfr * record\_size \leq block\_size$
- Spanned Records:
  - Chỉ mẫu tin phải trải từ block này qua block khác
  - Mẫu tin trong file có thể unspanned hoặc spanned

# Unspanned / Spanned record



# Files of Records

- Tập tin chứa các mẩu tin liên tiếp nhau
- Mỗi tập tin sẽ có một phần mô tả (file descriptor hoặc file header) chứa các thông tin mô tả tập tin:
  - Tên các trường
  - Kiểu dữ liệu của từng trường
  - Địa chỉ các khối trên đĩa
  - ...
- Hệ số phân khối của tập tin là giá trị trung bình của số mẩu tin chứa trong một block

# Các tác vụ trên tập tin

- Thường có hai nhóm tác vụ trên tập tin
  - Các tác vụ truy xuất (retrieval operations):
    - Không thay đổi dữ liệu
    - Truy xuất các mẫu tin
  - Các tác vụ cập nhật (update operations)
    - Thay đổi dữ liệu (chèn – insertion, xoá – delete hoặc hiệu chỉnh - modification)
- Các tác vụ này thường kèm theo một điều kiện chọn (selection condition) để lựa chọn các mẫu tin phù hợp
- Các tác vụ thực tế phụ thuộc vào hệ thống

# Các tác vụ (operations) trên tập tin

- OPEN: Mở một tập tin để truy cập và cung cấp một con trỏ tập tin trỏ đến mẫu tin hiện hành của tập tin
- FIND: Tìm đến mẫu tin đầu tiên thoả một điều kiện
- FINDNEXT: Tìm đến mẫu tin kế tiếp thoả điều kiện
- READ: Đọc mẫu tin hiện hành vào một biến bộ nhớ
- INSERT: Chèn một mẫu tin mới vào tập tin
- DELETE: Xoá mẫu tin hiện hành của tập tin (thường là đánh dấu đã xoá để không dùng nữa)
- MODIFY: Sửa giá trị của trong mẫu tin
- CLOSE: Đóng tập tin (không còn truy cập nữa)
- Các tác vụ trên (trừ OPEN và CLOSE) còn gọi là các tác vụ mỗi lần một bản ghi (record-at-a-time)

# Tổ chức tập tin - phương pháp truy cập

- Tổ chức tập tin (file organization):
  - Cách thức tổ chức dữ liệu của tập tin vào các mảng tin, khối, cấu trúc truy cập và cách lưu trữ các mảng tin, khối trên đĩa
- Phương pháp truy cập (access method):
  - Nhóm các tác vụ có thể dùng trên một dạng tập tin.
  - Có thể có nhiều hơn một phương pháp truy cập đối với tổ chức tập tin

# Tập tin không có thứ tự

- Files of Unordered Records
- Còn gọi là **heap** hay **pile file**
- Các mẫu tin mới sẽ được thêm vào cuối tập tin
  - Thêm vào rất nhanh
- Tìm kiếm một mẫu tin thường dùng phép tìm tuần tự (linear search)
  - Hiệu quả thấp
- Đọc các mẫu tin theo thứ tự của một trường nào đó đòi hỏi việc sắp xếp các mẫu tin

# Tập tin có thứ tự

- Files of Ordered Records
- Các mẫu tin trong tập tin được sắp xếp thứ tự theo một trường (ordering field)
- Thao tác thêm vào rất tốn kém do các mẫu tin phải được chèn vào theo đúng thứ tự
  - Có thể dùng một tập tin rời để lưu các mẫu tin mới thêm vào và định kỳ trộn tập tin này vào tập tin gốc
- Phép tìm nhị phân (binary search) được dùng để tìm các mẫu tin dựa trên trường xếp thứ tự
  - Chi phí khoảng  $\log_2 N$ , với N là số mẫu tin
- Đọc mẫu tin theo thứ tự đã xếp thứ tự rất hiệu quả

# Tập tin dùng kỹ thuật băm

- Các khối trong tập tin được chia thành M thùng (bucket) cùng kích thước được đánh số là bucket0, bucket1, ..., bucketM-1
- Thường mỗi thùng tương ứng cho một (hoặc một số cố định các) khối trên đĩa
- Một mục tin được chọn làm khoá băm (hash key)
- Mẫu tin với khoá băm là K sẽ được chứa trong thùng thứ i, với  $i = h(K)$  và h là hàm băm (hash function)
- Việc tìm kiếm rất nhanh trên mục tin là khoá băm

# RAID

- RAID – Redundant Array of Inexpensive Disks
- RAID là được dùng để tăng hiệu suất và sửa chữa những lỗi trong quá trình làm việc của ổ cứng
- Một tập các đĩa gom lại thành 1 đĩa duy nhất đối với người dùng
- Tổ chức RAID được định nghĩa dựa trên sự kết hợp của hai yếu tố:
  - Mức độ chia vệt (theo bit, byte, block)
  - Khuôn mẫu dùng tính các thông tin trùng lắp

# Data striping

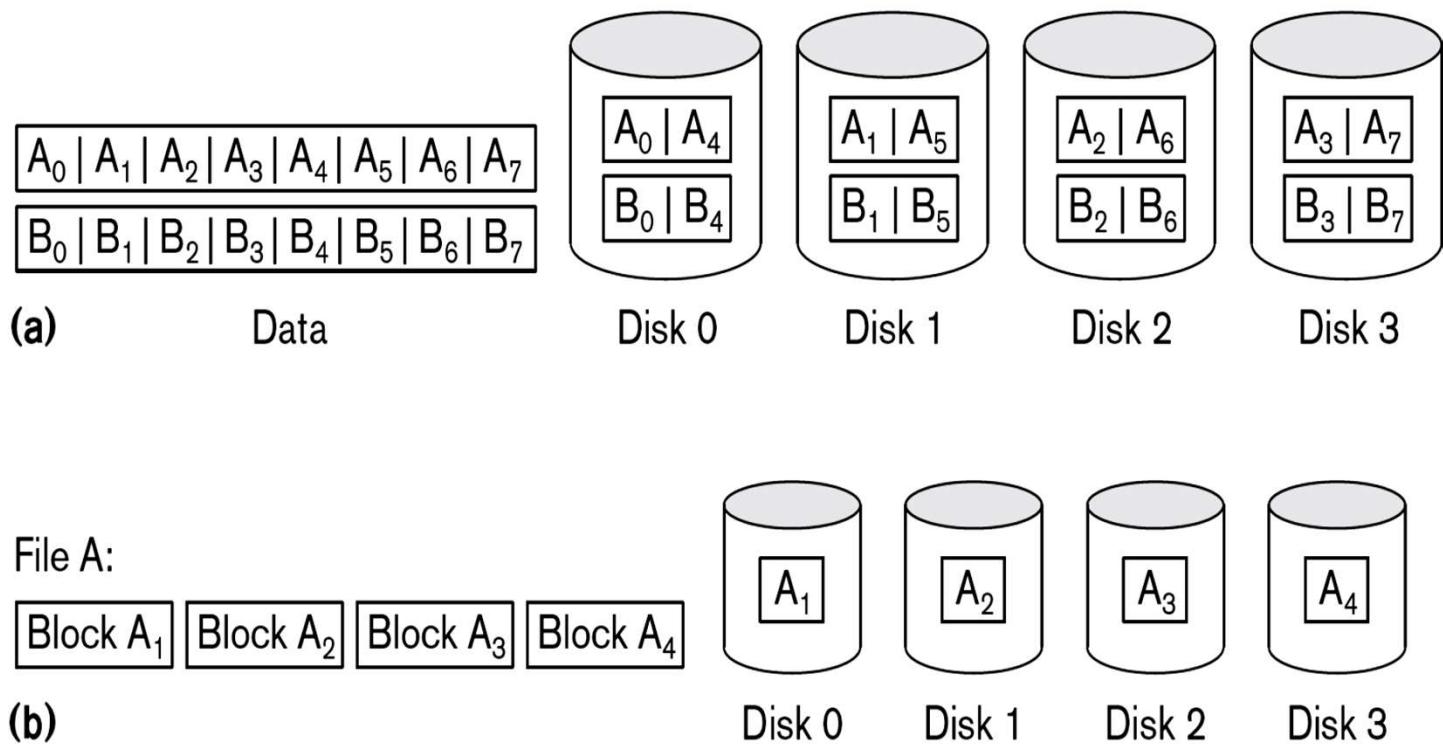
- Data striping: phân bổ các mảnh dữ liệu đến nhiều đĩa và xem như đó là một đĩa lớn. Tốc độ truy cập nhanh do các mảnh có thể được truy cập đồng thời

**Figure 17.13**

Striping of data across multiple disks.

(a) Bit-level striping across four disks.

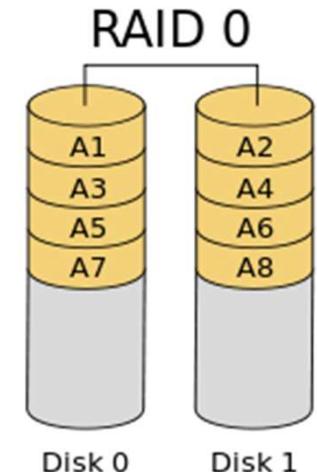
(b) Block-level striping across four disks.



# Các mức RAID

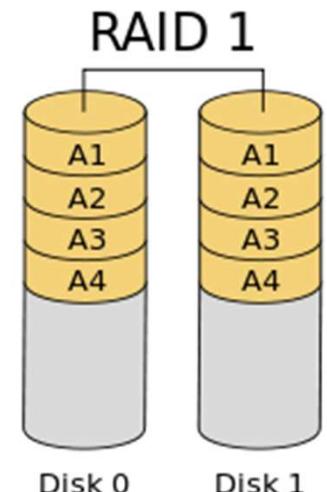
## ■ RAID 0 (stripping)

- Tối đa hóa tính song song
- Không có trùng lắp dữ liệu
- Tốc độ đọc/ ghi tốt nhất
- Không có khả năng chịu đựng lỗi



## ■ RAID 1 (Mirror)

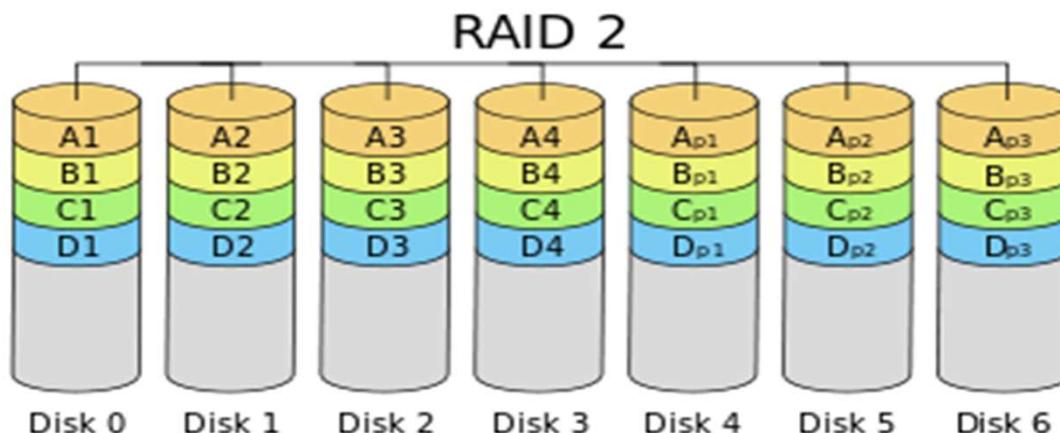
- Trùng lắp dữ liệu, khi một đĩa hỏng, đĩa còn lại làm việc bình thường => chịu lỗi
- Có thể đọc từ đĩa nào đang có tải thấp hơn => tăng tốc đọc, giảm tốc ghi



# Các mức RAID

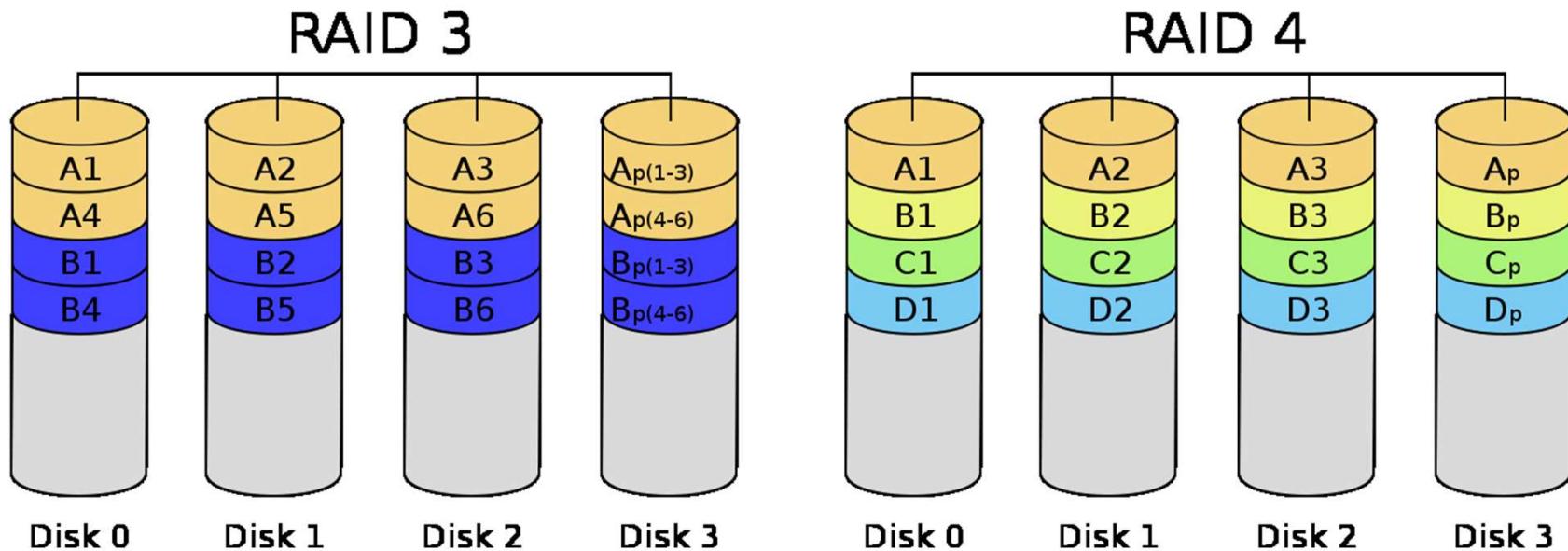
## ■ RAID 2

- Gồm hai cụm ổ đĩa, cụm 1 chứa dữ liệu được phân tách mức bit giống RAID 0, cụm 2 chứa các bit kiểm tra chẵn lẻ kiểu mã Hamming dành cho sửa chữa lỗi ở cụm 1 (có thể giúp kiểm tra một hoặc hai bit bị lỗi và cách sửa chúng)
- Không có trùng lắp dữ liệu



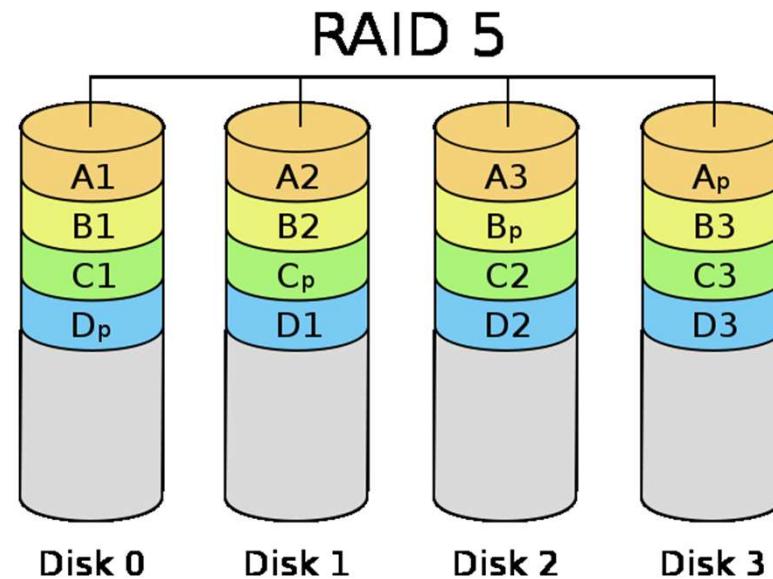
# Các mức RAID

- RAID 3 chia dữ liệu mức byte (byte-level) và một đĩa kiểm tra chẵn lẻ (parity disk)
- RAID 4 chia dữ liệu ở mức khối (block-level) và dùng một đĩa để lưu thông tin kiểm tra chẵn lẻ



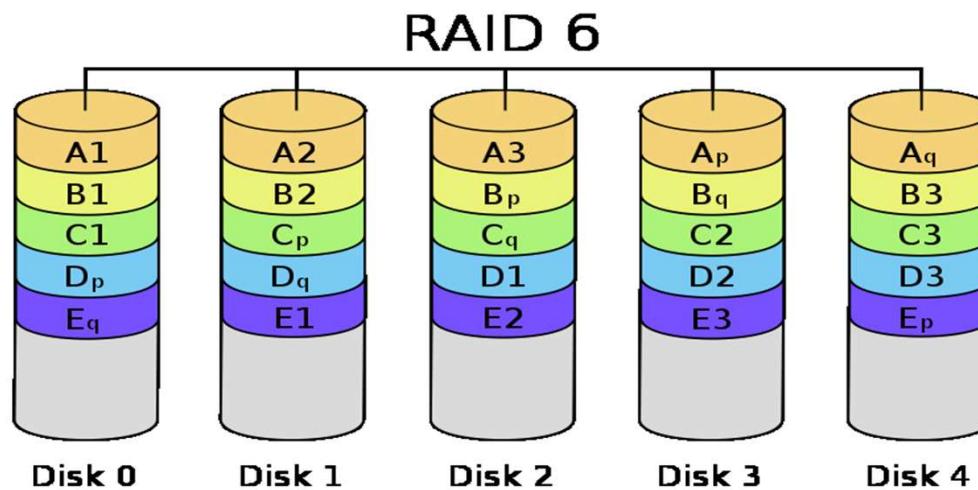
# Các mức RAID

- RAID 5 chính là RAID 4 nhưng các thông tin kiểm tra chẵn lẻ được phân bố đều khắp các đĩa.
- Khi có sự cố trên 1 đĩa, các thông tin kiểm tra chẵn lẻ được dùng để tái tạo lại dữ liệu trên đĩa hỏng trong khi chờ thay thế



# Các mức RAID

- RAID mức 6 áp dụng lược đồ trùng lắp gọi là P+Q và mã Reed-Solomon (Reed-Solomon codes) để bảo vệ hệ thống đến mức có 2 đĩa hỏng bằng việc dùng hai đĩa trùng lắp
  - Dùng 2 bản thông tin kiểm tra chẵn lẻ
  - Phân bố 2 bản này giữa các đĩa



# Mạng lưu trữ SAN

- Do nhu cầu các bộ lưu trữ lớn ngày càng nhiều, chúng ta cần phải dùng các hạ tầng động và linh hoạt hơn để lưu trữ và xử lý thông tin
- Mạng lưu trữ (Storage Area Networks – SANs):
  - Các thiết bị ngoại vi lưu trữ dữ liệu trực tuyến được cấu hình như một nút trong một mạng tốc độ cao. Có thể được gắn vào hoặc lấy ra khỏi các máy chủ dễ dàng
  - Cho phép các hệ thống lưu trữ được đặt xa hơn các máy chủ và cung cấp các hiệu suất khác nhau cũng như các tùy biến kết nối khác nhau

# DATABASE TRONG SQL SERVER

# Tổ chức tập tin CSDL

- Mỗi database của SQL Server gồm:
  - **Data files**: dùng để lưu dữ liệu (data) và đối tượng (objects) như tables, indexes, views, triggers, và stored procedures. Có 2 loại data files: **primary** và **secondary**.
  - **Log files** dùng để lưu thông tin transaction log.
- Mỗi database phải có ít nhất **1 data file** và **1 log file**
- Kích thước tối đa của một data file là **32 terabytes** (TB), của một log file là 4 TB
- Các tập tin có thể nhóm lại thành filegroups để dễ dàng quản lý và nâng cao hiệu suất

# Primary data file

- Chứa tất cả các thông tin khởi động của database
- Chứa con trỏ đến các file còn lại trong database.
- Lưu dữ liệu và các đối tượng của hệ thống cũng như của người dùng.
- **Mỗi database có đúng một primary file.**
- Phần mở rộng mặc định là **.mdf**

# Secondary data files

- Có thể có hoặc không (tùy chọn)
- Có thể dùng mở rộng lưu dữ liệu và đối tượng của người dùng không có trong primary file.
- Có thể dùng secondary files để phân bổ dữ liệu trên nhiều đĩa nhằm tăng hiệu suất truy cập.
- Phần mở rộng mặc định là **.ndf**

# Transaction log files

- Lưu tất cả các thông tin cần thiết về các transaction giao dịch dùng trong phục hồi (recover) dữ liệu.
- Mỗi database phải có ít nhất một log file.
- Phần mở rộng mặc định là **.ldf**
- Có thể có **nhiều log files**, và nếu có nhiều đĩa thì nên đặt các log files này trên các đĩa khác nhau.
- Nên đặt log files trên một đĩa vật lý (physical disks) khác với đĩa chứa data files

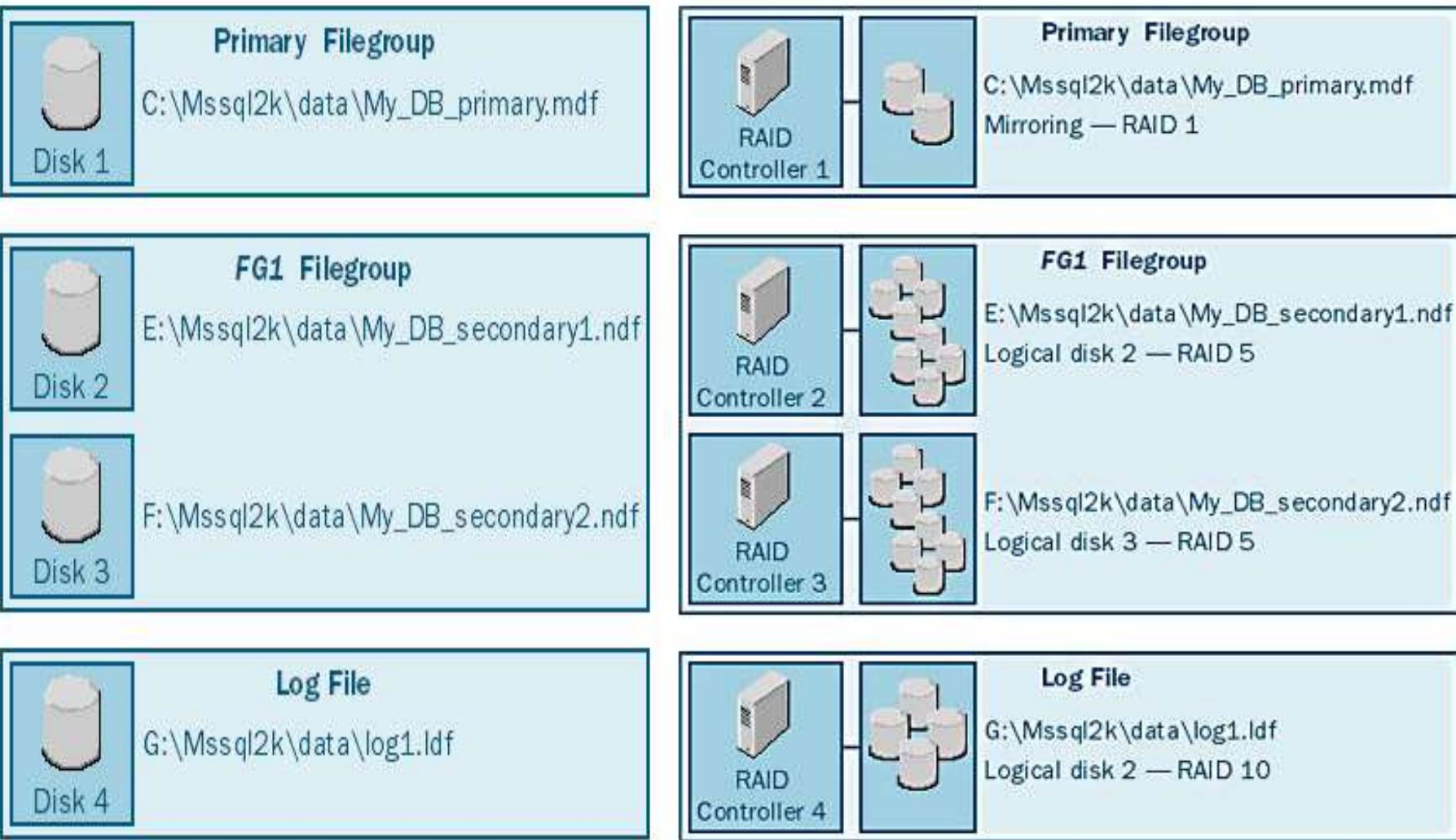
# Filegroups

- Filegroups cho phép nhóm các file để dễ quản lý.
  - Ví dụ khi database quá lớn không thể backup tất cả cùng lúc, ta có thể backup database theo file hay filegroup → kiểm soát hiệu quả việc backup
- Filegroups có thể nâng hiệu suất bằng cách cho phép một database có thể trãi trên nhiều đĩa hoặc hệ thống RAID.
  - Có thể tạo tables và indexes trên các filegroups khác nhau.
  - Các table có lượng truy cập nhiều được trãi trên nhiều đĩa → cải thiện vấn đề I/O của đĩa

# Filegroups

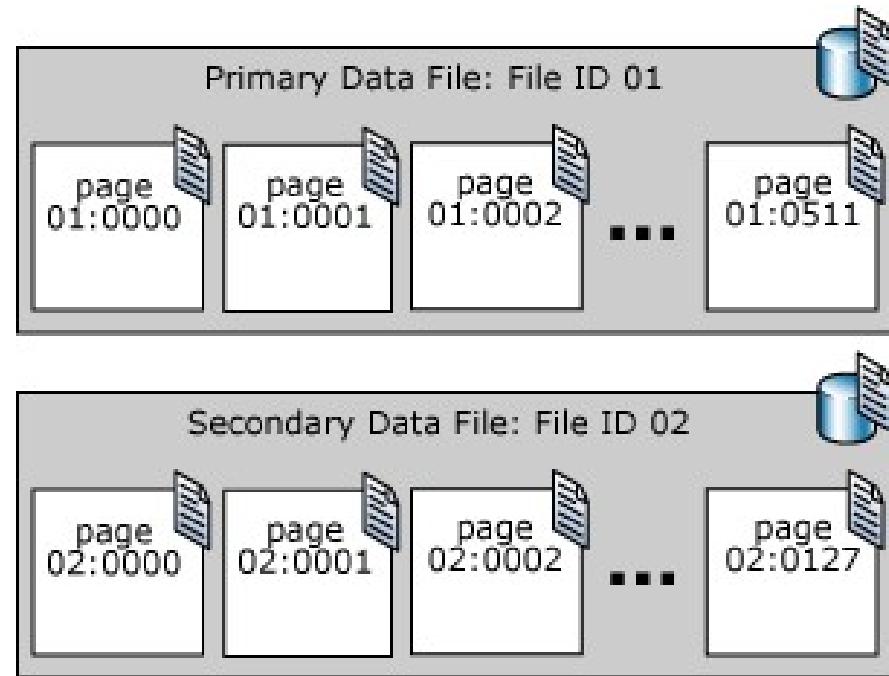
- Primary filegroup: Chứa primary data files, system tables và các file không đặt vào các filegroup khác.
  - System tables: các table do SQL Server tự động tạo ra khi ta tạo database.
- User-defined filegroups: Bao gồm các filegroups do user tạo ra. Một table hay index có thể được tạo và đặt trong các filegroup riêng của user
- Default filegroup: Chứa các tables và indexes khi tạo không chỉ rõ filegroup. Default filegroup mặc định là primary filegroup. Thành viên của db\_owner có thể thay đổi default filegroup.

