



HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

Chương 0: **GIỚI THIỆU**

Giáo viên lý thuyết:



Nguyễn Trường Sơn (ntson@fit.hcmus.edu.vn)

Quy tắc gửi email:



Subject: [DTTX] HQTCSDL 2014 Tieude email

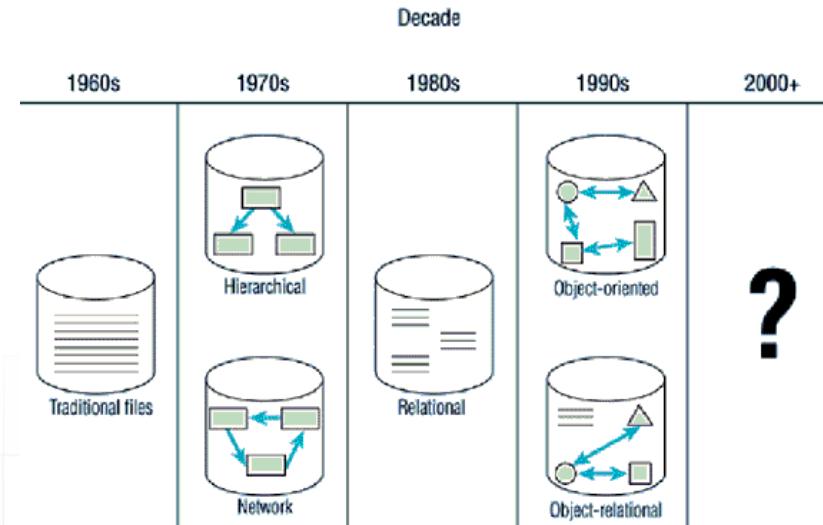


NỘI DUNG

- Đặt vấn đề
- Mục tiêu môn học
- Nội dung môn học
- Hình thức đánh giá
- Tài liệu tham khảo
- Trao đổi & thảo luận

ĐẶT VĂN ĐỀ

- Ứng dụng có sử dụng CSDL rất phổ biến hiện nay, bao phủ hầu hết trong các hoạt động kinh tế, xã hội, giáo dục, y tế → Tầm quan trọng của một công cụ trị CSDL
- Lịch sử phát triển của mô hình CSDL cũng qua nhiều giai đoạn:



- Tương ứng với sự phát triển của mô hình lưu trữ dữ liệu là sự phát triển của phần mềm cài đặt mô hình dữ liệu đó (HQTCSDL)



ĐẶT VĂN ĐỀ



Môn học: HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU



Các phần mềm
này hoạt động
như thế nào ? Tại
sao ? Có những
thành phần nào ?



MỤC TIÊU MÔN HỌC

LÝ
THUYẾT

Cung cấp cho sinh viên kiến thức nền tảng về các Hệ quản trị Cơ sở dữ liệu (HQTCSQL): Các thành phần của một HQTCSQL và các chức năng của chúng, các cơ chế **quản lý truy xuất đồng thời, an toàn** và **an ninh dữ liệu, tối ưu hóa câu hỏi**, các cấu trúc **tổ chức lưu trữ** bên trong.

THỰC
HÀNH

Tìm hiểu và vận dụng các kỹ thuật **quản lý truy xuất đồng thời** của một HQTCSQL cụ thể: **MS SQL Server**



NỘI DUNG MÔN HỌC

■ **Chương 1:** Tổng quan về HQTCSDL

- Yêu cầu về dữ liệu trong CSDL
- Khái niệm HQTCSDL
- Kiến trúc của HQTCSDL
- Phân loại HQTCSDL

■ **Chương 2:** Giao tác và lịch giao tác

- Giao tác
- Lịch giao tác
 - Lịch tuần tự
 - Lịch Khả tuần tự



NỘI DUNG MÔN HỌC

■ **Chương 3: Điều khiển truy xuất đồng thời**

- Các vấn đề của truy xuất đồng thời
- Kỹ thuật khoá
- Kỹ thuật nhãn thời gian
- Kỹ thuật lạc quan
- Một số vấn đề khác

■ **Chương 4: An toàn và an ninh dữ liệu**

- An toàn dữ liệu
- An ninh dữ liệu

■ **Chương 5: Xử lý câu truy vấn**

- Quy trình xử lý
- Phân tích cú pháp ngữ nghĩa



NỘI DUNG MÔN HỌC

- Chuyển về dạng biểu diễn trong
 - Tối ưu hóa câu hỏi
 - Ước lượng kích thước cây truy vấn
 - Phát sinh và thực thi mã lệnh
- **Chương 6:** Tổ chức dữ liệu
 - Mẫu tin
 - Tổ chức lưu trữ mẫu tin
 - **Chương 7:** Các hệ CSDL phân tán
 - Kiến trúc Client Server
 - Kiến trúc phân tán
 - Thiết kế CSDL phân tán
 - Các khái niệm cơ bản
 - Các vấn đề của hệ phân tán



HÌNH THỨC ĐÁNH GIÁ

■ LÝ THUYẾT

- Thi viết / trắc nghiệm (Không sử dụng tài liệu): $4 \rightarrow 5$ đ
- Bài tập / Kiểm tra: $2 \rightarrow 3$ đ
- Điểm tối đa: **7đ**

■ THỰC HÀNH

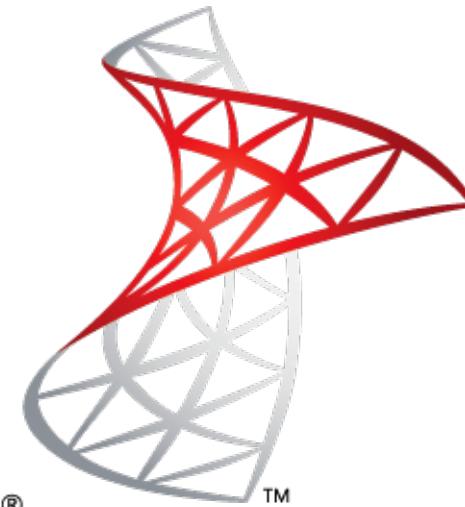
- Làm bài tập & bài thực hành
- Điểm tối đa: **3đ**

■ QUY ĐỊNH:

- Những bài thi giống nhau sẽ bị **0 ĐIỂM MÔN HỌC** (không quan tâm đến ai chép bài của ai)



PHẦN MỀM

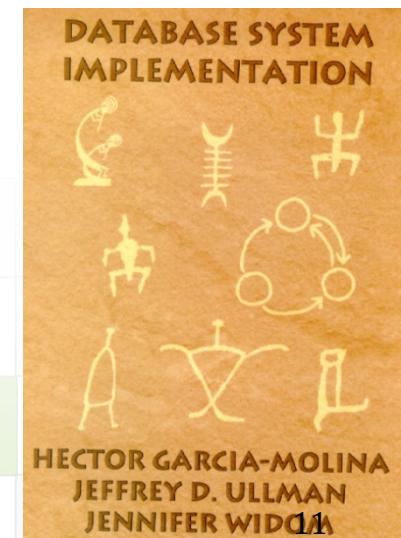
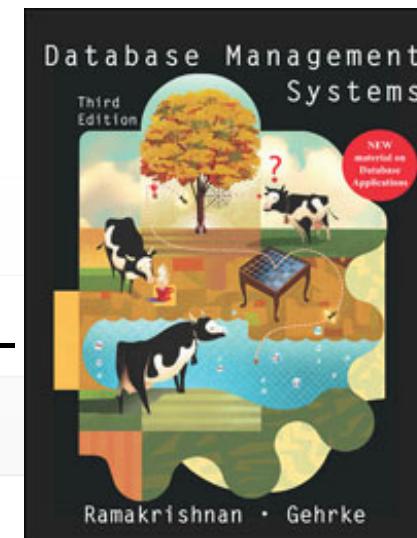
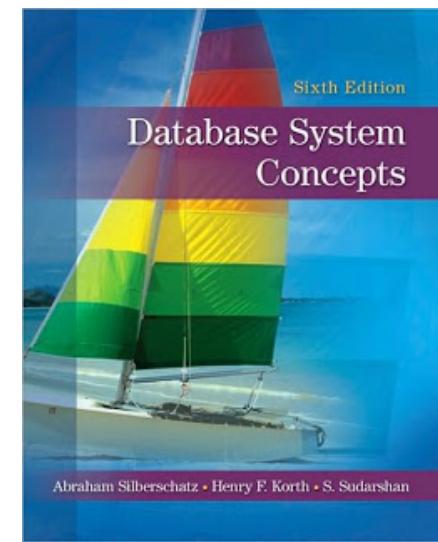
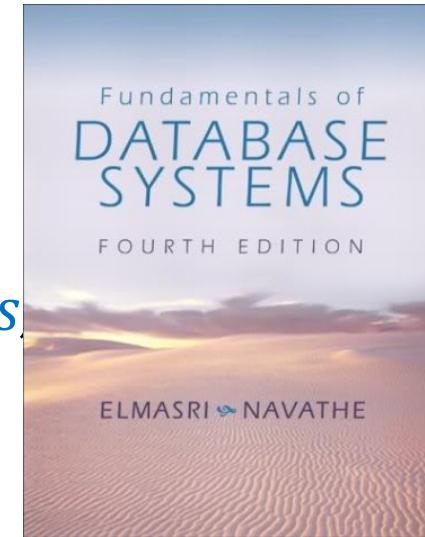


Microsoft®
SQL Server®



TÀI LIỆU THAM KHẢO

- *Fundamentals of Database Systems*, 4th Edition, Elmasri Navathe
- *Database Management Systems*, 3rd Edition, Raghu Ramakrishnan and Johannes Gehrke
- *Database System Concepts*, 4th Edition, Silberschatz–Korth –Sudarshan
- *Database Systems Implementation*, Hector Garcia-Molina, D. Ullman, Jennifer D. Widom



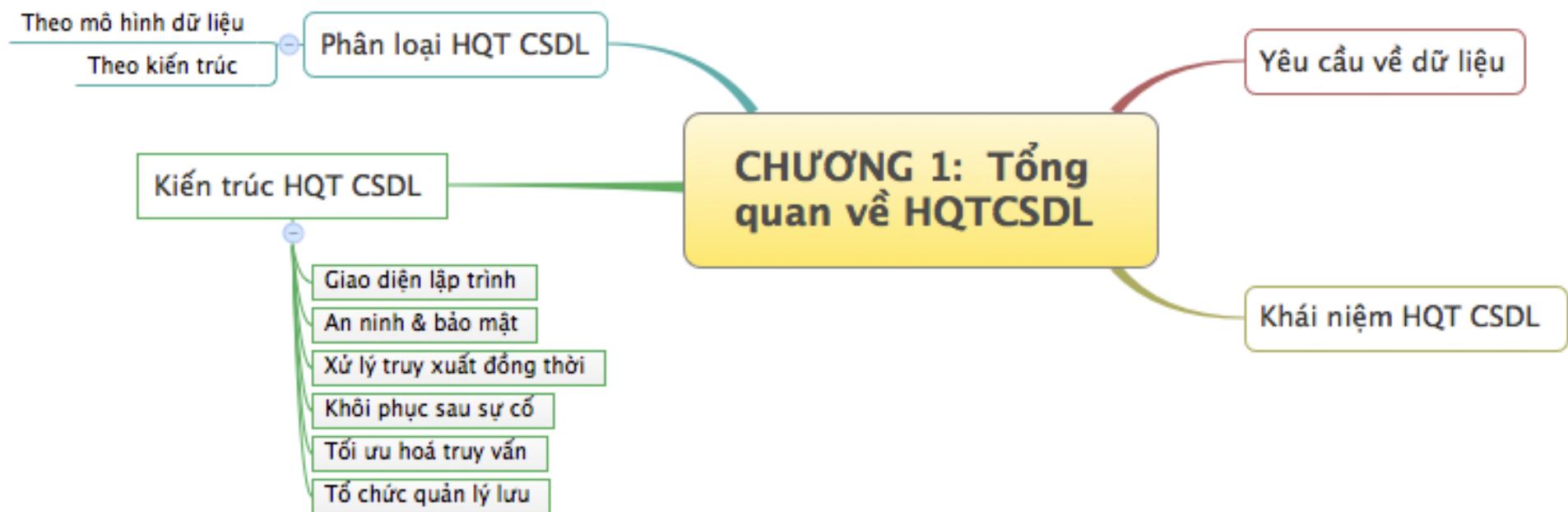


HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

Chương 1:
**TỔNG QUAN VỀ
HQT CSDL**

GVLT: *Nguyễn Trường Sơn*

Nội dung





Nội dung

- Yêu cầu về dữ liệu trong CSDL
- Khái niệm HQT CSDL
- Kiến trúc của một HQT CSDL
- Phân loại HQT CSDL



Yêu cầu về dữ liệu trong CSDL

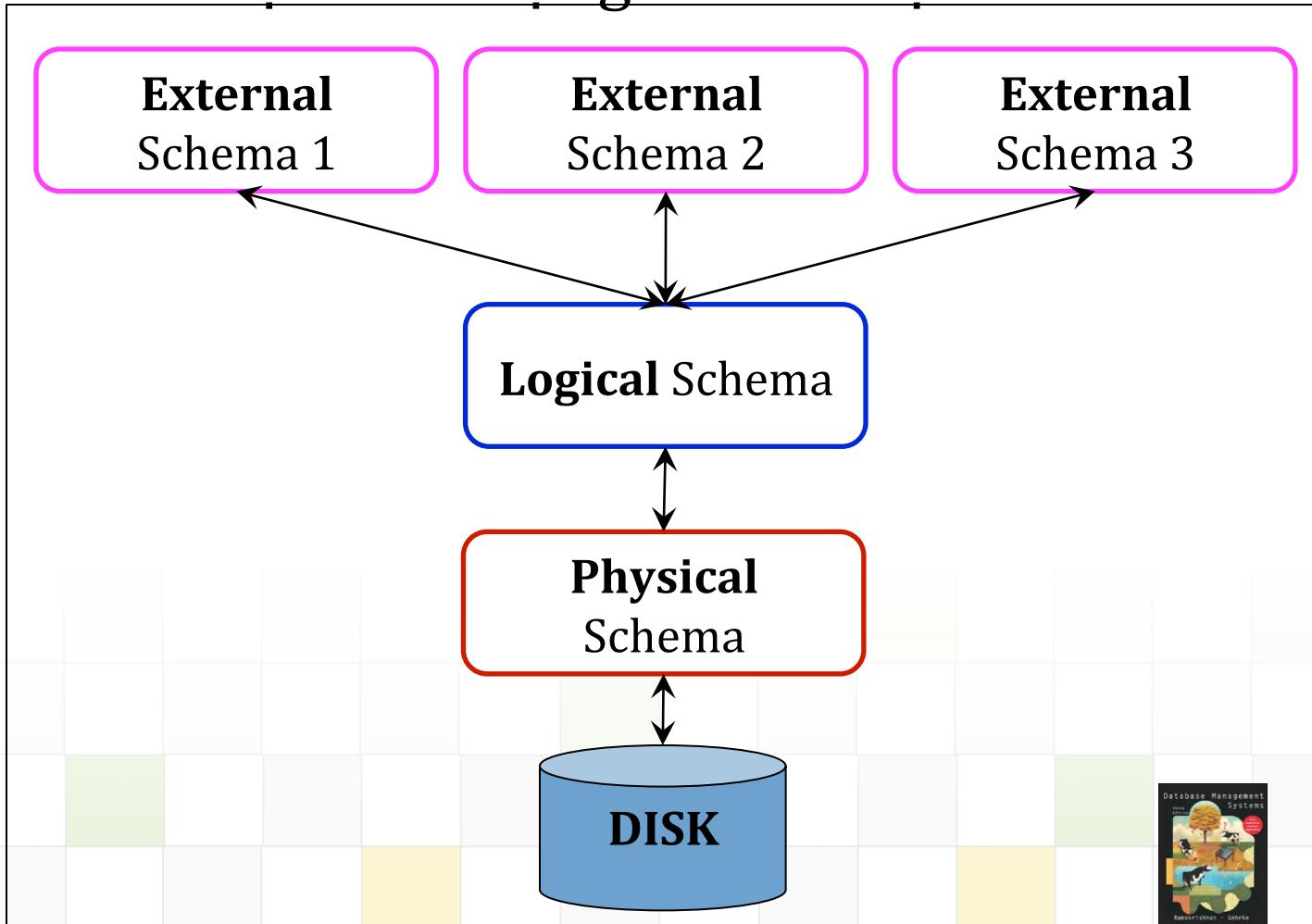
- Dữ liệu trong CSDL phải được thể hiện ở các mức độ trừu tượng khác nhau (3 mức độ):
 - **Mức ngoài** (External level)
 - Mô tả một phần của CSDL mà một đối tượng / một nhóm người dùng được quyền tiếp cận
 - **Mức luận lý** (Logic level)
 - Mô tả những thông tin gì được lưu trữ trong CSDL và những mối quan hệ giữa những thông tin đó
 - **Mức vật lý** (Physical level)
 - Dữ liệu được lưu trữ như thế nào trên thiết bị lưu trữ.

→ Làm tăng tính **độc lập** (*data independence*) của cách thức lưu trữ dữ liệu, thiết kế dữ liệu và chương trình sử dụng dữ liệu.



Yêu cầu về dữ liệu trong CSDL

- Các mức độ trùu tượng của dữ liệu:



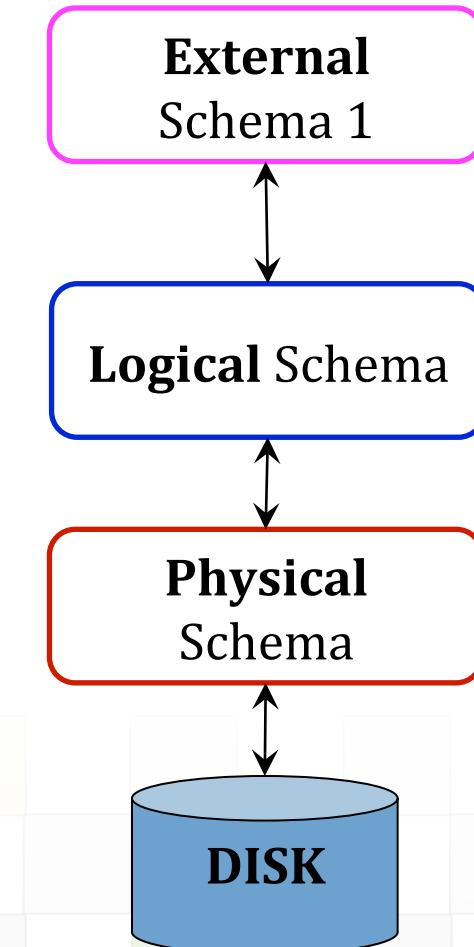


Yêu cầu về dữ liệu trong CSDL

- Dữ liệu trong CSDL cần có các đặc trưng:
 - Ít hoặc không trùng lắp dữ liệu
 - Chia sẻ cho nhiều người dùng mà không gây ra xung đột
 - An ninh, bảo mật
 - Khôi phục khi có sự cố
 - Độc lập dữ liệu
 - Độc lập luân lý: Khả năng thay đổi lược đồ mức luận lý mà không làm ảnh hưởng đến lược đồ ngoài cũng như chương trình ứng dụng.
 - Độc lập vật lý: Khả năng thay đổi tổ chức vật lý của CSDL mà không làm ảnh hưởng đến lược đồ luận lý.
- Vì vậy cần có một hệ thống quản lý hiệu quả dữ liệu trong CSDL.



Lợi ích của tính độc lập dữ liệu



- Độc lập luận lý:
 - Cho phép thêm bớt thuộc tính, bảng, các mối quan hệ mà không cần phải viết lại chương trình, ...

- Độc lập vật lý:
 - Cho phép thay đổi thiết bị lưu trữ, cách thức lưu trữ, các cấu trúc dữ liệu, các tổ chức tập tin khác nhau, các kiểu tổ chức chỉ mục khác nhau, ...



Khái niệm HQT CSDL

- Là một **hệ thống phần mềm** cung cấp các công cụ để xây dựng, khai thác và quản lý cơ sở dữ liệu.
 - **Xây dựng** (Sử dụng ngôn ngữ DDL): Định nghĩa cấu trúc CSDL, lưu trữ dữ liệu
 - **Khai thác** (Sử dụng ngôn ngữ DML): Truy vấn dữ liệu, Cập nhật dữ liệu
 - **Quản lý:**
 - Quản lý an toàn và bảo mật
 - Điều khiển truy xuất đồng thời.
 - Khôi phục khi có sự cố.
 - ...
- Một số HQTCSQL: MS SQL Server, Oracle, DB2, ...



Các lợi ích của HQT CSDL



- Độc lập dữ liệu
- Truy cập dữ liệu hiệu quả
- Toàn vẹn dữ liệu
- An ninh dữ liệu
- Truy xuất đồng thời
- Khôi phục sau sự cố
- Giảm thời gian phát triển ứng dụng



-
-
-

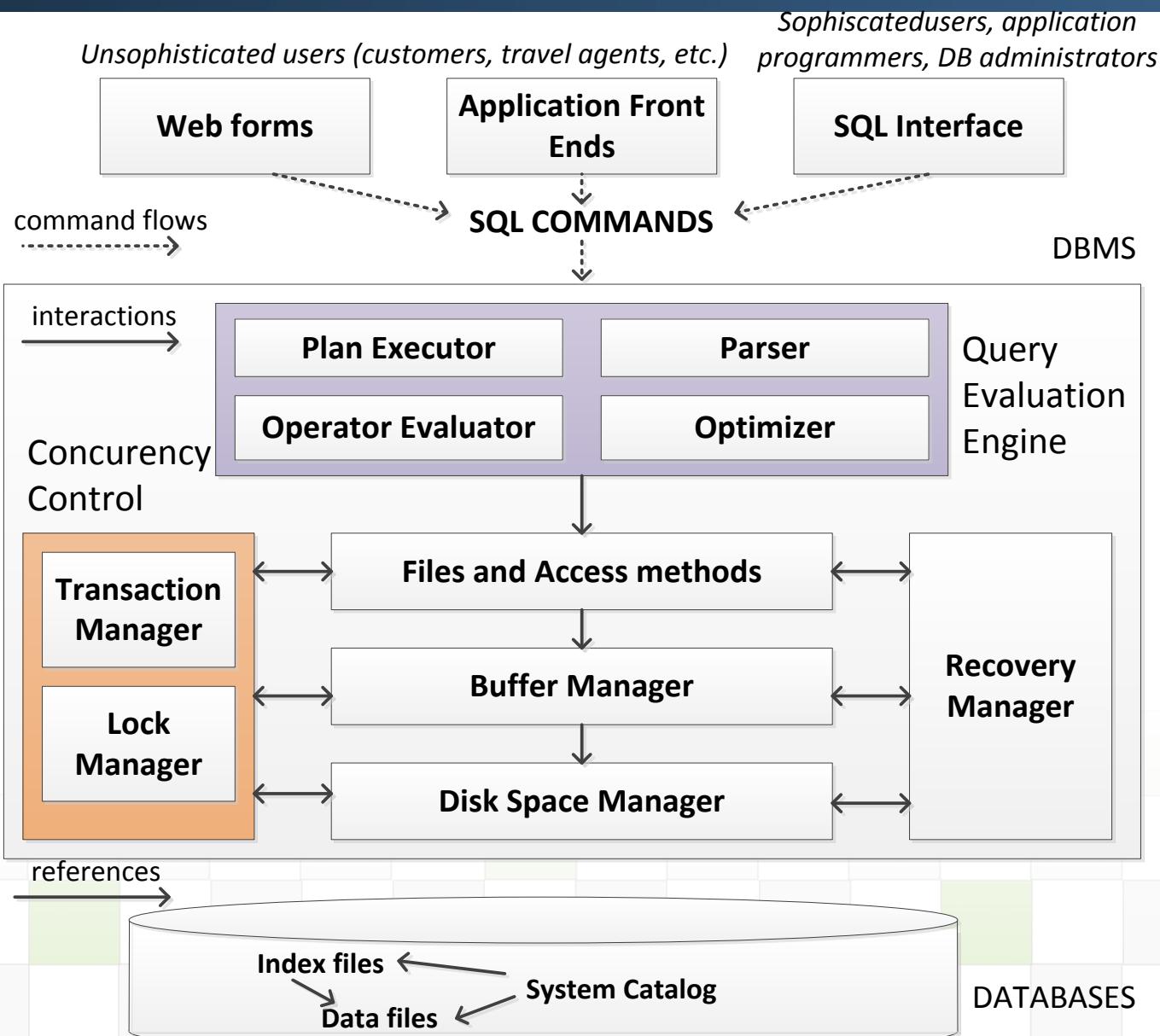


Lịch sử phát triển của các HQT CSDL

Decade of RDBMS

1960s	1970s	1980s – 1990s	2000s
Mô hình mạng CODASYL Mô hình phân cấp IMS	Mô hình quan hệ QUEL SEQUEL	Mô hình đối tượng SQL	No SQL Database
SABRE system	<i>Ingres</i> <u>PostgreSQL</u> <u>dBASE</u> Ingres Corp Sybase MS SQL Server <i>System R</i> Non-Stop SQL SQL/DS DB2 Allbase Oracle	Prototypes for ODBMS	MongoDB, Oracle NoSQL Database, Apache Cassandra , ...

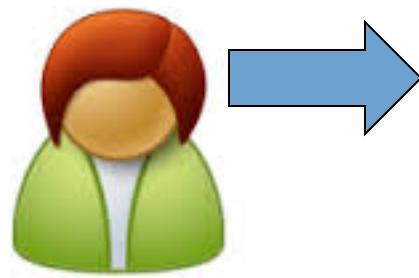
Kiến trúc của một HQT CSDL





Kiến trúc của một HQT CSDL

- Các thành phần chính:



Người sử dụng





Thành phần Giao diện lập trình

- HQTCSQL cung cấp giao diện lập trình dễ sử dụng với một ngôn ngữ lập trình CSDL:
 - Giao diện: tương tác dòng lệnh (command line), đồ họa (GUI)
 - Ngôn ngữ: SQL, T-SQL
 - VD: MS SQL Server cung cấp ngôn ngữ Transacion SQL (T-SQL)
- Các loại ngôn ngữ sử dụng trong HQTCSQL:
 - *Ngôn ngữ định nghĩa dữ liệu (DDL – Data Definition Language)*: Giúp người dùng ra lệnh cho HQTCSQL tạo ra các cấu trúc dữ liệu của CSDL (Cách tổ chức dữ liệu và mối liên hệ giữa các đối tượng dữ liệu).
 - *Ngôn ngữ thao tác CSDL (DML – Data Manupulation Language)* : Giúp người dùng tích luỹ, hiệu chỉnh và khai thác dữ liệu



Thành phần An ninh và bảo mật

- **Bảo mật dữ liệu:** HQTCSDL hỗ trợ các tính năng về chứng thực, phân quyền giúp kiểm soát tốt những người dùng hợp pháp của hệ thống..
- **An ninh dữ liệu:** HQTCSDL hỗ trợ các phương pháp mã hóa dữ liệu để ngăn chặn các tấn công của những đối tượng tin tặc (đánh cắp thông tin trên đường truyền, đánh cắp nội dung CSDL).



Thành phần Khôi phục sau sự cố

- Mục tiêu: Đảm bảo sự tổn thất, sai sót về mặt dữ liệu là ít nhất có thể.
- Cách tiếp cận: Để đảm bảo tính bền vững của CSDL, mọi thay đổi lên CSDL phải được ghi nhận lại trong nhật ký (Log)
- Các thành phần hỗ trợ quá trình khôi phục sau sự cố:
 - *Bộ phận quản lý nhật ký (Log manager)* : đảm bảo ghi nhận đầy đủ và chính xác mọi thay đổi trên CSDL vào nhật ký.
 - *Bộ phận quản lý khôi phục sự cố (Recovery Manager)*: dựa vào nhật ký để phục hồi lại CSDL về trạng thái nhất quán trước đó (Trạng thái thoả tất cả RBTV của CSDL)

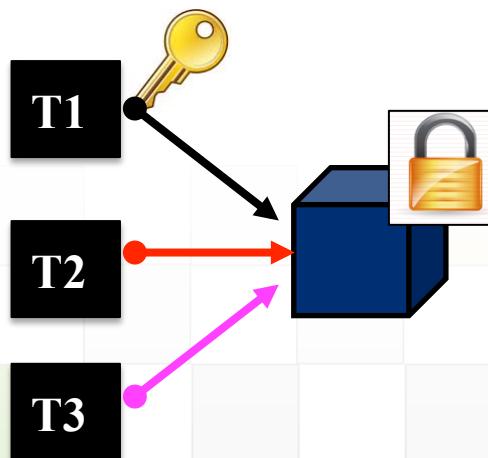
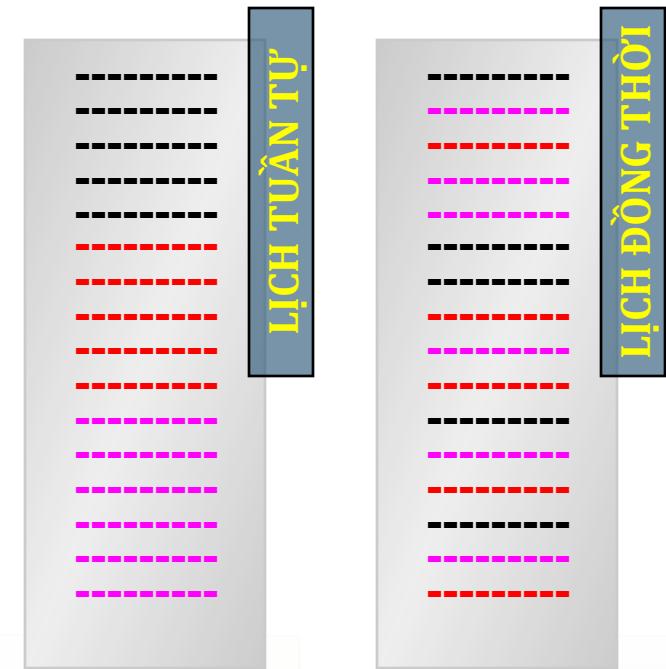
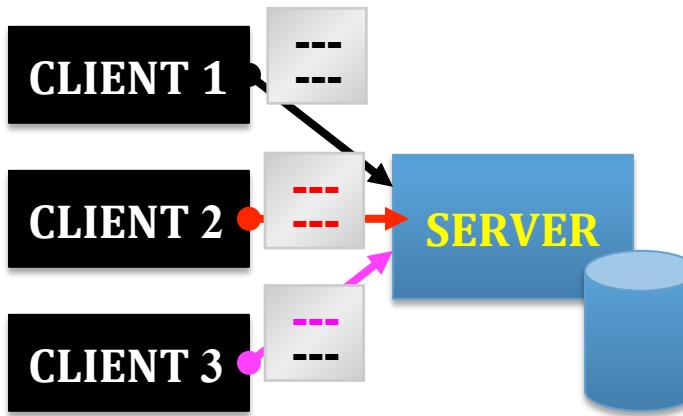


Xử lý truy xuất đồng thời

- Mục tiêu:
 - Đảm bảo các xử lý có thể được thực hiện đồng thời mà làm không làm cho dữ liệu bị mất tính nhất quán (vi phạm các ràng buộc toàn vẹn)
- Các thành phần con: Bộ phận quản lý giao tác (Transaction Manager & Locking Manager)
- Phương pháp:
 - Sử dụng khái niệm giao tác (**transaction**) để biểu diễn một đơn vị xử lý, một giao tác bao gồm các hành động mà được thực hiện tồn bộ hoặc không có hành động nào được thực hiện.
 - Bộ lập lịch (**scheduler**) có nhiệm vụ lập 1 lịch thực hiện từ n giao tác không tách biệt về thời gian sao cho kết quả không vi phạm tính nhất quán của CSDL.
 - Sử dụng cơ chế khóa (**lock**) để khóa các đơn vị dữ liệu nào đó khi cần → ngăn 2 giao tác cùng thao tác lên 1 đơn vị dữ liệu ấy tại cùng 1 điểm → Hỗ trợ để lập lịch.



Điều khiển đồng thời (tt)





Điều khiển đồng thời (tt)

■ Vấn đề deadlock

- Do sử dụng cơ chế khóa nên các giao tác sẽ phải chờ khi cần truy xuất 1 đơn vị dữ liệu đang bị khóa.
- Tình huống chờ vĩnh viễn mà vẫn không được truy xuất đơn vị dữ liệu bị khóa gọi là **Deadlock (khoá chết)**
 - Các giao tác chờ đợi lẫn nhau để được cấp phát tài nguyên và không giao tác nào có thể hoàn tất.
- Thành phần quản lý giao tác sẽ phải can thiệp vào:
 - Hoặc hủy bỏ một trong các giao tác gây deadlock
 - Hoặc ngăn chặn từ trước để không bao giờ xảy ra deadlock



Xử lý truy vấn

- Biểu diễn câu truy vấn ở dạng ngôn ngữ cấp cao (SQL) và thực hiện câu truy vấn có hiệu quả.
- Query compiler – biên dịch truy vấn

Query parser

- Xây dựng cấu trúc phân tích câu truy vấn dưới dạng cây

Query
preprocessor

- Kiểm tra ngữ nghĩa của câu truy vấn
- Chuyển đổi cấu trúc cây sang ngôn ngữ đại số quan hệ

Query
optimizer

- Sắp xếp các phép toán nhằm mục đích tối ưu hóa câu truy vấn



Quản lý lưu trữ

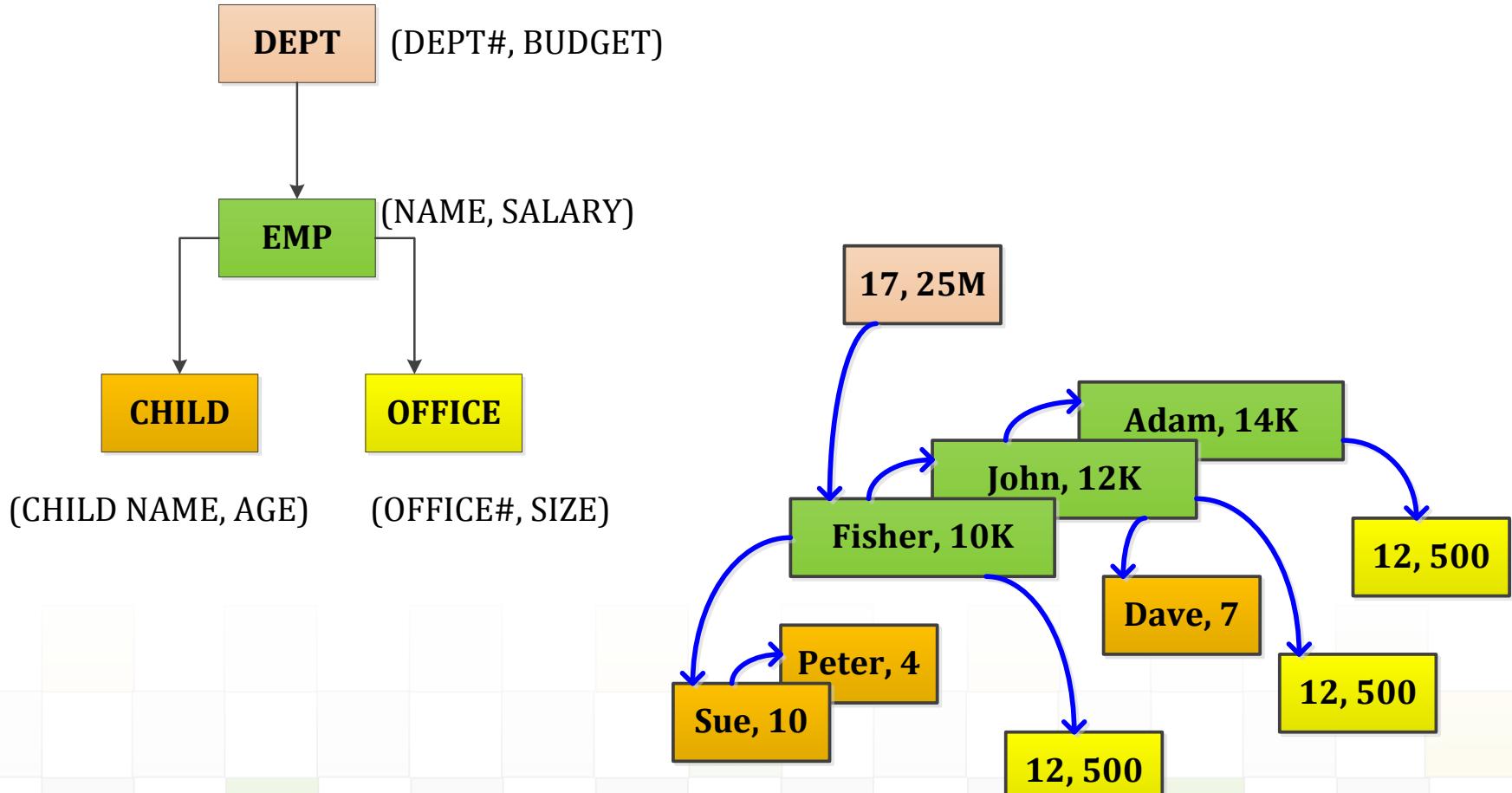
- Thành phần có nhiệm vụ điều khiển việc đọc/ghi dữ liệu qua lại giữa bộ nhớ và thiết bị lưu trữ
- Làm việc với các khái niệm:
 - Tập tin dữ liệu
 - Từ điển dữ liệu
 - Lưu trữ các metadata (Siêu dữ liệu) về cấu trúc của CSDL, đặc biệt là lược đồ của CSDL
 - Chỉ mục
 - Giúp cho việc tìm kiếm Dữ liệu được nhanh chóng



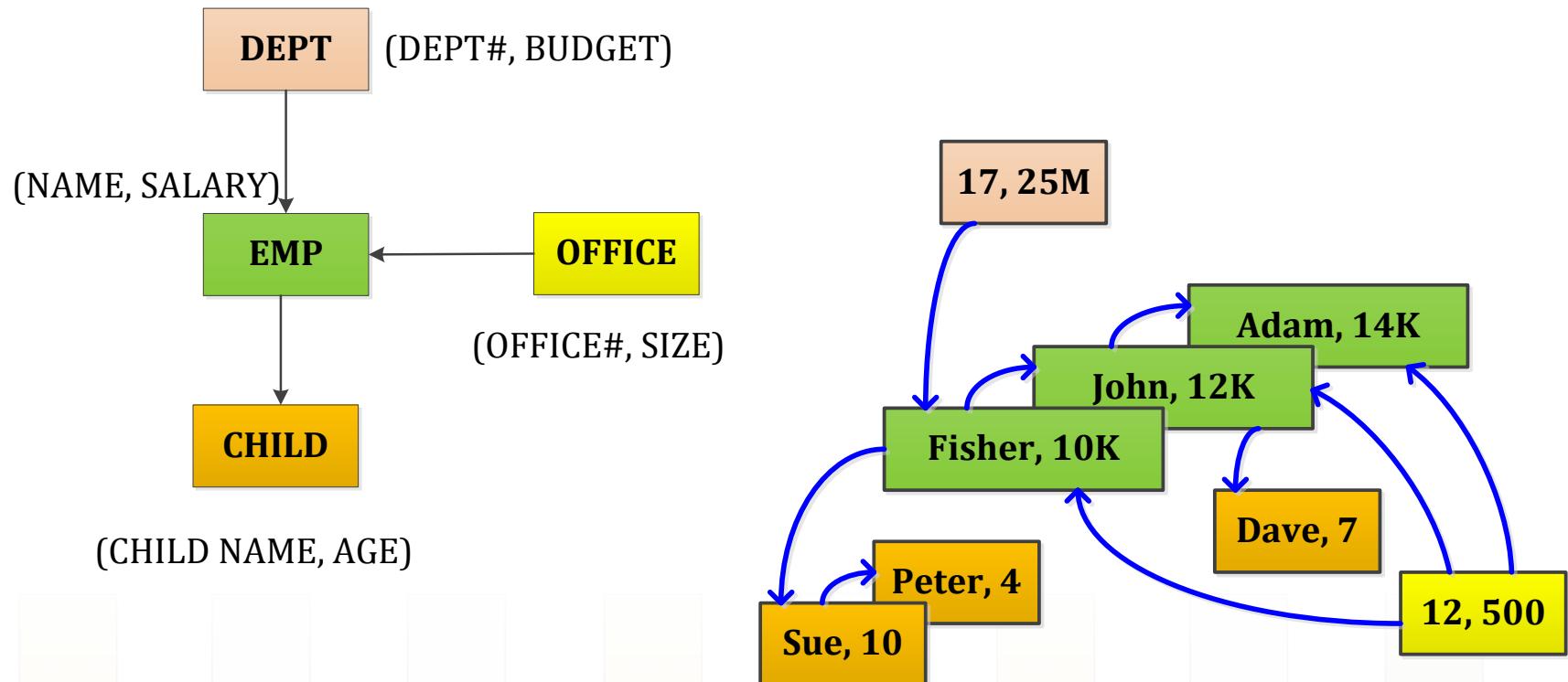
Phân loại HQT CSDL

- Theo mô hình dữ liệu:
 - Phân cấp
 - Mạng
 - Quan hệ
 - Đối tượng
- Theo kiến trúc tính toán:
 - Tập trung (Centralized database system)
 - Khách / chủ (Client server database system)
 - Phân tán (Distributed database system)
- Theo đặc tính:
 - HQTCSDL thời gian thực (real-time database system)
 - HQTCSDL chịu lỗi cao (high fault tolerance database system)
 - HQTCSDL đa phương tiện (multi-media database system)

Mô hình phân cấp



Mô hình mạng





Mô hình quan hệ

DEPT (DEPT #, BUDGET)

OFFICE (OFFICE #, SIZE)

EMP (NAME, SALARY)

CHILD (CHILD NAME, AGE)

WORKS (DEPT #, NAME)

OFFSPRING (NAME, CHILD NAME)

OCCUPIED (NAME, OFFICE #)

OCCUPIED		
Fisher	12	
John	12	
Adam	12	

DEPT		
17		25M
OFFICE		
12		500

CHILD		
Sue	10	
Peter	4	
Dave	7	

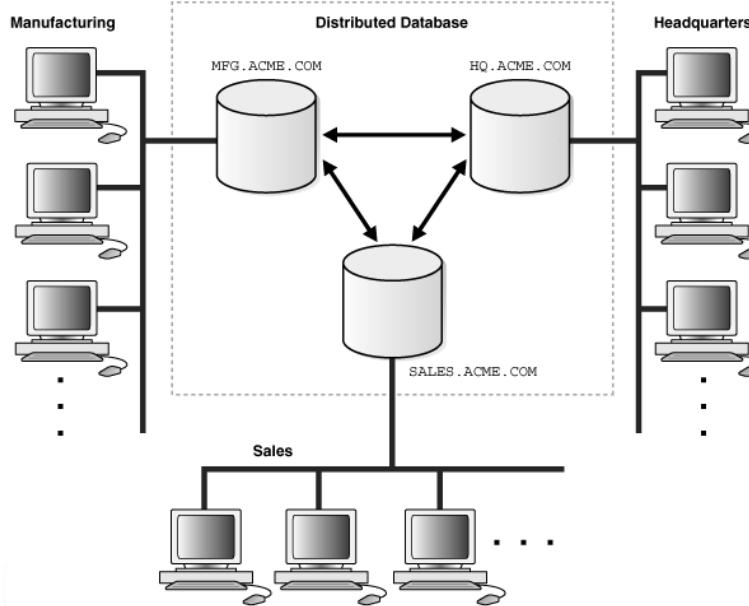
EMP		
Fisher	10K	
John	12K	
Adam	14K	

OFFSPRING		
Fisher	Sue	
Fisher	Peter	
Jone	Dave	

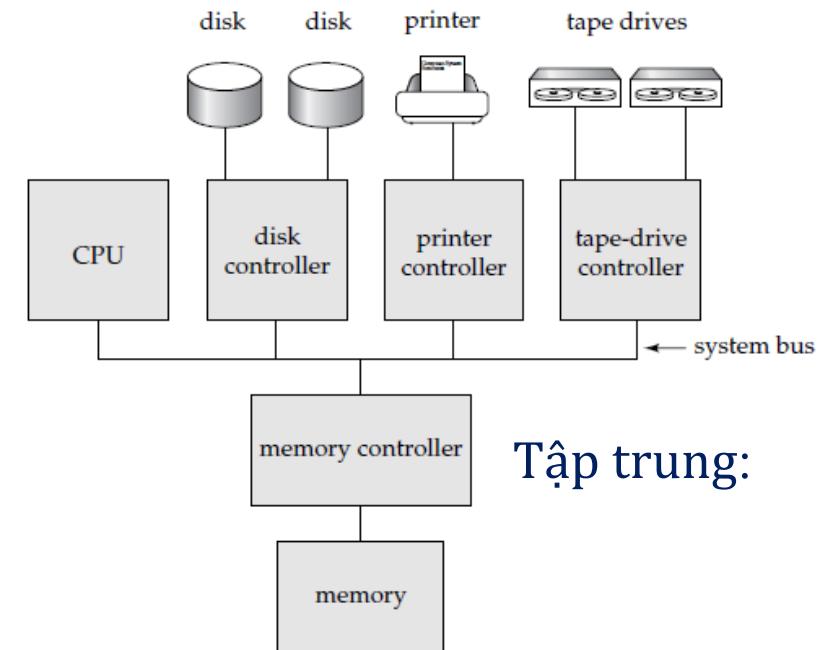
WORKS		
17		Fisher
17		John
17		Adam

Phân loại HQTCSDL

- Theo kiến trúc tính toán:



Phân tán



Tập trung:

Figure 18.1 A centralized computer system.

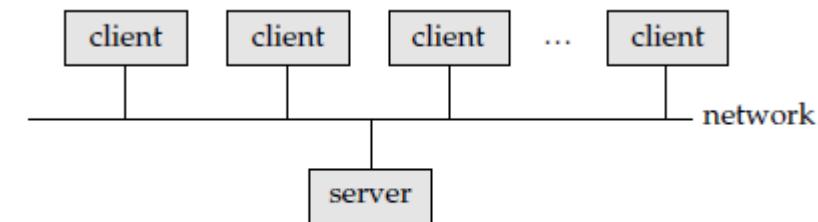


Figure 18.2 General structure of a client–server system.

Khách / chủ

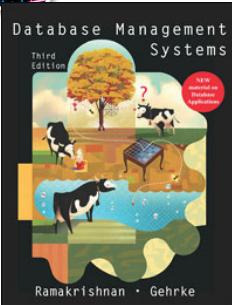


TÓM TẮT CHƯƠNG 1

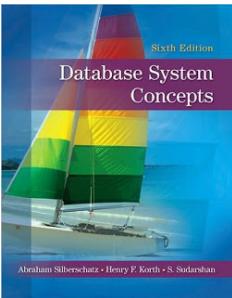
- Sự cần thiết phải có HQTCSDL
 - Dữ liệu cần được trình bày ở nhiều mức khác nhau
 - Các đặc trưng cần phải có của dữ liệu khi lưu trữ trong CSDL
 - Tính chất độc lập dữ liệu
- Một số thành phần chính của HQTCSDL
 - Giao diện lập trình
 - Xử lý đồng thời
 - An ninh và bảo mật
 - Khôi phục sau sự cố
 - Xử lý truy vấn
 - Quản lý lưu trữ
- Lịch sử phát triển của HQTCSDL
- Kiến trúc tổng quan của HQTCSDL
- Phân loại HQTCSL
 - Theo mô hình dữ liệu
 - Theo kiến trúc tính toán
 - Theo đặc tính



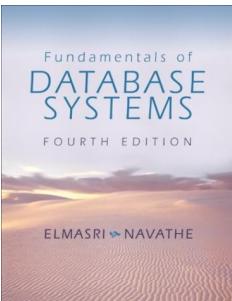
ĐỌC THÊM



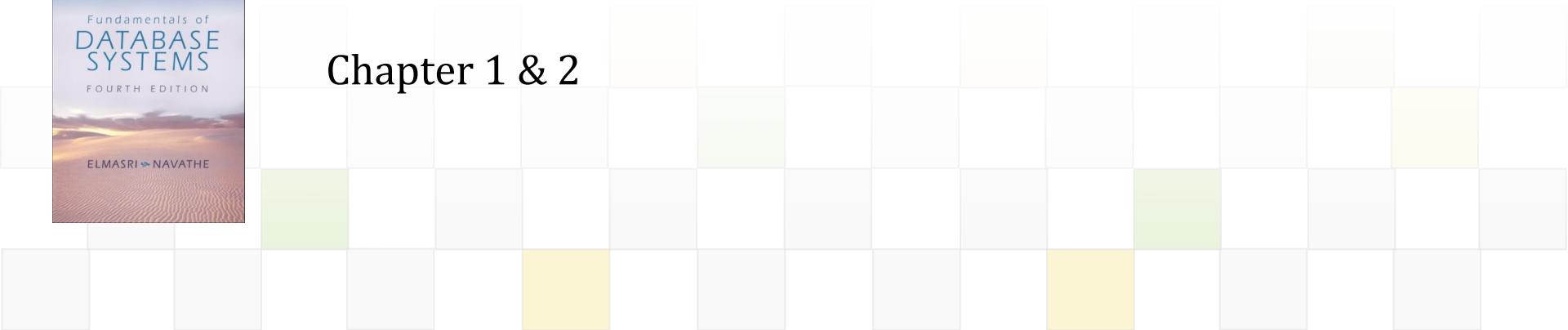
Chapter 1. Introduction to database systems (p.3 → p.23)



Chapter 1. Introduction (p.1 → p.24)



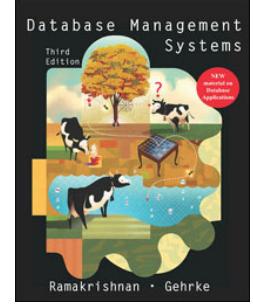
Chapter 1 & 2





BÀI TẬP

Đọc sách *Database Management Systems, 2nd Editon*
(Có thể tham khảo các sách khác & google) và làm
những nội dung sau:

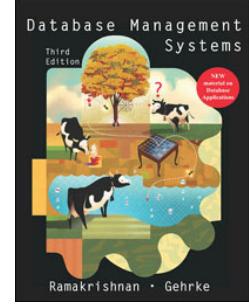


- A. Trình bày lại nội dung phần 1.10 trong sách
- B. Trả lời các câu hỏi trong phần bài tập Exercises 1.1 đến 1.8 (giải thích ngắn gọn, đầy đủ & súc tích):



BÀI TẬP

Đọc sách *Database Management Systems, 2nd Editon*
(Có thể tham khảo các sách khác & google) và làm
những nội dung sau:



- A. Trình bày lại nội dung phần 1.10 trong sách
- B. Trả lời các câu hỏi trong phần bài tập Exercises 1.1 đến 1.8 (giải thích ngắn gọn, đầy đủ & súc tích):

Chương 2:
**GIAO TÁC VÀ LỊCH
GIAO TÁC**

GVLT: Nguyễn Trường Sơn



Nội dung trình bày

- **Giới thiệu**
- Giao tác
 - Khái niệm
 - Tính ACID của giao tác
 - Các thao tác của giao tác
 - Các trạng thái của giao tác
- Lịch thao tác
 - Giới thiệu
 - Khái niệm
 - Lịch tuần tự
 - Lịch khả tuần tự



Giới thiệu

- Hai yêu cầu cơ bản của ứng dụng khai thác CSDL trong thực tế:
 - Cho phép nhiều người dùng *đồng thời* khai thác CSDL nhưng phải giải quyết được các *tranh chấp*.
 - Sự cố kỹ thuật có thể luôn luôn xảy ra nhưng phải giải quyết được vấn đề về *nhất quán dữ liệu*.
- Một số ví dụ về ứng dụng có sử dụng CSDL :
 - Hệ thống giao dịch ở ngân hàng
 - Hệ thống đặt vé máy bay
 - Hệ thống quản lý học sinh
 - ...



Giới thiệu – Một số tình huống

■ *Hệ thống đặt vé máy bay:*

- “Khi hành khách mua vé”
- “Khi hai hành khách cùng đặt một ghế trống”
- ...

GHẾ (Mã ghế, Mã CB, Trạng thái)

CHUYẾN BAY (Mã CB, Ngày giờ, Số ghế còn)

■ *Hệ thống ngân hàng:*

- “Khi chuyển tiền từ tài khoản A sang tài khoản B”
- “Khi rút tiền của một tài khoản”
- “Nhiều người cùng rút tiền trên một tài khoản”
- ...

TÀI KHOẢN (Mã TK, Số dư)

GIAO DỊCH (Mã GD, Loại, Số tiền)

■ *Hệ thống quản lý học sinh:*

- Thêm một học sinh mới
- Chuyển lớp
- ...

Lớp học (Mã lớp, Tên, Sĩ số)

Học sinh (Mã HS, Họ tên, Mã lớp)



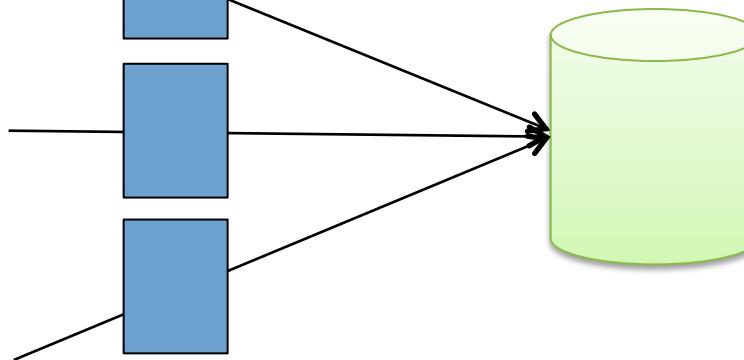
Giới thiệu – Một số tình huống

- “Hai (nhiều) hành khách cùng đặt một ghế trống”

– Lỗi: Có thể có nhiều hành khách đều đặt được dù chỉ còn 1 ghế



1. Tìm thấy một ghế trống
2. Đặt ghế



GHẾ (Mã ghế, Mã CB, Trạng thái)

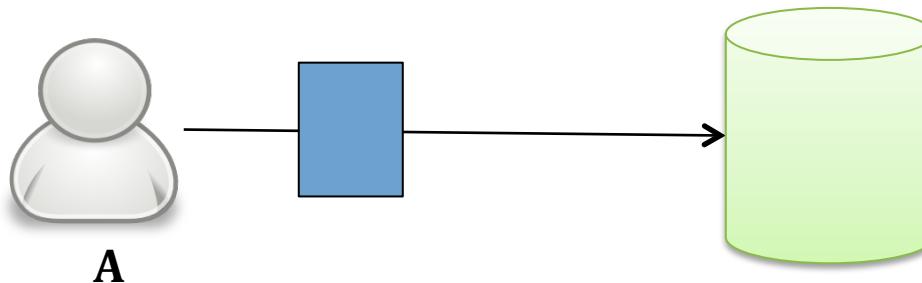
Mã ghế	Mã CB	Trạng thái
1001	100	No
1002	100	No
1003	100	Yes
...	...	No
1050	100	No

→ Phải giải quyết được tranh chấp để đảm



Giới thiệu – Một số tình huống

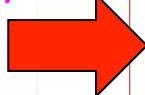
- “Chuyển tiền từ tài khoản A sang tài khoản B”
 - Lỗi: Có thể đã rút tiền từ A nhưng chưa cập nhật số dư của B



A

1. update TAIKHOAN set SoDu=SoDu-50 where MATK=A
2. update TAIKHOAN set SoDu=SoDu+50 where MATK=B

Sự cố



TAI KHOAN(Mã TK, Số dư)

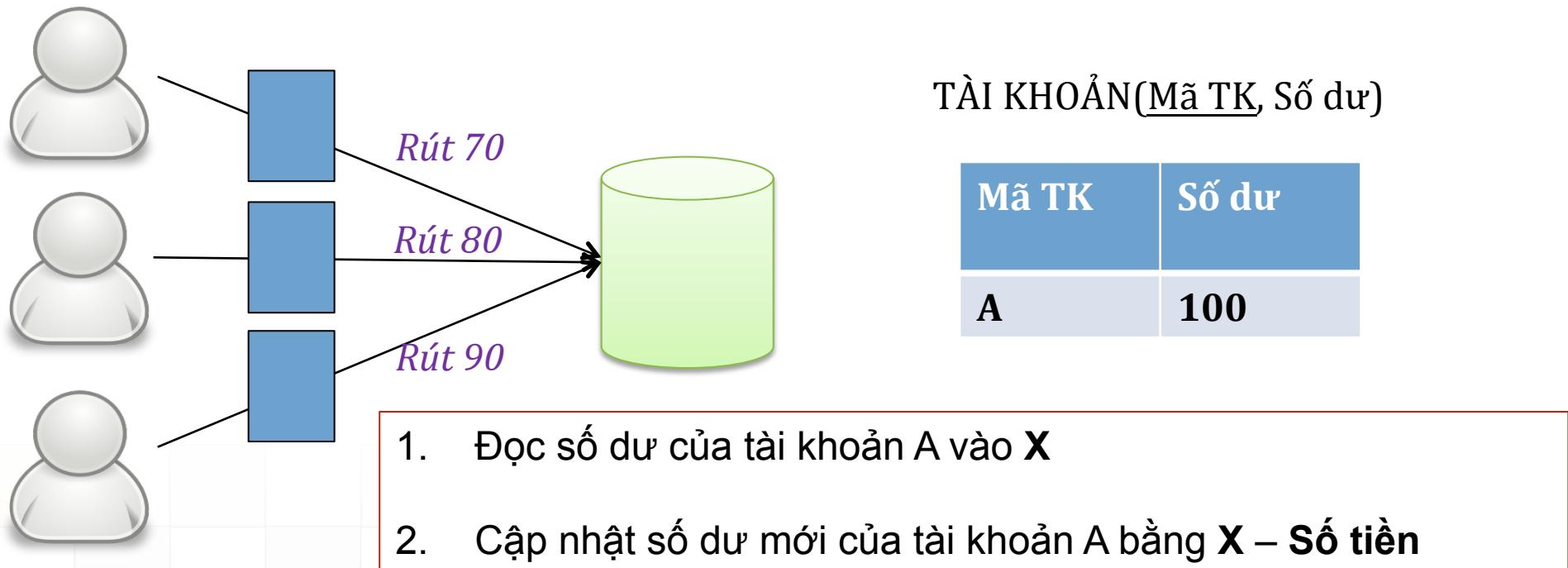
Mã TK	Số dư
A	50
B	100
C	60
...	...
N	90

→ Phải đảm bảo được nhất quán dữ liệu khi có sự cố.