# Ví dụ về Filegroups



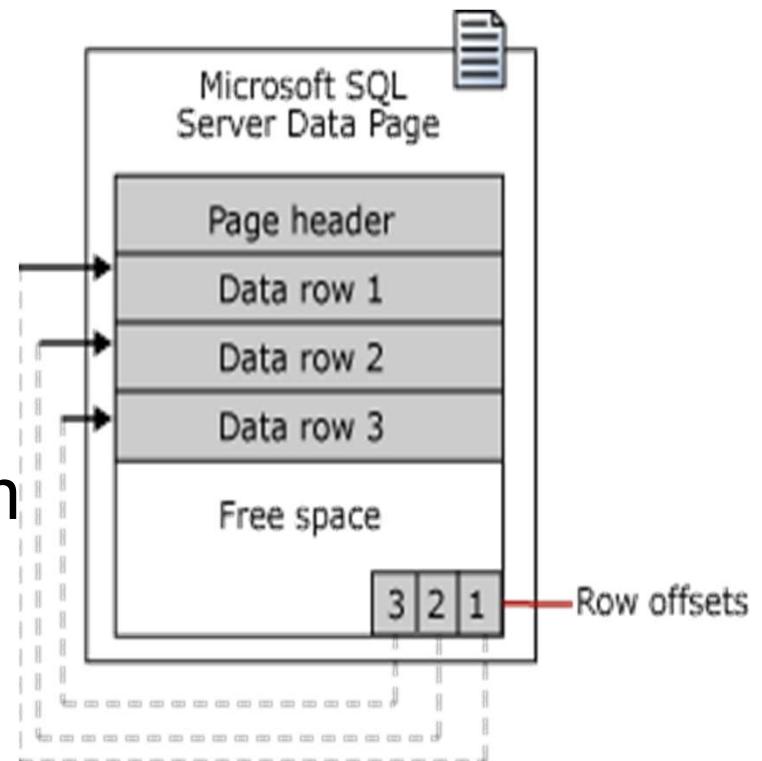
# Tổ chức bên trong tập tin

- Mỗi tập tin CSDL được chia thành các trang (page) 8 KB, được đánh số để phân biệt
- Trang đầu tiên trong mỗi tập tin là phần mô tả tập tin (file header)



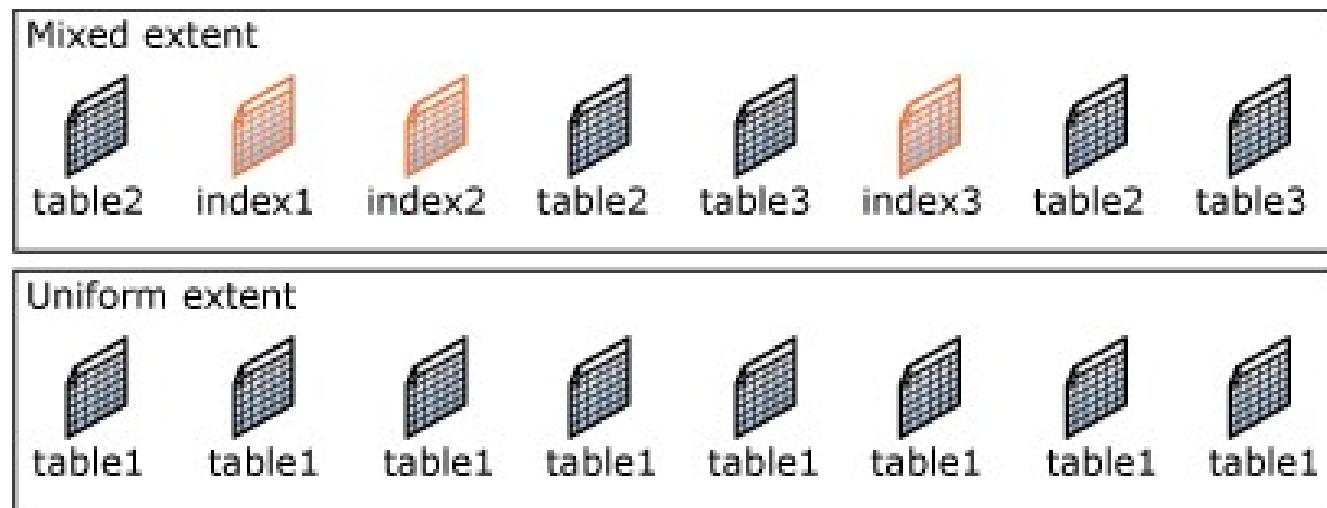
# Tổ chức bên trong tập tin

- Phần đầu trang (page header) có 96 bytes lưu số trang (page number), kiểu (page type), số không gian còn trống trong trang, ...
- Các dòng (row) dữ liệu được lưu liên tục nhau kế tiếp
- Bảng lưu địa chỉ các dòng nằm ở cuối trang



# Tổ chức bên trong tập tin

- Mỗi 8 trang kè nhau tạo thành một extent (64 KB)
- Có 2 loại extent:
  - Uniform: chiếm giữ bởi chỉ một đối tượng
  - Mixed: nhiều đối tượng chia sẻ nhau



# Tạo database

Có thể tạo databases bằng 2 cách:

- Dùng SQL Server Management
  - Đơn giản, dùng giao diện đồ họa để tạo và cấu hình database
- Dùng lệnh CREATE DATABASE
  - Yêu cầu phải hiểu rõ cấu trúc lệnh và các tham số. Có thể lưu thành T-SQL để sử dụng trong nhiều tình huống sau này

# Lệnh CREATE DATABASE

CREATE DATABASE TestDB

ON PRIMARY  
(NAME = 'TestDB\_data1',  
FILENAME = 'C:\TestDB\_data1.mdf',  
SIZE = 102400KB,  
MAXSIZE = UNLIMITED,  
FILEGROWTH = 15%),

FILEGROUP [FG1]  
(NAME = 'TestDB\_data2',  
FILENAME = 'C:\TestDB\_data2.ndf',  
SIZE = 102400KB,  
MAXSIZE = UNLIMITED,  
FILEGROWTH = 15%),

FILEGROUP [FG2]  
(NAME = 'TestDB\_data3',  
FILENAME = C:\TestDB\_data3.ndf',  
SIZE = 102400KB,  
MAXSIZE = UNLIMITED,  
FILEGROWTH = 15%)

LOG ON  
(NAME = 'TestDB\_log',  
FILENAME = N'C:\TestDB\_log.ldf',  
SIZE = 51200KB ,  
MAXSIZE = 2048GB ,  
FILEGROWTH = 10%)

# Chế độ Automatic File Growth

- SQL Server cho phép file tự động tăng kích thước (growth increment) khi cần thiết → giảm gánh nặng cho người quản trị
- Một file được tạo ra với kích thước ban đầu (initial size). Khi dữ liệu đầy, SQL Server sẽ tăng kích thước file, cho đến khi hết đĩa hoặc đạt tới kích thước tối đa (maximum file size).
- Để tránh tình trạng hết đĩa dẫn đến lỗi, cũng như vấn đề phân mảnh trên đĩa khi tập tin tăng dần kích thước, nên dự đoán kích thước tối đa cho mỗi file trước khi tạo.

# Thủ tục sp\_helpdb

- Dùng để xem thông tin chi tiết của database

The screenshot shows the SQL Server Management Studio interface. The query window contains the command:

```
exec sp_helpdb 'TestDB'
```

The results pane displays two tables of information about the TestDB database.

|   | name   | db_size   | owner       | dbid | created    | status                                  |
|---|--------|-----------|-------------|------|------------|---|
| 1 | TestDB | 350.00 MB | QUANG\admin | 11   | Aug 9 2009 | Status=ONLINE, Updateability=READ_WRITE |

|   | name         | fileid | filename            | filegroup | size      | maxsize       | growth | us |
|---|--------------|--------|---------------------|-----------|-----------|---------------|--------|----|
| 1 | TestDB_data1 | 1      | C:\TestDB_data1.mdf | PRIMARY   | 102400 KB | Unlimited     | 15%    | da |
| 2 | TestDB_log   | 2      | C:\TestDB_log.ldf   | NULL      | 51200 KB  | 2147483648 KB | 10%    | lo |
| 3 | TestDB_data2 | 3      | C:\TestDB_data2.ndf | FG1       | 102400 KB | Unlimited     | 15%    | da |
| 4 | TestDB_data3 | 4      | C:\TestDB_data3.ndf | FG2       | 102400 KB | Unlimited     | 15%    | da |

# Tóm tắt

- Cấu trúc đĩa từ
- Tổ chức tập tin
- Tập tin với mẫu tin không có thứ tự
- Tập tin với mẫu tin có thứ tự
- Tập tin dùng kỹ thuật băm
- RAID
- Tổ chức tập tin trong database của SQL server

# Chương 3

# QUẢN TRỊ ĐỐI TƯỢNG CSDL



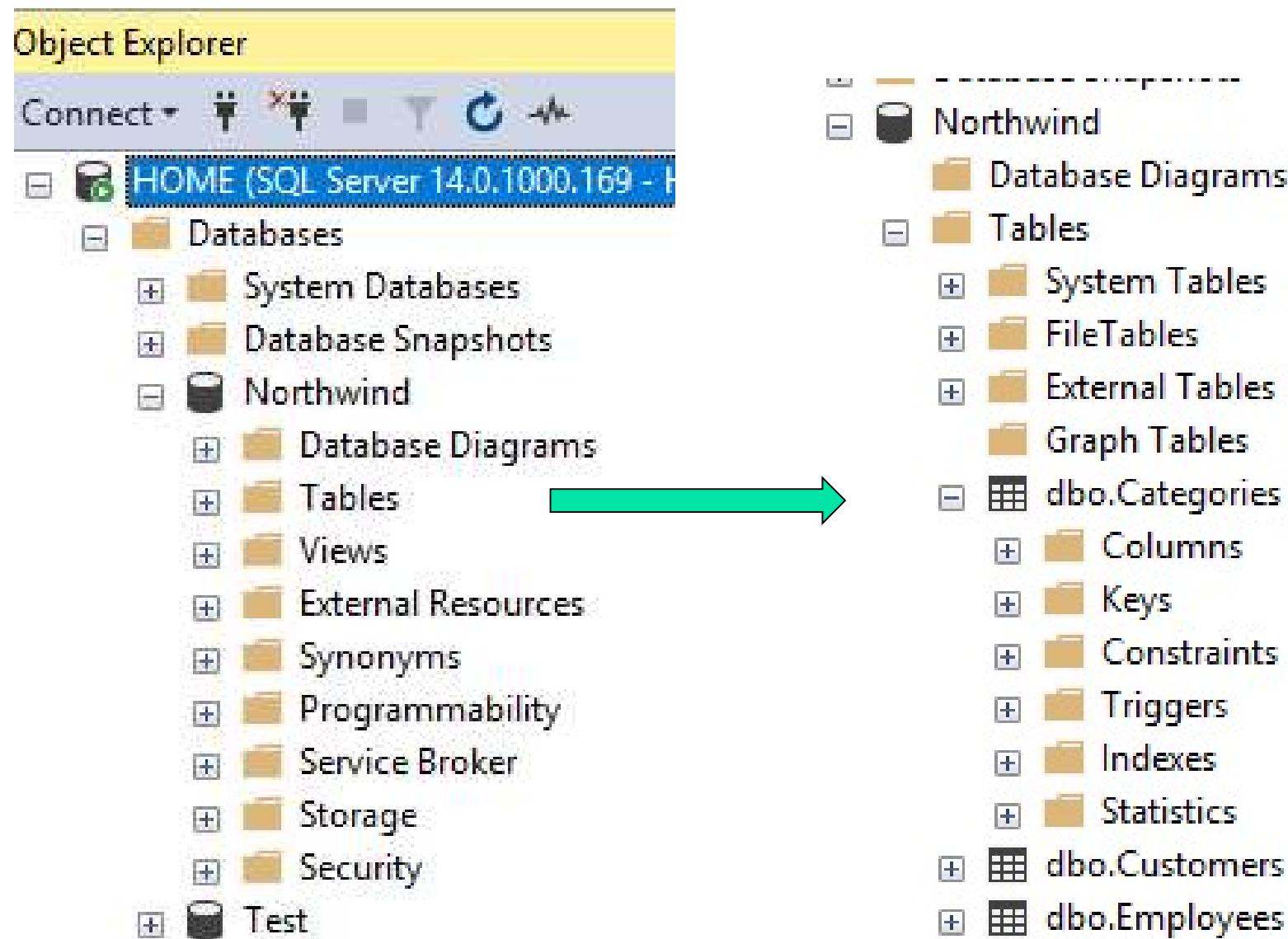
Môn: QUẢN TRỊ CƠ SỞ DỮ LIỆU



# Nội dung

- Database
- Các loại đối tượng cơ bản trong SQL Server
- Table
- View
- Index
- Giới thiệu một số đối tượng CSDL khác

# Database trong SQL Server



# System Databases

Có 4 database hệ thống được tạo ra khi cài đặt:

- **Master**: Lưu các thông tin của hệ thống, như thông tin khởi động, cấu hình của SQL Server, các login accounts, sự tồn tại của các database khác. Luôn giữ 1 bản backup gần nhất của master database
- **Tempdb**: Giữ table tạm (temporary tables) và thủ tục lưu trữ tạm (temporary stored procedures). Database này cũng dùng cho các lưu trữ tạm khi SQL Server cần, ví dụ cho việc sắp xếp dữ liệu. Một bản sạch (clean copy) của tempdb database được tạo lại mỗi khi SQL Server khởi động lại.

# System Databases

- **Model**: Dùng như một khuôn mẫu (template) cho tất cả các database tạo ra trong hệ thống. Nếu ta sửa trong model database thì các database mới tạo ra đều có các đặc tính này
- **Msdb**: Giữ các table mà SQL Server Agent dùng để định thời cho các job (scheduling jobs) và cảnh báo cho các tác vụ
- Mỗi system databases có riêng primary data file và log file. Các databases được lưu trong folder mà ta đã chỉ định khi cài đặt SQL Server

# Các đối tượng trong CSDL

| Đối tượng | Mô tả   |
|-----------|---|
| Table     | Bảng chứa dữ liệu   |
| View      | Table ảo. Thông qua câu truy vấn để lấy dữ liệu từ các table hoặc các view khác |
| Index     | Chỉ mục. Cấu trúc lưu trữ nhằm truy xuất nhanh dữ liệu                          |
| Synonym   | Cho phép đặt thêm các tên tương đương cho các đối tượng CSDL                    |

# Các đối tượng CSDL lập trình được

- +  External Resources
- +  Synonyms
-  Programmability
  - +  Stored Procedures
  - +  Functions
  - +  Database Triggers
  - +  Assemblies
  - +  Types
  - +  Rules
  - +  Defaults
  - +  Plan Guides
  - +  Sequences
- +  Service Broker
- +  Storage
- +  Security
- +  Test

# Các đối tượng CSDL lập trình được

| Đối tượng        | Mô tả  |
|------------------|--|
| Stored procedure | Các thủ tục lưu trữ trong CSDL   |
| Function         | Các hàm trong CSDL   |
| Trigger          | Là dạng thủ tục đặc biệt, được thực hiện tự động khi có một sự kiện xảy ra |
| Assembly         | Cho phép đưa các assembly đã viết bằng C#, ... vào SQL Server              |
| Type             | Các kiểu dữ liệu tự tạo  |
| Rule             | Định nghĩa luật để kiểm tra các giá trị hợp lệ                             |
| Default          | Các giá trị mặc định nếu giá trị của một field không được nhập vào         |
| Sequences        | Bộ đếm tuần tự   |

# TABLE

# Khái niệm

- Table là đối tượng trong database dùng lưu trữ dữ liệu trong 1 bảng 2 chiều gồm nhiều hàng (rows/records) và nhiều cột (columns/fields)
- Trong SQL Server:
  - 1 table có thể có đến 1,024 cột
  - 1 database có thể chứa 2 tỷ tables
  - Kích thước tối đa cho 1 hàng là 8,060 bytes, trừ kiểu có chiều dài biến đổi (variable length)
  - Tổng số hàng và kích thước của table chỉ bị giới hạn bởi dung lượng đĩa

# Table

Xem lại môn Cơ sở dữ liệu. Các vấn đề quan tâm:

- Quy tắc đặt tên
- Kiểu dữ liệu
- Cài đặt ràng buộc dữ liệu
- Identity
- Các lệnh SQL liên quan đến Table

# Các loại Table đặc biệt

- Temporary Table.
  - Table sẽ tự động xóa bỏ khi connection kết thúc.
- Partitioned Table.
  - Các dòng dữ liệu được phân chia ra đặt trên nhiều filegroup nhằm tăng hiệu quả truy xuất.
- Wide Tables.
  - Sử dụng **sparse columns**, có thể tăng số lượng của table đến 30,000
- File Table
- External Table
- Graph Table

# Bảng tạm (temporary table)

- Nếu chỉ dùng table để lưu dữ liệu tạm thời trong quá trình xử lý, ta có thể tạo table tạm
- Cách tạo tương tự như tạo một table thông thường, chỉ khác là tên table bắt đầu bằng dấu # (Local) hoặc ## (Global)
- Khi kết thúc quá trình xử lý thì table này sẽ được tự động xóa đi
- Do không phải lưu trữ thật trong hệ thống, nên hệ thống không phải kiểm soát từ điển dữ liệu (data dictionary), quyền truy cập, ... → truy xuất nhanh

# Bảng tạm (temporary table)

- Cách tạo 1:

CREATE TABLE <TênTableTạm>  
(<khai báo các cột>)

- Ví dụ: CREATE TABLE #TongHop  
(TongThu int, TongChi int )
- Sau khi tạo xong ta có thể sử dụng bình thường như các table khác

```
SELECT * FROM #TongHop  
INSERT INTO #TongHop VALUES (500, 200)
```

- Table này chỉ tồn tại trong connection này thôi
- Có thể thấy table này trong database TempDB

# Bảng tạm (temporary table)

- Cách tạo 2:

- `SELECT <danh sách các cột>`

- `INTO <TênTableTạm>`

- `FROM ...`

- `WHERE ...`

- Ví dụ:

- `SELECT MSNV, HoTen, MSPB`

- `INTO #NVP1`

- `FROM NhanVien`

- `WHERE MSPB = 'P1'`

# Memory-Optimized Tables

- Nơi lưu trữ chính của memory-optimized tables là bộ nhớ chính (RAM). Bản copy dữ liệu được duy trì trên đĩa (đối với durable memory-optimized tables). Dữ liệu trong tables chỉ được đọc trong quá trình recovery (như sau khi server restart).
- CREATE TABLE dbo.ShoppingCart
  - ( ShoppingCartId INT IDENTITY(1,1) PRIMARY KEY,
  - UserId INT NOT NULL INDEX ix\_UserId,
  - CreatedDate DATETIME2 NOT NULL,
  - TotalPrice MONEY
- ) WITH (MEMORY\_OPTIMIZED=ON)

# VIEW

# Standard VIEW

- Có 2 loại view trong SQL Server: standard view và indexed view.
- **Standard view** là một bảng ảo (virtual table), bản thân **không chứa dữ liệu bên trong**, dữ liệu lấy ra từ kết quả của một câu SELECT
- Có thể sử dụng view giống như 1 table thật. Có thể dùng view trong câu query, trong stored procedure, hoặc trong view khác

# Indexed VIEW

- Indexed view giống standard view ngoại trừ việc nó có một unique clustered index được tạo ra. Điều này dẫn đến dữ liệu của view sẽ được lưu trữ vật lý trên đĩa, giống như table
- Dùng indexed view sẽ tốn kém đĩa và bảo trì khi dữ liệu trong table thay đổi
- Lý do để sử dụng indexed view là muốn có hiệu suất tốt hơn cho các câu query sử dụng view. Thường chỉ dùng khi dữ liệu trong table bên dưới ít thay đổi

# Ứng dụng của view

- Che dấu sự phức tạp của câu truy vấn đối với người dùng
- Che dấu các thông tin nhạy cảm
- Đổi tên cột cho phù hợp với yêu cầu
- Tạo ra các câu truy vấn trung gian, dùng làm nguồn dữ liệu cho các câu truy vấn kế
- Tạo ra nhiều góc nhìn khác nhau trên cùng một dữ liệu nguồn
- Giúp cho sự phân quyền uyển chuyển hơn

# Tạo VIEW

- Cú pháp:  
**CREATE VIEW** view\_name [(column [ ,...n ])]  
[WITH { [ENCRYPTION] [SCHEMABINDING] }  
**AS** select\_statement  
[WITH CHECK OPTION]
- ENCRYPTION: mã hóa source code của view. Để sửa đổi view phải xóa view và tạo lại.
- SCHEMABINDING: Kết các cột của view với schema của base table. Base table không thể sửa nếu làm ảnh hưởng đến định nghĩa view

# Tạo VIEW

- Câu SELECT trong lệnh định nghĩa view không được chứa:
  - Mệnh đề COMPUTE hay COMPUTE BY
  - Mệnh đề ORDER BY, trừ khi có từ khóa TOP trên mệnh đề SELECT
  - Từ khóa INTO
  - Tham chiếu đến bảng tạm (temporary table) hoặc biến table (table variable)

# Tạo VIEW

- Ví dụ:

```
CREATE VIEW vwNVP1  
AS  
    SELECT MSNV, HoTen, MSPB  
    FROM NhanVien  
    WHERE MSPB = 'P1'
```

- Sau khi view được tạo ra, ta có thể dùng lệnh SELECT để truy vấn dữ liệu như đối với một table bình thường
- Ví dụ: `SELECT * FROM vwNVP1`

# Thao tác trên VIEW

- Đối với một số view không mang tính tổng hợp, ... (thường có nguồn lấy từ 1 table) → ta có thể thêm, xóa, sửa dữ liệu của table nguồn thông qua view
- Ví dụ:

```
INSERT INTO vwNVP1 VALUES ('10', 'Le An', 'P1')
```

```
SELECT * FROM vwNVP1
```

```
UPDATE vwNVP1 SET HoTen = 'Tran Thi Thu'
```

```
    WHERE MSNV = '10'
```

```
DELETE vwNVP1 WHERE MSNV = '10'
```

# WITH CHECK OPTION

- Giả sử ta thực hiện lệnh sau:

```
INSERT INTO vwNVP1 VALUES ('11', 'Le Minh', 'P2')
```

- Việc thực hiện vẫn diễn ra bình thường. Tuy nhiên khi dùng câu truy vấn

```
SELECT * FROM vwNVP1
```

- Ta không thấy nhân viên này. Nhưng nếu kiểm tra lại table bằng câu truy vấn

```
SELECT * FROM NhanVien
```

- Thì ta thấy nhân viên này đã thêm vào table

# WITH CHECK OPTION

- Lý do: Dữ liệu đưa vào view lại nằm ngoài tầm vực của mệnh đề WHERE trong câu SELECT định nghĩa view
- Nếu ta có mệnh đề WITH CHECK OPTION trong câu lệnh tạo view.
- Khi thực hiện câu lệnh  
`INSERT INTO vwNVP1  
VALUES ('12', 'Huynh Thi Nga', 'P2')`
- Ta sẽ nhận được thông báo lỗi, do mẫu tin này nằm ngoài tầm của view

# Một số ứng dụng

- VD: Tìm các nhân viên có lương cao nhất ở mỗi phòng
- Tạo view vwLgMax để tính lương cao nhất ở mỗi phòng

```
CREATE VIEW vwLgMax  
AS SELECT MSPB, Max(Luong) AS LgMax  
FROM NhanVien  
GROUP BY MSPB
```

- Kết table NhanVien với view này để lấy ra kết quả

# Indexed view

- Để tạo indexed view, trong câu CREATE VIEW ta dùng SCHEMABINDING option.
- Sau đó tạo unique clustered index
- Ví dụ:

```
CREATE TABLE T1 (F1 int, F2 int)
```

```
CREATE VIEW V1 WITH SCHEMABINDING AS
```

```
    SELECT F1, F2 FROM dbo.T1
```

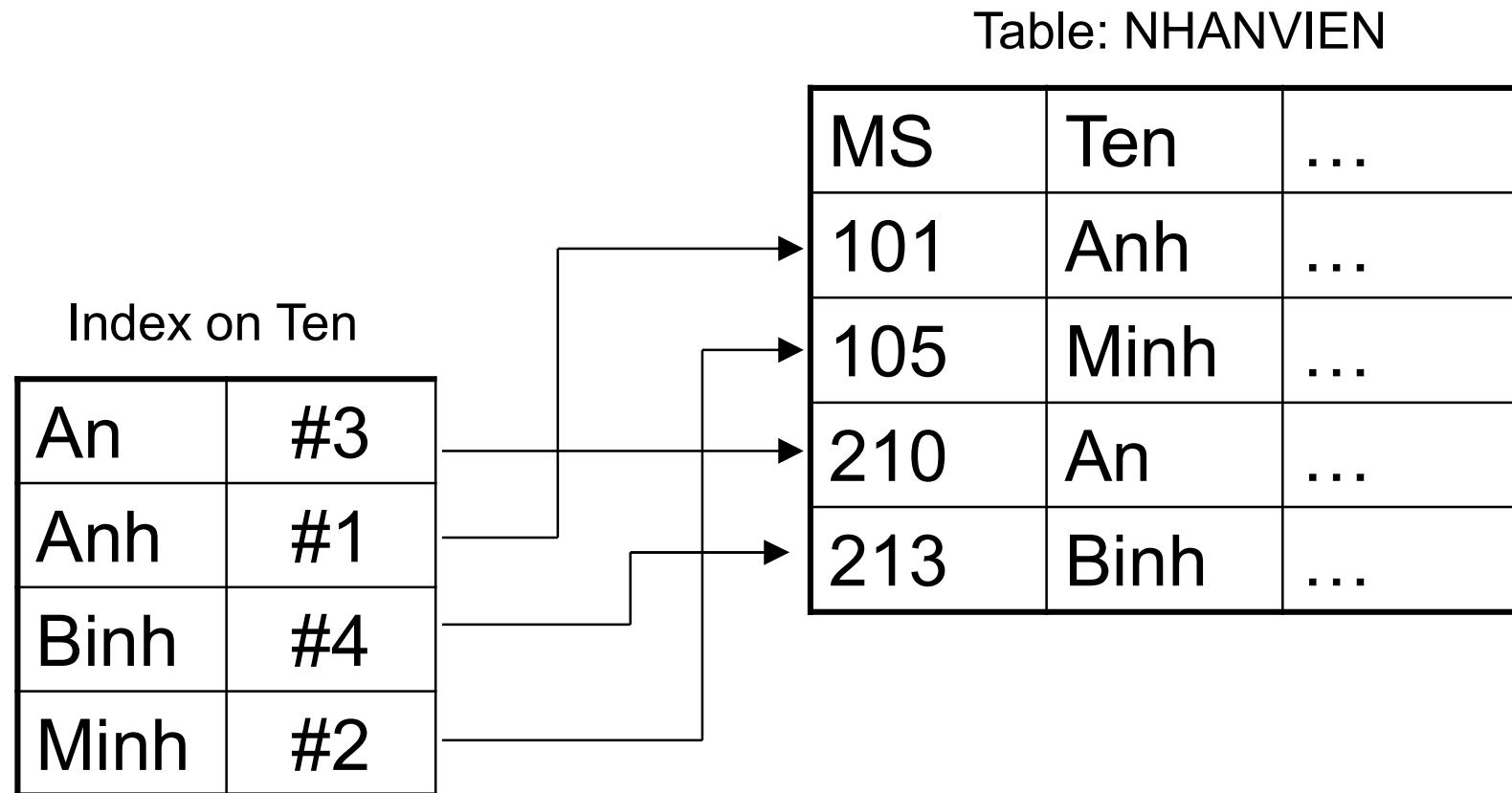
```
CREATE UNIQUE CLUSTERED INDEX idx1  
    ON V1(F1)
```

# INDEX

# Index

- Index giống như chỉ mục trong một quyển sách, nó giúp ta tìm kiếm thông tin một cách nhanh chóng.
- Index là một đối tượng trong database. Đây là một file với các index entries có dạng  
 $\langle \text{field value}, \text{pointer to record} \rangle$   
trong đó cột field value được sắp xếp
- Index được gọi là đường truy cập trên field
- Thường cài đặt theo cấu trúc của B-Tree

# Index



# Ví dụ

- Cho tập tin dữ liệu
  - EMPLOYEE(NAME, SSN, ADDRESS, JOB, SAL, ... )
- Biết rằng
  - Kích thước mẫu tin:  $R = 250$  bytes
  - Kích thước field SSN:  $V_{SSN} = 9$  bytes
  - Kích thước con trỏ mẫu tin:  $PR = 7$  bytes
  - Kích thước khối (block):  $B = 512$  bytes
  - Số lượng mẫu tin:  $r = 30,000$  records
  - Giả sử chỉ mục dày đã lập cho field SSN

# Ví dụ

- Tập tin dữ liệu

- Hệ số phân khối (blocking factor)

$$Bfr = \lfloor B/R \rfloor = \lfloor 512/250 \rfloor = 2 \text{ records/block}$$

- Số block sử dụng cho tập tin

$$b = \lceil r/Bfr \rceil = \lceil 30,000/2 \rceil = 15,000 \text{ blocks}$$

# Ví dụ

- Tập tin chỉ mục

- Kích thước của index entry

$$R_i = (V_{SSN} + PR) = (9+7) = 16 \text{ bytes}$$

- Hệ số phân khối của tập tin chỉ mục

$$Bfr_i = \lfloor B/R_i \rfloor = \lfloor 512/16 \rfloor = 32 \text{ entries/block}$$

- Số blocks sử dụng cho tập tin chỉ mục

$$b_i = \lceil r/Bfr_i \rceil = \lceil 30,000/32 \rceil = 938 \text{ blocks}$$

# Ví dụ

- Chi phí truy cập dùng tập tin chỉ mục
  - Tìm nhị phân:  $\lceil \log_2 b \rceil = \lceil \log_2 938 \rceil = 10$
  - Sau đó thông qua con trỏ mẫu tin truy cập thêm 1 lần tập tin dữ liệu để lấy mẫu tin cần tìm
  - Tổng cộng là **11** lần truy cập khối
- Chi phí truy cập không dùng tập tin chỉ mục
  - Tìm tuần tự:  $\lceil b/2 \rceil = \lceil 15,000 / 2 \rceil = 7,500$
  - Nếu tập tin có sắp thứ tự theo field SSN thì có thể dùng phép tìm nhị phân với chi phí:  
 $\lceil \log_2 b \rceil = \lceil \log_2 15000 \rceil = 14$

# Index

Khi tìm kiếm giá trị trên 1 table. VD:

```
SELECT * FROM NhanVien  
WHERE Ten = 'Binh'
```

Nếu trường tìm kiếm (Ten trong ví dụ này) KHÔNG có lập chỉ mục → hệ thống phải duyệt tuần tự qua tất cả các mẫu tin

Nếu trường tìm kiếm CÓ chỉ mục → hệ thống sẽ tự động dùng index với các giải thuật (VD: tìm kiếm nhị phân, ...) để duyệt → tăng hiệu suất truy vấn

→ **Dữ liệu càng lớn thì việc tìm kiếm với index càng hiệu quả**

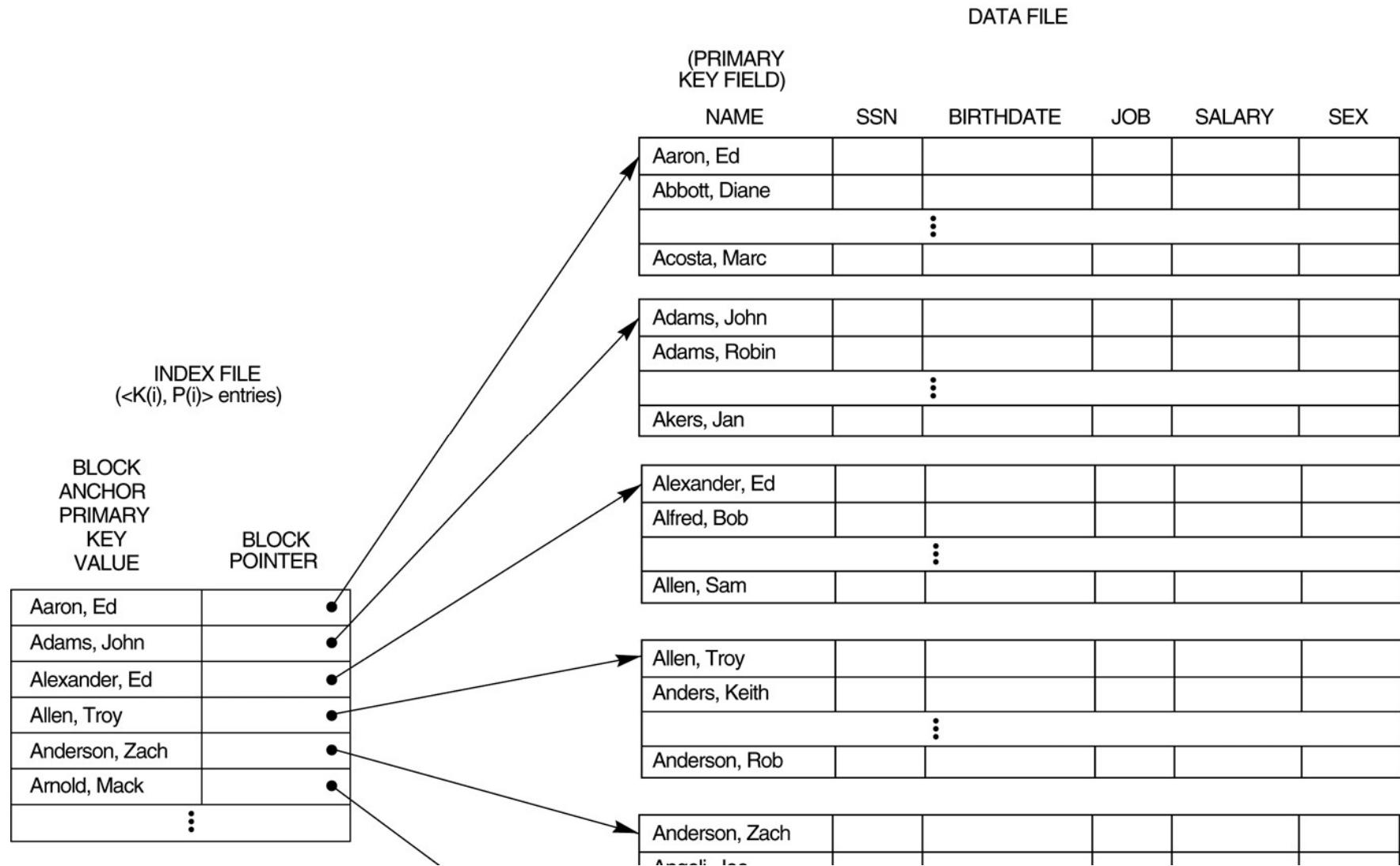
# Các loại Index

- **Dense index** có index entry cho mỗi giá trị tìm kiếm (tức tất cả mẫu tin) trong data file
- **Sparse (nondense) index**, chỉ có index entries cho một số giá trị tìm kiếm
- Single-level Ordered Indexes
  - Primary Indexes
  - Clustering Indexes
  - Secondary Indexes
- Multilevel Indexes
- Indexes on Multiple Keys
- ...