Giới thiệu – Một số tình huống

- “Nhiều người cùng rút tiền từ một tài khoản”
 - Lỗi: Có thể rút nhiều hơn số tiền thực có



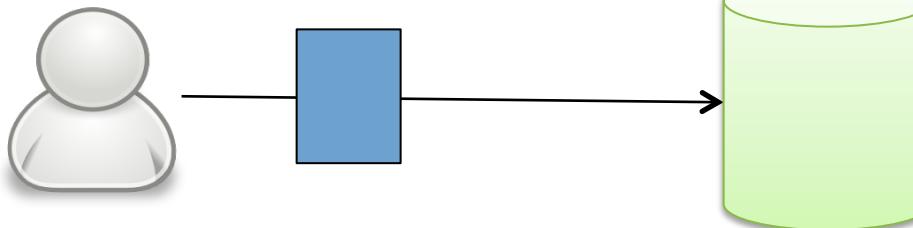
→ Phải giải quyết được tranh chấp để đảm bảo được nhất quán dữ liệu.



Giới thiệu – Một số tình huống

■ “Thêm một học sinh mới”

- Lỗi: Có thể xảy ra trường hợp học sinh đã được thêm nhưng số lượng không được cập nhật.



- Sự cố*
1. Thêm vào một học sinh của lớp
2. Cập nhật số lượng tăng lên 1
- Phải đảm bảo được nhất quán dữ liệu khi có

Lớp học (Mã lớp, Tên, Số lượng)

Mã lớp	Tên	Số lượng
1	10A	3

Học sinh (Mã HS, Họ tên, Mã lớp)

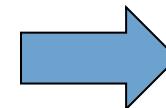
Mã HS	Họ tên	Mã lớp
1	An	1
2	Thảo	1
3	Bình	1



Giới thiệu

- Nhận xét:
 - Thường xuyên xảy ra vấn đề nhất quán dữ liệu nếu một xử lý gấp sự cố hoặc khi các xử lý được gọi truy xuất đồng thời.

- Cần 1 khái niệm biểu diễn một đơn vị xử lý với các tính chất:
 - Nguyên tố
 - Cô lập
 - Nhất quán
 - Bền vững



Giao tác

Là một khái niệm nền tảng của điều khiển truy xuất đồng thời và khôi phục khi có sự cố.



Nội dung trình bày

- Giới thiệu
- **Giao tác**
 - Khái niệm
 - Tính ACID của giao tác
 - Các thao tác của giao tác
 - Các trạng thái của giao tác
- Lịch thao tác
 - Giới thiệu
 - Khái niệm
 - Lịch tuần tự
 - Lịch khả tuần tự



Giao tác là gì ?

- **Giao tác (Transaction)** là một đơn vị xử lý **nguyên tố** gồm một chuỗi các hành động đọc / ghi trên các *đối tượng CSDL*
 - **Nguyên tố:** Không thể phân chia được nữa. Các hành động trong một giao tác hoặc là thực hiện được tất cả hoặc là không thực hiện được bất cứ hành động nào.
- Trong kiến trúc hệ quản trị CSDL:
 - Bộ phận **Điều khiển đồng thời** đóng vai trò quản lý giao tác.

-- statement 1
-- statement 2
-- statement 3
--
-- statement n

T



Tính chất ACID của giao tác

- Tính nguyên tố (**A**tomicity)
 - Hoặc là toàn bộ hoạt động được phản ánh đúng đắn trong CSDL, hoặc không có hoạt động nào cả.
- Tính nhất quán (**C**onsistency)
 - Khi một giao tác kết thúc (thành công hay thất bại), CSDL phải ở trạng thái nhất quán (Đảm bảo mọi RBTV). Một giao tác đưa CSDL từ trạng thái nhất quán này sang trạng thái nhất quán khác.
- Cô lập (**I**solation)
 - Một giao tác khi thực hiện sẽ không bị ảnh hưởng bởi các giao tác khác thực hiện đồng thời với nó.
- Bền vững (**D**urability)
 - Mọi thay đổi trên CSDL được ghi nhận bền vững vào thiết bị lưu trữ dù có sự cố có thể xảy ra.



Ví dụ về tính chất ACID

Chuyển khoản tiền từ tài khoản A sang tài khoản B

Giao tác Chuyển khoản

1. update TAIKHOAN set SoDu=SoDu-50 where MATK=A

2. update TAIKHOAN set SoDu=SoDu+50 where MATK=B

Cuối giao tác

Sự cố

Mã TK	Số dư
A	50
B	100
C	60
...	...
N	90

Atomicity: Hoặc cả 2 bước trên đều thực hiện hoặc không bước nào được thực hiện. Nếu có sự cố bước 2 thì HQT CSDL có cơ chế khôi phục lại dữ liệu như lúc ban đầu.

Consistency: Với giao tác chuyển tiền, tổng số dư của A và B luôn luôn không đổi.



Ví dụ về tính chất ACID

Thêm học sinh mới vào một lớp

Giao tác Thêm học sinh mới

1. Thêm một học sinh vào bảng học sinh
 2. Cập nhật sĩ số của lớp tăng lên 1
- Cuối giao tác

Sự cố

Atomicity: Hoặc cả 2 bước trên đều thực hiện hoặc không bước nào được thực hiện.
Nếu có sự cố bước 2 thì HQT CSDL có cơ chế khôi phục lại dữ liệu như lúc ban đầu.

Lớp học (Mã lớp, Tên, Sĩ số)

Mã lớp	Tên	Sĩ số
1	10A	3

Học sinh (Mã HS, Họ tên, Mã lớp)

Mã HS	Họ tên	Mã lớp
1	An	1
2	Thảo	1
3	Bình	1

Consistency: Sĩ số của lớp phải luôn bằng số học sinh thực sự và không quá 3.



Ví dụ về tính chất ACID

Rút tiền (TK1, 80)

T1

Đọc số dư: t

Cập nhật số dư ($=t-80$)

Gửi tiền (TK1, 50)

T2

Đọc số dư: t

Cập nhật số dư ($=t+50$)

Tài khoản (Mã TK, Số dư)

Mã TK	Số dư
1	100
2	500
3	200

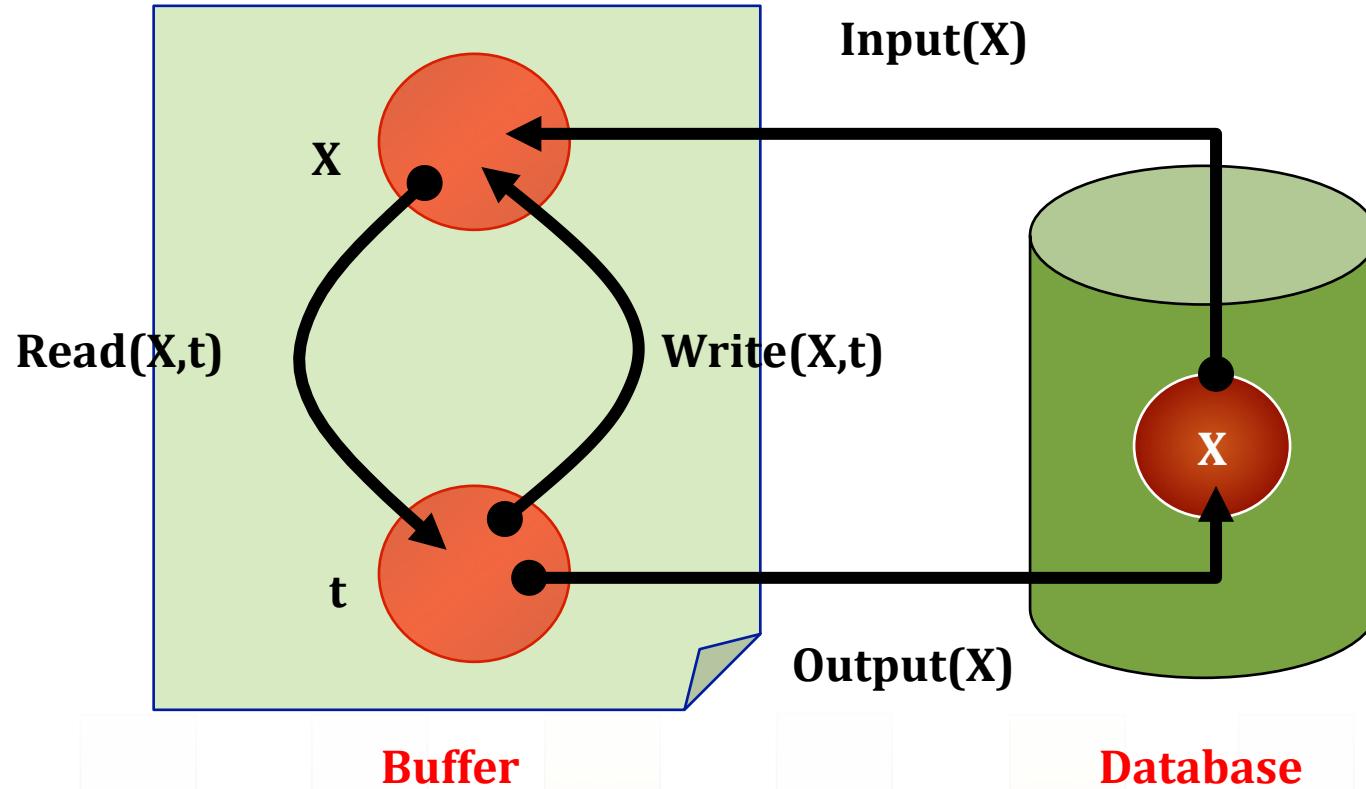
Isolation: Tính chất cô lập đảm bảo mặc dù các giao tác có thể đan xen nhau nhưng kết quả của chúng tương tự với một kết quả tuần tự nào đó → Các giao tác không bị ảnh hưởng bởi các giao tác khác khi thực thi.



Đơn vị dữ liệu

- Đối tượng CSDL mà giao tác thực hiện các xử lý đọc /ghi còn được gọi là **đơn vị dữ liệu**.
- Một đơn vị dữ liệu (element) có thể là các thành phần :
 - Quan hệ (Relations)
 - Khối dữ liệu trên đĩa (Blocks)
 - Bộ (Tuples)
- Một CSDL bao gồm nhiều đơn vị dữ liệu.

Các thao tác của giao tác



Read (A, t) : Đọc đơn vị dữ liệu A vào t

Write (A, t) : Ghi t vào đơn vị dữ liệu A



Ví dụ về biểu diễn giao tác

- Giả sử có 2 đơn vị dữ liệu A và B với ràng buộc $A = B$ (nếu có một trạng thái nào đó mà $A \neq B$ thì sẽ mất tính nhất quán)
- Giao tác T thực hiện 2 bước:
 - $A = A * 2$
 - $B = B * 2$
- Biểu diễn T:

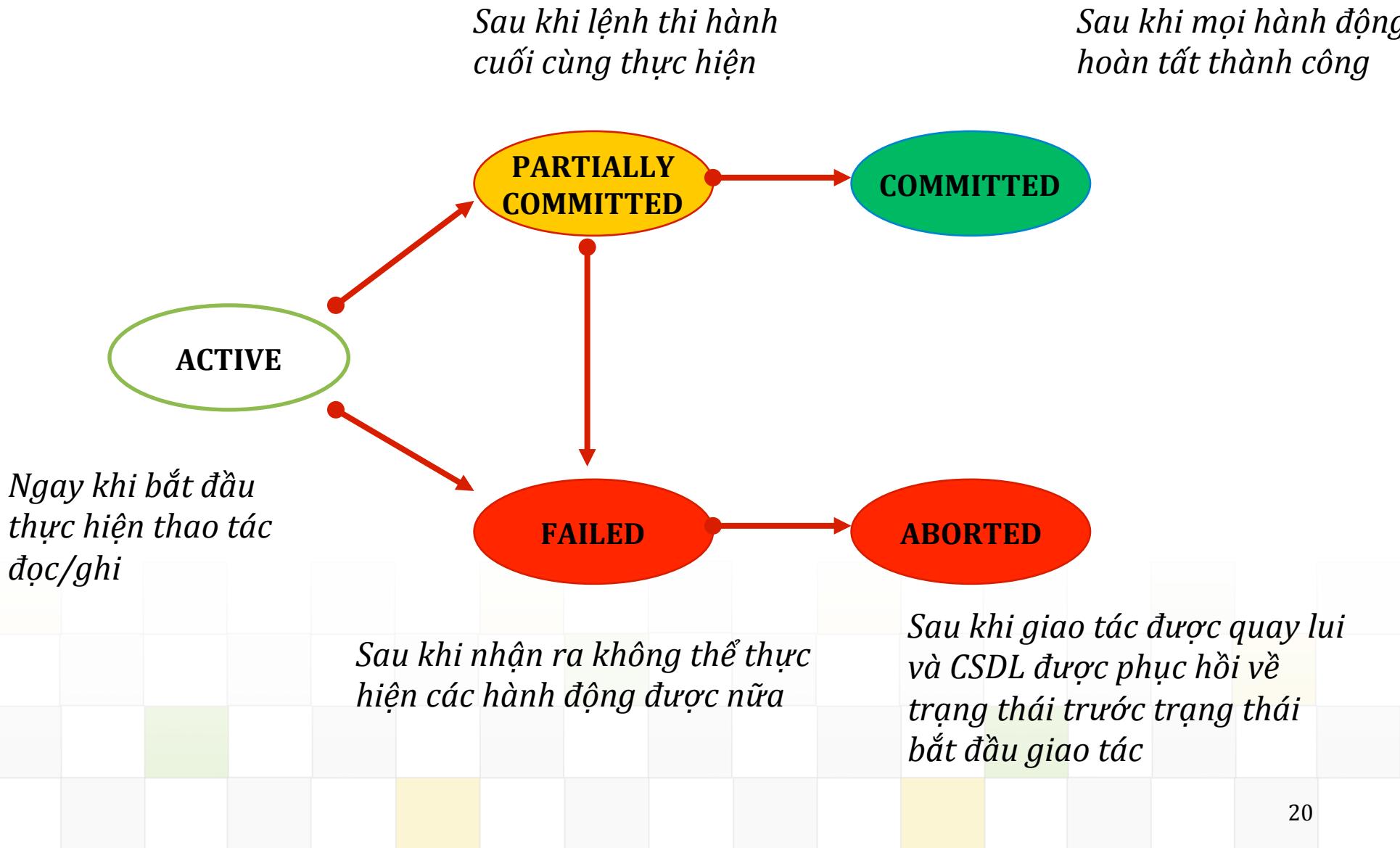
T
<i>Read(A, t); t = t*2; Write(A, t) Read(B, t); t = t*2; Write(B, t)</i>
- Hoặc:

T: Read(A, t); t = t*2; Write(A, t); Read(B, t); t = t*2; Write(B, t)

Giao tác: Ví dụ (tt)

Hành động	t	Mem A	Mem B	Disk A	Disk B
Input (A)		8		8	8
Read (A, t)	8	8		8	8
$t := t * 2$	16	8		8	8
Write (A, t)	16	16		8	8
Input (B)	16	16	8	8	8
Read (B, t)	8	16	8	8	8
$t := t * 2$	16	16	8	8	8
Write (B, t)	16	16	16	8	8
Output (A)	16	16	16	16	8
Output (B)	16	16	16	16	16

Các trạng thái của giao tác





Khai báo giao tác trong T-SQL

BEGIN TRANSACTION	Bắt đầu giao tác
COMMIT TRANSACTION	Kết thúc giao tác thành công
ROLLBACK TRANSACTION	Kết thúc giao tác không thành công. CSDL được đưa về tình trạng trước khi thực hiện giao tác.



Nội dung trình bày

- Giới thiệu
- Giao tác
 - Khái niệm
 - Tính ACID của giao tác
 - Các thao tác của giao tác
 - Các trạng thái của giao tác
- **Lịch thao tác**
 - Giới thiệu
 - Khái niệm
 - Lịch tuần tự
 - Lịch khả tuần tự



Các cách thực hiện của các giao tác

- **Thực hiện tuần tự**: Các thao tác khi thực hiện mà không giao nhau về mặt thời gian.
 - ↑ **Ưu**: Nếu thao tác đúng đắn thì luôn luôn đảm bảo nhất quán dữ liệu.
 - ↓ **Khuyết**: Không tối ưu về việc sử dụng tài nguyên và tốc độ.
- **Thực hiện đồng thời**: Các lệnh của các giao tác khác nhau xen kẽ nhau trên trực thời gian.
 - ↓ **Khuyết**: Gây ra nhiều phức tạp về nhất quán dữ liệu
 - ↑ **Ưu**:
 - **Tận dụng tài nguyên và thông lượng (throughput)**. Ví dụ: Trong khi một giao tác đang thực hiện việc đọc / ghi trên đĩa, một giao tác khác đang xử lý tính toán trên CPU.
 - **Giảm thời gian chờ**. Ví dụ: Chia sẻ chu kỳ CPU và truy cập đĩa để làm giảm sự trì hoãn trong các giao tác thực thi.



Lịch thao tác là gì ?

- Định nghĩa: Một lịch thao tác **S** được lập từ **n** giao tác T1, T2, ..., Tn được xử lý đồng thời là một thứ tự thực hiện xen kẽ các hành động của **n** giao tác này.
- Thứ tự xuất hiện của các thao tác trong lịch phải **giống** với thứ tự xuất hiện của chúng trong giao tác.
- Bộ lập lịch (Scheduler): Là một thành phần của DBMS có nhiệm vụ lập một lịch để thực hiện n giao tác xử lý đồng thời.
- Các loại lịch thao tác:
 - **Lịch tuần tự (Serial)**
 - **Lịch khả tuần tự (Serializable):**
 - Conflict – Serializability
 - View – Serializability

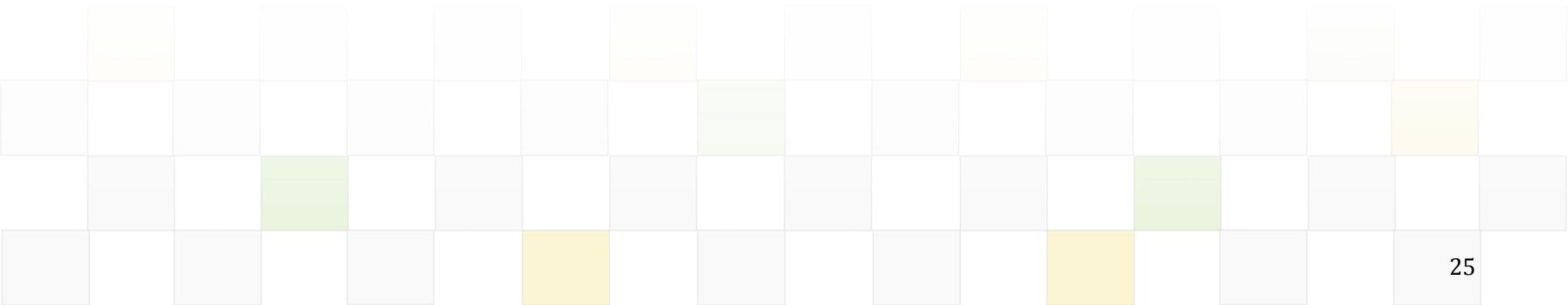
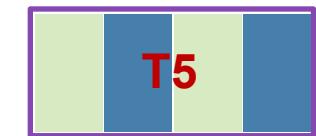
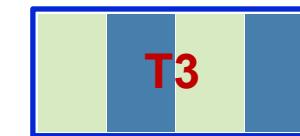
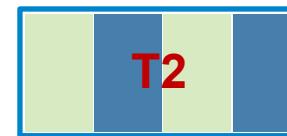


Lịch tuần tự

- Một lịch S được gọi là tuần tự nếu các hành động của các giao tác Ti được thực hiện liên tiếp nhau, không có sự giao nhau về mặt thời gian.



S



Lịch tuần tự

Ví dụ:

S↓1

Read(A, t)
 $t := t + 100$
 Write(A, t)
 Read(B, t)
 $t := t + 100$
 Write(B, t)

T1

A

B

25
 125
 125

Read(A, s)
 $s := s * 2$
 Write(A, s)
 Read(B, s)
 $s := s * 2$
 Write(B, s)

250
 250



S↓2

Read(A, s)
 $s := s * 2$
 Write(A, s)
 Read(B, s)
 $s := s * 2$
 Write(B, s)

T1

T2

A

B

25
 25

Read(A, t)
 $t := t + 100$
 Write(A, t)
 Read(B, t)
 $t := t + 100$
 Write(B, t)

150
 150

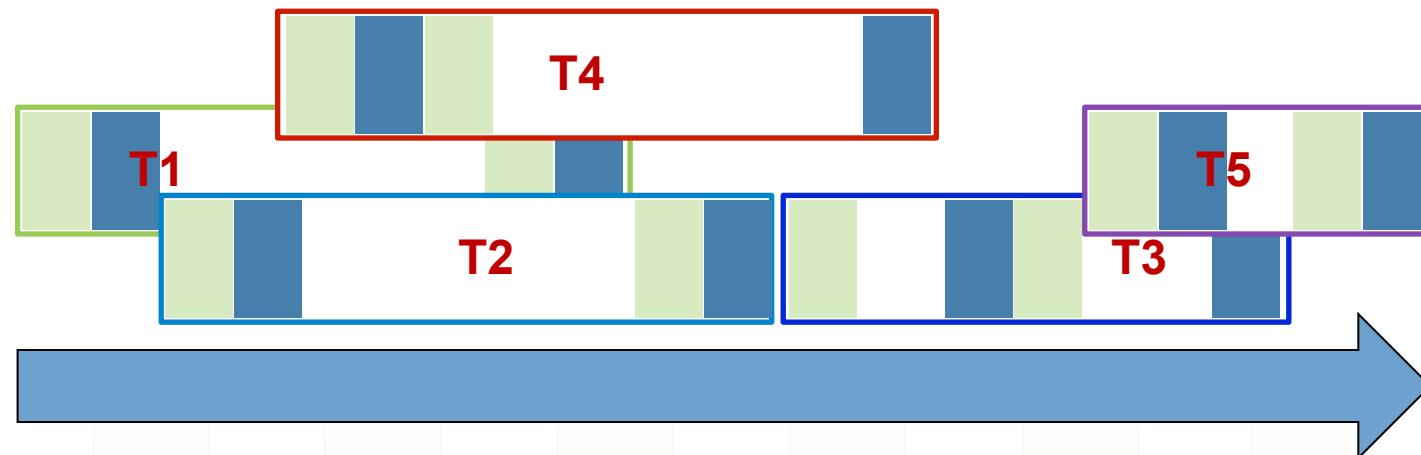


Lịch tuần tự luôn đảm bảo được tính nhất quán của CSDL



Lịch xử lý đồng thời

- Lịch xử lý đồng thời là lịch mà các giao tác trong đó giao nhau về mặt thời gian



Lịch xử lý đồng thời

Lịch đồng thời luôn luôn tiềm ẩn khả năng làm cho CSDL mất tính nhất quán

Lịch đồng thời

S3

	<i>T1</i>	<i>T2</i>	<i>A</i>	<i>B</i>
Read(A, t) t:=t+100 Write(A,t)	Read(A, s) s:=s*2 Write(A,s) Read(B, s) s:=s*2 Write(B, s)	25 125 250	25 50 150	
Read(B, t) t:=t+100 Write(B, t)				

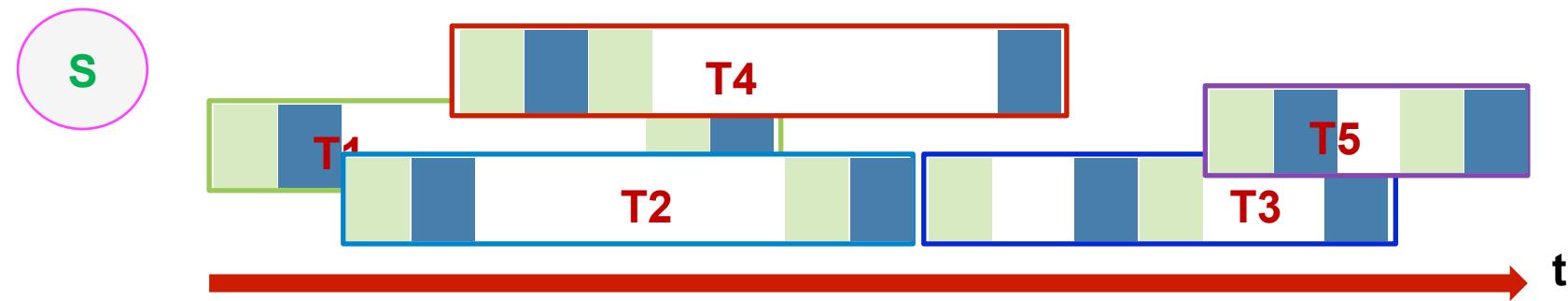
Ví dụ:

- S3 là một lịch xử lý đồng thời vì các giao tác giao thoa với nhau
- Lịch xử lý đồng thời S3 gây ra sự mất nhất quán dữ liệu
 - Trước S khi thực hiện
 - A=B
 - Sau khi S kết thúc
 - A \neq B

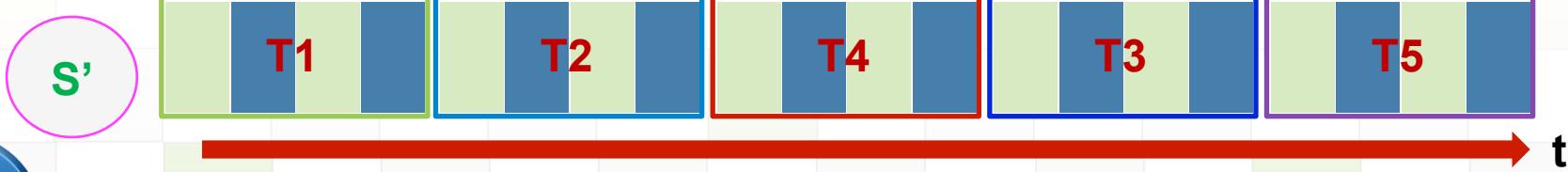


Lịch khả tuần tự

- Một lịch S được lập ra từ n giao tác T1, T2, ..., Tn *xử lý đồng thời* được gọi là **lịch khả tuần tự** nếu nó cho cùng kết quả với một lịch tuần tự nào đó được lập ra từ n giao tác này.



Tương đương



Lịch khả tuần tự cũng không gây nên tình trạng mất nhất quán dữ liệu

Lịch khả tuần tự

Ví dụ:

- Trước S_4 khi thực hiện
 - $A=B=c$
 - với c là hằng số
- Sau khi S_4 kết thúc
 - $A=2*(c+100)$
 - $B=2*(c+100)$
- Trạng thái CSDL nhất quán
- S_4 là **khả tuần tự**

S_4	T1	T2	A	B
	Read(A, t) $t:=t+100$ Write(A,t)		25	25
		Read(A, s) $s:=s*2$ Write(A,s)	125	250
	Read(B, t) $t:=t+100$ Write(B, t)		125	250
		Read(B, s) $s:=s*2$ Write(B, s)		250

(Kết quả của lịch S_4 tương tự với kết quả của lịch tuần tự $S < T1, T2 >$)

Lịch khả tuần tự

Ví dụ:

- Trước S_5 khi thực hiện
 - $A=B=c$
 - với c là hằng số
- Sau khi S_5 kết thúc
 - $A = 2*(c+100)$
 - $B = 2*c + 100$

→ Trạng thái CSDL không nhất quán

- S_5 **không** khả tuần tự

S_5	T1	T2	A	B
	Read(A, t) $t:=t+100$ Write(A,t)		25	25
		Read(A, s) $s:=s^*2$ Write(A,s) Read(B, s) $s:=s^*2$ Write(B, s)	125	250
	Read(B, t) $t:=t+100$ Write(B, t)		50	150

Lịch khả tuần tự

Ví dụ:

- Trước S_{5b} khi thực hiện
 - $A=B=c$
 - với c là hằng số
- Sau khi S_{5b} kết thúc
 - $A = 1*(c+100)$
 - $B = 1*c + 100$

→ Trạng thái CSDL vẫn nhất quán

S_{5b}	T1	T2	A	B
	Read(A, t) $t:=t+100$ Write(A,t)		25	25
		Read(A, s) $s:=s^*1$ Write(A,s) Read(B, s) $s:=s^*1$ Write(B, s)	125	125
	Read(B, t) $t:=t+100$ Write(B, t)		25	125

- ❑ Để xem xét tính nhất quán một cách kỹ lưỡng → phải hiểu rõ ngữ nghĩa của từng giao tác → không khả thi
- ❑ Thực tế chỉ xem xét các lệnh của giao tác là đọc hay ghi





Biểu diễn lịch thao tác

Cách 1

S	T1	T2
	Read(A, t) t:=t+100 Write(A,t)	Read(A, s) s:=s*2 Write(A,s)
	Read(B, t) t:=t+100 Write(B, t)	Read(B, s) s:=s*2 Write(B, s)

Cách 3

S: R1(A); W1(A); R2(A); W2(A); R1(B); W1(B); R2(B); W2 (B)

Cách 2

S	T1	T2
	Read(A) Write(A)	Read(A) Write(A)
	Read(B) Write(B)	Read(B) Write(B)



Lịch khả tuần tự

- Có 2 loại lịch khả tuần tự:

**Conflict
Serializable**

- Dựa trên ý tưởng hoán vị các hành động không xung đột để chuyển một lịch đồng thời S về một lịch tuần tự S'. Nếu có một cách biến đổi như vậy thì S là một lịch conflict serializable.

**View
Serializable**

- Dựa trên ý tưởng lịch đồng thời S và lịch tuần tự S' đọc và ghi những giá trị dữ liệu giống nhau. Nếu có một lịch S' như vậy thì S là một lịch view serializable.



Conflict Serializability

- **Ý tưởng:** Xét 2 hành động liên tiếp nhau của 2 giao tác khác nhau trong một lịch thao tác, khi 2 hành động đó được đảo thứ tự có thể dẫn đến 1 trong 2 hệ quả:
 - Hoạt động của cả hai giao tác chứa 2 hành động ấy **không** bị ảnh hưởng gì → **2 hành động đó không xung đột với nhau**
 - Hoạt động của ít nhất một trong 2 giao tác chứa 2 hành động ấy bị ảnh hưởng → **2 hành động xung đột**

T	T'
Hành động 1	
Hành động 2	Hành động 1' Hành động 2'
	Hành động 3' Hành động 4'
Hành động 3	
Hành động 4	Hành động 3' Hành động 4'



Conflict Serializability (tt)

- Cho lịch S có 2 giao tác T_i và T_j , xét các trường hợp:
 - $r_i(X) ; r_j(Y)$
 - **Không bao giờ có xung đột**, ngay cả khi $X=Y$
 - Cả 2 thao tác không làm thay đổi giá trị của X và Y
 - $r_i(X) ; w_j(Y)$
 - **Không xung đột khi $X \neq Y$**
 - T_j không thay đổi dữ liệu đọc của T_i
 - T_i không sử dụng dữ liệu ghi của T_j
 - **Xung đột khi $X = Y$**
 - $w_i(X) ; r_j(Y)$
 - **Không xung đột khi $X \neq Y$, Xung đột khi $X=Y$**
 - $w_i(X) ; w_j(Y)$
 - **Không xung đột khi $X \neq Y$, Xung đột khi $X=Y$**



Conflict Serializability (tt)

- Tóm lại, hai hành động xung đột nếu chúng:

- ❖ Thuộc 2 giao tác khác nhau
- ❖ Truy xuất đến 1 đơn vị dữ liệu
- ❖ Trong chúng có ít nhất một hành động ghi (write)

- Hai hành động xung đột thì không thể nào đảo thứ tự của chúng trong một lịch thao tác.

Conflict Serializability (tt)

- Ví dụ:

S_6	T_1	T_2	
Read(A)			Read(A)
Write(A)			Write(A)
	Read(A)		Read(B)
	Write(A)		Write(B)
Read(B)			Read(A)
Write(B)			Write(A)
	Read(B)		Read(B)
	Write(B)		Write(B)

S_6 có khả năng chuyển đổi thành S'_6 bằng cách hoán vị các cặp hành động không xung đột hay không ?



Conflict Serializability (tt)

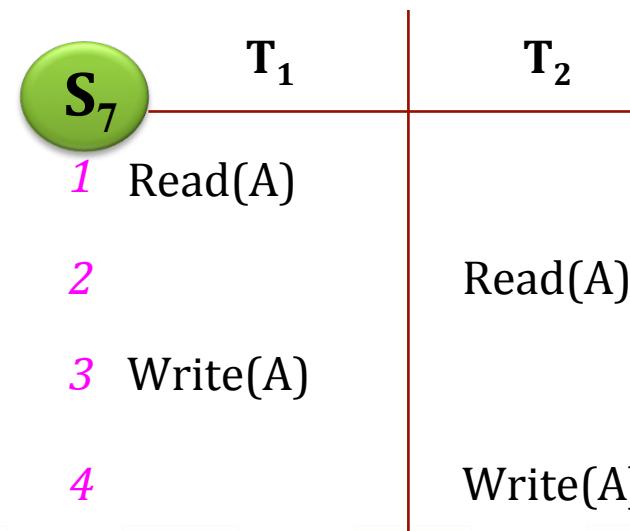
- Định nghĩa:

- S và S' là những lịch thao tác *conflict equivalent* nếu S có thể chuyển được thành S' thông qua một chuỗi các hoán vị những thao tác không xung đột.
- Một lịch thao tác S là conflict serializable nếu S là *conflict equivalent* với một lịch thao tác tuần tự nào đó.

- S là conflict serializable thì S khả tuần tự.
- S là khả tuần tự thì không chắc S conflict serializable
- Lịch S ở slide trước có phải conflict serializable hay không ?

Conflict Serializability (tt)

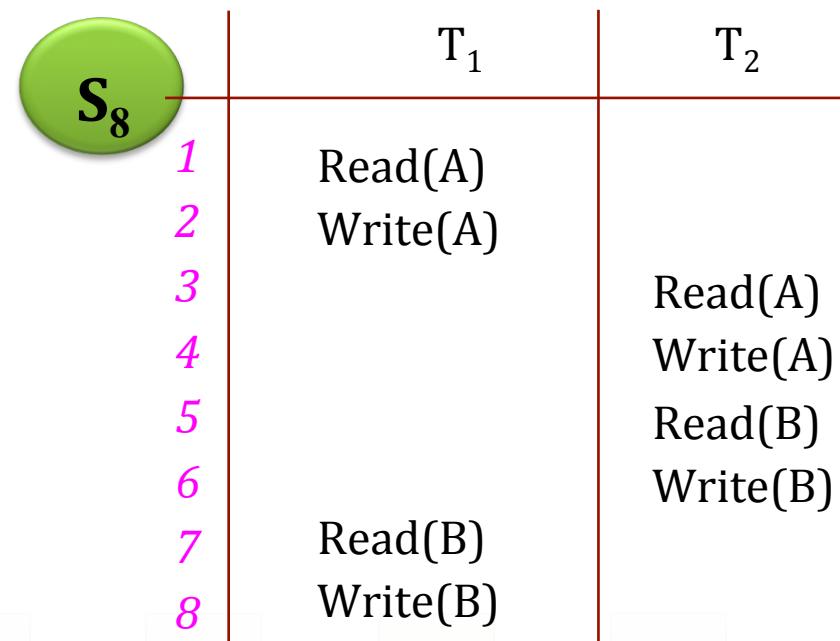
- Xét lịch S7 như sau:



- Lịch S7 trên có thỏa Conflict Serializability hay không ?

Conflict Serializability (tt)

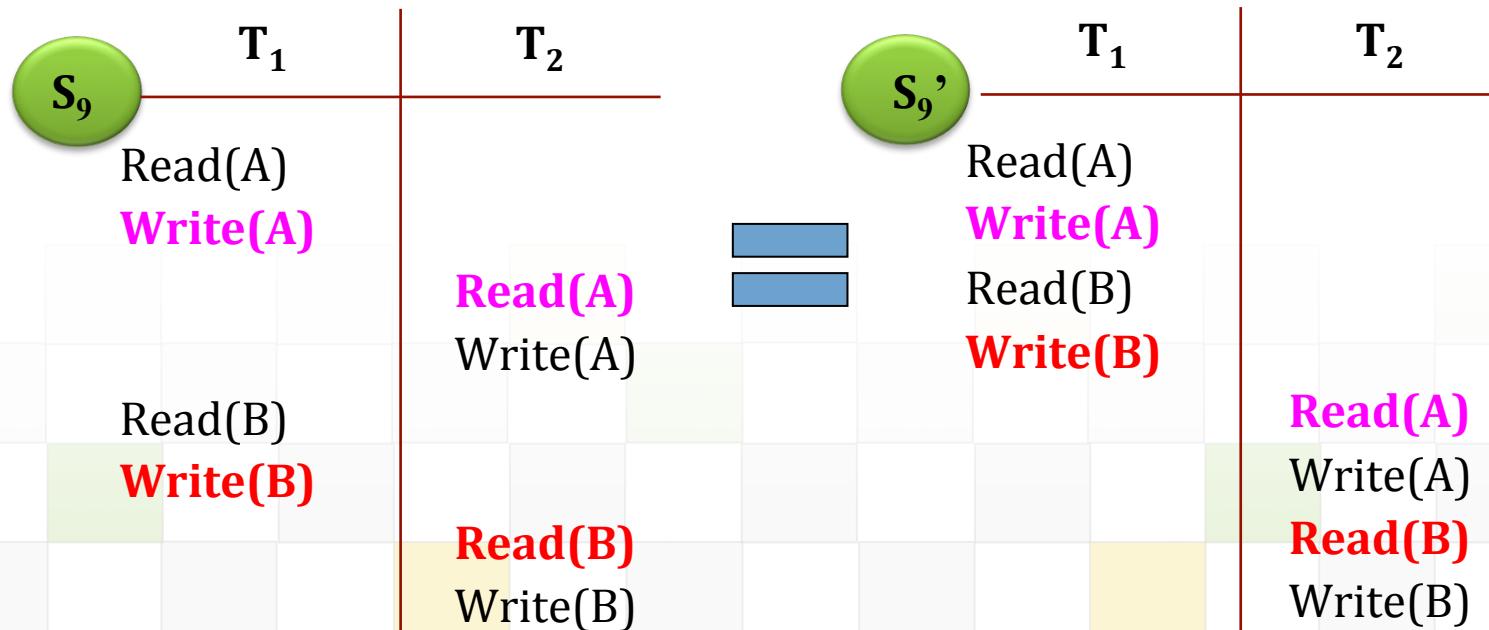
- Xét lịch S_8 như sau:



- Lịch S_8 trên có thỏa Conflict Serializability hay không ?

Kiểm tra Conflict Serializability

- Cho lịch S_9 :
 - S_9 có conflict serializability hay không ?
- Ý tưởng:
 - Các hành động xung đột trong lịch S được thực hiện theo thứ tự nào thì các giao tác thực hiện chúng trong S' (kết quả sau hoán vị) cũng theo thứ tự đó.





Kiểm tra Conflict Serializability

- Cho lịch S có 2 giao tác T1 và T2:
 - T1 thực hiện hành động A1
 - T2 thực hiện hành động A2
 - Ta nói T1 thực hiện trước hành động **T2** trong S, ký hiệu **T1 <_S T2**, khi:
 - **A1** được thực hiện trước **A2** trong S
 - A1 không nhất thiết phải liên tiếp A2
 - **A1** và **A2** là 2 hành động xung đột (A1 và A2 cùng thao tác lên 1 đơn vị dữ liệu và có ít nhất 1 hành động là ghi trong A1 và A2)



Kiểm tra Conflict Serializability

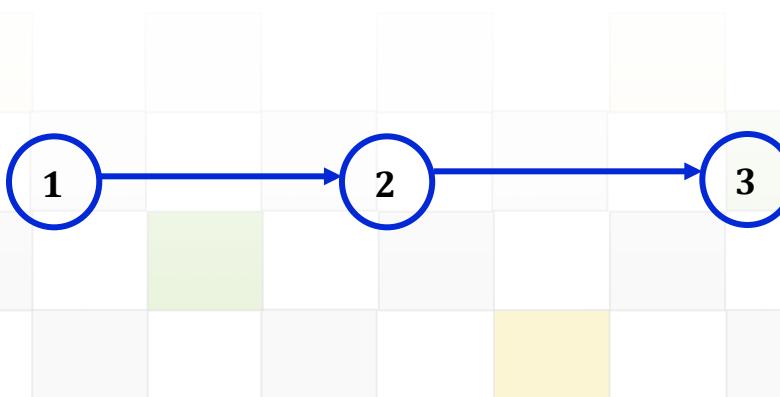
Phương pháp Precedence Graph:

- Cho lịch S bao gồm các thao tác T1, T2, ..., Tn
- Đồ thị trình tự của S (Precedence graph) của S ký hiệu là P(S) có:
 - Đỉnh là các giao tác **T_i**
 - Cung đi từ **T_i** đến **T_j** nếu $T_i <_S T_j$
- S Conflict Serializable khi và chỉ khi P(S) không có chu trình
- Thứ tự hình học các đỉnh là thứ tự của các giao tác trong lịch tuần tự tương đương với S.
- Với 2 lịch S và S' được lập từ cùng các giao tác, S và S' conflict equivalent khi và chỉ khi $P(S)=P(S')$

Kiểm tra Conflict Serializability

- Ví dụ 1: S_{10} có Conflict Serializability hay không ?

S_{10}	T_1	T_2	T_3
	Read(B)	Read(A) Write(A)	Read(A)
	Write(B)	Read(B) Write(B)	Write(A)



- * $P(S_{10})$ không có chu trình
- * S_{10} conflict-serializable theo thứ tự T_1, T_2, T_3

Kiểm tra Conflict Serializability

- Ví dụ 1: S_{10} có Conflict Serializability hay không ?

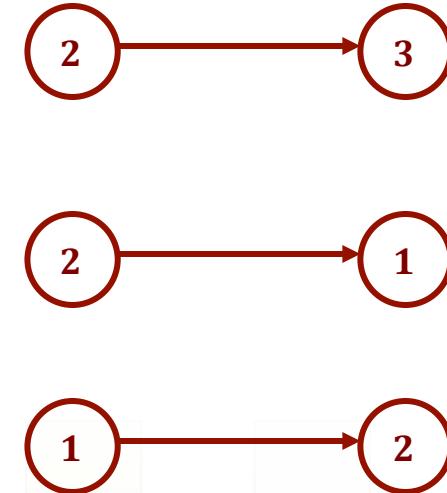
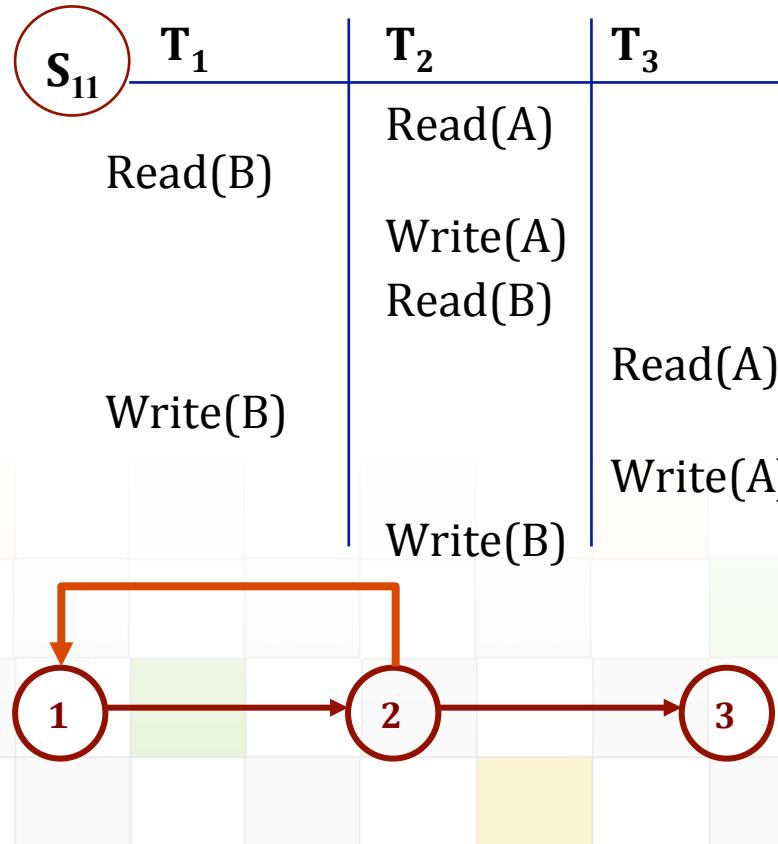
S_{10}	T_1	T_2	T_3
		Read(A)	
Read(B)		Write(A)	
		Read(A)	
Write(B)			Write(A)
		Read(B)	
		Write(B)	

S	T_1	T_2	T_3
		Read(B)	
		Write(B)	
			Read(A)
			Write(A)
			Read(B)
			Write(B)
			Read(A)
			Write(A)

Kiểm tra Conflict Serializability

- Ví dụ 2: S_{11} có Conflict Serializability hay không ?

S_{11} : r2(A) r1(B) w2(A) r2(B) r3(A) w1(B) w3(A) w2(B)



* $P(S_{11})$ có chu trình
* S_{11} không conflict-serializable



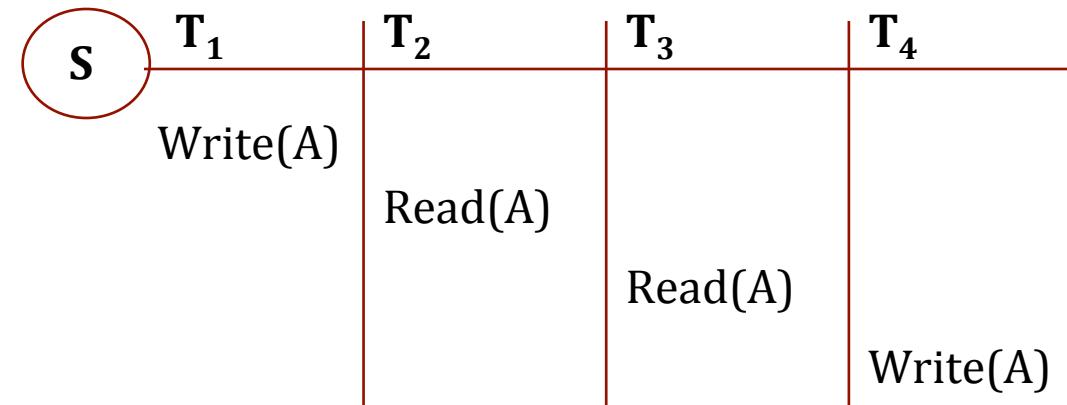
Bài tập Conflict-Serializability

- Cho các lịch S sau:
 1. S: w1(A) r2(A) r3(A) w4 (A)
 2. S: w3(A) w2(C) r1(A) w1(B) r1(C) w2(A) r4(A) w4(D)
 3. **S: r1(A) w2(A) w1(A) w3(A)**
 4. **S: r2(B) w2(A) r1(A) r3(A) w1(B) w2(B) w3(B)**
 5. S: w1(X) w2(Y) w2(X) w1(X) w3(X)
 6. S: r2(A) r1(B) w2(A) r3(A) w1(B) r2(B) w2(B)
 7. S: r2(A) r1(B) w2(A) r2(B) r3(A) w1(B) w3(A) w2(B)
- Vẽ P(S)
- S có conflict-serializable không? Nếu có thì S tương đương với lịch tuần tự nào ?

Bài tập 1

- Cho lịch S:

S: w1(A) r2(A) r3(A) w4 (A)

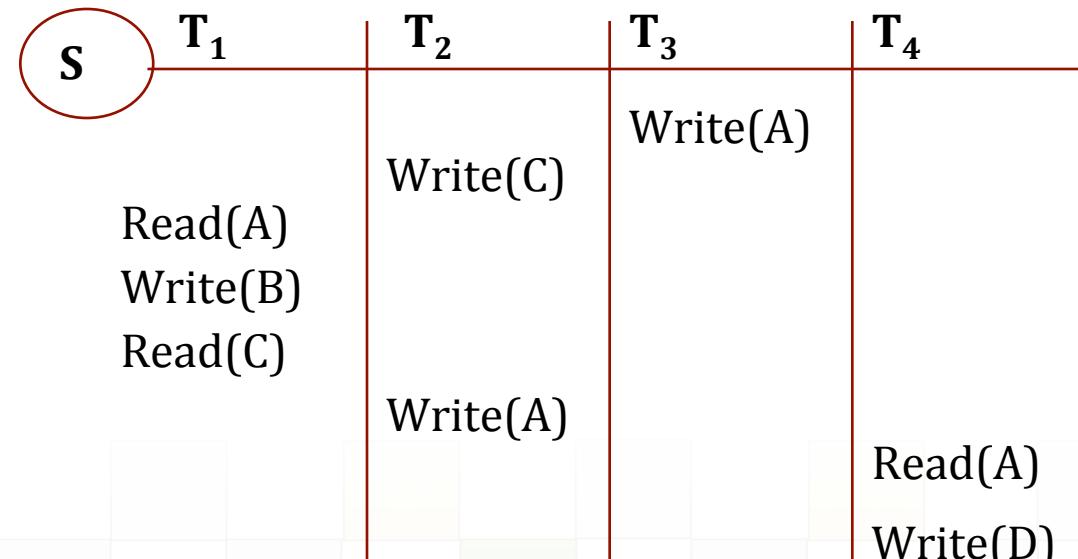


- Vẽ P(S)
- S có conflict-serializable không?

Bài tập 2

- Cho lịch S:

S: w3(A) w2(C) r1(A) w1(B) r1(C) w2(A) r4(A) w4(D)

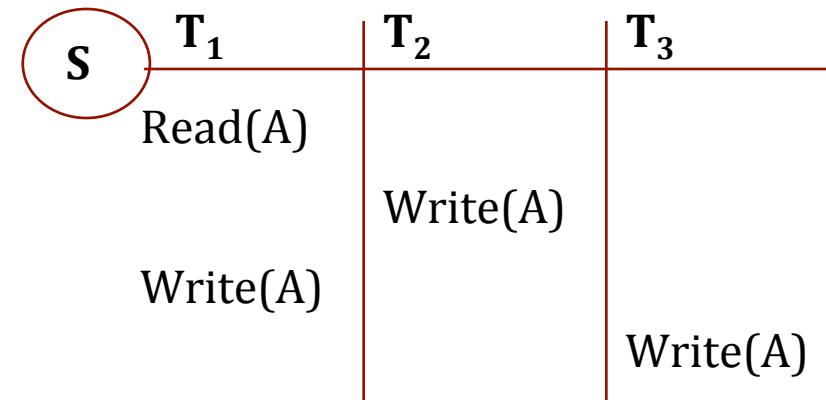


- Vẽ P(S)
- S có conflict-serializable không?

Bài tập 3

- Cho lịch S:

S: r1(A) w2(A) w1(A) w3(A)



- Vẽ P(S)
- S có conflict-serializable không?



Bài tập 4

- Cho lịch S:

S: r2(B) w2(A) r1(A) r3(A) w1(B) w2(B) w3(B)

- Vẽ P(S)
- S có conflict-serializable không ?

Bài tập 5

- Cho lịch S:

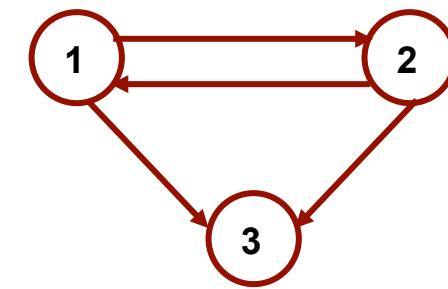
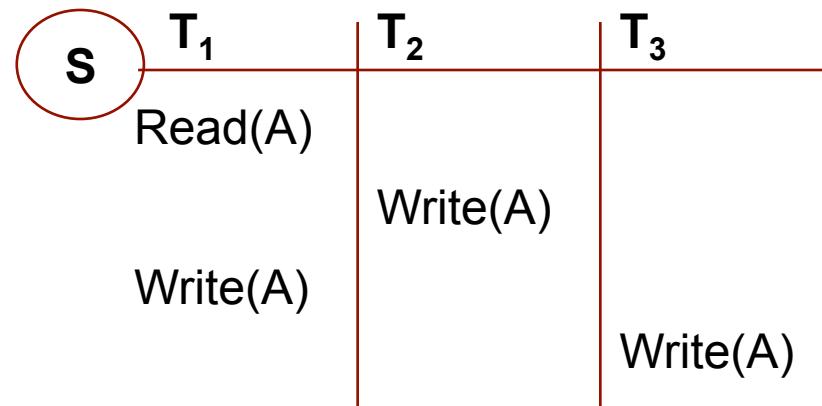
S: w1(X) w2(Y) w2(X) w1(X) w3(X)

S	T1	T2	T3
	Write(Y)		
		Write(Y)	
		Write(X)	
	Write(X)		
			Write(X)

- Vẽ đồ thị P(S)
- S có conflict serializable hay không ?