# Primary Index

- Tạo trên data file có **sắp xếp** trên key field
- Bao gồm 1 index entry cho mỗi block trong data file.
- Index entry có giá trị key field cho mẫu tin đầu tiên trong block, gọi là **block anchor**
- Mô hình tương tự có thể sử dụng mẫu tin cuối cùng trong block
- Primary index là nondense (sparse) index

# Primary index

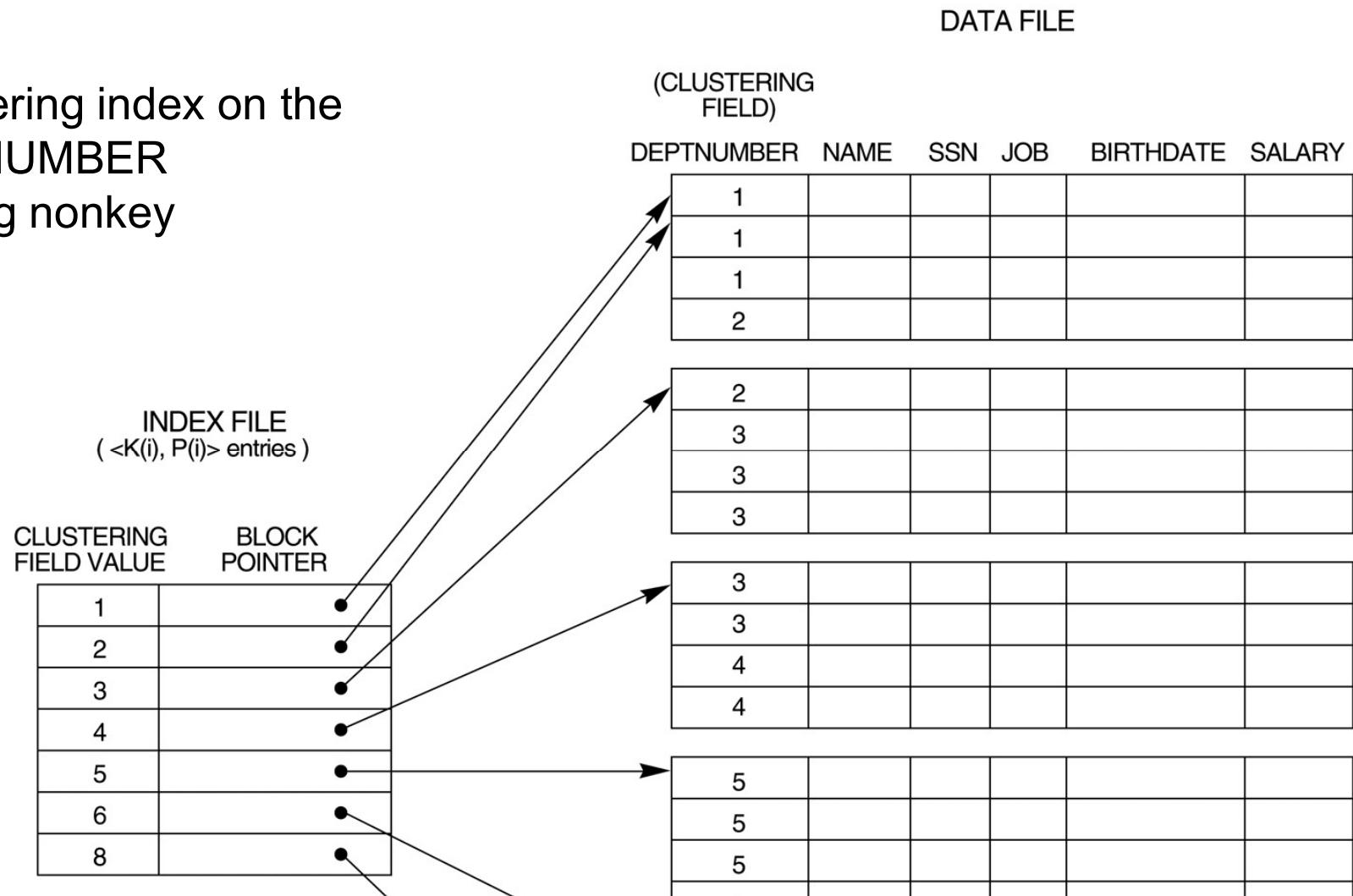


# Clustering Index

- Cũng xây dựng trên data file có sắp xếp.
- Data file được **sắp xếp trên non-key field** nên không yêu cầu giá trị phân biệt trên field sắp xếp trong data file
- Chứa 1 index entry cho mỗi giá trị phân biệt của field; index entry trả đến data block đầu tiên có mẩu tin chứa giá trị này
- Đây là trường hợp nondense index

# Clustering index

A clustering index on the  
DEPTNUMBER  
ordering nonkey

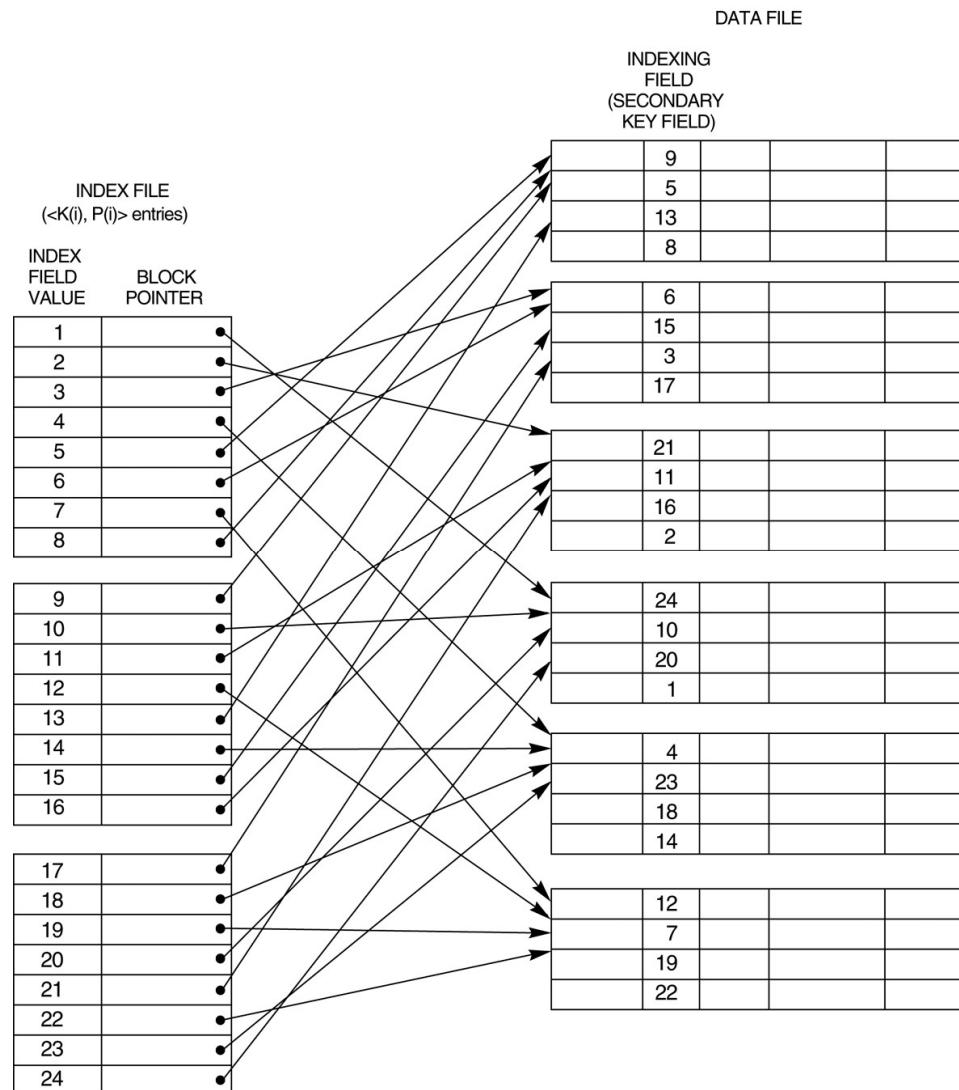


# Secondary Index

- Cung cấp phương tiện để truy cập file đã tồn tại primary index.
- Index là tập tin có thứ tự với 2 fields:
  - Field đầu chứa cùng giá trị nonordering field của data file, gọi là **indexing field**.
  - Field thứ 2 là **block pointer** hoặc **record pointer**
- Có thể có nhiều secondary index trên cùng data file
- Chứa một entry cho mỗi record trong data file; vì thế đây là dense index

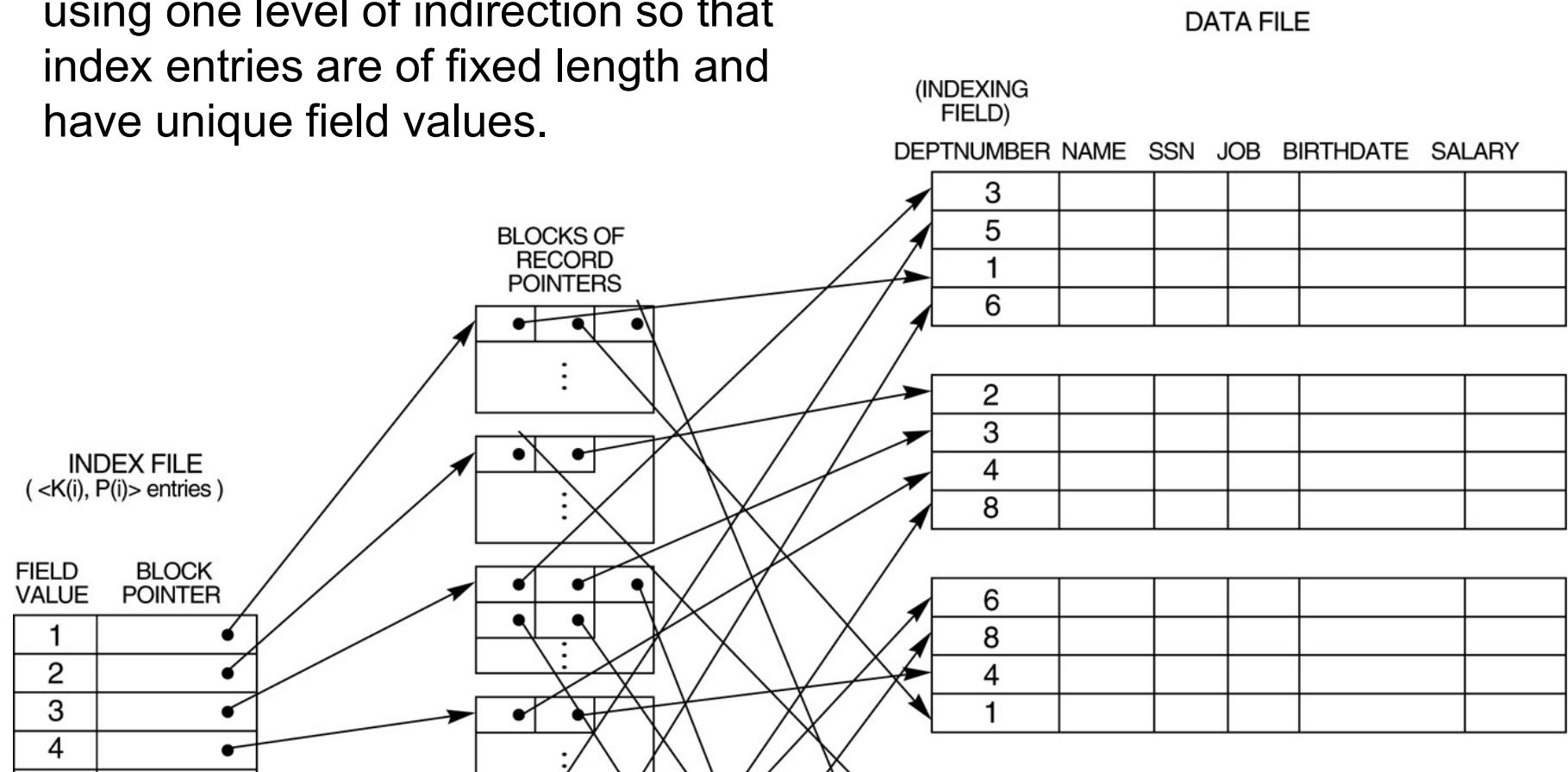
# A dense secondary index

A dense secondary index (with block pointers) on a nonordering key field of a file



# A dense secondary index

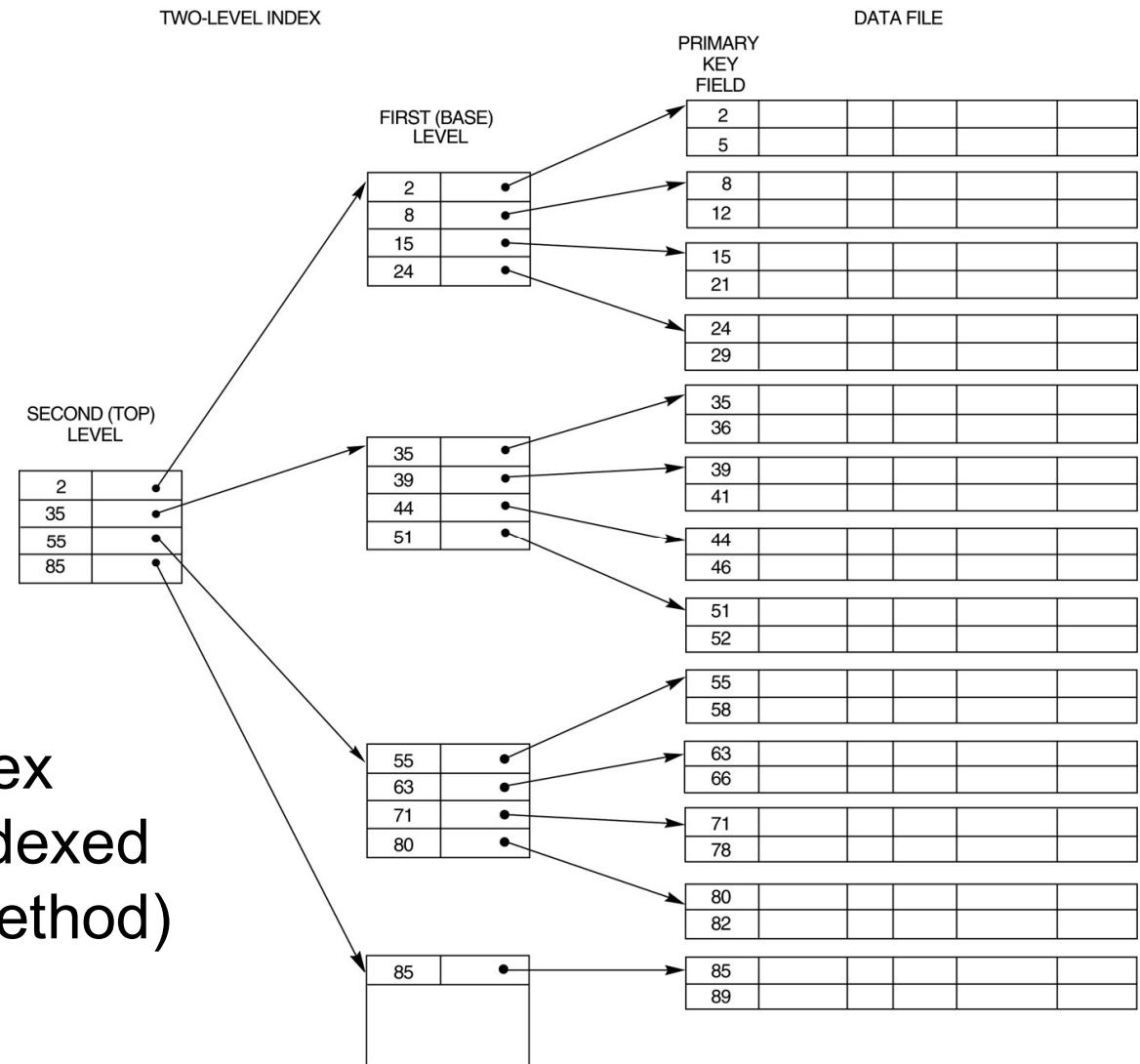
A secondary index (with record pointers) on a nonkey field implemented using one level of indirection so that index entries are of fixed length and have unique field values.



# Multi-Level Indexes

- Vì single-level index là tập tin có thứ tự, ta có thể tạo 1 primary index cho chính file index; trong trường hợp này index file gốc gọi là first-level index và index cho file này gọi là second-level index.
- Ta có thể lặp lại quá trình này, tạo ra third, fourth, ..., top level cho đến khi tất cả entries của top level vừa chứa đủ trong 1 disk block
- Multi-level index có thể tạo cho bất kỳ loại first-level index nào (primary, secondary, clustering)

# A two-level primary index



two-level primary index  
resembling ISAM (Indexed  
Sequential Access Method)  
organization.

# Lệnh Create Index

CREATE

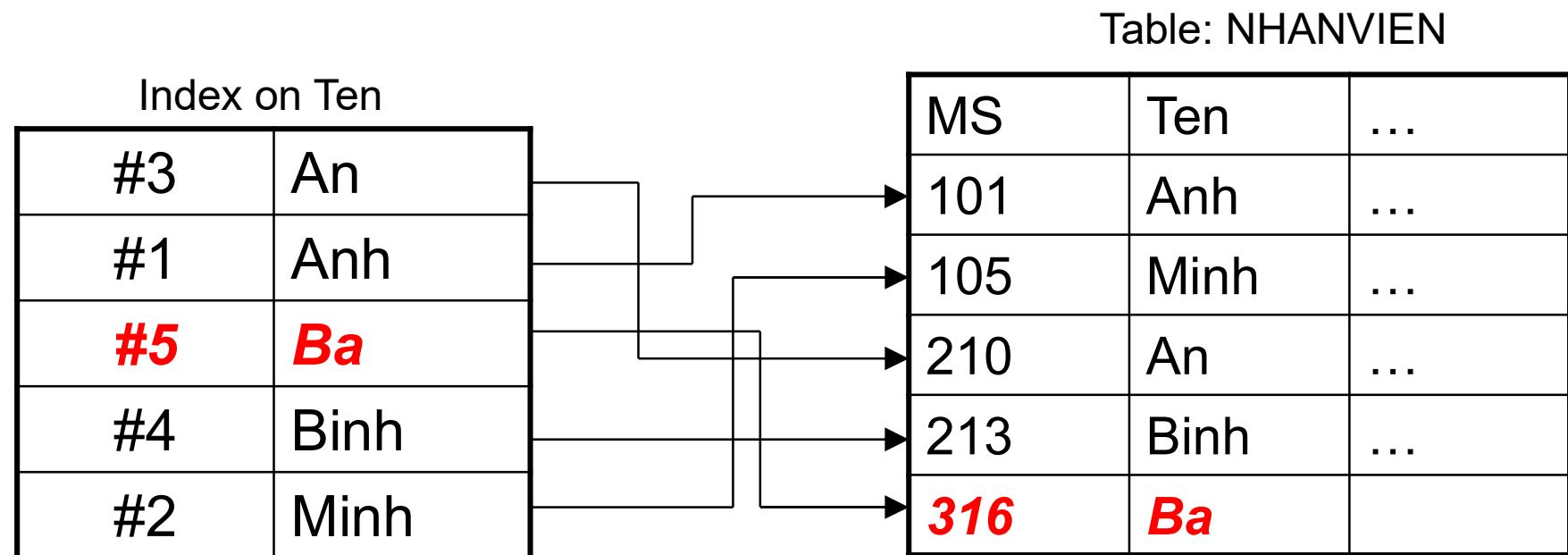
[UNIQUE] [CLUSTERED | NONCLUSTERED ]  
INDEX <tên index>  
ON <table/view> (<cột> [ ASC | DESC ] [ ,...n ] )

Ví dụ:

CREATE INDEX idxLuong  
ON NhanVien (Luong DESC)

# Vấn đề Reindex

- Index thường được bảo trì tự động. Nên mỗi cập nhật (thêm, xóa, sửa) dữ liệu, hệ thống sẽ phải cập nhật lại index → chậm hệ thống



# Các đối tượng CSDL khác

# Type (kiểu tự định nghĩa)

- Ta có thể tự tạo kiểu dữ liệu mới dựa trên các kiểu SQL đã cung cấp sẵn
- Cú pháp:

Create Type TênKiểuMới From KiểuGốc

- Ví dụ:

Create type KieuHoTen from nvarchar(30)

Create table SinhVien (

    MSSV int primary key,

    HoTen kieuhoten

)

# Rule (luật kiểm tra ràng buộc)

- Rule được tạo như một đối tượng độc lập,sau đó mới thiết đặt cho cột dữ liệu
- Tạo rule:

**CREATE RULE** range\_rule

AS @range>= \$1000 AND @range <\$20000;

- Áp dụng quy tắc kiểm tra vào table

**EXEC sp\_bindrule** <tên rule>, Têntable.Têncột

# Synonym (tên tương đương)

- Synonym là một đối tượng CSDL nhằm mục đích tạo ra một tên tương đương cho một đối tượng CSDL khác (trên cùng server hoặc trên server khác).
- Cú pháp lệnh tạo Synonym:

**CREATE SYNONYM** TênSynonym

**FOR** TênGốc

# Synonym

- Ví dụ ta đã có table NhanVien
- Tạo Synonym NV thay thế cho NhanVien

**Create synonym nv for NhanVien**

- Khi đó ta dùng NV hay NhanVien trong các câu SQL là tương đương nhau

Select \* from nv

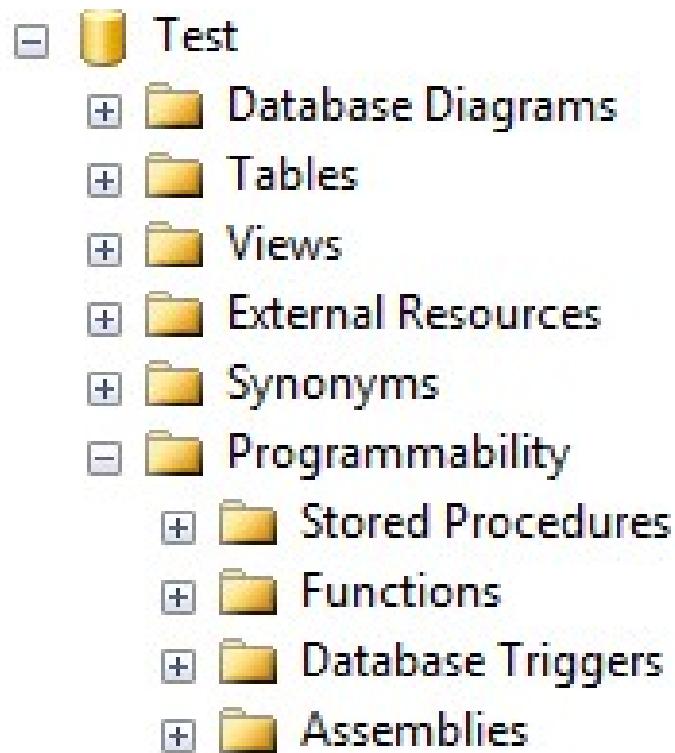
Select \* from nhanvien

# Sequence (bộ đếm)

- CREATE SEQUENCE PetIDSequence  
AS INT  
START WITH 1  
INCREMENT BY 1
- CREATE TABLE PetFood  
(pet\_food int DEFAULT  
(NEXT VALUE FOR PetIDSequence),  
food\_name VARCHAR (20));

# Programmability

- Các đối tượng có thể lập trình được như: Stored Procedures, Functions, Triggers, ... sẽ tìm hiểu trong chương sau



# Tóm tắt

- Database
- Các loại đối tượng cơ bản trong SQL Server
- Table, Temporary table
- View
- Index
- Giới thiệu một số đối tượng CSDL khác

# Chương 4

# Quản trị các đối tượng khả lập trình



Môn: QUẢN TRỊ CƠ SỞ DỮ LIỆU



# Nội dung

- Ngôn ngữ T-SQL
- Cursor
- Stored-Procedure
- Function
- Trigger

# Ngôn ngữ T-SQL

# Ngôn ngữ SQL

## Câu lệnh SELECT

SELECT      <danh\_sách\_các\_cột>  
FROM          <danh\_sách\_các\_table>  
WHERE        <điều\_kiện\_lọc\_trên\_mẫu\_tin>  
GROUP BY     <các\_biểu\_thúc\_nhóm>  
HAVING       <điều\_kiện\_lọc\_trên\_nhóm>  
ORDER BY     <các\_biểu\_thúc\_sắp\_xếp>

(Xem lại phần SQL trong môn cơ sở dữ liệu)

# Transact-SQL (T-SQL)

- Transact-SQL là ngôn ngữ SQL mở rộng dựa trên SQL chuẩn của **ISO** (International Organization for Standardization) và **ANSI** (American National Standards Institute)
- T-SQL cho phép ta sử dụng các lệnh SQL chuẩn kết hợp với các cấu trúc điều khiển để lập trình trên SQL Server

# Khai báo biến

- Cú pháp:

DECLARE @<Tên\_bien> <kiểu dữ liệu> [, ...]

- Trong đó:

Tên\_bien: luôn luôn bắt đầu bằng @

Kiểu\_dữ\_liệu: là kiểu cơ bản hoặc do người dùng tự định nghĩa. (Các kiểu text, ntext, image không thể dùng để khai báo biến)

- Ví dụ:

DECLARE @SoLuong Int, @HoTen Varchar(50)

DECLARE @NgaySinh DateTime

# Các kiểu dữ liệu cơ bản

| Tên kiểu            | Loại         | Miền giá trị                     |
|---------------------|--------------|----------------------------------|
| Int                 | Số nguyên    | - 2 tỷ → 2 tỷ                    |
| Smallint            | Số nguyên    | -32768 → 32767                   |
| Tinyint             | Số nguyên    | 0 → 255                          |
| Bit                 | Số nguyên    | 0, 1 hoặc Null                   |
| Decimal,<br>Numeric | Số thập phân | Từ -10^38 → + 10^38 (17 bytes)   |
| Float               | Số thực      | -1.79E+38 → +1.79E+38            |
| Real                | Số thực      | -1.79E+308 → +1.79E+308          |
| Char                | Chuỗi        | 1 → 8000 ký tự (Fixed Length)    |
| Varchar             | Chuỗi        | 1 → 8000 ký tự (Variable Length) |
| Text                | Chuỗi        | 1 → 2 tỷ ký tự                   |

# Các kiểu dữ liệu cơ bản

| Tên kiểu      | Loại              | Miền giá trị                       |
|---------------|-------------------|------------------------------------|
| Nchar         | Chuỗi (Unicode)   | 1 → 4000 ký tự (mỗi ký tự 2 bytes) |
| Nvarchar      | Chuỗi (Unicode)   | 1 → 4000 ký tự                     |
| Ntext         | Chuỗi (Unicode)   | 1 → 1 tỷ ký tự                     |
| Money         | Số (dạng tiền tệ) | - 900 ngàn tỷ → + 900 ngàn tỷ      |
| Smallmoney    | Số (dạng tiền tệ) | - 900 ngàn tỷ → + 900 ngàn tỷ      |
| Datetime      | Ngày giờ          | 01/01/1753 → 31/12/9999            |
| Smalldatetime | Ngày giờ          | 01/01/1900 → 06/06/2079            |
| Binary        | Chuỗi nhị phân    | 1 → 8000 bytes                     |
| Varnbinary    | Chuỗi nhị phân    | 1 → 8000 bytes                     |
| Image         | Chuỗi nhị phân    | 1 → 2 tỷ bytes                     |

# Hàm Datalength()

- Mỗi kiểu dữ liệu có đặc trưng và chiếm một số byte trong bộ nhớ
- Sử dụng hàm DATALENGTH để xác định số byte của một biến
- Ví dụ:

```
DECLARE @name VARCHAR(20)
```

```
SET @name = 'STU'
```

```
SELECT DATALENGTH(@name)
```

# Gán giá trị cho biến

- Cú pháp:

SET @<Tên\_variabile> = <giá\_trị>

Hoặc

SELECT @<tên\_variabile> = <giá\_trị>, ...

FROM ...

- Ví dụ:

```
DECLARE @TongSL Int, @HoTen Varchar(50)
```

```
SET @HoTen = 'Le An'
```

```
SELECT @TongSL = Sum(Luong)
```

```
FROM NhanVien
```

# In nội dung biến ra màn hình

- Cú pháp:

PRINT @<Tên\_variabile> | <bíểu thức chuỗi>

- Ghi chú:

Có thể dùng hàm CAST hoặc CONVERT để chuyển đổi kiểu dữ liệu, trong trường hợp cần đưa vào bíểu thức chuỗi

- Ví dụ:

```
DECLARE @NS datetime
```

```
set @ns = '10/5/2006'
```

```
print 'Thang: ' + convert(char(2), month(@ns))
```

# Biến hệ thống

- Do SQL Server cung cấp sẵn, người lập trình không thể can thiệp để gán giá trị. Các biến này nhằm cung cấp thông tin của hệ thống, giúp người lập trình xử lý tiếp theo
- Tên biến bắt đầu bằng @@
- Ví dụ: biến @@rowcount  
Trả về số lượng mẫu tin lấy ra được trong câu lệnh truy vấn gần nhất.

```
SELECT * FROM nhanvien WHERE luong < 500
```

```
SELECT @@rowcount
```

# Một số biến hệ thống khác

- @@ServerName → tên máy chủ cục bộ đang thực hiện lệnh
- @@ServiceName → tên dịch vụ trên máy chủ cục bộ
- @@version → phiên bản SQL server
- @@language → ngôn ngữ sử dụng
- @@error → số mã lỗi của câu lệnh gần nhất. Nếu thực hiện thành công thì biến bằng 0

# Cấu trúc IF ... ELSE

- Cú pháp:

IF <biểu thức logic>

<Lệnh1>

[ ELSE

<Lệnh2> ]

- Ghi chú:

- Nếu Lệnh1 hoặc Lệnh2 gồm nhiều lệnh thì phải kẹp giữa cặp từ khóa BEGIN ... END

- Có thể dùng IF EXISTS (<câu SQL>) để kiểm tra có tồn tại ít nhất một dòng kết quả hay không?