View-Serializability

- Xét lịch S

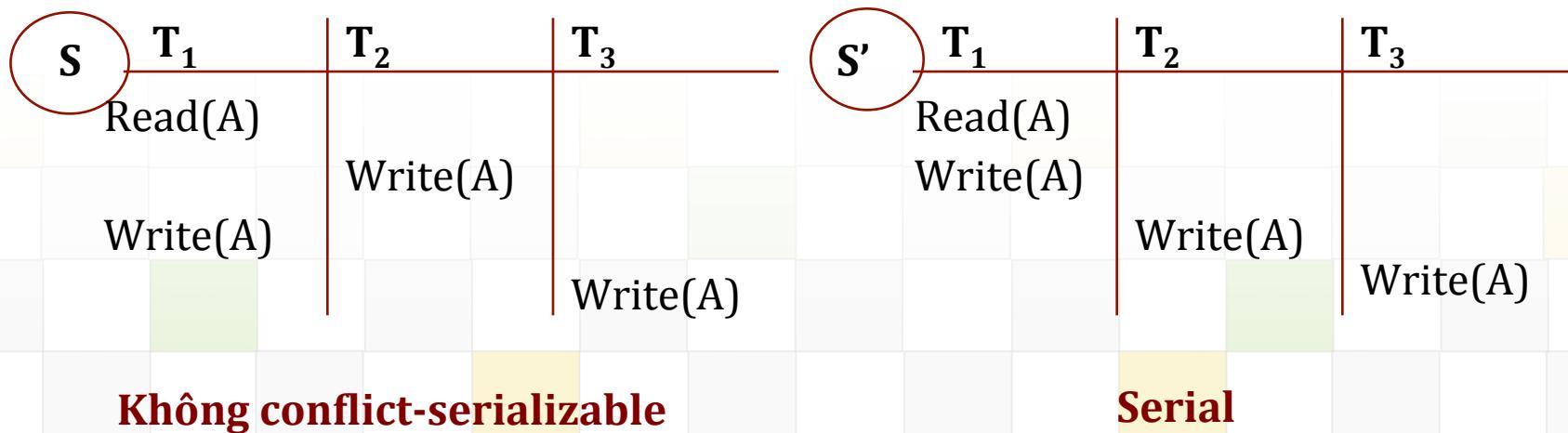


- * *P(S) có chu trình*
- * *S không conflict-serializable*

- * *S khả tuần tự hay không ?*

View-Serializability (tt)

- So sánh lịch S và 1 lịch tuần tự S'
 - Giả sử trước khi lịch S thực hiện, có giao tác Tb thực hiện việc ghi A và sau khi S thực hiện có giao tác Tf thực hiện việc đọc A.
 - Nhận xét lịch S và S':
 - Đều có T1 thực hiện read(A) từ giao tác Tb → Kết quả đọc luôn giống nhau
 - Đều có T3 thực hiện việc ghi cuối cùng lên A. T2, T3 không có lệnh đọc A → Dù S hay S' được thực hiện thì kết quả đọc A của Tf luôn giống nhau
 - Kết quả của S và S' giống nhau → S vẫn khả tuần tự





View-Serializability (tt)

■ Khả tuần tự View (View-serializability):

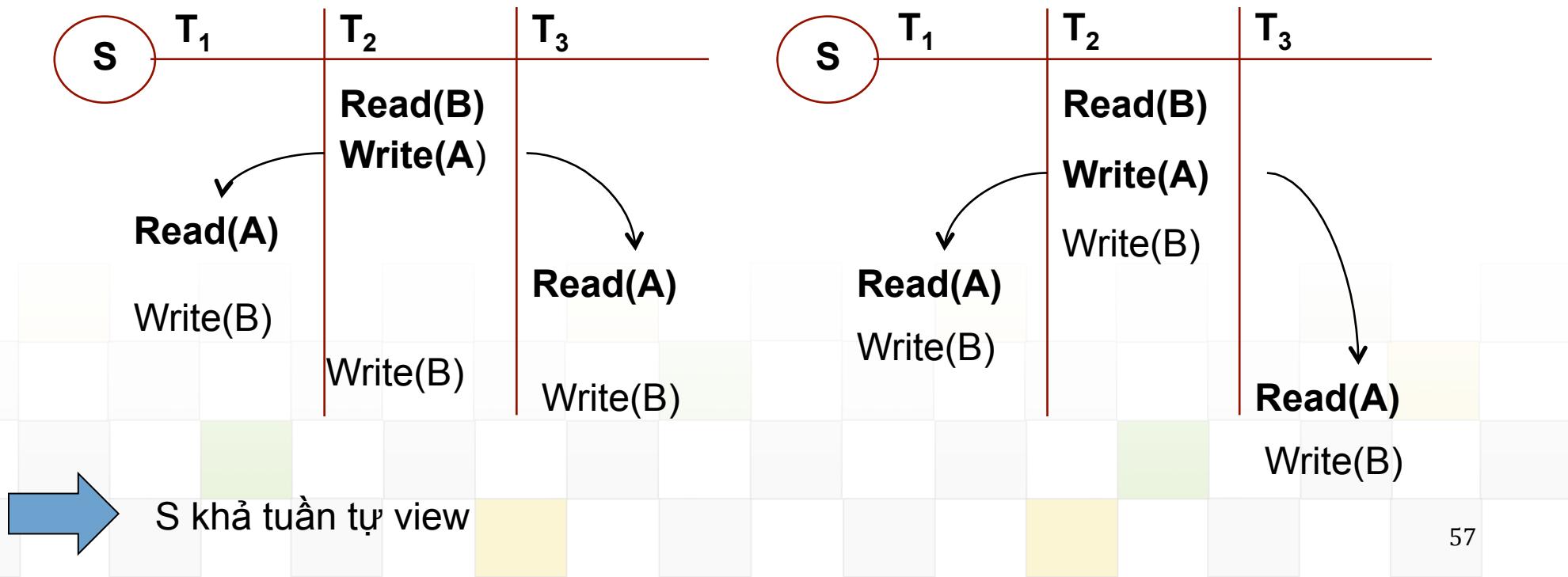
- Một lịch S được gọi là khả tuần tự view nếu tồn tại một lịch tuần tự S' được tạo từ các giao tác của S sao cho S và S' đọc và ghi những giá trị giống nhau.
- Lịch S được gọi là khả tuần tự view khi và chỉ khi nó tương đương view (*view-equivalent*) với một lịch tuần tự S'

View-Serializability (tt)

- Ví dụ: Cho lịch S và S' như sau:

S : r2(B) w2(A) r1(A) r3(A) w1(B) w2(B) w3(B)

S': r2(B) w2(A) w2(B) r1(A) w1(B) r3(A) w3(B)



S khả tuân tự view



View-Serializability (tt)

- ***View-equivalent***: S và S' là những lịch view-equivalent nếu thỏa các điều kiện sau:
 - Nếu trong S có Ti đọc giá trị ban đầu của A thì nó cũng đọc giá trị ban đầu của A trong S'.
 - Nếu Ti đọc giá trị của A được ghi bởi Tj trong S thì Ti cũng phải đọc giá trị của A được ghi bởi Tj trong S'.
 - Với mỗi dvdl A, giao tác thực hiện lệnh ghi cuối cùng lên A (nếu có) trong S thì giao tác đó cũng phải thực hiện lệnh ghi cuối cùng lên A trong S'.
- Một lịch giao tác S là view-serializable:
 - Nếu S là view-equivalent với một Lịch giao tác tuần tự S' nào đó
- Nếu S là conflict-serializable \rightarrow S view-serializable.



Chứng minh (*)



View Serializability (tt)

Lịch thao tác

View-Serializable

Conflict-
Serializable

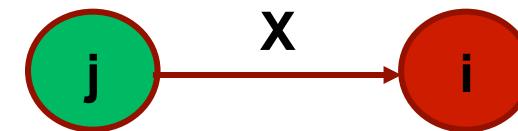


Kiểm tra View Serializability

- Cho 1 Lịch giao tác S
- Thêm 1 giao tác cuối **Tf** vào trong S sao cho **Tf** thực hiện việc *đọc hết tất cả* đơn vị dữ liệu ở trong S
 - $S = \dots w1(A) \dots \dots \dots w2(A) \text{ rf}(A)$
- Thêm 1 giao tác đầu tiên **Tb** vào trong S sao cho **Tb** thực hiện việc *ghi các giá trị ban đầu* cho tất cả đơn vị dữ liệu
 - $S = \text{wb}(A) \dots w1(A) \dots \dots \dots w2(A) \dots$

Kiểm tra View-Serializability (tt)

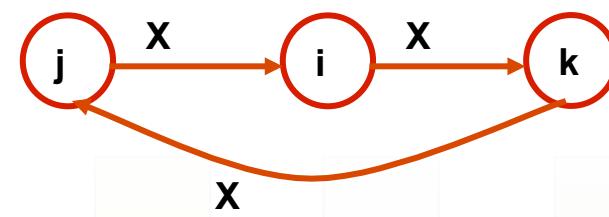
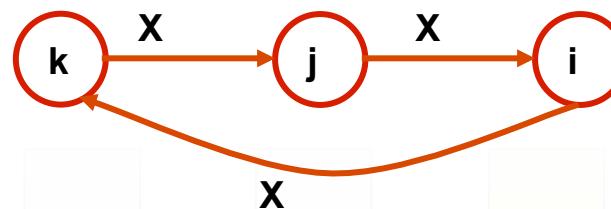
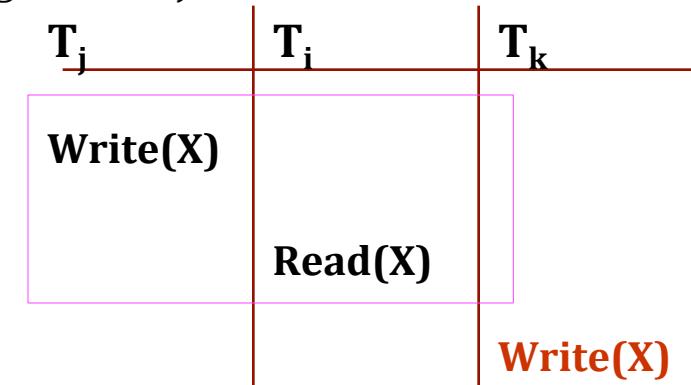
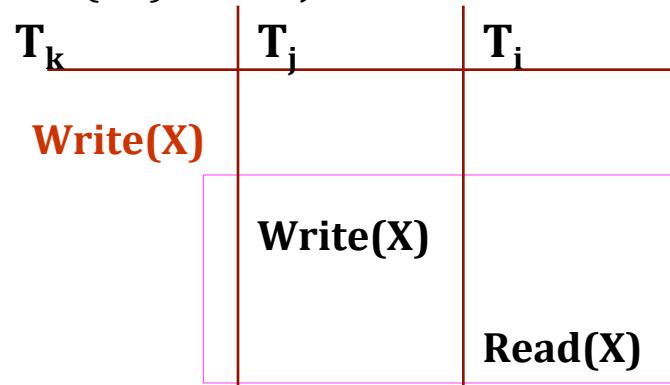
- Vẽ đồ thị phức (*PolyGraph*) cho S, ký hiệu **G(S)** với
 - Đỉnh là các giao tác **T_i** (bao gồm cả **T_b** và **T_f**)
 - Cung:
 - (1) Nếu giá trị mà **ri(X)** đọc được là do **T_j** ghi (*ri(X) có gốc là T_j*) thì vẽ cung đi từ **T_j** đến **T_i**



- (2) Với mỗi $wj(X) \dots ri(X)$, xét $wk(X)$ khác **T_b** sao cho **T_k** không chèn vào giữa **T_j** và **T_i**

Kiểm tra View-Serializability (tt)

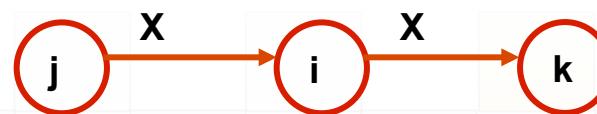
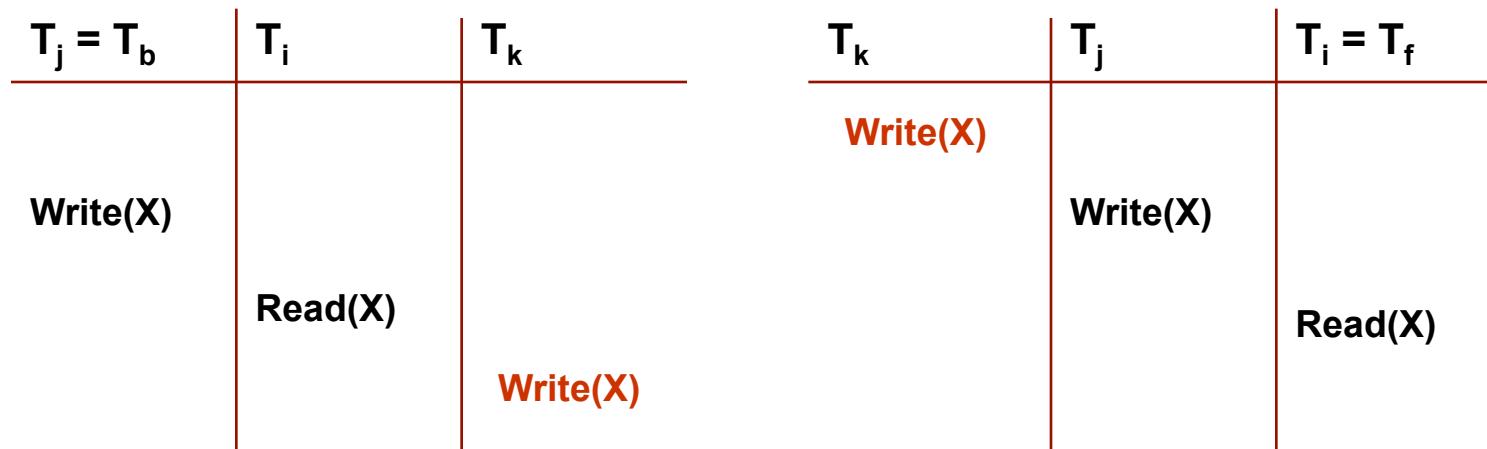
- (2a) Nếu $T_j \neq T_b$ và $T_i \neq T_f$ thì vẽ cung $T_k \rightarrow T_j$ và $T_i \rightarrow T_k$



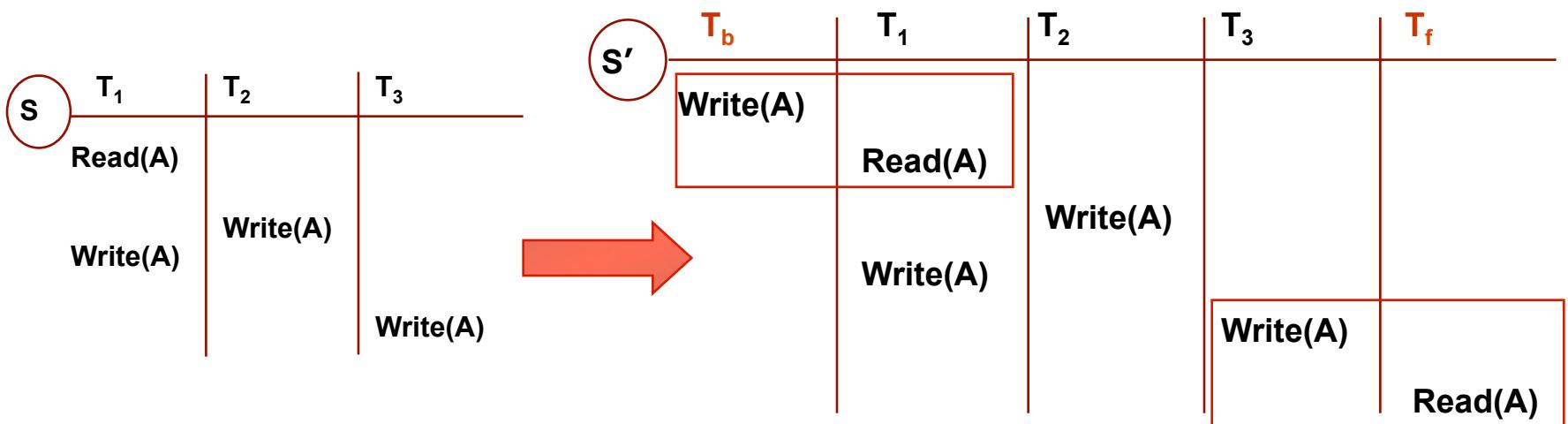
T_k có thể nằm trước T_j hoặc sau T_i

Kiểm tra View-Serializability (tt)

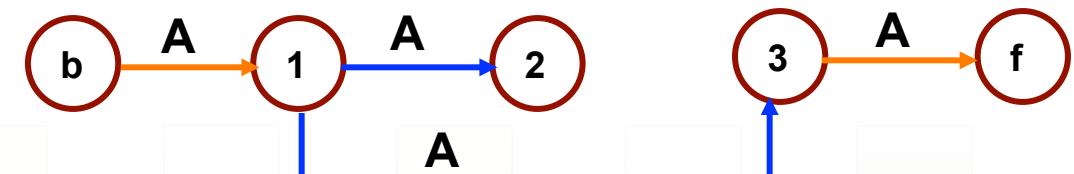
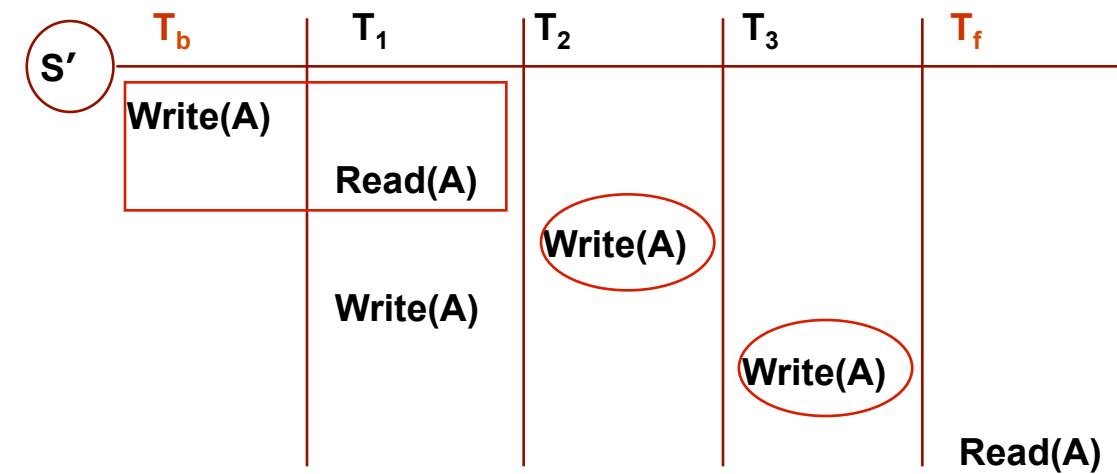
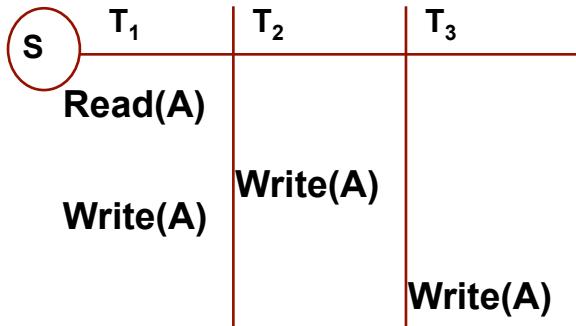
- (2b) Nếu $T_j = T_b$ thì vẽ cung $T_i \rightarrow T_k$
 - (2c) Nếu $T_i = T_f$ thì vẽ cung $T_k \rightarrow T_j$



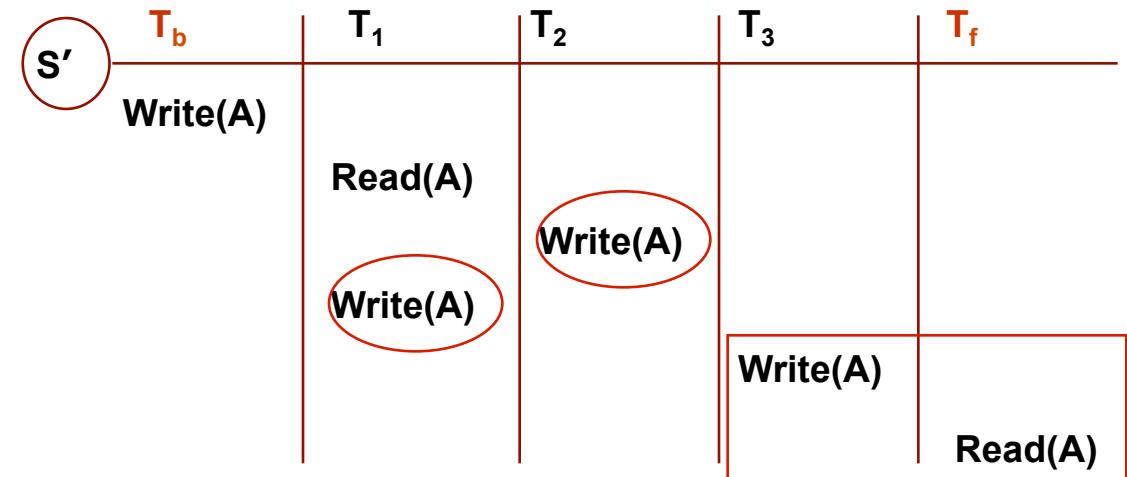
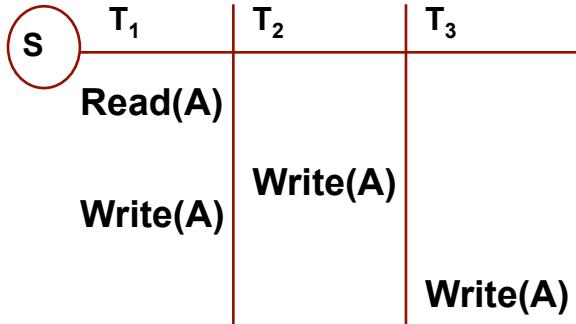
Ví dụ



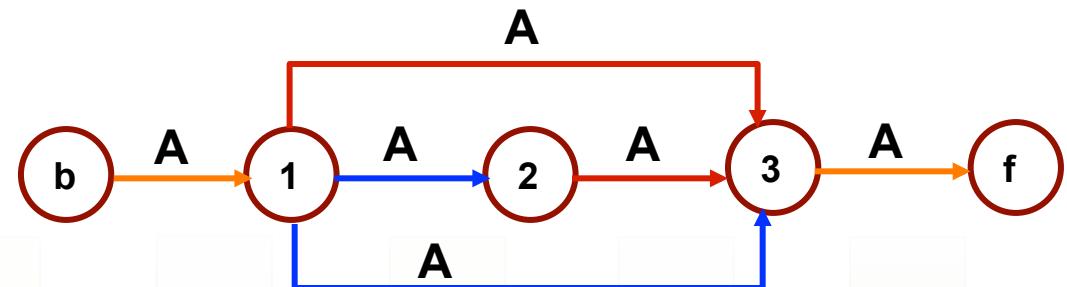
Ví dụ



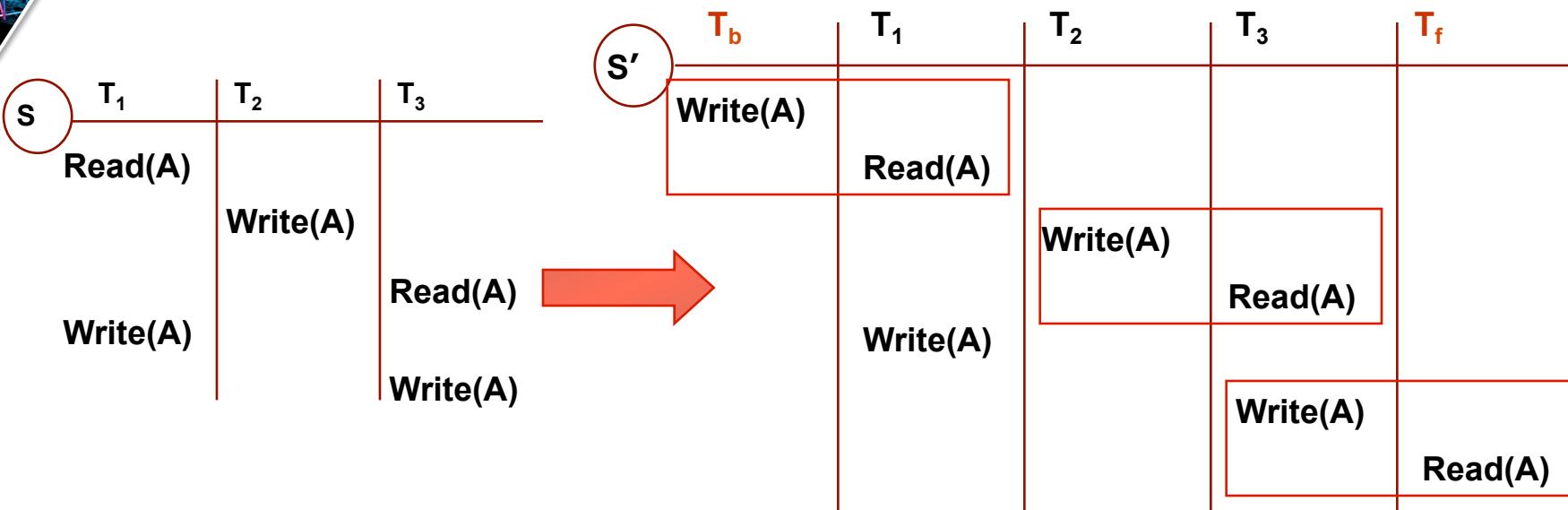
Ví dụ (tt)



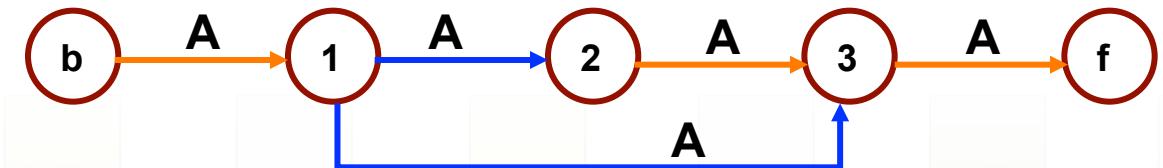
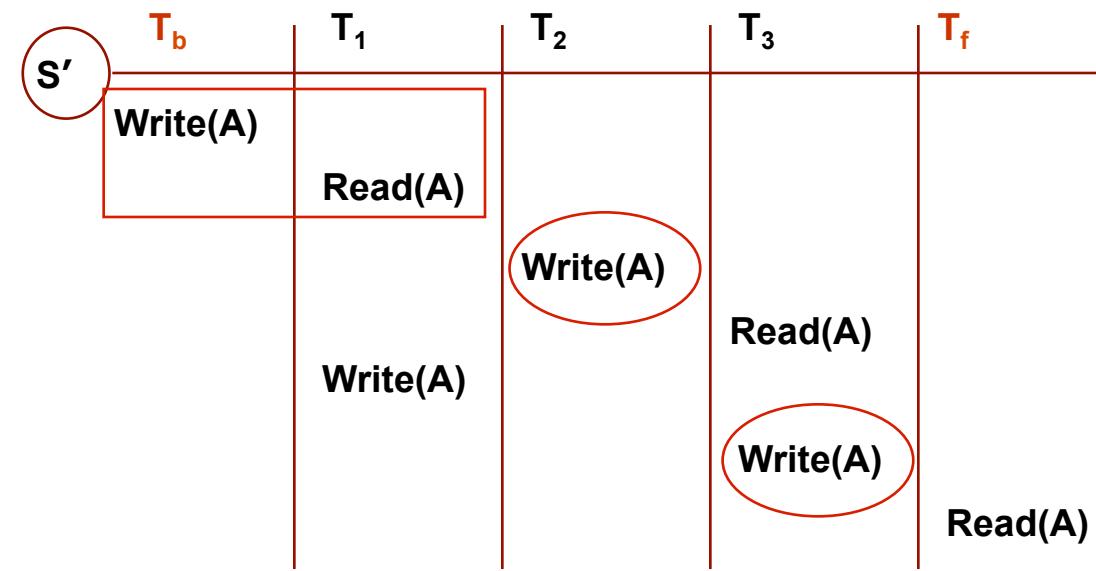
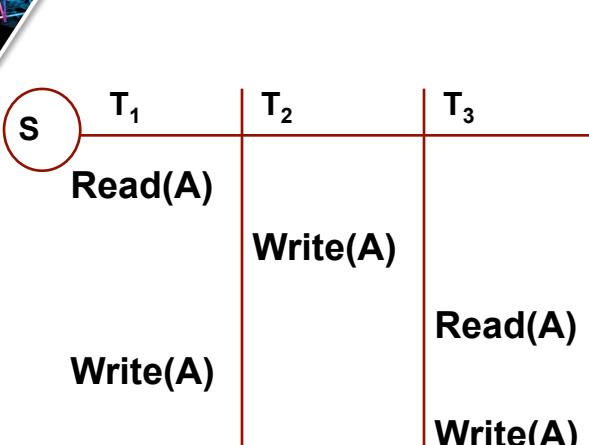
- * $G(S)$ không có chu trình
- * S view-serializable theo thứ tự T_b, T_1, T_2, T_3, T_f



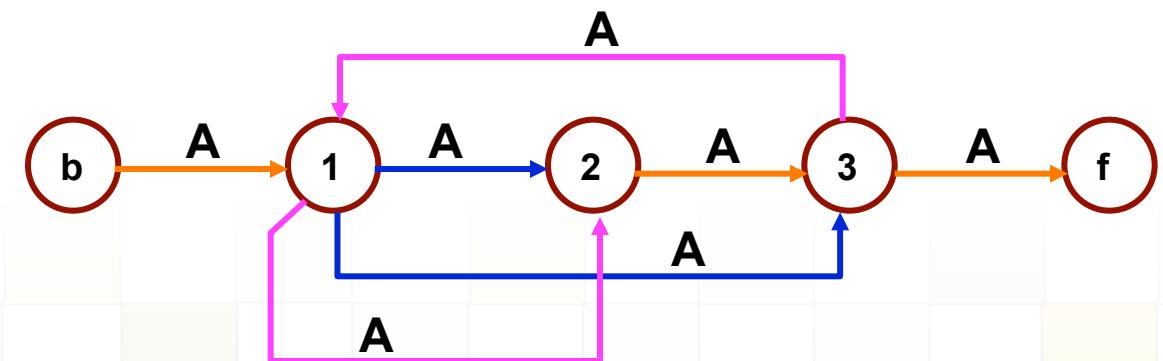
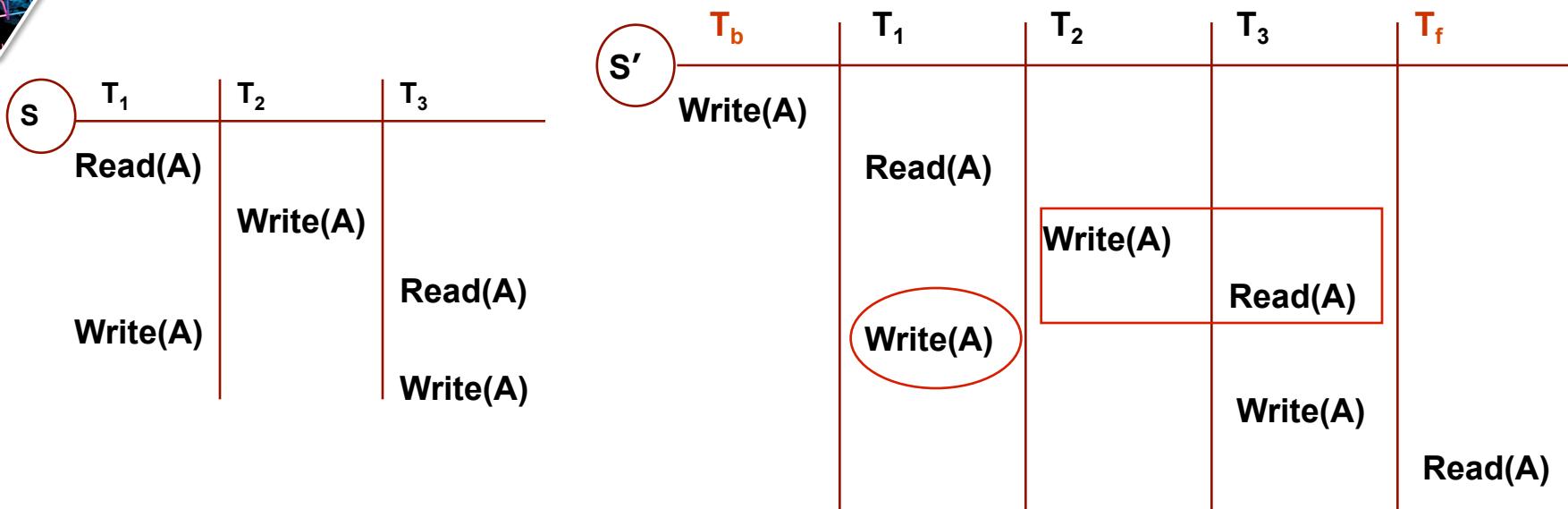
Ví dụ (tt)



Ví dụ (tt)

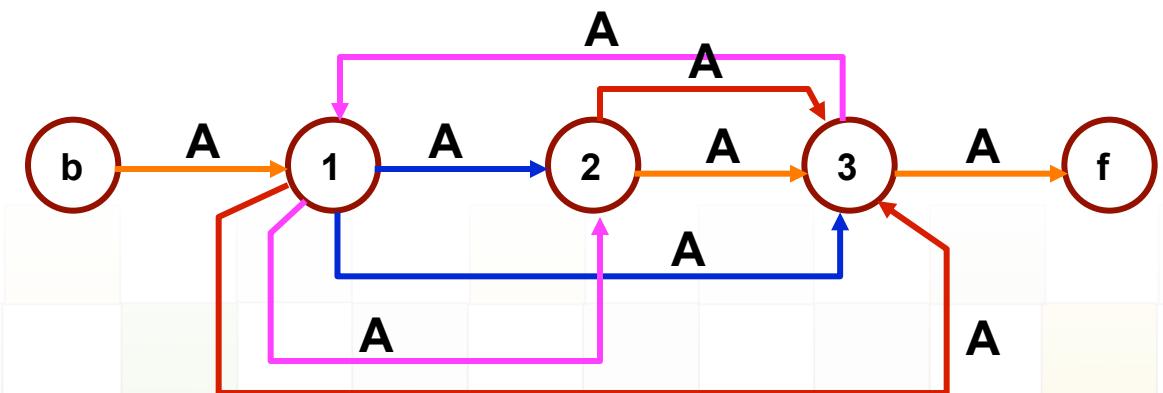
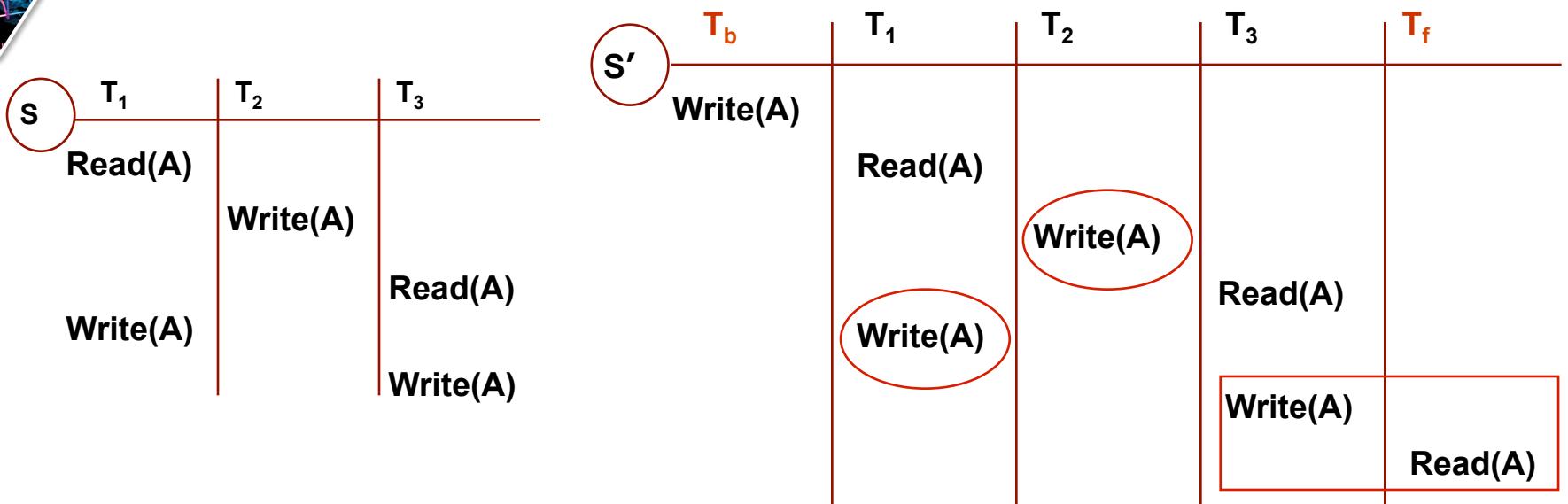


Ví dụ (tt)



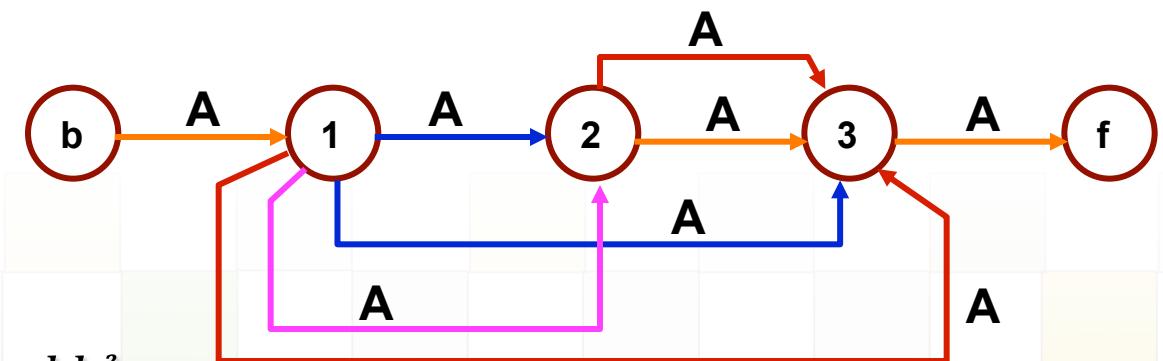
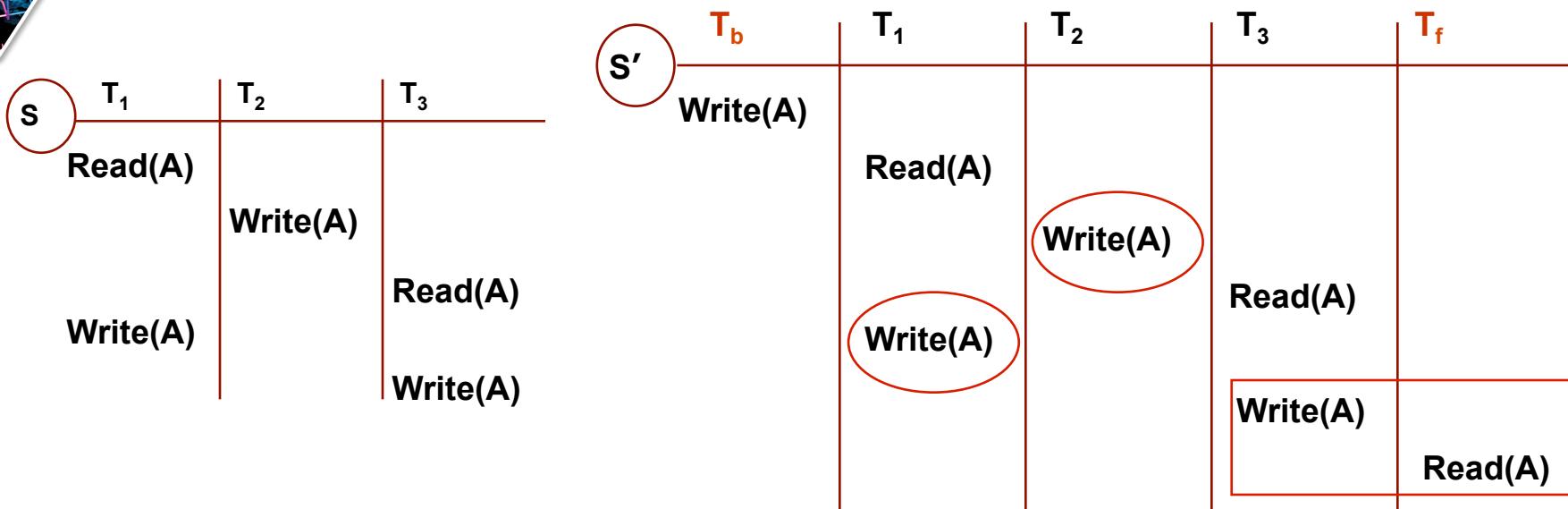
Chọn 1 trong 2 cung sao cho không có chu trình

Ví dụ (tt)



* $G(S)$ có chu trình

Ví dụ (tt)



- * $G(S)$ không có chu trình sau khi bỏ cung
- * S view-serializable



Bài tập View-Serializability

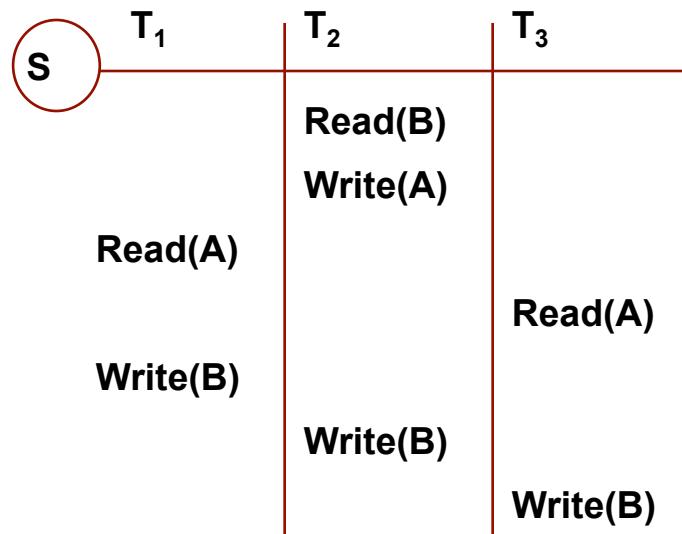
- Cho lịch S:
 1. S: r2(B) w2(A) r1(A) r3(A) w1(B) w2(B) w3(B)
 2. S: w1(A) r3(A) r2(A) w2(A) r1(A) w3(A)
 3. S: r2(A) r1(A) w1(C) r3(C) w1(B) r4(B) w3(A) r4(C) w2(D) r2(B) w4(A) w4(B)
 4. S: w1(A) r2(A) w2(A) r1(A)
 5. S: r1(A) r3(D) w1(B) r2(B) w3(B) r4(B) w2(C) r5(C) w4(E) r5(E) w5(B)
 6. S: w1(A) r2(A) w3(A) r4(A) w5(A) r6(A)
 7. S: r1(X) r2(X) w1(X) w2(X)
- Yêu cầu:
 - Vẽ G(S)
 - S có view-serializable hay không ?



TÓM TẮT CHƯƠNG 2

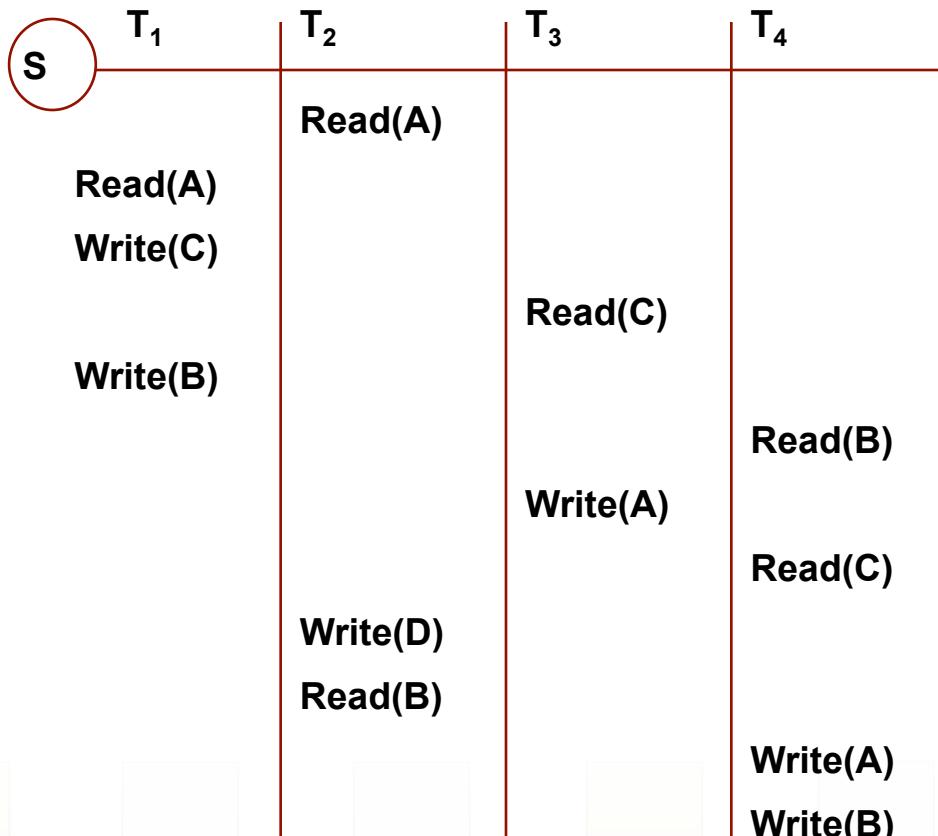
- Một số tình huống về tranh chấp
- Khái niệm giao tác
- Tính chất ACID của giao tác
- Lịch thao tác:
 - Lịch tuần tự
 - Lịch đồng thời
 - Lịch Khả tuần tự
 - Lịch khả tuần tự xung đột (Conflict Serializability)
 - Kiểm tra khả tuần tự xung đột bằng đồ thị trình tự (Prcedence graph)
 - Lịch khả tuần tự view (View Serializability)
 - Kiểm tra khả tuần tự view bằng đồ thị phức (Poly graph)

Bài tập



- * *Vẽ G(S)*
- * *S có view-serializable?*

Bài tập (tt)



* $Vẽ G(S)$

* S có view-serializable?



TÀI LIỆU THAM KHẢO

- [1] *Database Management Systems*, 3rd Edition, Raghu Ramakrishnan and Johannes Gehrke
- [2] *Fundamentals of Database Systems*, 4th Edition, Elmasri Navathe
- [3] *Database System Concepts*, 4th Edition, Silberschatz–Korth –Sudarshan
- [4] *Database Systems Implementation*, Hector Garcia-Molina, D. Ullman, Jennifer D. Widom
- [5] *Database systems: the complete book*, Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom, Pearson Prentice Hall, 2009



TÀI LIỆU THAM KHẢO

- <http://infolab.stanford.edu/~ullman/dscb/vs-old.pdf>
- http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed-english/Ch17_CC-95.pdf
- <http://djitz.com/neu-mscs/how-to-check-for-view-serializable-and-conflict-serializable/>
- <http://inst.eecs.berkeley.edu/~cs186/fa05/lecs/18cc-6up.pdf>
- http://folk.uio.no/inf212/handouts/uke19_2.pdf



HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

Chương 3:
**ĐIỀU KHIỂN TRUY
XUẤT ĐỒNG THỜI**

GVLT: *Nguyễn Trường Sơn*



Nội dung trình bày

- Phân loại các vấn đề của truy xuất đồng thời
- Các kỹ thuật điều khiển đồng thời:
 - Kỹ thuật khoá (Locking)
 - Kỹ thuật nhãn thời gian (Timestamp)
 - Kỹ thuật xác nhận hợp lệ (Validation)
- Vấn đề khoá chết
- Các vấn đề khác



Nội dung trình bày

- Phân loại các vấn đề của truy xuất đồng thời:
 - Mất dữ liệu cập nhật
 - Không đọc lại được dữ liệu
 - Bóng ma
 - Đọc phải dữ liệu rác
- Các kỹ thuật điều khiển đồng thời:
 - Kỹ thuật khoá
 - Kỹ thuật nhãn thời gian
 - Kỹ thuật xác nhận hợp lệ
- Vấn đề khoá chết
- Các vấn đề khác



Vấn đề mất dữ liệu đã cập nhật

- Xét 2 giao tác T1 và T2 và đơn vị dữ liệu A vốn có giá trị ban đầu là 50:

- T1: Read(A); $A:=A+10$; Write(A)
- T2: Read(A); $A:=A+20$; Write(A)

★ Nếu T1 và T2 thực hiện tuần tự (T1 rồi T2 hoặc T2 rồi T1) thì cuối cùng $A = 80$

★ Nếu T1 và T2 thực hiện đồng thời như lịch bên: $A = 70$

A=50	T1	T2
t1	Read(A)	
t2		Read(A)
t3	$A:=A+10$	
t4		$A:=A+20$
t5	Write(A)	
t6		Write(A)

Nhận xét:

- ✧ Dữ liệu do T1 ghi đã bị T2 làm mất
- ✧ Lỗi: MẤT DỮ LIỆU CẬP NHẬT (LOST UPDATE)



Vấn đề không thể đọc lại

- Xét 2 giao tác T1 và T2 và đơn vị dữ liệu A vốn có giá trị ban đầu là 50

★ Kết quả quan sát:

- ❖ Nếu T1 và T2 thực hiện tuần tự thì các lần đọc A của T2 giống nhau.
- ❖ Nếu T1 và T2 thực hiện đồng thời như lịch bên → 2 lần đọc dữ liệu của T2 có kết quả khác nhau

A=50	T1	T2	
t1	Read(A)		
t2		Read(A)	A=50
t3	A:=A-10		
t4		Print(A)	A=50
t5	Write(A)		
t6		Read(A)	A=40
t7		Print(A)	A=40

★ Lỗi không đọc lại được dữ liệu:

- ❖ Trong một giao tác mà các lần đọc cùng 1 đơn vị dữ liệu cho kết quả khác nhau



Vấn đề “bóng ma”

- Xét 2 giao tác T1 và T2 được xử lý đồng thời
 - A là một tập các đơn vị dữ liệu $a_1, a_2, a_3, a_4, \dots$
 - T1 xử lý trên toàn bộ tập A
 - Khi T1 đang xử lý, T2 thêm hay xóa một hay một số phần tử trong tập A

	T1	T2
t1	Read(A)	
t2	Xử lý 1 trên A	
t3		Thêm ai vào A
t4	Xử lý 2 trên A	
t5		Xoá aj khỏi A
t6	Xử lý 3 trên A	



Vấn đề đọc dữ liệu rác

- Xét 2 giao tác T1 và T2 được xử lý đồng thời
 - T2 đã đọc dữ liệu được ghi bởi T1 nhưng sau đó T1 yêu cầu hủy việc ghi

A=50	T1	T2
t1	Read(A)	
t2	A:=A+10	
t3	Write(A)	
t4		Read(A)
t5		Print(A)
t6	Abort	



Nhận xét

- Các lỗi truy xuất đồng thời của các giao tác T1, ..., Tn là do:
 - Kết quả của lịch tuần tự được lập từ các giao tác T1, ..., Tn và lịch đồng thời S từ các giao đó khác nhau: Không đảm bảo tính nhất quán dữ liệu.
 - Lịch đồng thời S không phải là một lịch khả tuần tự.
- Câu hỏi:
 - Làm sao bộ lập lịch có thể tạo được một lịch S khả tuần tự ?

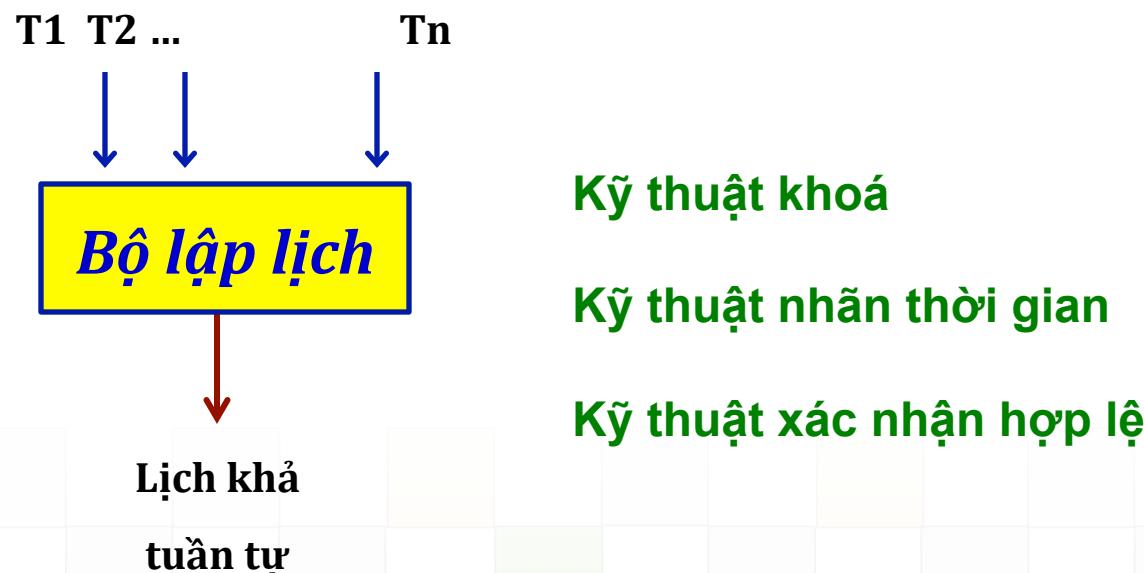


Các kỹ thuật điều khiển đồng thời



Các kỹ thuật điều khiển đồng thời

- Là những kỹ thuật cho phép bộ lập lịch sử dụng để tạo một lịch khả tuần tự từ n giao tác thực hiện đồng thời.





Nội dung trình bày

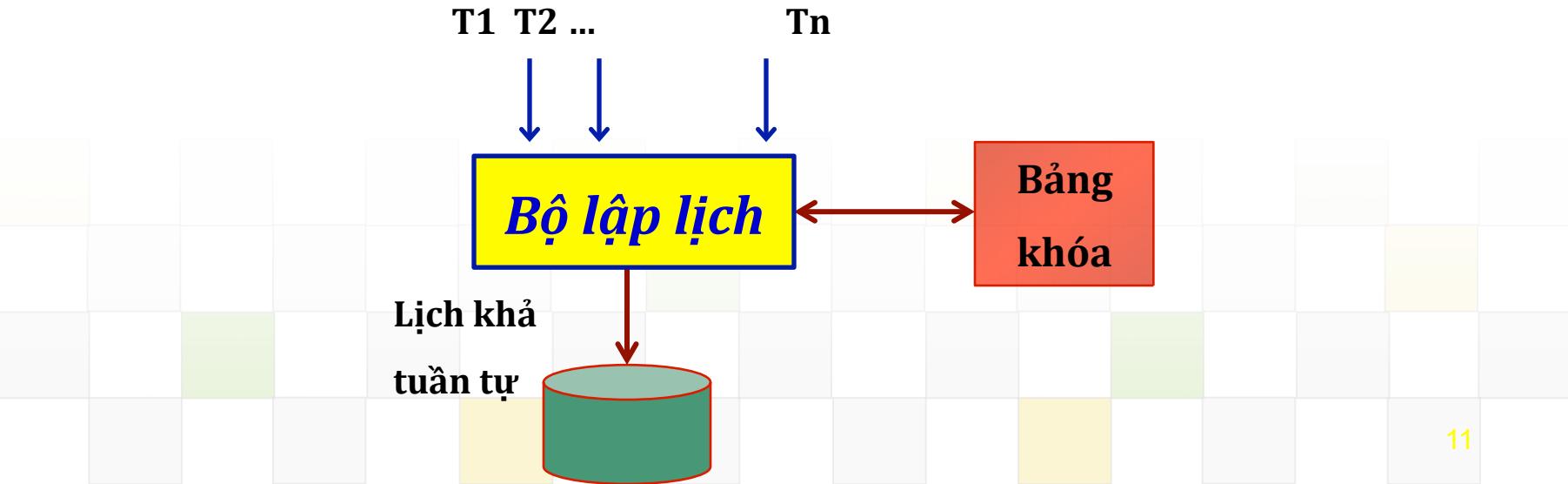
- Các vấn đề của truy xuất đồng thời
- **Các kỹ thuật điều khiển đồng thời:**
 - **Kỹ thuật khoá**
 - * Khoá đơn giản
 - * Khoá đọc ghi
 - * Khoá đa hạt
 - **Kỹ thuật nhãn thời gian**
 - * Nhãn thời gian toàn phần
 - * Nhãn thời gian riêng phần
 - * Nhãn thời gian nhiều phiên bản
 - **Kỹ thuật lạc quan**

- Vấn đề khoá chết
- Các vấn đề khác



Kỹ thuật khoá đơn giản

- Kỹ thuật khoá đơn giản còn gọi khoá nhị phân (Binary locks)
- Bộ lập lịch với cơ chế khoá đơn giản (locking scheduler)
 - Là bộ lập lịch với thêm **2** hành động:
 - * **Lock** : Phát khoá
 - * **Unlock** : Giải phóng khoá
 - Các khoá được ghi nhận trong bảng khoá (Lock Table)





Kỹ thuật khóa đơn giản

Quy định:

- Các giao tác trước khi muốn đọc/ghi lên 1 đơn vị dữ liệu phải phát ra 1 yêu cầu xin khóa (lock) đơn vị dữ liệu đó
 - * Ký hiệu: **Lock(A)** hay **l(A)**
- Yêu cầu này được bộ phận quản lý khóa xử lý (Lock Manager)
 - * Nếu yêu cầu được chấp thuận thì giao tác mới được phép đọc/ghi lên đơn vị dữ liệu
 - * Yêu cầu xin khóa được bộ cấp phát chấp thuận nếu đơn vị dữ liệu chưa bị khóa bởi một giao tác nào khác
- Sau khi thao tác xong thì giao tác phải phát ra lệnh giải phóng đơn vị dữ liệu (unlock)
 - * Ký hiệu: **Unlock(A)** hay **u(A)**

Bảng khóa ghi nhận giao
tác T1 đang giữ khóa
trên đơn vị dữ liệu A

BẢNG KHÓA

Element	Transaction
A	T1

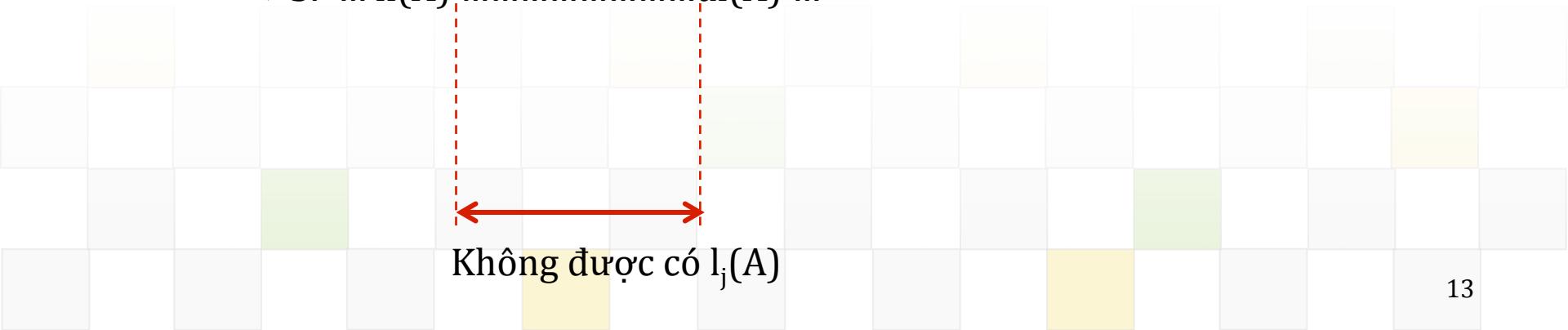


Kỹ thuật khóa đơn giản (tt)

■ Quy tắc:

- **1. Giao tác đúng đắn:** Việc giao tác Ti đọc hay ghi lên đơn vị dữ liệu A phải sau khi Ti phát khoá trên A và trước khi Ti giải phóng khoá trên A. Phát khoá và giải phóng khoá phải đi đôi với nhau (lock trước, unlock sau)
 - * Ti: ... l(A) ... r(A) / w(A) ... u(A) ...
- **2. Lịch thao tác hợp lệ:** Khi Ti đang giữ khoá trên một đơn vị dữ liệu A thì không Ti nào khác được phát khoá trên A.

* S: ... li(A) ui(A) ...



Không được có $l_j(A)$



Kỹ thuật khóa đơn giản (tt)

- Ví dụ:
 - Giao tác T1 và T2 có đúng đắn hay không ?
 - Lịch S có hợp lệ hay không ?

S	T1	T2
	Lock(A) Read(A, t) $t := t + 100$ Write(A, t) Unlock(A)	Lock(A) Read(A, s) $s := s * 2$ Write(A, s) Unlock(A)
	Lock(B) Read(B, t) $t := t + 100$ Write(B, t) Unlock(B)	Lock(B) Read(B, s) $s := s * 2$ Write(B, s) Unlock(B)



Kỹ thuật khóa đơn giản (tt)

■ Ví dụ

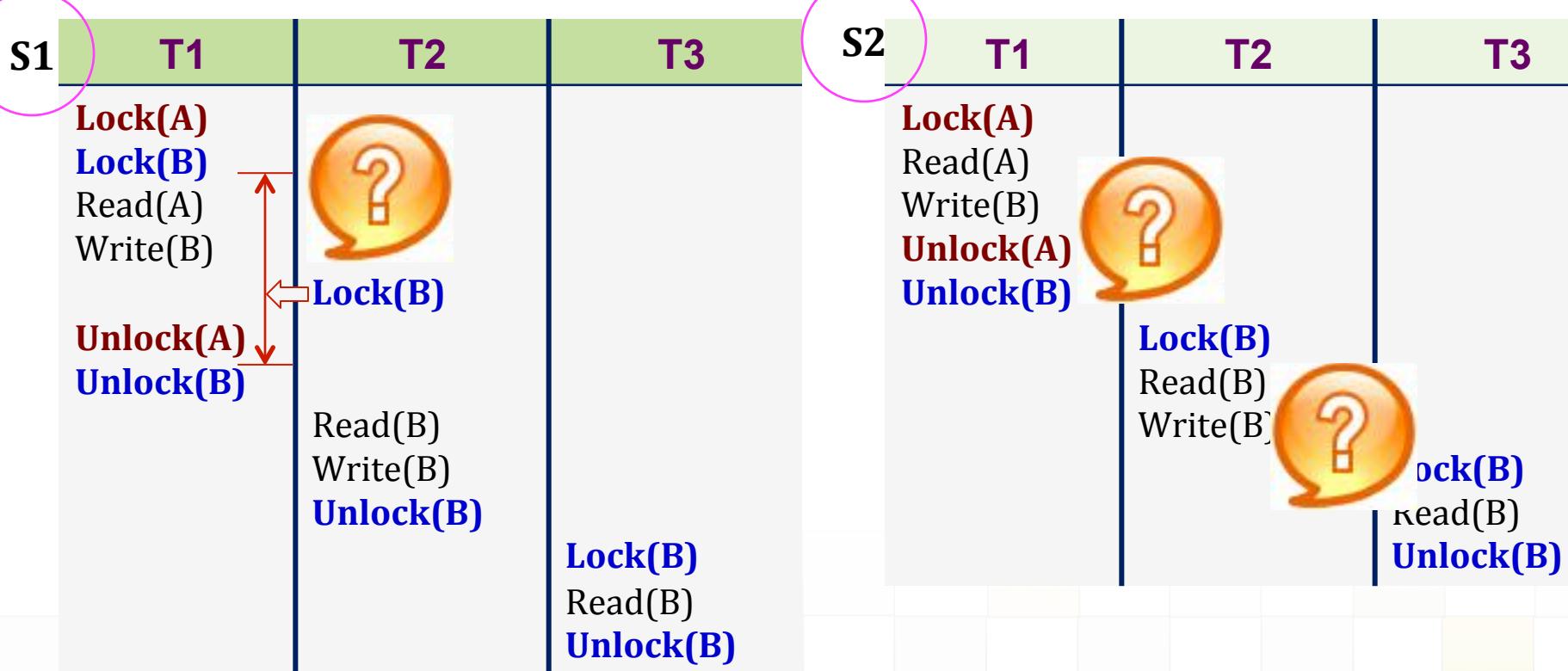
S1	T1	T2	T3	S2	T1	T2	T3
	Lock(A) Lock(B) Read(A) Write(B)				Lock(A) Read(A) Write(B) Unlock(A) Unlock(B)		
		Lock(B)				Lock(B) Read(B) Write(B)	
			Read(B) Write(B) Unlock(B)				Lock(B) Read(B) Unlock(B)
				Lock(B) Read(B) Unlock(B)			

Cho biết lịch nào hợp lệ? Giao tác nào là đúng đắn?



Kỹ thuật khóa đơn giản (tt)

- Ví dụ:



Cho biết lịch nào hợp lệ? Giao tác nào là đúng đắn ?



Kỹ thuật khóa đơn giản (tt)

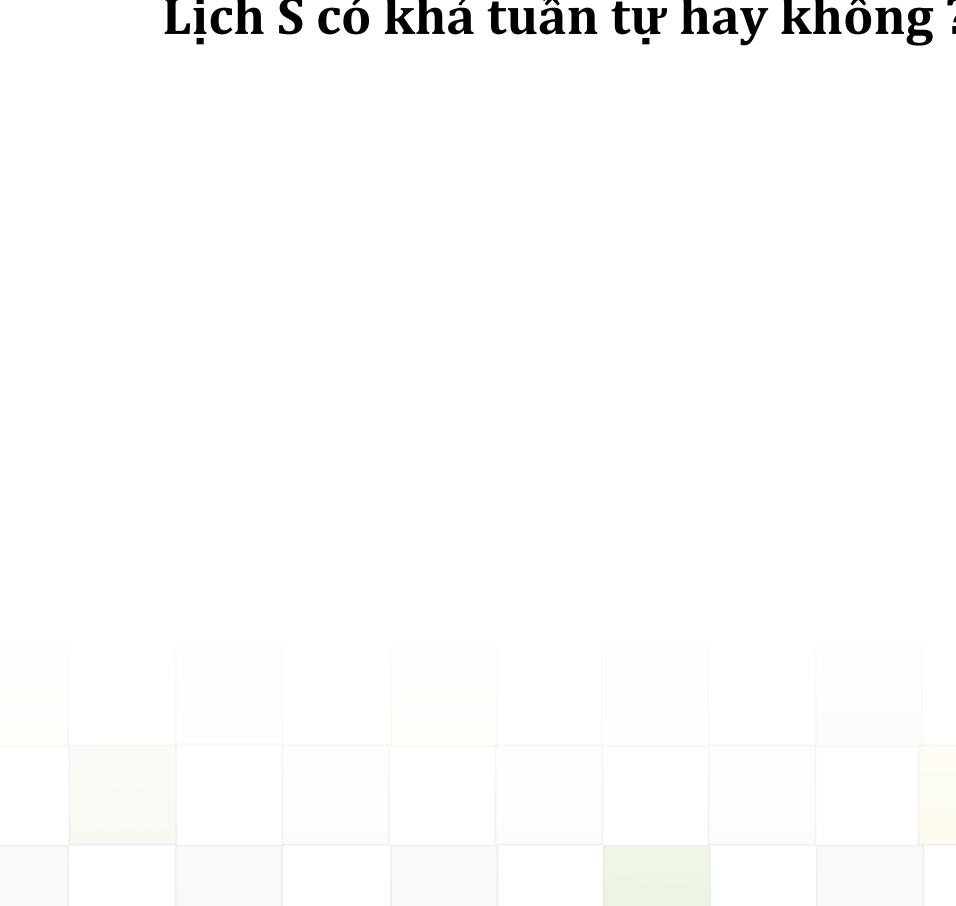
- **Bài toán:** Kiểm tra tính khả tuần tự của lịch S
 - Input: Lịch **S** được lập từ n giao tác xử lý đồng thời T_1, T_2, \dots, T_n theo kỹ thuật khóa đơn giản
 - Output: **S** khả tuần tự hay không?

- **Phương pháp:** Xây dựng 1 đồ thị có hướng G
 - B1: Mỗi giao tác T_i là 1 đỉnh của đồ thị
 - B2: Nếu một giao tác T_j phát ra $\text{Lock}_j(A)$ sau một giao tác T_i phát ra $\text{Unlock}_i(A)$ thì sẽ vẽ cung từ T_i đến T_j , $i \neq j$
 - B3: S khả tuần tự nếu G không có chu trình

Kỹ thuật khóa đơn giản (tt)

S

S

T1	T2	Lịch S có khả tuần tự hay không ?
Lock(A) Read(A, t) $t := t + 100$ Write(A, t) Unlock(A)	Lock(A) Read(A, s) $s := s * 2$ Write(A, s) Unlock(A) Lock(B) Read(B, s) $s := s * 2$ Write(B, s) Unlock(B)	
Lock(B) Read(B, t) $t := t + 100$ Write(B, t) Unlock(B)		

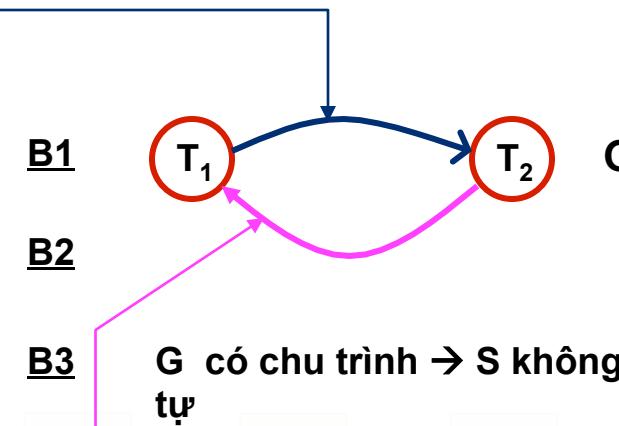
Lịch S có khả tuần tự hay không ?

Kỹ thuật khóa đơn giản (tt)

S

	T1	T2
	Lock(A) Read(A, t) $t := t + 100$ Write(A, t) Unlock(A)	
	Lock(A) Read(A, s) $s := s * 2$ Write(A, s) Unlock(A) Lock(B) Read(B, s) $s := s * 2$ Write(B, s) Unlock(B)	

Lịch S có khả tuần tự hay không ?

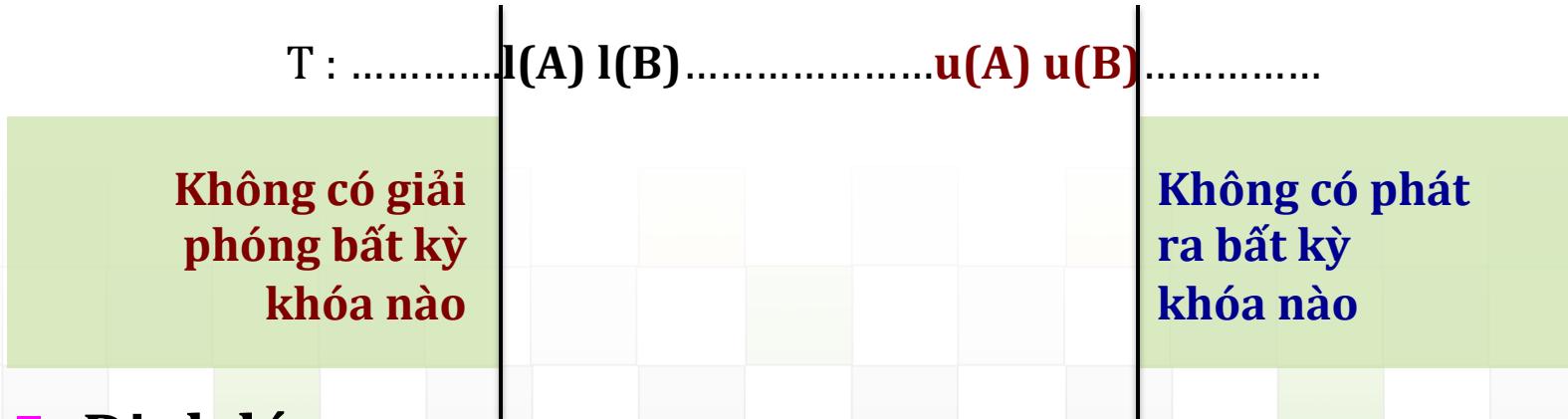


G có chu trình \rightarrow S không khả tuần tự



Kỹ thuật khóa đơn giản (tt)

- **Mục tiêu:** Vậy làm sao để kỹ thuật khóa cho ta lịch khả tuẫn tự
- **Giải pháp :** Tuân theo quy tắc sau:
 - (1) và (2): Giống như cũ
 - (3) Giao tác phải thỏa **nghi thức khóa 2 giai đoạn (2PL: two phases locking)** : Trong 1 giao tác, tất cả các thao tác phát khóa đều xảy ra trước tất cả các thao tác giải phóng khóa.



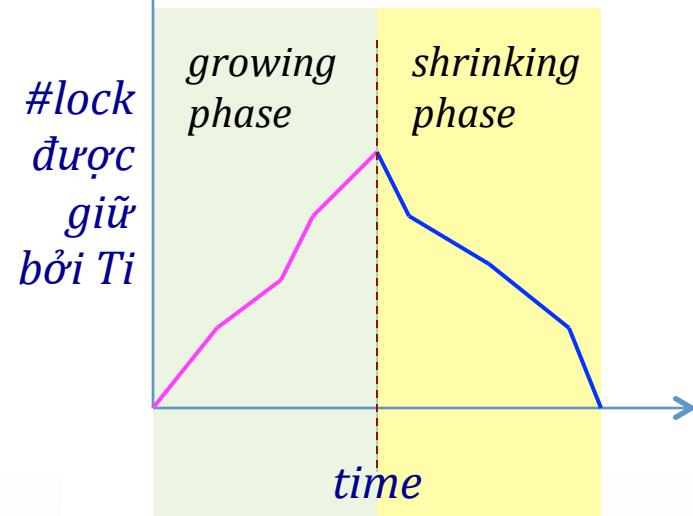
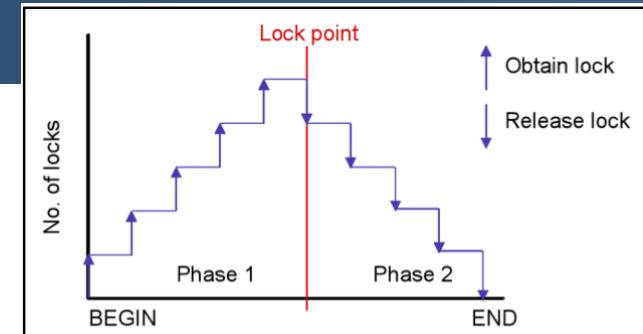
- **Định lý:**

- Nếu lịch S thoả nghi thức 2PL thì S conflict-serializable



Nghi thức khoá 2 giai đoạn

- Nghi thức khoá 2 giai đoạn chia việc xin khoá và giải phóng khoá của giao tác thành 2 giai đoạn (phase) phân biệt:
 - **Growing phase (pha phát khoá):**
 - ★ Giao tác chỉ được phép xin khoá chứ không được phép giải phóng khoá trong pha này (số lượng khoá được giữ chỉ được phép ngày càng tăng)
 - ★ Giai đoạn này kết thúc ở lệnh xin khoá cuối cùng.
 - **Shrinking phase (pha giải phóng khoá):**
 - ★ Giao tác chỉ được phép giải phóng khoá chứ không được phép xin khoá trong pha này
 - ★ Giao tác này bắt đầu từ khi lệnh giải phóng khoá đầu tiên.





Tại sao lại cần nghi thức khoá 2 giai đoạn ?

Problem with Serializability

- Definition: "Equivalent to some serial schedule"
- Calculation of Equivalence takes too long
- Example: 10 transactions in schedule
 - How many serial schedules?
 - $10 \times 9 \times 8 \times \dots \times 1 = 10! = 3,628\ 800$

Solution

- Every transaction follows a protocol
 - protocol = rules of behavior
- Protocol guarantees serializable schedule



Kỹ thuật khóa đơn giản (tt)

T1	T2	T3	T4
Lock(A)	Lock(B)	Lock(B)	Lock(A)
Read(A)	Read(B)	Read(B)	Read(A)
Lock (B)	Lock(A)	B=B-50	Unlock(A)
Read(B)	Read(A)	Write(B)	Lock(B)
B:= B + A	Unlock(B)	Unlock(B)	Read(B)
Write(B)	A:= A+B	Lock(A)	Unlock(B)
Unlock(A)	Write(A)	Read(A)	Print (A+B)
Unlock(B)	Unlock(A)	A:=A+50	
		Write(A)	
		Unlock(A)	

*Giao tác nào thoả nghi thức
2 giai đoạn ?*



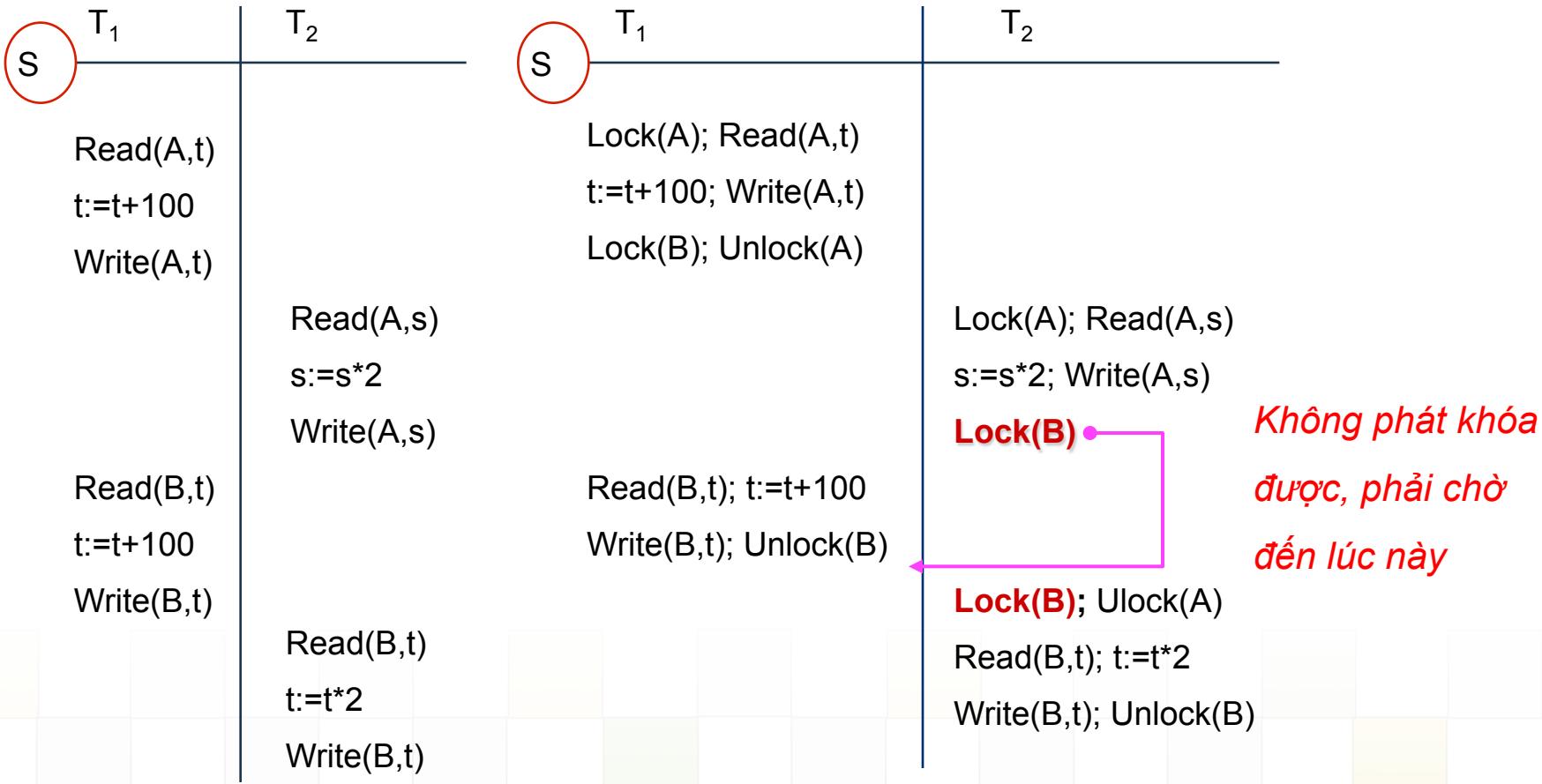
Kỹ thuật khóa đơn giản (tt)

T1	T2	T3	T4
Lock(A)	Lock(B)	Lock(B)	Lock(A)
Read(A)	Read(B)	Read(B)	Read(A)
Lock (B)	Lock(A)	B=B-50	Unlock(A)
Read(B)	Read(A)	Write(B)	Lock(B)
B:= B + A	Unlock(B)	Unlock(B)	Read(B)
Write(B)	A:= A+B	Lock(A)	Unlock(B)
Unlock(A)	Write(A)	Read(A)	Print (A+B)
Unlock(B)	Unlock(A)	A:=A+50	
		Write(A)	
		Unlock(A)	

Thỏa nghi thức khóa 2 giai đoạn

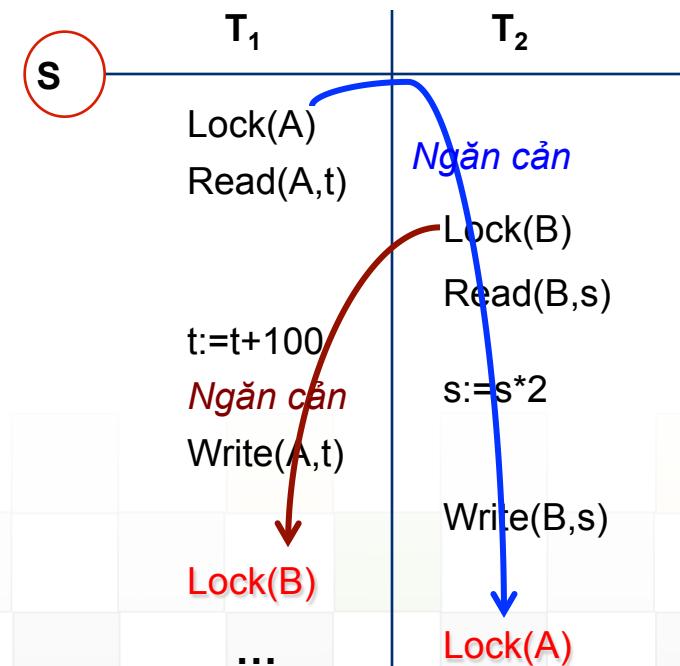
Không thỏa nghi thức khóa 2 giai đoạn

Kỹ thuật khóa đơn giản (tt)



Kỹ thuật khóa đơn giản (tt)

- **Vấn đề khoá chết (Dead lock):** Hiện tượng chờ khi cần phát khóa có thể dẫn đến chờ lẫn nhau **vĩnh viễn**





Kỹ thuật khoá đơn giản (tt)

- Ngăn ngừa khoá chết:
 - Mọi giao tác phải xin khoá tất cả các đơn vị dữ liệu mà mình sẽ thao tác. Nếu được chấp thuận tất cả thì sẽ thao tác ngược lại thì phải giải giải phóng tất cả khoá được cấp trên các đơn vị dữ liệu.
- Phát hiện khoá chết:
 - Xây dựng đồ thị chờ (Waiting graph):
 - ★ Mỗi giao tác T_i là một node của đồ thị G
 - ★ Nếu có giao tác T_j phát ra yêu cầu khoá đơn vị dữ liệu A đã bị khoá trước đó bởi T_i , thì vẽ cung có hướng từ $T_i \rightarrow T_j$ (Biểu diễn việc T_j phải chờ T_i)
 - ★ Nếu G xuất hiện chu trình \rightarrow Xảy ra tình trạng khoá chết



Nội dung trình bày

- Các vấn đề của truy xuất đồng thời
- **Các kỹ thuật điều khiển đồng thời:**
 - **Kỹ thuật khoá**
 - * Khoá đơn giản
 - * **Khoá đọc ghi**
 - * Khoá đa hạt
 - **Kỹ thuật nhãn thời gian**
 - * Nhãn thời gian toàn phần
 - * Nhãn thời gian riêng phần
 - * Nhãn thời gian nhiều phiên bản
 - **Kỹ thuật lạc quan**

- Vấn đề khoá chết
- Các vấn đề khác



Kỹ thuật khóa đọc ghi

- Vấn đề tồn tại của 2PL theo kỹ thuật khoá đơn giản:
 - Có những tình huống mà T_i không thực sự cần phải ngăn cản một T_j truy xuất đơn vị dữ liệu của nó, nhưng theo 2PL vẫn phải ngăn cản
 - Không tối ưu về mặt tốc độ vì có những khoảng chờ không cần thiết, thậm chí gây nên deadlock
- Do đó, bộ lập lịch cần các hành động:
 - **Khóa đọc** (Read lock, Shared lock)
 - * Ký hiệu : **RLock(A)** hay **rl(A)**
 - **Khóa ghi** (Write lock, Exclusive lock)
 - * Ký hiệu : **WLock(A)** hay **wl(A)**
 - **Giải phóng khóa**
 - * Ký hiệu : **Unlock(A)** hay **u(A)**



Kỹ thuật khóa đọc ghi (tt)

- Cho 1 đơn vị dữ liệu A bất kỳ
 - **WLock(A)**
 - * Hoặc có 1 khóa ghi duy nhất lên A
 - * Hoặc không có khóa ghi nào lên A
 - **RLock(A)**
 - * Có thể có nhiều khóa đọc được thiết lập lên A

- Ma trận tương thích:

Yêu cầu khoá

	RLock(A)	WLock(A)
RLock(A)	✓	✗
WLock(A)	✗	✗

✓ *Tương thích* ✗ *Không tương thích*



Kỹ thuật khóa đọc ghi (tt)

- Giao tác T muốn Write(A):
 - Yêu cầu WLock(A)
 - * WLock(A) sẽ được chấp thuận nếu hiện không có khóa nào trên A
 - * Từ đó sẽ không có giao tác nào khác nhận được WLock(A) hay RLock(A) cho đến khi T giải phóng khóa trên A
- Giao tác muốn Read(A):
 - Yêu cầu RLock(A) hoặc WLock(A)
 - * RLock(A) sẽ được chấp thuận nếu A **không** đang giữ một WLock nào
 - * Từ đó sẽ không có giao tác nào khác nhận được WLock(A) cho đến khi T giải phóng khóa trên A. Nhưng không ngăn chặn các thao tác khác cùng xin Rlock(A) nên các giao tác không cần phải chờ nhau khi đọc A
- Sau khi thao tác xong thì giao tác phải giải phóng khóa trên đơn vị dữ liệu A



Kỹ thuật khóa đọc ghi (tt)

■ Qui tắc

- (1) *Giao tác đúng đắn* :

- ★ Đã có phát khóa thì sau đó phải có giải phóng khóa, giải phóng khóa chỉ có khi trước đó có phát khóa mà chưa giải phóng
- ★ Thao tác đọc chỉ được thực hiện sau khi phát khóa đọc hoặc ghi và trước khi giải phóng khóa ấy
- ★ Thao tác ghi chỉ được thực hiện sau khi phát khóa ghi và trước khi giải phóng khóa ghi ấy
- ★ Các thao tác đọc, ghi, phát khóa và giải phóng khóa đề cập trên đây là xét trong cùng một giao tác và trên cùng 1 đơn vị dữ liệu



Kỹ thuật khóa đọc ghi (tt)

■ Qui tắc

- (2) - *Lịch thao tác hợp lệ*

- * Khi T_i đang giữ khóa đọc trên 1 đơn vị Dữ liệu A thì **không** một T_j nào khác được phép ghi trên A
- * Khi T_i đang giữ khóa ghi trên 1 đơn vị Dữ liệu A thì **không** một T_j nào khác được phép đọc hay ghi trên A



Kỹ thuật khóa đọc ghi (tt)

■ Qui tắc

- (3) - Giao tác 2PL

* Ngoại trừ trường hợp nâng cấp khóa, các trường hợp còn lại đều giống với nghi thức khóa hai giai đoạn

* T : ... rli(A) ... | wli(A) ui(A) ...

**Không có giải
phóng bất kỳ
khóa nào**

**Không có phát
ra bất kỳ khóa
nào**

* Trường hợp nâng cấp khóa được giải phóng khóa đọc trong pha phát khóa

* T : ... rli(A) | uli(A) | wli(A) ui(A) ...

**Chấp nhận giải phóng
khóa đọc khi nâng cấp
khóa**

■ Định lý :

- S thoả (1), (2) và (3) → S conflict-serializable

Ví dụ

S1	T ₁	T ₂
	RLock(A)	
	Read(A)	
	U(A)	
		RLock(B)
		Read(B)
		U(B)
		WLock(A)
		Read(A)
	A:=A+B	
	Write(A)	
	U(A)	
	WLock(B)	
	Read(B)	
	B:=B+A	
	Write(B)	
	U(B)	

1. Giao tác nào đúng đắn ?
2. Lịch thao tác có hợp lệ hay không ?
3. Giao tác nào không thỏa nghi thức 2PL ?
4. S có khả tuần tự ?

Ví dụ

S1	T ₁	T ₂
	RLock(A)	
	Read(A)	
	U(A)	
		RLock(B)
		Read(B)
		U(B)
		WLock(A)
		Read(A)
		A:=A+B
		Write(A)
		U(A)
	WLock(B)	
	Read(B)	
	B:=B+A	
	Write(B)	
		U(B)

1. Giao tác nào đúng đắn ?
2. Lịch thao tác có hợp lệ hay không ?
3. Giao tác nào không thỏa nghi thức 2PL ?
4. S có khả tuần tự ?



Ví dụ

S2	T ₁	T ₂	T ₃	T ₄
		RL(A)	RL(A)	
		WL(B) U(A)	WL(A)	
	RL(B)	U(B)	U(A)	RL(B)
	RL(A)		U(B)	
	WL(C) U(A)			WL(B) U(B)
	U(B) U(C)			

1. Giao tác nào đúng đắn ?
 2. Lịch thao tác có hợp lệ hay không ?
 3. Giao tác nào không thỏa nghi thức 2PL ?
 4. S có khả tuần tự ?



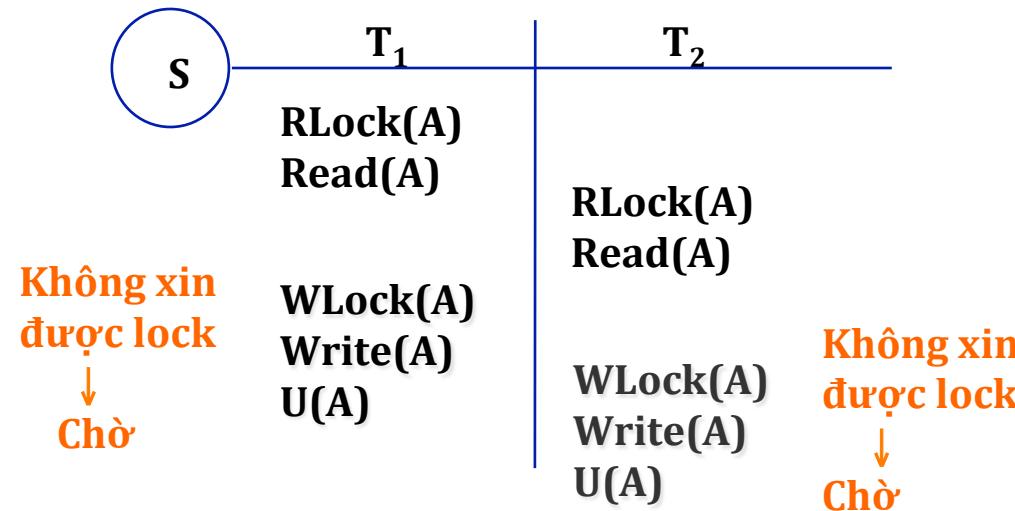
Ví dụ

S2	T ₁	T ₂	T ₃	T ₄
		RL(A)	RL(A)	
		WL(B) U(A)	WL(A)	
	RL(B)	U(B)	U(A)	RL(B)
	RL(A)		U(B)	
	WL(C) U(A)		WL(B) U(B)	
	U(B) U(C)			

1. Giao tác nào đúng đắn ?
 2. Lịch thao tác có hợp lệ hay không ?
 3. Giao tác nào không thỏa nghi thức 2PL ?
 4. S có khả tuần tự ?

Kỹ thuật khóa đọc ghi (tt)

- Vấn đề nâng cấp khoá của nhiều giao tác có thể dẫn đến deadlock



Nâng cấp khoá: Giao tác đầu tiên đọc dữ liệu và sau đó cũng ghi lên đơn vị dữ liệu đó.



Kỹ thuật khóa đọc ghi (tt)

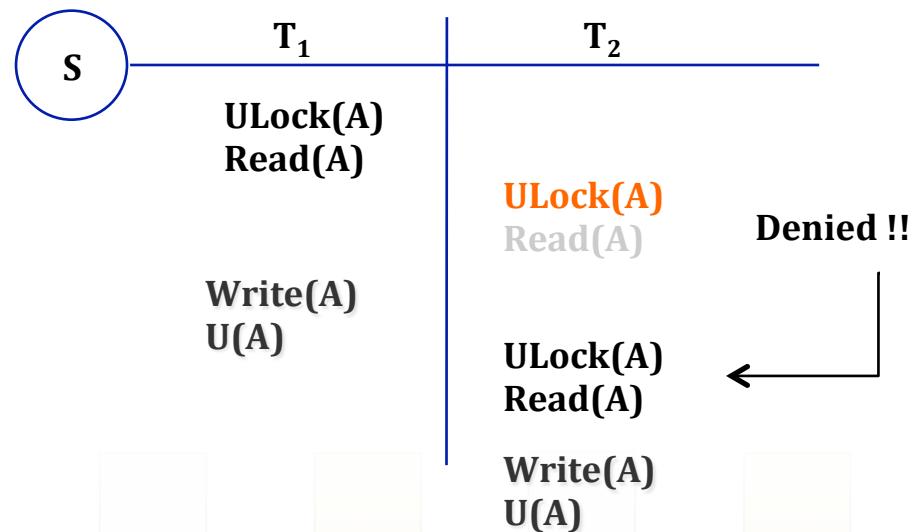
- Khóa cập nhật (update lock)
 - ULock(A)
 - Giao tác muốn read(A) và sau đó cũng muốn write(A)
 - * Yêu cầu ULock(A)
 - * ULock(A) được chấp thuận khi A tự do hoặc đang giữ RLock
 - Khi đó, không có giao tác nào nhận được RLock(A), WLock(A) hay ULock(A)
 - Ma trận tương thích

		Yêu cầu khoá		
		RLock	WLock	ULock
Trạng thái hiện hành	RLock	✓	✗	✓
	WLock	✗	✗	✗
	ULock	✗	✗	✗



Kỹ thuật khóa đọc ghi (tt)

- Sử dụng khoá cập nhật thể tránh được deadlock





Nội dung trình bày

- Các vấn đề của truy xuất đồng thời
- **Các kỹ thuật điều khiển đồng thời:**
 - **Kỹ thuật khoá**
 - * Khoá đơn giản
 - * Khoá đọc ghi
 - * Khoá đa hạt**
 - **Kỹ thuật nhãn thời gian**
 - * Nhãn thời gian toàn phần
 - * Nhãn thời gian riêng phần
 - * Nhãn thời gian nhiều phiên bản
 - **Kỹ thuật lạc quan**
- Vấn đề khoá chết
- Các vấn đề khác



Kỹ thuật khóa đa hạt

- Xét ví dụ hệ thống ngân hàng
 - Quan hệ TàiKhoản(mãTK, sốDư)
 - Giao tác gửi tiền và rút tiền
 - ★ Khóa relation
 - Các giao tác thay đổi giá trị của số dư của 1 tài khoản X sẽ yêu cầu khóa độc quyền
 - Vì khóa ở mức độ quan hệ nên toàn bộ bảng bị khóa
 - Các giao tác khác muốn truy cập tài khoản Y (Y khác X) cũng phải chờ → vô lý, tốc độ xử lý đồng thời chậm
 - ★ Khóa tuple hay disk block
 - 2 tài khoản ở 2 blocks khác nhau có thể được cập nhật cùng thời điểm
 - Xử lý đồng thời nhanh
 - Giao tác tính tổng số tiền của các tài khoản
 - ★ Khóa relation?
 - ★ Khóa tuple hay disk block?



Kỹ thuật khóa đa hạt (tt)

- Phải quản lý khóa ở nhiều mức độ
 - Tính chất hạt (granularity) : Tính hạt càng tăng khi đơn vị dữ liệu bị khóa càng lớn
 - Tính đồng thời (concurrency) : Tính đồng thời càng tăng khi đơn vị dữ liệu bị khóa càng nhỏ
 - Tính hạt tăng thì tính đồng thời giảm và ngược lại → phải thỏa hiệp

Tính hạt tăng

Quan hệ (Relation)

Khối (Block)

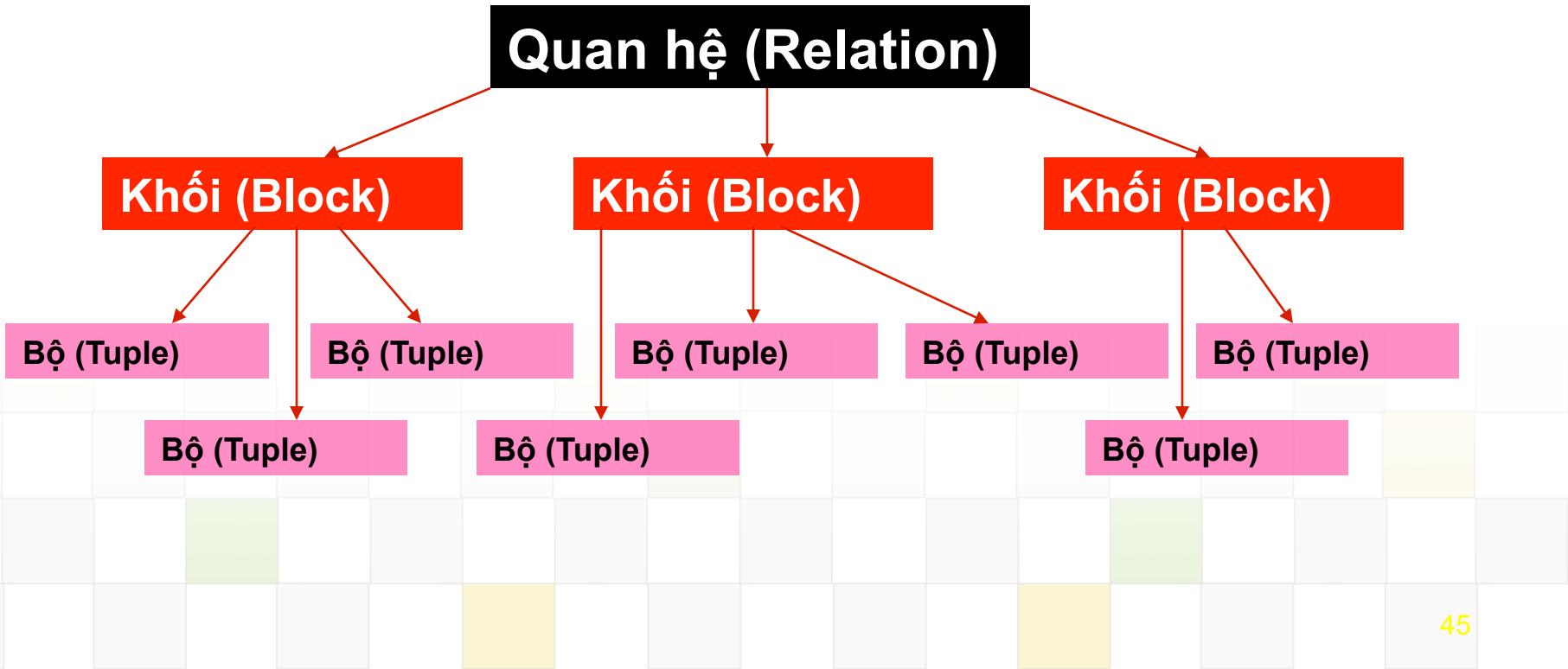
Bộ (Tuple)

Tính đồng thời tăng



Kỹ thuật khóa đa hạt (tt)

- Phân cấp Dữ liệu
 - Relations là đơn vị dữ liệu khóa lớn nhất
 - Một relation gồm 1 hoặc nhiều blocks (pages)
 - Một block gồm 1 hoặc nhiều tuples





Kỹ thuật khóa đa hạch (tt)

- Gồm các khóa

- Khóa thông thường
 - * Shared lock: S
 - * Exclusive lock: X
 - Khóa cảnh báo (warning lock)
 - * Warning (intention to) shared lock: IS
 - * Warning (intention to) exclusive lock: IX



Kỹ thuật khóa đa hạt (tt)

- Ma trận tương thích trên cùng một node
 - Cho biết các khóa có thể cùng tồn tại trên 1 node dữ liệu

	IS	IX	S	X
IS	✓	✓	✓	✗
IX	✓	✓	✗	✗
S	✓	✗	✓	✗
X	✗	✗	✗	✗

Kỹ thuật khóa đa hạt (tt)

- Ma trận tương thích giữa node cha và node con
 - Cho biết các khóa có thể phát trên node con khi node cha đang có một khoá nào đó → Cho biết muốn phát 1 khoá trên node con thì trước đó phải phát khoá nào trên node cha

<i>Node cha đã khoá bằng phương thức</i>	<i>Node con có thể khoá bằng các phương thức</i>
IS	IS, S
IX	IS, S, IX, X
S	S, IS
X	Không có



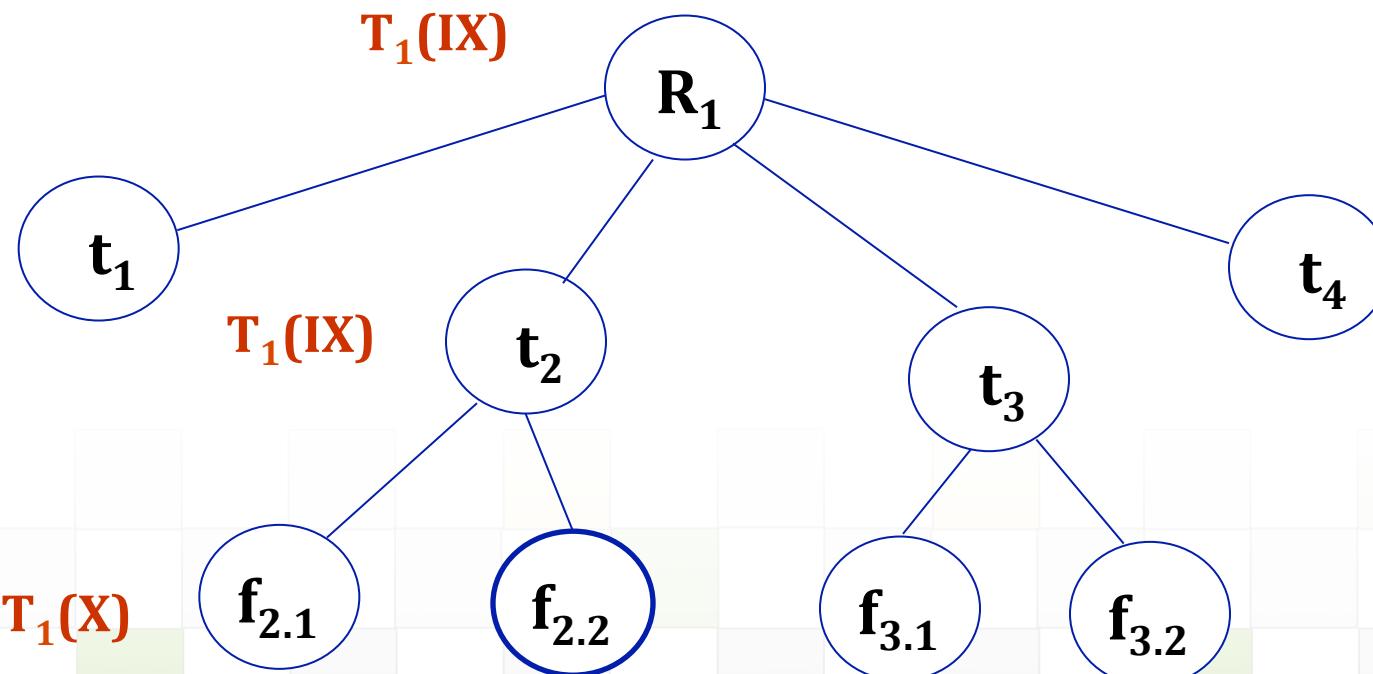
Kỹ thuật khóa đa hạt (tt)

Quy tắc:

- (1) Thỏa ma trận tương thích trên cùng một node
- (2) Khóa nút gốc của cây trước
- (3) Thỏa ma trận tương thích giữa node cha và node con
- (4) T_i thỏa 2PL
- (5) T_i có thể giải phóng nút Q khi không có nút con nào của Q bị khóa bởi T_i
- (6) Trong trường hợp thêm mới hay xóa bỏ một node Q thì cha của Q phải được khóa bằng X trước → tránh Phantom

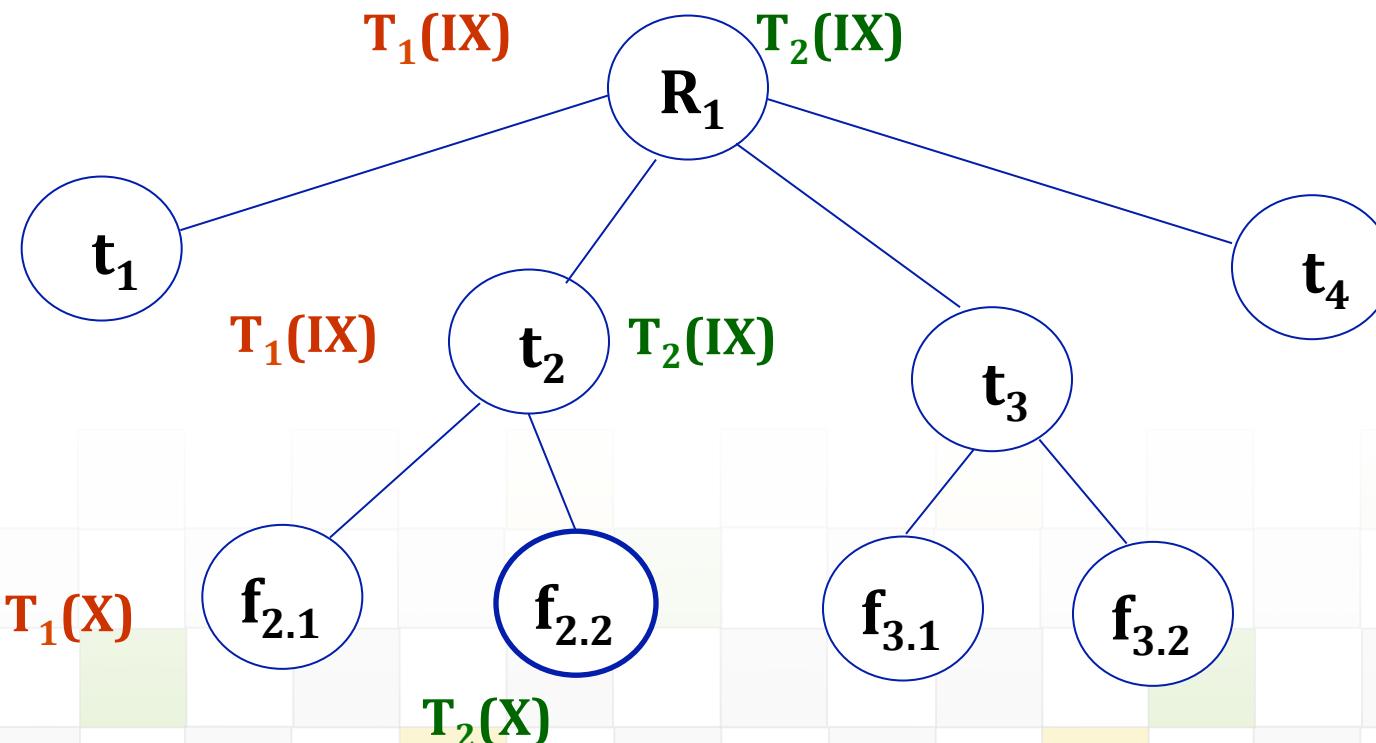
Bài tập

- T2 có thể truy xuất f2.2 bằng khóa X được không?
- T2 sẽ có những khóa gì?

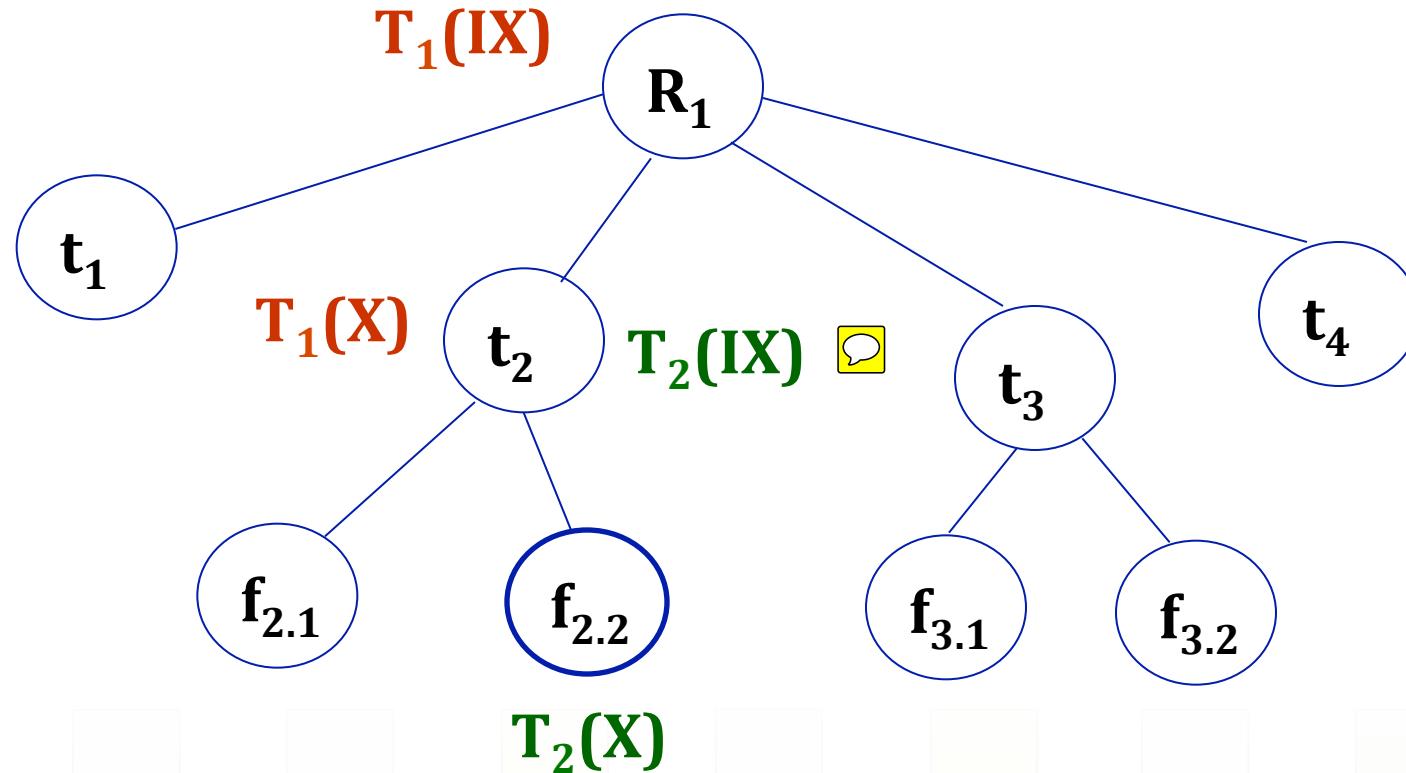


Bài tập

- T2 có thể truy xuất $f_{2.2}$ bằng khóa X được không?
- T2 sẽ có những khóa gì?