# Ví dụ

Ví dụ 1:

```
IF (SELECT Sum(Luong) FROM NhanVien) > 1000  
    PRINT 'tong luong tren 1000'  
ELSE  
    PRINT 'tong luong it'
```

Ví dụ 2:

```
IF EXISTS (SELECT * FROM NhanVien WHERE Luong < 500)  
BEGIN  
    PRINT 'Co nguoi luong duoi 500'  
    PRINT 'Tiep tuc ...'  
END
```

# Cấu trúc CASE

- Không phải là cấu trúc điều khiển, tức là nó chỉ lồng vào các câu lệnh khác, mà không thể thực hiện đơn lẻ như các cấu trúc điều khiển khác
- Cú pháp 1:  
CASE <biểu thức>  
    WHEN <giá trị 1> THEN <kết quả 1>  
    WHEN <giá trị 2> THEN <kết quả 2>  
    ...  
    [ ELSE kết quả n ]  
END

# Cấu trúc CASE

- Cú pháp 2:

CASE

    WHEN <bt logic 1> THEN <kết quả 1>

    WHEN <bt logic 2> THEN <kết quả 2>

    ...

    [ ELSE kết quả n ]

END

# Ví dụ

```
SELECT MSNV, HoTen,  
      GTinh = CASE phai  
        WHEN 'True' THEN 'Nam'  
        WHEN 'False' THEN 'Nu'  
      END  
FROM NhanVien
```

---

```
SELECT MSNV, HoTen,  
      Loai = CASE  
        WHEN Luong >= 500 THEN 'cao'  
        WHEN Luong < 500 THEN 'thap'  
      END  
FROM NhanVien
```

---

# Cấu trúc WHILE

- Cú pháp:

WHILE <biểu\_thức logic>

BEGIN

<Các lệnh trong vòng lặp>

END

- Ghi chú:

- Phát biểu BREAK trong vòng lặp sẽ kết thúc vòng lặp mà không cần chờ điều kiện lặp bị sai
- Phát biểu CONTINUE sẽ bỏ qua các lệnh còn lại trong lần lặp đó để quay về đầu vòng lặp mới

# Các hàm chuyển đổi kiểu

- Chuyển kết quả của biểu thức sang một kiểu dữ liệu bất kỳ
- Hàm **CAST**

CAST (biểu\_thức AS kiểu\_dữ\_liệu)

Ví dụ: CAST (luong AS varchar(10))

- Hàm **CONVERT**

CONVERT (kiểu\_dữ\_liệu, biểu\_thức, [, định\_dạng])

Ví dụ: CONVERT (char(10), NgaySinh, 102)

# Định dạng

| Without century (yy) | With century (yyyy) | Standard        | Input/Output                |
|----------------------|---------------------|-----------------|-----------------------------|
| -                    | 0 or 100            | Default         | mon dd yyyy hh:miAM (or PM) |
| 1                    | 101                 | U.S.            | mm/dd/yyyy                  |
| 2                    | 102                 | ANSI            | yy.mm.dd                    |
| 3                    | 103                 | British /French | dd/mm/yy                    |
| 4                    | 104                 | German          | dd.mm.yy                    |
| 5                    | 105                 | Italian         | dd-mm-yy                    |
| 6                    | 106                 | -               | dd mon yy                   |
| 7                    | 107                 | -               | Mon dd, yy                  |
| ...                  | ...                 | ...             | ...                         |

# Các hàm kiểm tra kiểu dữ liệu

## ■ Hàm ISDATE

Kiểm tra biểu thức ngày tháng có hợp lệ không?  
Kết quả trả về:

- 0: không hợp lệ
- 1: hợp lệ

Cú pháp:

ISDATE (biểu thức)

Ví dụ:

Print IsDate('5/25/2006') → 1

Print IsDate('15/13/2006') → 0

# Các hàm kiểm tra kiểu dữ liệu

## ■ Hàm ISNUMERIC

Kiểm tra biểu thức có là một số không? Kết quả:

- 0: không phải là một số
- 1: biểu thức là một số

Cú pháp:

ISNUMERIC (biểuthức)

Ví dụ:

Print IsNumeric('1234') → 1

Print IsNumeric('123 Le Loi') → 0

# Các hàm kiểm tra kiểu dữ liệu

## ■ Hàm ISNULL

Kiểm tra biểu thức có rỗng hay không?

- Nếu không rỗng trả về biểu thức
- Nếu rỗng sẽ thay thế bằng giá trị thay thế

Cú pháp:

ISNULL (biểu thức, giá trị thay thế)

Ví dụ:

PrintIsNull ('1234', 'abc') → '1234'

PrintIsNull (Null, 'xyz') → 'xyz'

# Các hàm thời gian

## ■ Hàm GETDATE

Trả về ngày giờ hiện hành của hệ thống

Cú pháp: GETDATE()

Ví dụ: SELECT Convert(Char(20), GetDate(), 13)

## ■ Các hàm DAY, MONTH, YEAR

Trả về phần ngày, tháng, năm. Kết quả trả về là một số nguyên

Cú pháp:

DAY(ngày) , MONTH(ngày), YEAR(ngày)

# Các hàm thời gian

## ■ Hàm DATEPART

Lấy ra một phần trong giá trị kiểu ngày tháng

Cú pháp:

DATEPART (datepart, date)

Ví dụ:

Print GetDate() → Oct 21 2006 6:31AM

Print DatePart(d, GetDate()) → 21

Print Datepart(dw, '25-dec-2006') → 2

# datepart

| Datepart    | Abbreviations |
|-------------|---------------|
| year        | yy, yyyy      |
| quarter     | qq, q         |
| month       | mm, m         |
| dayofyear   | dy, y         |
| day         | dd, d         |
| week        | wk, ww        |
| weekday     | dw            |
| hour        | hh            |
| minute      | mi, n         |
| second      | ss, s         |
| millisecond | Ms            |

## weekday (dw)

Trả về một con số  
chỉ thứ trong tuần  
VD: Sunday = 1,  
Saturday = 7.

# Các hàm toán học

## ■ Hàm ABS

Trả về giá trị tuyệt đối của một số

Cú pháp: ABS (số)

Ví dụ: SELECT ABS (-105) → 105

## ■ Hàm POWER

Tính lũy thừa của một số

Cú pháp: POWER (số, số mũ)

Ví dụ: SELECT POWER ( 2, 3 ) → 8 (=  $2^3$ )

# Các hàm toán học

- Hàm **SIN, COS, TAN, COT**

Các hàm lượng giác

Cú pháp: SIN (số), COS (số), TAN (số)

- Hàm **ASIN, ACOS, ATAN**

Các hàm lượng giác ngược

Cú pháp: ASIN (số), ACOS (số), ATAN (số)

# Các hàm toán học

## ■ Hàm PI

Trả về số PI

Cú pháp: PI()

## ■ Hàm SQRT

Lấy căn bậc hai của một số

Cú pháp: SQRT (số)

Ví dụ: SELECT SQRT ( 25 ) → 5

# Các hàm xử lý chuỗi

## ■ Hàm STR

Chuyển một số sang kiểu chuỗi

Cú pháp: STR (sốthực, sốkýsố [, sốsốlẻ])

Ví dụ: STR (PCCV, 6, 2)

## ■ Hàm LEN

Trả về chiều dài của một chuỗi

Cú pháp: LEN (chuỗi)

Ví dụ: LEN ('Hello') → 5

# Các hàm xử lý chuỗi

## ■ Các hàm UPPER, LOWER

Hàm UPPER đổi một chuỗi thành chữ in hoa

Hàm LOWER đổi một chuỗi thành chữ thường

Cú pháp: UPPER (chuỗi) , LOWER (chuỗi)

Ví dụ: SELECT Upper( 'hELLo' ), Lower('hELLo' )

## ■ Hàm REVERSE

Đảo ngược một chuỗi

Cú pháp: REVERSE (chuỗi)

Ví dụ: REVERSE ('HELLO') → 'OLLEH'

# Các hàm xử lý chuỗi

## ■ Các hàm LEFT, RIGHT

Hàm LEFT lấy n ký tự bên trái của một chuỗi

Hàm RIGHT lấy n ký tự bên phải của chuỗi

Cú pháp:

LEFT (chuỗi, số ký tự)

RIGHT (chuỗi, số ký tự)

Ví dụ:

LEFT ('Hello World', 5) → 'Hello'

RIGHT ('Hello World', 5) → 'World'

# Các hàm xử lý chuỗi

## ■ Hàm **SUBSTRING**

Trích lấy chuỗi con trong một chuỗi

Cú pháp: SUBSTRING (chuỗi, vị trí, số ký tự)

Ví dụ: SUBSTRING ('Hello The World', 7, 3) → 'The'

## ■ Hàm **PATINDEX**

Trả về vị trí đầu tiên tìm thấy chuỗi con trong một chuỗi lớn

Cú pháp: PATINDEX ('%chuỗicon%', chuỗi lớn )

Ví dụ: PATINDEX ('%e%', 'Hello') → 2

# Các hàm xử lý chuỗi

## ■ Hàm **REPLICATE**

Lặp lại một chuỗi nhiều lần

Cú pháp: REPLICATE (chuỗi , sốlầnlặp)

Ví dụ: REPLICATE ('Hello', 3) → 'HelloHelloHello'

## ■ Hàm **SPACE**

Trả về n khoảng trắng

Cú pháp: SPACE (n)

Ví dụ: Print 'Hello' + space (3) + 'World'  
→ 'Hello\_ \_ \_ World'

# Các hàm xử lý chuỗi

## ■ Các hàm LTRIM, RTRIM

Loại bỏ các ký tự trắng vô nghĩa

Cú pháp:

LTRIM (chuỗi) → bỏ các ký tự trắng bên trái chuỗi

RTRIM (chuỗi) → bỏ các ký tự trắng bên phải chuỗi

Ví dụ:

Print 'a' + LTrim(' \_ \_Hello\_ \_') + 'b' → 'aHello\_ \_ b'

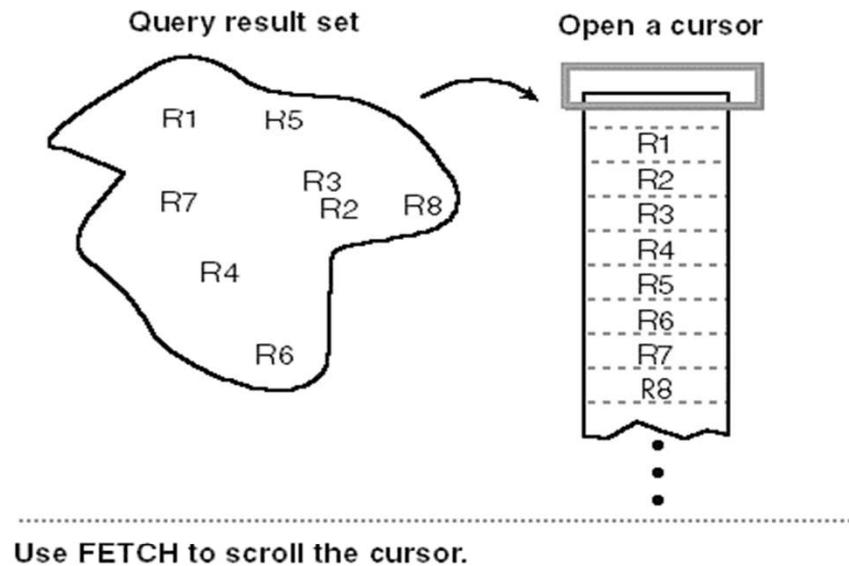
Print 'a' + Rtrim(' \_ \_Hello\_ \_) + 'b' → 'a \_ \_Hello b'

# Cursor

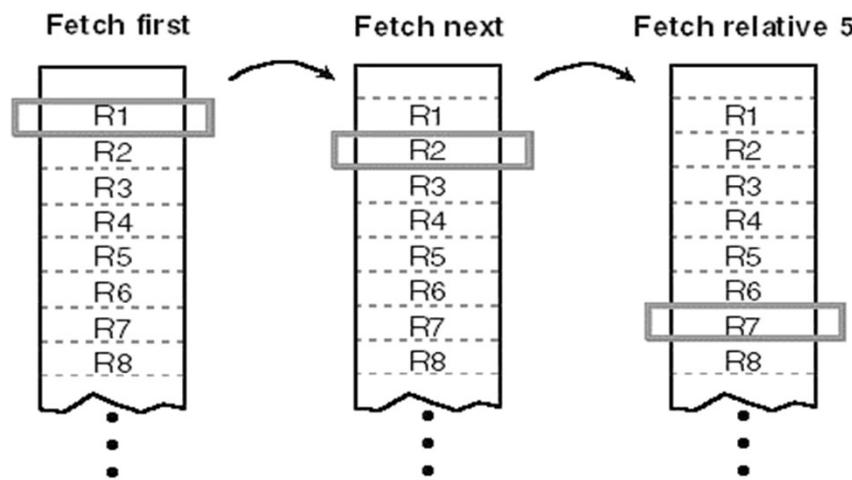
# Khái niệm cơ bản

- Cursor là kiểu dữ liệu dùng để lấy một tập các mẫu tin từ table hay thông qua một câu SELECT SQL
- Sử dụng một cursor thường qua các bước sau:
  - Khai báo cursor
  - Mở cursor
  - Duyệt và xử lý các mẫu tin của cursor
  - Đóng cursor
  - Xóa bỏ cursor

# Sử dụng cursor



Use **FETCH** to scroll the cursor.



# Khai báo và mở cursor

## ■ Khai báo cursor

Cú pháp:

DECLARE tên\_cursor

**CURSOR FOR** câu lệnh select

Ví dụ: DECLARE C1 CURSOR FOR  
SELECT MSNV, Hoten FROM nhanVien

## ■ Mở cursor:

Cú pháp:

**OPEN** tên\_cursor

Ví dụ: OPEN C1

# Duyệt các mẫu tin trong cursor

- Cú pháp:

```
FETCH [vi_trí FROM] tên_cursor  
[INTO @v1, @v2, ...]
```

Vi\_trí: dùng một trong các từ khóa sau: NEXT, PRIOR, FIRST, LAST, ABSOLUTE n, RELATIVE n

- Kiểm tra trạng thái của lệnh Fetch:

Dùng hàm hệ thống **@@Fetch\_status**

- 0: lấy mẫu tin thành công
- Khác 0: lấy mẫu tin thất bại

- Dùng vòng lặp để duyệt các mẫu tin trong cursor

# Đóng và xóa cursor

- Đóng cursor

CLOSE tên\_cursor

- Xóa bỏ cursor

DEALLOCATE tên\_cursor

# Ví dụ

```
DECLARE @MS varchar(5), @Ten varchar(20)
DECLARE C1 CURSOR FOR
    SELECT MSNV, Hoten FROM nhanVien
OPEN C1
FETCH C1 INTO @MS, @ten
WHILE (@@fetch_status = 0)
BEGIN
    PRINT @MS + @ten
    FETCH C1 INTO @MS, @ten
END
CLOSE C1
DEALLOCATE C1
```

# Cursor forward only

- Giá trị mặc định của cursor là **forward-only**, tức là chỉ cho duyệt đến (forward) với lệnh:  
*FETCH Cursor INTO các\_biến*  
Hoặc  
*FETCH NEXT FROM Cursor INTO các\_biến*
- Không thể đi đến các mẫu tin khác nhau bằng lệnh  
*FETCH FIRST FROM Cursor INTO các\_biến*  
*FETCH PRIOR FROM Cursor INTO các\_biến*  
*FETCH LAST FROM Cursor INTO các\_biến*  
*FETCH RELATIVE n FROM Cursor INTO các\_biến*

# Cursor Scrollable

- Cho phép di chuyển tới lui giữa các mẩu tin trong cursor như NEXT, FIRST, PRIOR, LAST, ABSOLUTE, RELATIVE
- Khai báo
  - DECLARE *tên\_cursor*
  - CURSOR **SCROLL**
  - FOR *câu\_lệnh\_select*

# Cursor Scrollable

- Ví dụ: (giả sử table có 5 mẫu tin)

```
DECLARE @MS varchar(5), @Ten varchar(20)
DECLARE C1 CURSOR SCROLL FOR
    SELECT MSNV, Hoten FROM nhanVien
OPEN C1
FETCH FIRST FROM C1 INTO @MS, @ten          → 1
FETCH RELATIVE 3 FROM C1 INTO @MS, @ten      → 4
FETCH ABSOLUTE 2 FROM C1 INTO @MS, @ten        → 2
FETCH PRIOR FROM C1 INTO @MS, @ten           → 1
FETCH LAST FROM C1 INTO @MS, @ten            → 5
CLOSE C1
DEALLOCATE C1
```

# Cursor tĩnh (static)

- Dữ liệu được chụp (snapshot) và đưa vào cursor.
  - Sau đó mọi biến đổi trên dữ liệu nguồn đều không ảnh hưởng đến dữ liệu trong cursor.
- Khi ta đóng và mở lại cursor thì dữ liệu mới sẽ được cập nhật
- Khai báo

```
DECLARE tên_cursor CURSOR  
STATIC  
FOR câu_lệnh_select
```

# Cursor tĩnh (Static)

```
DECLARE @MS varchar(5), @Ten varchar(20)
DECLARE C1 CURSOR STATIC FOR
    SELECT MSNV, Hoten FROM nhanVien
OPEN C1
FETCH FIRST FROM C1 INTO @MS, @ten
    PRINT @MS + @ten                                → 'Le Minh'
UPDATE NhanVien SET HoTen='Ly An' WHERE MSNV='01'
FETCH FIRST FROM C1 INTO @MS, @ten
    PRINT @MS + @ten                                → 'Le Minh'
CLOSE C1
OPEN C1
FETCH FIRST FROM C1 INTO @MS, @ten
    PRINT @MS + @ten                                → 'Ly An'
CLOSE C1
DEALLOCATE C1
```

# Cursor động (Dynamic)

- Các thay đổi trên dữ liệu nguồn sẽ ảnh hưởng đến dữ liệu trong cursor
- Khai báo

```
DECLARE tên_cursor CURSOR  
DYNAMIC  
FOR câu_lệnh_select
```

# Cursor động (Dynamic)

```
DECLARE @MS varchar(5), @Ten varchar(20)
DECLARE C1 CURSOR DYNAMIC FOR
    SELECT MSNV, Hoten FROM nhanVien
OPEN C1
FETCH FIRST FROM C1 INTO @MS, @ten
    PRINT @MS + @ten                                → 'Ly An'
UPDATE NhanVien SET HoTen='Le Minh' WHERE MSNV='01'
FETCH FIRST FROM C1 INTO @MS, @ten
    PRINT @MS + @ten                                → 'Le Minh'
CLOSE C1
OPEN C1
FETCH FIRST FROM C1 INTO @MS, @ten
    PRINT @MS + @ten                                → 'Le Minh'
CLOSE C1
DEALLOCATE C1
```

# Cập nhật dữ liệu từ cursor

- Ta có thể cập nhật dữ liệu (hoặc xóa) mẫu tin tương ứng với vị trí trong table nguồn của cursor
- Thay đổi dữ liệu trong table
  - UPDATE table SET column = value  
WHERE CURRENT OF cursor\_name
- Xóa dữ liệu trong table
  - DELETE FROM table  
WHERE CURRENT OF cursor\_name

# Cập nhật dữ liệu từ cursor

```
DECLARE @MS varchar(5), @Ten varchar(20)
DECLARE C1 CURSOR SCROLL FOR
    SELECT MSNV, Hoten FROM nhanVien
OPEN C1
FETCH ABSOLUTE 3 FROM C1 INTO @MS, @ten
Print @MS + @Ten          → 'Nguyen Chanh'
UPDATE NhanVien SET HoTen = 'Le Van Tuan'
    WHERE CURRENT OF C1
FETCH ABSOLUTE 3 FROM C1 INTO @MS, @ten
Print @MS + @Ten          → 'Le Van Tuan'
FETCH LAST FROM C1 INTO @MS, @ten
Print @MS + @Ten          → '10 Le Thi Hoa'
DELETE FROM NhanVien WHERE CURRENT OF C1
CLOSE C1
DEALLOCATE C1
```

# Thủ tục lưu trữ (Stored Procedure)

# Khái niệm cơ bản

- Stored Procedure (SP): thủ tục lưu trữ
- SP là một nhóm các câu lệnh T-SQL đã được biên dịch (compiled) và lưu trong cơ sở dữ liệu dưới một tên nào đó
- SP được xử lý như một đơn vị (chứ không phải nhiều câu lệnh)
- SP được quản lý như các đối tượng khác (table, view, ...) trong cơ sở dữ liệu

# Ưu điểm của Stored Procedure

## ■ Hiệu suất (Performance):

- Khi thực thi một câu lệnh SQL thì Server phải kiểm tra quyền của user **gởi câu lệnh** → **kiểm tra cú pháp** → **tạo ra execute plan** → **thực thi**. Nếu gửi các câu lệnh trong một đoạn chương trình như thế sẽ làm giảm tốc độ thực hiện
- Với SP ta chỉ gửi một câu lệnh đơn, Server kiểm tra một lần sau đó tạo ra execute plan và thực thi
- Cú pháp của các câu lệnh đã được Sever kiểm tra trước khi **lưu và biên dịch**, nên nó không cần kiểm lại và biên dịch lại khi thực thi

# Ưu điểm của Stored Procedure

## ■ Bảo trì (maintainability):

- SP sau khi được tạo ra có thể được sử dụng lại → bảo trì dễ dàng hơn do việc tách rời giữa quy tắc nghiệp vụ (business rules) với cơ sở dữ liệu.
- Khi có một sự thay đổi nào đó về mặt logic thì ta chỉ việc thay đổi các lệnh bên trong SP mà thôi. **Những ứng dụng dùng SP này có thể không cần phải thay đổi** mà vẫn tương thích với quy tắc nghiệp vụ mới.
- Ta có thể truyền tham số (parameter) như các ngôn ngữ lập trình khác

# Ưu điểm của Stored Procedure

## ■ Security:

- Dùng SP có thể giới hạn việc truy xuất dữ liệu trực tiếp của một user nào đó vào một số tables
- Chỉ cấp quyền cho phép user đó được sử dụng stored procedure đã viết sẵn mà thôi, chứ **không cấp quyền truy cập trực tiếp các tables** cho user này.
- SP có thể được mã hóa (encrypt) để tăng cường tính bảo mật.

# Các loại Stored Procedure

Stored Procedure có thể được chia thành các nhóm:

- System Stored Procedure
- Local Stored Procedure
- Temporary Stored Procedure
- Extended Stored Procedure
- Remote Stored Procedure

# Các loại Stored Procedure

## ■ System Stored Procedure:

Được chứa trong Master database, thường bắt đầu bằng tiếp đầu ngũ **sp\_**. Chủ yếu dùng trong việc quản lý database và security.

Ví dụ: để kiểm tra các processes đang được sử dụng bởi user 'sa', ta dùng thủ tục:

```
EXEC sp_who 'sa'
```

## ■ Local Stored Procedure:

Đây là loại do DBA hoặc lập trình viên tạo ra để xử lý các yêu cầu nghiệp vụ.

Chúng được chứa trong user database

# Các loại Stored Procedure

## ■ Temporary Stored Procedure:

Tương tự như local SP, nhưng các SP này được tạo ra trên TempDB của SQL Server → chúng sẽ bị xóa khi connection tạo ra chúng bị cắt đứt hay khi SQL Server down

Temporary SP được chia làm 3 loại :

- Local (bắt đầu bằng #),
- Global (bắt đầu bằng ##)
- SP được tạo ra trực tiếp trên TempDB.

# Các loại Stored Procedure

## ■ Extended Stored Procedure:

Đây là loại SP sử dụng một chương trình bên ngoài (external program) đã được biên dịch thành một DLL để mở rộng chức năng hoạt động của SQL Server.

Thường bắt đầu bằng `xp_`

Nhiều extend stored procedure được xem như system stored procedure

Ví dụ: dùng `xp_sendmail` dùng để gửi mail cho một người nào đó, ...

## ■ Remote Stored Procedure:

Là loại SP gọi stored procedure ở server khác

# Tạo và thực thi Stored Procedure

- Tạo SP:

CREATE PROCEDURE <tênSP>

[<các\_tham\_số>]

AS

<các\_lệnh\_trong\_SP>

- Tham số: <Tên tham số> <kiểu dữ liệu>

[= <giá trị mặc định>] [OUTPUT]

- Gọi thực thi SP:

EXEC <tênSP>

# Giá trị mặc định của tham số

- Khi khai báo tham số trong SP, có thể khởi tạo giá trị mặc định cho tham số. Khi gọi SP, nếu ta không gởi giá trị cho tham số, thì tham số sẽ lấy giá trị mặc định

Ví dụ:

```
CREATE PROCEDURE sp_Cong  
    @a int, @b int = 50, @c int = 100  
AS  
    DECLARE @kq int  
    Set @kq = @a + @b + @c  
    Print @kq  
RETURN
```

- Gọi SP: EXEC sp\_cong 10, 20 → 130  
EXEC sp\_cong @a=10, @c=20 → 80

# Tham số OUTPUT

- Mặc định tham số là loại INPUT (tham trị), khi truyền tham số thực, tham số sẽ nhận giá trị gởi vào để thực hiện
- Nếu có từ khóa OUTPUT thì tham số thuộc loại tham biến → tham số thực gởi vào sẽ là một biến để nhận giá trị trả về
- Ví dụ: CREATE PROCEDURE sp\_Cong
  - @a int, @b int, @kq int OUTPUT
  - AS
  - Set @kq = @a + @b
- Gọi SP:

```
Declare @KetQua int
EXEC sp_Cong 10, 20, @KetQua OUTPUT
Print @KetQua
```

# Hàm (Function)

# Hàm người dùng

- User-Defined Functions (UDFs)
- Là hàm do người dùng tự định nghĩa, bao gồm các câu lệnh T-SQL, và được gọi thực thi như một đơn vị độc lập
- UDFs có hầu hết các tính năng tương tự như thủ tục trữ sẵn (stored procedure), ngoại trừ:
  - Tham số trong UDFs không được mang thuộc tính OUTPUT
  - Phải trả giá trị về cho hàm bằng phát biểu RETURN

# Hàm vô hướng

- Scalar Functions
- Giá trị trả về thuộc kiểu dữ liệu vô hướng như: int, smallint, ...
- Trừ các kiểu sau không được dùng làm kiểu trả về: BLOBs, cursor, timestamp
- Có thể sử dụng các hàm này trong phát biểu SQL như SELECT

# Tạo hàm vô hướng

CREATE FUNCTION <tên\_hàm>

( [<các tham số>] )

**RETURNS** <kiểu trả về>

AS

BEGIN

<các lệnh trong thân hàm>

**RETURN** <kết quả trả về>

END

# Ví dụ về hàm vô hướng

```
CREATE FUNCTION DT_HCN
```

```
( @Dai decimal(4,1),  
    @Rong decimal(4,1)  
)
```

```
RETURNS decimal(10,2)
```

```
AS
```

```
BEGIN
```

```
    RETURN (@Dai * @Rong)
```

```
END
```

# Thực thi hàm

- Sử dụng như bất kỳ hàm nào của hệ thống
- Lưu ý: cần cung cấp tên sở hữu chủ (**owner**) trước tên hàm
- Ví dụ:

Print **dbo.DT\_HCN( 8.5, 10.3)**  
kết quả trả về: 87.55

# Ví dụ

- Create function SoNVPB (@MSPB char(2))

RETURNS int

AS

Begin

    declare @SL int;

    select @SL =Count(\*) from NhanVien

    where MSPB = @MSPB;

    return @SL;

End

- Thực thi hàm:

print dbo.SoNVPB ('P1')

# Hàm trả về table

- Có 2 loại:
  - Inline Table-Value Functions
  - Multistatement Table-Value Functions
- Có thể dùng các hàm này trong mệnh đề FROM của câu lệnh SELECT như một table thông thường

# Hàm Inline Table-Valued

- Giá trị trả về là một table lấy từ kết quả của một câu lệnh SELECT
- CREATE FUNCTION <tên hàm> [<các tham số>]  
**RETURNS TABLE**  
AS  
RETURN (<câu lệnh SELECT>)

# Hàm Inline Table-Valued

- Ví dụ:

```
CREATE FUNCTION DSNV (@MS char(2) )
RETURNS TABLE
AS
RETURN (SELECT * FROM NhanVien
        WHERE MSPB = @MS)
```

- Thực thi: `SELECT * FROM dbo.DSNV('P1')`

|   | MSNV | HoTen         | MSPB |
|---|------|---------------|------|
| 1 | 0001 | Lê Bình       | P1   |
| 2 | 0003 | Trần Văn Hùng | P1   |
| 3 | 0004 | Lý Văn Minh   | P1   |

# Hàm Multistatement Table-Valued

## ■ Multistatement Table-Value Functions

Thân hàm là một khối BEGIN ... END chứa các lệnh T-SQL dùng để xây dựng và xử lý các dữ liệu đưa vào các mẫu tin trong table mà nó trả về

## ■ CREATE FUNCTION <Tên hàm> ([<các tham số>])

RETURNS @BiếnTable TABLE <cấu trúc table>

AS

BEGIN

<các lệnh đưa dữ liệu vào biến table>

RETURN

END

# Ví dụ

```
CREATE FUNCTION fn_FindReports (@InEmpID INTEGER)
RETURNS @retFindReports TABLE
( EmployeeID int primary key NOT NULL,
  Name nvarchar(255) NOT NULL,
  Title nvarchar(50) NOT NULL,
  EmployeeLevel int NOT NULL)
AS
BEGIN
  <các xử lý>
  INSERT @retFindReports
    SELECT EmployeeID, Name, Title, EmployeeLevel
      FROM DirectReports
  <các xử lý>
  RETURN
END;
```

# Trigger

# Giới thiệu

- Trigger là một đối tượng trong CSDL
- Trigger là một đoạn chương trình được lưu trữ trên server (tương tự Stored Procedure hay Function)
- Tuy nhiên có một số điểm khác biệt:
  - **Tự động thực thi** khi có sự kiện tương ứng xảy ra, như thêm, xóa, sửa dữ liệu trong table (trong khi SP phải gọi mới thực thi)
  - Không sử dụng tham số và giá trị trả về như SP hay hàm

# Giới thiệu

- Một vài ứng dụng thông thường của trigger:
  - Kiểm tra toàn vẹn dữ liệu (Referential Integrity)
  - Theo dõi vết của dữ liệu (Audit Trails)
  - Thực hiện một số thao tác tự động,
  - ...
- Thường chia thành các loại lớn
  - DML trigger
  - DDL trigger
  - CLR trigger
  - Logon trigger

# DML Trigger

Bao gồm các loại sau:

- INSERT trigger
  - Tự động chạy khi thêm vào một mẫu tin
- DELETE trigger
  - Tự động chạy khi xóa một mẫu tin
- UPDATE trigger
  - Tự động chạy khi dữ liệu của một mẫu tin bị sửa
- Kết hợp các loại trên

# DDL Trigger

- Các câu lệnh DDL tác động trên CSDL  
(DDL Statements with Database Scope):
  - CREATE\_TABLE, ALTER\_TABLE, DROP\_TABLE
  - CREATE\_USER, ALTER\_USER, DROP\_USER
  - CREATE\_VIEW, ALTER\_VIEW, DROP\_VIEW, ...
- Các câu lệnh DDL tác động trên Server  
(DDL Statements with Server Scope)
  - CREATE\_DATABASE, DROP\_DATABASE
  - CREATE\_LOGIN, ALTER\_LOGIN, DROP\_LOGIN
  - GRANT\_SERVER, REVOKE\_SERVER, ...