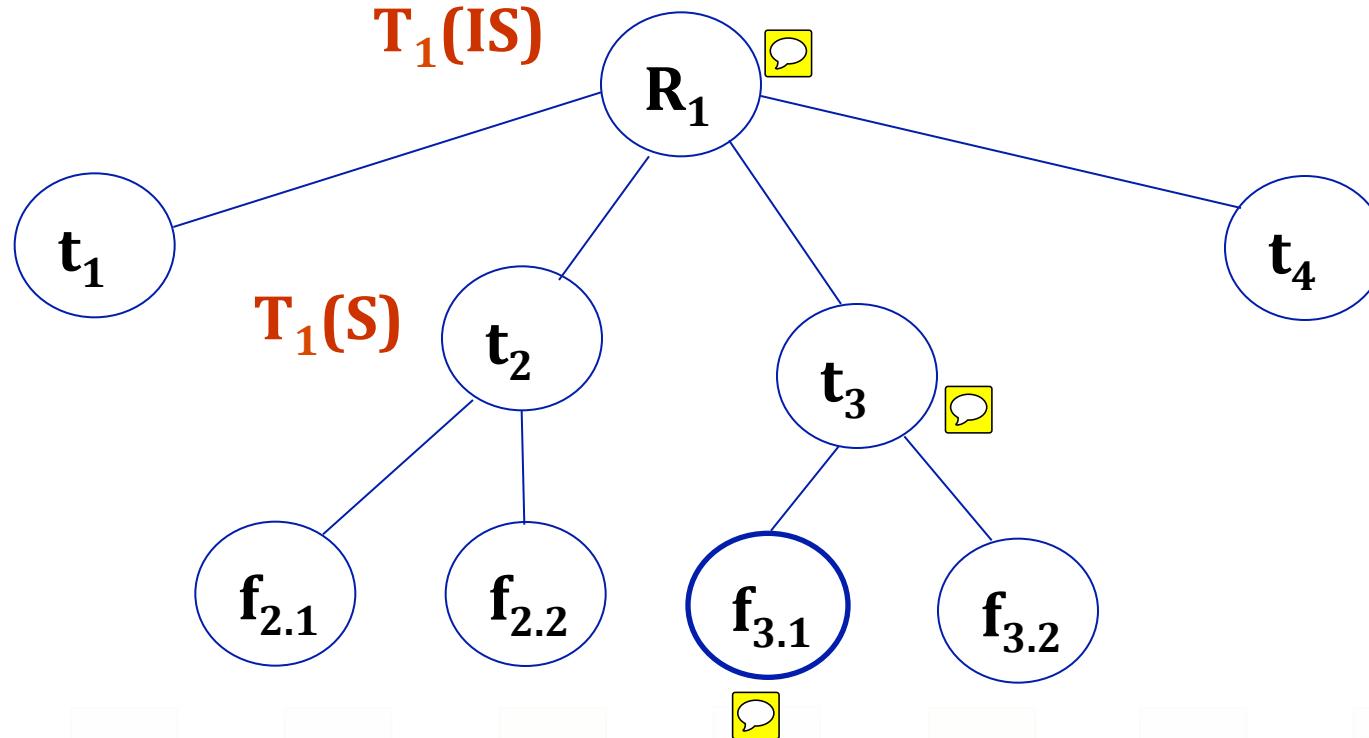


Bài tập (tt)



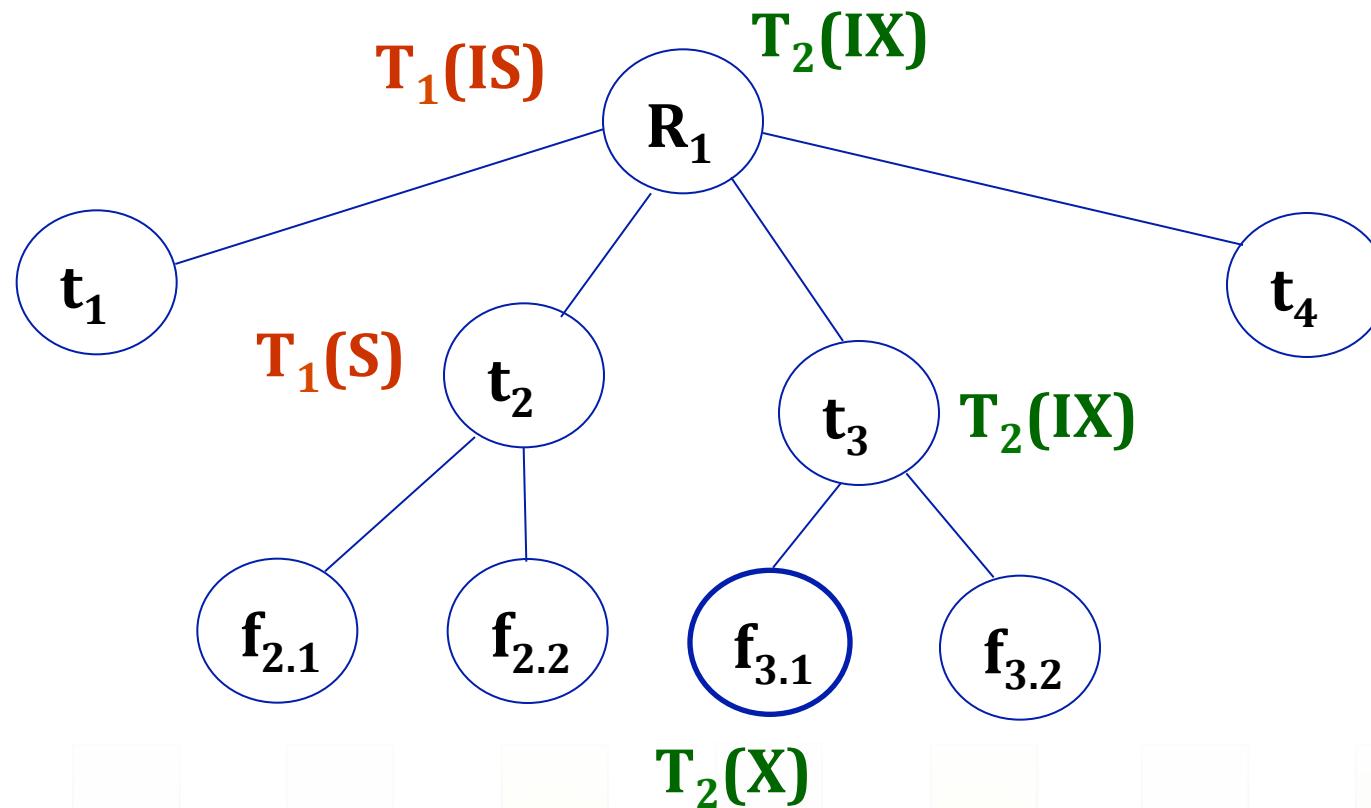
- T2 có thể truy xuất $f_{2.2}$ bằng khóa X được không?
- T2 sẽ có những khóa gì?

Bài tập (tt)



- T2 có thể truy xuất $f_{3.1}$ bằng khóa X được không?
- T2 sẽ có những khóa gì?

Bài tập (tt)



- T2 có thể truy xuất $f_{3.1}$ bằng khóa X được không?
- T2 sẽ có những khóa gì?

Kỹ thuật khóa đa hạch (tt)

■ Vấn đề

Tài Khoản

MãTK	SốDư	MãChiNhánh
A-101	500	Mianus
A-215	700	Mianus
A-102	400	Perryride
A-201	900	Mianus

T₃(IS)
Tài Khoản

T₂(IX)

Mianus

T₃(S)

Mianus

T₃((S))

Mianus

Perryride

T₂(X)

T₃ có còn đúng không?

T₁

```
select * from  
TaiKhoan where  
maCN="Mianus"
```

T₂

```
update TaiKhoan set  
soDu=400 where  
maCN="Perryride"
```

T₃

```
select sum(soDu)  
from TaiKhoan  
where maCN="Minanus"
```

T₄

```
insert TaiKhoan  
valuse(A-201, 900,  
Mianus )
```



Nội dung trình bày

- Các vấn đề của truy xuất đồng thời
- **Các kỹ thuật điều khiển đồng thời:**
 - **Kỹ thuật khoá** 
 - * Khoá đơn giản
 - * Khoá đọc ghi
 - * Khoá đa hạt
 - **Kỹ thuật nhãn thời gian** 
 - * **Nhãn thời gian toàn phần**
 - * Nhãn thời gian riêng phần
 - * Nhãn thời gian nhiều phiên bản
 - **Kỹ thuật xác nhận hợp lệ**

- Vấn đề khoá chết
- Các vấn đề khác

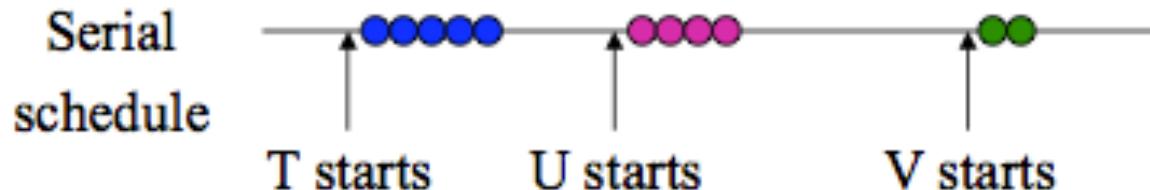


Giới thiệu

- Ý tưởng:
 - Mặc định cho rằng tất cả mọi thao tác của các giao tác dù diễn ra đồng thời cũng không gây mất nhất quán dữ liệu
 - Nếu lỡ có thao tác mang nguy cơ mất nhất quán dữ liệu → Hủy giao tác đó và chạy lại sau
- Chọn một thứ tự thực hiện nào đó cho các giao tác bằng cách gán nhãn thời gian (timestamping)
 - Mỗi giao tác T sẽ được bộ lập lịch gán 1 nhãn thời gian, ký hiệu $TS(T)$, nhãn này gán ngay lúc T bắt đầu bằng 1 trong 2 cách :
 - * Đồng hồ máy tính
 - * Bộ lập lịch tự đếm
 - Thứ tự của các nhãn tăng dần, T_i trễ hơn T_j thì $TS(T_i) > TS(T_j)$

Giới thiệu

- Chiến lược cơ bản:
 - Nếu $TS(T_i) < TS(T_j)$ thì lịch thao tác được phát sinh phải tương đương với lịch biểu tuần tự $\{T_i, T_j\}$





Nhãn thời gian toàn phần

- Mỗi giao tác T khi phát sinh sẽ được gán 1 nhãn TS(T) ghi nhận lại thời điểm phát sinh của T
- Mỗi đơn vị dữ liệu X cũng có 1 nhãn thời TS(X), nhãn này ghi lại TS(T) của giao tác T đã thao tác read/write thành công sau cùng lên X
- Khi đến lượt giao tác T thao tác trên dữ liệu X, so sánh TS(T) và TS(X)
 - **Nếu T muốn đọc X:**
 - * Nếu $TS(T) \geq TS(X)$ thì cho T đọc X và gán $TS(X) = TS(T)$
 - * Ngược lại T bị hủy (abort)
 - **Nếu T muốn ghi X:**
 - * Nếu $TS(T) \geq TS(X)$ thì cho T ghi X và gán $TS(X) = TS(T)$
 - * Ngược lại T bị hủy (abort)



Nhân thời gian toàn phần

Read(T, X)

```
If TS(X) <= TS(T)
    Read(X); //cho phép đọc X
    TS(X):= TS(T);
Else
    Abort {T}; //hủy bỏ T
```

Write(T, X)

```
If TS(X) <= TS(T)
    Write(X); //cho phép ghi
    TS(X):= TS(T);
Else
    Abort {T}; //hủy bỏ T
```



Ví dụ

T_1 $TS(T_1)=100$	T_2 $TS(T_2)=200$	A $TS(A)=0$	B $TS(B)=0$
Read(A)		$TS(A)=100$	
	Read(B)		$TS(B)=200$
$A=A^*2$			
Write(A)		$TS(A)=100$	
	$B=B+20$		
	Write(B)		$TS(B)=200$
Read(B)			



T1 bị hủy, sau đó khởi động lại T1 với một nhãn thời gian mới

$TS(A) \leq TS(T_1)$: T_1 đọc được A

$TS(B) \leq TS(T_2)$: T_2 đọc được B

$TS(A) \leq TS(T_1)$: T_1 ghi lên A được

$TS(B) \leq TS(T_2)$: T_2 ghi lên B được

$TS(B) > TS(T_1)$: T_1 không đọc được B



Nhãn thời gian toàn phần

T1 TS(T1)=100	T2 TS(T2)=120	A TS(A)=0
Read(A)		TS(A)=100
	Read(A)	TS(A)=120
	Write(A)	TS(A)=120
Write(A)		



T1 bị huỷ và bắt đầu thực hiện với timestamp mới

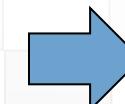
Mặc dù trong quản lý giao tác 2 hành động đọc/ghi có tác dụng rất khác nhau

Không phân biệt tính chất của thao tác là đọc hay ghi → T1 vẫn bị hủy và làm lại từ đầu với 1 timestamp mới

T1 TS(T1)=100	T2 TS(T2)=120	A TS(A)=0
Read(A)		TS(A)=100
	Read(A)	TS(A)=120
	Read(A)	TS(A)=120
Read(A)		



T1 bị huỷ và bắt đầu thực hiện với timestamp mới



Sử dụng nhãn thời gian riêng phần



Nội dung trình bày

- Các vấn đề của truy xuất đồng thời
- **Các kỹ thuật điều khiển đồng thời:**
 - **Kỹ thuật khoá**
 - * Khoá đơn giản
 - * Khoá đọc ghi
 - * Khoá đa hạt
 - **Kỹ thuật nhãn thời gian**
 - * Nhãn thời gian toàn phần
 - * **Nhãn thời gian riêng phần**
 - * Nhãn thời gian nhiều phiên bản
 - **Kỹ thuật lạc quan**
- Vấn đề khoá chết
- Các vấn đề khác



Nhãn thời gian riêng phần

- Nhãn của đơn vị dữ liệu X được tách ra thành 2 loại:
 - **RT(X)** – read time of X
 - * Ghi nhận TS(T) gần nhất (giao tác có nhãn thời gian lớn nhất) đọc X thành công
 - **WT(X)** – write time of X
 - * Ghi nhận TS(T) gần nhất (giao tác có nhãn thời gian lớn nhất) ghi X thành công
- Ngoài ra bộ lập lịch cũng quản lý trạng thái commit hay chưa của đơn vị dữ liệu X
 - $C(X)=1$ nếu dữ liệu X đã được commit. Ngược lại $C(X)=0$
 - Bộ lập lịch dựa vào hoạt động của các giao tác để gán nhãn $C(X)$ lên các đơn vị dữ liệu.
 - Kỹ thuật này được sử dụng để tránh việc lỗi đọc phải dữ liệu rác.

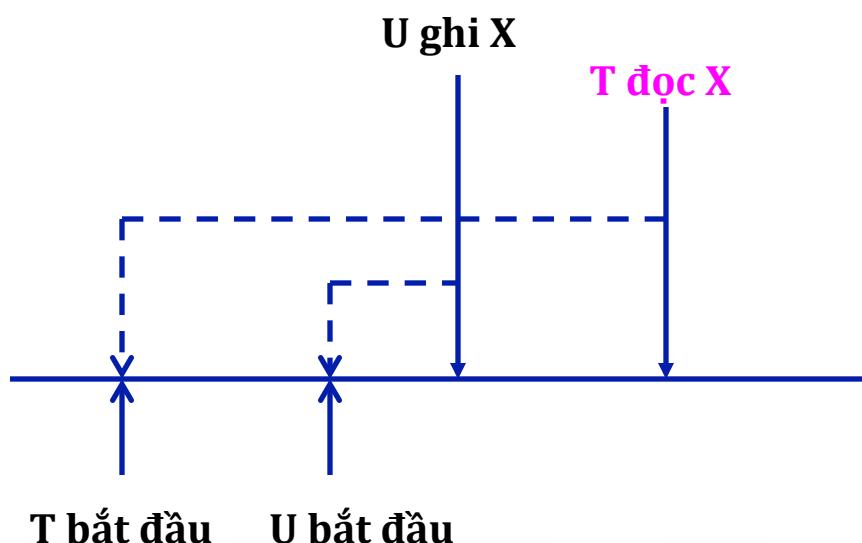


Nhãn thời gian riêng phần

- Công việc của bộ lập lịch:
 - Gán gắn các nhãn thời gian RT(X) và WT(X) và C(X)
 - Kiểm tra thao tác đọc/ghi xuất hiện khi nào để quyết định:
 - ★ Chấp nhận yêu cầu
 - ★ Trì hoãn giao tác
 - ★ Huỷ bỏ giao tác
 - ★ Bỏ qua hoạt động đó
 - Xử lý các tình huống:
 - ★ Đọc quá trễ
 - ★ Ghi quá trễ
 - ★ Đọc dữ liệu rác
 - ★ Quy tắc ghi Thomas

Nhận thời gian riêng phần

■ Đọc quá trễ



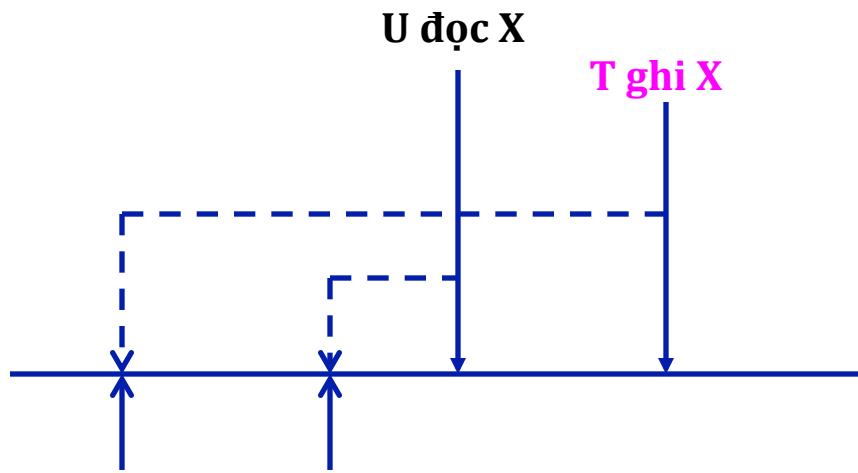
- T vào trước ($TS(T) < TS(U)$) muốn đọc X nhưng khi giá trị X hiện tại đã bị ghi bởi một giao tác khác vào sau T ($TS(T) < WT(X)$)
- T không nên đọc X do U ghi bởi vì T vào trước

→ Giải pháp: Hủy T



Nhận thời gian riêng phần

■ Ghi quá trễ



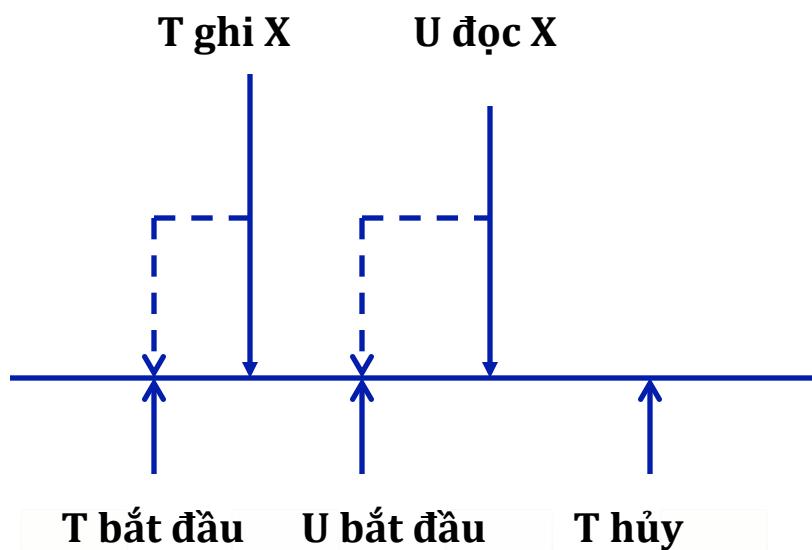
- T vào trước ($TS(T) < TS(U)$) và muốn ghi lên X, tuy nhiên X đã bị đọc bởi một giao tác vào sau T ($WT(X) < TS(T) < RT(X)$)
- T không nên ghi X do giao tác U đã đọc X. (U phải đọc giá trị do T ghi)

→ Giải pháp: Hủy T



Nhận thời gian riêng phần

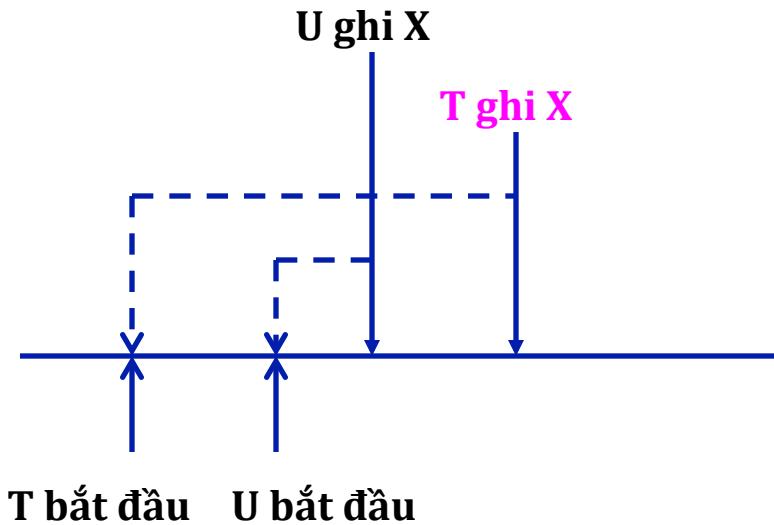
■ Đọc dữ liệu rác



- T vào trước U và thực hiện việc ghi X trước. U vào sau và thực hiện việc đọc X.
 - Nhưng T hủy → giá trị X mà U đọc là giá trị rác
- Giải pháp: U đọc X sau khi T commit hoặc sau khi T abort.

Nhân thời gian riêng phần

Quy tắc ghi Thomas

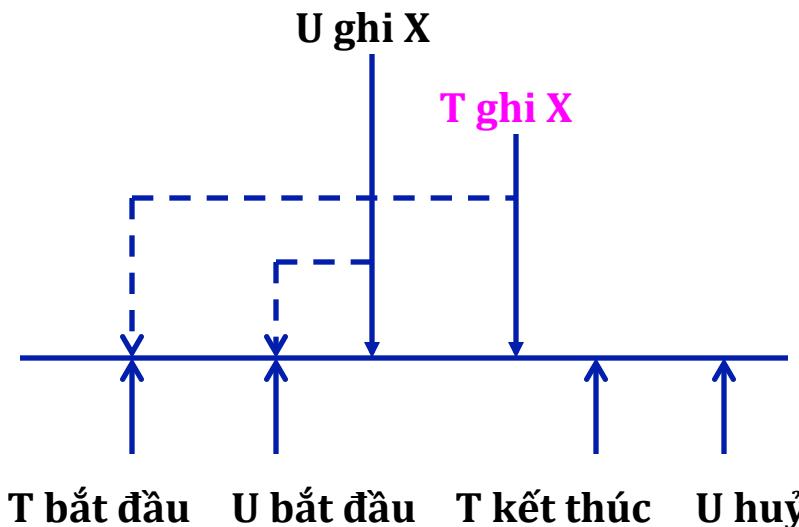


- T vào trước U vào sau nhưng khi T muốn ghi lên X thì U đã ghi lên X trước ($TS(T) < WT(X)$)
- T nếu có ghi xong X cũng không làm được gì vì:
 - T không thể đọc X vì nếu đọc X thì sẽ dẫn đến đọc trễ
 - Các giao tác khác sau T và U thì muốn đọc giá trị X được ghi bởi U
- ➔ Giải pháp: Bỏ qua thao thác ghi X của T [Quy tắc ghi Thomas]



Nhân thời gian riêng phần

■ Vấn đề với quy tắc ghi Thomas



- Trường hợp U ghi và sau đó bị huỷ → giá trị được ghi bởi U đã bị mất
- Do T đã kết thúc → cần khôi phục lại giá trị X từ T mà lệnh ghi X đã bị bỏ qua

→ Giải pháp: Khi T ghi X, nếu giao tác U đã commit thì bỏ qua T, hoặc đợi đến thời điểm U commit hoặc abort.



Nhãn thời gian riêng phần

- Tóm lại khi có yêu cầu đọc và ghi từ giao tác T. Bộ lập lịch sẽ:
 - Đáp ứng yêu cầu
 - Hủy T và khởi tạo lại T với 1 timestamp mới
 - ★ T rollback
 - Trì hoãn T, sau đó mới quyết định phải hủy T hoặc đáp ứng yêu cầu



Nhân thời gian riêng phần

■ Quy tắc :

- Nếu T cần đọc X

- * Nếu $WT(X) \leq TS(T)$ thì chờ cho X trở thành Dữ liệu đã Commit rồi cho T đọc X và gán $RT(X) = MAX(RT(X), TS(T))$
 - * Ngược lại hủy T và khởi động lại T với $TS(T)$ mới (đọc quá trễ)

- Nếu T cần ghi X

- * Nếu $RT(X) \leq TS(T)$

- Nếu $WT(X) \leq TS(T)$ thì cho T ghi X và gán $WT(X) = TS(T)$
 - Ngược lại bỏ qua thao tác ghi này của T (không hủy T)

- * Ngược lại hủy T và khởi động lại T với $TS(T)$ mới (ghi quá trễ)



Nhân thời gian riêng phần (tt)

- Quy tắc:

Read(T, X)

```
If WT(X) <= TS(T)
    Read(X); //cho phép đọc X
    RT(X):= max(RT(X), TS(T));
Else
    Rollback{T}; //hủy T và khởi tạo lại với TS mới
```

Write(T, X)

```
If RT(X) <= TS(T)
    If WT(X) <= TS(T)
        Write(X); //cho phép ghi X
        WT(X):= TS(T);
    Else
        Nếu X đã COMMIT → bỏ qua, ngược lại chờ cho
        đến khi giao tác thực hiện ghi trên X commit
        hoặc abort
    Else
        Rollback{T}; //hủy T và khởi tạo lại với TS mới
```



Ví dụ

	T_1 $TS(T_1)=100$	T_2 $TS(T_2)=200$	A $RT(A)=0$ $WT(A)=0$	B $RT(B)=0$ $WT(B)=0$	C $RT(C)=0$ $WT(C)=0$	
1	Read(A)		$RT(A)=100$ $WT(A)=0$			$WT(A) < TS(T_1)$ T_1 đọc được A
2		Read(B)		$RT(B)=200$ $WT(B)=0$		$WT(B) < TS(T_2)$ T_2 đọc được B
3	Write(A)		$RT(A)=100$ $WT(A)=100$			$RT(A) < TS(T_1)$ $WT(A) = TS(T_1)$ T_1 ghi lên A được
4		Write(B)		$RT(B)=200$ $WT(B)=200$		$RT(B) < TS(T_2)$ $WT(B) = TS(T_2)$ T_2 ghi lên B được
5		Read(C)			$RT(C)=200$ $WT(C)=0$	$WT(B) < TS(T_2)$ T_2 đọc được C
6	Read(C)				$RT(C)=200$ $WT(C)=0$	$WT(C) < TS(T_1)$ T_1 đọc được C
7	Write(C)					$RT(C) > TS(T_1)$ T_1 không ghi lên C được
						Abort

Ví dụ (tt)

T_1 $TS=200$	T_2 $TS=150$	T_3 $TS=175$	A $RT=0$ $WT=0$	B $RT=0$ $WT=0$	C $RT=0$ $WT=0$
Read(B)				$RT=200$ $WT=0$	
	Read(A)		$RT=150$ $WT=0$		
		Read(C)			$RT=175$ $WT=0$
Write(B)				$RT=200$ $WT=200$	
Write(A)			$RT=150$ $WT=200$		
	Write(C)				
		Write(A)			

Rollback

Giá trị của A đã sao lưu bởi $T_1 \rightarrow T_3$ không bị rollback và không cần ghi A



Bài tập

- Given the following sequence of events:
 - st1; st2; r1(A); r2(B); w2(A); w1(B)
 - st1; st2; st3; r1(A); r3(B); w1(C); r2(B); r2(C); w3(B); w2(A)
- Task: Tell what happens as each event occurs for a timestamp based scheduler.

Ví dụ (tt)

T_1 $TS=150$	T_2 $TS=200$	T_3 $TS=175$	T_4 $TS=255$	A $RT=0$ $WT=0$
Read(A)				$RT=150$ $WT=0$
Write(A)				$RT=150$ $WT=150$
	Read(A)			$RT=200$ $WT=0$
	Write(A)			$RT=200$ $WT=200$
		Read(A)		
			Read(A)	$RT=255$ $WT=200$

↓

Rollback

T3 bị hủy vì nó định đọc giá trị A ghi bởi T2 (mà T2 lại có nhãn thời gian lớn hơn nó). Giả sử T3 đọc giá trị A ghi bởi T1 thì T3 sẽ không bị hủy

Ý tưởng lưu giữ nhiều phiên bản của A



Nội dung trình bày

- Các vấn đề của truy xuất đồng thời
- **Các kỹ thuật điều khiển đồng thời:**
 - **Kỹ thuật khoá**
 - * Khoá đơn giản
 - * Khoá đọc ghi
 - * Khoá đa hạt
 - **Kỹ thuật nhãn thời gian**
 - * Nhãn thời gian toàn phần
 - * Nhãn thời gian riêng phần
 - * **Nhãn thời gian nhiều phiên bản**
 - **Kỹ thuật xác nhận hợp lệ**
- Vấn đề khoá chết
- Các vấn đề khác



Nhãn thời gian nhiều phiên bản

- Ý tưởng
 - Cho phép thao tác `read(A)` của T3 thực hiện
- Bên cạnh việc lưu trữ giá trị hiện hành của A, ta giữ lại các giá trị được sao lưu trước kia của A (phiên bản của A)
- Giao tác T sẽ đọc được giá trị của A ở 1 phiên bản thích hợp nào đó



Nhãn thời gian nhiều phiên bản (tt)

- Mỗi phiên bản của 1 đơn vị dữ liệu X có
 - RT(X)
 - ★ Ghi nhận lại giao tác sau cùng đọc X thành công
 - WT(X)
 - ★ Ghi nhận lại giao tác sau cùng ghi X thành công
- Khi giao tác T phát ra yêu cầu thao tác lên X
 - Tìm 1 phiên bản thích hợp của X
 - Đảm bảo tính khả tuần tự
- Một phiên bản mới của X sẽ được tạo khi hành động ghi X thành công



Nhận thời gian nhiều phiên bản (tt)

- Quy tắc:

Read(T, X)

i=“số thứ tự phiên bản sau cùng nhất của X”

While $WT(X_i) > TS(T)$

i:=i-1;//lùi lại

 Read(X_i);

$RT(X_i) := \max(RT(X_i), TS(T));$

Write(T, X)

i=“số thứ tự phiên bản sau cùng nhất của X”

While $WT(X_i) > TS(T)$

i:=i-1;//lùi lại

If $RT(X_i) > TS(T)$

 Rollback T//Hủy và khởi tạo TS mới

Else

 Tạo phiên bản X_{i+1} ;

 Write(X_{i+1});

$RT(X_{i+1}) = 0;$ //chưa có ai đọc

$WT(X_{i+1}) = TS(T);$

Ví dụ

T_1 TS=150	T_2 TS=200	T_3 TS=175	T_4 TS=255	A_0 RT=0 WT=0	A_1	A_2
Read(A)				RT=150 WT=0		
Write(A)					RT=0 WT=150	
	Read(A)				RT=200 WT=150	
	Write(A)					RT=0 WT=200
		Read(A)			RT=200 WT=150	
			Read(A)			RT=255 WT=200

Ví dụ (tt)

T_1 TS=100	T_2 TS=200	A_0 RT=0 WT=0	B_0 RT=0 WT=0	A_2	B_1
Read(A)		RT=100 WT=0			
	Write(A)		RT=0 WT=200	RT=0 WT=200	
	Write(B)				RT=0 WT=200
Read(B)				RT=100 WT=0	
Write(A)			RT=0 WT=100		



Nhận thời gian nhiều phiên bản (tt)

■ Nhận xét

- Thao tác đọc

- ★ Giao tác T chỉ đọc giá trị của phiên bản do T hay những giao tác trước T cập nhật
- ★ T không đọc giá trị của các phiên bản do các giao tác sau T cập nhật
→ Thao tác đọc không bị rollback

- Thao tác ghi

- ★ Thực hiện được thì chèn thêm phiên bản mới
- ★ Không thực hiện được thì rollback
- Tốn nhiều chi phí tìm kiếm, tốn bộ nhớ
- Nên giải phóng các phiên bản quá cũ không còn được các giao tác sử dụng



Tài liệu tham khảo

- [5] *Database systems: the complete book*, Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom, Pearson Prentice Hall, 2009
 - Chapter 18. **Concurrency Control**

Q & A



Tóm tắt CHƯƠNG 3

- Các kỹ thuật điều khiển truy xuất đồng thời
 - Kỹ thuật khoá
 - ★ Kỹ thuật khoá đơn giản
 - ★ Kỹ thuật khoá đọc ghi
 - ★ Nghi thức 2 giai đoạn
 - ★ Khoá cập nhật
 - ★ Các tình huống xảy ra deadlock, các loại deadlock
 - ★ Khoá đa hạt
 - Kỹ thuật nhän thời gian
 - ★ Kỹ thuật nhän thời gian toàn phần
 - ★ Kỹ thuật nhän thời gian riêng phần
 - ★ Kỹ thuật nhän thời gian nhiều phiên bản



Timestamp vs. Locking

- Schedule allowed by locks but not timestamps

<i>Schedule S14</i>	
<i>T34</i>	<i>T35</i>
read A	
	read B write B
read B	

- Schedule allowed by timestamps but not by locks:

<i>Schedule S15</i>		
<i>T36</i>	<i>T37</i>	<i>T38</i>
write A		
	write A	write A
write B		
	write B	



Cascading Rollbacks

- One transaction aborting can cause other transactions to abort.

Schedule S11		
<i>T</i> 22	<i>T</i> 23	<i>T</i> 24
LX A		
LX B		
UN A		
	LX A UN A	
		LX A

- *T*22 aborts \Rightarrow we have to rollback *T*23 and *T*24.
- How to eliminate these *cascading rollbacks*?
 - Don't let transactions read “dirty” uncommitted data.



Strict Timestamp Based Concurrency Control

- How to avoid cascading rollbacks?
 - Transactions should read only committed values.
- *Strict timestamp concurrency control protocol*

read X:

if $t \geq w\text{-max}(X)$

$r\text{-max}(X) \leftarrow \max(t, r\text{-max}(X))$

 wait for a committed value of X

 perform read

else abort

write X:

if $t \geq r\text{-max}(X)$ **and** $t \geq w\text{-max}(X)$

$w\text{-max}(X) \leftarrow t$

 wait until X value is a committed value and
 pending reads are done

 perform write

else

if $t < r\text{-max}(X)$

 abort

else

 do nothing



SQL isolation levels

- A transactions in SQL may be chosen to have one of four **isolation levels**:
 -  •READ UNCOMMITTED: "No locks are obtained."
 -  •READ COMMITTED: "Read locks are immediately released – read values may change during the transaction."
 -  •REPEATABLE READ: "2PL but no lock when adding new tuples."
 -  •SERIALIZABLE: "2PL with lock when adding new tuples."



Disadvantages of locking

- Lock management overhead.
- Deadlock detection/resolution.
- Concurrency is significantly lowered, when congested nodes are locked.
- To allow a transaction to abort itself when mistakes occur, locks can't be released until the end of transaction, thus currency is significantly lowered
- (Most Important) Conflicts are rare. (We might get better performance by not locking, and instead checking for conflicts at commit time.)



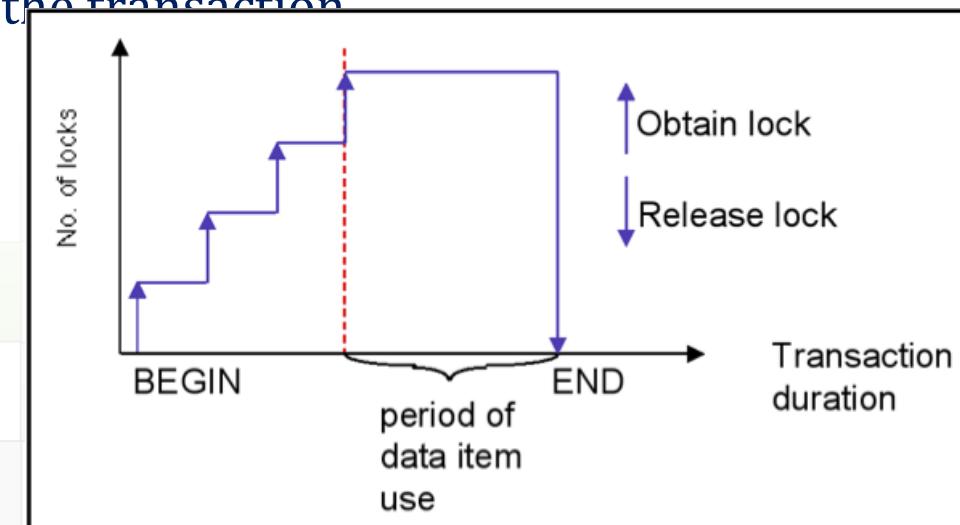
Optimism vs pessimism

-  Pessimistic concurrency is best in high- conflict situations:
 -  Smallest number of aborts.
 -  No wasted processing.
-  Optimistic concurrency control is best if conflicts are rare, e.g., if there are many read-only transactions.
 -  Highest level of concurrency.
 -  Hybrid solutions often used in practice.



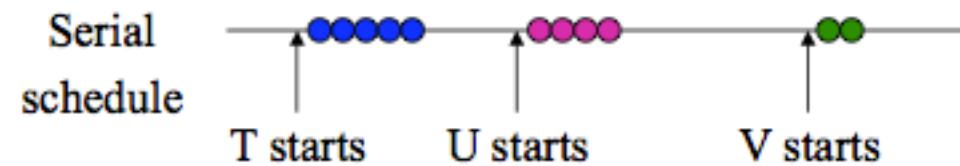
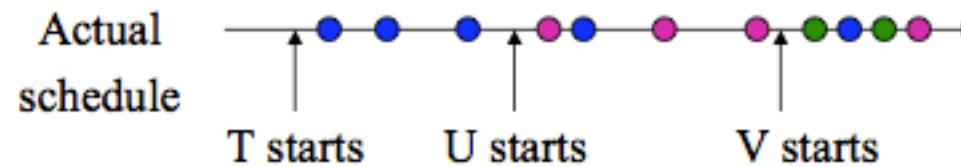
Two-Phase Locking (2PL) . . .

- **Properties of the 2PL protocol**
 - Generates **conflict-serializable** schedules
 - But schedules may cause **cascading aborts**
 - * If a transaction aborts after it releases a lock, it may cause other transactions that have accessed the unlocked data item to abort as well
- **Strict 2PL locking protocol**
 - Holds the locks till the end of the ~~transaction~~
 - Cascading aborts are avoided



Timestamp-based approach

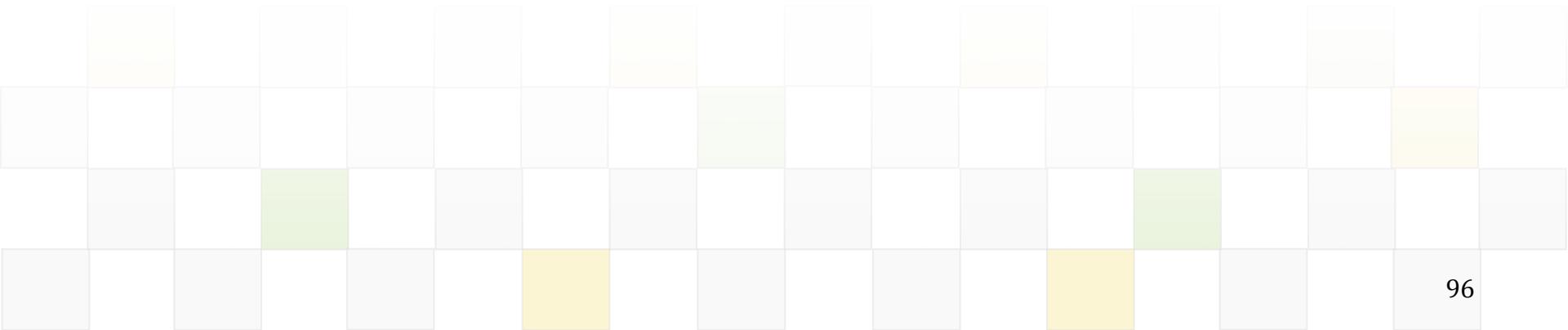
- Assumed Serial Schedule in Timestamp-based approach:
 - Conflict serializable schedule that is equivalent to a serial schedule in which the timestamp order of transactions is the order to execute them





Timestamp-based approach

- Scheduler's response to a T's request
 - Grant the request
 - Abort and restart (roll back) T with a new timestamp
 - Delay T and later decide whether to abort T or to grant the request





THUẬT NGỮ

- Bộ lập lịch
- Bộ phận quản lý giao tác
- Bảng khoá
- Bộ phận quản lý đồng thời



HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

Chương 4:

AN TOÀN VÀ AN NINH DỮ LIỆU

GVLT: *Nguyễn Trường Sơn*



Nội dung trình bày

■ An toàn dữ liệu

- Phân loại sự cố
- Nhật ký giao tác
- Điểm kiểm tra
- Undo logging
- Redo logging
- Undo / Redo logging

KHÔI PHỤC SAU SỰ CỐ

Từ khoá:

- database recovery
- crash recovery

■ An ninh dữ liệu

- Cơ chế phân quyền
- Cơ chế mã hoá



Giới thiệu

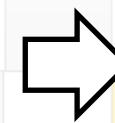
- Cơ sở dữ liệu luôn cần phải ở trạng thái nhất quán = đảm bảo tất cả các ràng buộc toàn vẹn (RBTV)
- Nguyên nhân dẫn đến RBTV bị vi phạm:
 - Do **chia sẻ dữ liệu** (data sharing): Dữ liệu được chia sẻ cho nhiều giao tác cùng lúc



Sử dụng các kỹ thuật quản lý giao tác
và xử lý đồng thời

- Do **sự cố**:

- * Lỗi lập trình của các giao tác
- * Lỗi lập trình của DBMS hoặc do hệ điều hành
- * Hư hỏng phần cứng & sự cố khác

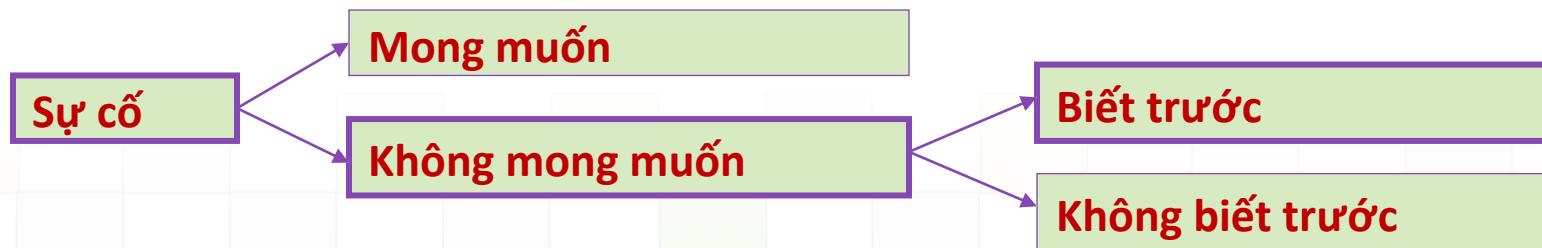


Sử dụng các kỹ thuật khôi phục sự cố



Các loại sự cố

- Phân loại theo nguyên nhân:
 - Sự cố do nhập liệu sai – Erronous Data Entry
 - Sự cố trên thiết bị lưu trữ – Media failures
 - Sự cố giao tác – Transaction failures
 - Sự cố hệ thống – System failures
- Phân loại theo tính chất:





Sự cố do nhập liệu sai (Erroneous Data Entry)

■ Bao gồm:

- *Dữ liệu sai hiển nhiên*: Là sự nhập sai dữ liệu mà máy tính có thể phát hiện được
 - * Vd : Nhập thiếu 1 số trong dãy số điện thoại, nhập sai khóa ngoại, nhập chuỗi tràn, sai kiểu dữ liệu...
- *Dữ liệu sai không hiển nhiên*: Là sự nhập sai dữ liệu liên quan đến ngữ nghĩa mà máy tính khó có thể tự nó phát hiện được
 - * Vd : Nhập sai 1 số trong dãy số điện thoại

■ Giải quyết: Hệ quản trị CSDL cung cấp các cơ chế cho phép phát hiện lỗi

- Ràng buộc khóa chính, khóa ngoại
- Ràng buộc miền giá trị
- Trigger



Sự cố trên thiết bị lưu trữ (Media Failures)

■ Là những sự cố:

- Là những sự cố gây nên việc mất hay không thể truy xuất dữ liệu ở bộ nhớ ngoài (ổ cứng, CD, băng từ...)
 - * Vd: Cháy nổ gây phá hủy thiết bị lưu trữ,...
 - * Vd: Đầu đọc của đĩa cứng hư, sector trên đĩa cứng hư, ...
- Đây là loại sự cố nguy hiểm nhất, khó khôi phục trọn vẹn

■ Giải quyết:

- Phải backup thường xuyên (tất cả hoặc chỉ phần thay đổi), chu kỳ không được quá thưa
- Chạy nhiều bản CSDL song hành (1 bản chính – primary và nhiều bản phụ – minor) và thực hiện đồng bộ tức thì
 - * Tốn thiết bị lưu trữ và đòi hỏi phần cứng rất mạnh
 - * Kìm hãm tốc độ hệ thống
 - * Bản minor phải đặt ở vị trí địa lý khác bản primary



Sự cố giao tác (Transaction Failures)

- Sự cố làm cho 1 giao tác kết thúc không bình thường (không đến được lệnh commit hay lệnh rollback của chính nó)
- Ví dụ
 - Chia cho không
 - Giao tác bị hủy
 - Dữ liệu nhập sai
 - Tràn số
- Giải quyết : Khi giao tác T bị sự cố, DBMS sẽ
 - Hủy T và các giao tác bị quay lui dây chuyền theo nó
 - Tra lock-table và giải phóng các khóa mà các giao tác này đang giữ
 - Reset lại các giá trị mà các giao tác này đã ghi
 - Thực hiện lại tất cả các giao tác này

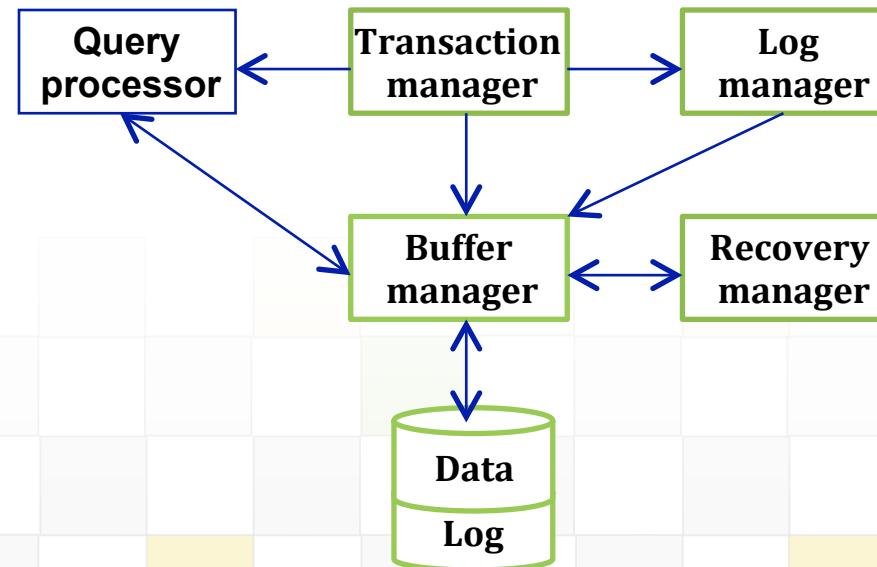


Sự cố hệ thống (System Failures)

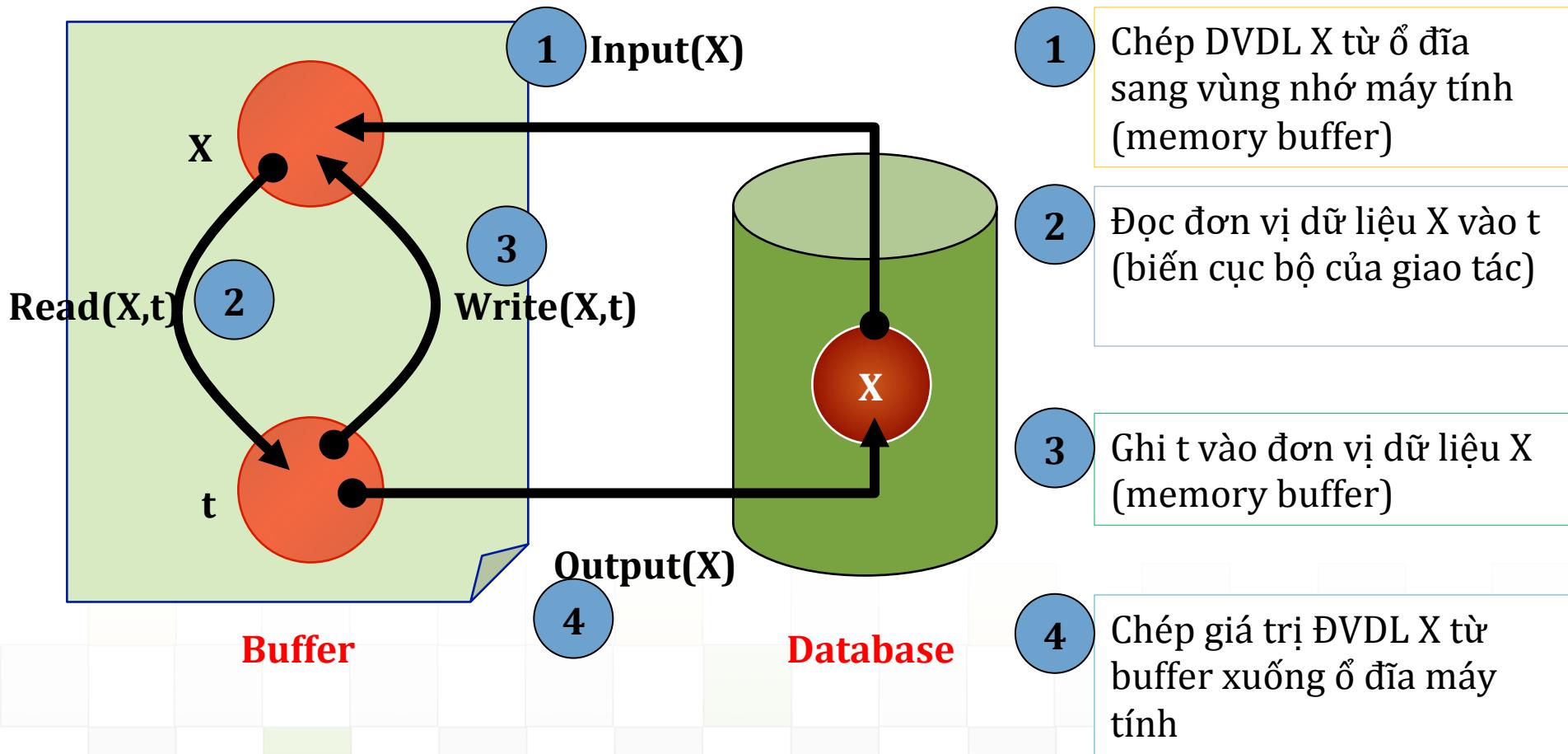
- Là những sự cố gây nên bởi
 - Lỗi phần cứng
 - ★ Cúp điện
 - ★ Hư bộ nhớ trong
 - ★ Hư CPU
 - ★ ...
 - Lỗi phần mềm
 - ★ Lỗi hệ điều hành
 - ★ Lỗi DBMS
 - ★ ...
- Giải quyết : Hệ quản trị CSDL cần cứu chữa và phục hồi dữ liệu
 - Nhật ký giao tác (transaction log)

Mục tiêu của khôi phục sự cố

- Đưa dữ liệu về trạng thái sau cùng nhất trước khi xảy ra sự cố
- Đảm bảo 2 tính chất của giao tác:
 - Nguyên tố (atomic)
 - Bền vững (durability)



Các thao tác đọc ghi dữ liệu trong DBMS



Việc đọc / ghi dữ liệu trong DBMS thực chất là việc chuyển đổi các giá trị từ vào các không gian địa chỉ: Bộ nhớ \leftrightarrow Ổ đĩa



Nội dung trình bày

■ An toàn dữ liệu

- Phân loại sự cố
- Nhật ký giao tác
- Điểm kiểm tra
 - * Điểm kiểm tra đơn giản
 - * Điểm kiểm tra linh động
- Undo logging
- Redo logging
- Undo / Redo logging

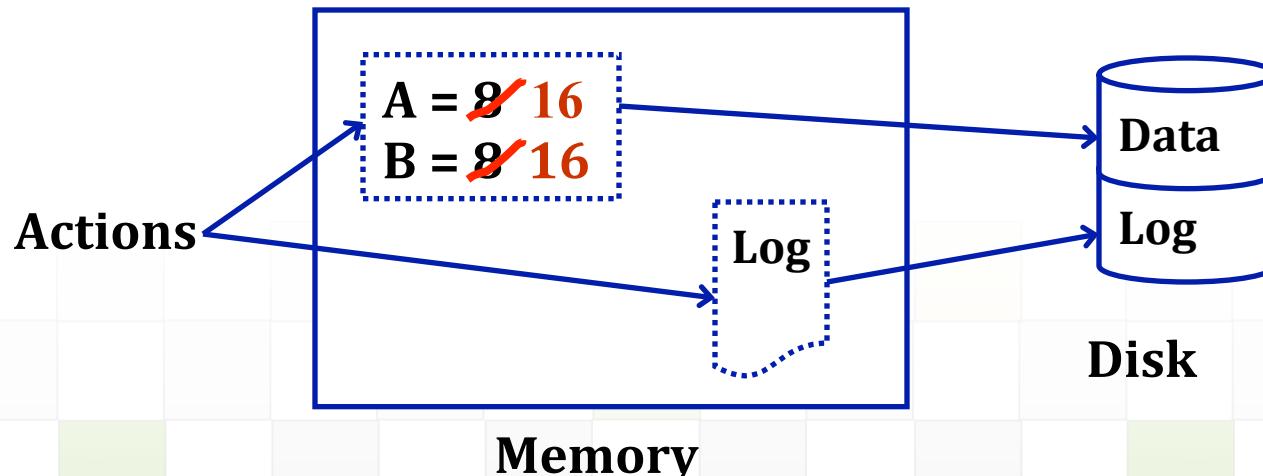
■ An ninh dữ liệu

- Cơ chế phân quyền
- Cơ chế mã hoá



Nhật ký giao tác

- Nhật ký giao tác là một chuỗi các mẫu tin (*log record*) ghi lại các hành động của DBMS
- Nhật ký là một tập tin tuần tự được lưu trữ trên bộ nhớ chính và được ghi xuống đĩa ngay khi có thể



Flush-log: là hành động chép những block mẫu tin nhật ký mới chưa được chép từ bộ nhớ vào ổ đĩa



Nhật ký giao tác (tt)

- Một mẫu tin nhật ký có thể là:

<start T>

Ghi nhận giao tác T bắt đầu hoạt động

<commit T>

Ghi nhận giao tác T đã hoàn tất

<abort T>

Ghi nhận giao tác T bị hủy

<T, X, v, w>

Ghi nhận giao tác T cập nhật lên đơn vị dữ liệu X

Nhật ký giao tác được sử dụng trong các phương pháp khôi phục sự cố, mỗi kỹ thuật khôi phục sự cố khác nhau sẽ có một cách ghi nhật ký khác nhau và các cú pháp mẫu tin khác nhau.



Nhật ký giao tác (tt)

- Khi sự cố hệ thống xảy ra:
 - DBMS sẽ tra cứu nhật ký giao tác để khôi phục lại trạng thái nhất quán của dữ liệu.
- Để sửa chữa các sự cố:
 - Một vài giao tác sẽ phải thực hiện lại (**redo**)
 - * Những giá trị mà giao tác này đã cập nhật xuống CSDL sẽ phải cập nhật lần nữa
 - Một vài giao tác không cần phải thực hiện lại (**undo**)
 - * CSDL sẽ được khôi phục về lại trạng thái trước khi các giao tác này được thực hiện



Điểm kiểm tra (Check point)

- Khi cần, DBMS không thể tra cứu toàn bộ nhật ký vì
 - Nhật ký tích lũy thông tin về tất cả các hành động của một giai đoạn rất dài
 - Quá trình tra cứu nhật ký → Phải quét hết tập tin nhật ký → Mất nhiều thời gian
 - Thực hiện lại các giao tác đã được ghi xuống đĩa làm cho việc khôi phục lặp lại → tốn thời gian vô ích.
- Giải pháp: Dùng điểm kiểm tra (check point)
 - Nhật ký giao tác có thêm mẫu tin <checkpoint> hay <ckpt>
 - Mẫu tin <checkpoint> sẽ được ghi xuống nhật ký định kỳ vào thời điểm mà DBMS ghi tất cả những gì thay đổi của CSDL từ vùng đệm xuống đĩa



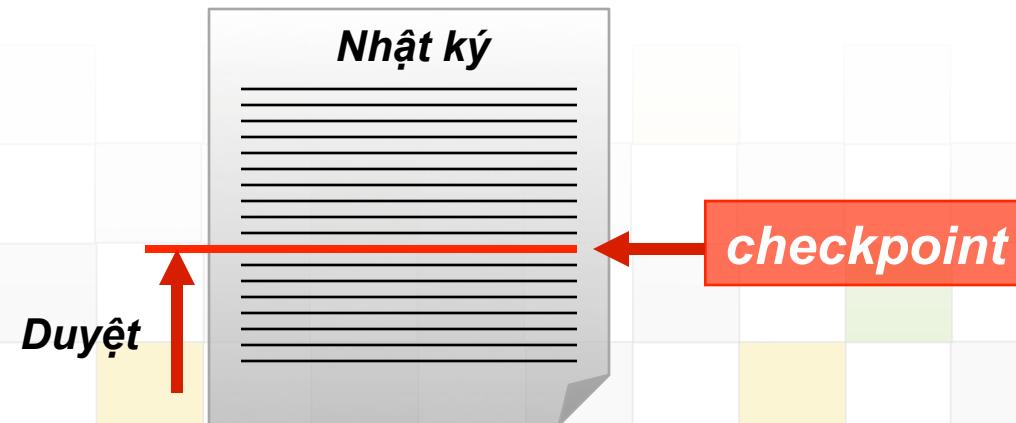
Điểm kiểm tra đơn giản

- Khi đến điểm kiểm tra, DBMS sẽ
 1. Tạm dừng tiếp nhận các giao tác mới
 2. Đợi các giao tác đang thực hiện
 - * Hoặc là hoàn tất (commit)
 - * Hoặc là hủy bỏ (abort)
 - * và ghi mẫu tin <commit T> hay <abort T> vào nhật ký
 3. Tiến hành ghi dữ liệu và nhật ký từ vùng đệm xuống đĩa
 4. Tạo 1 mẫu tin <checkpoint> và ghi xuống đĩa
 5. Trở về công việc bình thường (tiếp tục nhận các giao tác mới)



Điểm kiểm tra đơn giản (tt)

- Khi có sự cố, DBMS dùng nhật ký để khôi phục :
 - Các giao tác ở phía trước điểm kiểm tra **mới nhất** là những giao tác đã kết thúc → không cần làm lại
 - Các giao tác ở phía sau điểm kiểm tra là những giao tác chưa thực hiện xong → cần khôi phục
- Như vậy, BDMS sẽ :
 - Không phải duyệt hết nhật ký
 - Chỉ duyệt ngược từ cuối nhật ký đến điểm kiểm tra





Điểm kiểm tra linh động

■ Đặc điểm:

- Mẫu tin **<checkpoint>** trong *Điểm kiểm tra đơn giản* này được chia thành 2 mẫu tin :
 - * Mẫu tin **<start ckpt (T₁, T₂, ..., T_k)>** : Bắt đầu checkpoint
 - T₁, T₂, ..., T_k là các giao tác đang chờ dang khi bắt đầu checkpoint
 - * Mẫu tin **<end ckpt>** : Kết thúc checkpoint
- Cho phép tiếp nhận các giao tác mới trong quá trình checkpoint



Điểm kiểm tra linh động (tt)

- Khi đến điểm kiểm tra, DBMS sẽ
 1. Tạm ngưng mọi hoạt động
 2. Tạo và ghi mẫu tin **<start ckpt (T₁, T₂, ..., T_k)>**
 - * T₁, T₂, ..., T_k là những giao tác đang thực thi khi ấy
 3. Chờ cho đến khi T₁, T₂, ..., T_k hoàn tất hay hủy bỏ
 4. Trong khi ấy không ngăn các giao tác mới bắt đầu
 5. Khi T₁, T₂, ..., T_k kết thúc, tạo mẫu tin **<end ckpt>** và ghi xuống đĩa



Nội dung trình bày

■ An toàn dữ liệu

- Phân loại sự cố
- Nhật ký giao tác
- Điểm kiểm tra
- **Undo loging**
- Redo loging
- Undo / Redo loging

■ An ninh dữ liệu

- Cơ chế phân quyền
- Cơ chế mã hoá



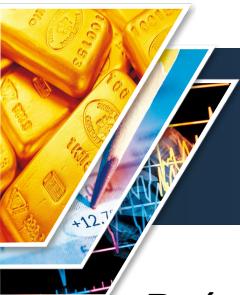
Phương pháp Undo-Logging

- Qui tắc
 - (1) Một thao tác cập nhật đơn vị dữ liệu X phát sinh ra 1 mẫu tin nhật ký ghi nhận lại giá trị cũ của X **<T, X, v>**
 - (2) Trước khi X được cập nhật xuống đĩa, mẫu tin **<T, X, v>** đã phải có trên đĩa.
- (3) Trước khi mẫu tin **<commit T>** được ghi xuống đĩa, tất cả các cập nhật của T đã được phản ánh lên đĩa

*Phải ghi nhật ký xuống đĩa (**flush log**) trước khi thực hiện các câu lệnh OUTPUT*

*Mẫu tin **<commit T>** phải sau các câu lệnh OUTPUT*

Flush-log: chép những block mẫu tin nhật ký mới chưa được chép từ bộ nhớ vào ổ đĩa



Ví dụ

Bước	Hành động	t	Mem A	Mem B	Disk A	Disk B	Mem Log
1							<start T>
2	Read(A,t)	8	8		8	8	
3	$t:=t^2$	16	8		8	8	
4	Write(A,t)	16	16		8	8	<T, A, 8>
5	Read(B,t)	8	16	8	8	8	
6	$t:=t^2$	16	16	8	8	8	
7	Write(B,t)	16	16	16	8	8	<T, B, 8>
8							<commit T>
9	Flush log						
10	Output(A)	16	16	16	16	8	
11	Output(B)	16	16	16	16	16	
12	Flush log						



Các hành động flush log, OUTPUT và cách ghi nhật ký đúng hay không ?



Ví dụ

Bước	Hành động	t	Mem A	Mem B	Disk A	Disk B	Mem Log
1							<start T>
2	Read(A,t)	8	8		8	8	
3	$t:=t^2$	16	8		8	8	
4	Write(A,t)	16	16		8	8	<T, A, 8>
5	Read(B,t)	8	16	8	8	8	
6	$t:=t^2$	16	16	8	8	8	
7	Write(B,t)	16	16	16	8	8	<T, B, 8>
8	Flush log						
9	Output(A)	16	16	16	16	8	
10	Output(B)	16	16	16	16	16	
11							<commit T>
12	Flush log						



Các hành động flush log, OUTPUT và cách ghi nhật ký đúng hay không ?



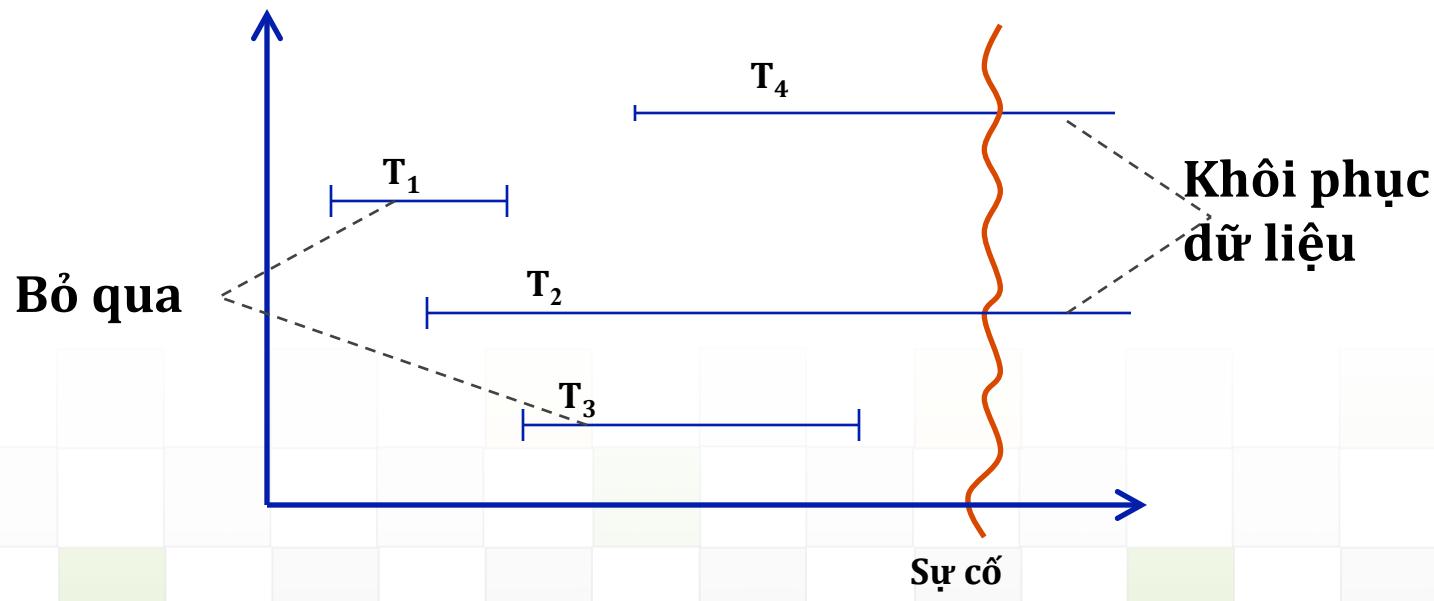
Phương pháp Undo-Logging (tt)

- Quá trình khôi phục
 - (1) Gọi S là tập các giao tác chưa kết thúc
 - * Có **<start Ti>** trong nhật ký nhưng
 - * Không có **<commit Ti>** hay **<abort Ti>** trong nhật ký
 - (2) Với mỗi mẫu tin **<Ti, X, v>** trong nhật ký (theo thứ tự **cuối** tập tin → **đầu** tập tin) → thực hiện ghi lại giá trị cũ cho ĐVDL X
 - * Nếu $Ti \in S$ thì $\begin{cases} - Write(X, v) \\ - Output(X) \end{cases}$
 - (3) Với mỗi $Ti \in S$
 - * Ghi mẫu tin **<abort Ti>** lên nhật ký



Phương pháp Undo-Logging (tt)

- Khi có sự cố
 - T1 và T3 đã hoàn tất
 - T2 và T4 chưa kết thúc





Ví dụ

Bước	Hành động	t	Mem A	Mem B	Disk A	Disk B	Mem Log
1							<start T>
2	Read(A,t)	8	8		8	8	
3	$t:=t^2$	16	8		8	8	
4	Write(A,t)	16	16		8	8	<T, A, 8>
5	Read(B,t)	8	16	8	8	8	
6	$t:=t^2$	16	16	8	8	8	
7	Write(B,t)	16	16	16	8	8	<T, B, 8>
8	Flush log						A và B không thay đổi nên không cần khôi phục
9	Output(A)	16	16	16	16	8	
10	Output(B)	16	16	16	16	16	
11							<commit T>
12	Flush log						Không cần khôi phục A và B

Sự cố →

Sự cố →

Sự cố →



Undo-Logging & Checkpoint

- Giả sử nội dung nhật ký như sau:

<**start T₁**>

<T₁, A, 5>

<**start T₂**>

Thời điểm DBMS cần
thực hiện Checkpoint

<T₂, B, 10>

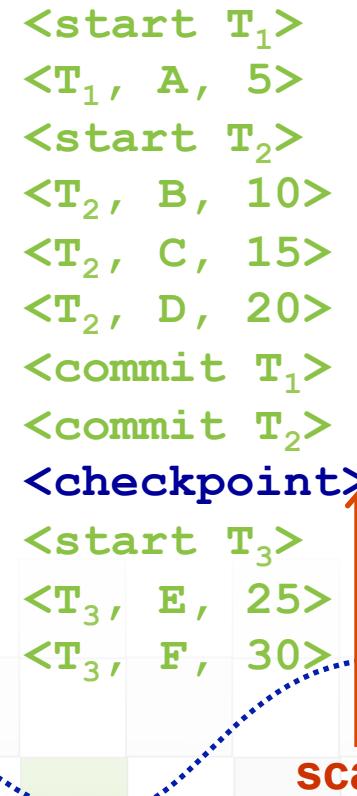
- Cách ghi nhật ký:

- Vì T1 và T2 đang thực thi nên chờ
- Sau khi T1 và T2 hoàn tất hoặc hủy bỏ
- Ghi mẫu tin <**checkpoint**> lên nhật ký

Undo-Logging & Checkpoint (tt)

■ Ví dụ:

```
<start T1>  
<T1, A, 5>  
<start T2>  
<T2, B, 10>  
<T2, C, 15>  
<T2, D, 20>  
<commit T1>  
<commit T2>  
<checkpoint>  
<start T3>  
<T3, E, 25>  
<T3, F, 30>
```



- <T₃, F, 30>
 - * T₃ chưa kết thúc
 - * Khôi phục F=30
- < T₃, E, 25 >
 - * Khôi phục E=25
- <checkpoint>
 - * Dừng
 - * Ghi **<abort T₃>**



Undo-Logging & Checkpoint (tt)

- Phương pháp khôi phục Undo logging với Checkpoint đơn giản:
 - QUÉT nhật ký từ cuối → Tìm các giao tác chưa hoàn tất và khôi phục lại giá trị cho các ĐVDL đã bị thay đổi bởi các giao tác này.
 - Chỉ quét từ cuối nhật ký cho đến mẫu tin <CHECKPOINT> để tìm các giao tác chưa hoàn tất. Vì các giao tác trước <CHECKPOINT> đã chắc chắn hoàn tất.



Undo-Logging & Checkpoint (tt)

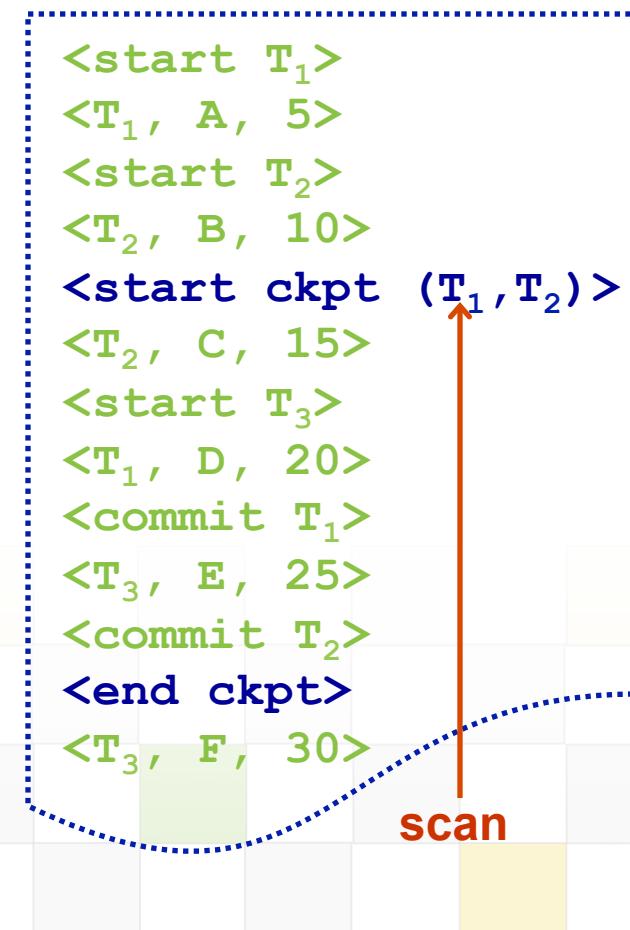
Thời điểm DBMS cần thực
hiện check point
(Nonquiescent Checkpoint)

<start T₁>
<T₁, A, 5>
<start T₂>
<T₂, B, 10>

- Vì T1 và T2 đang thực thi nên tạo **<start ckpt (T1,T2)>**
- Trong khi chờ T1 và T2 kết thúc, DBMS vẫn tiếp nhận các giao tác mới
- Sau khi T1 và T2 kết thúc, ghi **<end ckpt>** lên nhật ký

Undo-Logging & Checkpoint (tt)

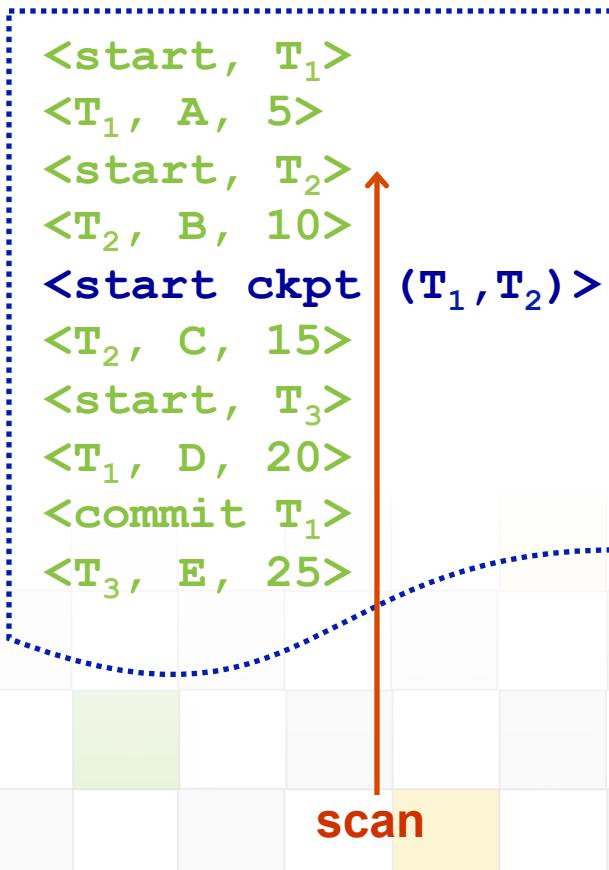
- Trường hợp 1: Khôi phục với đầy đủ `<start ckpt>` `<end ckpt>`



- `<T3, F, 30>`
 - * T₃ chưa kết thúc
 - * Khôi phục F=30
- `<end ckpt>`
 - * Những giao tác bắt đầu trước `<start ckpt>` đã hoàn tất
 - * T₁ và T₂ đã hoàn tất
- `<T3, E, 25>`
 - * Khôi phục E=25
- `<start ckpt (T1, T2)>`
 - * Dừng
 - * Ghi `<abort T3>`

Undo-Logging & Checkpoint (tt)

- Trường hợp 2: Khôi phục khi thiếu `<end ckpt>`



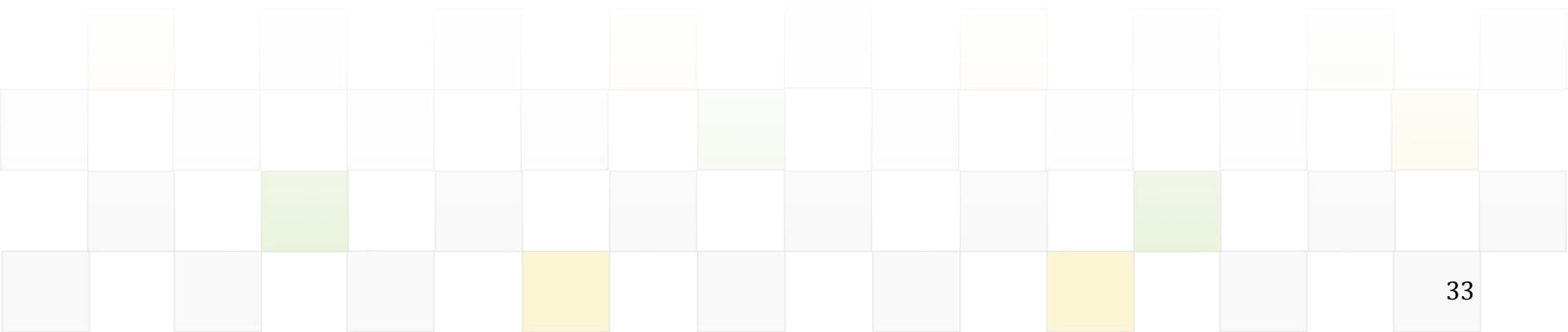
- `<T3, E, 25>`
 - * T₃ chưa kết thúc
 - * Khôi phục E=25
- `<commit T1>`
 - * T₁ bắt đầu trước `<start ckpt>` và đã hoàn tất
- `<T2, C, 15>`
 - * T₂ bắt đầu trước `<start ckpt>` và chưa kết thúc
 - * Khôi phục C=15
- `<start ckpt (T1, T2)>`
 - * Chỉ có T₁ và T₂ bắt đầu trước đó
- `<T2, B, 10 >`
 - * Khôi phục B=10
- `<start T2>`
 - * Dừng, ghi abort(T₃) và ghi abort(T₂)



Undo-Logging & Checkpoint (tt)

- Trường hợp 3: Khôi phục khi thiếu <end ckpt>

```
<start, T1>
<T1, A, 5>
<start, T2>
<T2, B, 10>
<start ckpt (T1, T2)>
```





Undo-Logging & Checkpoint (tt)

- Phương pháp khôi phục với Checkpoint linh động:
 - QUÉT nhật ký từ cuối về trước → Tìm các giao tác chưa hoàn tất và khôi phục lại giá trị cho các ĐVDL đã bị thay đổi bởi các giao tác này.
 - PHẠM VI QUÉT NHẬT KÝ:
 - ★ Nếu quét từ cuối mà gặp mẫu tin <END CKPT> trước → Các giao tác nằm trong mẫu tin <START CKPT(Ti, ..., Tk)> đã hoàn tất → Chỉ cần quét từ cuối đến mẫu tin <START ...> để tìm các giao tác chưa hoàn tất.
 - ★ Nếu quét từ cuối mà gặp mẫu tin <START CKPT(Ti, ..., Tk)> trước → sự cố xuất hiện trong khi thực hiện checkpoint → các giao tác nằm trong mẫu tin <START CKPT(Ti, ..., Tk)> chưa hoàn tất hết → Phải quét từ sau về trước qua luôn cả mẫu tin <START CKPT(Ti, ..., Tk)> để tìm các giao tác chưa hoàn tất.



Nội dung trình bày

■ An toàn dữ liệu

- Phân loại sự cố
- Nhật ký giao tác
- Điểm kiểm tra
- Undo logging
- Redo logging**
- Undo / Redo logging

■ An ninh dữ liệu

- Cơ chế phân quyền
- Cơ chế mã hoá



Phương pháp Redo-Logging

■ Qui tắc:

- Một thao tác T muốn cập nhật đơn vị dữ liệu X sẽ phát sinh ra 1 mẫu tin nhật ký
 - * Mẫu tin của thao tác cập nhật chỉ ghi nhận lại giá trị mới
 - * Cấu trúc mẫu tin $\langle T, X, w \rangle$ với w là giá trị mới của X

Khi thực hiện WRITE(X) thì phải ghi mẫu tin $\langle T, X, new_value \rangle$

- Trước khi X được cập nhật xuống đĩa
 - * Tất cả các mẫu tin nhật ký $\langle T_i, X, w \rangle$ của các giao tác T_i cập nhật X đã phải có trên đĩa
 - * Kể cả các mẫu tin hoàn tất **<commit T_i >** của các T_i cũng đã phải được ghi xuống đĩa

Các hành động $\langle T, X, w \rangle$ **<commit T >** và
Flush log phải trước OUTPUT (X)

Ví dụ

Bước	Hành động	t	Mem A	Mem B	Disk A	Disk B	Mem Log
1							<start T>
2	Read(A,t)	8	8		8	8	
3	$t:=t^2$	16	8		8	8	
4	Write(A,t)	16	16		8	8	<T, A, 8>
5	Read(B,t)	8	16	8	8	8	
6	$t:=t^2$	16	16	8	8	8	
7	Write(B,t)	16	16	16	8	8	<T, B, 8>
8	Flush log						
9	Output(A)	16	16	16	16	8	
10	Output(B)	16	16	16	16	16	
11							<commit T>
12	Flush log						



Các hành động flush log, OUTPUT và cách ghi nhật ký đúng quy tắc hay không ?

Ví dụ

Bước	Hành động	t	Mem A	Mem B	Disk A	Disk B	Mem Log
1							<start T>
2	Read(A,t)	8	8		8	8	
3	$t:=t*2$	16	8		8	8	
4	Write(A,t)	16	16		8	8	<T, A, 16>
5	Read(B,t)	8	16	8	8	8	
6	$t:=t*2$	16	16	8	8	8	
7	Write(B,t)	16	16	16	8	8	<T, B, 16>
8							<commit T>
9	Flush log						
10	Output(A)	16	16	16	16	8	
11	Output(B)	16	16	16	16	16	



Các hành động flush log, OUTPUT và cách ghi nhật ký đúng hay không ?



Phương pháp Redo-Logging (tt)

■ Quá trình khôi phục:

- Gọi S là tập các giao tác hoàn tất

- * Có mẫu tin **<commit Ti>** trong nhật ký

- Với mỗi mẫu tin $\langle Ti, X, w \rangle$ trong nhật ký

- * Nếu $Ti \in S$ thì

- Write(X, w)

- Output(X)

- (theo thứ tự **đầu** tập tin đến **cuối** tập tin)

- Với mỗi $Tj \notin S$

- * Ghi mẫu tin **<abort Tj>** lên nhật ký

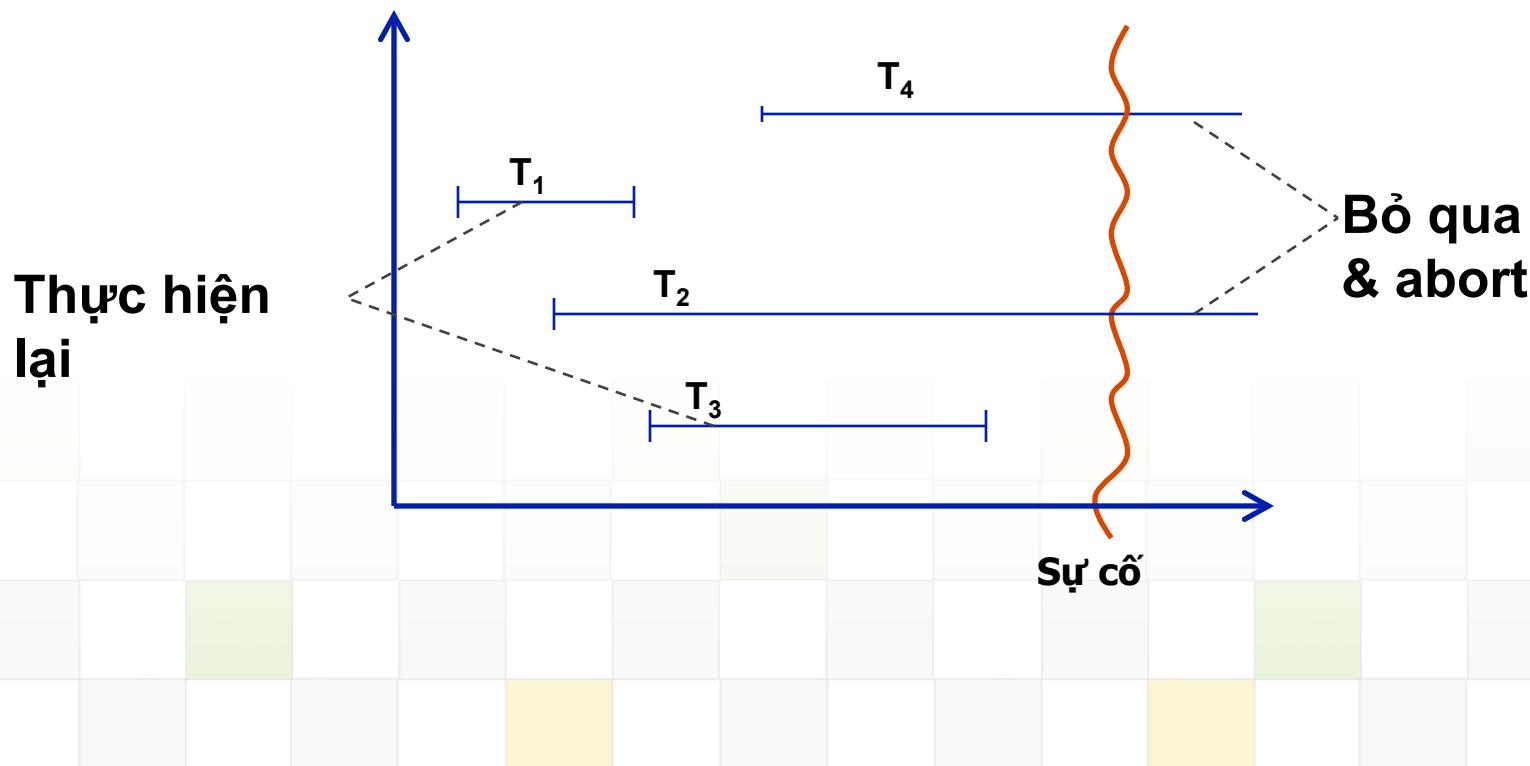
- (theo thứ tự **cuối** tập tin đến **đầu** tập tin)



Phương pháp Redo-Logging (tt)

■ Khi có sự cố

- T_1 và T_3 đã hoàn tất
- T_2 và T_4 chưa kết thúc





Ví dụ

Bước	Hành động t		Mem A	Mem B	Disk A	Disk B	Mem Log
1							<start T>
2	Read(A,t)	8	8		8	8	
3	t:=t*2	16	8		8	8	
4	Write(A,t)	16	16		8	8	<T, A, 16>
5	Read(B,t)	8	16	8	8	8	
6	t:=t*2	16	16	8	8	8	
7	Write(B,t)	16	16	16	8	8	<T, B, 16>
8							<commit T>
9	Flush log						Xem như T chưa hoàn tất, A và B không có thay đổi
10	Output(A)	16	16	16	16	8	Thực hiện lại T, ghi A=16 và B=16
11	Output(B)	16	16	16	16	16	Thực hiện lại T, ghi A=16 và B=16



Redo-Logging & Checkpoint

- Nhận xét

- Phương pháp Redo thực hiện ghi dữ liệu xuống đĩa trễ so với thời điểm hoàn tất của các giao tác → đến điểm lưu trữ thì sẽ ghi tất cả các dữ liệu đang còn ở buffer của những giao tác đã hoàn tất vào đĩa.

<start ckpt>

Thực hiện ghi ĐVDL đang ở trên buffer mà chưa được ghi xuống đĩa của những giao tác đã COMMIT khi mẫu tin <start ckpt> được ghi xuống nhật ký.

<end ckpt>



Redo-Logging & Checkpoint (tt)

QUY TẮC GHI CHECKPOINT

■ Đến điểm lưu trữ, DBMS

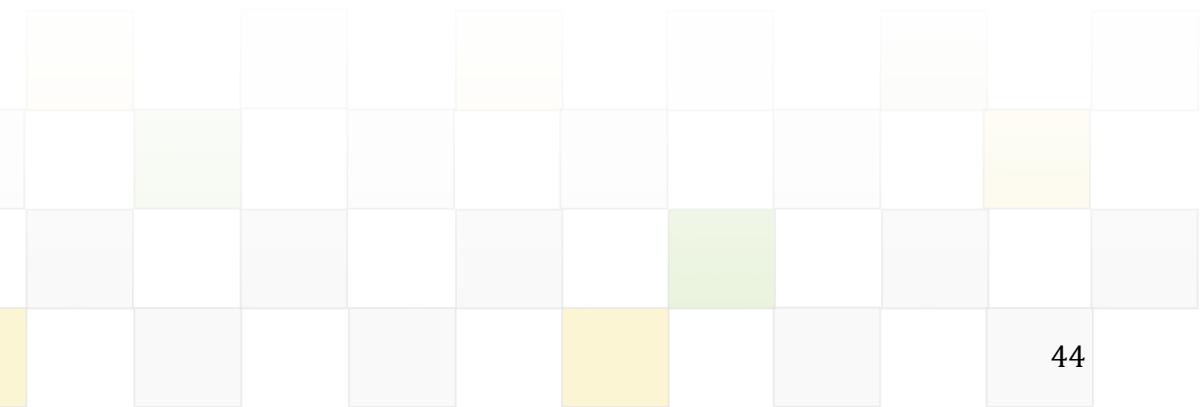
- 1. Ghi mẫu tin $\langle \text{start ckpt}(T_1, \dots, T_k) \rangle$ với T_1, \dots, T_k là những giao tác chưa hoàn tất và flush-log
- 2. Thực hiện ghi ĐVDL đang ở trên buffer mà chưa được ghi xuống đĩa của những giao tác đã COMMIT khi mẫu tin $\langle \text{start ckpt} \rangle$ được ghi xuống nhật ký.
- 3. Ghi mẫu tin $\langle \text{end ckpt} \rangle$ vào nhật ký và flush-log.
★ Lưu ý: Không cần đợi các giao tác hoàn tất T_1, \dots, T_k hoặc huỷ bỏ

Redo-Logging & Checkpoint (tt)

■ Ví dụ 1

```
<start T1>  
<T1, A, 5>  
<start T2>  
<commit T1>  
<T2, B, 10>  
<start ckpt (T2)>  
<T2, C, 15>  
<start T3>  
<T3, D, 20>  
<end ckpt>  
<commit T2>  
<commit T3>
```

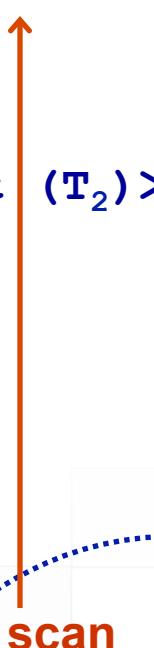
- T1 đã hoàn tất trước <start ckpt>
 - * Có thể đã được ghi xuống đĩa
 - * Nếu chưa thì trước khi <end ckpt> cũng được ghi xuống đĩa
- Sau <start ckpt>
 - * T2 đang thực thi
 - * T3 bắt đầu
- Sau <end ckpt>
 - * T2 và T3 hoàn tất



Redo-Logging & Checkpoint (tt)

■ Ví dụ 1

```
<start T1>  
<T1, A, 5>  
<start T2>  
<commit T1>  
<T2, B, 10>  
<start ckpt  
<T2, C, 15>  
<start T3>  
<T3, D, 20>  
<end ckpt>  
<commit T2>  
<commit T3>
```



- Tìm thấy <end ckpt>
- Chỉ xét T2 và T3
- <commit T2>
 - * Thực hiện lại T2
 - * Ghi C=15 và B=10
- <commit T3>
 - * Thực hiện lại T3
 - * Ghi D=20



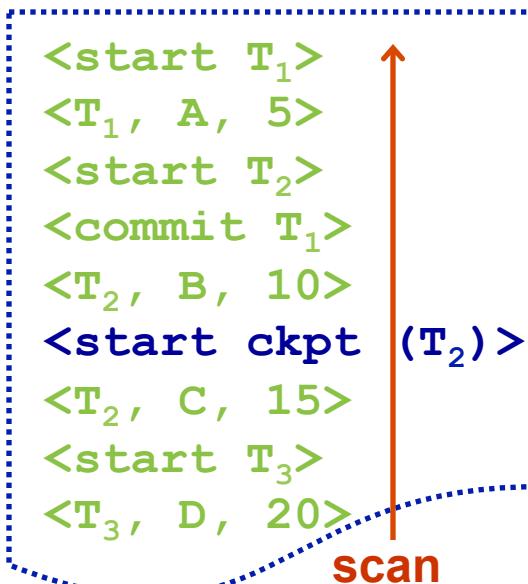
Redo-Logging & Checkpoint (tt)

■ Ví dụ 1b

```
<start T1>
<T1, A, 5>
<start T2>
<commit T1>
<T2, B, 10>
<start ckpt (T2)>
<T2, C, 15>
<start T3>
<T3, D, 20>
<end ckpt>
<commit T2>
```

Redo-Logging & Checkpoint (tt)

■ Ví dụ 2



- T2 và T3 chưa hoàn tất
 - * Không thực hiện lại
- T1 đã hoàn tất
 - * Thực hiện lại T1
 - * Ghi A=5



Nhận xét

■ Undo-logging

- Khi giao tác kết thúc, dữ liệu có thể được ghi xuống đĩa ngay lập tức
- Truy xuất đĩa nhiều nhưng không chiếm dụng nhiều bộ nhớ

→ Immediate modification

**UNDO: GHI DỮ LIỆU
XUỐNG ĐĨA TRƯỚC → GHI
COMMIT SAU**

■ Redo-logging

- Phải giữ lại các cập nhật trên vùng đệm cho đến khi giao tác hoàn tất và mẫu tin nhật ký `<commit T>` được ghi xuống đĩa
- Tốn nhiều bộ nhớ nhưng giảm tần suất truy xuất đĩa

→ Deferred modification

**REDO: GHI COMMIT
TRƯỚC → ĐƯA DỮ LIỆU
LÊN ĐĨA SAU**



Nội dung trình bày

■ An toàn dữ liệu

- Phân loại sự cố
- Nhật ký giao tác
- Điểm kiểm tra
- Undo logging
- Redo logging
- Undo / Redo logging**

■ An ninh dữ liệu

- Cơ chế phân quyền
- Cơ chế mã hoá



Phương pháp Undo/Redo-Logging

■ Qui tắc:

- **(1)** Khi một giao tác muốn ghi dữ liệu thì sẽ phát sinh ra một mẫu tin nhật ký tương ứng ghi nhận lại giá trị cũ và mới của ĐVDL X.
 - ★ Cấu trúc mẫu tin: $\langle T, X, v, w \rangle$ với v là giá trị cũ và w là giá trị mới
- **(2)** Trước khi X được cập nhật xuống đĩa, các mẫu tin cập nhật $\langle T, X, v, w \rangle$ đã phải có trên đĩa
 - Lệnh flush-log phải trước các lệnh OUTPUT
- **(3)** Khi T hoàn tất, tạo mẫu tin $\langle \text{commit } T \rangle$ trên nhật ký và ghi xuống đĩa



Ví dụ

Bước	Hành động	t	Mem A	Mem B	Disk A	Disk B	Mem Log
1							<start T>
2	Read(A,t)	8	8		8	8	
3	$t:=t^2$	16	8		8	8	
4	Write(A,t)	16	16		8	8	<T, A, 8, 16>
5	Read(B,t)	8	16	8	8	8	
6	$t:=t^2$	16	16	8	8	8	
7	Write(B,t)	16	16	16	8	8	<T, B, 8, 16>
8	Flush log						
9	Output(A)	16	16	16	16	8	
10							<commit T>
11	Output(B)	16	16	16	16	16	



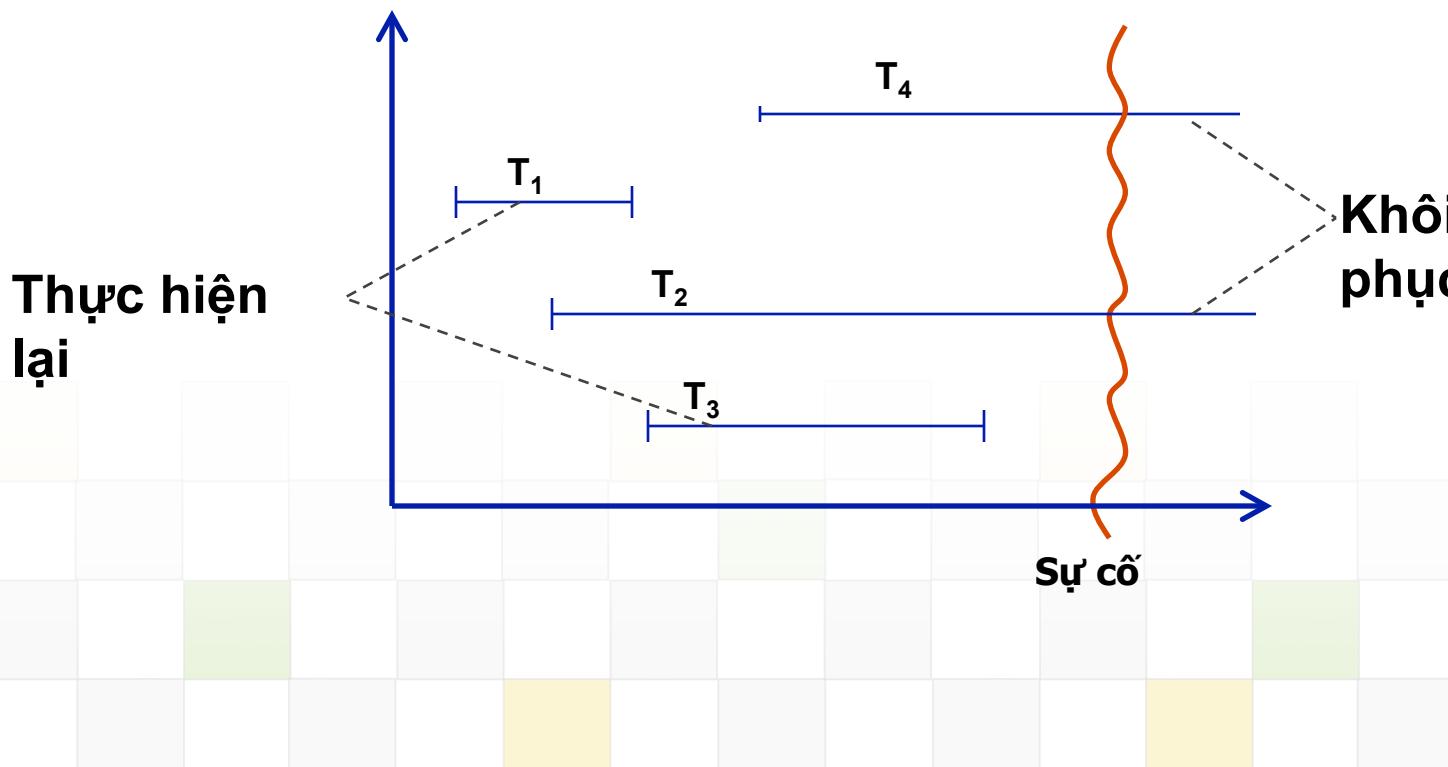
Phương pháp Undo/Redo-Logging (tt)

■ Khôi phục:

- (1) Khôi phục lại (undo) những giao tác chưa kết thúc
 - * Theo thứ tự từ cuối nhật ký đến đầu nhật ký
- (2) Thực hiện lại (redo) những giao tác đã hoàn tất
 - * Theo thứ tự từ đầu nhật ký đến cuối nhật ký

Phương pháp Undo/Redo-Logging (tt)

- Khi gặp sự cố
 - T1 và T3 đã hoàn tất
 - T2 và T4 chưa kết thúc



Ví dụ

Bước	Hành động	t	Mem A	Mem B	Disk A	Disk B	Mem Log
1							<start T>
2	Read(A,t)	8	8		8	8	
3	$t:=t*2$	16	8		8	8	
4	Write(A,t)	16	16		8	8	<T, A, 8, 16>
5	Read(B,t)	8	16	8	8	8	
6	$t:=t*2$	16	16	8	8	8	
7	Write(B,t)	16	16	16	8	8	<T, B, 8, 16>
8	Flush log						
9	Output(A)	16	16	16	16	8	
10							<commit T>
11	Output(B)	16	16	16	16	16	

T chưa kết thúc,
khôi phục A=8
<commit T> đã
được ghi xuống
đĩa, thực hiện lại T,
A=16 và B=16



Undo/Redo-Logging & Checkpoint

- Khi đến điểm lưu trữ, DBMS
 - (1) Tạo mẫu tin $\langle \text{start ckpt } (T_1, \dots, T_k) \rangle$ và ghi xuống đĩa
 - * T_1, \dots, T_k là những giao tác đang thực thi
 - (2) Ghi xuống đĩa những dữ liệu đang nằm trên vùng đệm
 - * Những đơn vị dữ liệu được cập nhật bởi các giao tác [Kể cả những giao tác đã COMMIT hay chưa COMMIT]
 - (3) Tạo mẫu tin $\langle \text{end ckpt} \rangle$ trong nhật ký và ghi xuống đĩa

Undo/Redo-Logging & Checkpoint (tt)

Ví dụ 1

```
<start T1>  
<T1, A, 4, 5>  
<start T2>  
<commit T1>  
<T2, B, 9, 10>  
<start ckpt (T2)>  
<T2, C, 14, 15>  
<start T3>  
<T3, D, 19, 20>  
<end ckpt>  
<commit T2>  
<commit T3>
```

- T₁ đã hoàn tất trước <start ckpt>
 - * Có thể đã được ghi xuống đĩa
 - * Nếu chưa thì trước khi <end ckpt> cũng được ghi xuống đĩa
- Giá trị B=10 đã được ghi xuống đĩa

Undo/Redo-Logging & Checkpoint (tt)

Ví dụ 1

```
<start T1>  
<T1, A, 4, 5>  
<start T2>  
<commit T1>  
<T2, B, 9, 10>  
<start ckpt (T2)>  
    ↑  
<T2, C, 14, 15>  
<start T3>  
<T3, D, 19, 20>  
<end ckpt>  
<commit T2>  
<commit T3>
```

scan

- Tìm thấy <end ckpt>
 - * T1 không cần thực hiện lại
- Xét T2 và T3
- <commit T2>
 - * Thực hiện lại T2 và ghi C=15
 - * Không cần ghi B
- <commit T3>
 - * Thực hiện lại T3 và ghi D=20

Undo/Redo-Logging & Checkpoint (tt)

Ví dụ 2

```
<start T1>  
<T1, A, 4, 5>  
<start T2>  
<commit T1>  
<T2, B, 9, 10>  
<start ckpt (T2)>  
    ↑  
<T2, C, 14, 15>  
<start T3>  
<T3, D, 19, 20>  
<end ckpt>  
<commit T2>
```

scan

- Tìm thấy <end ckpt>
 - * T1 không cần thực hiện lại
- Xét T2 và T3
- <commit T2>
 - * Thực hiện lại T2 và ghi C=15
 - * Không cần ghi B
- T3 chưa kết thúc
 - * Khôi phục D=19

Undo/Redo-Logging & Checkpoint (tt)

Ví dụ 3

```
<start T1>  
<T1, A, 4, 5>  
<start T2>  
<commit T1>  
<start T3>  
<T2, B, 9, 10>  
<T3, E, 6, 7>  
<start ckpt (T2, T3)>  
<T2, C, 14, 15>  
<T3, D, 19, 20>  
<end ckpt>  
<commit T2>
```

scan

- Tìm thấy **<end ckpt>**
 - * T1 không cần thực hiện lại
- Xét T2 và T3
- **<commit T2>**
 - * Thực hiện lại T2 và ghi C=15
 - * Không cần ghi B
- T3 chưa kết thúc
 - * Khôi phục D=19 và E=6



Nội dung trình bày

■ An toàn dữ liệu

- Phân loại sự cố
- Nhật ký giao tác
- Điểm kiểm tra
- Undo logging
- Redo logging
- Undo / Redo logging

■ An ninh dữ liệu

- Cơ chế phân quyền
- Cơ chế mã hoá



An ninh Dữ liệu

- Thực hiện hai bài toán :
 - Bài toán phân quyền
 - * Quản lý tốt việc truy xuất Dữ liệu của các đối tượng người dùng hợp pháp → Bảo mật Dữ liệu
 - * Thông qua 2 cơ chế
 - Cơ chế chứng thực
 - Cơ chế phân quyền
 - » Quan điểm phân quyền cụ thể
 - » Quan điểm phân cấp mức độ MẬT
 - Bài toán mã hóa
 - * Ngăn chặn hiệu quả sự tấn công, xâm nhập của các đối tượng tin tặc → An ninh Dữ liệu



Cơ chế chứng thực

- Mỗi người dùng DBMS được xác định bởi
 - Một tên đăng nhập – user name
 - Một mật mã đăng nhập – password
- Thông tin về user name và password
 - Không được lưu trữ tường minh trong dữ liệu
 - User name và password của DBMS và của OS có thể tách bạch nhau hay dùng chung cho nhau là tùy hệ thống
 - ★ Vd : Mixed-mode của Microsoft SQL Server



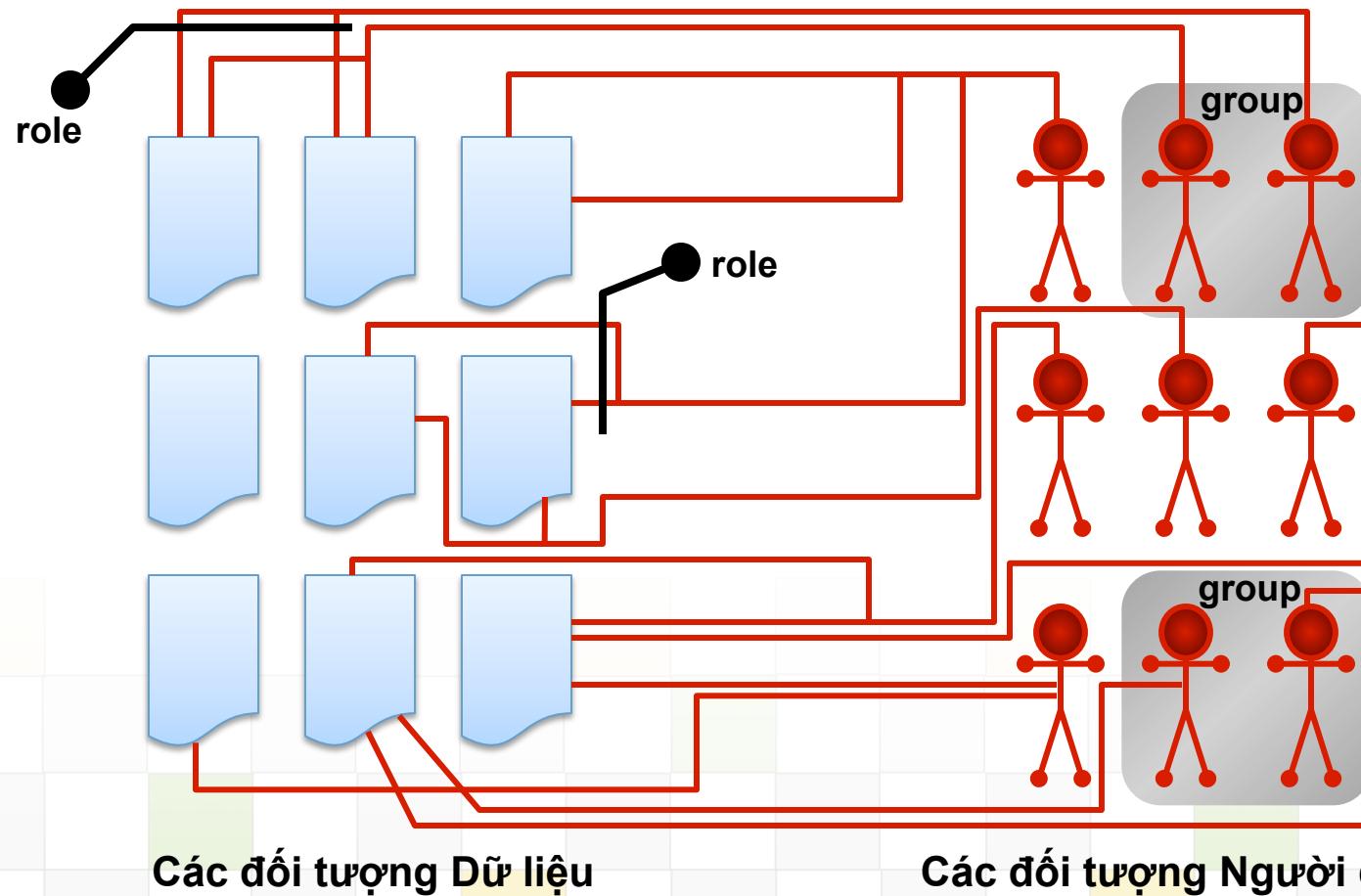
Cơ chế phân quyền

- Một tài khoản chứng thực
 - Được phép đăng nhập vào hệ thống DBMS
 - Được nhìn thấy các CSDL
 - Chưa được phép truy xuất các đối tượng trong các CSDL
- Tài khoản chứng thực muốn truy xuất các đối tượng dữ liệu thì cần được phân quyền cụ thể chi tiết trên các đối tượng dữ liệu đó



Cơ chế phân quyền (tt)

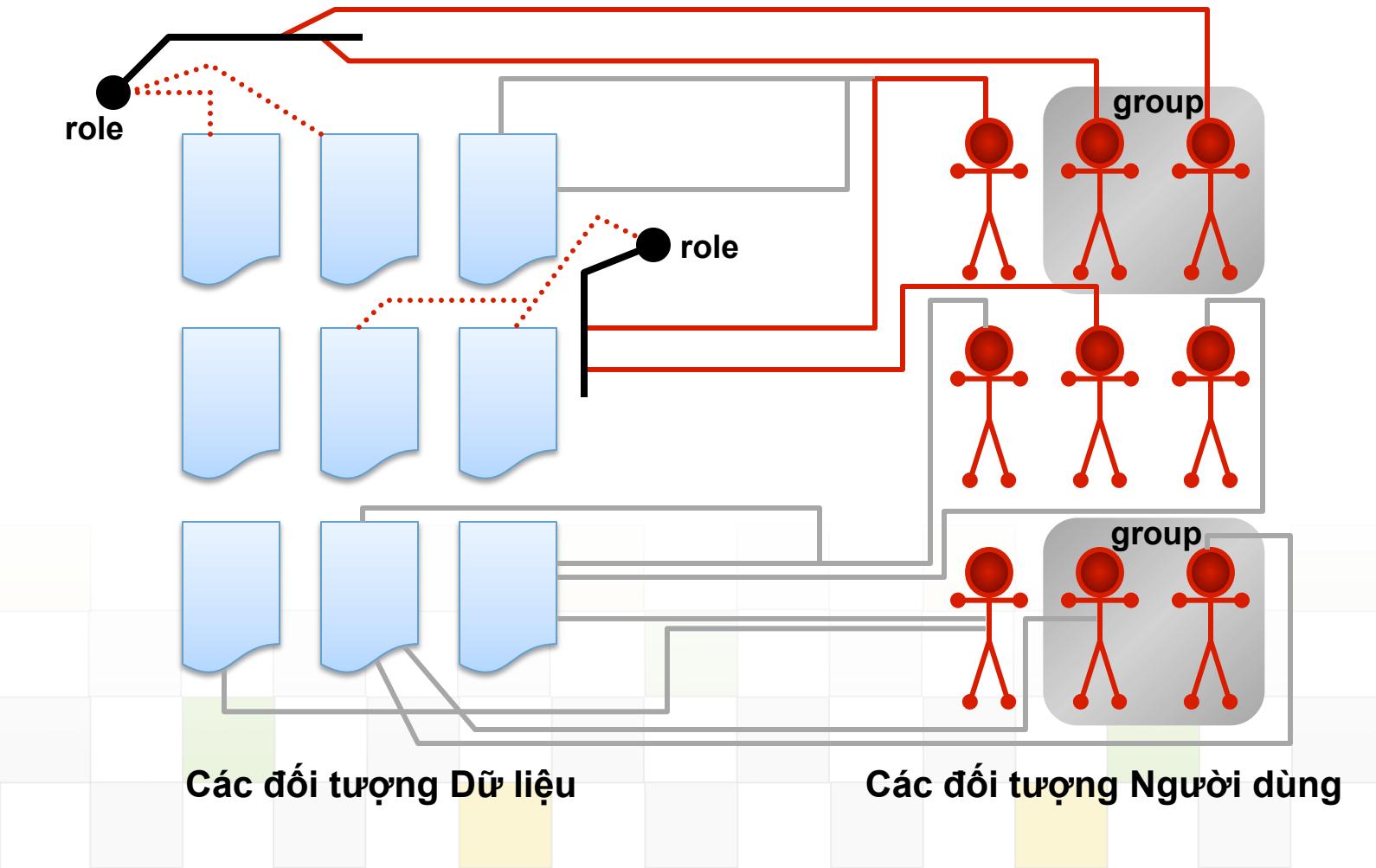
- Quan điểm phân quyền cụ thể





Cơ chế phân quyền (tt)

- Quan điểm phân quyền cụ thể





Cơ chế phân quyền (tt)

- Quan điểm phân cấp mức độ MẬT
 - Các đối tượng Dữ liệu được phân ra các cấp độ bảo mật khác nhau
 - * Vd :
 - Cấp 3 : Dành cho tài liệu tuyệt mật
 - Cấp 2 : Dành cho tài liệu mật
 - Cấp 1 : Dành cho tài liệu công khai
 - Các đối tượng Người dùng cũng được phân ra các cấp độ bảo mật khác nhau
 - * Vd :
 - Cấp 3 : Dành cho ban giám đốc
 - Cấp 2 : Dành cho các trưởng phòng
 - Cấp 1 : Dành cho nhân viên
 - Khó khăn : Làm sao phân cấp cho hợp lý (♣)



Cơ chế phân quyền (tt)

- Quan điểm phân cấp mức độ MẬT
 - Phân quyền
 - * Quyền đọc dữ liệu : Người dùng cấp i được đọc các tài liệu cấp i trở xuống
 - * Quyền ghi dữ liệu : ( )
 - Ban giám đốc đọc các tài liệu mật nhưng tài liệu ấy không nhất định do họ tạo ra, thông thường lại do nhân viên tạo ra
 - Người dùng cấp i được ghi tài liệu cấp i trở xuống
 - Nếu người dùng X thuộc cấp i tạo ra tài liệu A thuộc cấp j (với $j > i$) thì chỉ có X được đọc A trong khi các X' cùng cấp không được đọc A
 - Vì ( ) và ( ) nên quan điểm này gặp nhiều thách thức và ít được ứng dụng trong các DBMS thương mại



Cơ chế mã hóa

- Bất chấp cơ chế phân quyền, nhiều đối tượng người dùng bất hợp pháp vẫn có thể xâm nhập vào CSDL
 - Ví dụ :
 - ★ Thâm nhập từ mức Hệ điều hành để chép các file dữ liệu của DBMS (như file *.mdf và *.ndf của SQL Server)
 - ★ Chặn trên đường truyền mạng để hứng lấy dữ liệu luân chuyển giữa Client và Server
- Giải pháp : Mã hóa thông tin trước lưu trữ hoặc truyền trên đường truyền
 - Tin tặc lấy được file hay dữ liệu cũng không hiểu được
 - Việc mã hóa không được xung đột với hệ thống index → thách thức
 - Thuật toán mã hóa được chọn sao cho việc giải mã của tin tặc là khó khăn nhất



Tóm tắt chương 4

■ An toàn dữ liệu

- Phân loại sự cố
- Nhật ký giao tác
- Điểm kiểm tra
- Phương pháp Undo logging
- Phương pháp Redo logging
- Phương pháp Undo / Redo logging

■ An ninh dữ liệu

- Cơ chế phân quyền
- Cơ chế mã hoá



Tóm tắt chương 4

- Các thuật ngữ:
 - Transaction Management
 - Database elements
 - Logging
 - Recovery
 - Logging methods
 - Undo logging
 - Redo logging
 - Undo/Redo logging
 - Checkpointing
 - Nonquiescent checkpointing
 - Archiving
 - Incremental Backups
 - Nonquiescent Archiving
 - Recovery from media failures



Tài liệu tham khảo

- [5] *Database systems: the complete book*, Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom, Pearson Prentice Hall, 2009
 - Chapter 17. **Copy with System failures**



Backup & Recovery in SQL Server

- Tham khảo thêm:
 - Understanding Logging and Recovery in SQL Server
 - ★ <http://technet.microsoft.com/en-us/magazine/2009.02.logging.aspx>
 - SQL Server Recovery Models
 - ★ <http://databases.about.com/od/sqlserver/a/recoverymodels.htm>
 - Restore your SQL Server database using transaction logs
 - ★ <http://www.techrepublic.com/blog/the-enterprise-cloud/restore-your-sql-server-database-using-transaction-logs/>
- Keywords: How SQL Server logging and recovery

Q & A



HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

Chương 5:
**XỬ LÝ CÂU TRUY
VĂN**

GVLT: *Nguyễn Trường Sơn*



Nội dung chi tiết

- Giới thiệu
- Phân tích cú pháp - ngũ nghĩa
- Biến đổi sang Đại số Quan hệ
- Tối ưu hóa cây truy vấn
- Ước lượng kích thước cây truy vấn
- Phát sinh và thực thi mã lệnh



Giới thiệu

- Xét hai quan hệ R và S nhu sau :
 - $R(A, B, C)$
 - $S(C, D, E)$
- Xét câu truy vấn sau đây trên R và S

```
SELECT R.B, S.D  
FROM R, S  
WHERE R.A= 'c' And S.E=2 And R.C=S.C
```

- Nhận xét
 - Một câu truy vấn có rất nhiều cách thực hiện
 - Tùy trường hợp mà các cách thực hiện được đánh giá là tốt hay dở



Giới thiệu (tt)

■ Xử lý của DBMS

- Cách 1:

$$\Pi_{B,D} [\sigma_{R.A = 'c' \wedge S.E=2 \wedge R.C = S.C} (R \times S)]$$

- Cách 2:

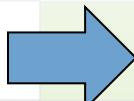
$$\Pi_{B,D} [\sigma_{R.A = 'c'} (R) \bowtie \sigma_{S.E=2} (S)]$$

- Cách 3: Sử dụng chỉ mục trên R.A và S.C

- Tìm các bộ trong R thỏa R.A= ‘c’
- Với mỗi bộ tìm thấy, tìm tiếp các bộ trong S thỏa R.C=S.C
- Bỏ đi những bộ S.E ≠ 2
- Chiếu trên thuộc tính B và D

■ DBMS chọn cách nào ?