# CLR Trigger

- CLR Trigger là loại đặc biệt của trigger dựa trên CLR (Common Language Runtime) trong .NET framework.
- Chúng ta có thể viết mã cho cả DDL và DML trigger, bằng cách sử dụng một ngôn ngữ được hỗ trợ CLR như C#, Visual basic, F#

# Logon trigger

- Logon trigger là loại đặc biệt của trigger được thực thi khi có sự kiện LOGON vào SQL Server.
- Chúng ta có thể sử dụng các trigger để theo dõi và kiểm soát các phiên làm việc trên máy chủ, chẳng hạn như để theo dõi hoạt động đăng nhập hoặc giới hạn thời gian cho một phiên làm việc nào đó.

# DML Trigger

Cú pháp:

```
CREATE TRIGGER <tên trigger>
    ON <table | view>
    FOR [INSERT] [,DELETE] [,UPDATE]
```

AS

```
<các lệnh trong thân trigger>
```

# Table INSERTED (DELETED)

- Chỉ tồn tại trong quá trình trigger thực hiện
- Mỗi khi một mẫu tin được thêm vào table, thì một bản sao dữ liệu của mẫu tin này được lưu trong table INSERTED
- Mỗi khi một mẫu tin trong table bị xóa, thì một bản sao dữ liệu của mẫu tin này được lưu trong table DELETED
- Mỗi khi một mẫu tin trong table bị sửa chữa, thì bản sao của dữ liệu cũ được lưu trong table DELETED, bản sao của dữ liệu mới sẽ lưu trong table INSERTED

# Insert Trigger

```
CREATE TRIGGER trgThemNV
```

```
    ON NhanVien
```

```
    FOR INSERT
```

```
AS
```

```
    UPDATE PhongBan SET TSNV = TSNV + 1
```

```
        WHERE MSPB = (SELECT MSPB FROM Inserted)
```

- Đặt vấn đề:

Nếu ta dùng lệnh đưa cùng lúc nhiều mẫu tin. Ví dụ:

```
Insert into NhanVien
```

```
Select * From NV
```

→ việc gì sẽ xảy ra nếu đã cài trigger trên? Sửa lại thế nào?

# Delete Trigger

```
CREATE TRIGGER trgXoaSV
```

```
    ON SinhVien
```

```
    FOR DELETE
```

```
AS
```

```
    UPDATE Lop SET SiSo = SiSo - 1
```

```
        WHERE MSLOP = (SELECT MSLOP FROM Deleted)
```

- Đặt vấn đề:

Nếu ta dùng lệnh để xóa cùng lúc nhiều mẫu tin. Ví dụ:

```
Delete from SinhVien
```

```
Where hoten = 'minh'
```

→ việc gì sẽ xảy ra nếu đã cài trigger trên? Sửa lại thế nào?

# Update Trigger

```
CREATE TRIGGER trgSuaSV
    ON SinhVien
    FOR Update
AS
    IF Update(Mslop)
        Begin
            UPDATE Lop SET SiSo = SiSo - 1
                WHERE MSLOP = (Select MSLop From Deleted)
            UPDATE Lop SET SiSo = SiSo + 1
                WHERE MSLOP = (Select MSLop From Inserted)
        End
```

# Ứng dụng kiểm tra ràng buộc

```
CREATE TRIGGER trgThemSV
    ON SinhVien FOR INSERT
AS
    Declare @SLCon int
    SELECT @SLCon = (DuKien-SiSo) FROM Lop
        WHERE MSLop = (Select MSLop from Inserted)
    IF @SLCon > 0
        UPDATE Lop SET SiSo = SiSo + 1
            WHERE MSLOP = (Select MSLop from Inserted)
    ELSE
        Begin
            Print 'Lop da du si so'
            ROLLBACK TRAN
        End
```

# Ứng dụng kiểm tra ràng buộc

```
CREATE TRIGGER trgThemSV
    ON SinhVien
    FOR INSERT
AS
    IF NOT EXISTS (SELECT Lop.MSLop
                    FROM Lop, Inserted
                    WHERE Lop.MSLop=Inserted.MSLop)
        Begin
            Print 'Khong co lop nay !'
            ROLLBACK TRAN
        End
    ELSE
        < các xử lý khác>
```

# INSTEAD OF Trigger

- Ta có thể thêm, xóa, sửa dữ liệu trên các view đơn giản (chỉ áp dụng 2 phép chiếu và chọn) trên một table (xem lại phần view)
- Đối với các view kết hợp từ nhiều table. Khi thực hiện các lệnh thêm, xóa, sửa dữ liệu → ta sẽ nhận được thông báo lỗi:

View ‘xxx’ is not updatable because  
the modification affects multiple base table

# INSTEAD OF Trigger

- Muốn thực hiện việc này, ta có thể cài đặt Instead of Trigger cho view này
- Cú pháp:  
CREATE TRIGGER <tên trigger>  
ON <view>  
**INSTEAD OF [INSERT] [,DELETE] [,UPDATE]**  
AS  
<các lệnh trong thân trigger>

# Ví dụ

- Tạo view SVLop

```
CREATE VIEW SVLop  
AS SELECT SinhVien.*, TenLop  
        FROM SinhVien, Lop  
        WHERE SinhVien.MSLop = Lop.MSLop
```

- Thêm 1 mẫu tin vào view

```
Insert into SVLop values ('10', 'Linh', 'L2', 'Excel')
```

→ Báo lỗi: Msg 4405, Level 16, State 1, Line 1

View or function 'SVLop' is not updatable because  
the modification affects multiple base tables.

# Ví dụ

- Viết Instead of trigger cho view:

```
CREATE TRIGGER TrgVwSVLop
```

```
ON SVLop
```

```
INSTEAD OF INSERT
```

```
AS
```

```
Declare @MSSV nChar(4), @TenSV nchar(20)
```

```
Declare @MSL nChar(2), @TenL nChar(30)
```

```
SELECT @MSSV=MSSV, @TenSV=HoTen,
```

```
        @MSL=MSLop, @TenL=TenLop FROM Inserted
```

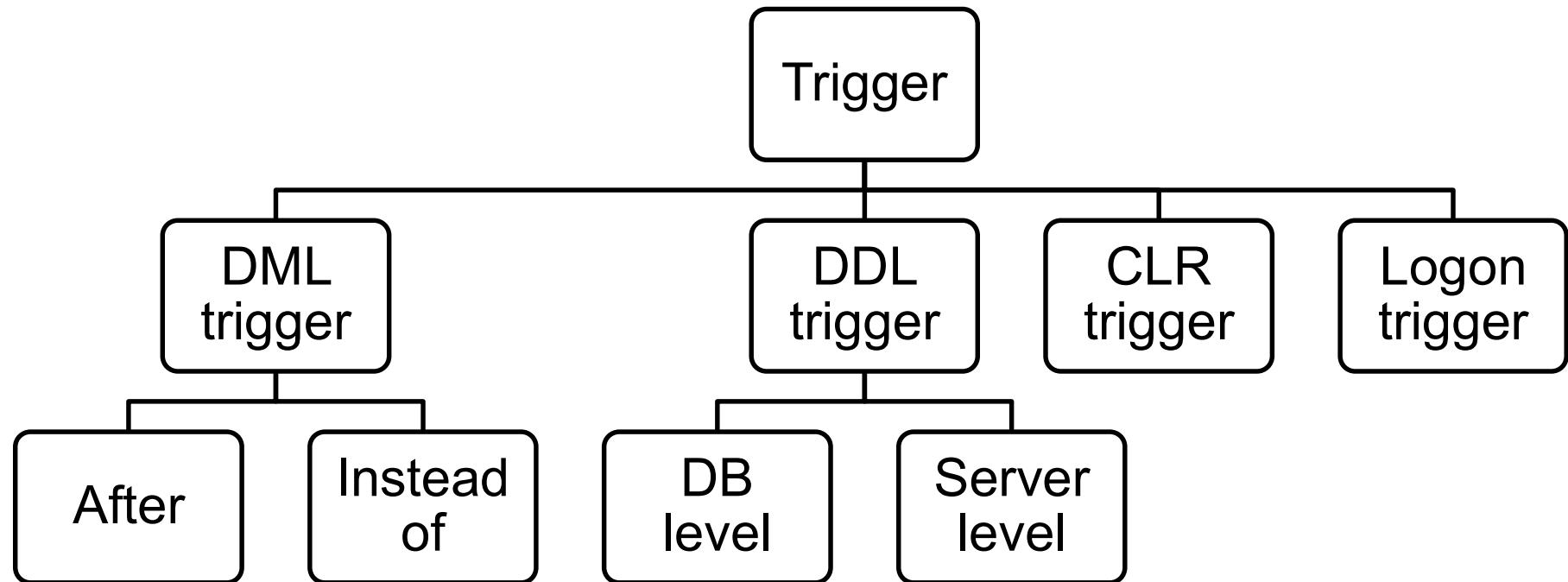
```
IF NOT EXISTS
```

```
    (SELECT MSLop FROM Lop WHERE MSLop = @MSL)
```

```
        INSERT INTO Lop VALUES (@MSL, @TenL, 0, 0)
```

```
        INSERT INTO SINHVIEN VALUES (@MSSV, @TenSV, @MSL)
```

# Tóm tắt các loại trigger



Sinh viên tự tìm hiểu thêm về DDL trigger,  
CLR trigger và Logon trigger

# Tóm tắt

- Ngôn ngữ T-SQL
- Cursor
- Stored-Procedure
- Function
- Trigger

# Chương 5

# Quản lý người dùng



Môn: QUẢN TRỊ CƠ SỞ DỮ LIỆU



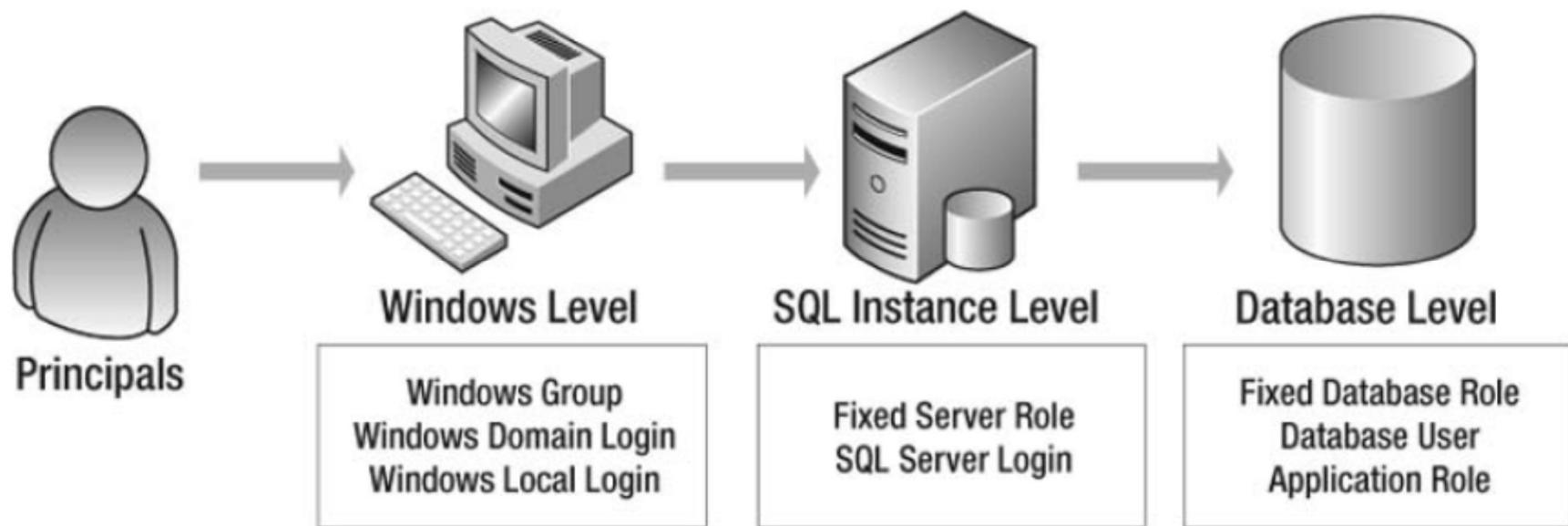
# Nội dung

- Các chế độ xác nhận người dùng (authentication)
- User login, Database user
- Quyền trên hệ thống
- Quyền trên các đối tượng cơ sở dữ liệu
- Role
- Cấp quyền và rút quyền

# Principals

- Principals là các thực thể có nhu cầu truy xuất các đối tượng trong database.
- Có 3 loại principal cơ bản:
  - Window's principals
    - Đăng nhập SQL Server bằng Windows authentication
  - SQL Server principals
    - Đăng nhập bằng SQL Server authentication
  - Database principals
    - Database user, database role, application role
- Windows và SQL server principal được ánh xạ thành database principal để truy xuất đối tượng của database

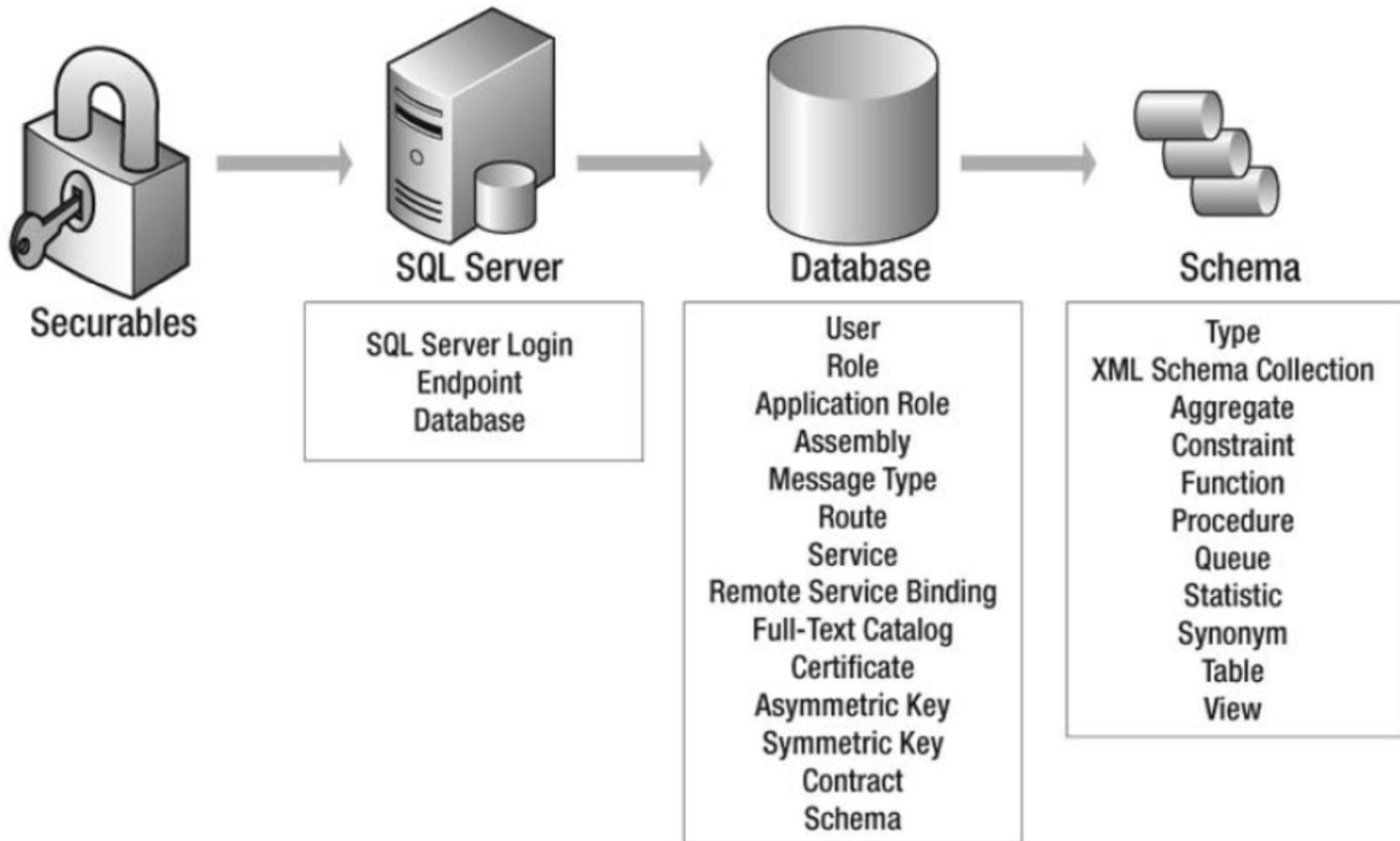
# Principals



# Securables

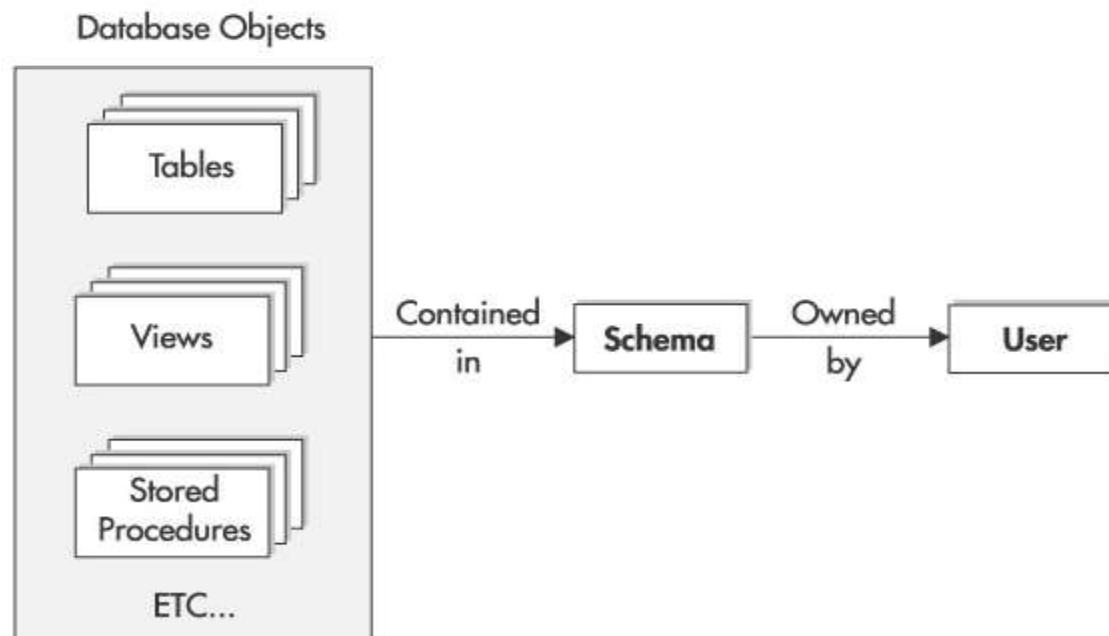
- Là các đối tượng bên trong SQL Server có thể kiểm soát và quản lý qua SQL Server security
- Có 3 mức tầm vực (scope):
  - Server scope
    - Như logins, endpoints, databases
  - Database scope
    - Như users, roles, assemblies, ...
  - Schema scope
    - Như tables, view, stored procedure, ...

# Securables



# Schema (Lược đồ)

- Bao gồm tất cả các đối tượng CSDL thuộc quyền quản lý (sở hữu chủ) của một user
- Khi tạo database user bằng sp\_adduser → một schema của user này cũng được tự động tạo ra



# Schema (Lược đồ)

- Tạo schema:

```
CREATE SCHEMA schema_name  
    AUTHORIZATION owner_name
```

- Ví dụ:

```
CREATE USER ThucTap FOR LOGIN Hanh  
CREATE SCHEMA Hanh  
    AUTHORIZATION ThucTap
```

# Logins

SQL Server hỗ trợ 2 chế độ xác thực đăng nhập

- **Windows Authentication:**

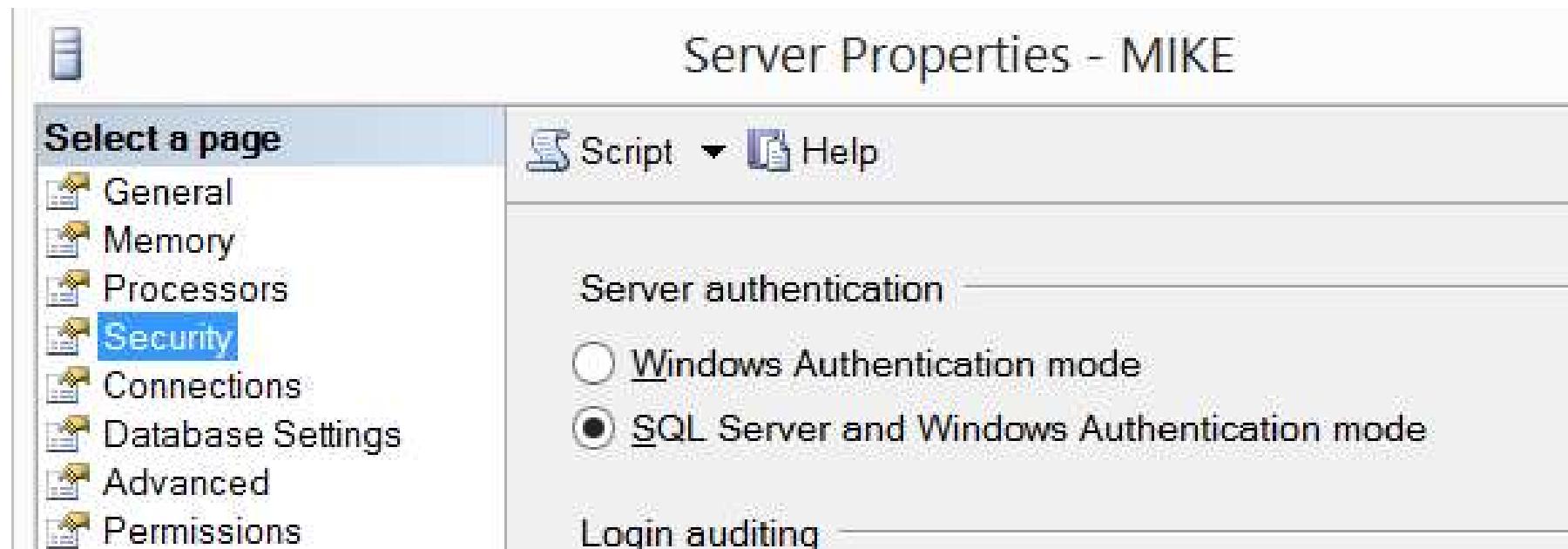
- Dùng tài khoản (account) của Windows để đăng nhập vào SQL Server.
- Không cần nhập lại username và password

- **SQL Server Authentication:**

- Quản lý độc lập với tài khoản của Windows
- Khi đăng nhập SQL Server phải nhập loginname và password

# Logins

- Mặc định khi cài đặt SQL để chế độ chỉ hỗ trợ Windows authentication
- Để sử dụng cả 2 chế độ, cần chỉnh cấu hình thành **Mixed Mode authentication**



# Lệnh CREATE LOGIN

- Tạo Login SQL Server Authentication

```
CREATE LOGIN login_name
```

**WITH PASSWORD** = 'password'

[,DEFAULT\_DATABASE = database]

- Tạo Login Windows Authentication

```
CREATE LOGIN login_name
```

**FROM WINDOWS**

[,DEFAULT\_DATABASE = database]

# Ví dụ

- Tạo Login Windows Authentication:
  - Giả sử tên máy là “STU”
  - Tạo user “Tuan” trên windows
  - CREATE LOGIN “STU\Tuan” FROM WINDOWS
- Tạo Login SQL Server Authentication:
  - CREATE LOGIN Ngoc  
WITH Password = 'abcd',  
DEFAULT\_DATABASE = BaiTap

# Tạo login bằng sp\_addlogin

sp\_addlogin

[@ loginame = ] 'loginname'

[@ passwd = ] 'password'

[@ defdb = ] 'database'

- Ví dụ:

EXEC sp\_addlogin 'Lan', 'abcd'

# Thay đổi mật khẩu của Login

- Dùng lệnh ALTER LOGIN:

```
ALTER LOGIN login name WITH  
PASSWORD = 'new_password'  
[ OLD_PASSWORD = 'old_password' ]
```

- Ví dụ:

```
ALTER LOGIN an  
WITH PASSWORD = 'aaa'  
OLD_PASSWORD = '789'
```

# Thay đổi mật khẩu của Login

- Dùng stored procedure sp\_password

sp\_password

[ @old = ] 'old\_password' , ]

[ @new = ] 'new\_password'

[, [ @loginame = ] 'login' ]

- Ví dụ:

EXEC sp\_password '123', '456'

→ đổi password của login từ '123' thành '456'

# Xóa Login

- Dùng stored procedure sp\_droplogin  
EXEC sp\_droplogin 'loginname'

Ví dụ:

EXEC sp\_droplogin Linh

- Dùng lệnh DROP LOGIN:  
DROP LOGIN 'loginname'

Ví dụ:

DROP LOGIN Lan

# Quyền quản trị hệ thống

Là các quyền có thể cấp cho các Login để thực hiện các công việc ở mức hệ thống.

|  |   |
|--|---|
| CREATE DATABASE<br>DENY<br>GRANT<br>ALTER DATABASE<br>BACKUP DATABASE<br>SHUTDOWN<br>INSERT permission on any object | RESTORE DATABASE<br>RESTORE LOG<br>REVOKE<br>SELECT permission on any object<br>TRUNCATE TABLE<br>UPDATE permission on any object<br>USE to a suspect database<br>... |
|--|---|

# SQL Server-Level Roles

- SQL Server đã tạo sẵn một số role (vai trò), mỗi role bao gồm nhiều quyền hệ thống
- Login sẽ được gán cho một (hoặc nhiều) role để thực hiện các công việc của mình
- Các role tiêu biểu:
  - SysAdmin
  - ServerAdmin
  - Public (default) ...
- Có thể dùng lệnh sau để xem quyền của các role:  
`EXEC sp_srvrolepermission`

# SQL Server-Level Roles

- Gán server role cho Login

```
sp_addsrvrolemember [@loginame =] 'login',
[@rolename =] 'role'
```

Ví dụ: exec **sp\_addsrvrolemember** Minh, sysadmin

- Thu hồi server role của Login

```
sp_dropsrvrolemember [@loginame =] 'login',
[@rolename =] 'role'
```

Ví dụ: exec **sp\_dropsrvrolemember** Minh, sysadmin

# Database User

- Mỗi database chỉ cho phép một số login làm việc trên database đó → gọi là database user
- Mỗi login sau khi đăng nhập thành công, có thể được cho phép làm việc trên một số database (quyền truy xuất vào cơ sở dữ liệu)

# Tạo Database User

- Dùng lệnh CREATE USER  
CREATE USER user\_name  
[FOR LOGIN login\_name]

- Ví dụ:  
Use STU  
CREATE USER Ngoc  
CREATE USER QuanLy FOR LOGIN Linh

# Database User

- Dùng stored procedure

```
sp_adduser [@loginame = ] 'login'  
          [@name_in_db = ] 'user' ]
```

- Ví dụ:

```
Use STU
```

```
EXEC sp_adduser 'Ngoc'
```

```
EXEC sp_adduser 'Linh', 'QuanLy'
```

# Xóa Database User

- Dùng lệnh DROP USER:

DROP USER user\_name

Ví dụ: DROP USER an

- Dùng stored procedure sp\_dropuser

sp\_dropuser [ sp\_dropuser ] 'user'

Ví dụ: EXEC sp\_dropuser 'an'

# Database Role

- Mỗi database role là một tập các quyền trên các đối tượng trong database
- SQL Server hỗ trợ 2 loại database role:
  - Fixed roles
    - Các role đã được SQL Server tạo sẵn
  - Flexible roles
    - Các role do người dùng tự tạo, để dễ dàng phân quyền cho từng nhóm người dùng

# Fixed Database Roles

- Các fixed database role tiêu biểu:
  - db\_owner (dbo)
  - db\_accessadmin
  - db\_backupoperator
  - db\_datareader
  - db\_datawriter
  - db\_denydatareader
  - db\_denydatawriter
  - db\_securityadmin
  - db\_ddladmin
- Để xem các quyền của các role này ta dùng:  
**Exec sp\_dbfixedrolepermission**

# Tạo Database Role

- Ta có thể tạo một role mới theo cú pháp:

**CREATE ROLE *role\_name***

[AUTHORIZATION *owner\_name* ]

- Ví dụ:

CREATE ROLE KeToanVien

- Thay vì gán các quyền cho từng user. Ta có thể gán các quyền cho role. Sau đó gán role cho các user thích hợp → dễ bảo trì

# Thêm/xóa user cho Database Role

- Thêm user vào một database role

```
sp_addrolemember [@rolename =] 'role',  
[@membername =] 'user'
```

- Xóa user khỏi role

```
sp_droprolemember [@rolename = ] 'role' ,  
[@membername = ] 'user'
```

# Application Roles

- Là một database principal cho phép một ứng dụng kết nối vào database và thực hiện các hành động tương ứng với quyền (permission) đã gán cho role
- Khác với database role là ta không thể add user vào, và được tạo ra cùng với password
- Application role sẽ disabled cho đến khi ứng dụng kết nối vào database và đưa vào đúng password

# Application Roles

- Tạo application role:

```
CREATE APPLICATION ROLE authorApps
```

```
WITH PASSWORD = '@uth0rA@pp5@!'
```

```
GRANT SELECT, UPDATE, INSERT to AuthorApps
```

- Sau khi tạo role, ta có thể enable nó từ ứng dụng bằng cách cho ứng dụng đó gọi stored procedure:  
`exec sp_setapprole authorApps, '@uth0rA@pp5@!'`

# Server securables

- Gán quyền

**GRANT** SHUTDOWN,CREATE ANY DATABASE  
TO Minh

- Cấm quyền

**DENY** SHUTDOWN,CREATE ANY DATABASE  
TO Minh

- Thu hồi quyền

**REVOKE** SHUTDOWN,CREATE ANY DATABASE  
FROM Minh

# Database securables

- Gán quyền

USE STU

**GRANT** BACKUP DATABASE, BACKUP LOG

TO Minh

- Cấm quyền

**DENY** BACKUP DATABASE, BACKUP LOG

TO Minh

- Thu hồi quyền

**REVOKE** BACKUP DATABASE, BACKUP LOG

FROM Minh

# Permission

- Là các quyền cấp cho principal để xác định các hành vi được thực hiện trên các securable
- Một số quyền trên các đối tượng trong CSDL:
  - CREATE DEFAULT
  - CREATE FUNCTION
  - CREATE PROCEDURE
  - CREATE TABLE
  - CREATE VIEW
  - ...

# Gán quyền xử lý trên CSDL

- Cú pháp:

```
GRANT { ALL | permission [ (column [ ,...n ] ) ] }
[ ON securable ]
TO principal [ ,...n ]
[ WITH GRANT OPTION ]
```

- Ví dụ:

```
GRANT CREATE TABLE TO Minh WITH GRANT OPTION
```

```
GRANT INSERT ON NhanVien TO Minh
```

```
GRANT SELECT(Luong) ON NhanVien TO Minh
```

```
GRANT ALL ON NhanVien TO Minh
```

# Gán quyền xử lý trên CSDL

- Tùy từng loại đối tượng ALL có nghĩa khác nhau
- Database:
  - BACKUP DATABASE, BACKUP LOG, CREATE DATABASE, CREATE DEFAULT, CREATE FUNCTION, CREATE PROCEDURE, CREATE RULE, CREATE TABLE, CREATE VIEW.
- Scalar function:
  - EXECUTE, REFERENCES.
- Table-valued function:
  - DELETE, INSERT, REFERENCES, SELECT, UPDATE.
- Stored procedure:
  - DELETE, EXECUTE, INSERT, SELECT, UPDATE.
- Table:
  - DELETE, INSERT, REFERENCES, SELECT, UPDATE.
- View:
  - DELETE, INSERT, REFERENCES, SELECT, UPDATE

# Thu hồi quyền

- Cú pháp:

REVOKE [ GRANT OPTION FOR ]

{ ALL | permission [ ( column [ ,...n ] ) ] }

[ ON securable ]

FROM principal [ ,...n ]

[ CASCADE ]

- Ví dụ:

REVOKE CREATE TABLE FROM Minh

REVOKE ALL ON T1 FROM Minh

# Tóm tắt

- Các chế độ xác nhận người dùng (authentication)
- User login, Database user
- Quyền trên hệ thống
- Quyền trên các đối tượng cơ sở dữ liệu
- Role
- Cấp quyền và rút quyền

# Chương 6

# Quản lý giao tác

# (Transaction)



Môn: QUẢN TRỊ CƠ SỞ DỮ LIỆU



# Nội dung

- Các khái niệm về giao tác
- Thuộc tính ACID của giao tác
- Các trạng thái và tác vụ của giao tác
- Các vấn đề khi xảy ra tranh chấp
- Các sự cố gây hỏng giao tác
- Sổ ghi hệ thống (System Log)
- Commit point
- Các lệnh liên quan Transaction trong SQL Server

# Thuật ngữ

- Hệ thống đơn người dùng (single-user system)
  - Có tối đa một người dùng tại một thời điểm
- Hệ thống đa người dùng (multi-user system)
  - Có thể có nhiều người truy cập đồng thời
- Các loại tương tranh (concurrency)
  - Xử lý đan xen (interleaved processing)
    - Các tiến trình được thi hành xen kẽ trên một CPU
  - Xử lý song song (parallel processing)
    - Các tiến trình được đồng thời thi hành trên nhiều CPU

# Giao tác (transaction)

- Giao tác được hiểu là một chuỗi tác vụ (lệnh) và các lệnh này hoặc được **thực hiện toàn bộ**, hoặc **không làm gì cả** để đảm bảo sự toàn vẹn dữ liệu (tính nguyên tố)
- Một giao tác có thể được gửi đến hệ thống như một lệnh độc lập (stand-alone)
- Biên của giao tác (transaction boundary) xác định bằng cặp lệnh Begin transaction ... End transaction
- Một chương trình ứng dụng có thể có nhiều giao tác khác nhau

# Giao tác (transaction)

Ví dụ:

- Có giao tác T thực hiện việc chuyển một khoản tiền 1000\$ từ tài khoản A sang tài khoản B. Giao tác này thực hiện 2 công việc:
  - Trừ 1000\$ trong tài khoản A
  - Cộng 1000\$ vào tài khoản B
- Nếu vì lý do nào đó, một công việc không thực hiện được, thì toàn bộ giao tác này cần được hủy bỏ

# Thuộc tính ACID của giao tác

- **Atomicity** (tính nguyên tố):
  - Một giao tác được xem là một đơn thể, tức các tác vụ hoặc được thực thi toàn bộ hoặc không thực thi tác vụ nào cả
- **Consistency preservation** (tính nhất quán):
  - Khi thực thi giao tác, CSDL chuyển từ trạng thái nhất quán này sang trạng thái nhất quán khác
  - Trong khi thực hiện đồng thời nhiều giao tác, CSDL vẫn không vi phạm các ràng buộc toàn vẹn đã cài đặt

# Thuộc tính ACID của giao tác

## ■ Isolation (tính cô lập):

- Đảm bảo rằng các dữ liệu trong transaction không bị tác động bởi các transaction khác đang hoạt động đồng thời

## ■ Durability (tính vững bền):

- Sau khi giao tác đã commit, dữ liệu thay đổi phải được lưu bền vững vào CSDL, đảm bảo không để mất mát do các hỏng hóc xảy ra sau đó

# Các trạng thái của giao tác

- Trạng thái hoạt động (Active)
- Trạng thái commit bán phần (Partially committed)
- Trạng thái đã commit (Committed)
- Trạng thái hỏng (Failed)
- Trạng thái đã kết thúc (Terminated)

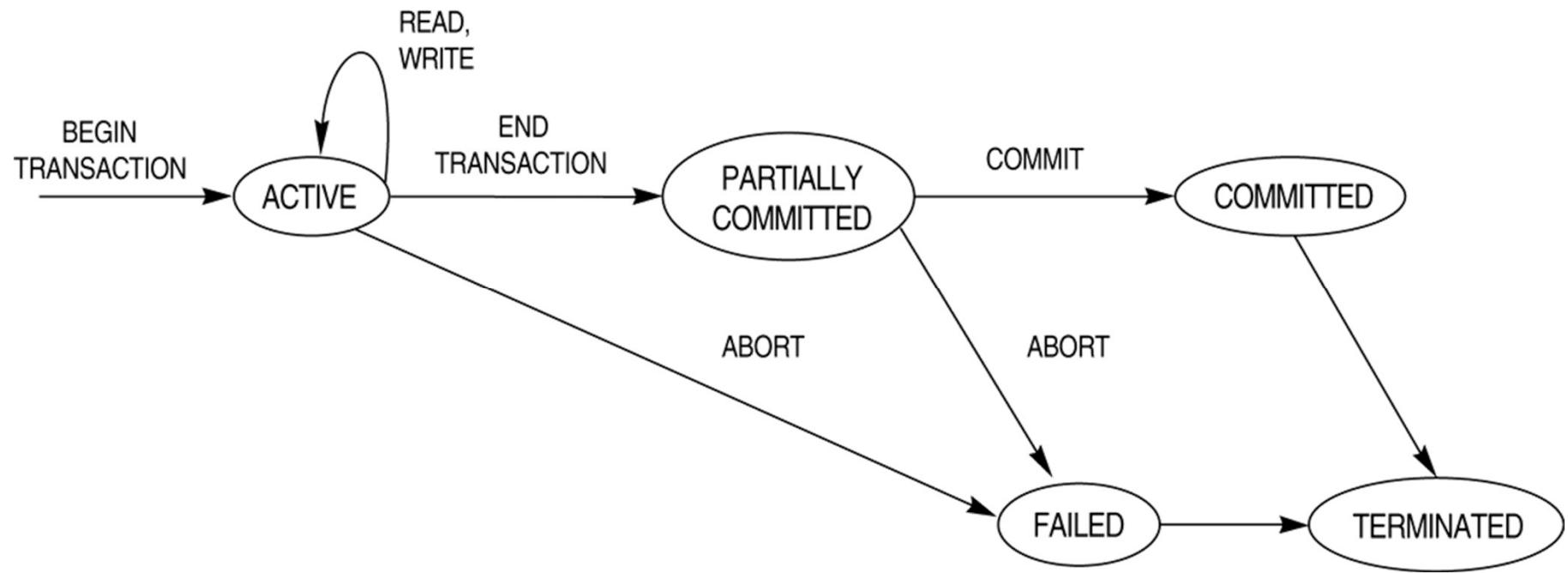
# Các tác vụ của giao tác

- Begin\_transaction: bắt đầu thực hiện giao tác
- Read/Write: đọc/ghi các mục dữ liệu trong giao tác
- End\_transaction: kết thúc giao tác
  - commit\_transaction:
    - Kết thúc giao tác và tất cả các thay đổi trong CSDL tạo ra bởi giao tác đã commit sẽ lưu trữ bền vững và không thể undo được nữa
  - rollback (hoặc abort):
    - Kết thúc giao tác không thành công, các thay đổi tạo ra bởi giao tác sẽ phải được undo, trả lại nguyên trạng ban đầu.

# Các tác vụ của giao tác

- Kỹ thuật khôi phục còn dùng thêm các tác vụ sau:
  - undo: giống rollback nhưng áp dụng cho một tác vụ chứ không phải cho cả giao tác
  - redo: một số tác vụ trong giao tác sẽ được tái thực thi để đảm bảo tất cả các tác vụ của một giao tác đã commit được thực hiện đầy đủ vào CSDL

# Các trạng thái và tác vụ của giao dịch



# Mô hình một hệ CSDL đơn giản

- Dùng để giải thích vấn đề xử lý giao tác
- Một CSDL là một tập các mục dữ liệu (data item) được đặt tên
- Độ mịn dữ liệu (granularity of data) là kích thước của mục dữ liệu:
  - Có thể là một trường (field), mẫu tin (record) hoặc khối (block)
  - Chúng ta chỉ xem xét các mục dữ liệu độc lập với độ mịn của chúng

# Mô hình một hệ CSDL đơn giản

Có 2 tác vụ cơ bản:

- **read\_item(X)**

- Tìm địa chỉ khối trên đĩa chứa mục dữ liệu X
- Sao chép khối chứa mục dữ liệu X lên vùng đệm (buffer) trong bộ nhớ chính (nếu vùng đệm chưa có khối này)
- Sao chép mục dữ liệu X từ vùng đệm vào biến X trong chương trình

# Mô hình một hệ CSDL đơn giản

## ■ `write_item(X)`

- Tìm địa chỉ của khối đĩa chứa mục dữ liệu X
- Sao chép khối này vào vùng đệm trong bộ nhớ chính (nếu vùng đệm chưa có khối này)
- Sao chép biến X từ chương trình vào đúng mục dữ liệu X trong vùng đệm
- Lưu và cập nhật khối (chứa mục dữ liệu X) từ vùng đệm xuống đĩa (ngay lập tức hoặc vào một thời điểm sau này)

# Ví dụ với 2 giao tác đơn giản

(a)

$T_1$

```
read_item ( $X$ );  
 $X:=X-N;$   
write_item ( $X$ );  
read_item ( $Y$ );  
 $Y:=Y+N;$   
write_item ( $Y$ );
```

Chuyển  $N$  đồng từ tài  
khoản  $X$  sang tài khoản  $Y$

(b)

$T_2$

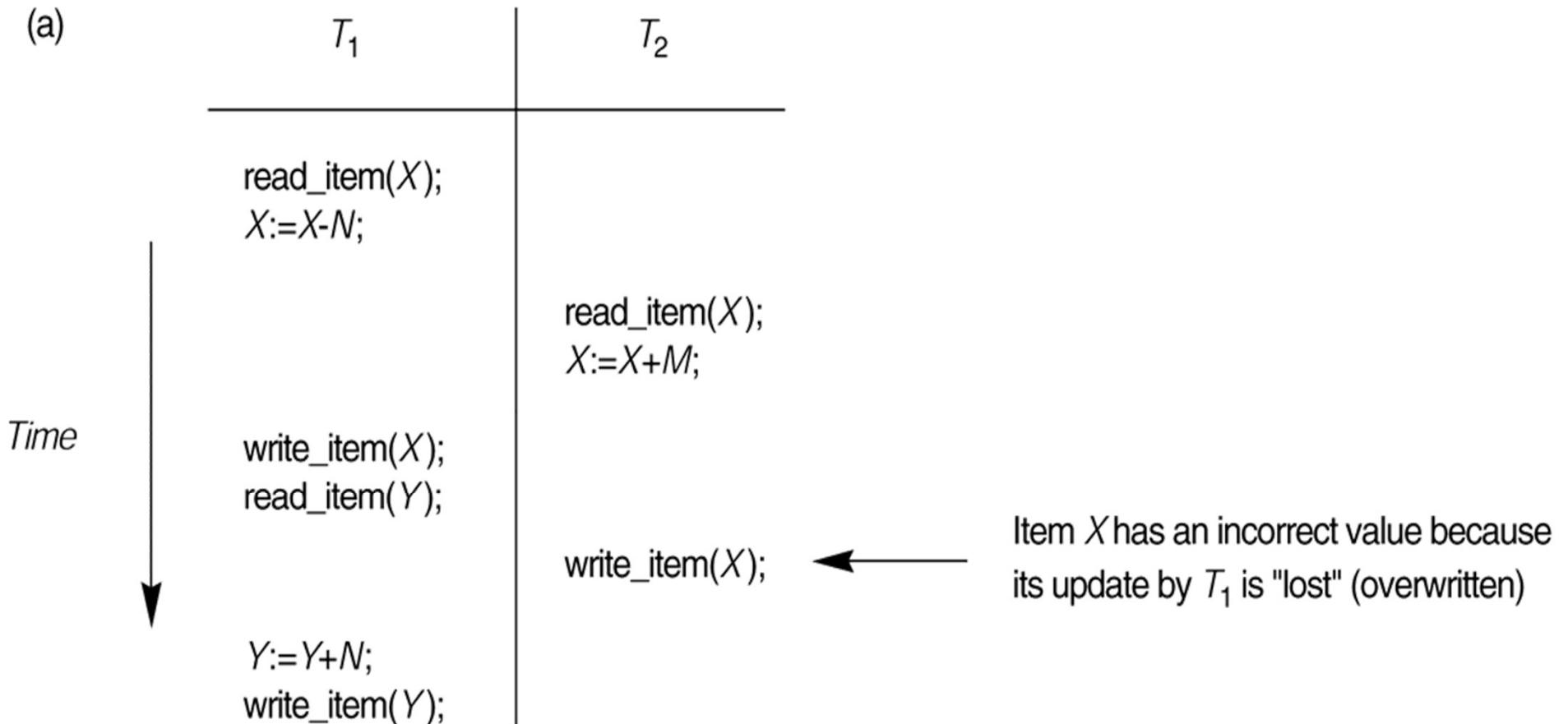
```
read_item ( $X$ );  
 $X:=X+M;$   
write_item ( $X$ );
```

Nạp  $M$  đồng vào tài  
khoản  $X$

# Các vấn đề khi xảy ra tương tranh

- Mất cập nhật (**Lost Update Problem**)
  - Xảy ra khi hai giao tác truy cập vào cùng các mục dữ liệu theo cách đan xen và làm cho giá trị của một số mục dữ liệu không còn đúng.
  - Ví dụ: giao tác sau ghi đè lên một mục dữ liệu mà không biết là mục dữ liệu này đã được cập nhật bởi giao tác trước

# Lost Update



# Các vấn đề khi xảy ra tương tranh

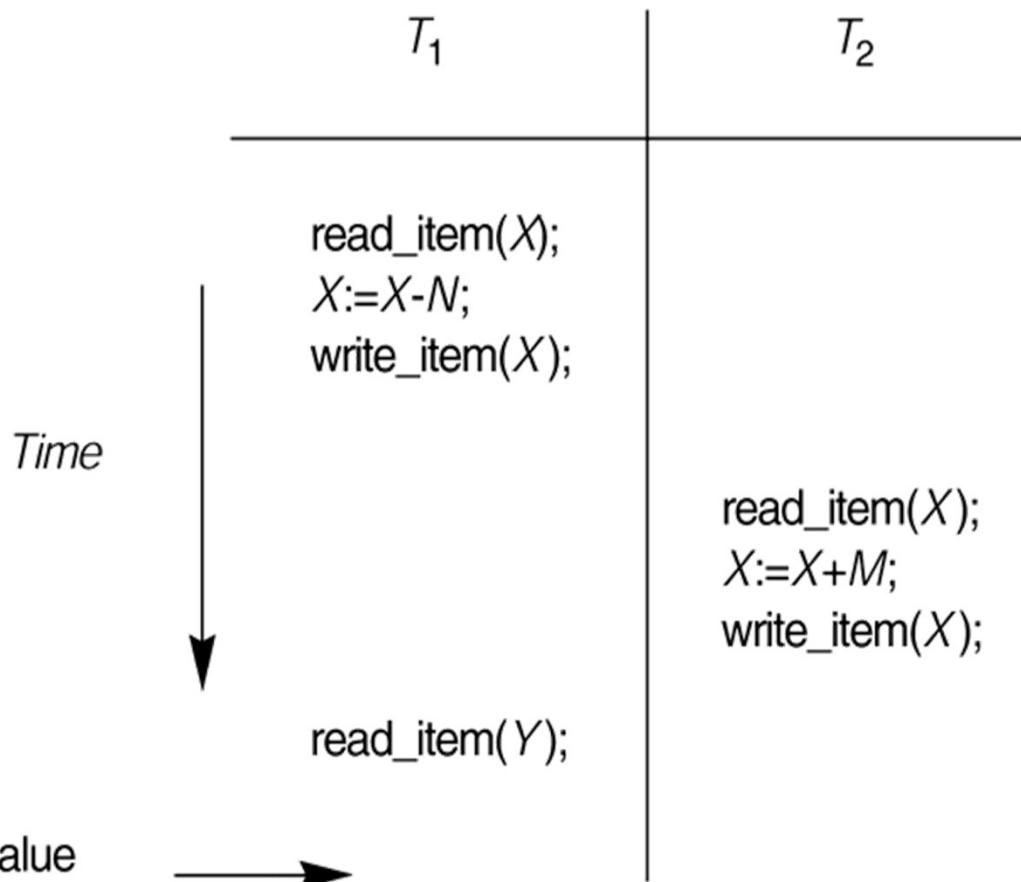
## ■ Đọc dữ liệu dơ

(The Temporary Update or **Dirty Read** Problem)

- Còn gọi là dữ liệu chưa commit (**Uncommit** data)
- Xảy ra khi một giao tác cập nhật một mục dữ liệu và sau đó bị hỏng, trong khi đó, một giao tác khác truy cập vào mục dữ liệu này trước khi nó được khôi phục giá trị gốc

# Dirty read

(b)



Transaction  $T_1$  fails and must change the value of  $X$  back to its old value; meanwhile  $T_2$  has read the "temporary" incorrect value of  $X$ .

# Các vấn đề khi xảy ra tương tranh

- Dữ liệu không thể đọc lặp lại (Unrepeatable data)
  - Tổng hợp sai (Incorrect Summary Problem)
  - Giao tác có thể đọc một dữ liệu 2 lần và giữa hai lần đó có một giao tác khác cập nhật, dẫn đến dữ liệu còn không nhất quán → các thao tác tính toán (như SUM, AVG, ...) không còn chính xác
- Hiện tượng bóng ma (Phantom)
  - Là tình trạng mà một giao tác đang thao tác trên một tập dữ liệu nhưng giao tác khác lại chèn thêm (hoặc xóa bỏ) các dòng dữ liệu mà giao tác kia đang sử dụng.

# Incorrect Summary

| (c) | $T_1$   | $T_3$  |
|-----|---|--|
|     |   | <pre>sum:=0; read_item(A); sum:=sum+A;  • • •</pre>  |
|     | <pre>read_item(X); X:=X-N; write_item(X);</pre> | <pre>read_item(X); sum:=sum+X; read_item(Y); sum:=sum+Y;</pre>   |
|     | <pre>read_item(Y); Y:=Y+N; write_item(Y);</pre> | <p><math>T_3</math> reads <math>X</math> after <math>N</math> is subtracted and reads <math>Y</math> before <math>N</math> is added; a wrong summary is the result (off by <math>N</math>). </p> |

# Các sự cố gây hỏng giao tác

- Sụp hệ thống (system crash) do lỗi phần cứng hoặc phần mềm xảy ra khi đang thực hiện giao tác
- Lỗi do lệnh trong giao tác gây ra, như lỗi chia cho không, dữ liệu dùng cho giao tác không tồn tại, ...
- Lỗi do xử lý điều kiện ngoại lệ làm cho giao tác bị huỷ bỏ. Ví dụ lệnh huỷ được lập trình trong giao tác
- Phương pháp điều khiển tương tranh có thể quyết định huỷ và khởi động lại một giao tác để giải quyết vấn đề khoá chết (deadlock), khả tuần tự hoá (serializability)

# Sổ ghi hệ thống (System Log)

- Còn gọi là nhật ký (journal)
- Dùng để theo dõi tất cả các tác vụ của giao tác
- Thông tin trong sổ ghi dùng để khôi phục khi giao tác thất bại
- Sổ ghi được lưu trên đĩa, nhằm tránh các hỏng hóc bất thường xảy ra
- Sổ ghi nên được sao chép dự phòng thường xuyên đến các bộ lưu trữ ổn định dự phòng khác để tránh lỗi đĩa hoặc các tai họa (như mất cắp, phá hoại, ...)

# Các loại log record trong sổ ghi

- [start\_transaction, T]: ghi nhận là giao tác T đã bắt đầu thực hiện
- [write\_item, T, X, old\_value, new\_value]: ghi nhận là giao tác T đã thay đổi giá trị của mục tin X từ giá trị old\_value thành new\_value
- [read\_item, T, X]: ghi nhận là giao tác T đã đọc giá trị từ mục tin X
- [commit, T]: ghi nhận là giao tác T đã hoàn tất thành công, tất cả các thay đổi từ giao tác này có thể được lưu trữ vĩnh viễn trong CSDL
- [abort, T]: ghi nhận là giao tác T đã bị huỷ

# Khôi phục dùng sổ ghi

- Khi hệ thống sụp, có thể khôi phục hệ thống về một trạng thái CSDL nhất quán bằng cách rà soát sổ ghi và dùng các kỹ thuật khôi phục, do sổ ghi chứa các thông tin về tất cả các tác vụ thay đổi giá trị của các mục tin CSDL,
- Có thể tháo gỡ (undo) về giá trị trước đó của nó (dùng old\_value)
- Có thể tái thực thi (redo) bằng cách thay đổi các mục tin trong các lệnh WRITE tương ứng về giá trị mới của nó (new\_value). Các tác vụ ghi rồi nhưng chưa kịp lưu trữ bền vững có thể được redo.

# Commit point

- Giao tác T đạt đến commit point khi tất cả tác vụ truy cập CSDL của nó đã hoàn thành và các thay đổi CSDL của nó đã được ghi nhận trong sổ ghi.
- Buộc ghi sổ ghi trước: trước khi một giao tác đạt đến commit point, các phần chưa ghi xuống đĩa của sổ ghi phải được ghi xuống đĩa
- Sau commit point, giao tác được gọi là đã commit:
  - Các tác động của nó đối với CSDL được xem là đã lưu trữ bền vững trong CSDL
  - Ghi mục [commit, T] vào sổ ghi

# Commit point

- Nếu có trục trặc xảy ra:
  - Các giao tác đã ghi mục [start\_transaction, T] vào sổ ghi nhưng chưa có [commit, T] sẽ bị quay ngược (rollback)
  - Các giao tác đã có đủ 2 mục trên được tái thực thi (redo)

# Giao tác trong SQL Server

Có 2 loại:

- Giao tác tự động commit (autocommit transaction) hay giao tác không tường minh.
  - Đây là chế độ mặc nhiên. Mỗi lệnh SQL sẽ được commit hoặc rollback sau khi lệnh kết thúc
- Giao tác tường minh (explicit transaction)
  - Bắt đầu bằng lệnh BEGIN TRANSACTION và kết thúc bằng COMMIT hay ROLLBACK

# Các lệnh liên quan Transaction

- Bắt đầu 1 transaction
  - BEGIN TRAN[SACTION]** [TênTransaction]
- Kết thúc hoàn thành transaction
  - COMMIT TRAN[SACTION]** [TênTransaction]
- Hủy bỏ transaction hoặc quay lui đến điểm dừng
  - ROLLBACK TRAN[SACTION]**  
[TênTransaction | TênSavePoint]
- Đánh dấu vị trí để rollback
  - SAVE TRAN[SACTION]** [TênSavePoint]

# Ví dụ

Begin transaction

insert into Lop (MSLop, TenLop) values ('L5', 'THCB')

select \* FROM Lop → có lớp 'L5'

Save Tran S1

insert into Lop (MSLop, TenLop) values ('L6', 'Mang MT')

select \* FROM Lop → có 2 lớp 'L5' và 'L6'

Rollback Tran S1

select \* FROM Lop → mất lớp 'L6', còn lớp 'L5'

insert into Lop (MSLop, TenLop) values ('L7', 'Do Hoa')

select \* FROM Lop → có lớp 'L5' và 'L7'

Rollback

select \* FROM Lop → mất các lớp 'L5' và 'L7'

# @@ERROR

- Trong một transaction ta thường dùng biến hệ thống @@Error để biết kết quả câu lệnh gần nhất có thực hiện thành công hay không.
- Kết quả trả về
  - Bằng 0 → thực hiện thành công
  - Khác 0 → có lỗi
- Dựa vào kết quả này ta sẽ thực hiện cụ thể lệnh COMMIT hay ROLLBACK

# Tóm tắt

- Các khái niệm về giao tác
- Thuộc tính ACID của giao tác
- Các trạng thái vá tác vụ của giao tác
- Các vấn đề khi xảy ra tương tranh
- Các sự cố gây hỏng giao tác
- Sổ ghi hệ thống (System Log)
- Commit point
- Các lệnh liên quan Transaction trong SQL Server

# Chương 7

# Điều khiển tương tranh



Môn: QUẢN TRỊ CƠ SỞ DỮ LIỆU



# Nội dung

- Kỹ thuật khóa (locking)
- Kỹ thuật khóa hai pha (two-phase locking)
- Khóa chết (Deadlock)
- Các mức độ cô lập của một giao dịch
- Điều khiển tương tranh trong SQL Server

# Điều khiển tương tranh

- Concurrency control
- Mục đích:
  - Thực thi tính cô lập giữa các giao tác có xung đột
  - Thực thi bảo toàn tính nhất quán cho các giao tác
  - Giải quyết các xung đột read-write và write-write
- Ví dụ:
  - Khi T1 và T2 xung đột trên dữ liệu A, thì bộ điều khiển tương tranh sẽ quyết định T1 hoặc T2 sẽ truy cập A và giao tác còn lại sẽ đợi hoặc bị quay ngược

# Kỹ thuật khoá (Lock)

- Khoá (locking):
  - Lock(X): dữ liệu X được khoá để dành quyền đọc hoặc quyền ghi cho giao tác yêu cầu
- Mở khoá (unlocking)
  - Unlock(X): dữ liệu X được chuyển sang trạng thái sẵn sàng để dùng cho tất cả các giao tác
- Well-formed transaction:
  - Tất cả các giao tác cần phải là khoá dữ liệu trước khi đọc hay ghi dữ liệu này
  - Không được khoá một dữ liệu đã khoá bởi chính nó và không được mở khoá một dữ liệu không bị khoá

# Khoá nhị phân (binary lock)

- Mỗi khoá có hai giá trị: locked và unlocked
- Luật về khóa nhị phân:
  - Giao tác T phải dùng `lock_item(X)` trước bất kỳ lệnh `read_item(X)` hoặc `write_item(X)` nào trong T
  - Giao tác T phải dùng `unlock_item(X)` sau khi các lệnh `read_item(X)` và `write_item(X)` đã hoàn thành trong T
  - Giao tác T không dùng lệnh `lock_item(X)` nếu nó đang khoá dữ liệu X
  - Giao tác T không dùng lệnh `unlock_item(X)` trừ khi chính nó đang khoá dữ liệu X

# Khoá và mở khoá nhị phân

## Giải thuật LOCK ITEM(X)

```
B: if LOCK (X) = 0 (*item is unlocked*) then  
    LOCK (X) ← 1 (*lock the item*)  
else begin  
    wait (until lock (X) = 0 and  
          the lock manager wakes up the transaction);  
    goto B  
end;
```

## Giải thuật UNLOCK ITEM(X)

```
LOCK (X) ← 0 (*unlock the item*)  
if any transactions are waiting then  
    wake up one of the waiting transactions;
```

# Chế độ khoá (lock mode)

- Khóa chia sẻ (**shared/read mode**): shared\_lock (X)
  - Các giao tác có thể dùng share lock cùng lúc để đọc mục liệu X
  - Không thể dùng write lock để ghi dữ liệu trên X đang có shared lock
- Khóa loại trừ (**exclusive/write mode**): write\_lock (X)
  - Chỉ có tối đa một write lock trên dữ liệu X tại bất kỳ thời điểm nào
  - Không thể có thêm bất kỳ lock nào khác trên X (kể cả share lock) tại cùng thời điểm

# Giải thuật read\_lock(X)

B: if LOCK (X) = “unlocked” then  
    begin LOCK (X)  $\leftarrow$  “read-locked”;  
        no\_of\_reads (X)  $\leftarrow$  1;  
    end  
else if LOCK (X) = “read-locked” then  
    no\_of\_reads (X)  $\leftarrow$  no\_of\_reads (X) +1  
else begin wait (until LOCK (X) = “unlocked” and  
                 the lock manager wakes up the transaction);  
          go to B  
    end;

# Giải thuật write\_lock(X)

B: if LOCK (X) = “unlocked” then  
    LOCK (X)  $\leftarrow$  “write-locked”;  
    else begin  
        wait (until LOCK (X) = “unlocked” and  
             the lock manager wakes up the transaction);  
        go to B  
    end;

# Giải thuật unlock(X)

```
if LOCK (X) = “write-locked” then begin
    LOCK (X) ← “unlocked”;
    wakes up one of the transactions, if any
End
else if LOCK (X) = “read-locked” then begin
    no_of_reads (X) ← no_of_reads (X) -1
    if no_of_reads (X) = 0 then begin
        LOCK (X) = “unlocked”;
        wake up one of the transactions, if any
    end
end;
```

# Chuyển đổi khoá

Cho phép chuyển đổi khoá:

- Upgrade: đang giữ read\_lock, nâng thành write\_lock
  - if  $T_i$  has a read-lock (X) and  $T_j$  has no read-lock (X) ( $i \neq j$ ) then convert read-lock (X) to write-lock (X)
  - else force  $T_i$  to wait until  $T_j$  unlocks X
- Downgrade: đang giữ write\_lock, chuyển xuống thành read\_lock
  - $T_i$  has a write-lock (X)
    - (\*no transaction can have any lock on X\*)
    - convert write-lock (X) to read-lock (X)