Mục tiêu chương:

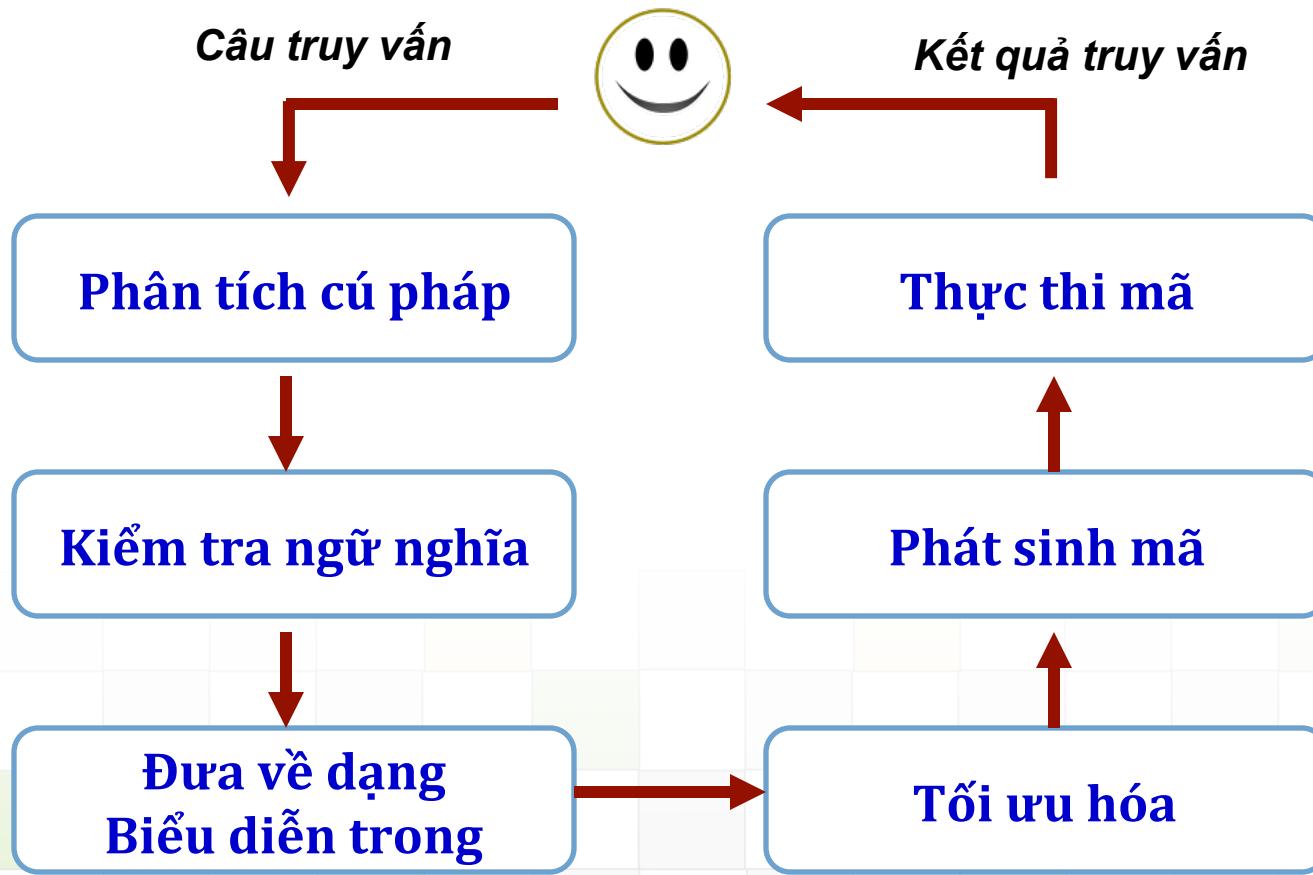


Tập trung vào xử lý truy vấn của
RDBMS



Giới thiệu (tt)

- Quy trình xử lý câu truy vấn

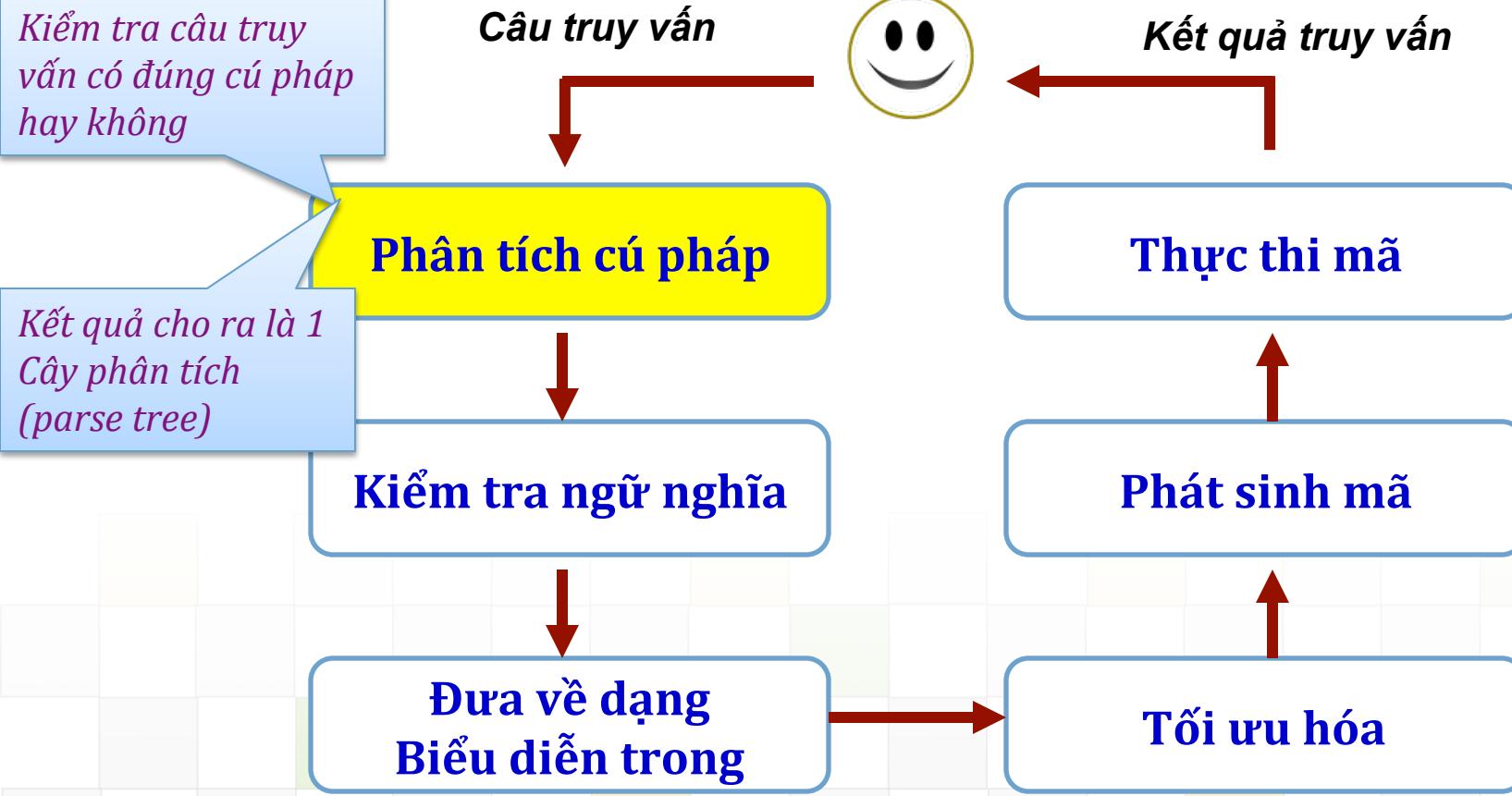




Nội dung chi tiết

- Giới thiệu
- **Phân tích cú pháp - ngữ nghĩa**
- Biến đổi sang Đại số Quan hệ
- Tối ưu hóa cây truy vấn
- Ước lượng kích thước cây truy vấn
- Phát sinh và thực thi mã lệnh

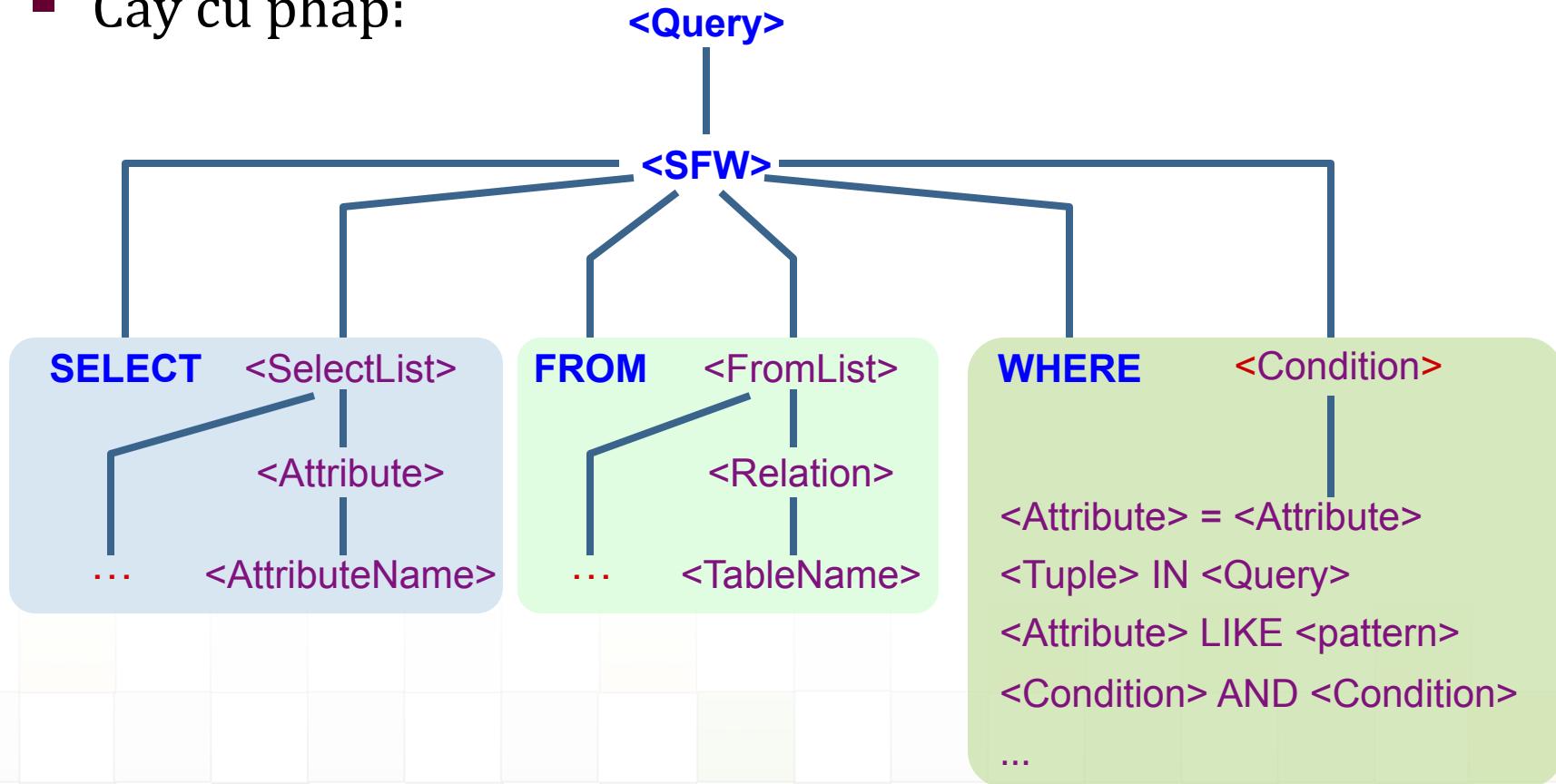
Phân tích cú pháp và ngữ nghĩa





Phân tích cú pháp và ngữ nghĩa (tt)

- Cây cú pháp:



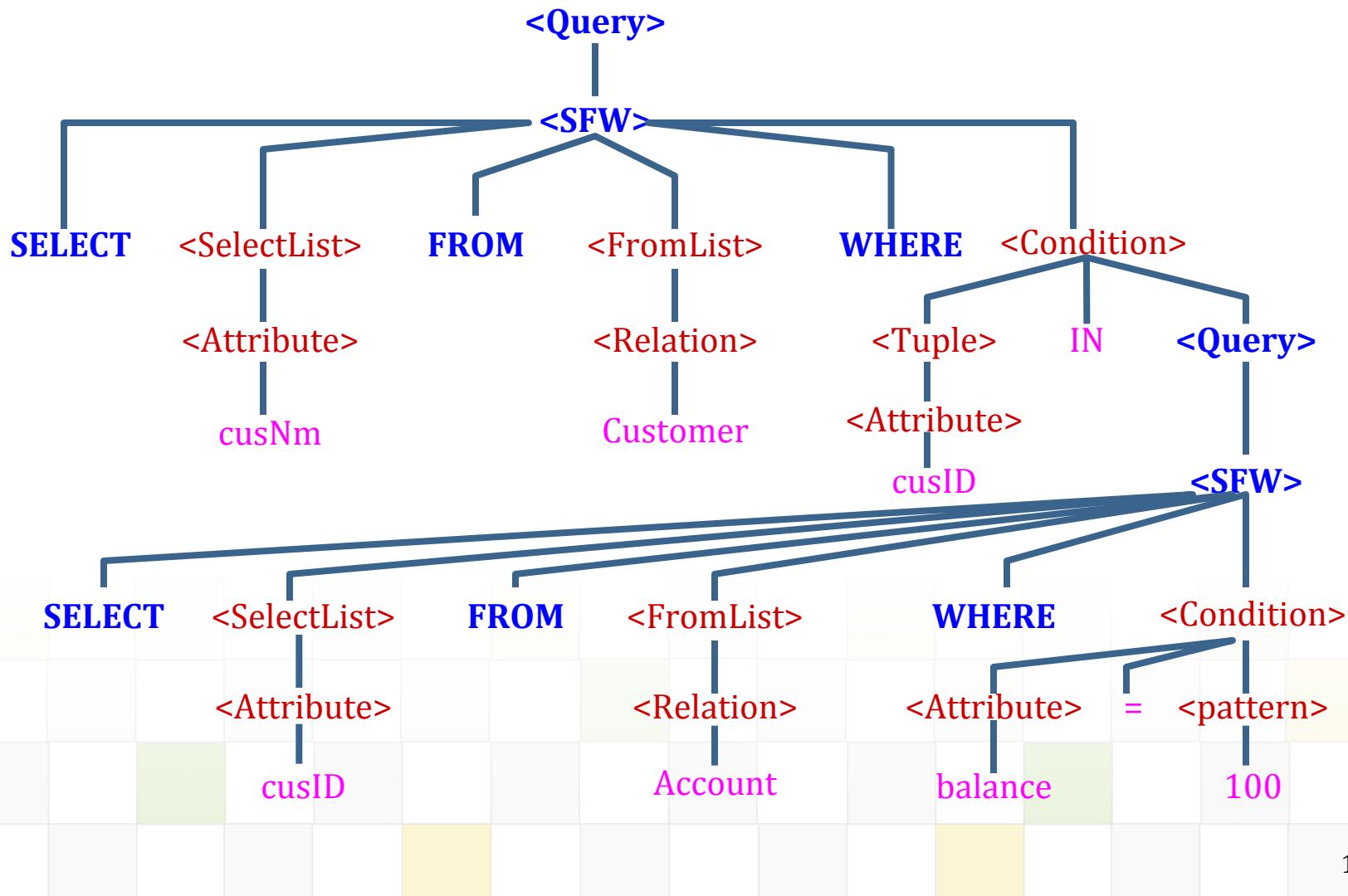


Ví dụ 1

- Xét hai quan hệ sau :
 - Customer(cusID, cusNm, cusStreet, cusCity)
 - Account(accID, cusID, balance)
- Và câu truy vấn

```
SELECT cusNm  
FROM Customer  
WHERE cusID IN (  
    SELECT cusID  
    FROM Account  
    WHERE balance = 100)
```

Ví dụ 1 (tt)



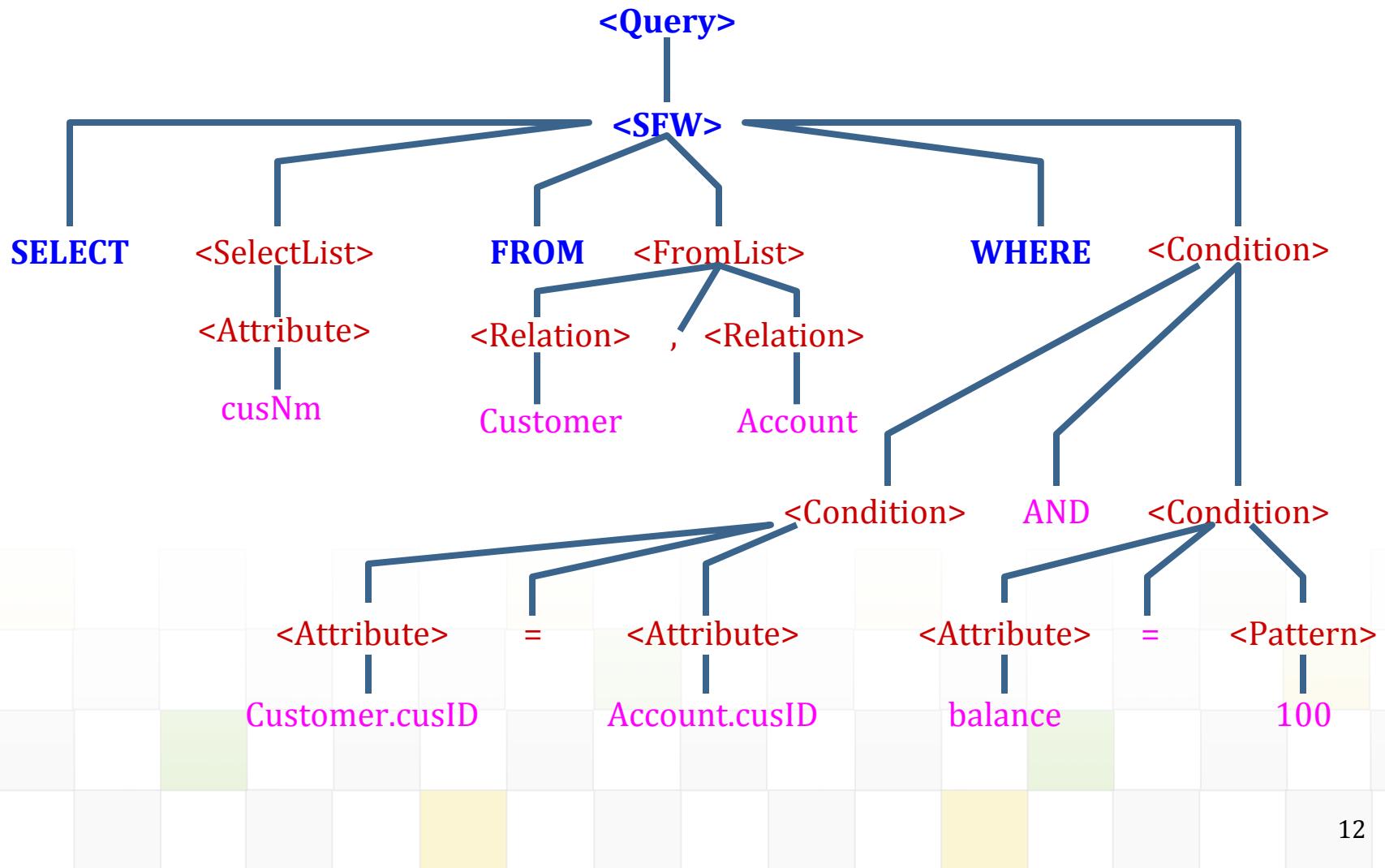


Ví dụ 2

- Xét hai quan hệ sau đây :
 - Customer(cusID, cusNm, cusStreet, cusCity)
 - Account(accID, cusID, balance)
- Và câu truy vấn sau:

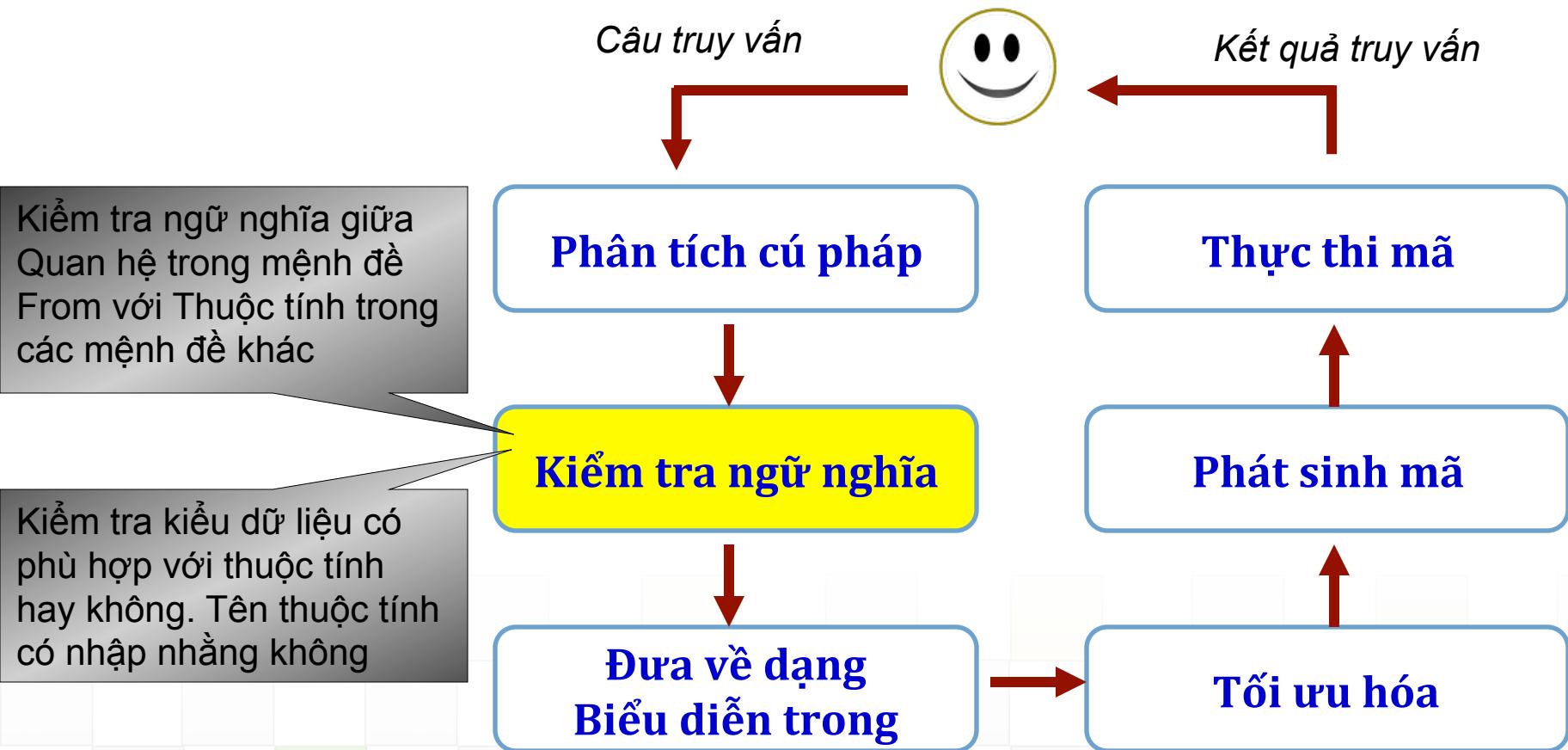
```
SELECT cusNm  
FROM Customer, Account  
WHERE Customer.cusID = Account.cusID  
AND balance = 100
```

Ví dụ 2 (tt)





Phân tích cú pháp và ngữ nghĩa



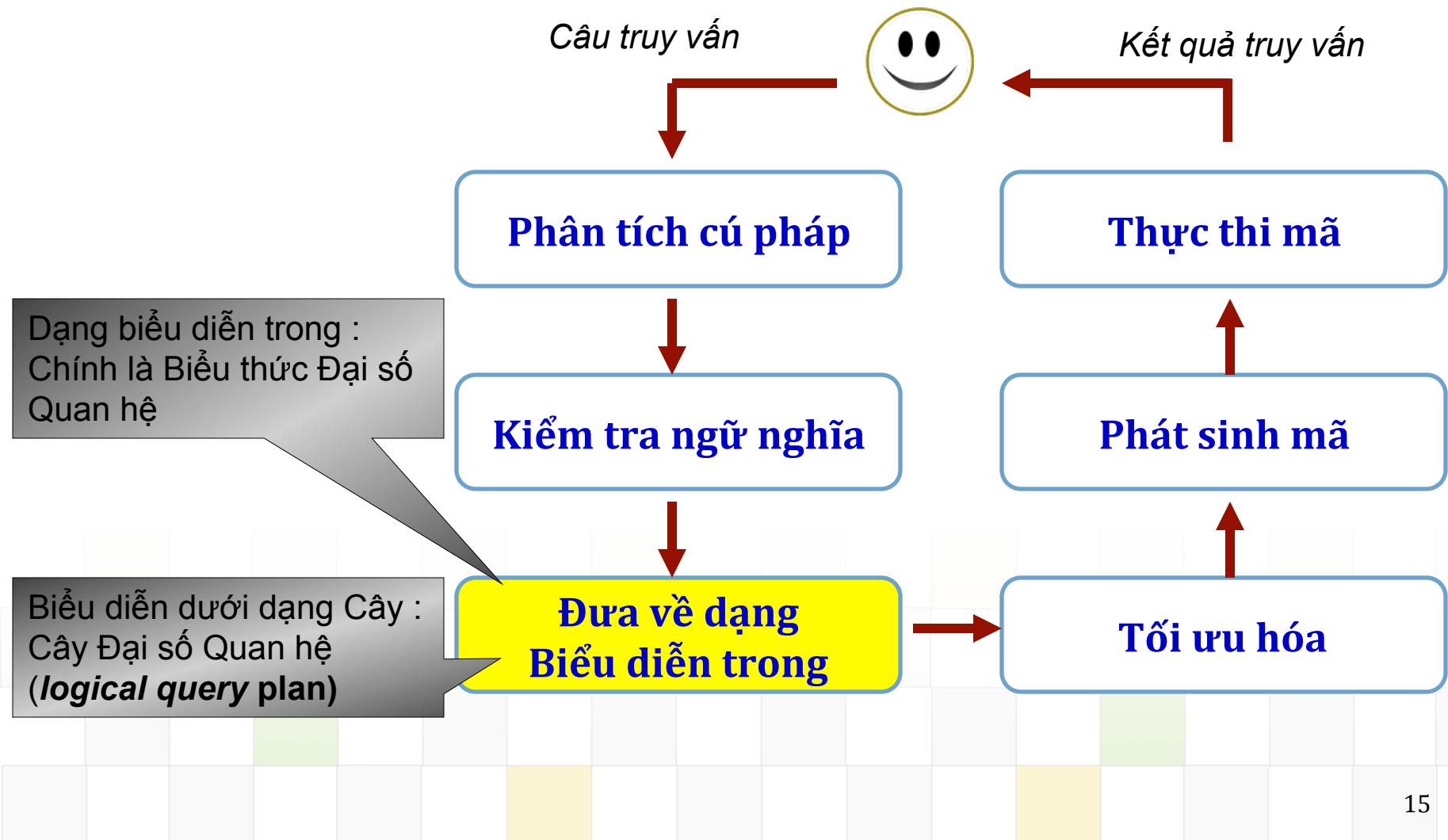


Nội dung chi tiết

- Giới thiệu
- Phân tích cú pháp - ngữ nghĩa
- **Biến đổi sang Đại số Quan hệ**

- Tối ưu hóa cây truy vấn
- Ước lượng kích thước cây truy vấn
- Phát sinh và thực thi mã lệnh

Biến đổi sang ĐSQH



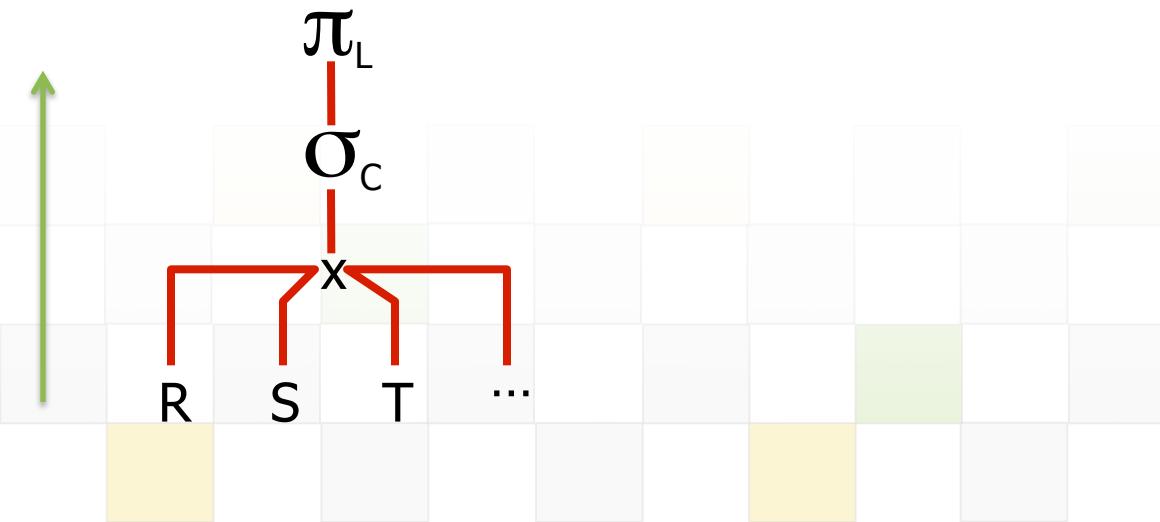
- Câu truy vấn được phân rã thành các query block (QB).
 - Query Block là đơn vị cơ bản để có thể chuyển sang các biểu thức ĐSQH và tối ưu hoá
 - Một QB chứa một biểu thức đơn SELECT- FROM-WHERE-GROUP BY - HAVING
 - Các câu truy vấn lồng trong 1 câu truy vấn là các QB độc lập.
 - Các toán tử gom nhóm (max, min, sum, count) được thể hiện dùng ĐSQH mở rộng.
 - Mỗi câu truy vấn được biểu diễn sang dạng ĐSQH dạng biểu thức hoặc cây truy vấn (query tree)



Biến đổi sang ĐSQH (tt)

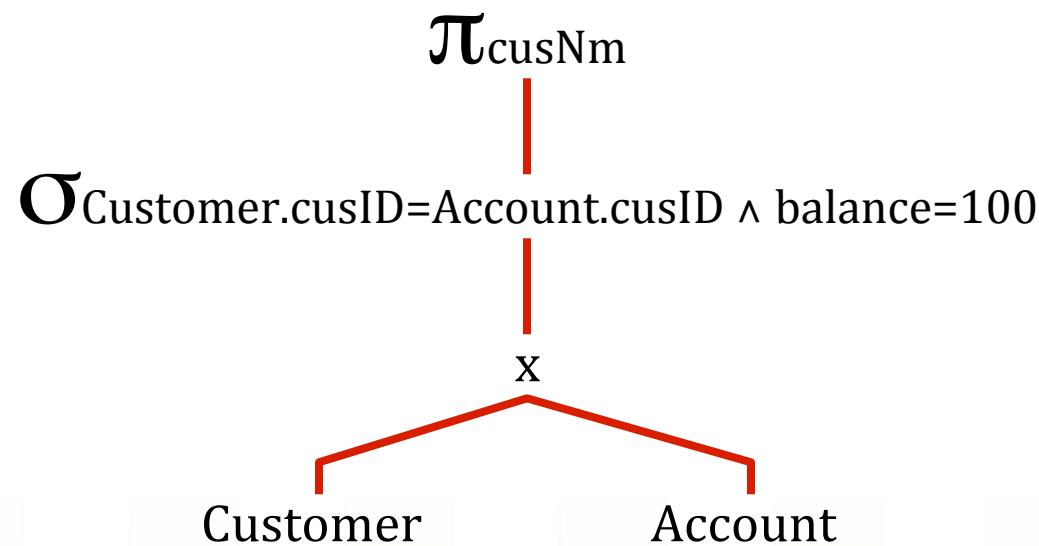
■ Truy vấn đơn:

- Xét câu trúc **<SFW>**, sử dụng quy tắc **<SFW>**
 - Thay thế **<FromList>** thành các biến quan hệ
 - Sử dụng phép tích cartesian (X) cho các biến quan hệ
 - Thay thế **<Condition>** thành phép chọn σ_c
 - Thay thế **<SelectList>** thành phép chiếu π_L
- Kết quả là một Cây truy vấn





Xét ví dụ 2



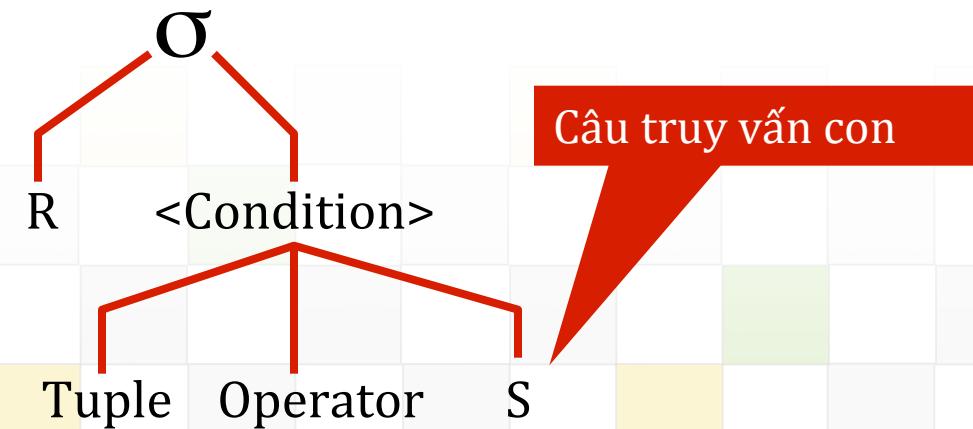


Biến đổi sang ĐSQH (tt)

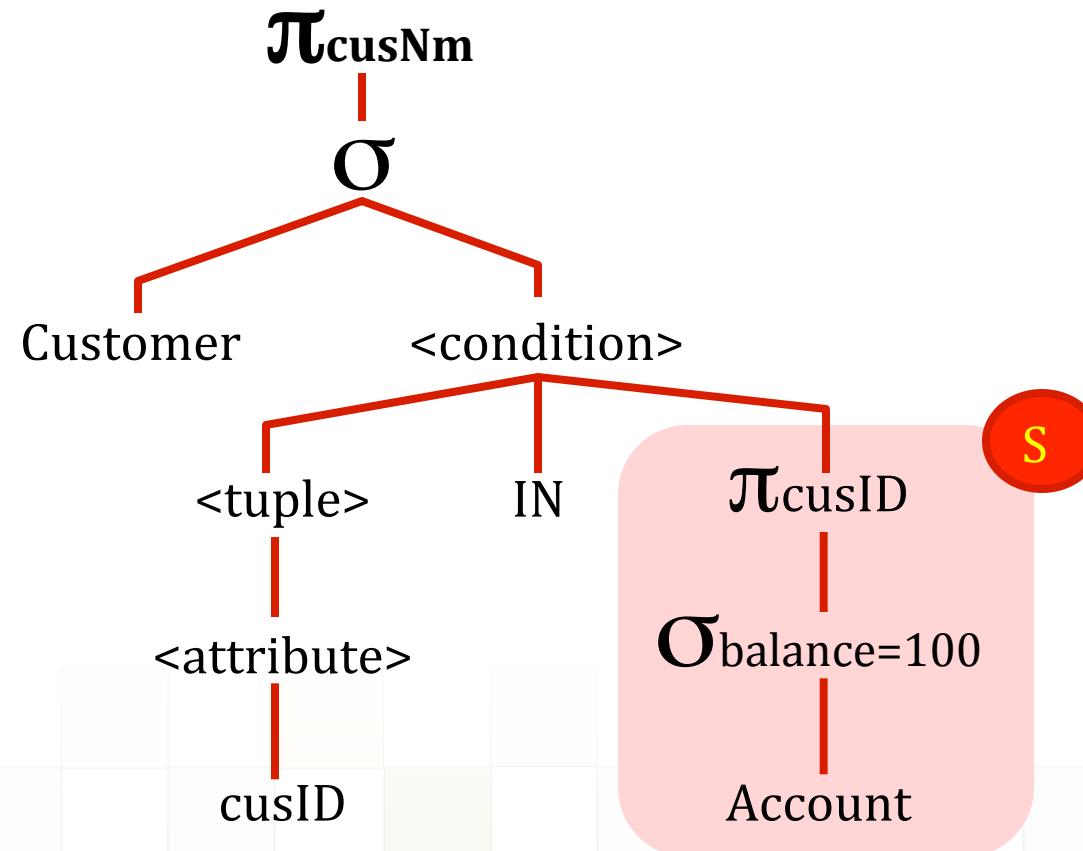
■ Truy vấn lồng:

Tồn tại câu truy vấn con S trong *<Condition>*

- Áp dụng qui tắc **<SFW>** cho truy vấn con **S**
- Sử dụng *phép chọn 2 biến* (two-argument selection)
 - Nút là phép chọn không có tham số
 - Nhánh con trái là biến quan hệ R
 - Nhánh con phải là *<condition>* áp dụng cho **mỗi bộ trong R**



Xét ví dụ 1 (Lồng phân cấp)



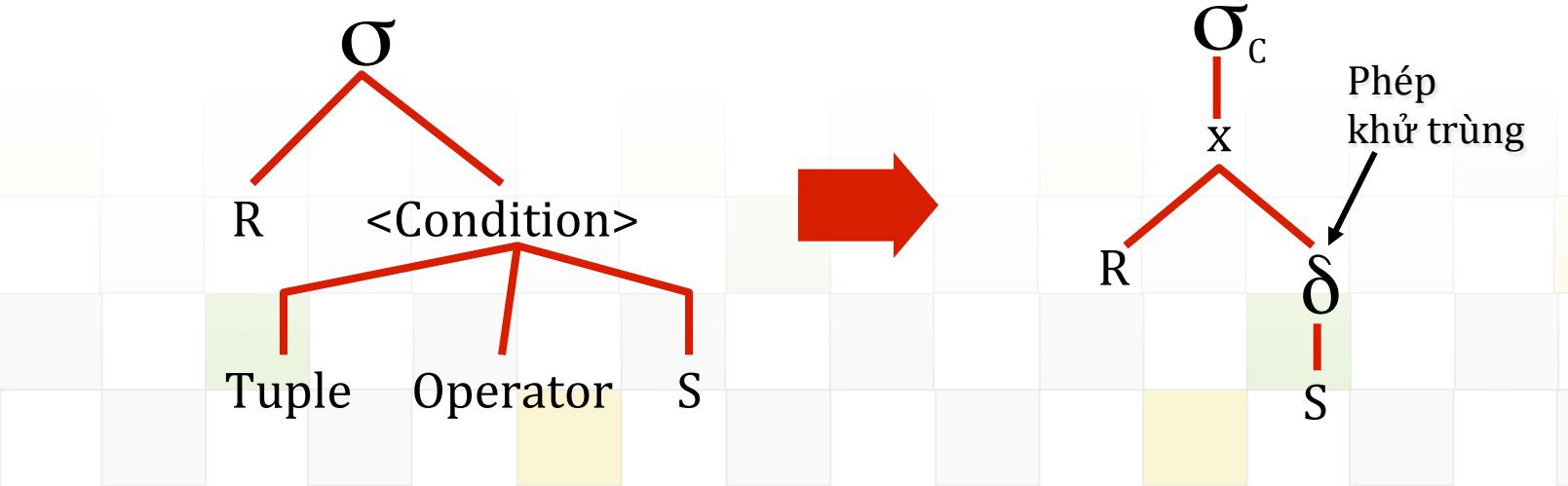
Biến đổi sang ĐSQH (tt)



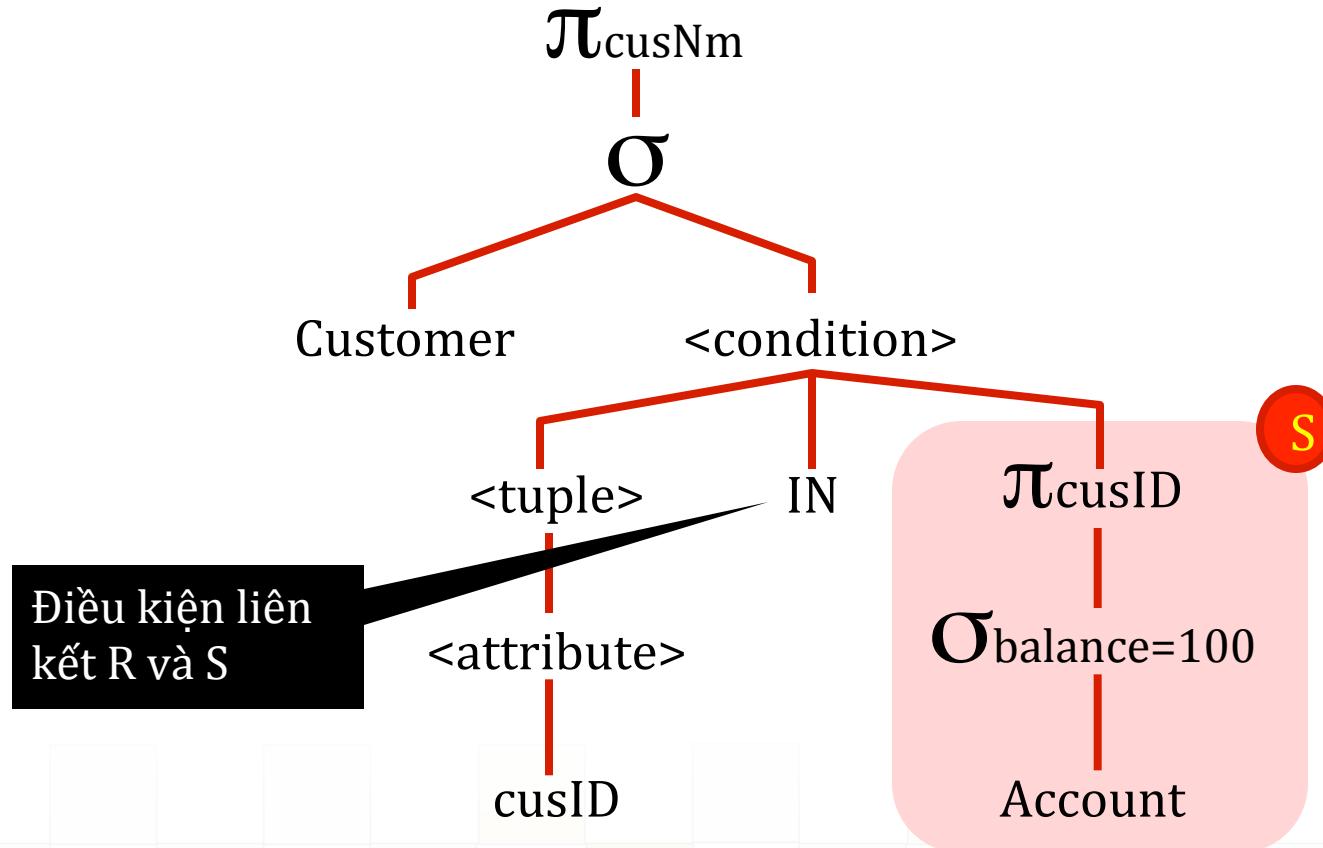
Truy vấn lồng:

Biến đổi phép chọn 2 biến

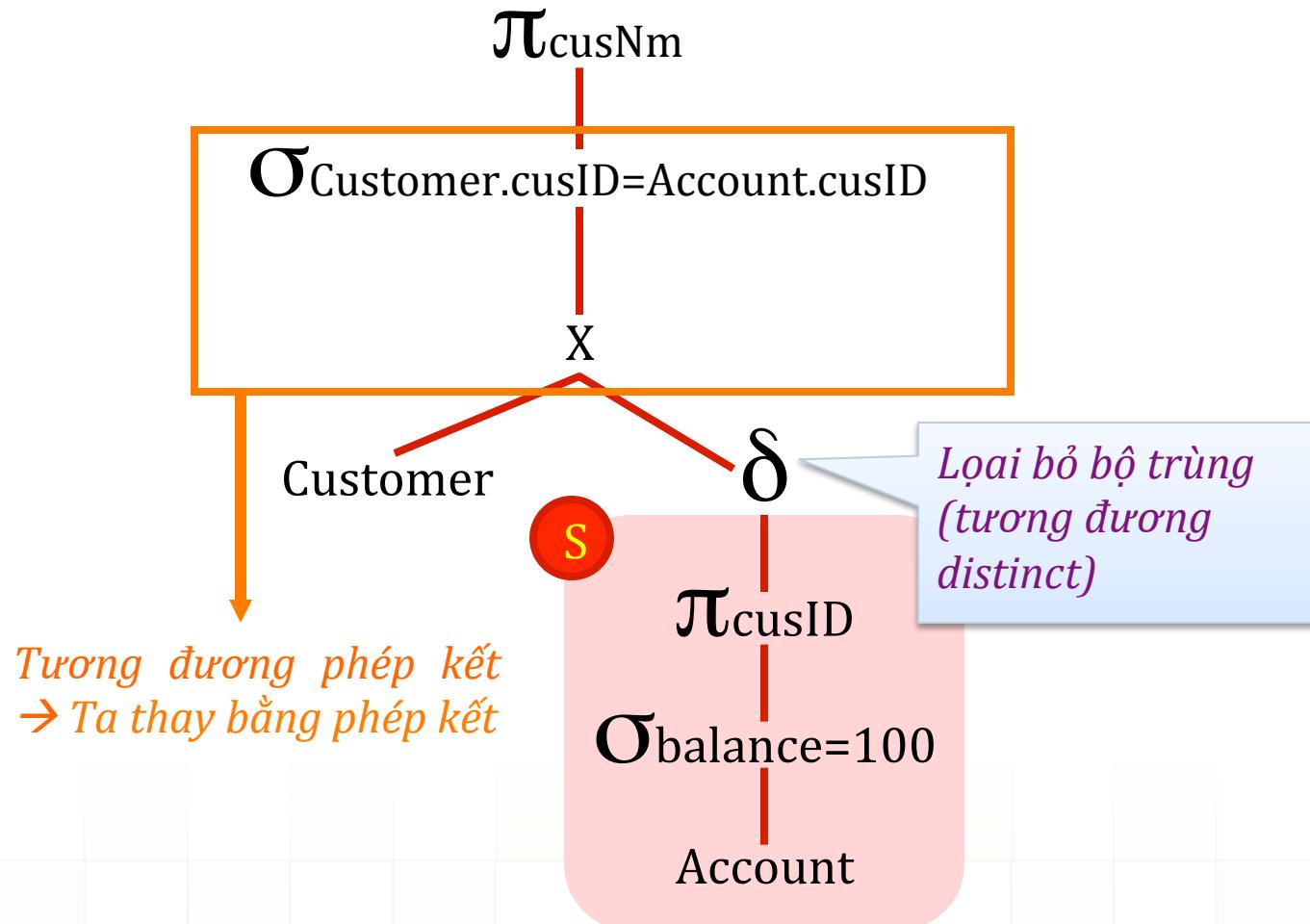
- Thay thế $\langle \text{Condition} \rangle$ bằng 1 cây có ngọn là S
 - Nếu S có các bộ trùng nhau thì phải lược bỏ bớt bộ trùng nhau đi. Sử dụng phép δ để lược bỏ (giống Distinct)
- Thay thế phép chọn 2 biến thành σ_C với C là điều kiện liên kết (không đơn thuần là kết) R với S
- σ_C làm trên kết quả của phép cartesian của R và S



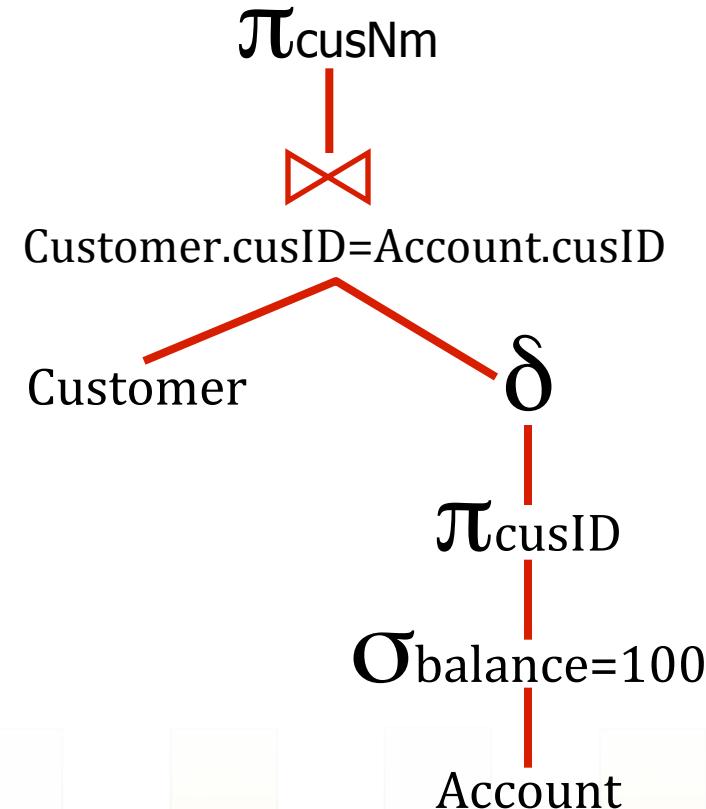
Xét ví dụ 1 (Lồng phân cấp)



Xét ví dụ 1 (tt)



Xét ví dụ 1 (tt)