# Khoá hai pha (Two-phase locking)

- Viết tắt: 2PL
- Hai pha (Two Phases):
  - Pha khoá (Locking) hay pha giãn (Growing)
    - Giao tác tiến hành khoá, đọc hay ghi các dữ liệu cần thiết từng cái một
  - Pha mở khoá (Unlocking) hay pha co (Shrinking)
    - Giao tác tiến hành mở khoá các dữ liệu từng cái một
- Yêu cầu trong một giao tác, hai pha này phải loại trừ lẫn nhau:
  - Trong pha khoá, không có tác vụ mở khoá nào
  - Trong pha mở khoá, không có tác vụ khoá nào

# Kỹ thuật khoá không phải hai pha

T1

```
read_lock (Y);  
read_item (Y);  
unlock (Y);  
write_lock (X);  
read_item (X);  
X:=X+Y;  
write_item (X);  
unlock (X);
```

T2

```
read_lock (X);  
read_item (X);  
unlock (X);  
write_lock (Y);  
read_item (Y);  
Y:=X+Y;  
write_item (Y);  
unlock (Y);
```

Giả sử giá trị ban đầu:

X=20; Y=30

Nếu thực hiện tuần tự

T<sub>1</sub> rồi T<sub>2</sub>

X=50, Y=80

Nếu thực hiện tuần tự

T<sub>2</sub> rồi T<sub>1</sub>

X=70, Y=50

T1

```
read_lock (Y);  
read_item (Y);  
unlock (Y);
```

```
write_lock (X);  
read_item (X);  
X:=X+Y;  
write_item (X);  
unlock (X);
```

T2

```
read_lock (X);  
read_item (X);  
unlock (X);  
write_lock (Y);  
read_item (Y);  
Y:=X+Y;  
write_item (Y);  
unlock (Y);
```

Tương tranh, kết quả:

X=50; Y=50

**Không khả tuần tự hoá**

# Kỹ thuật khoá hai pha

T'1

```
read_lock (Y);  
read_item (Y);  
write_lock (X);
```

```
unlock (Y);  
read_item (X);  
X:=X+Y;  
write_item (X);  
unlock (X);
```

Đảm bảo tính  
khả tuần tự hoá

T'2

**read\_lock (X);**

```
read_lock (X);  
read_item (X);  
write_lock (Y);  
unlock (X);  
read_item (Y);  
Y:=X+Y;  
write_item (Y);  
unlock (Y);
```

Không khoá được  
nên T'2 phải đợi

# Kỹ thuật khoá hai pha

- Nếu tất cả các giao tác trong lịch biểu đều dùng kỹ thuật khoá hai pha, lịch biểu được đảm bảo sẽ là khả tuần tự hoá
- Giảm mức độ tương tranh:
  - T1 đang khoá X và đã dùng xong
  - T1 cần khoá thêm Y, Z, ... để tiếp tục các tác vụ
  - T1 không mở khoá X vì đang ở pha khoá
  - Những giao tác khác cần truy cập X buộc phải đợi T1 chuyển qua pha mở, và sau khi T1 mở khoá X thì mới có thể khoá X được

# Khoá chết (deadlock)

T'1

**read\_lock (Y);**  
read\_item (Y);

**write\_lock (X);**

Không khoá được  
nên T'1 phải đợi T'2

T'2

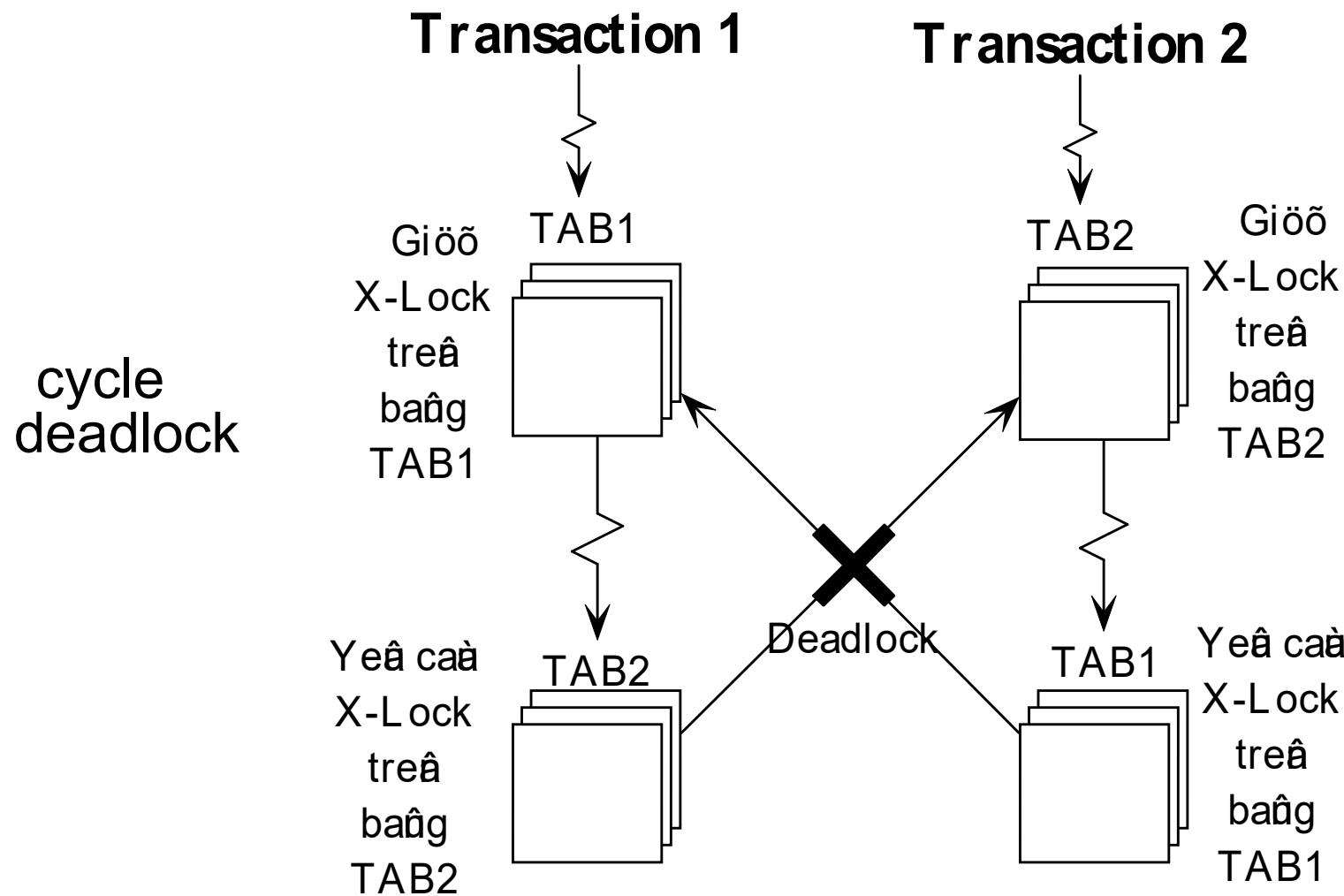
**read\_lock (X);**  
read\_item (X);

**write\_lock (Y);**  
~~unlock (X);~~  
~~read\_item (Y);~~  
~~Y:=X+Y;~~  
~~write\_item (Y);~~  
~~unlock (Y);~~

Không khoá được  
nên T'2 phải đợi T'1

~~unlock (Y);~~  
~~read\_item (X);~~  
~~X:=X+Y;~~  
~~write\_item (X);~~  
~~unlock (X);~~

# Khoá chết (deadlock)



# Khoá hai pha bảo thủ

- Khoá hai pha bảo thủ (Conservative 2PL) hay khoá hai pha tĩnh (Static 2PL)
- Ngăn chặn deadlock bằng cách khoá tất cả các dữ liệu cần thiết trước khi bắt đầu thực hiện giao tác
- Cách thức:
  - Khai báo hai tập các dữ liệu cần đọc và cần ghi của giao tác, tiến hành khoá các dữ liệu trong hai tập này
  - Nếu không khoá được bất kỳ một dữ liệu nào, thì huỷ toàn bộ các khoá và đợi để khoá lại từ đầu
- Là giao tác không có khoá chết (deadlock-free)
- Tuy nhiên không thực tế

# Khoá hai pha nghiêm cách

- Strict 2PL
- Giao tác không mở khoá ghi (write/ exclusive lock) của nó cho đến khi kết thúc bằng commit hay abort
- Như vậy, không giao tác nào có thể đọc hoặc ghi các mục liệu đã ghi bởi giao tác này cho đến khi nó kết thúc
- Thoả lịch biểu khả khôi phục nghiêm cách (strict schedule)
- **Không phải** là giao tác deadlock-free

# Khoá hai pha khắt khe

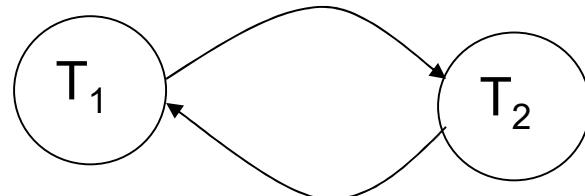
- Rigorous 2PL
- Khắt khe hơn Strict 2PL
- Đòi hỏi các giao tác chỉ mở khoá (đọc hoặc ghi) khi kết thúc bằng commit hay abort
- Thoả mãn lịch biểu nghiêm cách, và dễ hiện thực hơn

# Phát hiện khoá chết

- Cho phép khoá chết xảy ra, nhưng phát hiện để giải quyết
- Bộ định lịch biểu (scheduler) duy trì một đồ thị đợi (wait-for-graph) để phát hiện chu trình
  - Mỗi giao tác trong lịch biểu là một đỉnh của đồ thị
  - Khi giao tác  $T_i$  đợi để khoá mục liệu X đang khoá bởi giao tác  $T_j$ , thêm cạnh  $T_i \rightarrow T_j$  vào đồ thị
- Nếu có chu trình  $T_i \rightarrow T_j \rightarrow T_k \rightarrow \dots \rightarrow T_i$  (tức có khoá chết), một giao tác nào đó sẽ được chọn là vật hy sinh (victim) để quay ngược (rollback)

# Đồ thị đợi (wait-for graph)

| <u>T1</u>              | <u>T2</u>              |
|------------------------|------------------------|
| read_lock (Y);         |                        |
| read_item (Y);         |                        |
| <b>write_lock (X);</b> | read_lock (X);         |
| (đợi T'2 mở khoá X)    | read_item (X);         |
|                        | <b>write_lock (Y);</b> |
|                        | (đợi T'1 mở khoá Y)    |



# Tránh khoá chết

- Không cho phép chu trình xuất hiện. Khi phát hiện một giao tác có nguy cơ gây ra chu trình, quay ngược giao tác này
- Giải thuật Wound-Wait và Wait-Die sử dụng tem thời gian (timestamp) để tránh khoá chết
- Tem thời gian (timestamp):
  - Là một giá trị duy nhất gán cho mỗi giao tác. Thường dựa trên thứ tự xuất hiện của giao tác
  - Tem thời gian của T viết tắt là TS(T)
  - Nếu giao tác T1 bắt đầu trước (older) giao tác T2 (younger), thì  $TS(T1) < TS(T2)$

# Tránh khoá chết

## ■ Wait-die:

- Nếu  $TS(T_i) < TS(T_j)$ , tức  $T_i$  vào trước  $T_j$ ,  $T_i$  sẽ đợi
- Ngược lại, tức  $T_i$  vào sau  $T_j$ , huỷ  $T_i$  và khởi động lại sau với cùng một tem thời gian

## ■ Wound-wait:

- Nếu  $TS(T_i) < TS(T_j)$ , tức  $T_i$  vào trước  $T_j$ , huỷ  $T_j$  và khởi động lại sau với cùng một tem thời gian
- Ngược lại, tức  $T_i$  vào sau  $T_j$ ,  $T_i$  sẽ đợi

# Tình trạng đợi quá lâu (Starvation)

- Tình trạng đợi quá lâu (starvation) xảy ra khi một giao tác cứ tiếp tục đợi hoặc bị tái khởi động và không bao giờ có cơ hội được thi hành
- Ví dụ khi giải quyết khoá chết, một giao tác có thể bị lựa chọn là vật hy sinh mãi để quay ngược (có thể do bộ định lịch biểu dùng nguyên lý ưu tiên)
- Kỹ thuật Wait-die và Wound-wait có thể giúp tránh tình trạng này

# Điều khiển tương tranh trong SQL-Server

# Cơ chế điều khiển tương tranh

- Optimistic concurrency control:
  - Giả định có rất ít xung đột giữa các người dùng, nên tài nguyên không cần khoá mà chỉ được kiểm tra khi giao tác thay đổi dữ liệu
  - Không có sẵn trong SQL-Server
- Pessimistic concurrency control
  - Khoá tài nguyên muốn dùng khi thực hiện giao tác
  - SQL-Server dùng cơ chế này
- Người dùng đặc tả kiểu điều khiển tương tranh qua transaction isolation level và concurrency options dành cho cursor

# Mức độ cô lập

- Ý nghĩa :

Mức độ cô lập của 1 transaction quy định mức độ nhạy cảm đối với những thay đổi trên CSDL do các transaction khác tạo ra .

Mức độ cô lập của transaction cũng quy định thời gian giữ lock.

- Các mức độ cô lập:

- Read Uncommitted
- **Read Committed (mặc định)**
- Repeatable Read
- Serializable

# Read Uncommitted

- Đặc điểm:
  - Không đặt Shared Lock trên những dữ liệu cần đọc
  - Không bị ảnh hưởng bởi lock của các giao tác khác trên những dữ liệu cần đọc
  - Không phải chờ khi đọc dữ liệu (kể cả khi dữ liệu đang bị lock bởi giao tác khác)
- Ưu điểm là **tốc độ xử lý rất nhanh**. Không cần chờ những giao tác khác thực hiện việc cập nhật dữ liệu
- Khuyết điểm là có khả năng xảy ra mọi vấn đề trong việc xử lý các giao dịch tương tranh, đặc biệt là **Dirty Reads**

# Read Committed

- Đặc điểm:
  - Đây là mức độ cô lập **mặc định** của SQL Server
  - **Tạo Shared Lock** trên dữ liệu được đọc, Shared Lock được giải phóng ngay khi đọc xong dữ liệu
  - **Tạo Exclusive Lock** trên đơn vị dữ liệu được ghi, Exclusive Lock được **giữ cho đến hết giao tác**

# Read Committed

- Ưu điểm:
  - Giải quyết được vấn đề Dirty Reads
  - Shared Lock được giải phóng ngay, không giữ đến hết giao tác nên không ngăn cản thao tác cập nhật của các giao tác khác
- Khuyết điểm:
  - Chưa giải quyết được vấn đề **Unrepeatable Reads, Phantoms, Lost Updates**
  - Phải chờ khi chưa thể được đáp ứng yêu cầu lock trên dữ liệu đang bị giữ lock bởi giao tác khác.

# Repeatable Read

- Đặc điểm:
  - Tạo Shared Lock trên dữ liệu cần đọc
  - Shared Lock được giữ cho đến hết giao tác → không cho phép các giao tác khác cập nhật, thay đổi giá trị trên dữ liệu này.
  - Tạo Exclusive Lock trên dữ liệu cần ghi, Exclusive Lock được giữ cho đến hết giao tác.

# Repeatable Read

- Ưu điểm:
  - Giải quyết được Dirty Reads, Unrepeatable Reads và Lost Update
- Khuyết điểm:
  - Chưa giải quyết được vấn đề **Phantoms**. Vì vẫn cho phép Insert những dòng dữ liệu mới
  - Phải chờ khi chưa được đáp ứng yêu cầu lock trên dữ liệu đang bị giữ lock bởi giao tác khác.
  - Các giao tác khác không được phép cập nhật trên những dữ liệu đang bị giữ Shared Lock.

# Serializable

- Đặc điểm:
  - Tạo Shared Lock trên dữ liệu cần đọc
  - Shared Lock được giữ cho đến hết giao tác
  - Không cho Insert dữ liệu thỏa mãn điều kiện thiết lập Shared Lock (sử dụng khóa trên 1 miền giá trị - Key Range Lock)
  - Tạo Exclusive Lock trên đơn vị dữ liệu cần ghi, Exclusive Lock được giữ cho đến hết giao tác.

# Serializable

- Ưu điểm:
  - Giải quyết được Dirty Read, Unrepeatable Read và Phantom
- Khuyết điểm:
  - Phải chờ khi chưa thể được đáp ứng yêu cầu lock trên đơn vị dữ liệu đang bị giữ lock bởi giao tác khác.

# Thiết lập mức độ cô lập

- Cú pháp:

SET TRANSACTION ISOLATION LEVEL

```
{ READ COMMITTED  
| READ UNCOMMITTED  
| REPEATABLE READ  
| SERIALIZABLE  
}
```

# Thiết lập lock khi dùng lệnh SELECT

```
SELECT <danh sách các cột>
FROM <table> WITH (
    { FASTFIRSTROW | HOLDLOCK
    | NOLOCK | NOWAIT | PAGLOCK
    | READCOMMITTED
    | READCOMMITTEDLOCK
    | READPAST | READUNCOMMITTED
    | REPEATABLEREAD | ROWLOCK
    | SERIALIZABLE | TABLOCK | TABLOCKX
    | UPDLOCK | XLOCK } )
```

# Ví dụ

- Transaction 1

```
SET TRANSACTION
```

```
    ISOLATION LEVEL SERIALIZABLE
```

```
BEGIN TRANSACTION
```

```
INSERT INTO Phong(MSPB) VALUES ('10')
```

- Transaction 2

```
SELECT * FROM Phong
```

```
    WITH (NOLOCK)
```

# Giải quyết khoá chết

- Dùng đồ thị đợi (wait-for graph) để mô tả khoá chết
- Dùng chế độ lock timeout trong khi khoá tài nguyên để tránh khoá chết
- Khi SQL-Server chọn một giao tác làm vật hy sinh khi xảy ra khoá chết, giao tác sẽ bị quay ngược với thông báo lỗi 1205
  - *Your transaction (process ID #52) was deadlocked on {lock | communication buffer | thread} resources with another process and has been chosen as the deadlock victim. Rerun your transaction*

# Two-phase commit trong SQL-Server

- Pha yêu cầu commit (commit request phase):
  - Gửi yêu cầu commit đến tất cả các giao tác liên quan, như giao tác phân bố hoặc giao tác lồng
  - Các giao tác liên quan sẽ commit và trả kết quả về là thành công hoặc thất bại

# Two-phase commit trong SQL-Server

- Pha commit (commit phase):
  - Nếu giao tác nhận được commit thành công từ tất cả các giao tác liên quan
    - Gửi thông báo commit thành công cho tất cả các giao tác liên quan
  - Nếu một giao tác liên quan không thành công
    - Gửi yêu cầu quay ngược (rollback) cho tất cả các giao tác liên quan
    - Các giao tác liên quan quay ngược tất cả các tác vụ của mình

# Two-phase commit trong SQL-Server

- Khi thực hiện lệnh COMMIT, SQL-Server sẽ:
  - Đánh dấu kết thúc giao tác
  - Nếu `@@TRANCOUNT = 1`, ghi tất cả các thay đổi vào CSDL, gán `@@TRANCOUNT = 0`
  - Nếu `@@TRANCOUNT > 1`, giảm `@@TRANCOUNT` đi 1 và giao tác vẫn còn hoạt động
- Ở trong chế độ phân bố:
  - Nếu là các server khác nhau: dùng tiến trình MSDTC (Microsoft Distributed Transaction Coordinator)
  - Nếu là các CSDL trong cùng một server: giải thuật commit hai pha nội

# Tóm tắt

- Kỹ thuật khóa (locking)
- Kỹ thuật khóa hai pha (two-phase locking)
- Khóa chết (Deadlock)
- Các mức độ cô lập của một giao dịch
- Điều khiển tương tranh trong SQL Server

# Chương 8

## Sao lưu và phục hồi



Môn: QUẢN TRỊ CƠ SỞ DỮ LIỆU



# Nội dung

- Các loại hư hỏng trong hệ thống
- Các loại lưu trữ và việc thực hiện lưu trữ bền vững
- Chiến lược sao lưu
- Phân biệt restore và recovery
- Các phương pháp phục hồi

# Khái niệm

- Backup - restore là nhiệm vụ quan trọng của người quản trị CSDL (DBA - Database Administrator)
- DBA có trách nhiệm giữ cho hệ thống CSDL vận hành thông suốt, ngay cả khi hệ thống gặp sự cố
- DBA phải luôn theo dõi, kiểm tra thường xuyên để phát hiện các trục trặc trước khi nó xảy ra → giảm tối đa số lần phải phục hồi dữ liệu
- DBA phải dự phòng các sự cố có thể xảy ra và bảo đảm rằng có thể nhanh chóng phục hồi dữ liệu trong thời gian sớm nhất có thể được

# Nguyên nhân gây hư hỏng hệ thống

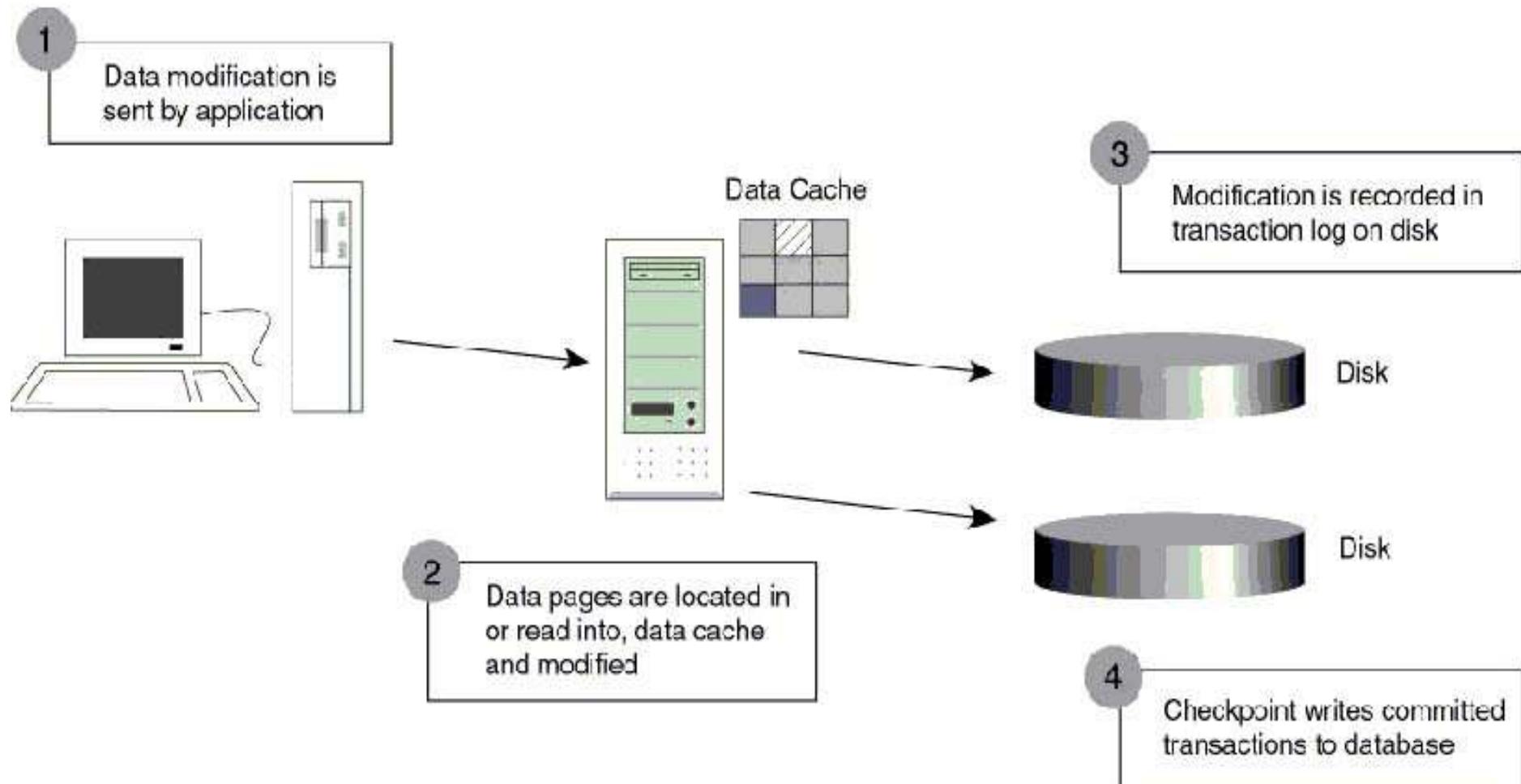
- Hư phần cứng
  - Server bị hư, đĩa cứng hư, ...
  - Những thảm họa tự nhiên như hỏa hoạn, ...
  - Toàn bộ server bị đánh cắp hoặc phá hủy
- Hư phần mềm
  - Hệ điều hành, SQL Server
  - Data file, transaction log file, system file bị hư, ...
- Con người
  - Những lỗi do vô ý như lỡ tay delete toàn bộ table
  - Những hành vi mang tính phá hoại của nhân viên
  - Bị hack ...

# Transaction Log

- Transaction log file dùng để ghi lại các thay đổi xảy ra trong database.
- Khi có lệnh Insert, Update, Delete từ các ứng dụng, SQL Server sẽ tải (load) trang dữ liệu (data page) tương ứng lên bộ nhớ (gọi là data cache)
- Dữ liệu trong data cache được cập nhật theo lệnh gọi (các trang bị sửa đổi còn gọi là dirty-page).
- Các thay đổi sau đó được ghi vào transaction log file (vì vậy còn gọi là write-ahead log).
- Cuối cùng một quá trình gọi là Check Point Process sẽ kiểm tra và ghi tất cả những transaction đã hoàn tất (committed) vào đĩa cứng (gọi là flushing the page)

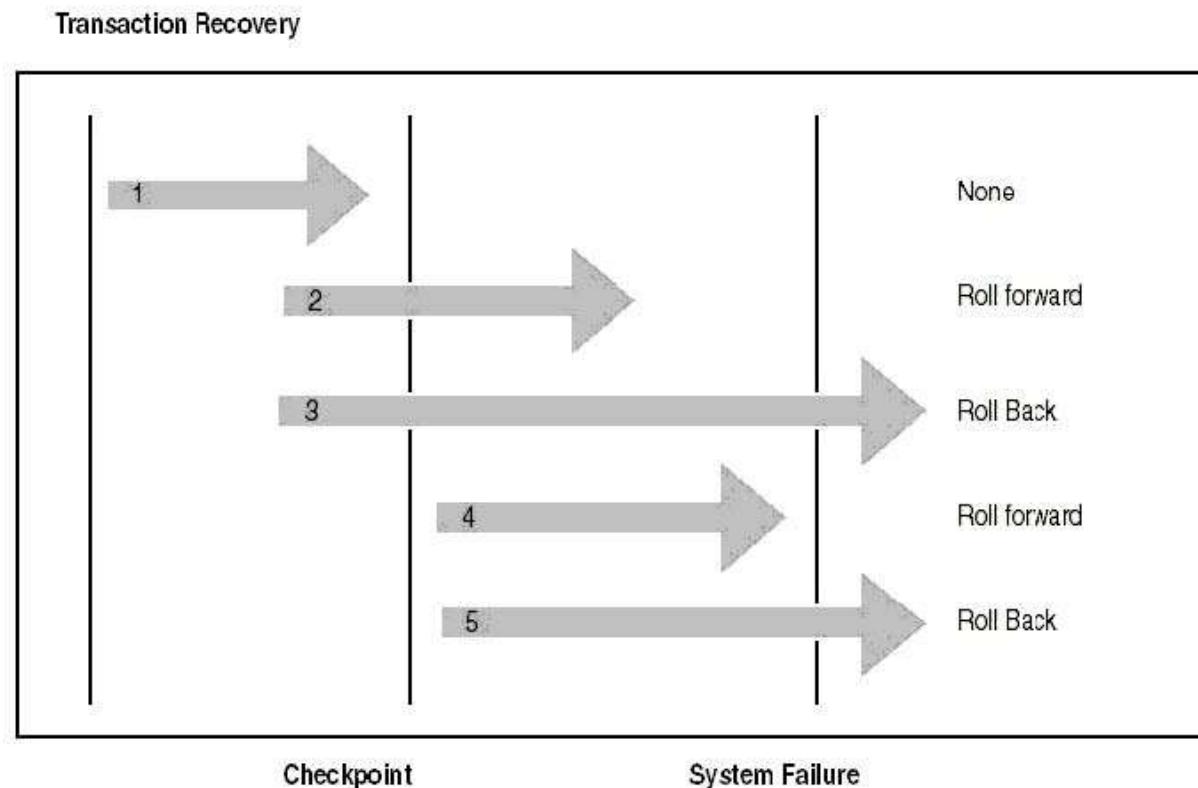
# Transaction Log

- Ngoài Check Point Process, dirty-page còn được đưa vào đĩa bởi một Lazy writer. Đây là background process, nó quét data cache theo một chu kỳ nhất định.



# Transaction Log

- Giả sử một Check point xảy ra vào thời điểm giữa transaction 2 và 3 như hình vẽ và sau đó sự cố xảy ra trước khi gấp một Check point kế tiếp.



# Transaction Log

- Khi SQL Server được khởi động lại nó sẽ dựa vào transaction log file để phục hồi dữ liệu
- Không cần làm gì cả đối với transaction 1 vì tại thời điểm Check point dữ liệu đã được lưu vào đĩa.
- Transaction 2 và 4 sẽ được **Roll Forward**, vì tuy đã commit nhưng do sự cố xảy ra trước thời điểm check point kế tiếp nên dữ liệu chưa kịp lưu vào đĩa. SQL Server sẽ ghi lại dữ liệu vào đĩa cứng từ log file
- Transaction 3 và 5 chưa committed, nên sẽ được **Rollback** dựa vào dữ liệu trong log file

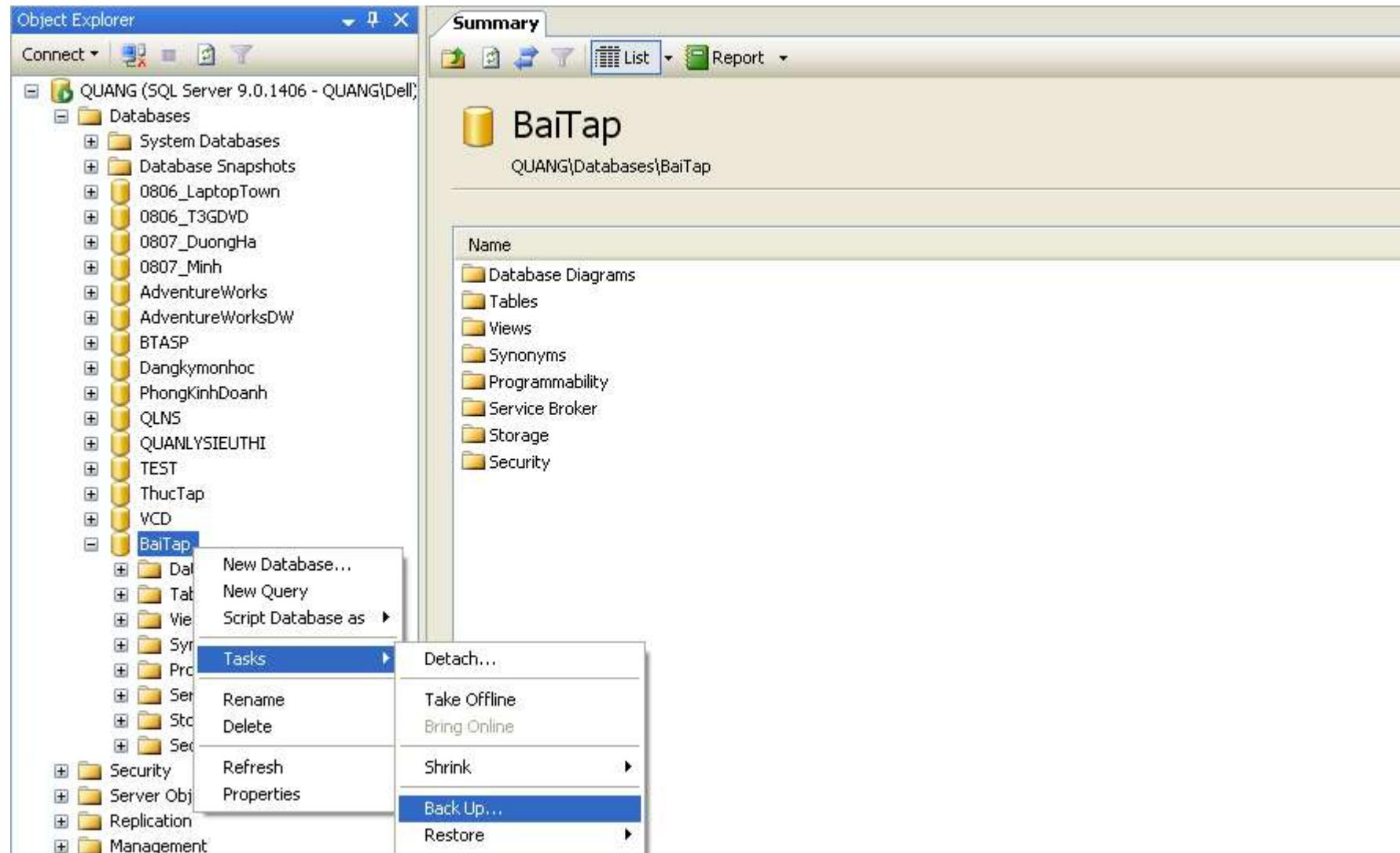
# Lệnh CHECKPOINT

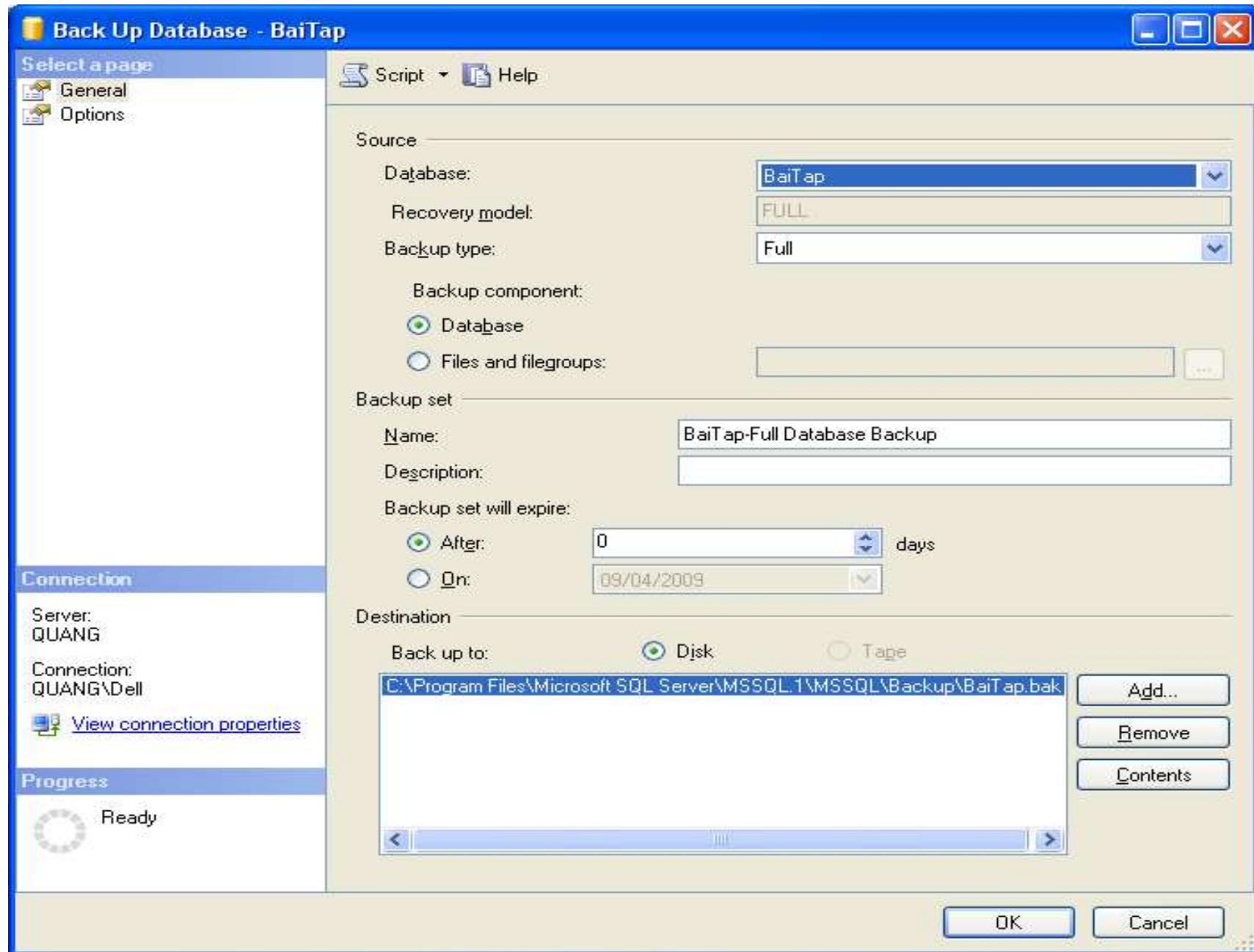
- Cú pháp:  
    CHECKPOINT [ checkpoint\_duration ]
- Công dụng:
  - Ghi tất cả các dirty page vào đĩa
  - SQL-Server tự động thực hiện lệnh này trước khi sao lưu và trong một số trường hợp như khi transaction log đầy đến 70%

# Các loại backup

- Full (Complete) database backup:
  - Sao chép toàn bộ dữ liệu trong database
- Differential database backup:
  - Chỉ sao lưu những gì thay đổi so với lần full backup gần nhất → quá trình xử lý nhanh hơn
- File or File Group Backup:
  - Sao chép một data file đơn hay một file group.
- Differential File or File Group Backups:
  - Tương tự differential database backup nhưng chỉ lưu những thay đổi trong data file đơn hay một file group
- Transaction Log Backups:
  - Lưu theo thứ tự tất cả các transactions chứa trong transaction log file

# Backup bằng SQL Server Management





# Backup Full Database

## ■ BACKUP DATABASE

{*database\_name* | @*database\_name\_var*}

TO < backup\_device > [ ,...n ]

WITH <option>

## ■ Dùng backup device:

```
EXEC sp_addumpdevice 'disk', 'BaiTapData',
'D:\Backup\BaiTapData.bak'
```

BACKUP DATABASE BaiTap TO BaiTapData

## ■ Chỉ trực tiếp file backup:

BACKUP DATABASE BaiTap

TO DISK = 'D:\Backup\BaiTapData.bak'

# Backup Differential Database

## ■ BACKUP DATABASE

{database\_name | @database\_name\_var }

TO < backup\_device > [ ,...n ]

WITH DIFFERENTIAL

## ■ Ví dụ:

BACKUP DATABASE BaiTap

TO DISK = "D:\Backup\BaiTapData.bak"

WITH DIFFERENTIAL

# Backup Log

## ■ BACKUP LOG

```
{ database_name | @database_name_var }  
TO <backup_device> [ ,...n ]
```

## ■ Ví dụ:

```
BACKUP LOG BaiTap  
TO DISK = "D:\Backup\BaiTapLog.bak"
```

## ■ Backup Tail-log

```
BACKUP LOG Test  
TO DISK='C:\Backup\Test_logDetail2.bak'  
WITH no_truncate
```

# Backup files và filegroups

## ■ BACKUP DATABASE

{ database\_name | @database\_name\_var }

<file\_or\_filegroup> [ ,...f ]

TO <backup\_device> [ ,...n ]

## ■ Ví dụ:

Backup database BaiTap

**FILE= 'BaiTap'**

TO DISK = 'D:\Backup\BTapData.bk'

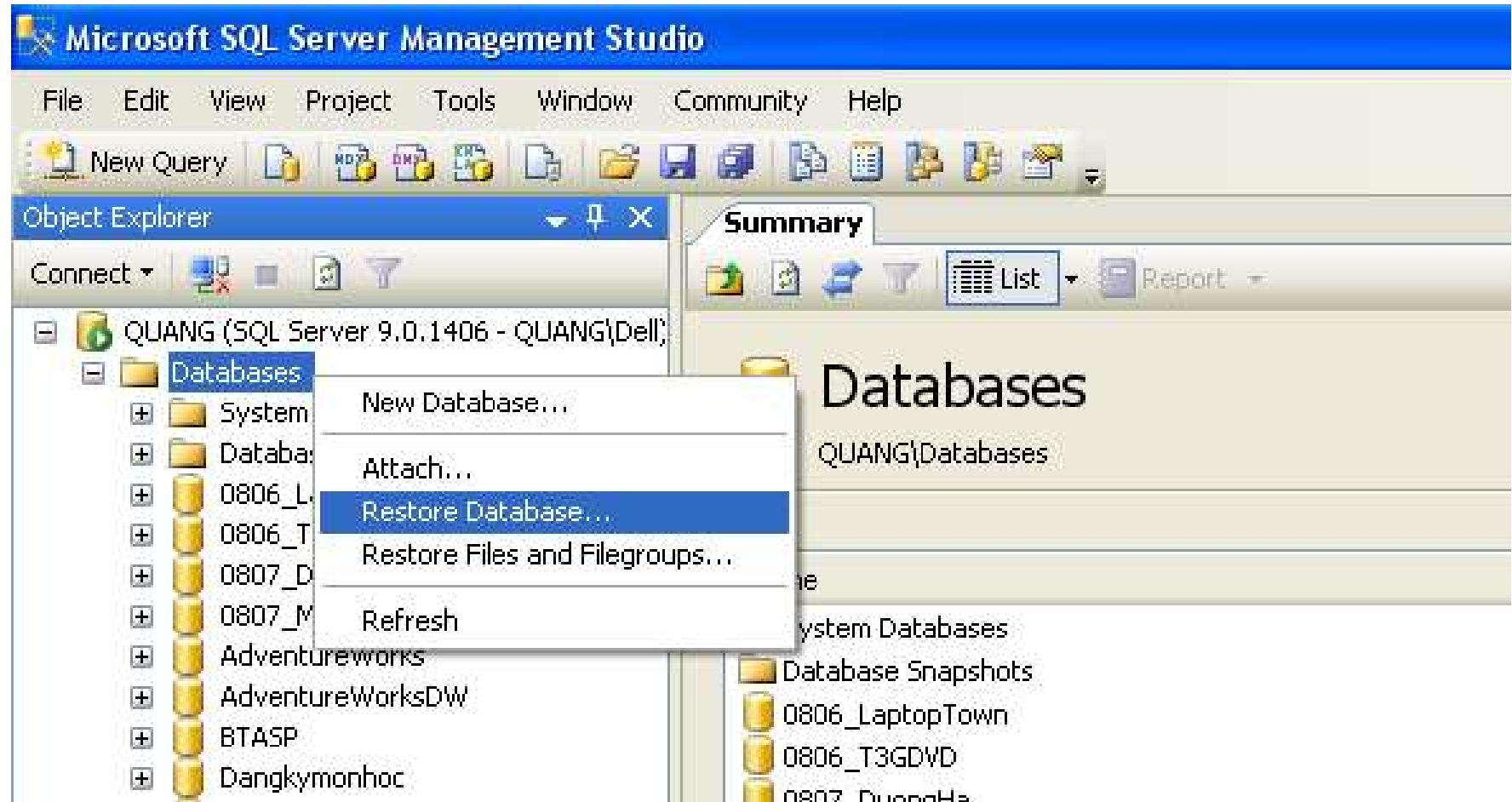
# Các mô hình khôi phục

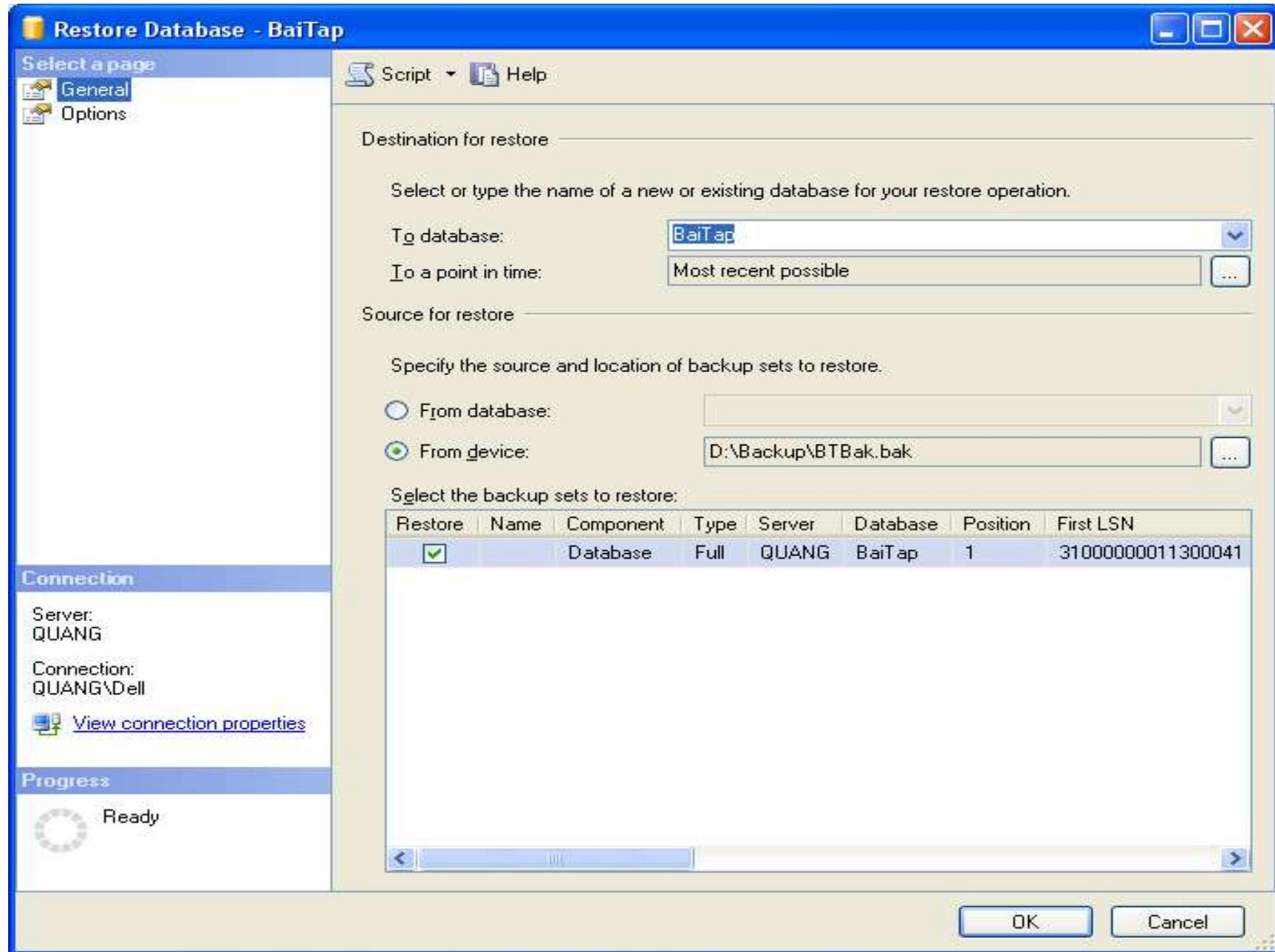
- Đơn giản (Simple recovery model):
  - Không cần sao lưu log
  - Chỉ có thể khôi phục đến một bản sao lưu gần nhất
    - Mất các dữ liệu từ lần sao lưu gần nhất đến hiện tại
- Đầy đủ (Full recovery model):
  - Cần phải sao lưu log
  - Có thể khôi phục CSDL đến một thời điểm nào đó trước khi khôi phục (có sự cố xảy ra)
- Ghi sổ gộp (Bulk logged recovery model):
  - Giống full recovery model nhưng các tác vụ gộp (nạp gộp dữ liệu – bulk import) sẽ được ghi dạng gộp

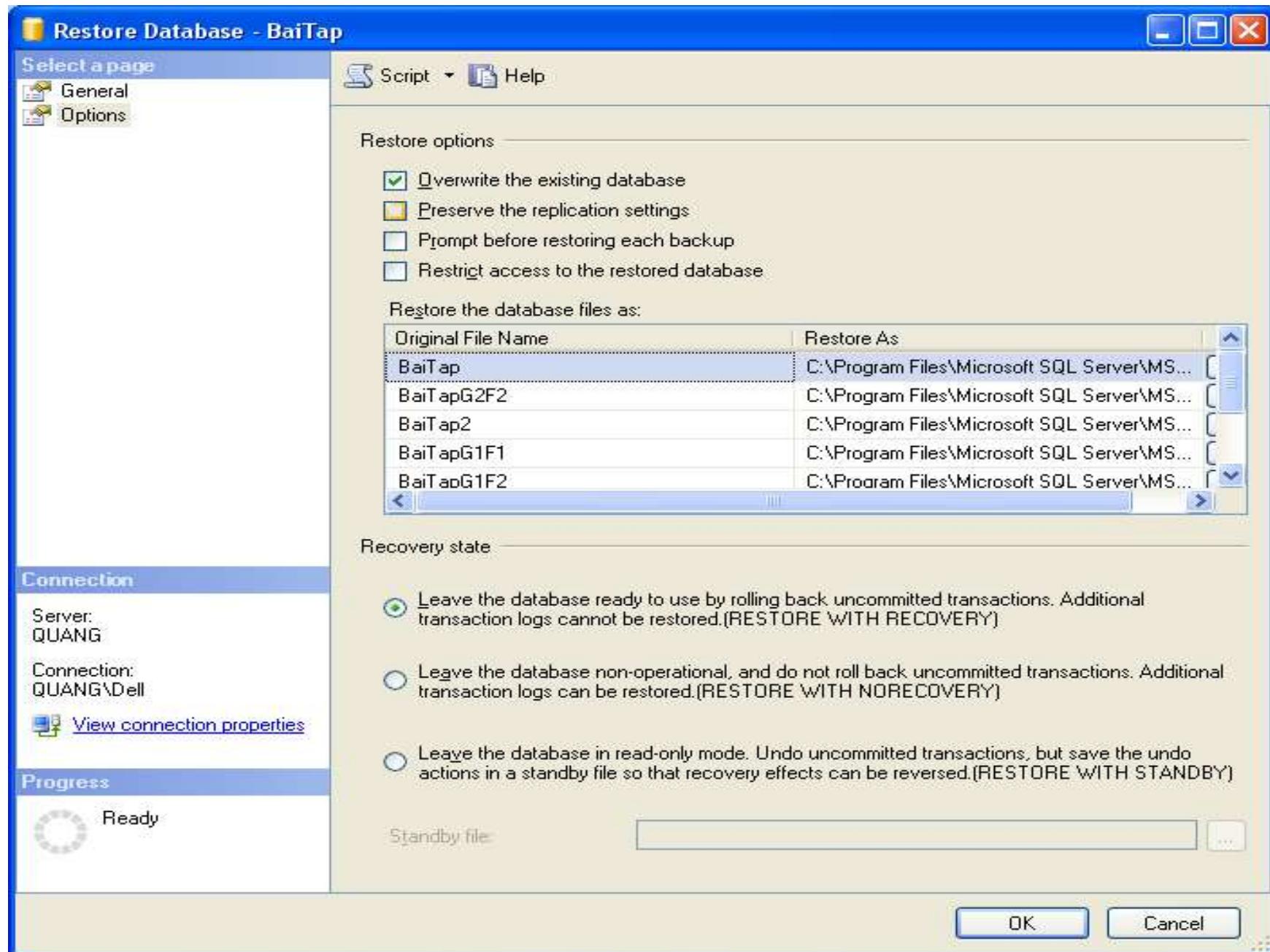
# Restore

- Restore CSDL từ tập tin full backup khá dễ dàng
- Restore từ tập tin differential backup: trước hết phải restore bản full backup lần cuối cùng, sau đó mới restore bảng differential backup
- Để phục hồi nguyên trạng CSDL trước khi bị hỏng, sau khi restore bản differential backup cuối cùng, ta lần lượt phục hồi tất cả các bản transaction log backup

# Restore bằng SQL Server Management







# Lệnh Restore Database

- RESTORE DATABASE

```
{ database_name | @database_name_var }
```

```
[FROM <backup_device> [ ,...n ]]
```

```
[ WITH option ]
```

- Ví dụ:

RESTORE DATABASE BaiTap

FROM DISK = 'D:\Backup\Data1.bak'

# Restore File, FileGroup

## ■ RESTORE DATABASE

```
{ database_name | @database_name_var }  
<file_or_filegroup_or_pages> [ ,...f ]  
[ FROM <backup_device> [ ,...n ] ]  
[ WITH option]
```

# Restore Log

## ■ RESTORE LOG

```
{ database_name | @database_name_var }
[ <file_or_filegroup_or_pages> [ ,...f ] ]
[ FROM <backup_device> [ ,...n ] ]
[ WITH option]
```

# Phân biệt Restore và Recovery

- Restore database từ một file backup, chỉ đơn giản chép lại vào database từ những file backup và thực thi lại những transaction đã được commit
- Nhưng database có thể ở trạng thái chưa nhất quán (inconsistent) và chưa sử dụng được.
- Recovery: không chỉ phục hồi lại dữ liệu mà còn bảo đảm cho nó ở trạng thái nhất quán (consistent) và có thể sử dụng được.

# Các option khi Restore database

- Nếu chọn option **WITH RECOVERY** sẽ roll back các transaction chưa được committed và database có thể hoạt động bình thường, nhưng ta **không thể restore thêm backup file** nào nữa, thường chọn khi restore file backup cuối cùng trong chuỗi backup.
- Nếu chọn option **WITH NORECOVERY** các transaction chưa được committed sẽ không được roll back, do đó SQL Server sẽ không cho phép ta sử dụng database, nhưng ta **có thể tiếp tục restore các file backup** kế tiếp, thường chọn khi sau đó ta còn phải restore các file backup khác.

# Ví dụ

- Restore full backup

```
RESTORE DATABASE BaiTap  
FROM DISK = 'D:\Backup\DataFull.bak'  
WITH NORECOVERY
```

- Restore tiếp differential backup và recovery

```
RESTORE DATABASE BaiTap  
FROM DISK = 'D:\Backup\DataDiff.bak'
```

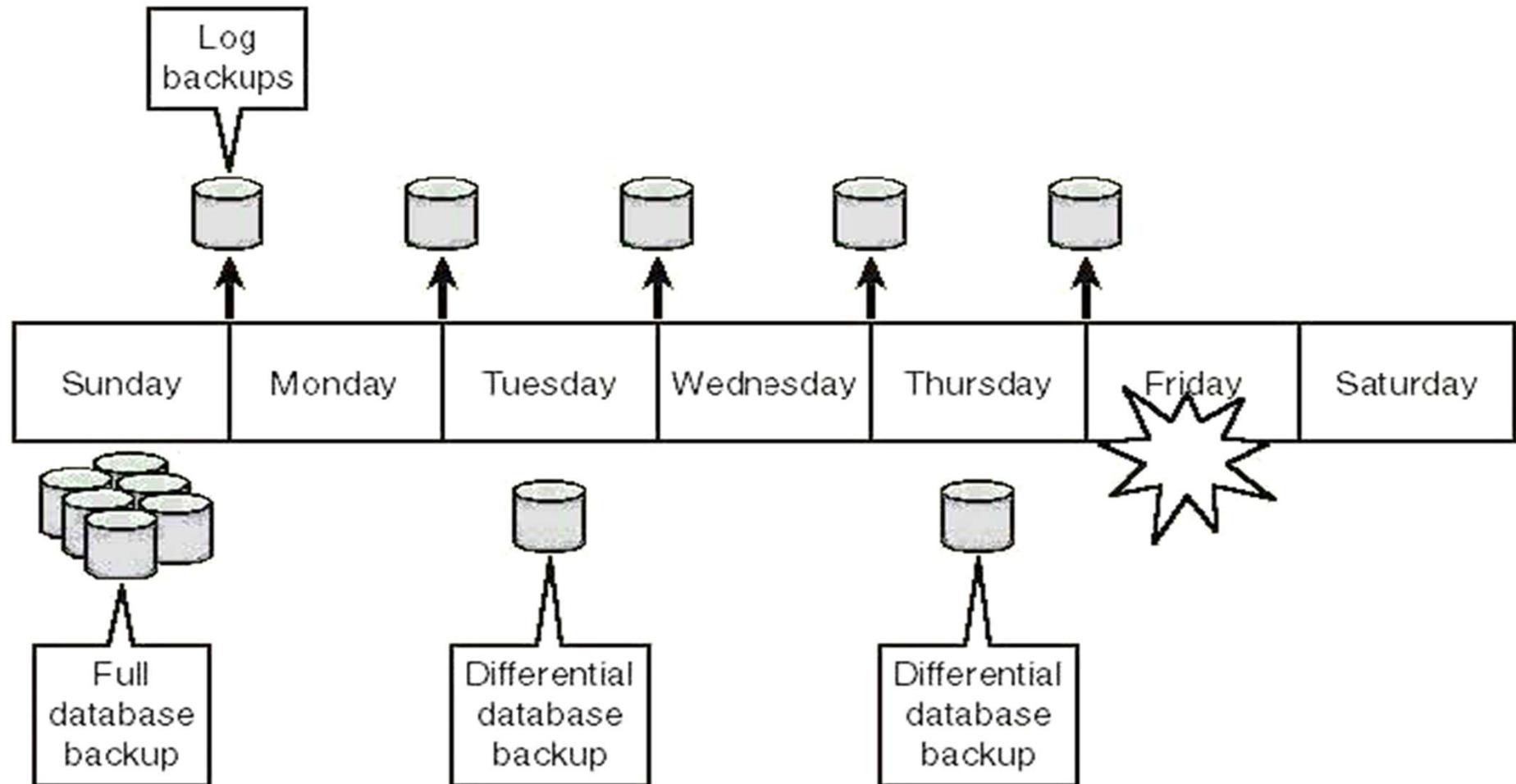
# Các option khi Restore database

## ■ Option **WITH STANDBY**

Có đặc tính của cả hai option trên, các transaction chưa hoàn tất sẽ được roll back để đảm bảo tính nhất quán (consistent) và có thể sử dụng được nhưng chỉ dưới dạng Read-only mà thôi. Sau đó ta vẫn có thể tiếp tục restore các file backup còn lại.

Dùng option này khi muốn restore database trở lại một thời điểm nào đó, nhưng không rõ đó có phải là thời điểm mong muốn không, nên ta restore từng backup file ở dạng Standby và kiểm tra một số data xem có đúng điểm muốn restore hay không

# Xây dựng chiến lược backup



# Xây dựng chiến lược backup

- Ta có thể lập lịch (schedule) tự động cho việc backup. Ví dụ:
  - Full Database Backup vào Chủ Nhật hàng tuần
  - Differential Backup vào các ngày thứ Ba và Thứ Năm hàng tuần
  - Transaction Log Backup thực hiện hằng ngày
- Giả sử vào ngày Thứ Sáu, sự cố xảy ra: đĩa chứa data file của database bị hư.

Làm cách nào phục hồi dữ liệu và đưa database trở lại hoạt động bình thường?

# Phục hồi dữ liệu theo tình huống

- Nếu Transaction Log File không bị hư do chứa trong một đĩa khác với đĩa chứa Data File → backup ngay Transaction Log File.
- Người ta gọi file backup này là "the tail of the log" (phần đuôi). Ta dùng option NO\_TRUNCATE để đảm bảo tính nhất quán của CSDL
- Nếu Log File được chứa trên cùng một đĩa với data file thì ta có thể sẽ không backup được tail-log và như vậy ta phải dùng đến log file backup gần nhất → chấp nhận mất một phần dữ liệu

# Phục hồi dữ liệu theo tình huống

- Restore database từ file Full Backup của ngày Chủ Nhật.
  - Lưu ý dùng option WITH NORECOVERY để có thể tiếp tục restore các file khác.
- Kế tiếp restore differential backup của ngày thứ 5
- Sau đó lần lượt restore các Transaction Log Backup kể từ sau lần Differential Backup cuối cùng (Transaction Log Backup của ngày thứ 5)
- Cuối cùng restore Tail-log backup dùng option WITH RECOVERY.

# Detach Database

- Data và transaction log files của một database có thể được gỡ ra (detach) và sau đó gán lại (reattach) vào SQL Server
- Detach và attach rất thuận lợi khi cần di chuyển một database từ SQL Server của máy này sang máy khác

```
sp_detach_db [ @dbname= ] 'dbname'
```

- Ví dụ: EXEC sp\_detach\_db 'BaiTap'

# Attach Database

- CREATE DATABASE database\_name  
ON <filespec> [ ,...n ]  
FOR { ATTACH | ATTACH\_REBUILD\_LOG }
  
- Ví dụ:  
CREATE DATABASE BaiTap  
ON (FILENAME = 'D:\Backup\BaiTap.mdf'),  
    (FILENAME = 'D:\Backup\BaiTap\_log.ldf')  
FOR ATTACH

# Attach Database

- sp\_attach\_db
  - [ @dbname= ] 'dbname' ,
  - [ @filename1= ] 'filename\_n' [ ,...16 ]
- Ví dụ:
  - EXEC sp\_attach\_db 'BaiTap',  
'D:\Backup\BaiTap.mdf'.  
'D:\Backup\BaiTap\_log.ldf'
- Ghi chú: Sp này được Microsoft cảnh báo có thể bỏ trong các phiên bản sau, vì vậy được khuyến cáo không nên sử dụng

# Tóm tắt

- Các loại hư hỏng trong hệ thống
- Các loại lưu trữ và việc thực hiện lưu trữ bền vững
- Chiến lược sao lưu
- Phân biệt restore và recovery
- Các phương pháp phục hồi