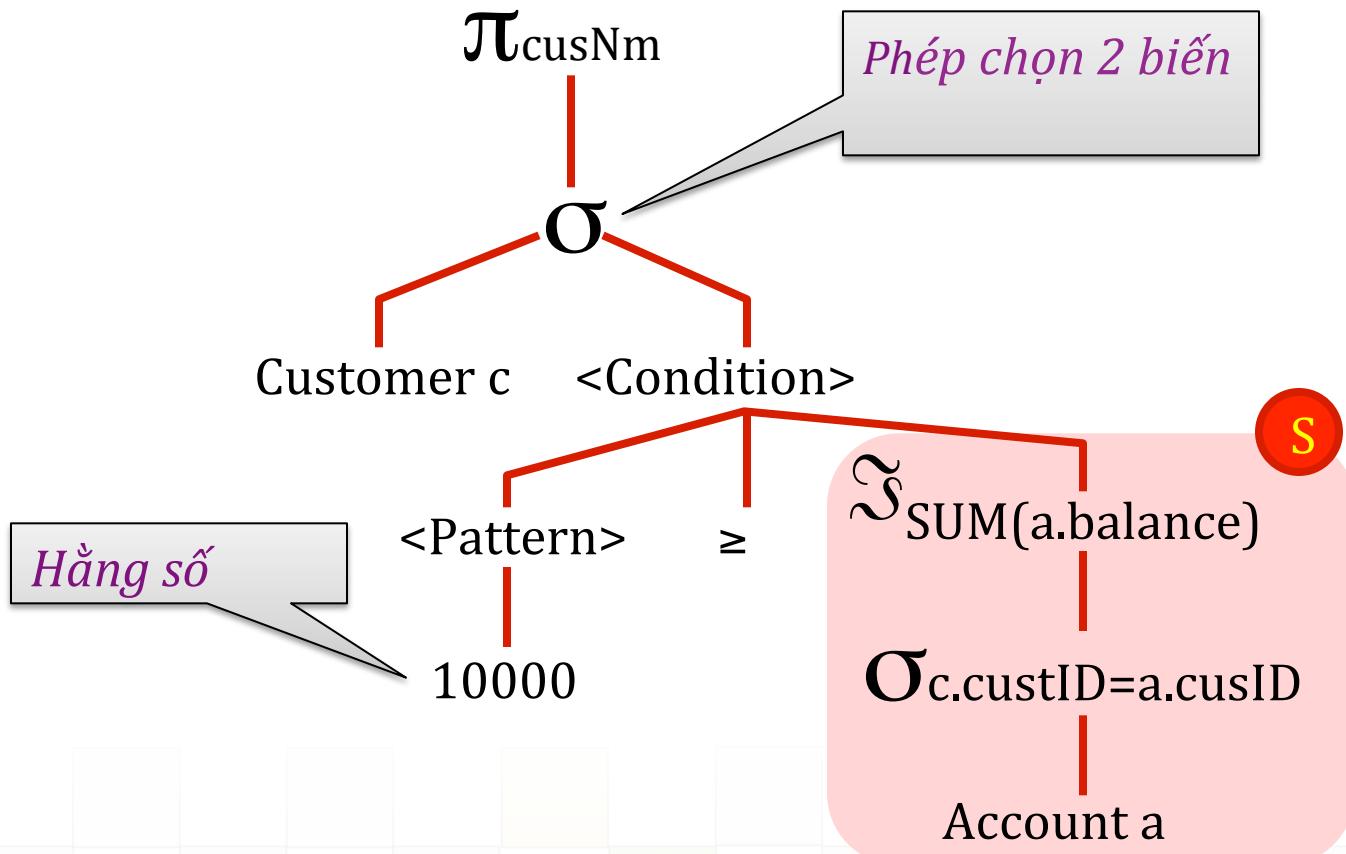
Ví dụ 3 (Lòng tương quan)

- Xét hai quan hệ sau đây :
 - Customer(cusID, cusNm, cusStreet, cusCity)
 - Account(accID, cusID, balance)
- Xét câu truy vấn sau đây :

```
SELECT c.cusNm  
FROM Customer c  
WHERE 10000 >= (  
    SELECT SUM(a.balance)  
    FROM Account a  
    WHERE a.cusID=c.cusID)
```

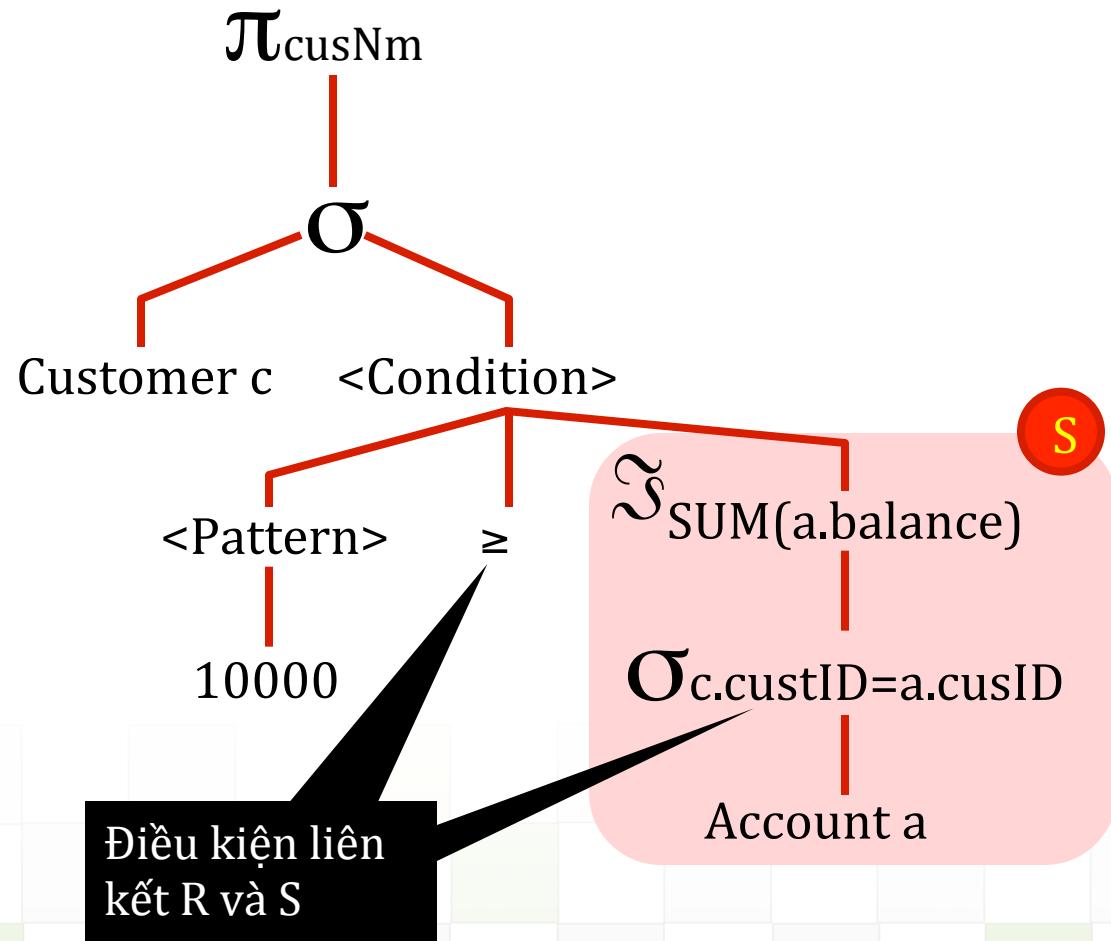


Ví dụ 3 (tt)



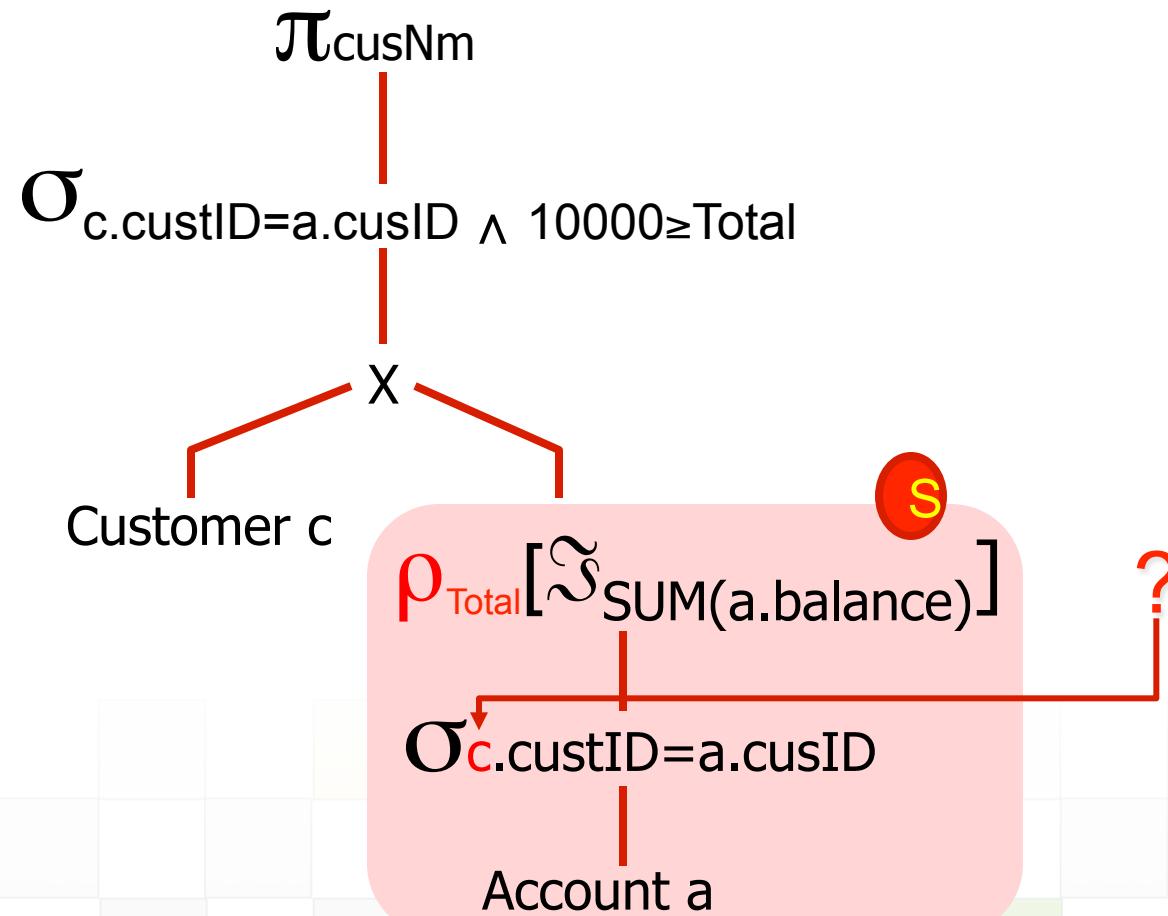


Ví dụ 3 (tt)



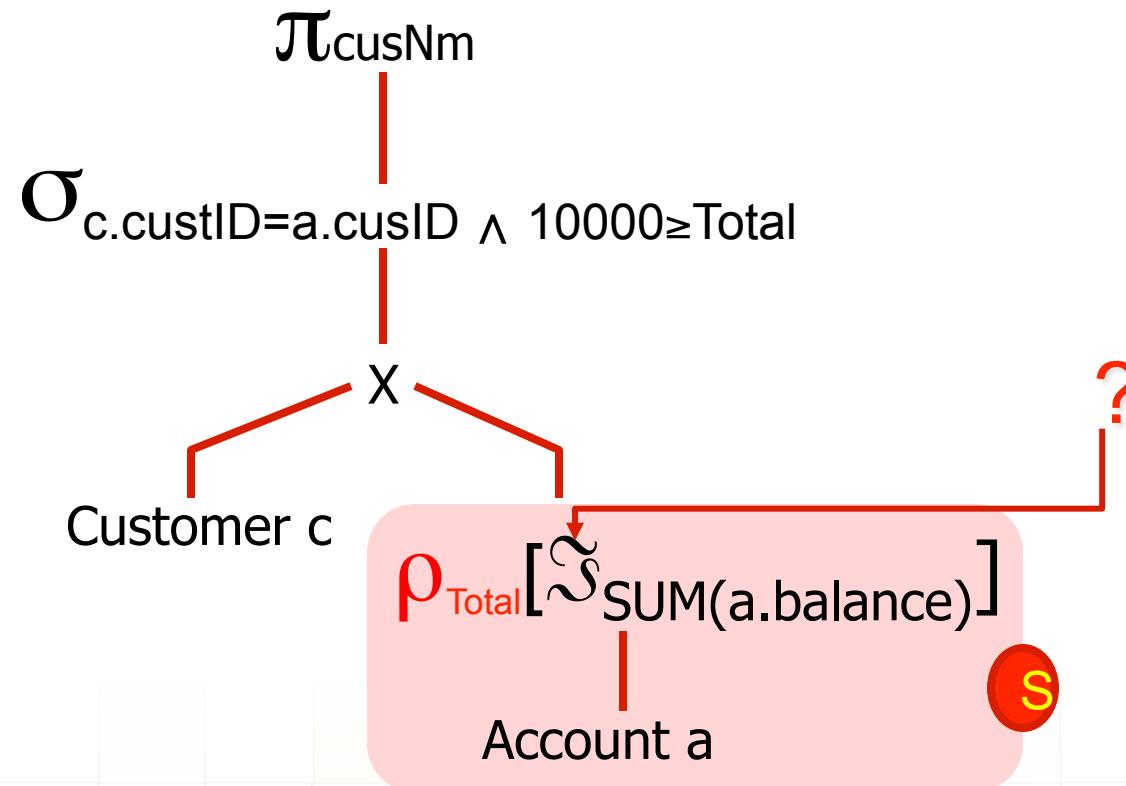


Ví dụ 3 (tt)



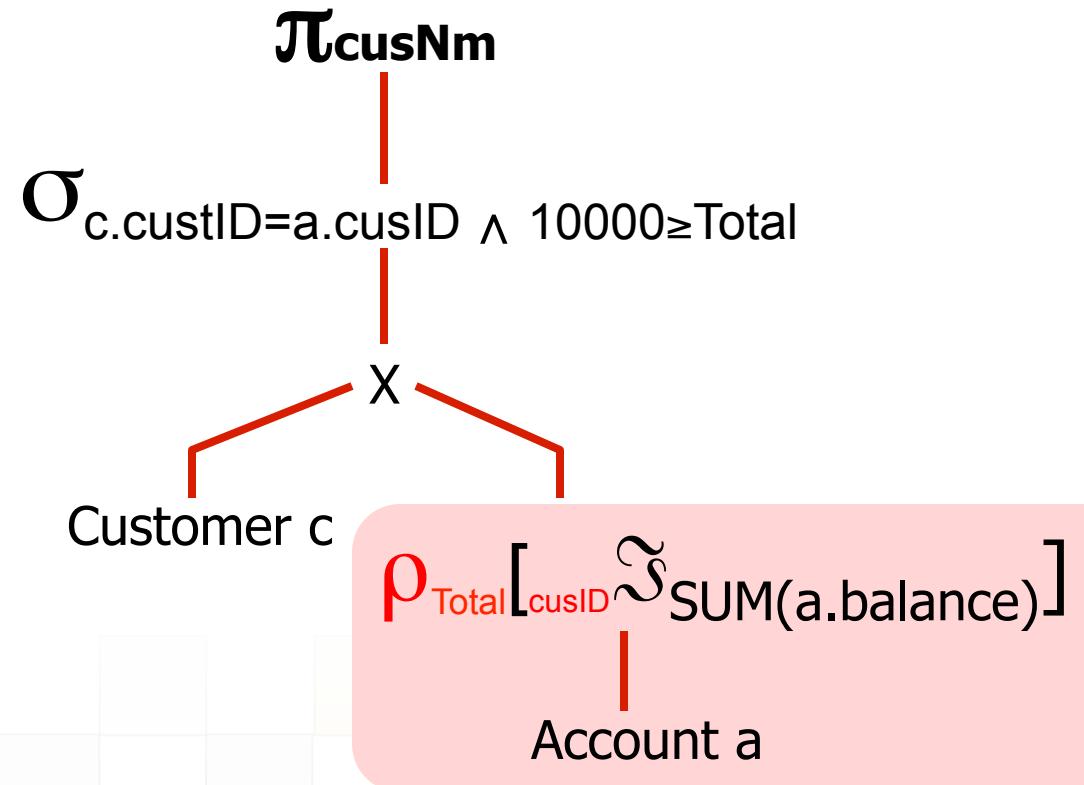


Ví dụ 3 (tt)





Ví dụ 3 (tt)



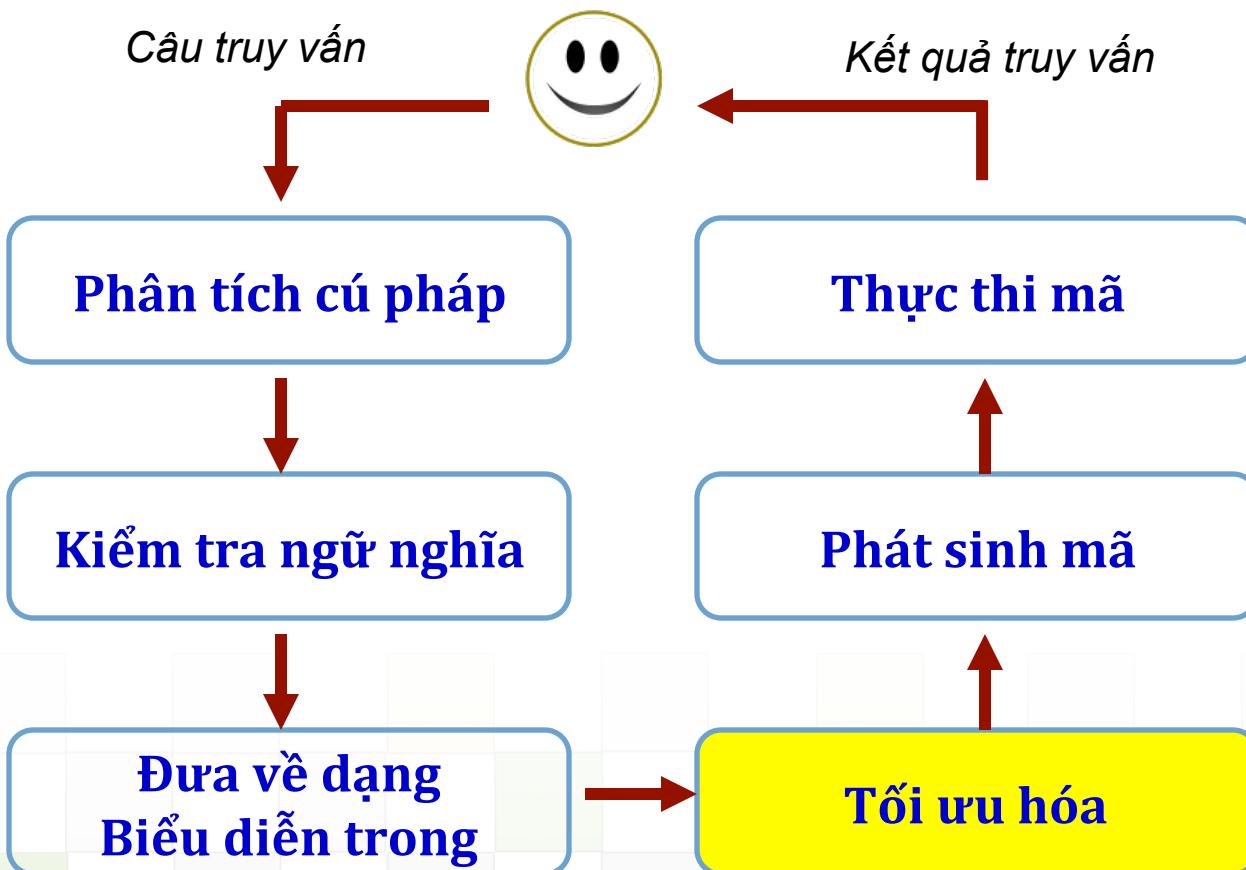


Nội dung chi tiết

- Giới thiệu
- Phân tích cú pháp - ngữ nghĩa
- Biến đổi sang Đại số Quan hệ
- **Tối ưu hóa cây truy vấn**
- Ước lượng kích thước cây truy vấn
- Phát sinh và thực thi mã lệnh



Tối ưu hóa cây truy vấn





Tối ưu hóa cây truy vấn (tt)

- Chiến lược tối ưu hóa
 - Chiến lược
 - Tốc độ thực thi câu truy vấn nhanh nhất có thể
 - Việc xử lý câu truy vấn chiếm dụng bộ nhớ ít nhất có thể
 - Nhận xét
 - Hai yêu cầu trên mâu thuẫn nhau
 - Cần phải dung hòa, thỏa hiệp
- Chiến thuật
 - Thực hiện các phép toán quan hệ 1 ngôi trước (nếu có thể)
 - Sau đó thực hiện các phép toán 2 ngôi và các phép toán 1 ngôi còn lại
 - Áp dụng các quy tắc để tối ưu



Áp dụng quy tắc

■ 1. Qui tắc giao hoán & kết hợp:

- $R \times S = S \times R$
- $(R \times S) \times T = R \times (S \times T)$
- $R \bowtie S = S \bowtie R$
- $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
- $R \cup S = S \cup R$
- $R \cup (S \cup T) = (R \cup S) \cup T$



Áp dụng quy tắc (tt)

■ 2. Qui tắc liên quan đến phép chọn σ

- Cho

- p là vị từ chỉ có các thuộc tính của R
- q là vị từ chỉ có các thuộc tính của S
- m là vị từ có các thuộc tính của R và S

- $\sigma_{p_1 \wedge p_2}(R) = \sigma_{p_1} [\sigma_{p_2}(R)]$

splitting laws

- $\sigma_{p_1 \vee p_2}(R) = [\sigma_{p_1}(R)] \cup [\sigma_{p_2}(R)]$

Quan hệ R là tập hợp
 \cup là phép hối trên tập hợp



Áp dụng quy tắc (tt)

■ 3. Qui tắc phép chọn σ, \bowtie

- $\sigma_p (R \bowtie S) = [\sigma_p(R)] \bowtie S$
- $\sigma_q (R \bowtie S) = R \bowtie [\sigma_q(S)]$
- $\sigma_{p \wedge q} (R \bowtie S) = [\sigma_p(R)] \bowtie [\sigma_q(S)]$
- $\sigma_{p \wedge q \wedge m} (R \bowtie S) = \sigma_m [\sigma_p(R) \bowtie \sigma_q(S)]$
- $\sigma_{p \vee q} (R \bowtie S) = [\sigma_p(R) \bowtie S] \cup [R \bowtie \sigma_q(S)]$

p là điều kiện chỉ liên quan thuộc tính của R và q là điều kiện chỉ liên quan thuộc tính của S

Phép chọn có tính chất quyết định trong vấn đề tối ưu hóa câu truy vấn, vì có khuynh hướng làm giảm kích thước truy vấn.

Qui tắc: đưa phép chọn xuống càng sâu trong cây biểu diễn càng tốt mà không làm thay đổi kết quả - *push selections down the tree*

m là điều kiện liên quan thuộc tính của R và S



Áp dụng quy tắc (tt)

■ 4. Qui tắc phép chọn: σ , \cup và σ , $-$

- $\Omega_c(R \cup S) = \Omega_c(R) \cup \Omega_c(S)$
- $\Omega_c(R - S) = \Omega_c(R) - S = \Omega_c(R) - \Omega_c(S)$



Áp dụng quy tắc (tt)

■ 5. Qui tắc: Phép chiếu π

- Cho

- $X = \text{tập thuộc tính con của } R$
- $Y = \text{tập thuộc tính con của } R$

- Ta có

- $XY = X \cup Y$

- Ta KHÔNG có

- $\pi_{XY}(R) = \pi_X[\pi_Y(R)]$



Áp dụng quy tắc (tt)

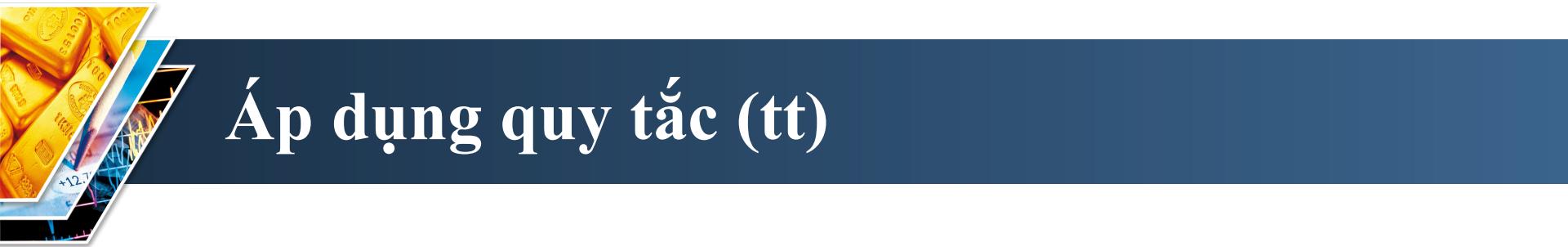
■ 6. Qui tắc: π , \bowtie

- Cho

- $X = \text{tập thuộc tính con của } R$
- $Y = \text{tập thuộc tính con của } S$
- $Z = \text{tập giao thuộc tính của } R \text{ và } S$

- Ta có

- $\pi_{XY}(R \bowtie S) = \pi_{XY} [\pi_{XZ}(R) \bowtie \pi_{YZ}(S)]$



Áp dụng quy tắc (tt)

■ 7. Qui tắc: σ , π

- Cho

- $X = \text{tập thuộc tính con của } R$
- $Z = \text{tập thuộc tính con của } R \text{ xuất hiện trong vị từ } p$

- Ta có

- $\pi_X [\sigma_p (R)] = \pi_X \{\sigma_p [\pi_{XZ} (R)]\}$



Áp dụng quy tắc (tt)

■ 8. Qui tắc: σ , π , \bowtie

- Cho

- $X = \text{tập thuộc tính con của } R$
- $Y = \text{tập thuộc tính con của } S$
- $Z = \text{tập giao thuộc tính của } R \text{ và } S$
- $Z' = Z \cup \{\text{các thuộc tính xuất hiện trong vị trí từ } p\}$

- Ta có

$$\bullet \pi_{XY} [\sigma_p (R \bowtie S)] = \pi_{XY} \{ \sigma_p [\pi_{XZ'} (R) \bowtie \pi_{YZ'} (S)] \}$$

■ 9. Qui tắc: \times, \bowtie (16.2.5 Laws About Joins and Products)

- $\sigma_C (R \bowtie S) = R \bowtie_C S$
- $R \bowtie S = \pi_L [\sigma_C (R \times S)]$



Áp dụng quy tắc (tt)

■ 10. Qui tắc δ

- $\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$
- $\delta(R \times S) = \delta(R) \times \delta(S)$
- $\delta[\sigma_C(R)] = \sigma_C[\delta(R)]$
- $\delta(R \cap_B S) = \delta(R) \cap_B S = R \cap_B \delta(S) = \delta(R) \cap_B \delta(S)$



Áp dụng quy tắc (tt)

■ 11. Qui tắc \mathfrak{I}

– Cho

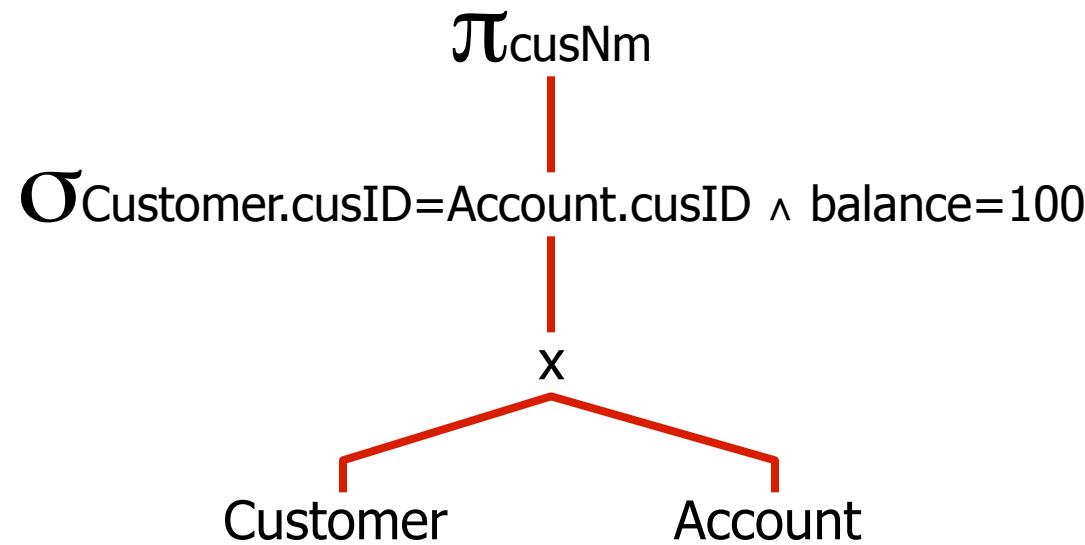
- $X = \text{tập thuộc tính trong } R \text{ được gom nhóm}$
- $Y = X \cup \{\text{một số thuộc tính khác của } R\}$

– Ta có

- $\delta[x\mathfrak{I}(R)] = {}_x\mathfrak{I}(R)$
- ${}_x\mathfrak{I}(R) = {}_x\mathfrak{I}[\pi_Y(R)]$



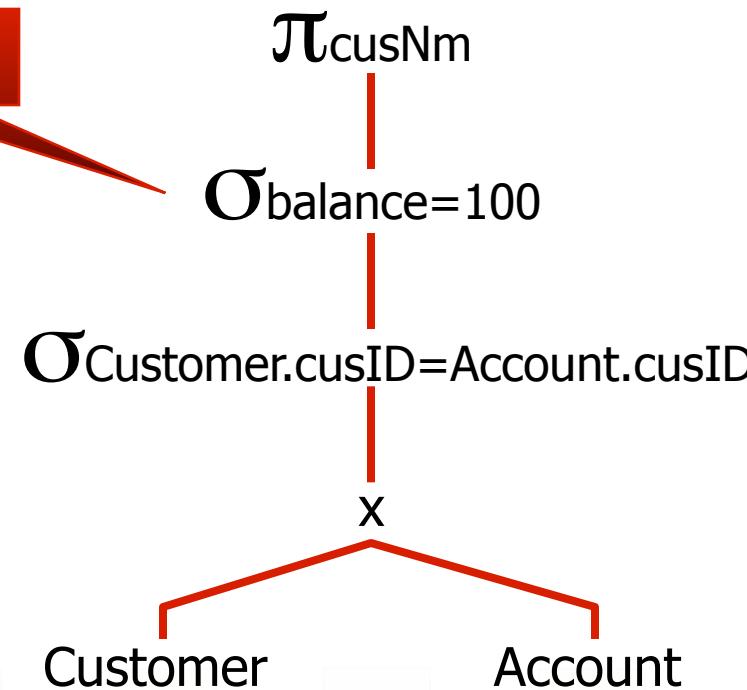
Xét ví dụ 2





Xét ví dụ 2

Qui tắc σ



Xét ví dụ 2

Qui tắc σ

π_{cusNm}

$\sigma_{\text{balance}=100}$

$\sigma_{\text{Customer.cusID}=\text{Account.cusID}}$

x

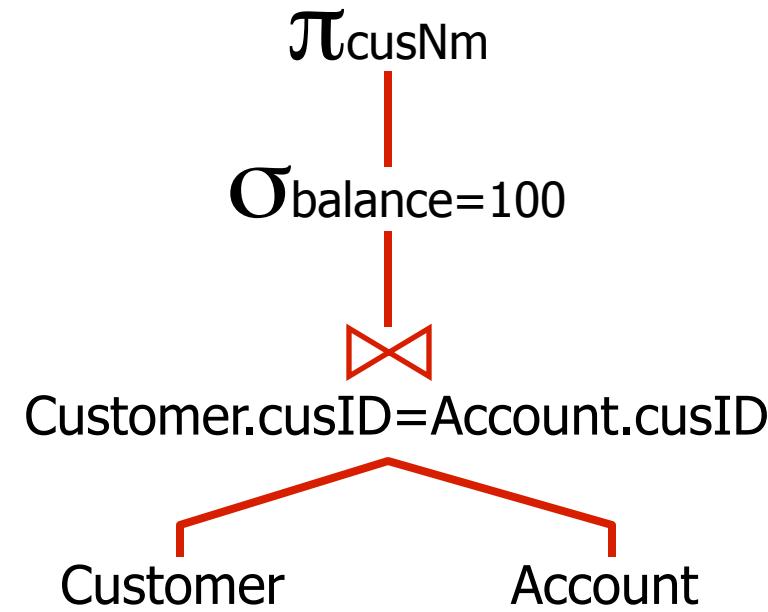
Customer

Account

Tương đương phép kết
có điều kiện

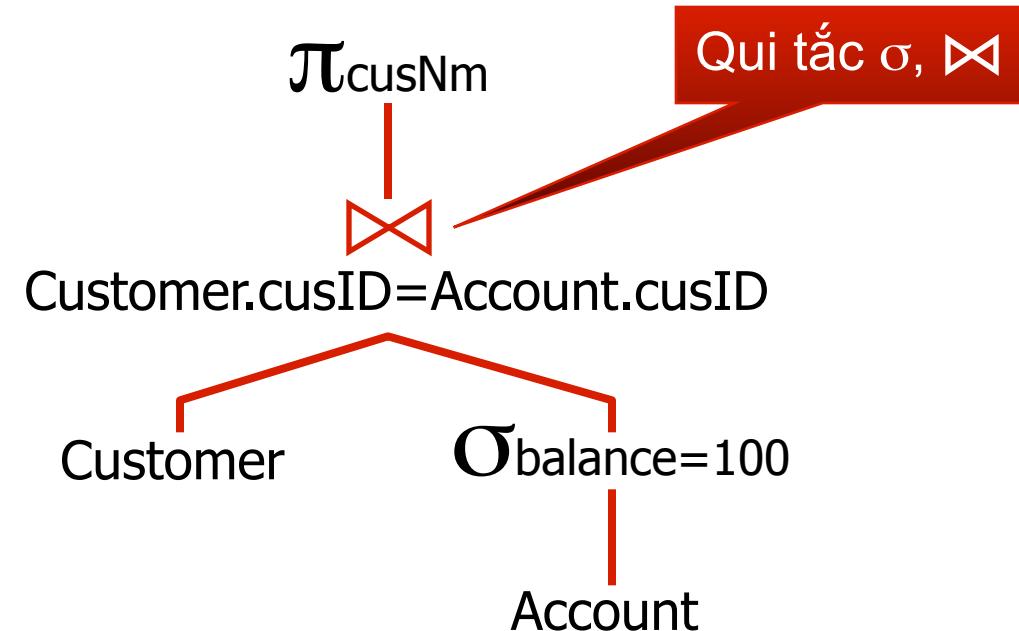


Xét ví dụ 2 (tt)



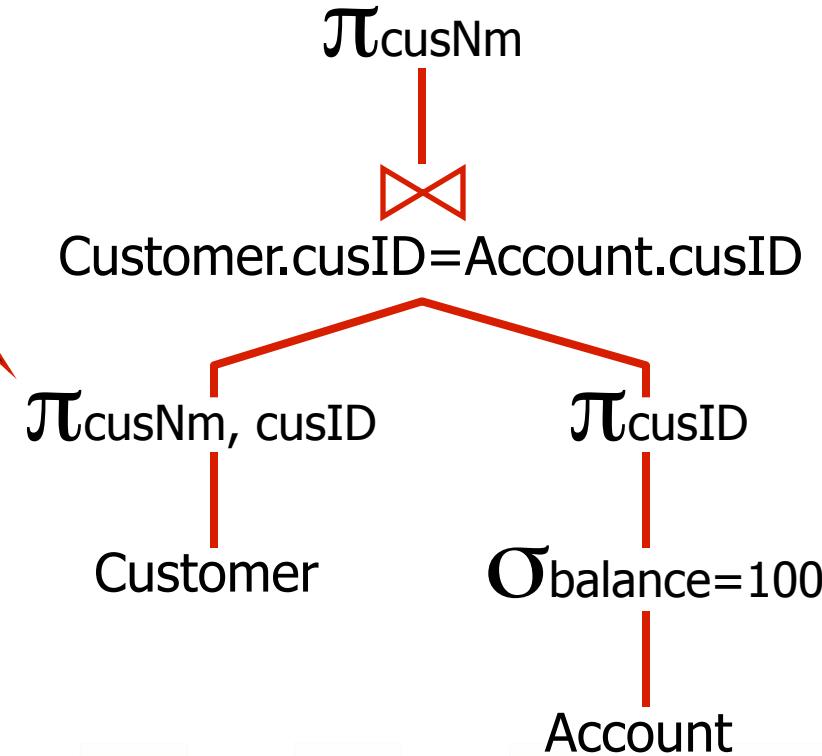


Xét ví dụ 2 (tt)



Xét ví dụ 2 (tt)

Qui tắc π , \bowtie





Nội dung chi tiết

- Giới thiệu
- Phân tích cú pháp - ngũ nghĩa
- Biến đổi sang Đại số Quan hệ
- Tối ưu hóa cây truy vấn
- **Ước lượng kích thước cây truy vấn**
- Phát sinh và thực thi mã lệnh



Ước lượng kích thước cây truy vấn

- Trong quá trình tối ưu hóa câu truy vấn, có thể có nhiều giải pháp khác nhau
 - Các giải pháp này ngang nhau về mặt chiến thuật tối ưu hóa
 - Chỉ được chọn 1 giải pháp để thực thi
 - Việc lựa chọn không được thực hiện theo cảm tính
- Do đó, cần một cách đánh giá bằng định lượng → Ước lượng kích thước cây truy vấn
 - Cây truy vấn A tốt hơn cây truy vấn B khi kích thước A nhỏ hơn kích thước B
 - Cây truy vấn được chọn để thực thi là cây truy vấn có kích thước nhỏ nhất trong các ứng viên



Ước lượng kích thước

■ Thống kê quan hệ R

- $T(R)$: số bộ trong R
- $S(R)$: tổng số byte của 1 bộ trong R
- $B(R)$: tổng số block chứa tất cả các bộ của R
- $V(R, A)$: số giá trị khác nhau mà thuộc tính A trong R có thể có



Ví dụ

- Cho quan hệ R như sau
 - A: chuỗi 20 bytes
 - B: số nguyên 4 bytes
 - C: ngày 8 bytes
 - D: chuỗi 68 bytes
 - 1 block = 1024 bytes
- Vậy
 - $T(R) = 5$
 - $S(R) = 100$
 - $B(R) = 1$
 - $V(R, A) = 3, V(R, B) = 1$
 - $V(R, C) = 5, V(R, D) = 4$

R	A	B	C	D
x	1	1	a	
x	1	2	b	
y	1	3	a	
y	1	4	c	
z	1	5	d	



Ước lượng kích thước (tt)

- Ước lượng: $W = R1 \times R2$
 - $S(W) = S(R1) + S(R2)$
 - $T(W) = T(R1) \times T(R2)$
- Ước lượng: $W = \sigma_{Z = \text{val}}(R)$
 - $S(W) = S(R)$
 - $T(W) = T(R) / V(R, Z)$
- Ước lượng: $W = \sigma_{Z \leq \text{val}}(R)$
 - $T(W) = T(R) / 2$
 - Hoặc $T(W) = T(R) / 3$

Inequality comparison



Ví dụ

- Cho
 - $R(A, B, C)$
 - $T(R) = 10000$
 - $V(R, A) = 50$
- Ước lượng kích thước biểu thức $S = \sigma_{A=10 \wedge B<20}(R)$
 - $T(S) = T(R) / [V(R, A) \times 3] = 10000 / [50 \times 3] = 67$
- Ước lượng kích thước biểu thức $S = \sigma_{A=10 \vee B<20}(R)$
 - Giả sử :
 - m_1 là số bộ thỏa $A=10$ trong R
 - m_2 là số bộ thỏa $B<20$ trong R
 - Đặt $n = T(R)$
 - $T(S) = n[1 - (1 - m_1/n)(1 - m_2/n)]$



Ước lượng kích thước (tt)

- Ước lượng: $W = R1 \bowtie R2$
- Cho
 - $X = \text{tập thuộc tính của } R1$
 - $Y = \text{tập thuộc tính của } R2$
- Xét trường hợp $X \cap Y = \emptyset$
 - Tương tự $W = R1 \times R2$
- Xét trường hợp $X \cap Y = A$
 - Nếu mọi giá trị của A trong $R1$ đều có trong $R2$
 - $T(W) = T(R1) [T(R2) / V(R2,A)]$
 - Nếu mọi giá trị của A trong $R2$ đều có trong $R1$
 - $T(W) = T(R2) [T(R1) / V(R1,A)]$
 - Tổng quát
 - $T(W) = T(R1).T(R2) / \text{Max}[V(R1,A), V(R2,A)]$



Ví dụ

Cho

- R1

- $T(R1) = 1000$
- $V(R1, A) = 50$
- $V(R1, B) = 100$

- R2

- $T(R2) = 2000$
- $V(R2, B) = 200$
- $V(R2, C) = 300$

- R3

- $T(R3) = 3000$
- $V(R3, C) = 90$
- $V(R3, D) = 500$



Ví dụ (tt)

- Hãy ước lượng $U = R1(A, B) \bowtie R2(B, C)$
 - $T(U) = (1000 \times 2000) / \text{Max}(100, 200) = 10000$
 - $V(U, A) = 50$
 - $V(U, B) = 100$
 - $V(U, C) = 300$
- Hãy ước lượng $Z = R1(A, B) \bowtie R2(B, C) \bowtie R3(C, D)$
 - Nhận xét : $Z = U(A, B, C) \bowtie R3(C, D)$
 - Vậy
 - $T(Z) = (10000 \times 3000) / \text{Max}(300, 90) = 100000$
 - $V(Z, A) = 50$
 - $V(Z, B) = 100$
 - $V(Z, C) = 90$
 - $V(Z, D) = 500$



Ước lượng kích thước (tt)

■ Ước lượng: $W = R1 \cup R2$

- Nếu $R1$ và $R2$ chấp nhận giá trị lặp
 - $T(W) = T(R1) + T(R2)$
- Nếu $R1$ và $R2$ không chấp nhận giá trị lặp
 - TH1: $R1 \cup R2$ không tạo giá trị lặp $T_1(W) = T(R1) + T(R2)$
 - TH2: $R1 \cup R2$ có tạo giá trị lặp $T_2(W) < T(R1) + T(R2)$
 - Tổng quát : $T(W) = [T_1(W) + T_2(W)] / 2$

■ Ước lượng: $W = R1 \cap R2$

- TH1 : (trường hợp nhỏ nhất) $R1 \cap R2 = \emptyset$ thì
 - $T_1(W) = 0$
- TH2 : (trường hợp lớn nhất) $R1 \cap R2 = R1$ hay $R2$ thì
 - $T_2(W) = T(R1)$ hay $T(R2)$
- Tổng quát : $T(W) = [T_1(W)+T_2(W)] / 2$



Ước lượng kích thước (tt)

- Ước lượng: $W = R_1 - R_2$
 - TH1 : (trường hợp lớn nhất) $R_1 - R_2 = R_1$ thì
 - $T_1(W) = T(R_1)$
 - TH2 : (trường hợp nhỏ nhất) $R_1 \cap R_2 = R_2$ thì
 - $T_2(W) = T(R_1) - T(R_2)$
 - Tổng quát : $T(W) = [T_1(W) + T_2(W)] / 2 = T(R_1) - T(R_2)/2$
- Ước lượng: $W = \delta(R)$
 - Giả sử $R(a_1, a_2, a_3, \dots, a_n)$
 - Vậy số bộ phân biệt tối đa là $\prod_{i \in [1, n]} V(R, a_i)$
 - Trường hợp nhỏ nhất : R rỗng $\rightarrow T(W) = 0$
 - $T(W) = \text{Min}(T(R)/2, \prod_{i \in [1, n]} V(R, a_i))$



Ước lượng kích thước (tt)

- Ước lượng: $W = \Im(R)$
 - TH1 : (trường hợp lớn nhất) số bộ phân biệt trong R cũng là số nhóm
 - $T_1(W) = T(\delta(R))$
 - TH2 : (trường hợp nhỏ nhất) R rỗng
 - $T_2(W) = 0$
 - TH3 : Toàn bộ R tạo 1 nhóm
 - $T_3(W) = 1$
 - Tổng quát : $T(W) = \text{Min}(T(R)/2, \prod_{i \in [1,n]} V(R, a_i))$



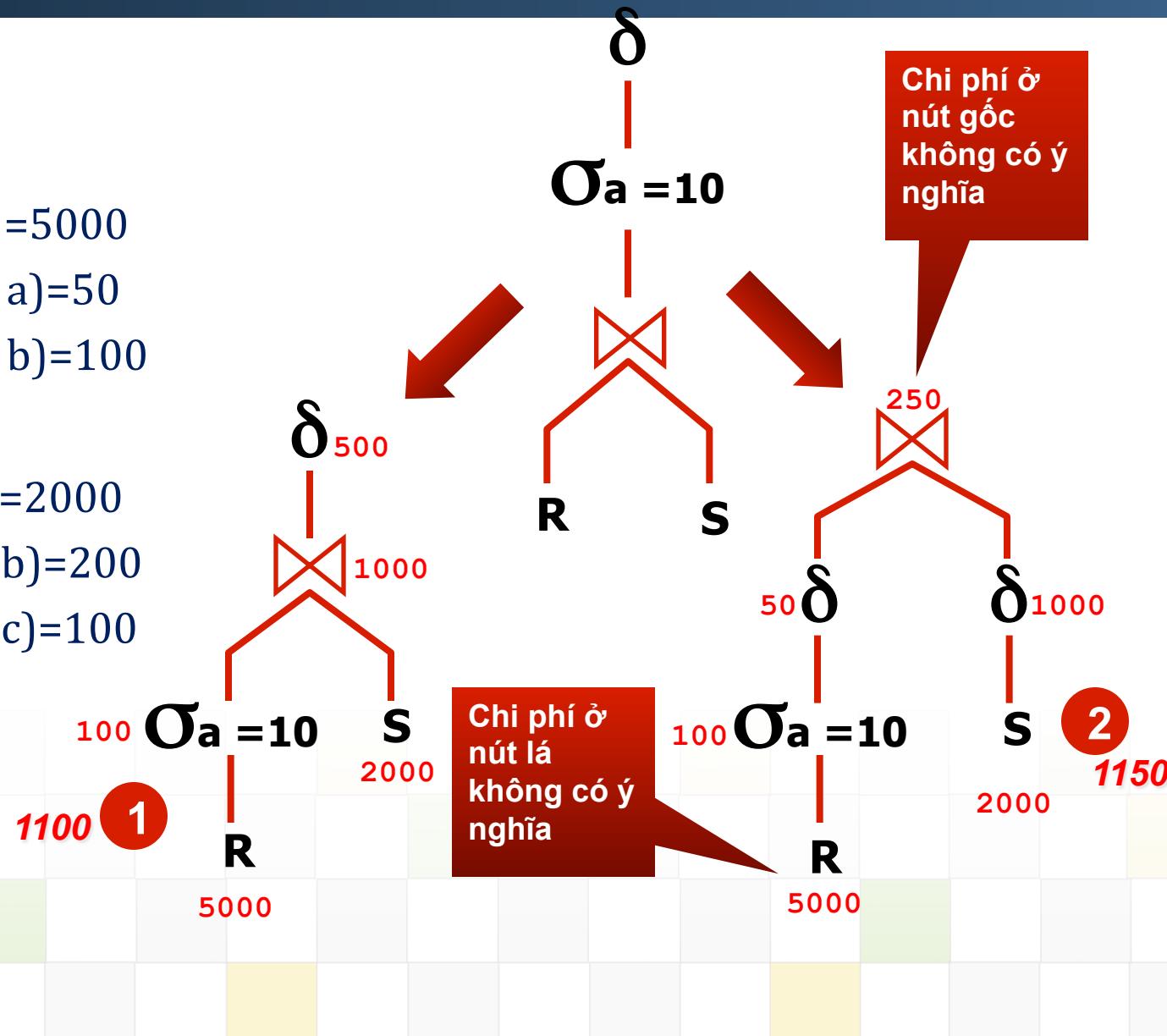
Ước lượng kích thước (tt)

- Kích thước sau cùng của cây truy vấn
 - Là tổng kích thước của phép toán ở tất cả các node, ngoại trừ node lá và node gốc.

Ví dụ

- $R(a, b)$
 - $T(R)=5000$
 - $V(R, a)=50$
 - $V(R, b)=100$

- $S(b, c)$
 - $T(S)=2000$
 - $V(S, b)=200$
 - $V(S, c)=100$



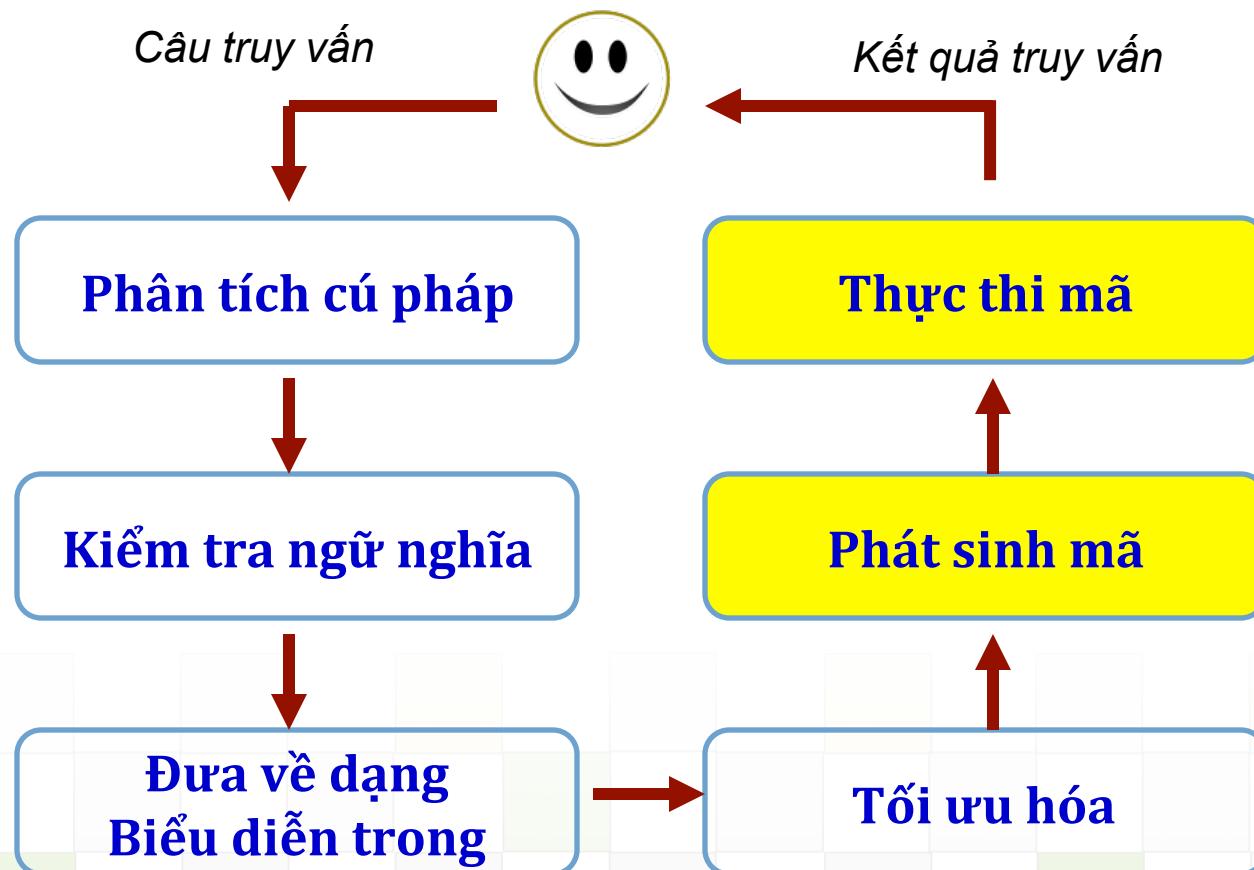


Nội dung chi tiết

- Giới thiệu
- Phân tích cú pháp - ngũ nghĩa
- Biến đổi sang Đại số Quan hệ
- Tối ưu hóa cây truy vấn
- Ước lượng kích thước cây truy vấn
- **Phát sinh và thực thi mã lệnh**



Tối ưu hóa cây truy vấn





Phát sinh mã (tt)

- Từ cây Truy vấn sau bước tối ưu hóa DBMS sẽ
 - Phát sinh mã lệnh của ngôn ngữ chủ (ngôn ngữ dùng để viết chính DBMS) để thực thi cây truy vấn ấy
 - Các phép toán của Đại số quan hệ
 - Được cài đặt trước thành một bộ các hàm (với hệ thống tham số đầy đủ).
 - Ví dụ
 - Projection (R: Relation,A: Array of Attribute) As Relation
 - Selection (R: Relation,C: Array of Condition) As Relation
 - ...
 - Việc phát sinh mã lệnh thực chất là việc phát sinh các lời gọi các hàm trên và truyền cho chúng đối số cụ thể



Phát sinh mã (tt)

■ Sắp xếp ngoài

- Việc sắp xếp là cần thiết cho thực thi truy vấn (Vd : Order by, join, union, distinct...)
- Có trường hợp yêu cầu truy vấn liên quan thuộc tính không có chỉ mục trên ấy
- Tập tin CSDL lớn → không chứa đủ trong bộ nhớ chính để sắp xếp
→ Cần phải sắp xếp ngoài (dùng file tạm trên đĩa)
- Thuật toán : merge short
 - Ban đầu sắp xếp trong các run nhỏ của tập tin chính
 - Sau đó trộn các run nhỏ và lại sắp xếp để có run lớn hơn
 - Lặp lại quá trình đến khi chỉ còn 1 run



Phát sinh mã (tt)

■ Cài đặt hàm phép chọn 1 điều kiện

- Tìm tuyến tính : Đọc từng mẫu tin và kiểm tra điều kiện chọn
- Nếu điều kiện chọn là so sánh bằng trên thuộc tính là khóa sắp xếp file → tìm nhị phân
- Nếu điều kiện chọn là so sánh bằng trên thuộc tính là khóa có primary index / hash key → dùng primary index / hash key
- Nếu điều kiện chọn là so sánh bằng trên thuộc tính không là khóa nhưng có clustering index → dùng clustering index
- Nếu điều kiện chọn không phải so sánh bằng → dùng Secondary Index
- Nếu điều kiện là so sánh \leq, \geq thì tìm cho điều kiện = trước



Phát sinh mã (tt)

- Cài đặt hàm phép chọn nhiều điều kiện (nối bởi AND)
 - Chọn 1 điều kiện để thực hiện như phép chọn đơn. Khi có kết quả, loại dần những bộ không thỏa các điều kiện còn lại
 - Thực hiện từng điều kiện như từng phép chọn đơn và giao kết quả với nhau



Phát sinh mã (tt)

■ Cài đặt hàm phép kết $R \bowtie_{R.A=S.B} S$

- Dùng 2 vòng lặp lồng nhau : Duyệt mỗi bộ r trong R, duyệt mỗi bộ s trong S và kiểm tra điều kiện $r.A = s.B$
- Nếu có chỉ mục trên B \rightarrow dùng 1 vòng lặp : Với mỗi bộ r trong R, truy cập trực tiếp (bằng chỉ mục) các bộ s trong S thỏa $s.B = r.A$
- Nếu R và S đều được sắp xếp vật lý theo A và B thì duyệt trên file tương ứng và so khớp các giá trị A và B
- Dùng hàm băm
 - Băm trên khóa A \rightarrow phân các dòng r trong R vào các lô R_i
 - Băm trên khóa B \rightarrow phân các dòng s trong S vào các lô S_i
 - Quét qua R_i và S_i và tìm các lô mà $R_i.A = S_i.B$



Thực thi mã lệnh (tt)

- Hiệu quả của việc thực thi mã lệnh đã phát sinh ở bước trước phụ thuộc vào 2 yếu tố
 - Mức độ tối ưu của cây truy vấn
 - Mức độ tối ưu của các hàm cài đặt các phép toán đại số quan hệ
- Tối ưu hóa cây truy vấn
 - Áp dụng các quy tắc (đã học trong chương này)
- Mức độ tối ưu của các hàm
 - Vận dụng các cấu trúc lưu trữ Dữ liệu (chương 5) và các thuật toán truy xuất, tìm kiếm trên các cấu trúc Dữ liệu (môn Cấu trúc Dữ liệu 1 & 2)
 - Đặc biệt quan tâm cài đặt cho phép chọn và phép kết



Tài liệu tham khảo

- [5] *Database systems: the complete book*, Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom, Pearson Prentice Hall, 2009
 - Chapter 16. Query Optimizer